# Guide to the VAXlab Laboratory Signal-Processing Routines

Order Number: AA-KP01B-TE

**August 1988**

This document describes the VAXlab laboratory signal-processing routines. It provides an overview of laboratory signal-processing concepts and presents detailed reference information about the procedures you use to perform Fourier transforms, correlation functions, filtering of data, and thermocouple conversion.

**Revision/Update Information:** This is a revised document.

**Operating System and Version:** VMS Version 5.0

**Software Version:** VAXlab Software Library Version 1.3

**digital equipment corporation**
**maynard, massachusetts**

**First Printing, December 1987**
**Revised, August 1988**



The Reader's Comments form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.



This document was prepared using VAX DOCUMENT, Version 1.0.

# Contents

PREFACE ix

CHAPTER 1 INTRODUCTION TO THE VAXlab SIGNAL-PROCESSING ROUTINES 1-1

1.1 OVERVIEW OF SIGNAL-PROCESSING ROUTINES 1-1

1.2 DATA FORMAT TRANSLATION FOR ADCs AND DACs 1-3
1.2.1 Binary Number Representation 1-3
1.2.2 Offset Binary Number Representation 1-4
1.2.3 Two's Complement Number Representation 1-4

1.3 LANGUAGES THAT DO NOT SUPPORT COMPLEX*8 DATA TYPES 1-5

CHAPTER 2 PERFORMING FOURIER TRANSFORMS AND CORRELATION FUNCTIONS 2-1

2.1 DEFINITION OF THE FOURIER TRANSFORM 2-1
2.1.1 Mathematical Definition of Continuous Fourier Transform 2-3
2.1.2 Mathematical Definition of Discrete Fourier Transform 2-3

2.2 DEFINITION OF FOURIER TRANSFORM IN TWO DIMENSIONS 2-5

2.3 DEFINITION OF THE CORRELATION FUNCTION 2-7

2.4 FOURIER TRANSFORM AND CORRELATION FUNCTION REFERENCES 2-8

CHAPTER 3 DIGITAL FILTERING 3–1

3.1 DEFINITION OF DIGITAL FILTERING 3–1
3.1.1 Polynomial Filtering 3–1
3.1.2 Nonrecursive Filtering 3–3

3.2 DIGITAL FILTERING REFERENCES 3–13

CHAPTER 4 SPECTRAL WINDOW FILTERING 4–1

4.1 OVERVIEW OF SPECTRAL WINDOW FILTERING 4–1

4.2 THE PERIODOGRAM TECHNIQUE 4–2

4.3 SPECTRAL WINDOW ROUTINES AND ALGORITHMS 4–4

4.4 SPECTRAL WINDOW FILTERING REFERENCES 4–14

CHAPTER 5 THERMOCOUPLE CONVERSION 5–1

5.1 OVERVIEW OF THERMOCOUPLE CONVERSION 5–1

CHAPTER 6 SIGNAL-PROCESSING ROUTINE CALL REFERENCE DESCRIPTIONS 6–1

6.1 OVERVIEW OF THE LABORATORY SIGNAL-PROCESSING ROUTINE FORMAT 6–1

6.2 SIGNAL-PROCESSING ROUTINE CALL SUMMARY AND DESCRIPTIONS 6–2
LSP$APPLY_WINDOW_TABLE 6–5
LSP$BUILD_WINDOW_TABLE 6–8

LSP$CORRELATION 6-11
LSP$FFT_COMPLEX 6-14
LSP$FFT_COMPLEX_2D 6-17
LSP$FFT_REAL 6-20
LSP$FILTER_NONREC 6-23
LSP$FILTER_POLY 6-26
LSP$FILTER_POLY_1ST_DERIV 6-29
LSP$FILTER_POLY_2ND_DERIV 6-32
LSP$FILTER_POLY_3RD_DERIV 6-35
LSP$FORMAT_TRANSLATE_ADC 6-38
LSP$FORMAT_TRANSLATE_DAC 6-41
LSP$HIST_F 6-45
LSP$HIST_I 6-49
LSP$PHASE_ANGLE 6-53
LSP$PHASE_ANGLE_2D 6-56
LSP$POWER_SPECTRUM 6-59
LSP$SPECTRAL_WINDOWS 6-62
LSP$THERMOCOUPLE_X 6-65

CHAPTER 7 LABORATORY SIGNAL-PROCESSING ERROR HANDLING 7-1

7.1 OVERVIEW 7-1

7.2 CHECKING ROUTINE CALL STATUS 7-2

7.3 SYMBOLIC STATUS VALUES AND DESCRIPTIONS 7-3

CHAPTER 8 OVERVIEW OF ONLINE SAMPLE PROGRAMS 8–1

APPENDIX A MATHEMATICS AND STATISTICS ROUTINES A–1

A.1 OVERVIEW OF MATHEMATICS AND STATISTICS ROUTINES A–1

A.2 MATHEMATICS ROUTINE CALL SUMMARY A–2

A.3 STATISTICS ROUTINE CALL SUMMARY A–7

APPENDIX B THE PEAK-PROCESSING ROUTINE B–1

B.1 BUILDING THE PEAK ROUTINE B–1
B.1.1 Enabling the No Filter Option B–2

B.2 OVERVIEW OF THE PEAK ROUTINE B–3

B.3 DEFINITION OF BASIC TERMS AND CONVENTIONS B–4

B.4 THE PEAK-PROCESSING ALGORITHM: PROCESSING RAW DATA B–5
B.4.1 Averaging of Input Data B–5
B.4.2 Use of the Digital Filter B–6
B.4.3 Trend Detection — Application of the Gate Factor B–7
B.4.4 Calculation of Area Under the Peak B–8
B.4.5 Algorithm Definition of the Width of a Peak B–8
B.4.6 Algorithmic Detection of the Baseline B–10
B.4.7 Flow Charts for the PEAK Routine B–11

B.5 HOW TO CALL THE PEAK-PROCESSING ROUTINE B–28

B.6 USING THE PEAK-PROCESSING ROUTINE B–33

B.7 SAMPLE PROGRAM USING THE PEAK ROUTINE B–35

## INDEX

## EXAMPLES


| | | |
|---|---|---|
| 3–1 | Using the LSP$FILTER_NONREC Routine | 3–7 |
| 4–1 | Applying a Spectral Window | 4–9 |
| B–1 | Sample Program Using the PEAK Routine | B–36 |


## FIGURES


| | | |
|---|---|---|
| 2–1 | Forward Fourier Transform | 2–2 |
| 3–1 | Digital Filter Transfer Function Forms | 3–3 |
| 3–2 | Lowpass Nonrecursive Filter for Varying nterms | 3–5 |
| 3–3 | Lowpass Nonrecursive Filter for Varying wiggles | 3–6 |
| 4–1 | The Five Spectral-Window Types | 4–6 |
| 4–2 | Raw, Unwindowed Data | 4–7 |
| 4–3 | Windowed Data | 4–8 |
| B–1 | Flow of the PEAK Routine | B–4 |
| B–2 | Calculation of True Peak Width | B–9 |
| B–3 | Calculation of Estimated Peak Width | B–10 |
| B–4 | Flow Chart for Peak Processing: Initialization, Data Averaging, and Application of Digital Filter | B–15 |
| B–5 | Flow Chart for Peak Processing: Calculation of Peak Width and Search for Baseline | B–17 |
| B–6 | Flow Chart for Peak Processing: Area Calculation | B–19 |
| B–7 | Flow Chart for Peak Processing: Determining the Baseline | B–21 |
| B–8 | NEXTPT Routine — Peak Processing | B–23 |
| B–9 | RITOUT Routine — Peak Processing | B–24 |
| B–10 | Flow Chart of Peak Events | B–25 |
| B–11 | INPTR, INLAST, and NPEAKS Point to Slots | B–34 |
| B–12 | Actual Plot of the Input Data in Example C-1 | B–38 |

## TABLES


| | | |
|---|---|---|
| 2–1 | Fourier Transform of a Real-Valued Dataset | 2–5 |
| 3–1 | Controlling Filtering Type | 3–4 |
| 4–1 | Spectral Window Function Symbolic Status Definition Files | 4–14 |
| 5–1 | Thermocouples with Conversion Routines | 5–1 |
| 5–2 | Thermocouple Temperature and Voltage Ranges | 5–2 |
| 6–1 | Signal-Processing Routine Call Summary | 6–2 |
| 7–1 | Error-Handling Symbolic Status Definition Files | 7–2 |
| 8–1 | LSP Online Sample Programs | 8–2 |
| A–1 | VAXlab Mathematics Routine Call Summary | A–2 |
| A–2 | VAXlab Statistics Routine Call Summary | A–7 |
| B–1 | Switch Settings for Significant Events in Peak Definition | B–12 |
| B–2 | Definition of Symbols | B–13 |
| B–3 | Definition of Peak Events | B–26 |

# Preface

## Intended Audience

The *Guide to the VAXlab Laboratory Signal-Processing Routines* is intended for use by scientists and engineers working in a laboratory environment. You can use this document initially as a training guide for learning the basic components of the Laboratory Signal-Processing (LSP) application software. Later, you can use it as a reference guide to look up specific information about the LSP application routines, such as how to use an optional parameter.

This guide assumes a basic understanding of computer concepts and an extensive understanding of signal-processing concepts and techniques.

## Document Structure

The *Guide to VAXlab Laboratory Signal-Processing Routines* describes how to use the signal-processing routines to perform Fourier transforms, correlation functions, filtering of data, and thermocouple conversion.

The document is divided into 8 chapters and 2 appendixes:

| Chapter Number | Contents |
|---|---|
| Chapter 1 | Presents an overview of the laboratory signal-processing routines, including information about data format translation, and binary, offset binary, and two's compliment number representation. |
| Chapter 2 | Presents definitions of Fourier transform and correlation functions, including mathematical definitions of discrete and continuous Fourier transform, and Fourier transform in two dimensions. |
| Chapter 3 | Presents digital filtering, including information on polynomial filtering, and nonrecursive filtering |
| Chapter 4 | Presents an overview of data filtering with spectral windows and the periodogram technique, including a brief description of the spectral windowing routines and the algorithms used to generate them. |
| Chapter 5 | Presents an overview of thermocouple conversion including tables showing the accuracy of conversion and thermocouple temperature and voltage ranges. |
| Chapter 6 | Provides detailed reference descriptions of the LSP routines, including routine call syntax, argument definitions, and error message condition values. |
| Chapter 7 | Explains the LSP error handling facility and provides a list of all LSP error messages and suggested recovery procedures. |
| Chapter 8 | Describes the online LSP sample programs shipped with your VAXlab system. |
| Appendix A | Describes the mathematics and statistics routines, including information about how to use the routines to perform mathematical and statistical analysis of data. |
| Appendix B | Provides a summary of the PEAK-Processing (PEAK) Routine, including information on how to create a FORTRAN program that calls the PEAK routine under the VMS operating system. |

## Associated Documents

In addition to this guide, the VAXlab documentation set includes the following Manuals:

- The *VAXlab Master Index* provides index entries from all documents in the VAXlab V1.2 documentation set.
- The *VAXlab Installation Guide* details how to install the VAXlab software.
- *Getting Started with VAXlab* is your introduction to the VAXlab system and application software. This document describes the optional hardware you can configure in a VAXlab system, the VAXlab software, and the related software you need to use with your VAXlab system, such as VAX GKS and a high-level programming language.

  This document also describes MANAGER, an interactive, menu-driven utility you can use to perform routine system management tasks. Lastly, this document presents guidelines for developing application programs with VAXlab and programming language-specific considerations, such as array dimensioning and declaring variables and data types.
- The *Guide to the VAXlab Laboratory I/O Routines* gives an overview of the LIO facility and describes how to initiate, control, process, and terminate I/O to and from VAXlab I/O devices.
- The *Guide to the VAXlab Interactive Data Acquisition Tool* describes how to communicate with VAXlab through the Interactive Data Acquisition Tool (IDAT) to establish parameters for data acquisition and to initiate, control, obtain, analyze, and plot real-time data.
- The *Guide to the VAXlab Laboratory Graphics Package* provides a comprehensive overview of the LGP facility, and explains how to specify plotting attributes and plot real-time data produced by calculations in two-dimensions, three-dimensions, and two-dimensional contours from a three-dimensional view.

The following is a list of associated software documents to reference for additional information about programming concepts and techniques not covered in this guide.

- The *Laboratory Interfacing Handbook* presents detailed descriptions of laboratory I/O concepts. If you are unfamiliar with laboratory data acquistion and control techniques, such as instruments, signals, and interfaces, or if you require additional information about computers, I/O hardware, or applications, read this handbook before you begin using the VAXlab system.
- The *VAX GKS Reference Manual, Volume I and Volume II* provide detailed information about advanced graphics programming concepts and techniques.
- The *VAX Realtime User's Guide* describes those features of VAX systems which pertain to real-time applications in scientific and industrial settings.

## Conventions

The *Guide to VAXlab Laboratory Signal-Processing Routines* uses the following documentation conventions:

| Convention | Meaning |
|---|---|
| *Italics* | Words, phrases, or characters appearing in *italics* indicates one of the following:<br>• An associated document<br>• A group of related LSP routines |
| **Bold** | A boldface word or phrase indicates one of the following:<br>• Emphasis is on an important concept or word<br>• A subroutine argument in text<br>• A subsection within a routine or parameter reference description |
| RETURN | Press the key labeled **Return** on the terminal keyboard. |
| CTRL/x | Press the key labeled **CTRL** on the terminal keyboard while simultaneously pressing the "x" key. Here, "x" is C, Y, or Z. |
| Ellipses | Vertical ellipses indicate that portions of a display or programming example were excluded for presentation purposes. |
| [Brackets] | Square brackets enclose optional parameters or arguments in routine lines. |
| UPPERCASE letters | All VAXlab routine names, VAXlab utilities (MANAGER and IDAT), and DCL commands and command strings are presented in UPPERCASE letters. |
| LSP$THERMOCOUPLE_x | Here x designates the thermocouple type, and is either B, E, J, K, R, S, or T. See Chapter 5. |

Chapter 1

# Introduction to the VAXlab Signal-Processing Routines

This chapter provides an overview of the VAXlab Laboratory Signal-Processing Routines (LSP), as well as additional information about data format translation; binary, offset binary, and two's complement number representation; and languages that do not support COMPLEX*8 data types.

## 1.1 Overview of Signal-Processing Routines

The VAXlab Laboratory Signal-Processing Routines (LSP) are a set of subroutines designed to perform a variety of standard tasks commonly encountered in the laboratory environment. Subroutines are provided to perform the following:

- Data format translation

  The LSP$FORMAT_TRANSLATE_ADC routine translates raw numbers obtained from an analog-to-digital converter into floating-point voltages. The LSP$FORMAT_TRANSLATE_DAC routine translates floating-point voltages into raw numbers appropriate for input to a digital-to-analog converter.

- Fast Fourier transformation

  The routines LSP$FFT_COMPLEX, LSP$FFT_COMPLEX_2D, and LSP$FFT_REAL calculate the discrete Fourier transform of complex-valued data in one and two dimensions and of real-valued data in one dimension. You can also perform inverse transformation using these routines.

- Power spectrum

  The LSP$POWER_SPECTRUM routine determines the power spectrum (the relationship between power and signal frequency) in a set of Fourier coefficients.

- Phase angle

  The LSP$PHASE_ANGLE and LSP$PHASE_ANGLE_2D routines convert complex numbers to phase angles and amplitudes in one and two dimensions, respectively.

- Digital filtering

  The following routines perform general-purpose nonrecursive filtering, filtering for smoothing, filtering with first, second, or third derivative output, respectively, and spectral window filtering:

  - LSP$FILTER_NONREC
  - LSP$FILTER_POLY
  - LSP$FILTER_1ST_DERIV
  - LSP$FILTER_2ND_DERIV
  - LSP$FILTER_3RD_DERIV
  - LSP$SPECTRAL_WINDOWS
  - LSP$BUILD_WINDOW_TABLE
  - LSP$APPLY_WINDOW_TABLE

- Interval histogramming

  The LSP$HIST_I and LSP$HIST_F routines count the number of elements in a data stream that fall into one or more predefined categories.

- Correlation function

  The LSP$CORRELATION routine provides a discrete method of calculating the correlation function.

- Thermocouple conversion routines

  The LSP thermocouple conversion routines provide a method for converting thermocouple voltages to temperatures.

Each routine is individually discussed and arranged alphabetically in Chapter 6, Signal-Processing Routine Call Reference Descriptions. Chapter 8, Overview of Online Sample Programs, summarizes the online sample programs that illustrate the appropriate use of each of the LSP routines.

## 1.2 Data Format Translation for ADCs and DACs

Continuous quantities, such as voltages, can be represented in digital format. However, the discrete numbering systems used by analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) can be different from the usual 16- and 32-bit integer formats. Instead, ADCs and DACs use the following types of number representation:

- Binary
- Offset binary
- Two's complement

### 1.2.1 Binary Number Representation

A common data format for ADCs and DACs is the binary format. This format uses a direct linear relationship between the ADC or the DAC and the data received from (ADC) or sent to (DAC) the device. This data format is unipolar. Zero volts is the minimum. All integers are unsigned; no bit determines whether the signal is positive or negative.

The following table gives an example of the binary number representation for both ADCs and DACs:

**12-BIT UNIPOLAR ADC AND 12-BIT UNIPOLAR DAC with BINARY CODING**

| ADC | | DAC | |
|---|---|---|---|
| Full-scale input voltage | Output Code (HEX) | Full-scale output voltage | Input Code (HEX) |
| +9.9976 | 0FFF | +9.9976 | 0FFF |
| 0.0 | 0000 | 0.0 | 0000 |

## 1.2.2 Offset Binary Number Representation

The offset binary format is similar to the binary format, except that offset binary is used with a bipolar converter. The scale is offset so that zero volts is represented by the mid point of the scale. Zero represents the lowest voltage, and all bits high represent the highest voltage possible. The highest voltage possible with binary number representation is higher than that obtainable with offset binary number representation.

The following table gives an example of the offset binary number representation for both ADCs and DACs:

**12-BIT BIPOLAR ADC AND 12-BIT BIPOLAR DAC with OFFSET BINARY CODING**

| ADC | | DAC | |
|---|---|---|---|
| Input Voltage | Output Code (HEX) | Output Voltage | Input Code (HEX) |
| +9.9951 | OFFF | +9.9951 | OFFF |
| 0.0 | 0800 | 0.0 | 0800 |
| -10.0000 | 0000 | -10.0000 | 0000 |

## 1.2.3 Two's Complement Number Representation

The two's complement format uses the most significant bit (MSB) with respect to the number of bits of resolution of the device to denote the sign (+ or -) of the converted voltage. If the MSB is high, then the voltage is negative. If the MSB is low, then the voltage is positive.

Integer numbers are most commonly represented in two's complement format. The MSB is the sign bit.

The following table shows the two's complement number representation for both ADCs and DACs:

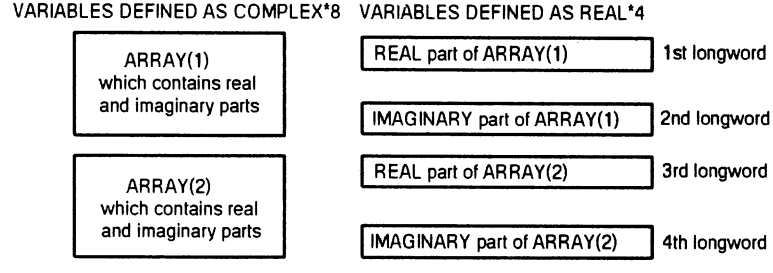
**12-BIT BIPOLAR ADC AND 12-BIT BIPOLAR DAC with TWO'S COMPLEMENT CODING**

| ADC | | DAC | |
|---|---|---|---|
| Input Voltage | Output Code (HEX) | Output Voltage | Input Code (HEX) |
| +9.9951 | 07FF | +9.9951 | 07FF |
| 0.0 | 0000 | 0.0 | 0000 |
| -10.0000 | F800 | -10.0000 | F800 |

## 1.3 Languages That Do Not Support COMPLEX*8 Data Types

Some high-level languages, such as Pascal, do not support the COMPLEX*8 data type used throughout the VAXlab signal-processing routines. If you are programming in a language that does not support the COMPLEX*8 data type, you can declare a REAL*4 array or variable as twice the length of a COMPLEX*8 array or variable. You need to distinguish between the real and imaginary components of the complex number.

The following figure shows the storage format of COMPLEX*8 variables, and of REAL*4 variables with separation of the real and imaginary components.



Alternatively, you can set up the complex array as a two-dimensional (REAL*4) array so that you can access either the real or the imaginary components with the same index. The ordering of the array is dependent upon the language in which you are programming. See the individual language manual for language-specific information about the ordering of two-dimensional arrays.

Chapter 2

# Performing Fourier Transforms and Correlation Functions

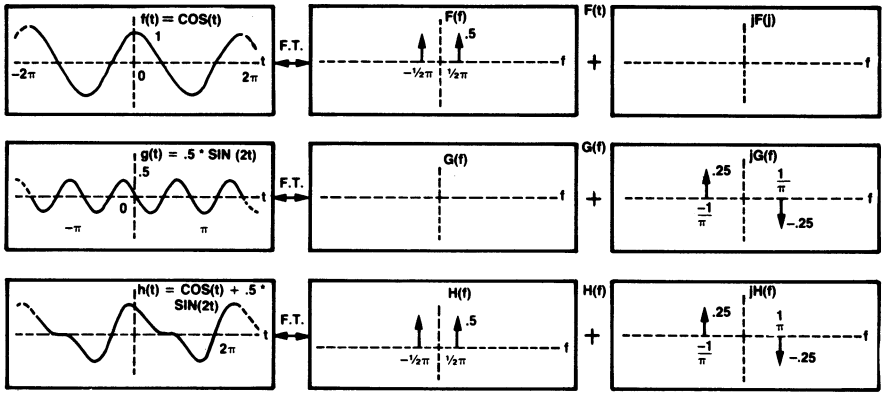
This chapter provides an introduction to Fourier transforms and correlation functions, including mathematical definitions of discrete and continuous Fourier transforms and the Fourier transform in two dimensions. A reference section is also included.

## 2.1 Definition of the Fourier Transform

Fourier transformation decomposes a signal into component sine and cosine representation. LSP provides three routines that perform the fast Fourier transform (FFT). The FFT routines provide you with an efficient means of numerically approximating the analytical or continuous Fourier transform.

The forward Fourier transform is a mathematical operation that converts numbers in the time domain to numbers in the frequency domain. Figure 2-1 shows the effects of the forward Fourier transform.

**Figure 2–1: Forward Fourier Transform**




where

f(t), g(t), and h(t) are functions of time.

F(f), G(f), and H(f) are the real components of the forward Fourier transforms of f(t), g(t), and h(t), respectively.

jF(f), jG(f), and jH(f) are the imaginary components of the forward Fourier transforms of f(t), g(t), and h(t), respectively.

You can also perform the inverse Fourier transform. The inverse Fourier transform converts a function in the frequency domain to an expression in the time domain. Both forward and inverse transform operations are provided in the FFT routines LSP$FFT_COMPLEX, LSP$FFT_COMPLEX_2D, and LSP$FFT_REAL. See Chapter 6, Signal-Processing Routine Call Reference Descriptions, for information about using these routines.

### 2.1.1 Mathematical Definition of Continuous Fourier Transform

The analytical expression for the forward Fourier transform for continuous functions is given as:

$$H(f) = \int_{-\infty}^{\infty} h(t)e^{(-j2\pi ft)}dt$$

where

H(f) is a function in the frequency (f) domain.
h(t) is a function in the time (t) domain.
j is the square root of -1.

The inverse operation is given as:

$$h(t) = \int_{-\infty}^{\infty} H(f)e^{(j2\pi tf)}df$$

Variations on the above definitions do exist. See the list of references in Section 2.4, Fourier Transform and Correlation Function References, for information about where to find the various definitions of the forward and inverse Fourier transforms.

### 2.1.2 Mathematical Definition of Discrete Fourier Transform

A digital computer cannot perform the integration indicated by the mathematical expressions for the continuous Fourier transform. A digital computer can only deal with discrete data points. Thus, the FFT routines must use a method known as the discrete Fourier transform to approximate the continuous Fourier transform at discrete frequencies.

The discrete Fourier transform does not process a continuous function. Instead, it processes discrete points that give only an approximation of the continuous function. The mathematical expression of the discrete Fourier transform is:

$$H(k) = \sum_{n=0}^{N-1} h(n)e^{(-j2\pi kn)/N}$$

The inverse operation is given as:

$$h(n) = \frac{1}{N} \sum_{k=0}^{N-1} H(k) e^{(j2\pi nk)/N}$$

Note that the n and the k used in these two equations are indices, and that h(n) and H(k) represent discrete functions of equispaced data in the time and frequency domains, respectively, with length N.

Although the FFT routines use the discrete Fourier transform algorithm as a model, they also take advantage of certain computational shortcuts to reduce the time required to evaluate the resulting data. Because of this computational time reduction, the shortcut method used by the routines is known as fast Fourier transform.

Using these discrete formulas does not present an efficiency problem when transforming small numbers of data points. However, when N is a large number, analysis shows that the number of computational steps required when using the discrete Fourier transform algorithm is proportional to $N^2$. The fast techniques enable the calculation to be proportional to N ln (N) computational steps.

The LSP$FFT_COMPLEX and LSP$FFT_REAL routines transform N number of data points. A restriction inherent in these subroutines is that N be a power of 2; that is, $N = 2^M$

where

M is between 1 and 15, inclusive.

When this constraint is impractical, techniques such as zero-filling can be used.

Because of the symmetry properties of a Fourier transform of real-valued data, only half of the output data needs to be stored. When the Fourier transform of a real-valued data sequence is transformed, the following identity results from the symmetry equation:

$$H(N - k) = H^*(k)$$

where

$h^*(k)$ is the complex conjugate of h(k).
k = 0, 1, 2, ... N/2.

This means that the resulting output array is symmetric around the (N/2)+1 transformed data point. There is no need to store data past the (N/2)+1 point since the complex conjugate is easily computed in one simple loop. This method is referred to as reduced-symmetric storage.

Consider the Fourier transform of a real-valued dataset of length 8. The resulting array of length 5 consists of complex numbers denoted as $A_r$ through $E_r$ (real) and $A_i$ through $E_i$ (imaginary). The $A_i$ imaginary term and the $E_i$ imaginary term are always equal to 0. Performing a Fourier transform produces array values 1-5, and the equation, $H(N-k) = H^*(k)$, produces array values 6-8. Note that $A_r$ is the term which is independent of frequency. Table 2–1 uses these array values.

**Table 2–1: Fourier Transform of a Real-Valued Dataset**

| Array Location | Term | Term |
|---|---|---|
| 1 | $A_r$ | $A_i$ |
| 2 | $B_r$ | $B_i$ |
| 3 | $C_r$ | $C_i$ |
| 4 | $D_r$ | $D_i$ |
| 5 | $E_r$ | $E_i$ |
| 6 | $D_r$ | $-D_i$ |
| 7 | $C_r$ | $-C_i$ |
| 8 | $B_r$ | $-B_i$ |

See Section 2.4, Fourier Transform and Correlation Function References, for further information.

## 2.2 Definition of Fourier Transform in Two Dimensions

The one-dimensional discrete Fourier transform (DFT) results from interpreting a finite-duration sequence as one period of a periodic sequence and applying the discrete Fourier series. In a similar manner, you can apply the two-dimensional Fourier series to represent a two-dimensional sequence that is nonzero for only a finite area in the x, y plane. Such a sequence is referred to as a finite-area sequence, and is the two-dimensional counterpart to a finite-duration sequence. The resulting Fourier representation is referred to as the two-dimensional discrete Fourier transform.

Thus, with $H(k_1,k_2)$ denoting the discrete Fourier transform of $h(n_1,n_2)$, the mathematical expressions for the discrete Fourier transform pair are:

$$H(k_1,k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} h(n_1,n_2) e^{(-j2\pi k_1 n_1)/N_1} e^{(-j2\pi k_2 n_2)/N_2}$$

$$h(n_1,n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} H(k_1,k_2) e^{(j2\pi k_1 n_1)/N_1} e^{(j2\pi k_2 n_2)/N_2}$$

The two-dimensional discrete Fourier transform can be rewritten as follows:

$$H(k_1,k_2) = \sum_{n_1=0}^{N_1-1} [ \sum_{n_2=0}^{N_2-1} h(n_1,n_2) e^{(-j2\pi k_2 n_2)/N_2} ] e^{(-j2\pi k_1 n_1)/N_1}$$

The quantity in brackets, $G(n_1,k_2)$, is a two-dimensional sequence which allows $H(k_1,k_2)$ to be rewritten as follows:

$$G(n_1,k_2) = \sum_{n_2=0}^{N_2-1} h(n_1,n_2) e^{(-j2\pi k_2 n_2)/N_2}$$

$$H(k_1,k_2) = \sum_{n_1=0}^{N_1-1} G(n_1,k_2) e^{(-j2\pi n_1 k_2)/N_1}$$

Each column of G is the one-dimensional discrete Fourier transform of the corresponding column of x. Each row of H is the one-dimensional discrete Fourier transform of the corresponding row of G. You can compute a two-dimensional Fourier transform by first performing a one-dimensional tranform on the columns of $h(n_1,n_2)$, then on the row of the resultant $G(n_1,k_2)$. You can also apply a similar method to the inverse discrete Fourier transform.

## 2.3 Definition of the Correlation Function

You use the correlation function to produce an estimate of the degree of similarity between two functions when one of the functions is shifted either in time or by some other independent variable. You can also use the correlation function on one function; this is known as autocorrelation.

Mathematically, the correlation function is:

$$R_{xy}(t) = \int_{-\infty}^{\infty} x(B) * y(B+t)dB$$

where

$R_{xy}$ (t) is the correlation function of the two functions x and y, and t is the time shift.

B is a dummy variable of integration.

Just as the continuous Fourier transform has a discrete analog, the correlation function also has a discrete analog, which is:

$$R_{xy}(t) = \frac{1}{N} \sum_{k=0}^{N-1-t} x(k)y(k+t)$$

where

k is an index.

N is the total number of data points.

The discrete autocorrelation equation is:

$$R_{xx}(t) = \frac{1}{N} \sum_{k=0}^{N-1-t} x(k)x(k+t)$$

For large records of data, it is impractical to calculate correlation or autocorrelation using the discrete analog equations above. Since the correlation calculation is closely related to the Fourier transform, computation time can be substantially reduced by using the fast Fourier transform methods.

See Section 2.4, Fourier Transform and Correlation Function References, for the further information about the correlation function. See the routine call reference description for information about the LSP$CORRELATION routine.

## 2.4 Fourier Transform and Correlation Function References

Further information about the fast Fourier transform operation and the correlation and autocorrelation functions can be obtained in the following references:

Blackman, R.B., and J.W. Tukey. *The Measurement of Power Spectra.* New York: Dover Publications, 1958.

Bracewell, R.N. *The Fourier Transform and Its Application.* New York: McGraw-Hill Book Company, 1978.

Brigham, E.O. *The Fast Fourier Transform.* Englewood: No Such Press, 1980.

Burrus, C.S., and T.W. Parks. *DFT/FFT and Convolution Algorithms.* New York: Wiley-Interscience, 1985.

Elliot, D.F., and K.R. Rao. *Fast Transforms: Algorithms, Analyses, Applications.* Orlando: Academic Press, 1982.

Chapter 3

# Digital Filtering

This chapter provides an overview of digital filtering, including information on polynomial filtering and nonrecursive filtering.

## 3.1 Definition of Digital Filtering

You use the technique of digital filtering to eliminate certain frequency components from a signal that is corrupted by noise. The VAXlab signal-processing routines provide two types of digital filters:

- Filters which are based on simple interpolating polynomials and which act as lowpass filters capable of producing derivative information.
- A nonrecursive (finite impulse response) filter which can be used as either a lowpass, highpass, bandpass, or bandstop (notch) filter.

### 3.1.1 Polynomial Filtering

You can achieve simple but effective filtering by the method of least-squares. Using this method, a group of equispaced data is presented and the best (in a least-squares sense) polynomial is fitted at neighboring data values. This method weights the individual datum by its neighboring data to substantially reduce noise.

Assume you have a group of equispaced data of length N. Pick a window size k that is substantially less than N. The method of polynomial least-squares (in principle) is used to calculate the polynomial coefficients $a_0$ through $a_j$, where j denotes the order of the interpolating basis function for a polynomial of form:

$$y(x) = a_0 + a_1 x + a_2 x^2 + a_j x^j$$

All k data points x = 1, 2, 3, ... k are used for this calculation. Note that k is an odd number. The y(x) is calculated for x = (k+1)/2 (the midpoint), and this ideal data point is used as the output value. The next k values used are x = 2, 3, 4, ... k+1. The least-squares function (above) is repeated, and the ideal midpoint is used for the next output value. This procedure is repeated until all of the data is input.

The derivative filters operate in the same manner, except that the derivative of the above equation is calculated by rearranging the equation to produce derivative output. The polynomial filters use the curve-fitting approach implemented in convolution weights, so they filter quickly. These filters are lowpass filters with good roll-off characteristics. However, you can obtain faster roll-off by using the nonrecursive filter (LSP$FILTER_NONREC). See Section 3.2, Digital Filtering References, for references containing further information about digital filtering.

In all of the filters (both polynomial and nonrecursive), certain data points at the beginning and end of the dataset are not filtered. For the polynomial filters (LSP$FILTER_POLY), the first (k-1)/2 and the last (k-1)/2 points are not filtered.

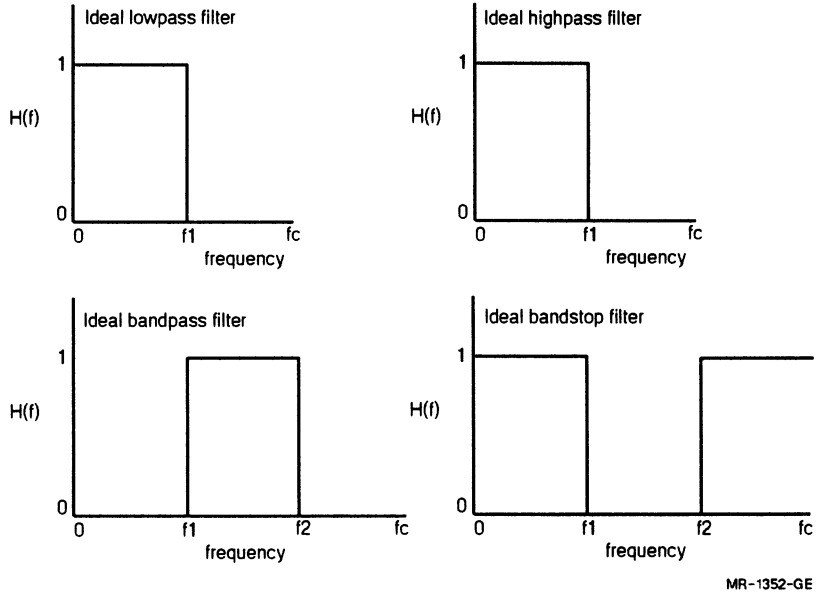
For the derivative filters, the first (k-1)/2 and the last (k-1)/2 data points are included as zeros in the output array. See Chapter 6, Signal-Processing Routine Call Reference Descriptions, for further information about filtering windows.

### 3.1.2 Nonrecursive Filtering

Use the LSP$FILTER_NONREC routine to perform nonrecursive filtering in either lowpass, highpass, bandpass, or bandstop (notch) mode.

The transfer function of this digital filter is denoted as H(f). Putting a sinusoidal function of frequency f into the filter results in the output being the same as the sinusoid, except that its amplitude is multiplied by H(f). The transfer function H(f) can take on any of the forms shown in Figure 3–1.

**Figure 3–1: Digital Filter Transfer Function Forms**




where

$fc$ refers to the Nyquist frequency: $fc = 1/(2\Delta t)$.
$\Delta t$ is the time between data samples.
$f_1$, $f_2$ refers to the frequency values where you apply filtering.

These ideal filters represent an infinitely sharp band; in practice, as is shown below, the vertical lines are skewed. Table 3–1 shows the **flow** and **fhigh** argument values you use in the LSP$FILTER_NONREC routine to control the type of filtering.

**Table 3–1: Controlling Filtering Type**

| For a filter type of: | Set FLOW to: | Set FHIGH to: |
|---|---|---|
| No filtering | 0 | 1 |
| Lowpass filter | 0 | 0 < FHIGH < 1 |
| Highpass filter | 0 < FLOW < 1 | 1 |
| Bandpass filter | 0 < FLOW < FHIGH | FLOW < FHIGH < 1 |
| Bandstop filter | FHIGH < FLOW < 1 | 0 < FHIGH < FLOW |

The filtering discussed here pertains only to nonrecursive filters of the form:

$$y_n = A_0 x_n + \sum_{k=1}^{nterms} A_k (x_{(n+k)} + x_{(n-k)})$$

where

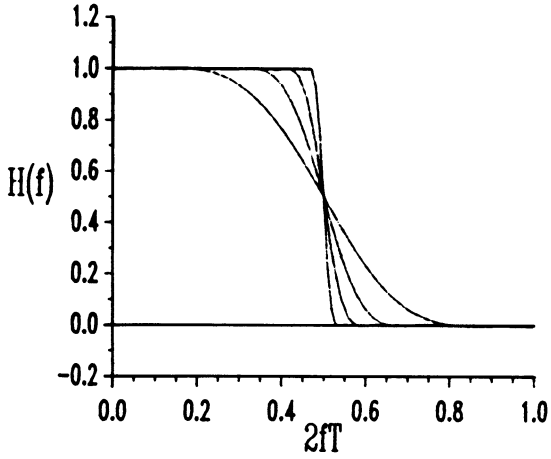
$y_n$ is the dependent variable synthesized by the use of previous dependent values (x).

$A_k$ are the filter-dependent coefficients.

**nterms** is the number of filter coefficients ( $A_0$ is not included).

Figure 3–2 shows the transfer function of a lowpass nonrecursive filter where **wiggles** = 50.0, **flow** = 0.0, and **fhigh** = 0.5 for **nterms** = 5, 10, 20, and 50.

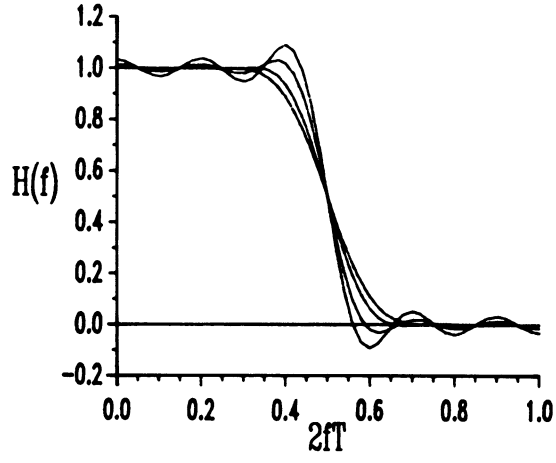
**Figure 3–2: Lowpass Nonrecursive Filter for Varying nterms**




The **nterms** argument determines the sharpness of the filter. When **nterms** is increased, the filter cutoff is sharper. Though it seems that using the largest possible value for **nterms** results in a sharper filter, **nterms***2 number of data points from the original set are not filtered. If the data set is large, the loss of data caused by the filtering process is inconsequential. However, the loss of data can be detrimental to smaller data sets. In addition, the computational time increases proportionally to **nterms**. Try to make the value of **nterms** as large as possible without losing too many end points or making the computational time too long.

Figure 3–3 shows the transfer function of a lowpass nonrecursive filter where **flow** = 0.0, **fhigh** = 0.5, and **nterms** = 10 for **wiggles** = 0, 30, 50, and 70.

**Figure 3–3: Lowpass Nonrecursive Filter for Varying wiggles**




The **wiggles** argument controls the smoothness of the filter. The wiggles, which are related to the size of Gibbs Phenomenon oscillations, are most prominent when the value of the **wiggles** argument is 0.0. As the value of **wiggles** is increased, the oscillations become less noticeable; however, the sharpness of the filter decreases. A good compromise is to set **wiggles** = 50.0.

The size of the oscillations (in -dB units) is related to the value of the **wiggles** argument:

$$|magnitude\ of\ oscillations| = 10^{(-wiggles/20.0)}$$

This nonrecursive filter is an adaptation of the $I_0 - sinh$ filter originally proposed by Kaiser. See *Digital Filters* by R.W. Hamming for a complete mathematical description of this filter.

Example 3–1 illustrates the use of the LSP$FILTER_NONREC routine in lowpass mode and in highpass mode.

## Example 3–1: Using the LSP$FILTER_NONREC Routine

```
C LSP_FILT_NONREC.FOR
C
C This sample program demonstrates use of the LSP$FILTER_NONREC routine in
C lowpass mode and in highpass mode.
C
        INCLUDE 'SYS$LIBRARY:LSPDEF.FOR'
C
C Define Variables:
C
        REAL*4 PI,WAVE(2000),LOWOUT(2000),HIGHOUT(2000),LOWAVE(2000)
        REAL*4 HIWAVE(2000),BIGAMP,SMALAMP,CONO,XO(2000)
        REAL*4 FLOW,FHIGH
        REAL*4 WIGGLES

        INTEGER*4 I,N,NTERMS,ISTAT,STATUS
C
C Create a sine wave of period 2 pi and superimpose on this waveform
C a smaller amplitude sine wave which oscillates 40 times faster.
C
C Filter in lowpass mode to get the primary sine wave and put
C this output in the array LOWOUT; filter in highpass mode to
C put the higher frequency signal into the array HIGHOUT.
C
        N=2000                      ! Number of points in array

        PI=3.141592654              ! The constant pi

        BIGAMP=20.0                 ! Peak-to-peak amplitude of the
                                    ! low frequency sine wave

        SMALAMP=2.0                 ! Peak-to-peak amplitude of the
                                    ! high frequency sine wave

        CONO=2.0*PI/(FLOAT(N-1))    ! A constant for the primary
                                    ! sine wave period

        DO 10 I=1,N                 ! Generate N values for WAVE

        XO(I)=(FLOAT(I-1))*CONO     ! Argument for low frequencies

        LOWAVE(I)=BIGAMP*SIN(XO(I))  ! Store low frequency component

        HIWAVE(I)=SMALAMP*SIN(XO(I)*40.0) ! Store high frequency component

        WAVE(I)=LOWAVE(I)+HIWAVE(I)       ! Generate the wave as a sum of
                                          ! high and low frequency terms
10      CONTINUE
```

Example 3–1 Cont'd. on next page

## Example 3–1 (Cont.): Using the LSP$FILTER_NONREC Routine

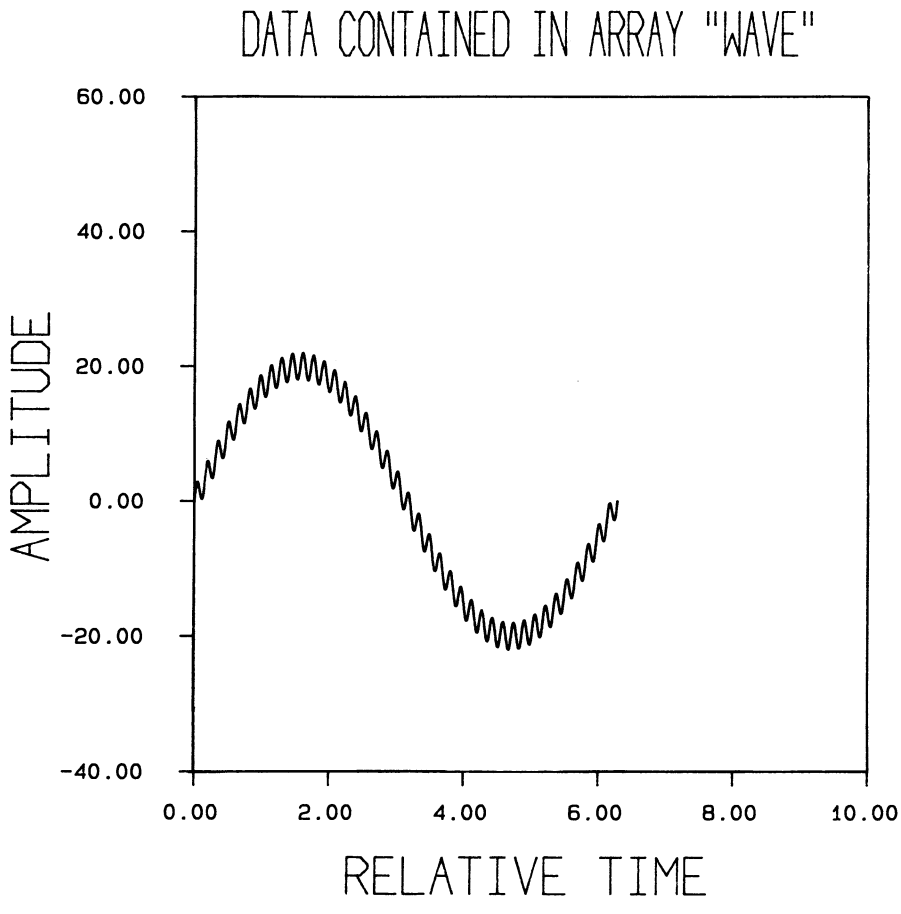
```
C
C Now lowpass filter the array WAVE:
C
        FLOW=0.0                ! Set for lowpass filter
        FHIGH=0.006             ! with roll-off at .006 f sub c
        WIGGLES=100.0           ! oscillations set
        NTERMS=60               ! with 60 terms in the filter coefficients

        ISTAT=LSP$FILTER_NONREC(WAVE,LOWOUT,N,FLOW,FHIGH,WIGGLES,NTERMS,STATUS)
        IF (.NOT. ISTAT) CALL LIB$SIGNAL(%VAL(ISTAT))
C
C Now highpass filter the array WAVE:
C
        FLOW=0.03               ! Set for highpass filter
        FHIGH=1.0               ! start roll-off at .02 f sub c
        WIGGLES=100.0           ! oscillations set
        NTERMS=100              ! with 100 terms in the filter coefficients

        ISTAT=LSP$FILTER_NONREC(WAVE,HIGHOUT,N,FLOW,FHIGH,WIGGLES,NTERMS,STATUS)
        IF (.NOT. ISTAT) CALL LIB$SIGNAL(%VAL(ISTAT))
C
C Plot the original, lowpass, and highpass filtered waveforms:

        XCONTROL(1) = 5.0
        XCONTROL(2) = 0.0
        XCONTROL(3) = 10.0
        XCONTROL(4) = 2.0

        YCONTROL(1) = 5.0
        YCONTROL(2) = -40.0
        YCONTROL(3) = 60.0
        YCONTROL(4) = 20.0
```

**Example 3–1 Cont'd. on next page**

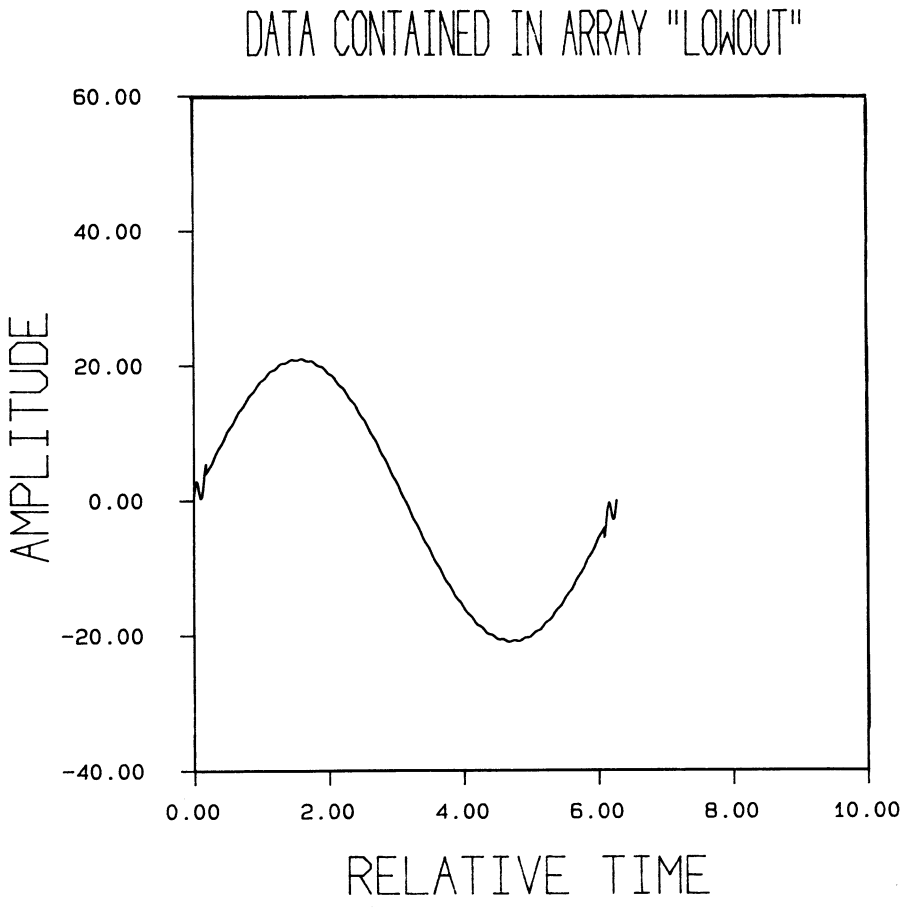
## Example 3–1 (Cont.): Using the LSP$FILTER_NONREC Routine

```
      CALL LGP$PLOT(1,'EXSY',XO,WAVE,N,'RELATIVE TIME','AMPLITUDE',,,,
     1         XCONTROL,YCONTROL,,'DATA CONTAINED IN ARRAY "WAVE"')
      CALL LIB$WAIT (10.0)
      CALL LGP$TERMINATE_PLOT (1,1)

      CALL LGP$PLOT(1,'EXSY',XO,LOWOUT,N,'RELATIVE TIME','AMPLITUDE',,,,
     1         XCONTROL,YCONTROL,,'DATA CONTAINED IN ARRAY "LOWOUT"')
      CALL LIB$WAIT (10.0)
      CALL LGP$TERMINATE_PLOT (1,1)

      CALL LGP$PLOT(1,'EXSY',XO,HIGHOUT,N,'RELATIVE TIME','AMPLITUDE',,,,
     1         XCONTROL,YCONTROL,,'DATA CONTAINED IN ARRAY "HIGHOUT"')
      CALL LIB$WAIT (10.0)
      CALL LGP$TERMINATE_PLOT (1,1)
      STOP
      END
```

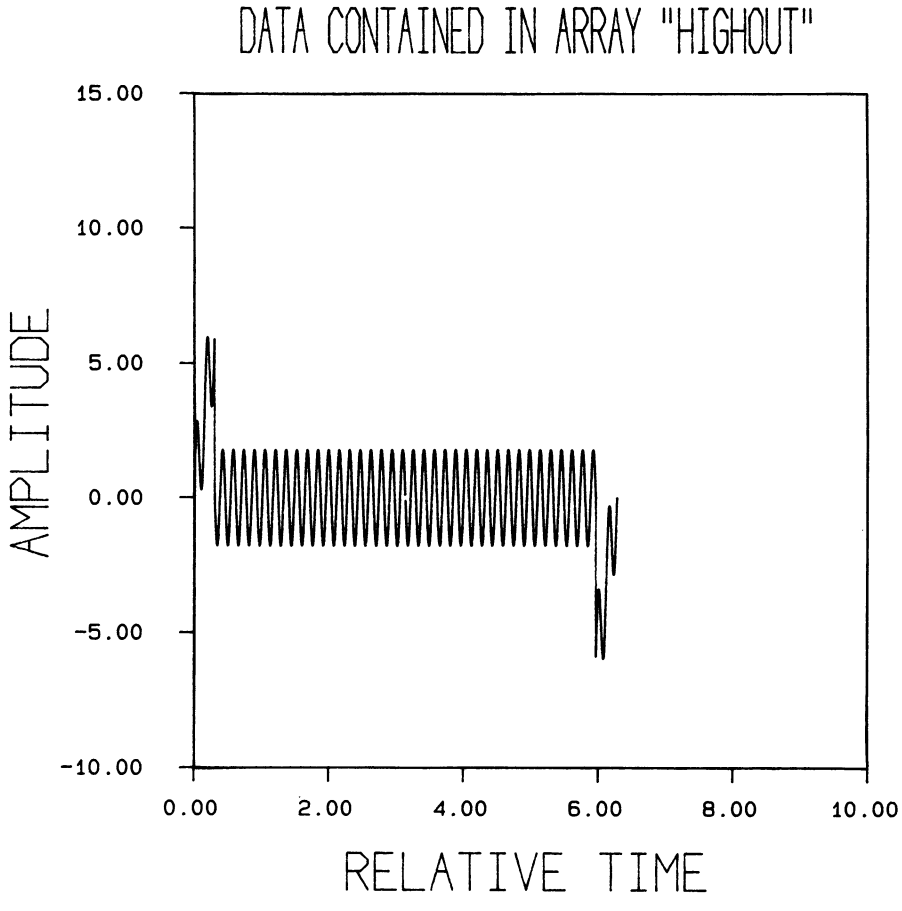
Example 3-1 produces the following graphical representation of the data contained in array WAVE:



Example 3–1 produces the following graphical representation of the data contained in array LOWOUT:



Example 3–1 produces the following graphical representation of the data contained in array HIGHOUT:



## 3.2 Digital Filtering References

You can obtain further information about digital filtering in the following references:

Antoniou, A. *Digital Filters: Analysis and Design*. New York:McGraw-Hill, 1979.

Hamming, R.W. *Digital Filters*. Prentice Hall, 1977.

Horlick, G. and K.R. Betty. *Analytical Chemistry*, 351, vol. 49, 1977.

Madden, Hannibal H. *Analytical Chemistry*, 1383, vol. 50, 1978.

Oppenheim, A.V. and R.W. Shafer. *Digital Signal Processing*. Prentice Hall, 1975.

Savitsky A. and M.J.E. Golay. *Analytical Chemistry*, 1627, vol. 36, 1964.

Steinier J., Y. Termonia, and J. Deltour. *Analytical Chemistry*, 1906, vol. 44, 1972.

Chapter 4

# Spectral Window Filtering

This chapter presents concepts and illustrations of spectral window filtering and the periodogram technique.

## 4.1 Overview of Spectral Window Filtering

The fast Fourier transform (FFT) and power spectrum analysis of a discretely sampled data sequence results in spectrum spreading and leakage. Applying a spectral window filter to the data increases accuracy in estimating the power spectrum and lessens leakage. The periodogram technique further improves the data analysis process by calculating the power spectrum more efficiently. The time savings is particularly evident when the data sequence is long.

If results of an FFT and power spectrum analysis of a pure, single-frequency signal display side lobes and spreading in addition to the main frequency component, spectrum spreading or leakage is present. Leakage results when discrete signal analysis techniques, such as FFT and power spectrum analysis, are applied to a signal. The resulting transform is a convolution of the spectrum of interest and the spectrum of a square window. Spectral window filtering attempts to compensate for the limitations of the discretely sampled portion of the signal you are analyzing.

The Fourier transform of any large sample of data has a significantly high frequency makeup when it is calculated using a square window function. To compensate for the limitations of the discretely sampled portion of the signal, apply other window functions that change more gradually from zero to a maximum, and back to zero. Applying a spectral window to the data results in greater accuracy in estimating the power spectrum and less leakage in data analysis.

To apply a spectral window, multiply the sampled data by an even function of the number of samples centered around the midpoint of the discrete sample.

Each of the five spectral window types LSP supports reduces the main lobe peak width and the magnitude of the side lobes (leakage) to a slightly different degree. The compromise between these two factors depends on the window function you select. See Section 4.3, Spectral Window Routines and Algorithms, for algorithms of each of the window types. See Section 4.4, Spectral Window Filtering References, for further information.

## 4.2 The Periodogram Technique

The periodogram is a faster method of calculating the power spectrum of a sequence of data. The terms **periodogram** and **power spectrum** are often used interchangeably. In this application, the term periodogram refers to an average of individual Fourier space spectra.

To calculate the power spectrum of a data sequence using the periodogram technique, break a data sequence of length N into K segments, each of length M. Perform a Fourier transform on the M data for each of the K segments. Average the K arrays of data and square the result.

The periodogram technique provides a considerable time savings when the original data sequence is long. Computation time using the FFT algorithm is:

$$t_{FFT} \propto N \log N$$

Compare the ratio of computation time between K number of M-length data, where M = N/K, and one N-length sequence of data (denote this ratio as Beta):

$$Beta = \frac{t_{periodogram}}{t_{powerspectrum}} = \frac{K([M \log M])}{N \log N} = 1 - \frac{\log K}{\log N}$$

Evaluate the expression for Beta with the following numbers.

| N | K | Beta |
|---|---|---|
| 256 | 1 | 1.0 |
| | 2 | 0.875 |
| | 4 | 0.750 |
| 1024 | 1 | 1.0 |
| | 2 | 0.900 |
| | 4 | 0.800 |
| | 8 | 0.700 |
| 10000 | 1 | 1.0 |
| | 2 | 0.925 |
| | 4 | 0.849 |
| | 10 | 0.750 |
| 50000 | 1 | 1.0 |
| | 2 | 0.936 |
| | 5 | 0.851 |
| | 20 | 0.723 |

The preceding table illustrates the time saved by calculating the power spectrum in multiple segments rather than in one segment. For example, processing 10000 data in 10 segments takes 0.750 of the time it takes to process 10000 data in 1 segment. This represents a 25% time savings.

An important aspect of the periodogram technique is that the frequency scaling changes. As is the case for any Fourier (frequency) domain axis, the frequency is measured as $f = k/(n \Delta t)$, where k is a running index such that k = 0,1,2,... n-1, and n is the total number of data points to be transformed. In the case of the periodogram, n is now M, so that $f = K/(M \Delta t)$. When you do not use the periodogram method, if N = 100, the range of the frequency domain with K = 1 is 0 to (99/100)f with a resolution of (1/100)f. If K = 10, the range is 0 to (9/10)f with a resolution of (1/10)f. Note, when K is large, you lose resolution.

## 4.3 Spectral Window Routines and Algorithms

This section describes the three LSP spectral window routines, provides you with algorithms for each of the five spectral window types, and includes a sample FORTRAN program illustrating spectral window filtering and the periodogram technique.

LSP includes three spectral window filtering routines:

- LSP$SPECTRAL_WINDOWS

  The LSP$SPECTRAL_WINDOWS routine dynamically allocates a coefficient table, applies the coefficients to the raw input data, and stores the resulting windowed output data in a table.

- LSP$BUILD_WINDOW_TABLE

  The LSP$BUILD_WINDOW_TABLE routine builds an array of window function coefficients.

  Building the coefficient array separately allows you to save set-up time in a real-time computing environment where you intend to use the same window type repeatedly.

- LSP$APPLY_WINDOW_TABLE

  The LSP$APPLY_WINDOW_TABLE routine applies the coefficient array generated by the LSP$BUILD_WINDOW_TABLE routine to the raw input data, and stores the resulting windowed output data in an array.

**NOTE**

For further information about the LSP spectral window filtering routines and their arguments, see the appropriate routine reference description section in Chapter 6, Signal-Processing Routine Call Reference Descriptions.

The following is a list of the five spectral-window types and the algorithms used to generate them. For each window type, n = the number of data points and j is the input data at points 0,1,2,...n-1.

- Blackman Window (2 parts)

$$x = 2\pi\left(\frac{j - \left(\frac{n}{2}\right)}{n-1}\right)$$

$$w(j) = 0.42 + 0.5(\cos(x)) + 0.08 * (\cos(2x))$$

- Hamming Window

$$w(j) = 0.54 + 0.46\cos\left(\frac{2\pi\left(j - \frac{n}{2}\right)}{n-1}\right)$$

- Hanning Window

$$w(j) = 0.5 * \cos\left(\frac{2\pi\left(j - \frac{n}{2}\right)}{n-1}\right)$$

- Triangle Window

$$w(j) = 1.0 - 2 * \left(\frac{\left|j - \left(\frac{n}{2}\right)\right|}{n-1}\right)$$

- Welch Window

$$w(j) = 1 - \left(\frac{j - \frac{1}{2}(n-1)}{\frac{1}{2}(n+1)}\right)^2$$

Figure 4–1 illustrates the five spectral window types:

**Figure 4–1: The Five Spectral-Window Types**

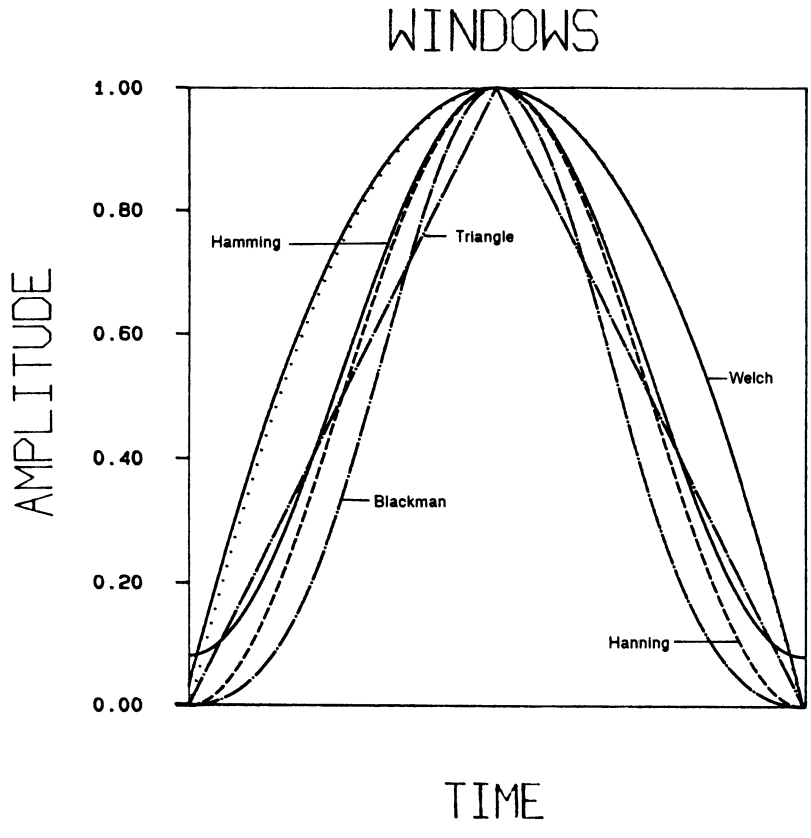


Figure 4–2 and Figure 4–3 illustrate the difference between raw, unwindowed output data and data windowed using the triangle window type.

**Figure 4–2: Raw, Unwindowed Data**

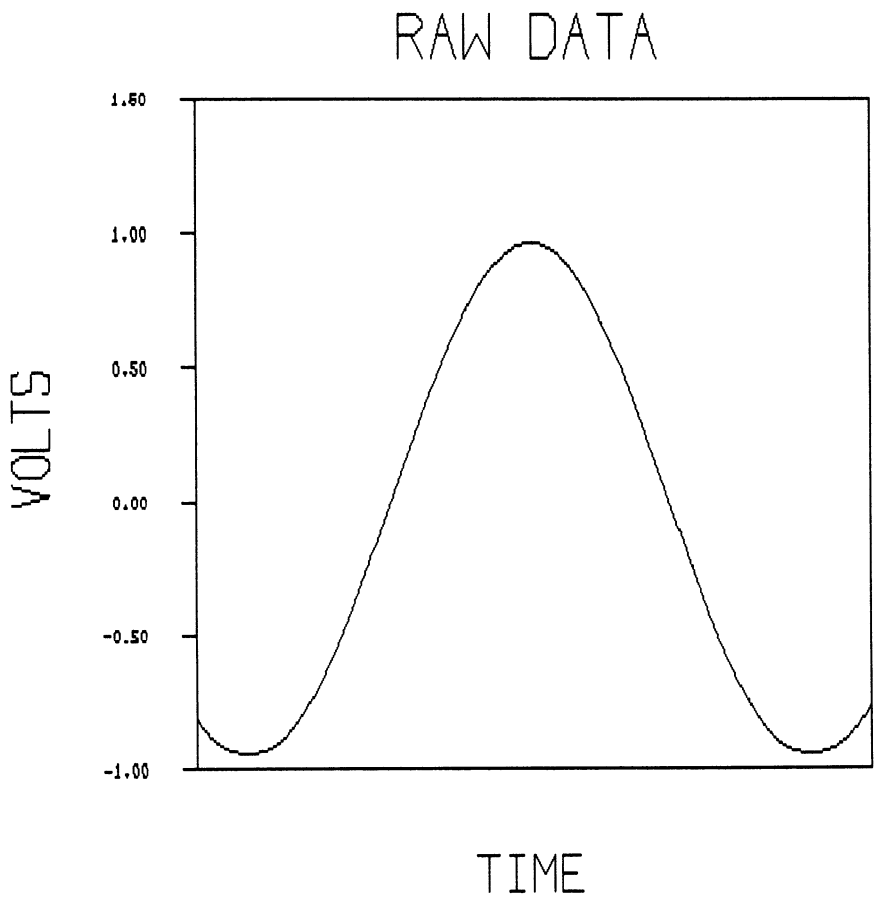
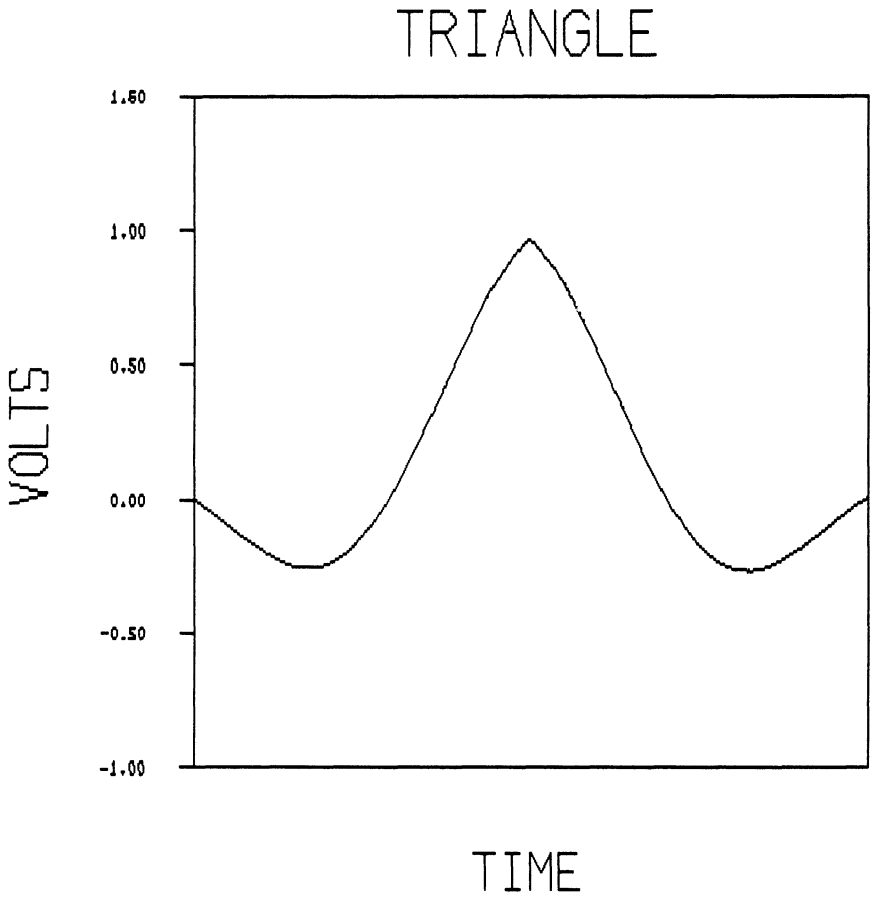


**Figure 4–3: Windowed Data**



Example 4–1, Applying a Spectral Window, illustrates the use of the periodogram technique, the LSP$BUILD_WINDOW_TABLE routine, and the LSP$APPLY_WINDOW_TABLE routine.

**Example 4–1: Applying a Spectral Window**

```
C LSP_WINDOW_1.FOR

C
C The following sample program demonstrates the use of the
C LSP$BUILD_WINDOW_TABLE routine, the LSP$APPLY_WINDOW_TABLE
C routine, and the periodogram technique.
C

C
C Include the symbolic-value definition files:
C

      INCLUDE 'sys$Library:LIOSET.FOR'
      INCLUDE 'sys$Library:LSPDEF.FOR'
      INCLUDE 'sys$Library:LSPSET.FOR'
C
C Declare variables and data types:
C

      REAL*4 PI,WAVE(2048),WINDOW(2048,5),FFT(2048),POWER(2048)
      REAL*4 RANGE(2),WAVE2(2048),XCONTROL(4),YCONTROL(4)
      INTEGER*4  I,J,L,N,NOW,NTERMS,Z,ISTAT,STATUS,CONTROL(3)

      INTEGER*2  BUFFER(2048+256) !2048 word buffer + 512 byte overrun
      INTEGER    AtoD_ID          !A/D device ID variable
      INTEGER    data_length      !number of data bytes read

C
C Use LIO to acquire a waveform:
C

      N = 2048  ! Number of points in array

C
C Attach to the A/D converter:
C
C Gets a device ID for the ADV and tells LIO to use QIO I/O.
C

      status = LIO$ATTACH(AtoD_ID, 'AZAO', LIO$K_QIO) !attach to ADV
               IF(.NOT.(status))       CALL lib$signal(%val(status))
```

**Example 4–1 Cont'd. on next page**

## Example 4–1 (Cont.): Applying a Spectral Window

```
C
C Set up the A/D:
C
C Set synchronous I/O (LIO$READ/LIO$WRITE)
C Set A/D channel zero
C Set A/D gain of 1
C Set trigger mode = start on LIO$READ and fill buffer
C as fast as possible
C

     status = LIO$SET_I(AtoD_ID, LIO$K_SYNCH, 0)
              IF(.NOT.(status))       CALL lib$signal(%val(status))
     status = LIO$SET_I(AtoD_ID, LIO$K_AD_CHAN, 1, 0)
              IF(.NOT.(status))       CALL lib$signal(%val(status))
     status = LIO$SET_I(AtoD_ID, LIO$K_AD_GAIN, 1, 1)
              IF(.NOT.(status))       CALL lib$signal(%val(status))
     status = LIO$SET_I(AtoD_ID, LIO$K_TRIG, 1, LIO$K_IMM_BURST)
              IF(.NOT.(status))       CALL lib$signal(%val(status))

          Z = ((N/2)+1)

C
C Set up the CONTROL and RANGE arrays for the
C LSP$FORMAT_TRANSLATE_ADC routine:
C

     CONTROL(1)=0       ! 2's complement format
     CONTROL(2)=12      ! number of bits in converter
     CONTROL(3)=1       ! gain of 1

     RANGE(1)= -10.0    ! set up low voltage range
     RANGE(2)= 9.9951   ! set up high voltage range

C
C Build a coefficient array for each of the five spectral
C window types:
C

     ISTAT=LSP$BUILD_WINDOW_TABLE(WINDOW(1,1),N,LSP$K_WELCH)
     ISTAT=LSP$BUILD_WINDOW_TABLE(WINDOW(1,2),N,LSP$K_HAMMING)
     ISTAT=LSP$BUILD_WINDOW_TABLE(WINDOW(1,3),N,LSP$K_HANNING)
     ISTAT=LSP$BUILD_WINDOW_TABLE(WINDOW(1,4),N,LSP$K_TRIANGLE)
     ISTAT=LSP$BUILD_WINDOW_TABLE(WINDOW(1,5),N,LSP$K_BLACKMAN)
```

**Example 4–1 Cont'd. on next page**

## Example 4–1 (Cont.): Applying a Spectral Window

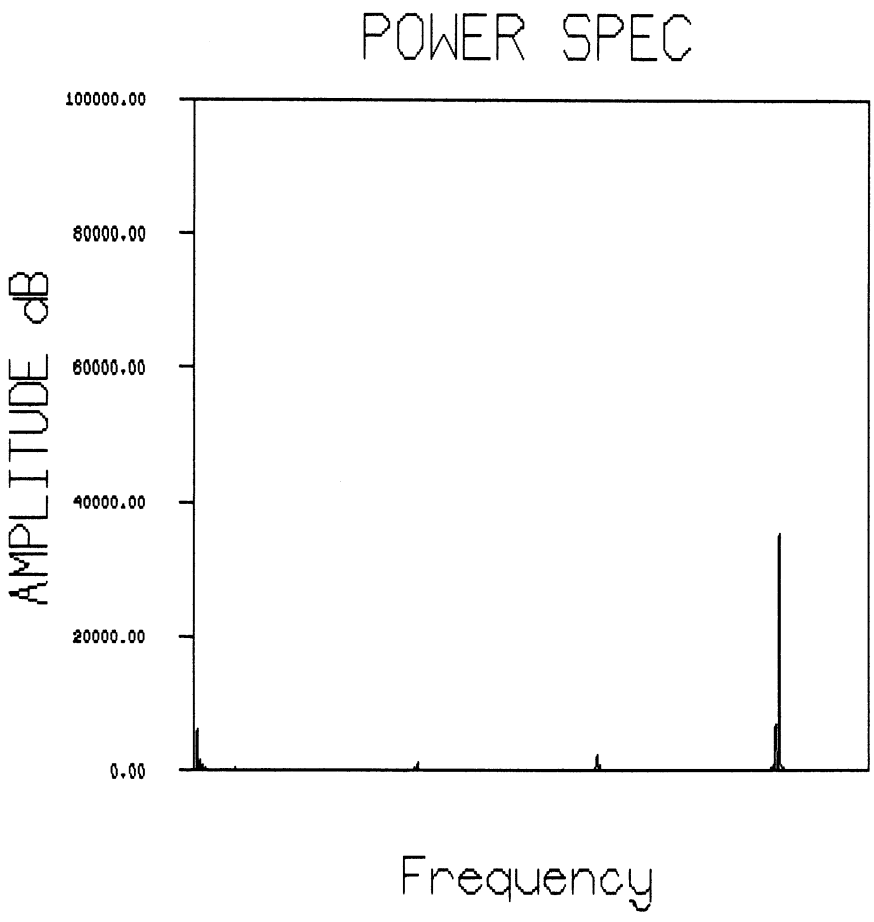
```
C
C Read 2048 A/D values (4096 bytes) into the buffer:
C
C The number of bytes transferred is returned in data_length.
C The device specific parameter is not used, so it is defaulted.
C

     status = LIO$READ(AtoD_ID, buffer, N, data_length, )
              IF(.NOT.(status))  CALL lib$signal(%val(status))

C
C Convert the A/D values to volts:
C

     ISTAT = LSP$FORMAT_TRANSLATE_ADC(BUFFER,WAVE,N,CONTROL,RANGE)
             IF (.NOT.ISTAT)      CALL LIB$SIGNAL (%VAL(ISTAT))

C
C Logically split the input data into 4 - 512 word buffers:
C
C The input data is split in order to calculate the power
C spectrum using the periodogram technique.

     NOW = 1

     DO 100 I=1,4

C
C Apply the Welch window function coefficient array to the segmented
C input data:
C
C Store the resulting array of windowed output data.  You can apply any
C of the five window function coefficient arrays to the input data by
C modifying the window_table argument in the LSP$APPLY_WINDOW_TABLE
C routine call.
C

     ISTAT=LSP$APPLY_WINDOW_TABLE(WAVE(NOW),WAVE2,(N/4),WINDOW(1,1))

     NOW = NOW + 512

C
C Calculate the fast Fourier transform for each of the four data segments
C in the table of windowed data:
C

     ISTAT=LSP$FFT_REAL(WAVE2,FFT,(N/4),0,STATUS)
```

Example 4–1 Cont'd. on next page

## Example 4–1 (Cont.): Applying a Spectral Window

```
C
C Add the four FFT calculations together:
C

     DO 75 J = 1,(N/4)
     POWER(J) = POWER(J) + FFT(J)
     75   CONTINUE

     100  CONTINUE

C
C Use the periodogram technique to calculate the power spectrum by
C averaging the sum of the four FFT calculations and squaring the
C result:
C

     DO 125 I = 1,(N/4)
     POWER(I) = POWER(I)/4.0
     POWER(I) = POWER(I)**2

     125  CONTINUE


C
C Plot the power spectrum results:
C

     CALL LGP$PLOT (1,'IXSY',,POWER,(N/4),'Frequency',
                    'AMPLITUDE dB',,,,,,,'POWER SPEC')
     CALL LGP$TERMINATE_PLOT (1,1)


C
C Detach the A/D device:
C
C Rundown is irrelevent for synchronous I/O.
C

     status = LIO$DETACH(AtoD_ID, )
              IF(.NOT.(status))       CALL lib$signal(%val(status))

     STOP
     END
```

Example 4–1 produces the following output:



The spectral window function symbolic status values are defined in definition files for each of the following program languages:

**Table 4–1: Spectral Window Function Symbolic Status Definition Files**

| Language | Symbolic Status Definition File |
|---|---|
| VAX Ada | SYS$LIBRARY:LSPSET.ADA |
| VAX BASIC | SYS$LIBRARY:LSPSET.BAS |
| VAX C | SYS$LIBRARY:LSPSET.H |
| VAX FORTRAN | SYS$LIBRARY:LSPSET.FOR |
| VAX MACRO | SYS$LIBRARY:LSPSET.MAR |
| VAX PASCAL | SYS$LIBRARY:LSPSET.PAS |

## 4.4 Spectral Window Filtering References

You can obtain further information about spectral window filtering from the following references:

Elliot, Douglas F. and Rao, K. Ramamohan. *Fast Transforms Algorithms, Analyses, Applications*. Academic Press, Inc., 1982.

Oppenheim, Alan V. and Shafer, Ronald W. *Digital Signal Processing*. Prentice Hall, Inc.,1975.

Stanley, William D. *Digital Signal Processing*. Reston Publishing Co. Inc., 1975.

Chapter 5

# Thermocouple Conversion

## 5.1 Overview of Thermocouple Conversion

The LSP$THERMOCOUPLE_X routines convert thermocouple voltages to temperatures. Table 5–1 shows the types of thermocouples that are supported.

**Table 5–1: Thermocouples with Conversion Routines**

| ANSI Symbol | Material | Trade Name |
|---|---|---|
| B | Platinum-6% Rhodium vs. Platinum-30% Rhodium | |
| E | Nickel-Chromium vs. Copper-Nickel | Chromel-Constantan |
| J | Iron vs. Copper-Nickel | Iron-Constantan |
| K | Nickel-Chromium vs. Nickel-Aluminum | Chromel-Alumel |
| R | Platinum vs. Platinum-13% Rhodium | |
| S | Platinum vs. Platinum-10% Rhodium | |
| T | Copper vs. Copper-Nickel | Copper-Constantan |

Each thermocouple type has a unique routine associated with it. The X in the LSP$THERMOCOUPLE_X routine is a placeholder for the ANSI symbol that represents the thermocouple type. When you use the LSP$THERMOCOUPLE_X routine, you substitute in place of the X the

ANSI symbol appropriate for the type of thermocouple. The following thermocouple conversion routines are provided:

LSP$THERMOCOUPLE_B
LSP$THERMOCOUPLE_E
LSP$THERMOCOUPLE_J
LSP$THERMOCOUPLE_K
LSP$THERMOCOUPLE_R
LSP$THERMOCOUPLE_S
LSP$THERMOCOUPLE_T

To use the thermocouple conversion routines, you set up your experiment with two identical thermocouples connected in series. You use one thermocouple for the actual experiment. You immerse the other thermocouple in an ice bath maintained at a constant temperature of 0 degrees Celsius as a reference. Alternatively, an electronic cold junction compensator can replace the reference thermocouple.

Table 5-2 shows the temperature and voltage ranges for each of the supported thermocouples.

**Table 5-2: Thermocouple Temperature and Voltage Ranges**

| Thermocouple ANSI Symbol | Temp. Range (Celsius) Min | Max | Voltage Range (Microvolts) Min | Max |
|---|---|---|---|---|
| B | 43 | 1820 | 0.21 | 13814.0 |
| E | -270 | 1000 | -9836.0 | 76358.0 |
| J | -210 | 1200 | -8096.0 | 69537.0 |
| K | -200 | 1372 | -5892.0 | 54875.0 |
| R | -50 | 1762 | -227.0 | 21108.17 |
| S | -50 | 1768 | -236.0 | 18698.16 |
| T | -270 | 400 | -6258.0 | 20870.0 |

See the LSP$THERMOCOUPLE_X routine call reference section in Chapter 6, Signal-Processing Routine Reference Descriptions, for more information about the thermocouple conversion routines.

Chapter 6

# Signal-Processing Routine Call Reference Descriptions

This chapter presents an overview and detailed reference descriptions of the Laboratory Signal-Processing routines used to perform Fourier transforms, correlation functions, filtering of data, and thermocouple conversion.

## 6.1 Overview of the Laboratory Signal-Processing Routine Format

Each LSP routine is described using a structured format:

- The **Routine name** appears at the top of the first page of each LSP routine reference description.
- The **Routine overview** explains, usually in a sentence or two, what the routine does.
- The **Format** section presents the routine entry point name, or routine name, and the routine argument list in the correct syntactical form.
- The **Returns** section lists the information returned by the routine.
- The **Arguments** section provides detailed information about each routine argument, such as what information the argument passes to the routine or what information the argument returns from the routine, and the data type, access, mechanism, and acceptable values of the argument.

- The **Description** section contains information about the specific actions taken by the routine. This includes interaction between routine arguments; interactions or dependencies between the routine and other LSP routines; restrictions for use; and actions specific to the routine when used with certain devices. In some cases, information of this nature is also found in the description of a routine argument under the **Arguments** heading.
- The **Condition Values** section contains a list of condition values a specific routine generates and a brief description of each condition value. See Section 7.3, Symbolic Status Values and Descriptions, for more detailed condition value information, including an explanation of symbolic status values generated by the routine and suggested user action.

## 6.2 Signal-Processing Routine Call Summary and Descriptions

Table 6–1 summarizes the signal-processing routines.

**Table 6–1: Signal-Processing Routine Call Summary**

| Routine Call | Function |
|---|---|
| LSP$APPLY_WINDOW_TABLE | Applies the coefficient array created by the LSP$BUILD_WINDOW_TABLE routine to the input data to create and store windowed output data |
| LSP$BUILD_WINDOW_TABLE | Builds an array of window function coefficients |
| LSP$CORRELATION | Calculates the cross-correlation or autocorrelation function of equispaced data |
| LSP$FFT_COMPLEX | Calculates the fast forward or inverse Fourier transform of complex-valued data |
| LSP$FFT_COMPLEX_2D | Calculates the fast forward or inverse Fourier transform of complex-valued data in two dimensions |

**Table 6–1 (Cont.): Signal-Processing Routine Call Summary**

| Routine Call | Function |
|---|---|
| LSP$FFT_REAL | Calculates the fast forward or inverse Fourier transform of real-valued data |
| LSP$FILTER_NONREC | General purpose, nonrecursive filter |
| LSP$FILTER_POLY | Polynomial filter for smoothing |
| LSP$FILTER_POLY_1ST_DERIV | Polynomial filter with first-derivative output |
| LSP$FILTER_POLY_2ND_DERIV | Polynomial filter with second-derivative output |
| LSP$FILTER_POLY_3RD_DERIV | Polynomial filter with third-derivative output |
| LSP$FORMAT_TRANSLATE_ADC | Translates raw numbers obtained from an analog-to-digital converter into floating-point voltages |
| LSP$FORMAT_TRANSLATE_DAC | Translates floating-point voltages into raw numbers appropriate for input to a digital-to-analog converter |
| LSP$HIST_F | Performs interval histogram analysis (multichannel analysis) with floating-point input |
| LSP$HIST_I | Performs interval histogram analysis (multichannel analysis) with integer input |
| LSP$PHASE_ANGLE | Calculates the phase angle and modulus spectrum for equispaced data in one dimension |
| LSP$PHASE_ANGLE_2D | Calculates the phase angle and modulus spectrum for equispaced data in two dimensions |
| LSP$POWER_SPECTRUM | Calculates the power spectrum of equispaced data |

**Table 6–1 (Cont.): Signal-Processing Routine Call Summary**

| Routine Call | Function |
|---|---|
| LSP$SPECTRAL_WINDOWS | Dynamically allocates a window function coefficient array, applies the coefficients to the input data, and stores the resulting windowed output data in an array |
| LSP$THERMOCOUPLE_X | Converts voltage to temperature for each thermocouple type |

The following reference section describes the signal-processing routines and their use.

# LSP$APPLY_WINDOW_TABLE

The LSP$APPLY_WINDOW_TABLE routine applies the window function coefficient array created by the LSP$BUILD_WINDOW_TABLE routine to the input data to be windowed.

## Format

**LSP$APPLY_WINDOW_TABLE** *(in, out, n, window_table, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*In*

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data to be windowed.

*out*

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the output data is returned. You can specify the same array for both the output array and the input array.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the size of the data set to be windowed. The value of **n** must be greater than or equal to 1.

***window_table***
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array of coefficients produced by the LSP$BUILD_WINDOW_TABLE routine.

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The LSP$APPLY_WINDOW_TABLE routine applies the window-function coefficient array created by the LSP$BUILD_WINDOW_TABLE routine to the input data contained in the **in** array. The LSP$APPLY_WINDOW_TABLE routine then places the resulting windowed output data into the **out** array.

See the LSP$BUILD_WINDOW_TABLE routine reference section for information on building a window-function coefficient array. Chapter 4, Spectral Window Filtering, provides further information about spectral windows.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$BUILD_WINDOW_TABLE

The LSP$BUILD_WINDOW_TABLE routine builds an array of spectral window-function coefficients. The spectral window-function coefficient array built by the LSP$BUILD_WINDOW_TABLE routine is used by the LSP$APPLY_WINDOW_TABLE routine.

## Format

**LSP$BUILD_WINDOW_TABLE** *(window_table, n, window_type, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***window_table***

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array of size n containing the returned spectral window-function coefficients.

***n***

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the number of table entries contained in the spectral window-function coefficient table.

***window_type***
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the type of window function you want to use. The following is a list of possible **window_type** values.

| Value | Window Type |
|---|---|
| LSP$K_BLACKMAN | Blackman |
| LSP$K_HAMMING | Hamming |
| LSP$K_HANNING | Hanning |
| LSP$K_TRIANGLE | Triangle |
| LSP$K_WELCH | Welch |

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The LSP$BUILD_WINDOW_TABLE routine calculates the **n** coefficients of the window function you select and places the results in the **window_table** array. The LSP$APPLY_WINDOW_TABLE routine applies the window function coefficient array to the input data. See the LSP$APPLY_WINDOW_TABLE routine reference section for information on applying the window function coefficient array generated by the LSP$BUILD_WINDOW_TABLE routine to the data.

Building the window function coefficient array and applying this coefficient array to the input data separately reduces set-up time in a realtime environment where a window function is used repeatedly. If you do not intend to use a window function repeatedly, you can use the LSP$SPECTRAL_WINDOWS routine. For further information on the LSP$SPECTRAL_WINDOWS routine, see the routine call reference description for that routine.

Chapter 4, Spectral Window Filtering, provides further information about spectral windows.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_ILL_WINDOW_TYPE | invalid window type specified |
| LSP$_SUCCESS | success |

# LSP$CORRELATION

The LSP$CORRELATION routine calculates the cross-correlation or autocorrelation function of equispaced data.

## Format LSP$CORRELATION *(in1, in2, out, n, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in1***

VMS Usage: **complex_number**
type: **F_floating complex**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data whose correlation function you want to estimate. The length of this array must be greater than or equal to **(n/2) + 1**.

The values contained in this array are the output of the Fourier transform routine LSP$FFT_REAL; all input arrays are given in the reduced-symmetric form. See Section 2.1.2, Mathematical Definition of Discrete Fourier Transform, for a description of the reduced-symmetric form.

*in2*

VMS Usage: **complex_number**
type: **F_floating complex**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data whose correlation function you want to estimate. The length of this array must be greater than or equal to **(n/2)** + 1. The data contained in **in1** and **in2** are the same when performing the autocorrelation function.

The values contained in this array are the output of the Fourier transform routine LSP$FFT_REAL; all input arrays are given in the reduced-symmetric form.

*out*

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the correlation function of the original time-based input data is returned. This array must be of length **n** + 2.

*n*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the total number of data points to be placed in the **out** array. This number must be a power of 2 and must range between 2 and 32,768, inclusive. Any number outside this range generates an error.

***status***

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

You use the correlation function to produce an estimate of the degree of similarity between two functions when one of the functions is shifted either in time or by some other independent variable. You can also use the correlation function on one function; this is known as autocorrelation.

See Section 2.3, Definition of the Correlation Function, for further information about the correlation and autocorrelation functions.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_NOT_2 | n is not a power of 2 |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FFT_COMPLEX

The LSP$FFT_COMPLEX routine calculates the fast forward or inverse Fourier transform of complex-valued data.

## Format LSP$FFT_COMPLEX *(in, out, n, direction, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*in*
VMS Usage: **complex_number**
type: **F_floating complex**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data. The array is of length **n**.

*out*
VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by reference, array reference**

An array into which the transform of the input data is returned. The array is of length **n**.

The output array can be the same as the input array. In this case, the output array overwrites the input array.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data points to be transformed. The total number of data points as input is $2^m$. The value of $m$ must be between 1 and 15, inclusive. The value of **n** must be between 2 and 32,768, inclusive.

*direction*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument that determines whether to perform the forward or inverse transform. If the value of **direction** is zero, forward transform is performed. If the value of **direction** is nonzero, inverse transformation is performed.

*status*
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The LSP$FFT_COMPLEX routine is used to compute the forward Fourier transformation of complex-valued data of length $n=2^m$, where $m$ is between 1 and 15, inclusive.

The time axis is considered to be equally spaced with time increment $\Delta t$. The resulting frequency axis from the forward transformation is in increments $k/(n\ \Delta t)$, where k = 0, 1, 2, 3, ... n-1 and $n$ is the total number of points to be transformed.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_NOT_2 | n is not a power of 2 |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FFT_COMPLEX_2D

The LSP$FFT_COMPLEX_2D routine calculates the fast forward or inverse Fourier transform of complex-valued data in two dimensions.

## Format

**LSP$FFT_COMPLEX_2D** *(in, out, n1, n2, direction, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*in*

VMS Usage: **complex_number**
type: **F_floating complex**
access: **read only**
mechanism: **by reference, array reference**

An array containing the data to be transformed. The array is dimensioned **n1**× **n2**.

*out*

VMS Usage: **complex_number**
type: **F_floating complex**
access: **write only**
mechanism: **by reference, array reference**

An array into which the result of the transformation is returned. The array is dimensioned **n1**×**n2**.

*n1*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the number of data points in each row. Input the number of data points as $2^m$. The value of $m$ must be between 1 and 15, inclusive. The value of **n1** must be between 2 and 32,768, inclusive.

*n2*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the number of data points in each column. Input the number of data points as $2^m$. The value of $m$ must be between 1 and 15, inclusive. The value of **n2** must be between 2 and 32,768, inclusive.

*direction*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument that determines whether to perform the forward or inverse transform. If the value of **direction** is 0, the forward transformation is performed. If the value of **direction** is not equal to 0, the inverse transformation is performed.

*status*
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

Use the LSP$FFT_COMPLEX_2D routine to compute the two-dimensional fast Fourier transform of complex-valued data. This routine calls LSP$FFT_COMPLEX to perform the calculations. The row (**n1**) and column (**n2**) sizes need not be the same size.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_NOT_2 | n1 or n2 is not a power of 2 |
| LSP$_ILL_N_RANGE | n1 or n2 is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FFT_REAL

The LSP$FFT_REAL routine calculates the fast forward or inverse Fourier transform of real-valued data.

## Format LSP$FFT_REAL *(in, out, n, direction, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*In*

VMS Usage: **floating_point or complex_number**
type: **F_floating or F_floating complex**
access: **read only**
mechanism: **by reference, array reference**

For forward transform, where **direction** equals zero, a real array containing the data to be transformed. This array is of length **n**.

For the inverse transform, where **direction** is nonzero, a complex array containing the data to be transformed. This array is of length *(n/2) + 1* and contains the first half of the symmetric array in reduced-symmetric form.

*out*
VMS Usage: **floating_point or complex_number**
type: **F_floating or F_floating complex**
access: **write only**
mechanism: **by reference, array reference**

For the forward transform, where **direction** equals zero, a complex array into which the transform of the input data is returned. This array is of length *(n/2)+1* and contains the first half of the symmetric array.

For inverse transform, where **direction** is nonzero, a real array into which the transformed data is returned. This array is of length **n**.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data points to be transformed. The total number of data points as input is $2^m$. The value of *m* must be between 1 and 15, inclusive. The value of **n** must be between 2 and 32,768, inclusive.

For both the forward and inverse transform, the **in** and **out** arrays can be the same array. However, the array must be dimensioned to the larger of the two arrays.

*direction*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument that determines whether to perform forward or inverse transform. If the value of **direction** is zero, forward transform is performed. If the value of **direction** is nonzero, inverse transform is performed.

## LSP$FFT_REAL

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

Use the LSP$FFT_REAL routine to compute the fast Fourier transform of real-valued data of length $n=2^m$, where $m$ is an integer between 1 and 15, inclusive. Because the input data is known to contain no imaginary values, the transform is twice as fast as the LSP$FFT_COMPLEX routine. Approximately half as many operations are required to perform the fast Fourier transform.

As is the case with all FFT methods, the time axis is considered to be equally spaced with time increment $\Delta t$. The resulting frequency axis from the forward transformation is in increments k/(n $\Delta t$) where k = 0, 1, 2, 3, ... n/2 and n is the total number of points to be transformed.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_NOT_2 | n is not a power of 2 |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FILTER_NONREC

The LSP$FILTER_NONREC routine performs filtering in lowpass, highpass, bandpass, or bandstop (notch) mode.

## Format

**LSP$FILTER_NONREC** *(in, out, n, flow, fhigh, wiggles, nterms, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*in*

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data to be filtered.

*out*

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the filtered data is returned. The **out** array can be the same as the **in** array.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data values to be filtered. This value must be greater than the value of **nterms***2 + 1. Any number less than or equal to this value generates an error.

*flow*
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference**

An argument representing the lower frequency of the filter. This number is given as a fraction of the Nyquist sampling frequency (1/(2 $\Delta$t)) and must be between 0.0 and 1.0, inclusive. See Section 3.1.2, Nonrecursive Filtering, for more information on the **flow** argument.

*fhigh*
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference**

An argument representing the upper frequency of the filter. This number is given as a fraction of the Nyquist sampling frequency (1/(2 $\Delta$t)) and must be between 0.0 and 1.0, inclusive. See Section 3.1.2, Nonrecursive Filtering, for more information on the **fhigh** argument.

*wiggles*
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference**

A number in -dB units which is proportional to the oscillation from the Gibbs phenomenon. This number must be between 0.0 and 500.0, inclusive. See Section 3.1.2, Nonrecursive Filtering, for more information on the **wiggles** argument.

***nterms***

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

A variable with a value between 2 and 500, inclusive. Any number outside this range generates an error.

***status***

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

Use the LSP$FILTER_NONREC routine to perform nonrecursive filtering in either lowpass, highpass, bandpass, or bandstop (notch) mode.

See Section 3.1.2, Nonrecursive Filtering, for further information.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_F_RANGE | flow or fhigh is out of range |
| LSP$_ILL_FLOW | flow is equal to fhigh |
| LSP$_ILL_N_NONREC | n is less than 2*nterms+1 |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_ILL_NTERMS | nterms is out of range |
| LSP$_ILL_WIGGLES | wiggles is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FILTER_POLY

The LSP$FILTER_POLY routine performs polynomial filtering for smoothing.

## Format LSP$FILTER_POLY *(in, out, n, filtyp, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data.

***out***
VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the output data is returned. The **out** array can be the same as the **in** array; then, the output array replaces the input array.

***n***
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data points to be filtered.

*filtyp*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing one of the following codes:

| Code | Description | |
|---|---|---|
| 1 | smoothing: 5-point window | ——+ |
| 2 | smoothing: 7-point window | \| |
| 3 | smoothing: 9-point window | \| |
| 4 | smoothing: 11-point window | \| |
| 5 | smoothing: 13-point window | QUADRATIC-CUBIC |
| 6 | smoothing: 15-point window | BASIS FUNCTION |
| 7 | smoothing: 17-point window | \| |
| 8 | smoothing: 19-point window | \| |
| 9 | smoothing: 21-point window | \| |
| 10 | smoothing: 23-point window | \| |
| 11 | smoothing: 25-point window | ——+ |
| 12 | smoothing: 7-point window | ——+ |
| 13 | smoothing: 9-point window | \| |
| 14 | smoothing: 11-point window | \| |
| 15 | smoothing: 13-point window | \| |
| 16 | smoothing: 15-point window | QUARTIC-QUINTIC |
| 17 | smoothing: 17-point window | BASIS FUNCTION |
| 18 | smoothing: 19-point window | \| |
| 19 | smoothing: 21-point window | \| |
| 20 | smoothing: 23-point window | \| |
| 21 | smoothing: 25-point window | ——+ |

*status*
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

# LSP$FILTER_POLY

## Description

You use the technique of digital filtering to eliminate certain frequency components from a signal that is corrupted by noise. Polynomial filters are based on simple interpolating polynomials which act as lowpass filters.

See Section 3.1.1, Polynomial Filtering, for further information.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_FILTYP | filtyp is out of range |
| LSP$_ILL_N_FILTER | too few data points per filter window |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FILTER_POLY_1ST_DERIV

The LSP$FILTER_POLY_1ST_DERIV routine performs polynomial filtering with first-derivative output.

## Format

**LSP$FILTER_POLY_1ST_DERIV** *(in, out, n, filtyp, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data.

***out***

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the output data is returned. The **out** array can be the same as the **in** array; then, the output array replaces the input array.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data points to be filtered.

*filtyp*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing one of the following codes:

| Code | Description | |
|---|---|---|
| 1 | 1st derivative: 5-point window | —— + |
| 2 | 1st derivative: 7-point window | \| |
| 3 | 1st derivative: 9-point window | \| |
| 4 | 1st derivative: 11-point window | \| |
| 5 | 1st derivative: 13-point window | QUADRATIC |
| 6 | 1st derivative: 15-point window | BASIS FUNCTION |
| 7 | 1st derivative: 17-point window | \| |
| 8 | 1st derivative: 19-point window | \| |
| 9 | 1st derivative: 21-point window | \| |
| 10 | 1st derivative: 23-point window | \| |
| 11 | 1st derivative: 25-point window | —— + |
| 12 | 1st derivative: 5-point window | —— + |
| 13 | 1st derivative: 7-point window | \| |
| 14 | 1st derivative: 9-point window | \| |
| 15 | 1st derivative: 11-point window | \| |
| 16 | 1st derivative: 13-point window | CUBIC-QUARTIC |
| 17 | 1st derivative: 15-point window | BASIS FUNCTION |
| 18 | 1st derivative: 17-point window | \| |
| 19 | 1st derivative: 19-point window | \| |
| 20 | 1st derivative: 21-point window | \| |
| 21 | 1st derivative: 23-point window | \| |
| 22 | 1st derivative: 25-point window | —— + |

```
23      1st derivative:  7-point window  --+
24      1st derivative:  9-point window    |
25      1st derivative: 11-point window    |
26      1st derivative: 13-point window    |
27      1st derivative: 15-point window  QUINTIC-SEXIC
28      1st derivative: 17-point window  BASIS FUNCTION
29      1st derivative: 19-point window    |
30      1st derivative: 21-point window    |
31      1st derivative: 23-point window    |
32      1st derivative: 25-point window  --+
```

***status***

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

Returns the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

You use the technique of digital filtering to eliminate certain frequency components from a signal that is corrupted by noise. Polynomial filters are based on simple interpolating polynomials which act as lowpass filters capable of producing derivative information.

See Section 3.1.1, Polynomial Filtering, for further information.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_FILTYP | filtyp is out of range |
| LSP$_ILL_N_FILTER | too few data points per filter window |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FILTER_POLY_2ND_DERIV

The LSP$FILTER_POLY_2ND_DERIV routine performs polynomial filtering with second-derivative output.

## Format LSP$FILTER_POLY_2ND_DERIV *(in, out, n, filtyp, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data.

***out***
VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the output data is returned. The **out** array can be the same as the **in** array; then, the output array replaces the input array.

*n*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data points to be filtered.

*filtyp*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing one of the following codes:

| Code | Description | |
|---|---|---|
| 1 | 2nd derivative: 5-point window | ——+ |
| 2 | 2nd derivative: 7-point window | \| |
| 3 | 2nd derivative: 9-point window | \| |
| 4 | 2nd derivative: 11-point window | \| |
| 5 | 2nd derivative: 13-point window | QUADRATIC-CUBIC |
| 6 | 2nd derivative: 15-point window | BASIS FUNCTION |
| 7 | 2nd derivative: 17-point window | \| |
| 8 | 2nd derivative: 19-point window | \| |
| 9 | 2nd derivative: 21-point window | \| |
| 10 | 2nd derivative: 23-point window | \| |
| 11 | 2nd derivative: 25-point window | ——+ |
| 12 | 2nd derivative: 7-point window | ——+ |
| 13 | 2nd derivative: 9-point window | \| |
| 14 | 2nd derivative: 11-point window | \| |
| 15 | 2nd derivative: 13-point window | \| |
| 16 | 2nd derivative: 15-point window | QUARTIC-QUINTIC |
| 17 | 2nd derivative: 17-point window | BASIS FUNCTION |
| 18 | 2nd derivative: 19-point window | \| |
| 19 | 2nd derivative: 21-point window | \| |
| 20 | 2nd derivative: 23-point window | \| |
| 21 | 2nd derivative: 25-point window | ——+ |

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

You use the technique of digital filtering to eliminate certain frequency components from a signal that is corrupted by noise. Polynomial filters are based on simple interpolating polynomials which act as lowpass filters capable of producing derivative information.

See Section 3.1.1, Polynomial Filtering, for further information.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_FILTYP | filtyp is out of range |
| LSP$_ILL_N_FILTER | too few data points per filter window |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FILTER_POLY_3RD_DERIV

The LSP$FILTER_POLY_3RD_DERIV routine performs polynomial filtering with third-derivative output.

## Format

**LSP$FILTER_POLY_3RD_DERIV** *(in, out, n, filtyp, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***In***

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data.

***out***

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the output data is returned. The **out** array can be the same as the **in** array; then, the output array replaces the input array.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data points to be filtered.

*filtyp*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing one of the following codes:

| Code | Description | |
|---|---|---|
| 1 | 3rd derivative: 5-point window | —— + |
| 2 | 3rd derivative: 7-point window | \| |
| 3 | 3rd derivative: 9-point window | \| |
| 4 | 3rd derivative: 11-point window | \| |
| 5 | 3rd derivative: 13-point window | CUBIC-QUARTIC |
| 6 | 3rd derivative: 15-point window | BASIS FUNCTION |
| 7 | 3rd derivative: 17-point window | \| |
| 8 | 3rd derivative: 19-point window | \| |
| 9 | 3rd derivative: 21-point window | \| |
| 10 | 3rd derivative: 23-point window | \| |
| 11 | 3rd derivative: 25-point window | —— + |
| 12 | 3rd derivative: 7-point window | —— + |
| 13 | 3rd derivative: 9-point window | \| |
| 14 | 3rd derivative: 11-point window | \| |
| 15 | 3rd derivative: 13-point window | \| |
| 16 | 3rd derivative: 15-point window | QUINTIC-SEXIC |
| 17 | 3rd derivative: 17-point window | BASIS FUNCTION |
| 18 | 3rd derivative: 19-point window | \| |
| 19 | 3rd derivative: 21-point window | \| |
| 20 | 3rd derivative: 23-point window | \| |
| 21 | 3rd derivative: 25-point window | —— + |

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

You use the technique of digital filtering to eliminate certain frequency components from a signal that is corrupted by noise. Polynomial filters are based on simple interpolating polynomials which act as lowpass filters capable of producing derivative information.

See Section 3.1.1, Polynomial Filtering, for further information.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_FILTYP | filtyp is out of range |
| LSP$_ILL_N_FILTER | too few data points per filter window |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FORMAT_TRANSLATE_ADC

The LSP$FORMAT_TRANSLATE_ADC routine translates raw numbers obtained from an analog-to-digital converter into floating-point voltages.

## Format

**LSP$FORMAT_TRANSLATE_ADC** *(in, out, [n], [control_i], [range], [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference, array reference**

An array containing the data from an analog-to-digital converter. This array is of length n.

***out***

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the result of the translation in units of voltage is returned. The array is of length n.

*n*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the number of data to be translated. The default is one datum.

*control_i*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference, array reference**

An array of length three specifying the type of number representation used by the converter, the number of bits, and the external gain:

control_i(1) — The number representation. You can designate binary or binary offset by specifying any nonzero integer. You can specify two's complement by specifying zero.

control_i(2) — The number of bits. Specify an integer between 6 and 16, inclusive. Any number outside the range generates an error. Bits to be operated on are right-justified in the 16-bit word. Bits to the left of the data bits are ignored during this translation.

control_i(3) — The external gain. This value is helpful when using programmable-gain ADCs. The gain factor divides the voltage reading to reflect the actual voltage used. For example, a signal of one volt is applied to an ADC with a gain of four. The output of the converter reads four volts. The converter reading must be divided by the gain factor to reflect the actual input voltage.

Specify an integer between 1 and 20,000, inclusive. Any number outside the range generates an error.

The default values are two's complement, 12 bits, and a gain of one.

***range***
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing two values specifying the range of data expressed as volts. The first value specifies the lowest voltage; the second value specifies the highest voltage. The default is -10 to 9.9951 volts.

If the two values are equal, an error occurs.

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The LSP$FORMAT_TRANSLATE_ADC routine converts ADC 16-bit word input data to a floating-point quantity in units of voltage.

See Section 1.2, Data Format Translation for ADCs and DACs, for further information.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_CTROL_2_T | 2nd control array entry is out of range |
| LSP$_ILL_CTROL_3_T | 3rd control array entry is out of range |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_ILL_RANGE | both entries in the range array are equal |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$FORMAT_TRANSLATE_DAC

The LSP$FORMAT_TRANSLATE_DAC routine translates a floating-point voltage into raw numbers appropriate for input to a digital-to-analog converter.

## Format

**LSP$FORMAT_TRANSLATE_DAC** *(in, out, [n], [control_i], [range], [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*in*

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the data for a digital-to-analog converter (DAC) in units of volts. This array is of length **n**.

*out*

VMS Usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
mechanism: **by reference, array reference**

An array into which the data to be passed to the DAC is returned. This array is of length **n**.

*n*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the number of data to be translated. The default is one datum.

*control_i*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An array of length three specifying the type of number representation used by the converter, the number of bits, and the external gain:

control_i(1) — The number representation. You can designate binary or binary offset by specifying any nonzero integer. You can specify two's complement by specifying zero.

control_i(2) — The number of bits. Specify an integer between 6 and 16, inclusive. Any number outside the range generates an error. Bits to be operated on are right-justified in the 16-bit word. Bits to the left of the data bits are filled with zeros in this translation.

control_i(2) — The external gain. This value is helpful when using programmable gain DACs. The gain factor divides into the voltage reading to reflect the actual voltage used. For example, a signal of one volt is requested to DAC with a gain of four. The output of the DAC reads one volt. The voltage is divided by the gain factor to reflect the actual input voltage.

Specify an integer between 1 and 20,000, inclusive. Any number outside the range generates an error.

The default values are two's complement, 12 bits, and a gain of one.

***range***
VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array containing two values specifying the range of data expressed as volts. The first value specifies the lowest voltage; the second value specifies the highest voltage. The default is -10 to 9.9951 volts.

If the two values are equal, an error occurs.

If you specify input data values greater than the maximum value or less than the minimum value of the **range** array, the **out** array contains the maximum or minimum values as set by the **range** array. For example, if you specify -20 volts but the **range** array specifies -10 as the minimum voltage, the **out** array contains the equivalent of -10. If you specify +20 volts but the **range** array specifies 9.9951 as the maximum voltage, the **out** array contains the equivalent of 9.9951 as the maximum voltage.

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The LSP$FORMAT_TRANSLATE_DAC routine converts DAC floating-point quantities in units of voltage to 16-bit word output format.

See Section 1.2, Data Format Translation for ADCs and DACs, for further information.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_CTROL_2_T | 2nd control array entry is out of range |
| LSP$_ILL_CTROL_3_T | 3rd control array entry is out of range |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_ILL_RANGE | both entries in the range array are equal |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$HIST_F

The LSP$HIST_F routine performs interval histogram analysis with floating-point input.

## Format LSP$HIST_F *(in, out, n, control, [info], [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array of length **n** containing the values to be histogrammed.

***out***
VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the number of occurrences of the input data is returned. The third entry in the **control** array determines the length of the **out** array.

If the input and output arrays are the same, this generates an error.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the number of data points used as input from the input array. This number must be greater than zero.

*control*
VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array of length four specifying the following histogram analysis parameters:

control(1) The first value specifies the lower limit of the first interval histogram element. This is the minimum range used for histogramming. If this value is equal to the second value in the array, an error occurs.

control(2) The second value specifies the upper limit of the last interval histogram element. This is the maximum range used for histogramming. If this value is equal to the first value in the array, an error occurs.

control(3) The third value specifies the total number of intervals. This number must be greater than zero. This value determines the length of the **out** array.

control(4) The fourth value specifies the inclusion of the input data if the data is equal to the highest value (as specified in the second array entry) in the histogram:

- The value 0.0 specifies omitting the input data from the histogram calculation.
- Nonzero specifies including the input data in the histogram calculation.

*Info*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference, array reference**

An array which produces two values specifying:

1. The number of input data less than the minimum data value included in the histogram calculation.
2. The number of input data greater than the maximum data value included in the histogram calculation. The fourth entry in the **control** array determines this value. When the fourth entry is zero and the input data is equal to the second entry in the **control** array, then the data is considered out of range.

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The LSP$HIST_F routine counts the number of floating-point data elements that fall into one or more predefined categories or limits. This routine can convert an array of data into a histogram representing the frequency of occurrence of the data in a window with defined limits.

The LSP$HIST_F and LSP$HIST_I **out** array has the following format:

$$\Delta x = \frac{control(2) - control(1)}{control(3)}$$

where

control(1), control(2), and control(3) are the first, second, and third entries in the **control** array.

The **out** array has the following format:

| | |
|---|---|
| 1st entry | The number of occurrences of input data greater than or equal to control(1), and less than $\Delta x$ plus control(1). |
| 2nd entry | The number of occurrences of input data greater than or equal to $\Delta x$ plus control(1), and less than (2 * $\Delta x$) plus control(1). |
| kth entry | If k is not equal to control(3), then the number of occurrences of input data greater than or equal to ((k - 1) * $\Delta x$) plus control(1), and less than k * $\Delta x$ plus control(1).<br>If control(4) is equal to 0 and k = control(3), then the number of occurrences of input data greater than or equal to ((k - 1) * $\Delta x$) plus control(1), and less than k * $\Delta x$ plus control(1) is obtained.<br>However, if control(4) is not equal to 0 and k = control(3), then the number of occurrences is determined by when the input data is greater than or equal to ((k - 1) * $\Delta x$) plus control(1), and less than or equal to k * $\Delta x$ plus control(1). |

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_ARRAY | input and output arrays cannot be the same |
| LSP$_ILL_CTROL_1_H | entries 1 and 2 in control array are equal |
| LSP$_ILL_CTROL_3_H | 3rd control array entry is less than 1 |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$HIST_I

The LSP$HIST_I routine performs interval histogram analysis with integer input.

## Format LSP$HIST_I *(in, out, n, control_i, [info], [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*in*
VMS Usage: **word_signed**
type: **word (signed)**
access: **read only**
mechanism: **by reference, array reference**

An array of length **n** containing the values to be histogrammed.

*out*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference, array reference**

An array into which the number of occurrences of the input data is returned. The third entry in the **control** array determines the length of the **out** array.

If the **in** and **out** arrays are the same, this generates an error.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the number of data points used as input from the in array. This number must be greater than zero.

*control_i*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference, array reference**

An array of length four specifying the following histogram analysis parameters:

control_i(1) The first value specifies the lower limit of the first interval histogram element. This is the minimum range used for histogramming. If this value is equal to the second value in the array, an error occurs.

control_i(2) The second value specifies the upper limit of the last interval histogram element. This is the maximum range used for histogramming. If this value is equal to the first value in the array, an error occurs.

control_i(3) The third value specifies the total number of intervals. This number must be greater than zero. This value determines the length of the **out** array.

control_i(4) The fourth value specifies the inclusion of the input data if the data is equal to the highest value (as specified in value two) in the histogram:

- The value 0 specifies omitting the input data from the histogram calculation.
- Nonzero specifies including the input data in the histogram calculation.

*info*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **write only**
mechanism: **by reference, array reference**

An array containing two values specifying:

1. The number of input data less than the minimum data value included in the histogram calculation.
2. The number of input data greater than the maximum data value included in the histogram calculation. The fourth entry in the **control_i** array determines this value. When the fourth entry is zero and the input data is equal to the second entry in the **control_i** array, then the data is considered out of range.

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

See the **Description** section under the LSP$HIST_F routine for details about the format of the LSP$HIST_I routine **out** array.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_ARRAY | input and output arrays cannot be the same |
| LSP$_ILL_CTROL_1_H | entries 1 and 2 in control array are equal |
| LSP$_ILL_CTROL_3_H | 3rd control array entry is less than 1 |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$PHASE_ANGLE

The LSP$PHASE_ANGLE routine calculates the phase angle and modulus spectrum for data in one dimension.

## Format

**LSP$PHASE_ANGLE** *(in, phase_out, modulus_out, n, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***
VMS Usage: **complex_number**
type: **F_floating complex**
access: **read only**
mechanism: **by reference, array reference**

An array containing the complex output values from the FFT routine. This array is **n** entries long.

***phase_out***
VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array returning the phase angle of the input data. This array is **n** entries long and can be the same as the **in** array.

*modulus_out*
VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array returning the modulus of the input data. This array is **n** entries long and can be the same as the **in** array.

The **phase_out** argument and the **modulus_out** argument can both be the same as the **in** argument. If both output arguments are the same as the input argument, the modulus appears in the array, and the phase angle information is lost.

*n*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of complex input data points for which the phase angle and modulus are to be calculated. The value of **n** must be greater than or equal to 1.

*status*
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The LSP$PHASE_ANGLE routine returns the phase angle and the vector resultant amplitude (modulus) of the signal whose Fourier transform h(f) has been previously obtained. This routine performs the following calculations:

$$pha(f_i) = arctan(I_i/R_i)$$

and

$$r(f_i) = \sqrt{(R_i)^2 + (I_i)^2}$$

where

*pha*($f_i$) is the phase angle (in radians) of a signal with Fourier components $R_i$ and $I_i$.

$R_i$ and $I_i$ are the real and imaginary coefficients of the Fourier transform.

r($f_i$)is the modulus of the signal.

Note that *i* is an index in the above equation.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_ARRAY | input and output arrays cannot be the same |
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$PHASE_ANGLE_2D

The LSP$PHASE_ANGLE_2D routine computes the magnitude and phase of a complex, two-dimensional array for which the Fourier transform has been previously obtained.

## Format

**LSP$PHASE_ANGLE_2D** *(in, out1, out2, n1, n2, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***

VMS Usage: **complex_number**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the data from which the phase angle and amplitude are calculated. This array is dimensioned **n1**×**n2**.

***out1***

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the phase angle of the input data is returned. This array is dimensioned **n1**× **n2**.

*out2*
VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the magnitude is returned. This array is dimensioned **n1**×**n2**.

*n1*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of points in each row. This value must be greater than or equal to 1.

*n2*
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of columns. This value must be greater than or equal to 1.

*status*
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

# LSP$PHASE_ANGLE_2D

## Description

The LSP$PHASE_ANGLE_2D routine returns the phase angle and the vector resultant amplitude (modulus) of the signal whose Fourier transform h(f) has been previously obtained. This routine performs the following calculations:

$$pha(f_{i,j}) = arctan(I_{i,j} / R_{i,j})$$

and

$$r(f_{i,j}) = \sqrt{(R_{i,j})^2 + (I_{i,j})^2}$$

where

$pha(f_{i,j})$ is the phase angle (in radians) of a signal with Fourier components $R_{i,j}$ and $I_{i,j}$
$R_{i,j}$ and $I_{i,j}$ are the real and imaginary coefficients of the Fourier transform
$r(f_{i,j})$ is the modulus of the signal

Note that $i$ and $j$ are indices in the above equation.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_ARRAY | input and output arrays cannot be the same |
| LSP$_ILL_N_RANGE | n1 or n2 is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$POWER_SPECTRUM

The LSP$POWER_SPECTRUM routine calculates the power spectrum of equispaced data.

## Format LSP$POWER_SPECTRUM *(in, out, n, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

*in*

VMS Usage: **complex_number**
type: **F_floating complex**
access: **read only**
mechanism: **by reference, array reference**

An array containing the complex output values from the FFT routine. This array is n entries long.

*out*

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the power spectrum of the input data is returned. This array is n entries long and can be the same array as the in array.

***n***
VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of complex input data points for which the power spectrum is to be calculated. The quantity **n** must be greater than or equal to 1.

***status***
VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

The power spectrum of a record of data can be estimated from a real-valued evenly-spaced sequence of data. The power spectrum is commonly defined as:

$$P(f) = h(f) * h^*(f)$$

where

h(f) is the Fourier transform of h(t).
$h^*$(f) is the complex conjugate of h(f).

Separating the real and imaginary terms of both h(f) and $h^*$(f) yields:

$$P(f_i) = (R_i + j * I_i) * (R_i - j * I_i)$$

where

$R_i$ are the real components of the Fourier transform of $h(t_i)$.
$I_i$ are the imaginary components of the Fourier transform of $h(t_i)$.
j is equal to the square root of -1.

Note that *i* is an index in the above equation.

Multiplying the two terms on the right side of the equation yields:

$$P(f_i) = (R_i)^2 + (I_i)^2$$

The Fourier transform of a real-valued data record has a symmetry property whereby only half of the transform needs to be stored. When the nonstored portion of the forward real FFT is substituted into the equation, the same result is produced. Because of this symmetry, only the first (n/2)+1 values need to be stored.

As is the case with all FFT methods, the time axis is considered to be equally spaced with time increment $\Delta t$. The resulting frequency axis from the forward transformation is in increments k/(n $\Delta t$)

where k = 0, 1, 2, 3, ... n-1, and n is the total number of points to be transformed.

The routine LSP$POWER_SPECTRUM is called after performing a forward FFT on the input data.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

# LSP$SPECTRAL_WINDOWS

The LSP$SPECTRAL_WINDOWS routine dynamically allocates a window function coefficient array, applies the coefficients to the input data, and stores the resulting windowed output data in an array.

## Format

**LSP$SPECTRAL_WINDOWS** *(in, out, n, window_type, [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***in***

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the input data to be windowed.

***out***

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array into which the windowed output data is returned. The **out** array can be the same as the in array; then, the output array replaces the input array.

*n*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data points to be windowed. The value of **n** must be greater than or equal to 1.

***window_type***

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument specifying the type of window to be used. The following is a list of possible **window type** values.

| Value | Window_Type |
|---|---|
| LSP$K_BLACKMAN | Blackman |
| LSP$K_HAMMING | Hamming |
| LSP$K_HANNING | Hanning |
| LSP$K_TRIANGLE | Triangle |
| LSP$K_WELCH | Welch |

***status***

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

# LSP$SPECTRAL_WINDOWS

## Description

The LSP$SPECTRAL_WINDOWS routine automatically allocates a window function coefficient array, applies the coefficients to the input data contained in the **in** array, and stores the resulting windowed output data in the **out** array. See Chapter 4, Spectral Window Filtering, for more information about spectral windows.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_ILL_WINDOW_TYPE | you specified an invalid window type |
| LSP$_SUCCESS | success |

# LSP$THERMOCOUPLE_X

The LSP$THERMOCOUPLE_X routine converts thermocouple voltages to temperatures.

## Format

**LSP$THERMOCOUPLE_X** *(volts, tempc, [n], [status])*

## Returns

VMS Usage: **cond_value**
type: **longword (unsigned)**
access: **write only**
mechanism: **by value**

## Arguments

***volts***

VMS Usage: **floating_point**
type: **F_floating**
access: **read only**
mechanism: **by reference, array reference**

An array containing the voltages produced from the thermocouple device referenced to a cold junction.

***tempc***

VMS Usage: **floating_point**
type: **F_floating**
access: **write only**
mechanism: **by reference, array reference**

An array returning the temperatures in degrees Centigrade.

*n*

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

An argument containing the number of data values to be converted. The default is 1.

***status***

VMS Usage: **longword_unsigned**
type: **longword (unsigned)**
access: **write only**
mechanism: **by reference**

An argument returning the status of the operation. If the **status** argument is omitted and an error occurs, the message is directed to both SYS$OUTPUT and SYS$ERROR.

## Description

Each thermocouple type has a unique routine associated with it. The X in the LSP$THERMOCOUPLE_X routine is a placeholder for the ANSI symbol that represents the thermocouple type. When you use the LSP$THERMOCOUPLE_X routine, substitute the ANSI symbol appropriate for the type of thermocouple for the X. The following thermocouple conversion routines are provided:

LSP$THERMOCOUPLE_B
LSP$THERMOCOUPLE_E
LSP$THERMOCOUPLE_J
LSP$THERMOCOUPLE_K
LSP$THERMOCOUPLE_R
LSP$THERMOCOUPLE_S
LSP$THERMOCOUPLE_T

See Chapter 5, Thermocouple Conversion, for information about converting thermocouple voltages to temperatures.

## Condition Values

| Symbolic Status | Description |
|---|---|
| LSP$_ILL_N_RANGE | n is out of range |
| LSP$_ILL_V_RANGE | a supplied voltage is out of range |
| LSP$_MAND_ARG | mandatory argument is missing |
| LSP$_SUCCESS | success |

Chapter 7

# Laboratory Signal-Processing Error Handling

This chapter describes the VAXlab Laboratory Signal-Processing (LSP) error-handling facility and explains the error messages and suggested recovery procedures.

## 7.1 Overview

The VAXlab software library provides an LSP error-message facility. When you execute an image that results in an error, the system optionally locates the error message associated with the error and directs it to the devices or files defined as SYS$ERROR and SYS$OUTPUT. The LSP routines use the same standards as the VMS Run-Time Library and System Services for passing back status information about routine calls.

The VMS Run-Time Library and System Services return a status value which is passed back to the user program through a longword variable when the routine is called as a function. A successful operation returns an LSP success status code. An unsuccessful operation returns one of the LSP symbolic values with bit zero clear (false).

In addition, you can use an optional argument in each routine call list to obtain the routine status. Again, a successful operation returns bit 0 set (true).

If you call the routine as either a function or a subroutine, when an error condition exists and you do not include the optional status argument in the routine call, a message is directed to both SYS$OUTPUT and SYS$ERROR.

The symbolic status values are defined in definition files for each of the following program languages:

**Table 7–1: Error-Handling Symbolic Status Definition Files**

| Language | Symbolic Status Definition File |
|---|---|
| VAX Ada | SYS$LIBRARY:LSPDEF.ADA |
| VAX BASIC | SYS$LIBRARY:LSPDEF.BAS |
| VAX C | SYS$LIBRARY:LSPDEF.H |
| VAX FORTRAN | SYS$LIBRARY:LSPDEF.FOR |
| VAX MACRO | SYS$LIBRARY:LSPDEF.MAR |
| VAX PASCAL | SYS$LIBRARY:LSPDEF.PAS |

See Section 7.3, Symbolic Status Values and Descriptions, for a description of the symbolic status values.

## 7.2 Checking Routine Call Status

A user program can check the status of routine calls in the following three ways:

- By testing for status after each operation and, upon receipt of any condition other than success, signaling the condition value to the device or file defined as SYS$ERROR.

```
C Do the inverse FFT on the complex data.  Include the status argument
C in the routine call argument list:

INTERGER*4 STATUS
ISTAT = LSP$FFT_COMPLEX(AN_H_OF_K,CM_H_OF_T,8,1,STATUS)
  IF(.NOT. STATUS) CALL LIB$SIGNAL(%VAL(STATUS))
```

- By testing status after each operation for a specific condition value.

```
      INCLUDE 'SYS$LIBRARY:LSPDEF.FOR  ! Get symbolic status definitions
      INTERGER*4 STATUS
          .
          .
          .

C Do the real FFT and replace the input array with the output array.
C The LSP routine returns the status.

      ISTAT = LSP$FFT_REAL(SIG,SIG,N,0,STATUS)
        IF(STATUS .NE. LSP$_SUCCESS) CALL LIB$SIGNAL(%VAL(STATUS))
```

**NOTE**

If your program is coded to check for specific condition values after one or more operations, you must include the symbolic status definition file appropriate for your programming language. The VAXlab-supplied symbolic status definition files are listed in Table 4–1, Error-Handling Symbolic Status Definition Files. If you do not include the symbolic definition file, your program will not recognize these values.

This program segment tests if status is not equal to success. If status is not equal to success, the routine signals the status. If status equals success, program execution continues.

- By omitting the **status** argument. If you omit the **status** argument and an error occurs, the message is directed to both SYS$ERROR and SYS$OUTPUT.

```
C Calculate the phase spectrum in the CA_PHASE array and the modulus in the
C CA_MODULUS array.

      CALL LSP$PHASE_ANGLE(Y,CA_PHASE,CA_MODULUS,N,)
```

## 7.3 Symbolic Status Values and Descriptions

This section presents the LSP symbolic status values and error messages with an explanation of each value and the appropriate user action suggested to recover from each error condition.

If LSP displays an error message:

- Check the error message **Explanation** and **User Action** information listed in this section.
- Check your program to make sure that all data is being passed correctly.

LSP$_ILL_ARRAY, Input and output arrays cannot be the same

**Explanation:** You made an LSP call specifying the input array and the output array as the same array. The input and output arrays cannot be the same.

**User Action:** Specify different input and output arrays.

LSP$_ILL_CTROL_1_H, Entries 1 and 2 in control array are equal

**Explanation:** You made a call to one of the LSP$HIST_x routines and specified equal values for the first and second entries of the **control** array argument.

The first value specifies the lower limit of the first interval histogram element. The second value specifies the upper limit of the last interval histogram element. If the first value is equal to the second value, an error occurs.

**User Action:** Specify different lower and upper limit values for the first and second entries of the **control** array argument.

LSP$_ILL_CTROL_3_H, 3rd control array entry is less than 1

**Explanation:** You made a call to one of the LSP$HIST_x routines and specified a value less than or equal to zero for the third entry of the **control** array argument.

The third entry specifies the total number of histogram intervals. This number must be greater than zero.

**User Action:** Replace the current entry for the third value of the **control** array argument with a value greater than zero.

LSP$_ILL_CTROL_2_T, 2nd control array entry is out of range

**Explanation:** You made a call to one of the LSP$FORMAT_TRANSLATE_x routines and specified a value that is out of range for the second entry of the **control_i** array argument.

The second entry specifies the number of bits. This number must be between 6 and 16, inclusive. Any number outside this range generates an error.

**User Action:** Replace the current entry for the second value of the **control_i** array argument with an integer between 6 and 16, inclusive.

LSP$_ILL_CTROL_3_T, 3rd control array entry is out of range

**Explanation:** You made a call to one of the LSP$FORMAT_TRANSLATE_x routines and specified a value that is out of range for the third entry of the **control_i** array argument.

The third entry specifies the external gain. This number must be an integer between 1 and 20,000, inclusive. Any number outside of the range generates an error.

**User Action:** Replace the current entry for the third value of the **control_i** array argument with an integer between 1 and 20,000, inclusive.

LSP$_ILL_F_RANGE, Flow or fhigh is out of range

**Explanation:** You made a call to the LSP$FILTER_NONREC routine and specified a value for the **flow** or the **fhigh** argument that is out of range.

The **flow** argument represents the lower frequency of the filter and the **fhigh** argument represents the upper frequency of the filter. The value for either argument must be between 0.0 and 1.0, inclusive.

**User Action:** Replace the current entry for the **flow** argument or the **fhigh** argument with a value between 0.0 and 1.0, inclusive.

LSP$_ILL_FILTYP, filtyp is out of range

**Explanation:** You made a call to one of the LSP$FILTER_POLY_x routines and specified a value for the **filtyp** argument that is outside of the range of supported values for this LSP function.

**User Action:** Replace the current entry for the **filtyp** argument with a legal filtering type value. See the appropriate LSP$FILTER_POLY_x routine call reference description for a list of available filter types.

LSP$_ILL_FLOW, flow is equal to fhigh

**Explanation:** You made a call to the LSP$FILTER_NONREC routine and specified the **flow** argument equal to the **fhigh** argument.

**User Action:** Provide different values for the **flow** argument and the **fhigh** argument so that **flow** is not equal to **fhigh**.

LSP$_ILL_N_FILTER, Too few data points per filter window

**Explanation:** You made a call to one of the LSP$FILTER_POLY_X routines and specifed an incorrect value for the **n** argument.

**User Action:** Specify a number of data points for the **n** argument that is greater than the number of data points specified for the filter window you are using.

LSP$_ILL_N_NONREC, n is less than 2*nterms+1

**Explanation:** You made a call to the LSP$FILTER_NONREC routine and specified the **n** argument less than or equal to the value of **nterms***2+1.

The **n** argument contains the number of data values to be filtered. This value must be greater than the value of **nterms***2+1. Any number less than or equal to this value generates an error.

**User Action:** Replace the current value of **n** with a value greater than the value of **nterms***2+1, or make the value of **nterms** smaller.

LSP$_ILL_N_NOT_2, n is not a power of 2

**Explanation:** You made a call to one of the fast Fourier transform routines or the LSP$CORRELATION routine and specified an incorrect value for the the **n** argument, the **n1** argument, or the **n2** argument.

If you are using LSP$CORRELATION routine, LSP$FFT_COMPLEX routine, or LSP$FFT_REAL routine, you specified an incorrect value for the **n** argument.

If you are using LSP$FFT_COMPLEX_2D routine, you specified an incorrect value for the **n1** argument or the **n2** argument.

For LSP$CORRELATION, LSP$FFT_COMPLEX, or LSP$FFT_REAL, the value of the **n** argument must be a power of 2 and range between 2 and 32,768. In this case $n = 2^m$, where m is between 1 and 15, inclusive. The power of 2 restriction also applies to the LSP$FFT_COMPLEX_2D routine.

**User Action:** Replace the current value of **n**, **n1**, or **n2** with a power of 2 value between 2 and 32,768, inclusive. For more information on discrete Fourier transform and correlation functions, see Chapter 2, Performing Fourier Transforms and Correlation Functions. You can also refer to the routine call reference sections for each of the four routines.

LSP$_ILL_N_RANGE, n is out of range

**Explanation:** You entered an incorrect value for the **n** argument. The value of **n** can not be less than or equal to zero.

**User Action:** Check the routine call reference description for the allowed range of values for the **n** argument.

LSP$_ILL_NTERMS, nterms is out of range

**Explanation:** You made a call to the LSP$FILTER_NONREC routine and specified an incorrect value for the **nterms** argument. The value of **nterms** must be between 2 and 500, inclusive. Any number outside this range generates an error.

**User Action:** Replace the current value of the **nterms** argument with a variable between 2 and 500, inclusive.

LSP$_ILL_RANGE, Both entries in the range array are equal

**Explanation:** You made a call to the LSP$FORMAT_TRANSLATE_ADC routine or the LSP$FORMAT_TRANSLATE_DAC routine and specified the same number for both the first and second values of the **range** argument.

The first value specifies the lowest voltage; the second value specifies the highest voltage. The default range is -10 to 9.9951 volts. If the two values are equal, an error occurs.

**User Action:** Adjust the first and second values of the **range** argument accordingly. See the appropriate routine call reference description for more information about the **range** argument.

LSP$_ILL_V_RANGE, A supplied voltage is out of range

**Explanation:** You made a call to one of the LSP$THERMOCOUPLE_X routines and specified an incorrect value for the **volts** argument.

**User Action:** Replace the current value of the **volts** argument with a value from Table 5–2, Thermocouple Temperature and Voltage Ranges.

LSP$_ILL_WIGGLES, wiggles is out of range

**Explanation:** You made a call to the LSP$FILTER_NONREC routine and specified an incorrect value for the **wiggles** argument.

The **wiggles** argument is a number in -dB units which is proportional to the oscillation from the Gibbs Phenomenon. This number must be between 0.0 and 100.0, inclusive.

**User Action:** Replace the current value of the **wiggles** argument with a number between 0.0 and 100.0, inclusive. See Section 3.1.2, Nonrecursive Filtering, for more information about the **wiggles** argument.

LSP$_ILL_WINDOW_TYPE, invalid window type specified

**Explanation:** You made a call to the LSP$SPECTRAL_WINDOWS routine or the LSP$BUILD_WINDOW_TABLE routine and specified an incorrect value for the **window_type** argument.

**User Action:** Replace the current value of the **window_type** argument with one of the five window type values listed in the routine call reference description for that routine.

LSP$_MAND_ARG, Mandatory argument is missing

**Explanation:** You omitted a mandatory argument in an LSP routine call.

**User Action:** Review your program for missing or defaulted arguments.

LSP$_SUCCESS, Success

**Explanation:** Your routine executed successfully.

**User Action:** No action required.

Chapter 8

# Overview of Online Sample Programs

This chapter provides an overview of sample programs showing how to use the Laboratory Signal-Processing routines. These programs are shipped with your VAXlab software kit and are placed on disk during the VAXlab software installation procedure. You can find the LSP sample programs in a directory with the logical name LSP$EXAMPLES. The logical name of this directory is defined in LSPSTARTUP.COM during installation.

The LSP sample program file names include:

- the facility code, LSP
- a descriptive abbreviation for the LSP routine or task the sample program illustrates
- a file extension indicating the programming language in which each sample program is coded

For example, the sample program LSP_FFT_RAND_DAT.FOR uses the LSP$FFT_REAL routine, shows how to perform a fast Fourier transform of random data stored in an array, and is written in VAX FORTRAN.

Table 8-1, LSP Online Sample Programs, lists the sample program names, the routines each sample program uses, and a brief description of what each sample program does. Review Table 8-1 to determine which of the sample programs will be helpful to you in learning how to use the LSP routines.

In order to print, edit, or read a sample program you must copy the program to your own directory. To copy a sample program, in this case LSP_FFT_RAND_DAT.FOR, to your directory, enter the following command line:

```
$COPY LSP$EXAMPLES:LSP_FFT_RAND_DAT.FOR *.* RETURN
```

**Table 8–1: LSP Online Sample Programs**

| Program Name | Routines |
|---|---|
| LSP_AUTOCOR_RAND_SEQ.FOR | LSP$CORRELATION<br>LSP$FFT_REAL<br>LGP$TABLE_MODIFY<br>LGP$PLOT<br>LGP$TERMINATE_PLOT |
| **Description:** Sample program LSP_AUTOCOR_RAND_SEQ.FOR demonstrates the use of the LSP$CORRELATION routine by autocorrelating a random sequence. | |
| LSP_CALC_COMP_PHASE_SPEC.FOR | LSP$PHASE_ANGLE |
| **Description:** Sample program LSP_CALC_COMP_PHASE_SPEC.FOR demonstrates the use of the LSP$PHASE_ANGLE routine by generating a phase spectrum where the phase is known analytically. The program-generated phase spectrum is then compared to the analytical function. | |
| LSP_CALC_PHASE_SPEC.FOR | LSP$PHASE_ANGLE |
| **Description:** Sample program LSP_CALC_PHASE_SPEC.FOR demonstrates the use of the LSP$PHASE_ANGLE routine by computing the phase spectrum of a function whose phase and amplitude are known analytically. | |
| LSP_CROSSCOR_SINE_COSINE.FOR | LSP$CORRELATION<br>LSP$FFT_REAL<br>LGP$TABLE_MODIFY<br>LGP$PLOT<br>LGP$TERMINATE_PLOT |
| **Description:** Sample program LSP_CROSSCOR_SINE_COSINE.FOR demonstrates the use of the LSP$CORRELATION routine by cross-correlating a sine and cosine waveform. | |
| LSP_FFT_COMP_FORW.FOR | LSP$FFT_COMPLEX |
| **Description:** Sample program LSP_FFT_COMP_FORW.FOR demonstrates the use of the LSP$FFT_COMPLEX routine by calculating the forward Fourier transform of a complex function. | |

**Table 8–1 (Cont.): LSP Online Sample Programs**

| Program Name | Routines |
|---|---|
| LSP_FFT_COMP_INVER.FOR | LSP$COMPLEX |

**Description:** Sample program LSP_FFT_COMP_INVER.FOR demonstrates the use of the LSP$FFT_COMPLEX routine by calculating the inverse Fourier transform of a complex function.

| | |
|---|---|
| LSP_FFT_FUNC.FOR | LSP$FFT_REAL |

**Description:** Sample program LSP_FFT_FUNC.FOR demonstrates the use of the LSP$FFT_REAL routine by calculating the fast Fourier transform of the function: h(t) = Q**t where Q = 0.9 and t = 0, 1, 2, ... N-1.

| | |
|---|---|
| LSP_FFT_RAND_DAT.FOR | LSP$FFT_REAL |

**Description:** Sample program LSP_FFT_RAND_DAT.FOR demonstrates the use of the LSP$FFT_REAL routine by performing a fast Fourier transformation on eight points stored in an array.

| | |
|---|---|
| LSP_FILT_NONREC.FOR | LSP$FILTER_NONREC<br>LGP$PLOT<br>LGP$TERMINATE_PLOT |

**Description:** Sample program LSP_FILT_NONREC.FOR demonstrates the use of the LSP$FILTER_NONREC routine in lowpass and highpass mode by creating a sine wave of period $2\pi$ and modulating this waveform with a smaller amplitude sine wave which oscillates 40 times faster. The program filters in lowpass mode to get the primary sine wave and outputs it to an array. The program filters in highpass mode to get the higher frequency signal and outputs it to another array. See Chapter 3, Digital Filtering, for a copy of this sample program and the graphical output it produces. The sample program is coded in VAX FORTRAN.

| | |
|---|---|
| LSP_FILT_NONREC.PAS | LSP$FILTER_NONREC<br>LGP$PLOT<br>LGP$TERMINATE_PLOT |

**Description:** Sample program LSP_FILT_NONREC.PAS demonstrates the use of the LSP$FILTER_NONREC routine in lowpass and highpass mode by creating a sine wave of period $2\pi$ and modulating this waveform with a smaller amplitude sine wave which oscillates 40 times faster. The program filters in lowpass mode to get the primary sine wave and outputs it to an array. The program filters in highpass mode to get the higher frequency signal and outputs it to another array. The sample program is coded in VAX Pascal.

**Table 8–1 (Cont.): LSP Online Sample Programs**

| Program Name | Routines |
|---|---|
| LSP_FILT_POLY.FOR | LSP$FILTER_POLY |

**Description:** Sample program LSP_FILT_POLY.FOR demonstrates the use of the LSP$FILTER_POLY routine by using this routine to smooth program-generated data.

| | |
|---|---|
| LSP_FILT_POLY_1ST_DERIV.FOR | LSP$FILTER_POLY_1ST_DERIV |

**Description:** Sample program LSP_FILT_POLY_1ST_DERIV.FOR demonstrates the use of the LSP$FILTER_POLY_1ST_DERIV by creating a table of a function and its analytical first derivation. The program passes the function values through the first derivative filter and subsequently compares the results of the analytical first derivative to the program-generated first derivative.

| | |
|---|---|
| LSP_FORM_TRANS_ADC.FOR | LSP$FORMAT_TRANSLATE_ADC |

**Description:** Sample program LSP_FORM_TRANS_ADC.FOR demonstrates the use of the LSP$FORMAT_TRANSLATE_ADC routine by converting two's complement data in an input array to voltage data which is written to an output array.

| | |
|---|---|
| LSP_FORM_TRANS_DAC.FOR | LSP$FORMAT_TRANSLATE_DAC |

**Description:** Sample program LSP_FORM_TRANS_DAC.FOR demonstrates the use of the LSP$FORMAT_TRANSLATE_DAC routine by converting data (in volts) in an input array to offset binary format data which is written to an output array.

| | |
|---|---|
| LSP_HIST_F.FOR | LSP$HIST_F<br>LGP$PLOT<br>LGP$HIST<br>LGP$TERMINATE_PLOT |

**Description:** Sample program LSP_HIST_F.FOR demonstrates the use of LSP$HIST_F by performing interval histogram analysis with program-generated floating-point data.

| | |
|---|---|
| LSP_HIST_I.FOR | LSP$HIST_I<br>LGP$PLOT<br>LGP$HIST<br>LGP$TERMINATE_PLOT |

**Description:** Sample program LSP_HIST_I.FOR demonstrates the use of the LSP$HIST_I routine by performing interval histogram analysis with a given set of integer data.

**Table 8–1 (Cont.): LSP Online Sample Programs**

| Program Name | Routines |
|---|---|
| LSP_PHASE_ANG.FOR | LSP$FFT_COMPLEX_2D<br>LSP$PHASE_ANGLE_2D<br>LGP$3D_SIMPLE<br>LGP$TERMINATE_PLOT |
| **Description:** Sample program LSP_PHASE_ANG.FOR demonstrates the use of the LSP$FFT_COMPLEX_2D routine and the LSP$PHASE_ANGLE_2D routine by calculating the fast Fourier transform of a complex two-dimensional function and then computing the phase angle and magnitude. | |
| LSP_POW_SPEC_SINE.C | LSP$FFT_REAL<br>LSP$POWER_SPECTRUM |
| **Description:** Sample program LSP_POW_SPEC_SINE.C demonstrates the use of the LSP$POWER_SPECTRUM routine by computing the power spectrum of a sine wave of one period. The sample program is coded in VAX C. | |
| LSP_POW_SPEC_SINE.FOR | LSP$FFT_REAL<br>LSP$POWER_SPECTRUM |
| **Description:** Sample program LSP_POW_SPEC_SINE.FOR demonstrates the use of the LSP$POWER_SPECTRUM routine by computing the power spectrum of a sine wave of one period. The sample program is coded in VAX FORTRAN. | |
| LSP_THERMOC_B.FOR | LSP$THERMOCOUPLE_B |
| **Description:** Sample program LSP_THERMOC_B.FOR demonstrates the use of the LSP$THERMOCOUPLE_B routine by converting voltage values to temperatures. | |

**Table 8–1 (Cont.): LSP Online Sample Programs**

| Program Name | Routines |
|---|---|
| LSP_WINDOW_1.FOR | LIO$ATTACH<br>LIO$DETACH<br>LIO$READ<br>LIO$SET<br>LIO$WRITE<br>LSP$APPLY_WINDOW_TABLE<br>LSP$BUILD_WINDOW_TABLE<br>LSP$FFT_REAL<br>LSP$FORMAT_TRANSLATE_ADC<br>LGP$PLOT<br>LGP$TERMINATE_PLOT |
| **Description:** Sample program LSP_WINDOW_1.FOR demonstrates the use of the LSP$BUILD_WINDOW_TABLE routine, the LSP$APPLY_WINDOW_TABLE routine, and the periodogram technique of calculating the power spectrum. See Chapter 4, Spectral Window Filtering, for a copy of this sample program and the output it produces. | |
| LSP_WINDOW_2.FOR | LIO$ATTACH<br>LIO$DETACH<br>LIO$READ<br>LIO$SET<br>LIO$WRITE<br>LSP$SPECTRAL_WINDOWS<br>LSP$FFT_REAL<br>LSP$FORMAT_TRANSLATE_ADC<br>LSP$POWER_SPECTRUM<br>LGP$PLOT<br>LGP$TERMINATE_PLOT |
| **Description:** Sample program LSP_WINDOW_2.FOR illustrates the use of the LSP$SPECTRAL_WINDOWS routine by collecting data with LIO; translating the data format from analog to digital; performing an FFT on the data; calculating the power spectrum of the data; running the data through a low-pass filter and plotting the results; and running the data through a high-pass filter and plotting the results. See Chapter 4, Spectral Window Filtering, for a copy of this sample program and the output it produces | |

Appendix A

# Mathematics and Statistics Routines

This chapter provides an overview and summary of the routines you use to perform mathematical and statistical analysis of real-time and static data. This chapter also provides information about how to print a hard copy of the document describing how to use these routines.

**NOTE**

Please note, the Scientific Subroutines Package (SSP) is included in the VAXlab Software Library at no cost to the purchaser. DIGITAL does not provide any support for SSP.

## A.1 Overview of Mathematics and Statistics Routines

To perform mathematical and statistical analysis of real-time and static data, you use the routines provided in the Scientific Subroutines Package (SSP). The *Scientific Subroutines Programmer's Reference Manual* describes how to use the SSP routines. This document is shipped in machine-readable form (there is no hardcopy manual) and is put on-line during the VAXlab software installation procedure. You need to print a hard copy of this document to use the SSP routines. The *Scientific Subroutines Programmer's Reference Manual* is located in SYS$SYSROOT:[UNSUPPORTED.SSP]SSP_GUIDE.MEM.

The following sections summarize the mathematics and statistics routines available to you.

## A.2 Mathematics Routine Call Summary

The following table summarizes the mathematics routines.

**Table A–1: VAXlab Mathematics Routine Call Summary**

| Routine | Function |
|---|---|
| ARRAY | Converts a data array from single to double dimension, or from double to single dimension. |
| BESI | Computes the I Bessel function for a given argument and order using series or asymptotic approximation. |
| BESJ | Computes the J Bessel function for a given argument and order using recurrence-relation technique. |
| BESK | Computes the K Bessel function for a given argument and order using series approximation and recurrence relations. |
| BESY | Computes the Y Bessel function for a given argument and order using recurrence relations and polynomial approximations. |
| CADD | Adds a column of one matrix to the column of another matrix. |
| CCPY | Copies a column of a matrix into a vector. |
| CCUT | Partitions a matrix between specified columns to form two resultant matrices. |
| CEL1 | Computes the complete elliptic integral of the first kind using Landens transformation. |
| CEL2 | Computes the generalized complete elliptic integral of the second kind. |
| CINT | Interchanges two columns of a matrix. |
| CS | Computes the Fresnel integrals using rational function approximations. |
| CSRT | Sorts columns of a matrix. |
| CSUM | Sums the elements of each column of a matrix to form a row vector. |
| CTAB | Adds the columns of one matrix into a new matrix in the columns specified by a floating-point number in the respective row of the input vector. |
| CTIE | Adjoins two matrices with the same row dimensions to form one resultant matrix. |

**Table A–1 (Cont.): VAXlab Mathematics Routine Call Summary**

| Routine | Function |
|---|---|
| DCLA | Sets each diagonal element of a matrix equal to a scalar. |
| DCPY | Copies the diagonal elements of a matrix into a vector. |
| EIGEN | Computes the eigenvalues and eigenvectors of a real symmetric matrix. |
| EXPI | Computes the exponential integral -EI(-X) using three different rational approximations. |
| FORIF | Computes the coefficient of the desired number of terms in the Fourier Series F(X)=A(0)+SUM(A(K)COS KX+B(K)SIN KX), where K = 1,2, ... M, to approximate the computed values of a given function subprogram. |
| FORIT | Computes the coefficients of a specified number of terms in the Fourier Series to approximate a given set of periodically tabulated values of a function. |
| GAMMA | Computes the GAMMA function for a given argument using the recursion relation and polynomial approximation. |
| GMADD | Adds two general matrices to form a resultant matrix. |
| GMPRD | Multiplies two general matrices to form a resultant general matrix. |
| GMSUB | Subtracts one general matrix from another general matrix to form a resultant general matrix. |
| GMTRA | Transposes a general matrix. |
| GTPRD | Premultiplies a general matrix by the transpose of another general matrix. |
| LEP | Computes the values of the Legendre polynomials P(N,X) for argument value X and orders 0 to N. |
| LOC | Computes a vector subscript for an element in a matrix of specified storage mode. |
| MADD | Adds two matrices to form a resultant matrix. |
| MATA | Premultiplies a matrix by its transpose to form a symmetric matrix. |
| MCPY | Copies an entire matrix. |
| MFUN | Applies a function to each element of a matrix to form a resultant matrix. |
| MINV | Inverts a matrix and calculates its determinant. |

**Table A–1 (Cont.): VAXlab Mathematics Routine Call Summary**

| Routine | Function |
|---|---|
| MPRD | Multiples two matrices to form a resultant matrix. |
| MSTR | Changes the storage mode of a matrix. |
| MSUB | Subtracts one matrix from another matrix, element by element, to form a resultant matrix. |
| MTRA | Transposes a matrix. |
| PADD | Adds two polynomials. |
| PADDM | Multiples a polynomial by a constant and adds the result to another polynomial. |
| PCLD | Performs complete linear synthetic division (shift of origin). |
| PDIV | Divides one polynomial by another. |
| PGCD | Determines the greatest common divisor of two polynomials. |
| PILD | Evaluates a polynomial and its first derivative for a given argument. |
| PINT | Determines the integral of a polynomial with a constant of integration equal to zero. |
| PMPY | Multiples two polynomials. |
| PNORM | Normalizes coefficient vector of a polynomial. |
| PQSD | Performs quadratic synthetic division of a polynomial. |
| PSUB | Subtracts one polynomial from another polynomial. |
| PVAL | Evaluates a polynomial for a given value of the variable. |
| PVSUB | Substitutes a polynomial for the variable of another polynomial. |
| POLRT | Determines the real and complex roots of a real polynomial using the Newton-Raphson iterative technique. |
| QATR | Uses Romberg's extrapolation method to approximate the integral of a given function by trapezoidal rule. |
| QSF | Computes the vector of integral values for a given equidistant table of function values. |
| RADD | Adds a row of one matric to the row of another matrix. |
| RCPY | Copies a row of a matrix into a vector. |
| RCUT | Partitions a matrix between specified rows to form two resultant matrices. |

**Table A–1 (Cont.): VAXlab Mathematics Routine Call Summary**

| Routine | Function |
|---|---|
| RECP | Calculates the reciprocal of an element. |
| RINT | Interchanges two rows of a matrix. |
| RK1 | Integrates a first-order differential equation up to a specified final value. |
| RK2 | Integrates a first-order differential equation by Runge-Kutta and produces a table of the integrated values. |
| RKGS | Solves a system of first-order ordinary differential equations with initial values by the Runge-Kutta method. |
| RSRT | Sorts the rows of a matrix. |
| RSUM | Sums the elements of each row of a matrix to form a column vector. |
| RTAB | Adds the rows of one matrix into a new matrix in the rows specified by a floating-point number in the respective row of the input vector. |
| RTIE | Adjoins two matrices with the same column dimension to form one resultant matrix. |
| RTMI | Solves the general nonlinear equation of the form FCT(X)=0 using Mueller's iteration method. |
| RTNI | Solves the general nonlinear equation of the form FCN(X)=0 using Newton's iteration method. |
| RTWI | Solves the general nonlinear equation of the form FCT(X)=0 using Wegstein's iteration method. |
| SADD | Adds a scalar to each element of a matrix to form a resultant matrix. |
| SCLA | Sets each element of a matrix equal to a given scalar. |
| SCMA | Multiplies a column of a matrix by a scalar and adds the product to another column of the same matrix. |
| SDIV | Divides each element of a matrix by a scalar to form a resultant matrix. |
| SICI | Commutes the sine and cosine integrals, where SI(x)=integral(Sin(X)/X)-PI/2, and CI(x)=integral(Cos(X)/X). |
| SIMQ | Solves a set of simultaneous linear equations, AX=B. |

**Table A–1 (Cont.): VAXlab Mathematics Routine Call Summary**

| Routine | Function |
|---|---|
| SMPY | Multiplies each element of a matrix by a scalar to form a resultant matrix. |
| SRMA | Multiplies a row of a matrix by a scalar and adds the product to another row of the same matrix. |
| SSUB | Subtracts a scalar from each element of a matrix to form a resultant matrix. |
| TPRD | Transposes and then postmultiplies a matrix by another matrix to form a resultant matrix. |
| XCPY | Copies a specified submatrix from a matrix. |

## A.3 Statistics Routine Call Summary

The following table summarizes the statistics routines.

**Table A–2: VAXlab Statistics Routine Call Summary**

| Routine | Function |
|---|---|
| ABSNT | Tests for missing (or zero) values for each observation in a general matrix. |
| AUTO | Determines the autocovariances of a series A for lags 0 to L-1. |
| AVCAL | Performs the calculus of a factorial experiment using operator sigma and operator delta. The AVCAL routine is preceded by the AVDAT routine and succeeded by the MEANQ routine in analyzing variance for a complete factorial design. |
| AVDAT | Places data for variance analysis in properly distributed positions of storage. The AVDAT routine precedes the AVCAL and MEANQ routines when analyzing variance for a complete factorial design. |
| BOUND | Selects from a set (or subset) of observations, the number of observations under, between, and over two given bounds for each variable. |
| CANOR | Calculates the canonical correlations between two sets of variables. CANOR is normally preceded by a call to the CORRE routine. |
| CHISQ | Computes the chi-square from a contingency table. |
| CORRE | Computes means, standard deviations, sums of cross-products of deviations, and correlation coefficients. This routine is normally used in a sequence of calls to the routines CORRE, EIGEN, TRACE, LOAD, and VARMX when performing a factor analysis. |
| CROSS | Determines the cross-covariances of a series A with a series B that leads and lags A. |
| DISCR | Computes a set of linear functions that are indices for classifying an individual into one of several groups. This routine is normally used when performing discriminant analysis. |

**Table A–2 (Cont.): VAXlab Statistics Routine Call Summary**

| Routine | Function |
|---|---|
| DMATX | Computes means of variables in each group and a pooled dispersion matrix for all the groups. This routine is normally used to perform discriminant analysis. |
| EXSMO | Determines the triple exponential smoothed series S of the given series X. |
| GAUSS | Computes a normally distributed random number with a given mean and standard deviation. |
| GDATA | Generates independent variables up to the Mth power (the highest degree polynomial specified) and computes means, standard deviations, and correlation coefficients. This routine normally precedes the ORDER, MINV, and MULTR routines when performing polynomial regression. |
| KRANK | Tests the correlation between two variables by the Kendall rank correlation coefficient. |
| LOAD | Computes a factor matrix (loading) from eigenvalues and associated eigenvectors. This routine is normally used in a sequence of calls to the routines CORRE, EIGEN, TRACE, LOAD, and VARMX when performing a factor analysis. |
| MEANQ | Computes sum of squares, degrees of freedom, and mean square using the mean square operator. This routine normally succeeds the AVDAT and AVCAL routines when analyzing variance for a complete factorial design. |
| MOMEN | Determines the first four moments for grouped data on equal class intervals. |
| MULTR | Performs a multiple linear regression analysis for a dependent variable and a set of independent variables. This routine is normally used to perform multiple and polynomial regression analyses. |
| NROOT | Computes the eigenvalues and eigenvectors of a real nonsymmetric matrix of the form *B-inverse times A*. This routine is normally called by the CANOR routine when performing a canonical correlation analysis. |
| ORDER | Constructs a subset matrix of intercorrelations among independent variables and a vector of intercorrelations of independent variables with a dependent variable from a larger matrix of correlation coefficients. This routine is normally used to perform multiple and polynomial regression analyses. |

**Table A–2 (Cont.): VAXlab Statistics Routine Call Summary**

| Routine | Function |
|---|---|
| QTEST | Tests whether three or more matched groups of dichotomous data differ significantly by the Cochran Q-test. |
| RANK | Ranks a vector of values. The RANK routine assigns tied values to the average rank. |
| SMO | Smooths or filters series A by weights W. |
| SRANK | Tests the correlation between two variables by the Spearman rank correlation coefficient. |
| SUBMX | Builds a subset matrix. Based on vector **S** derived from routine SUBST or ABSNT, SUBMX copies from a larger matrix of observation data a subset matrix of those observations which have satisfied a certain condition. |
| SUBST | Derives a subset vector indicating which observations in a set have satisfied certain conditions on the variables. |
| TAB1 | Tabulates for one variable in an observation matrix (or a matrix subset) the frequency and percent over given class intervals. In addition, the TAB1 routine calculates for the same variable the total, mean, standard deviation, minimum, and maximum. |
| TAB2 | Performs a two-way classification for two variables in an observation matrix (or matrix subset) of the frequency, percent frequency, and other statistics over given class intervals. |
| TALLY | Calculates the total, mean, standard deviation, minimum, and maximum for each variable in a set (or subset) of observations. |
| TIE | Calculates the correction factor resulting from rank ties. |
| TRACE | Computes the cumulative percentage of eigenvalues greater than or equal to a specified constant. This routine is normally used in a sequence of calls to the routines CORRE, EIGEN, TRACE, LOAD, and VARMX when performing a factor analysis. |
| TTSTT | Determines certain T-statistics on the means of populations. |
| TWOAV | Tests whether samples are from the same population by using the Friedman two-way analysis of variance test. |

**Table A–2 (Cont.): VAXlab Statistics Routine Call Summary**

| Routine | Function |
|---|---|
| UTEST | Determines whether two independent groups are from the same population by using the Mann-Whitney U-test. |
| VARMX | Performs orthogonal rotations of a factor matrix. This routine is normally used in a sequence of calls to the routines CORRE, EIGEN, TRACE, LOAD, and VARMX when performing a factor analysis. |
| WTEST | Tests the degree of association among a number of variables by using the Kendall coefficient of concordance. |

Appendix B

# The Peak-Processing Routine

The peak-processing routine detects significant fluctuations, called peaks, in data describing a waveform, and reports definitive characteristics for each peak found. The process is known as peak analysis.

This appendix explains how to build the peak-processing (PEAK) routine, provides an overview of its functionality and use, and details how to create a FORTRAN program that calls the PEAK routine under the VMS operating system.

**NOTE**

The Peak-Processing Package (PEAK) is included in the VAXlab Software Library at no cost to the purchaser. DIGITAL does not provide support for the PEAK routine.

## B.1 Building the PEAK Routine

The PEAK routine is installed on your system as part of the VAXlab installation procedure. However, before you can use the PEAK routine, you need to build it. During the build procedure, you can enable an option as characteristics of the routine, or you can build the routine without the option. See Section B.1.1, Enabling Routine Options, of this appendix for information about the option available to you.

### B.1.1 Enabling the No Filter Option

This section explains the option you can use with the peak-processing routine. If you want to use this option, you must enable it when you build the routine from the source file using the interactive build procedure.

The NOFLT$ (No Filter) option disables the software digital filter that the routine normally uses. Enable this option if you want to average and process data points without filtering them, or if you want to apply your own filter to the raw data before calling the PEAK routine.

Enabling the no filter option results in quicker processing of data points and decreases the size of the routine.

Before you begin the build procedure, the following system requirements must be met:

- The following files must be resident on the system device, in the SYS$PEAK directory.

| File Name | Description |
|---|---|
| FPEAK.MAR | MACRO source file for the peak-processing routine |
| EX1FPE.FOR | FORTRAN source file for the test routine |
| PEAKMAK.COM | Interactive build procedure |
| WRTBLD.COM | Part of the build procedure |
| WRTVER.COM | Part of the build procedure |

- The VAX FORTRAN compiler is built and resides on the system device.

PEAKMAK.COM, the interactive build procedure for the PEAK routine, lets you assemble the routine and optionally, place it in a library which you create by supplying a library name. When you run PEAKMAK.COM, it prompts you with questions. You enter directory specifications for input and output after the appropriate prompt. The input directory is the directory in which the above routines were installed. The output directory should be your own directory.

You are then asked if you want to build a library. Enter Y (yes) to build an object library. Enter N (no) to create simply an object file. Then, answer "Yes" after the subsequent questions if you want to enable the specified option. Answer "No" if you do not. After you answer all the questions, PEAKMAK.COM creates three files in the output directory you specified as follows:

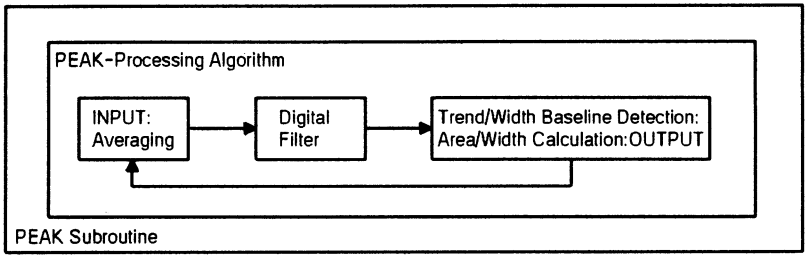
| File Name | Description |
|---|---|
| PEAKCND.MAR | This file sets the switches to enable the options you requested. |
| PEAKBLD.COM | This indirect-command procedure builds the PEAK routine. Building consists of assembling the routine with the switches set to enable the options you chose. If you specified a library while running PEAKMAK.COM, PEAKBLD.COM creates that library and includes the PEAK routine in it. |
| PEAKVER.COM | This indirect-command procedure verifies that the PEAK routine is in good working order. It does this by running an example program that tests the routine. |

## B.2 Overview of the PEAK Routine

The peak-processing routine detects significant fluctuations, called peaks, in data describing a waveform, and reports definitive characteristics for each peak found. The process is know as peak analysis.

Input to the routine is a series of discrete positive integers corresponding to values of a waveform function at evenly spaced intervals. To eliminate distortion-producing components in the data, the input is linearly averaged and filtered before final processing. Figure B-1 shows the peak-processing algorithm. You can change specified algorithm parameters to enhance detectability of directional trends and baselines for a given set of data.

**Figure B–1: Flow of the PEAK Routine**




Output from the routine is in the form of size and position for each peak detected. Size is defined by area, height, and width; and position is expressed in terms of when a peak begins, crests, and ends. The routine further reports how each peak ends — on a baseline or at a valley.

## B.3 Definition of Basic Terms and Conventions

It is important to understand how some of the terms and conventions describing the PEAK routine are used throughout this appendix.

- The term data (input) stream describes all values presented to the routine for processing. Actual values processed by the algorithm are sometimes called heights, for example, crest height, leading minimum height.
- The duration axis of the waveform is the time axis. Time is measured as the number of raw data points processed since the start of the input stream. Thus, the term crest time means that crest height was observed when a number of raw data points equal to crest time were processed.
- "Noise" is a generic term for all distortion-producing components in the input data.

- Point-to-point changes are local changes, as contrasted with overall changes during the course of the waveform, which are called trends.
- Changes are persistent in one direction if the number of changes in the direction exceeds the number in the opposite direction.

## B.4 The Peak-Processing Algorithm: Processing Raw Data

The peak-processing algorithm detects increasing and decreasing trends in a set of data. Output from the PEAK routine is directly related to the points where we observe changes in these trends. When we see an increasing trend, the point where the increase begins is labelled the start of the peak, and its value the leading minimum height. The point where a subsequent decreasing trend begins is the crest, or crest height, of the peak. And the point where the decreasing trend stops — or a baseline is detected — is taken as the end of the peak, or its trailing minimum height. We can then use this information to calculate the area and width of the peak. Under ideal conditions, this sequence defines the total function of the algorithm.

Actual conditions are seldom ideal, however. Environmental influences during data collection tend to distort the pure function being analyzed. To a great degree, the algorithm and any controls that you can exercise over the routine parameters are aimed at removing these distortions so that only the real (dominant) trends in the data are visible.

### B.4.1 Averaging of Input Data

The peak-processing algorithm first takes a linear average of input data points; you can specify the number of points to be averaged by means of the first variable parameter, the Original Point Density (OPD). Thereafter, the routine deals only with averaged heights, which can represent several raw data points. Keep in mind, however, that the time associated with each averaged height is based upon the total number of raw data points averaged since input began.

This averaging process smooths any "rough edges" from the data. You should give serious thought to the value you assign to the OPD. If too many points are averaged, real information may be lost. In an extreme case, you might miss an entire true peak, but a more common result is late detection of significant trend changes. By averaging too few points, on the other hand, you could detect false trend changes.

In certain applications of the routine, you may find that peaks are "tall and thin" at the outset of the waveform, then tend to become "short and fat" as it progresses. The algorithm compensates for this tendency by increasing automatically the number of points averaged when it detects a peak width that exceeds a preset optimum.[1] Thus, the algorithm makes wide, short peaks more visible and increases the likelihood of detecting real data fluctuations that might otherwise appear insignificant.

## B.4.2 Use of the Digital Filter

The averaged data points are not processed directly by the trend-detecting portion of the algorithm, but are first filtered by means of a digital filter. The equation for this nonlinear center-weighted filter involves seven averaged-data points having coefficients of a modified least-squares fit.

$$Y_0 = (-(Y_{-3} + Y_{+3}) + 4(Y_{-2} + Y_{+2}) + 11(Y_{-1} + Y_{+1}) + 14Y_0)/42$$

The coefficients are tuned to prevent area distortion for small peaks in the vicinity of large ones.

As each new averaged data point is calculated, it is placed in the filter as the last, or $Y_{+3}$, point. The new center point is calculated; after which the points used in the filter are shifted down by one, that is, $Y_{-1} = Y_0$. Prerequisite sites for this process are:

- Seven averaged points must be calculated before the digital filter may be applied.
- The first point to be considered for directional-trend detection is the center point resulting from application of the digital filter to these original seven points.

[1] When the half-width-at-half-height measurement of a peak exceeds 25, the algorithm doubles the number of points averaged.

- Each subsequent set of seven points used by the filter is chosen using a sliding "window," that is, each new averaged point (after the first six) is used seven times in successive applications of the filter. There is a slight twist to the sliding window in that once the filter has been applied three times, four of the points in the current application of the filter are the result of averaging raw data, while the other three ($Y_{-3}$, $Y_{-2}$, $Y_{-1}$) are the result of previous applications of the filter.

## B.4.3 Trend Detection — Application of the Gate Factor

Although averaged and filtered data have been smoothed in earlier processing, the resultant filtered data points may still exhibit slight point-to-point fluctuations unrelated to the dominant trend of the data. You may set two parameters — the gate factor and the minimum increase — so that the algorithm eliminates much of the effect of this fluctuation.

The gate factor (GT) specifies a valid directional trend in terms of the number of changes in direction, either persistent or consecutive, over a series of filtered points.

The minimum increase (IM) is a standard used to test for a real increase in filtered data from point to point.

At the outset of the input stream and at points where crests are detected, neither an increasing nor a decreasing directional trend has yet been established. The next established trend is determined at these points as the first direction in which the data changes "gate" times.

At intermediate points a current trend is already established. Changes in directional trend at these points may be established only if the number of consecutive local changes in the new direction is equal to the gating factor.

A local change is defined in terms of the relation between a given data point and the local minimum or maximum. If the current height is less than the local minimum, the change is downward. Conversely, if the height is greater than the sum of the total maximum and the minimum increase value (IM), the change is upward. If the height is between the local minimum and maximum, no change is indicated (although the area is updated).

When processing is initialized, and at the crest of each peak, the local minimum is set to a very high value, and the maximum is set to a very low value. Between crests, the local minimum and maximum can be best described by the flow diagram.

It should be stressed that the points of greatest interest on the waveform -- essentially the points that determine the peak — are found at the points of trend change: the beginning of a peak, the peak crest, and sometimes the end of a peak. This test is the heart of the algorithm.

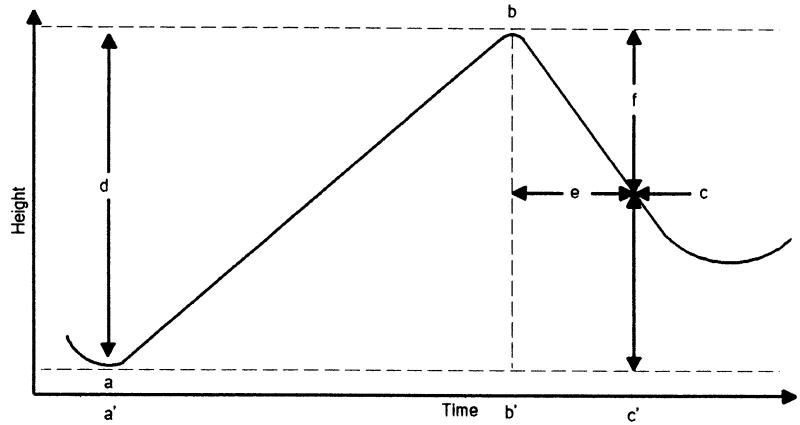
## B.4.4 Calculation of Area Under the Peak

Two peak characteristics that are not entirely dependent on points of dominant trend change are area and width. The area under the peak, or integral, is calculated by taking the sum of the area increments corresponding to each filtered point and half the area increment at the first and last points of a peak. The area increment at each filtered point is the product of its height times the number of points currently being averaged.

## B.4.5 Algorithm Definition of the Width of a Peak

Calculation of peak width must be explained in a little more detail. The peak-processing algorithm defines peak width as the difference between the time when the crest occurs and the time when a point is reached on the trailing side of the peak whose height is half the crest height as measured from the height of the leading minimum (Figure B-2).

**Figure B–2: Calculation of True Peak Width**

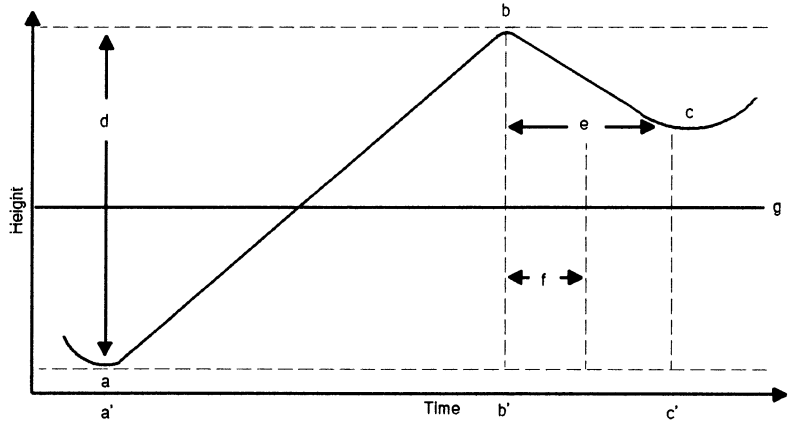



a = Leading Minimum Height
b = Crest Height
c = Point Whose Value is 1/2(a + b)
d = b-a
e = Peak Width = Time c'-Time b'
f = d/2

MR-1239-GE

It is possible that the data may establish an increasing trend on the trailing side of a peak before the point is reached where width is normally calculated. An increasing trend on the back of a peak is seen as terminating the peak; the width calculation for the peak is then made at the point where the increasing trend begins. The value calculated is called the *estimated peak width*, which is half the difference between time of crest occurrence and time at which the increasing trend is observed (Figure B–3).

**Figure B–3: Calculation of Estimated Peak Width**




a = Leading Minimum Height
b = Crest Height
c = Trailing Minimum Height (Point at Which Peak Ends Because Increasing Trend Is Detected)

d = b–a
e = Time c'–Time b'
f = e/2 (Estimated Width)
g = Height at Which Peak Width Would Have Been Calculated If Decreasing Trend Had Continued

MR-1240-GE

## B.4.6 Algorithmic Detection of the Baseline

A final and important step in peak analysis is to determine whether data reported for a peak have been affected by similar data observed for another peak. The algorithm checks to see whether recorded peak data indicate a period of relative quiescence before a new peak begins, or whether a new peak begins with no intervening quiescent period. Such quiescence relative to the overall peak contour is interpreted as a baseline. When a basline does not occur, the peak has ended at a valley. The problem thus becomes one of detecting when, or if, the baseline is reached.

Normally, assume that when the algorithm is initiated, input starts from a quiescent state. Therefore, you can take the point at which an increasing trend is first observed to be the current baseline height as well as the leading minimum height of the first peak. Because baseline

detection thereafter involves a comparison of relative minimums, this first detected minimum has a profound effect on the entire process.

Once a crest has been detected, any attempt to find a new baseline begins only after the width has been calculated. The time past crest detection when the baseline search begins is a function of the calculated width. Specifically, baseline detection begins at a time equal to crest time plus the product of the width and the baseline test factor (BT), an input variable parameter. The interval between crest detection and the start of baseline detection reflects the duration of a normal peak as it decays to a relatively quiescent state.

To detect an actual baseline height, calculate the slopes of successive tangent lines from the current baseline point to each new filtered point. If two successive increases in slope are observed before an increasing trend in the filtered data is established, the second of these points is taken as the termination of the peak, and the peak is seen as ending on a baseline.

If an increasing trend is established before two successive increases in slope are observed, the peak is said to end at a valley, the new peak begins at the point where the increasing trend is first observed, and the baseline data remains unchanged.

Note that even though two successive increases in slope indicate a baseline, the next peak does not begin until that point where another increasing trend is established. The leading minimum point for the next peak is interpreted as defining the height and time of the new baseline. The area between the trailing minimum of the last peak and the leading minimum of the new peak is ignored.[1]

## B.4.7 Flow Charts for the PEAK Routine

The series of flow charts presented as Figures B-4 through B-9 gives detailed logic for the PEAK routine. Supplementary information is presented in Tables B-1 through B-3. Table B-1 lists the combinations of switch/indicator settings that characterize significant events during peak detection. Table B-2 defines the symbols used in the flow charts and accompanying explanation. Figure B-10 and Table B-3 review and summarize the flow-charted events as they apply to three possible peak configurations:

- A peak starting on the baseline and ending on a new baseline

[1] Data taken during this period indicates that there is no peak-producing activity.

- A peak starting on the baseline and ending at a valley
- A peak starting at a valley and ending on either a baseline or a valley

**Table B–1: Switch Settings for Significant Events in Peak Definition**

| Significant Event Switch BS | Current Trend Indicators: Decreasing DI | Increasing II | What is Happening with Relation to Peak Processing |
|---|---|---|---|
| 0 | 0 | 0 | N/A |
| 0 | 0 | 1 | On front of peak that started on current baseline |
| 0 | 1 | 0 | Detected a baseline value; looking for a new peak to begin (initial condition) |
| 0 | 1 | 1 | Crest detected for peak that started on baseline |
| 1 | 0 | 0 | N/A |
| 1 | 0 | 1 | New peak begins before point is reached at which width is to be calculated (forced calculation of width) |
| 1 | 1 | 0 | After crest, looking for point where width is to be calculated |
| 1 | 1 | 1 | N/A |
| 2 | 0 | 0 | N/A |
| 2 | 0 | 1 | On front side of peak that started at a valley |
| 2 | 1 | 0 | Testing for new baseline value after width has been calculated |
| 2 | 1 | 1 | Crest detected for peak that started at a valley |

**Table B–2: Definition of Symbols**

| Symbol | Definition |
|---|---|
| BH | Current baseline height |
| BHT | Time of current baseline height |
| BS | Baseline switch<br>0 Peak starts on baseline<br>1 Looking for width<br>2 Looking for end on baseline |
| BT | Baseline test factor[1] |
| CH | Height of last crest[2] |
| CHT | Time of last crest[2] |
| DC | Current number of persistent decreases in filtered data |
| DI | Switch that is set (=1) if signal is decreasing |
| GT | Number of persistent changes (gating factor) that defines an increasing/decreasing trend[1] |
| IA | Accumulated area as signal increases |
| IC | Current number of persistent increases in filtered data |
| II | Switch that is set (=1) if signal is increasing |
| IM | Minimum differential between filtered data points that the algorithm interprets as signifying a real increase[1] |
| IPD | Switch that indicates whether an increase is needed in the number of points averaged:<br>IPD=PD if number of points is to be increased<br>IPD=0 if no increase is needed |
| LMH | Leading minimum height for peak[2] |
| LMT | Time of leading minimum height[2] |
| MNH | Current minimum height[2,3] |
| MNT | Time of current minimum height[2,3] |
| MXH | Current maximum height |
| MXT | Time of current maximum height |

[1]Value set by user.

[2]Value reported by algorithm.

[3]Value can change during peak detection; reported values are those that are current when the end of a peak is detected.

**Table B–2 (Cont.): Definition of Symbols**

| Symbol | Definition |
|---|---|
| OMH | Old minimum height (before increasing trend is established) |
| OMT | Time of old minimum height |
| OPD | Original point density[1] |
| OS | Old slope |
| PD | Point density; number of raw data points currently needed to obtain next average point[2,3] |
| SC | Slope increase counter; baseline test |
| SL | New or current slope |
| TA | Accumulated total area during peak formation[2,3] |
| TM | Raw point counter (current time) |
| WD | Width of peak[2] |
| XL | Large number used to reset small number |
| Y | Element of digital filter |
| $Y_0$ | Current filtered point, that is, center point of current window |
| Type | 0 Peak ends on valley<br>1 Peak ends on baseline[2] |

[1]Value set by user.

[2]Value reported by algorithm.

[3]Value can change during peak detection; reported values are those that are current when the end of a peak is detected.

**Figure B–4: Flow Chart for Peak Processing: Initialization, Data Averaging, and Application of Digital Filter**

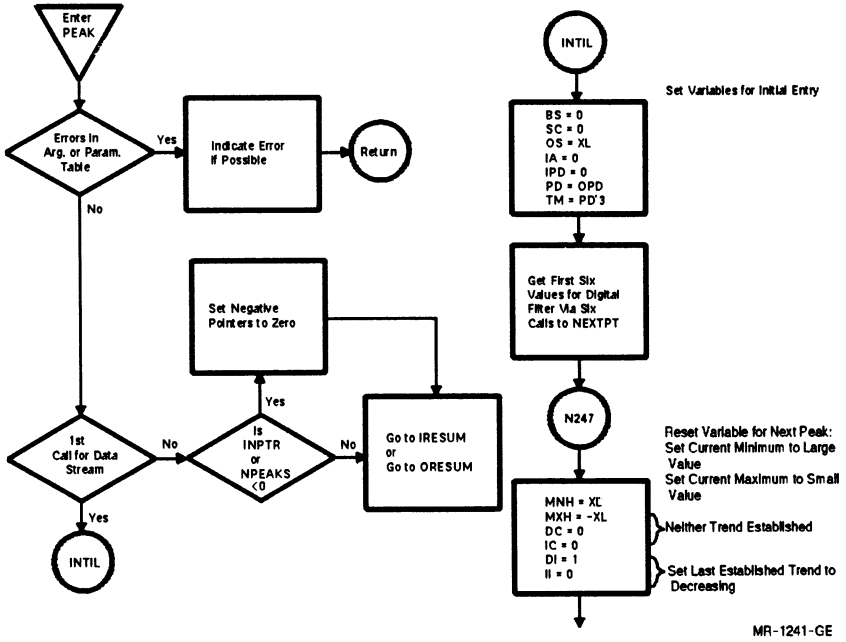
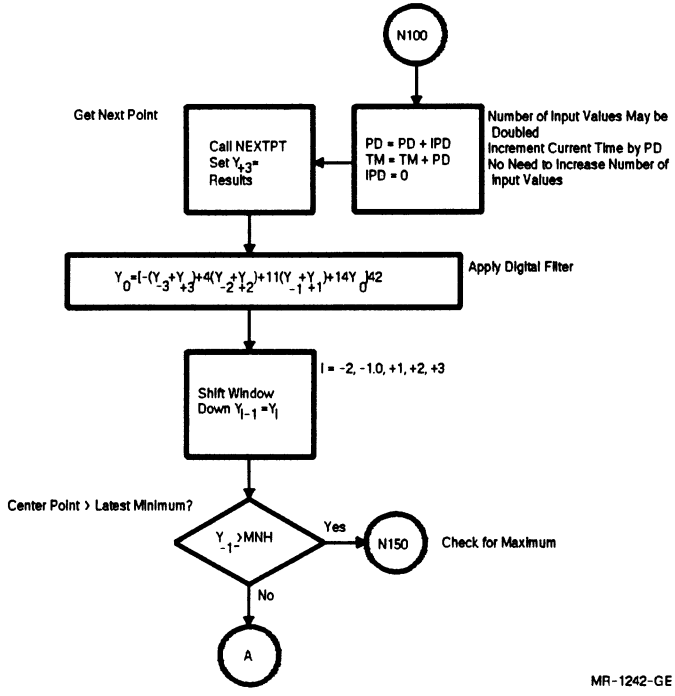



**Figure B–4 Cont'd. on next page**

**Figure B–4 (Cont.): Flow Chart for Peak Processing: Initialization, Data Averaging, and Application of Digital Filter**



## Figure B–5: Flow Chart for Peak Processing: Calculation of Peak Width and Search for Baseline

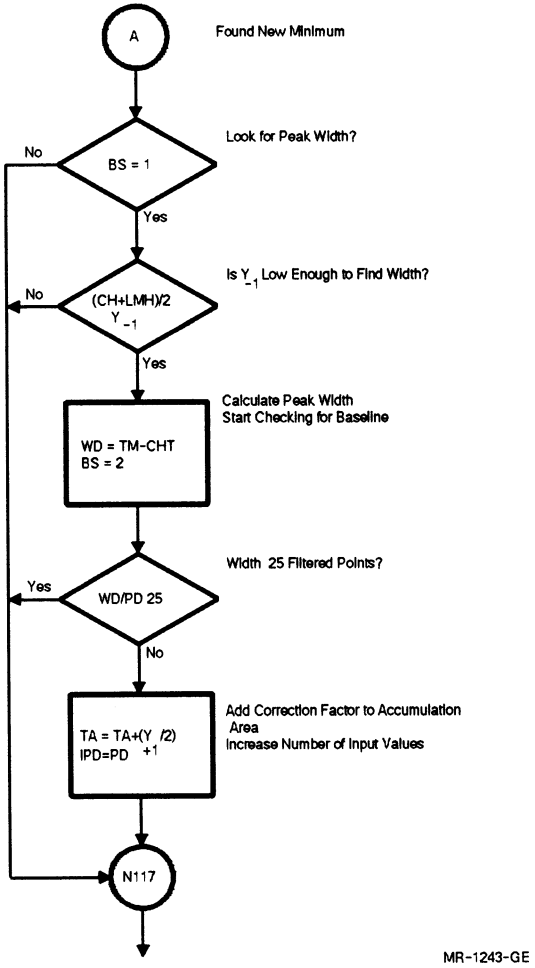
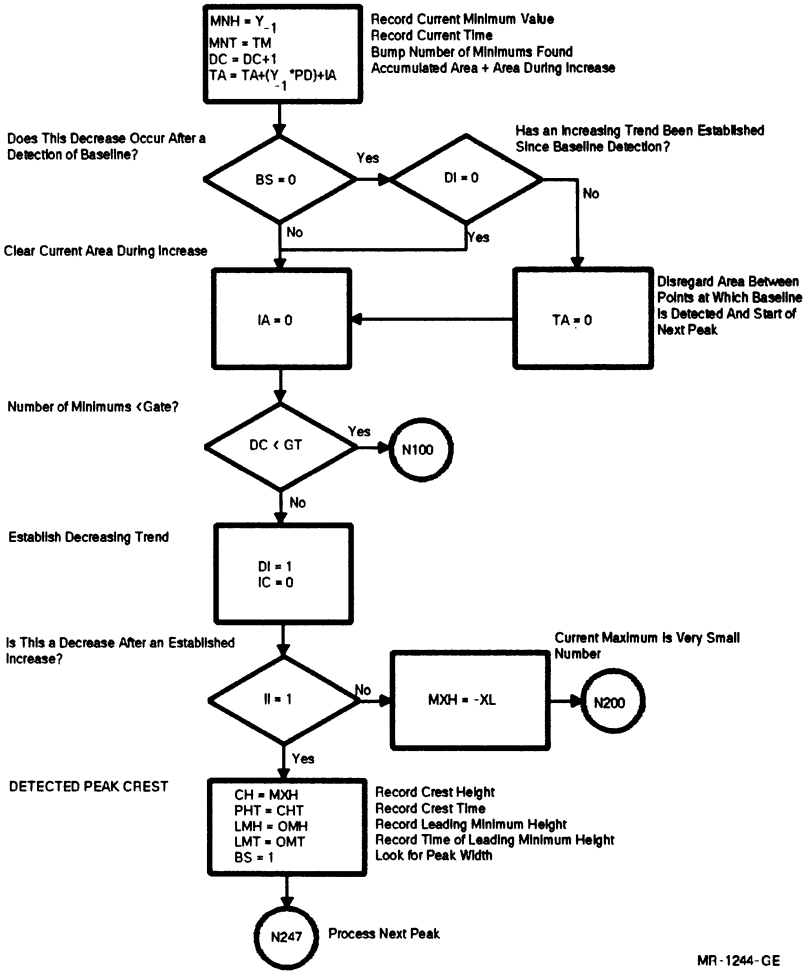



**Figure B–5 Cont'd. on next page**

## Figure B–5 (Cont.): Flow Chart for Peak Processing: Calculation of Peak Width and Search for Baseline



## Figure B–6: Flow Chart for Peak Processing: Area Calculation

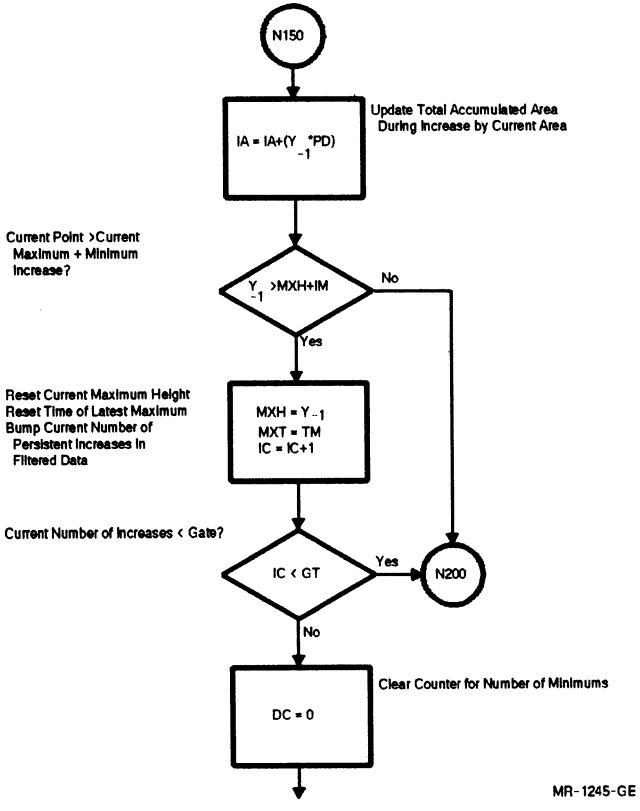
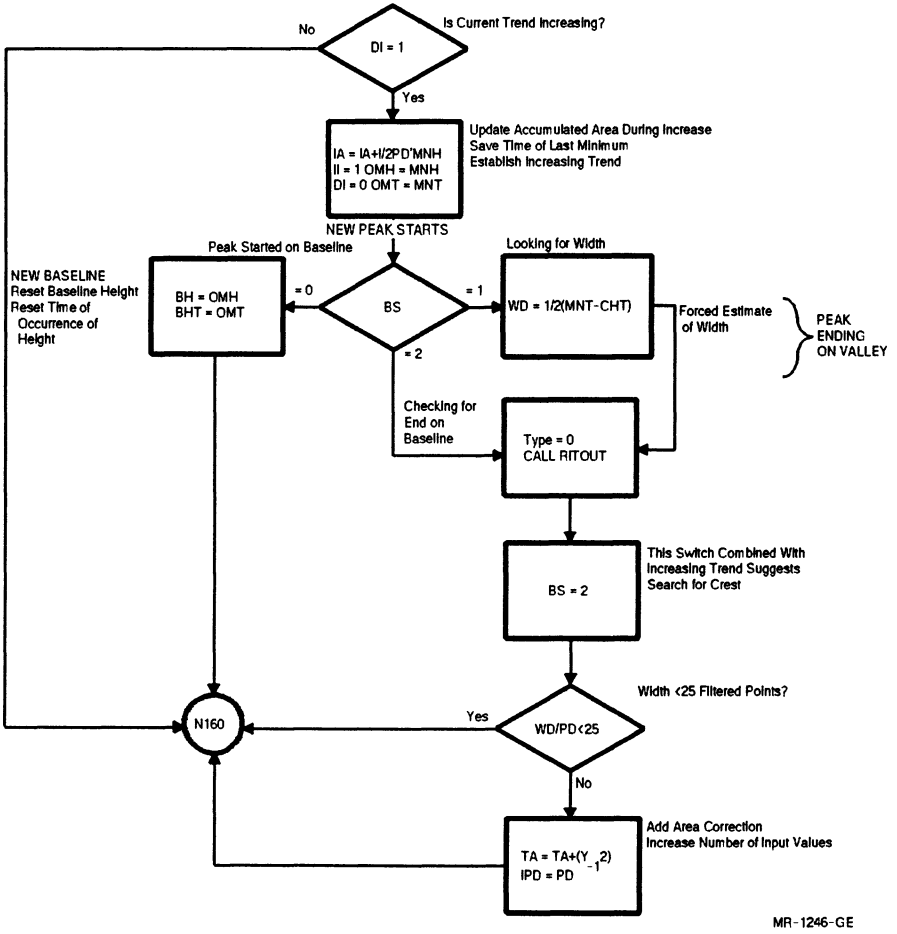



**Figure B–6 Cont'd. on next page**

## Figure B–6 (Cont.): Flow Chart for Peak Processing: Area Calculation



## Figure B–7: Flow Chart for Peak Processing: Determining the Baseline

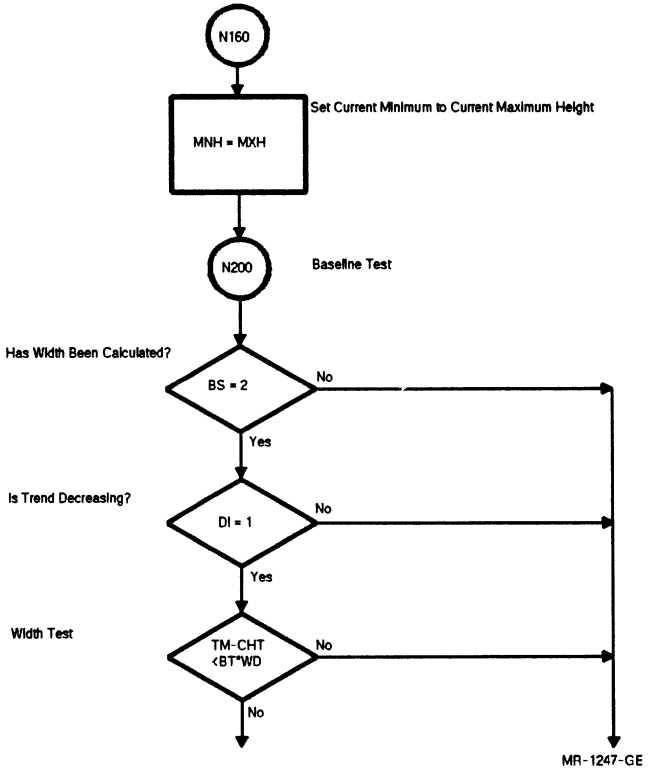
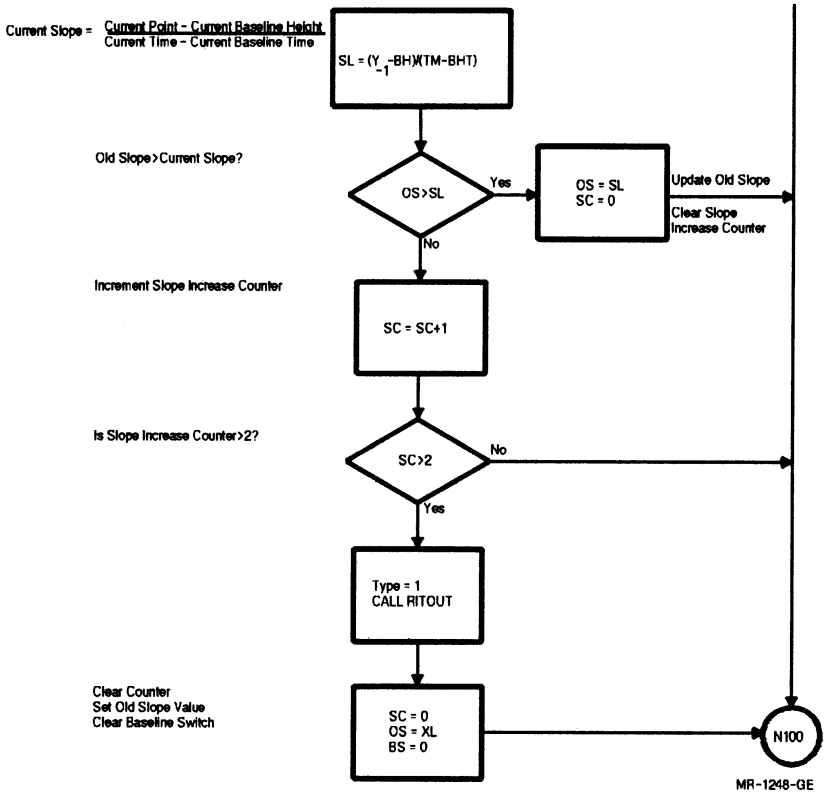
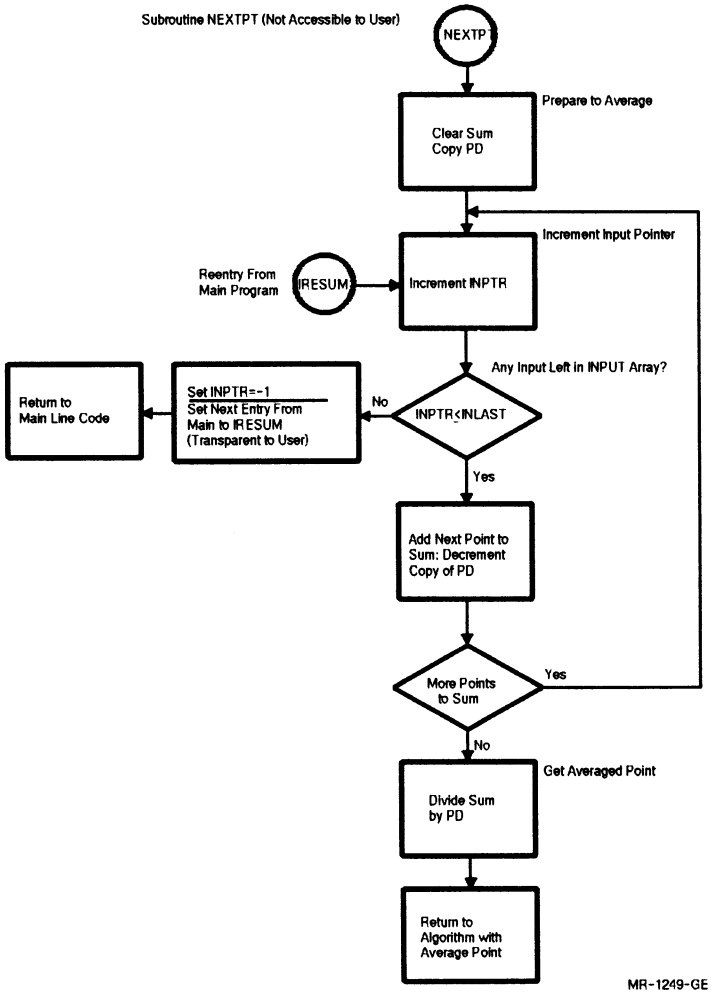



**Figure B–7 Cont'd. on next page**

## Figure B–7 (Cont.): Flow Chart for Peak Processing: Determining the Baseline



## Figure B–8: NEXTPT Routine — Peak Processing



## Figure B–9: RITOUT Routine — Peak Processing

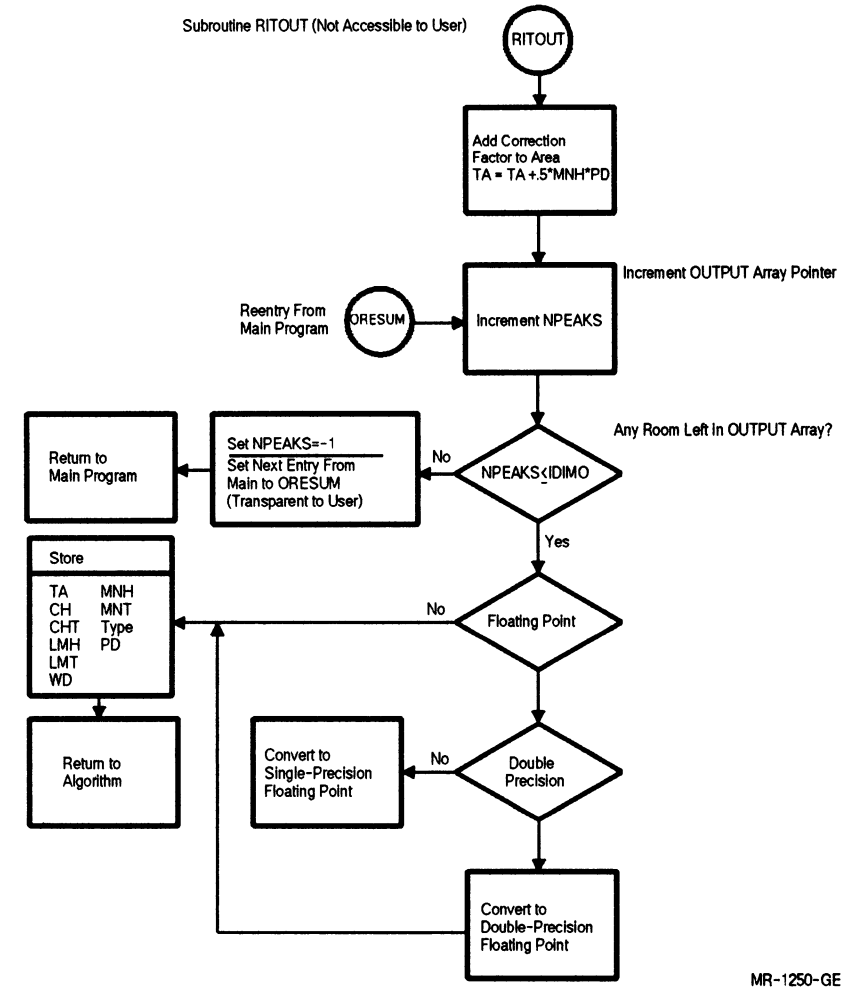


**Figure B–10: Flow Chart of Peak Events**

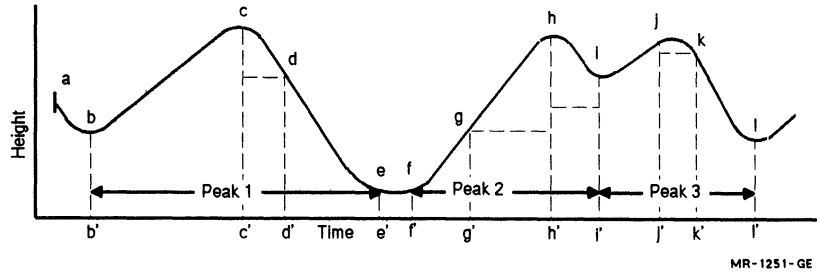


**Table B–3: Definition of Peak Events**

| Point/Section of Curve | Description | Flow Chart References(s) |
|---|---|---|
| | **START OF PEAK 1** | |
| a | Input begins | Flowchart begins |
| a-b | Decreasing trend in data after baseline detection | BS=0,DI=1,II=0<br>DC>GT, IC<GT |
| b | Increasing trend established; leading minimum height/time of Peak 1 detected; "new" baseline data (height and time) defined | OMH=b BH=b<br>OMT=b*t* BHT=b*t* |
| b-c | Increasing trend in data; change in established trend will indicate crest detection | BS=0,DI=0,II=1<br>DC<GT, IC>GT |
| c | Decreasing trend established; crest height and time detected and recorded; leading minimum data recorded; start looking for point where width is calculated | LMH=OMH CH=c<br>LMT=OMT CHT=c*t* |
| c-d | Decreasing trend in data after crest detection and before width calculation | BS=1,DI=1,II=0<br>DC>GT, IC<GT |
| d | Point where width is calculated; d=(b+c)/2 | WD=d*t*-c*t* |
| d-e | Decreasing trend in data after width is calculated and before baseline is detected | BS=2,DI=1,II=0<br>DC>GT, IC<GT |
| e | Baseline detected; Peak 1 ends at this point, which is recorded as trailing minimum | MNH=e<br>MNT=e*t*<br>Type=1 |
| | **END OF PEAK 1** | |
| e-f | Decreasing trend after baseline detection and before start of next peak; area under curve ignored | BS=0,DI=1,II=0<br>DC>GT, IC<GT |

**Table B–3 (Cont.): Definition of Peak Events**

| Point/Section of Curve | Description | Flow Chart References(s) |
|---|---|---|
| | **START OF PEAK 2** | |
| f | Increasing trend established; leading minimum (height and time) of Peak 2 detected; baseline data (height and time) redefined | OMH=f BH=f<br>OMT=*f* BHT=*f* |
| g | Height on Peak 2 (after crest detected) where width would be calculated if data were to decrease to this point before start of Peak 3;<br>g=(h+f)/2=($CH_2$-OMH)/2 | No corresponding point on flow chart |
| f-h | Increasing trend in data; change in established trend will indicate crest detection (see b-c) | BS=0,DI=0,II=1<br>DC<GT, IC>GT |
| h | Decreasing trend established; crest height and time detected and recorded; leading minimum data recorded; start looking for data value g | LMH=OMH CH=h<br>LMT=OMT CHT=h*i* |
| h-i | Decreasing trend in data after crest detection and before width calculation (see c-d) | BS=1,DI=1,II=0<br>DC>GT, IC<GT |
| i | Increasing trend established before width of Peak 2 calculated; forced estimation of width of Peak 2 as (*i*-n*i*); Peak 2 ends at valley with i as trailing minimum for Peak 2; Peak 3 begins with i as lead ing minimum; baseline data remain unchanged | WD=(MNT-CHT)/2<br>MNH=i BH=f OMH=i<br>MNT=*i* BHT=*f* OMT=*i*<br>Type=0 |

**Table B–3 (Cont.): Definition of Peak Events**

| Point/Section of Curve | Description | Flow Chart References(s) |
|---|---|---|
| | **END OF PEAK 2/START OF PEAK 3** | |
| i-j | Increasing trend in data; change in established trend will indicate crest detection (see b-c, f-h) | BS=2,DI=0,II=1<br>DC<GT, IC>GT |
| j | Decreasing trend established; crest height and time detected and recorded; leading minimum data recorded; start looking for point where width is to be calculated | LMH=OMH CH=j<br>LMT=OMT, CHT=j′ |
| j-k | Decreasing trend in data after crest detection and before width calculation (see c-d) | BS=1,DI=1,II=0<br>DC>GT, IC<GT |
| k | Point where width is calculated; k=(j+i)/2 | WD=k′-j′ |
| k-l | Decreasing trend in data after width is calculated and before baseline is detected (see d-e) | BS=2,DI=1,II=0<br>DC>GT, IC<GT |
| l | Increasing trend establish before baseline is detected; Peak 3 ends at valley with 1 as trailing minimum; Peak 4 begins with 1 as leading minimum; baseline data remain unchanged | MNH=1 BH=f OMH=1<br>MNT=1′ BHT=f′<br>OMT=1′<br>Type=0 |
| | **END OF PEAK 3/START OF PEAK 4** | |
| | Peak 4 not shown in illustration | |

## B.5 How to Call the Peak-Processing Routine

The symbolic name for the peak-processing routine is PEAK, and the general format for the FORTRAN call is:

**CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)**

For reference, argument names in the call to PEAK have been assigned arbitrarily. You can supply your own argument names, but you must state all of the arguments explicitly. There are no default values for any of the arguments. If you omit an argument, or if you supply too

many arguments, a FORTRAN error message results, and no data is processed. The arguments are described in the following paragraphs.

**ITABLE** is an integer array of length 79 used to store intermediate results and other information required by the algorithm. You must set the values of the following array elements to transmit variable parameters and other information to the routine.

ITABLE(1) Number of raw input values to be averaged to determine a point for use by the digital filter. This variable parameter is called the original point density (OPD) in the description of the algorithm. In general, the OPD should be chosen so that the number of averaged data points on the first peak is about 100.

ITABLE(2) The baseline test (BT) factor (Section B.4.6). On a peak whose width is WD, baseline detection begins at time WD-ITABLE(2) past crest time. In general, suggested values can range from 3 to 5.

ITABLE(3) The number of either persistent or consecutive local changes in one direction needed to establish a new dominant directional trend. It is the gate parameter discussed in Section B.4.3. In general, suggested values can range from 3 to 8.

ITABLE(4) Minimum differential (IM) between filtered data points that the algorithm interprets as a real increase. This element, with ITABLE(3), determines real changes in dominant directional trends (Section B.4.3). In general, suggested values can range from 1 to 5.

ITABLE(5) The data type of the output array:

=0 output type is INTEGER*4
=1 output type is REAL*4
=-1 output type is REAL*8

ITABLE(6) Error indicator in the calling sequence or input parameters:

=0 Indicates no error

=N Indicates ITABLE(N) is in error, for example:

ITABLE(1)≤0
ITABLE(2)≤0

=-N Indicates the Nth argument is in error, for example, INPTR>INLAST (see the following discussion).
=-8 Indicates that the calculated area to this point has caused an overflow. That is, it exceeds $2_{31}$ -1. When the overflow is detected, PEAK returns with INPTR and NPEAKS set as usual. However, OUTPUT (1, NPEAKS+1) will contain the value of the area of the current peak, up to and including the point of overflow. You must take corrective action by saving this value and returning to the PEAK routine for further processing. PEAK calculates the remaining area and peak characteristics. When PEAK returns again, the peak area reported is the area of the peak from the last point of overflow. To determine the actual area of the peak, simply convert the overflowed value to a positive, double-precision, real number and add it to the remaining area of the peak.

ITABLE(7) This element must be set to zero before the initial call is made to the routine for each new stream of data. When the routine processes a data stream in "parts" (Section B.6), it uses ITABLE(7) for reentry to process each subsequent part. Thus, this element should not be altered by a user until all parts have been processed.

ITABLE(8) This element specifies the data type of the input data as follows:

=0 Input data single-precision INTEGER*2
=1 Input data double-precision INTEGER*4

ITABLE(9) Elements used exclusively by the routine while the data stream is being processed.

.
.
.

ITABLE(79)

**INPUT** is a single- or double-precision array containing the raw data to be processed.

**INLAST** is an INTEGER*4 variable having the value of the subscript of the last element of INPUT containing data.

**INPTR** is an INTEGER*4 variable having the value of the subscript of the last element processed by PEAK. You can also think of it as having a value one less than the subscript of the next datum in INPUT to be *processed*. For example, if the first element of the array is to be processed, INPTR should be set to zero. You must set the value of INPTR before calling PEAK; however, PEAK changes the value before returning.

**OUTPUT** is a double-subscripted array used to store the results of applying the peak-processing algorithm. The first dimension specifies the number of data elements to be output for each peak detected; there are always 10. The second dimension specifies the number of sets of peak data that can be stored by the algorithm while processing the input data. The second dimension is defined by IDIMO.

The data type of the output array is optional and can be any of those specified by ITABLE(5).

The 10 data elements reported for each peak are:

| | |
|---|---|
| OUTPUT(1,N) | Area of Nth peak |
| OUTPUT(2,N) | Height of crest, Nth peak |
| OUTPUT(3,N) | Time of crest, Nth peak |
| OUTPUT(4,N) | Height of leading minimum for Nth peak |
| OUTPUT(5,N) | Time of leading minimum for Nth peak |
| OUTPUT(6,N) | Width of Nth peak |
| OUTPUT(7,N) | Height of trailing minimum for Nth peak |
| OUTPUT(8,N) | Time of trailing minimum for Nth peak |
| OUTPUT(9,N) | Indicator of how peak ended:<br>=0 ended on valley<br>=1 ended on baseline |
| OUTPUT(10,N) | *Current* number of input data points being averaged |

**IDIMO** is an INTEGER*4 variable that transmits to the routine the second dimension of the output array. It defines the number of peaks that can be reported before the output array is filled.

**NPEAKS** is an INTEGER*4 variable giving the number of peak data sets stored in the output array. We can also think of it as having a value of one less than the second subscript for the next set of output data to be stored. For example, for the initial set of peak data to be stored, set NPEAKS to zero.

You must set the value of this argument before calling the routine; however, the routine can change the value before returning.

**NOTE**

PEAK returns (assuming there are no errors) after either of the following events:

- All input data elements have been processed.
- The output array is filled, and there is another set of peak data to report.

  The arguments INPTR and NPEAKS indicate which event caused the return and the current status of I/O processing:

  — If condition 1 occurred then, INPTR=-1 and NPEAKS≤IDIMO, that is, the routine has set NPEAKS to the proper value for the next routine call.

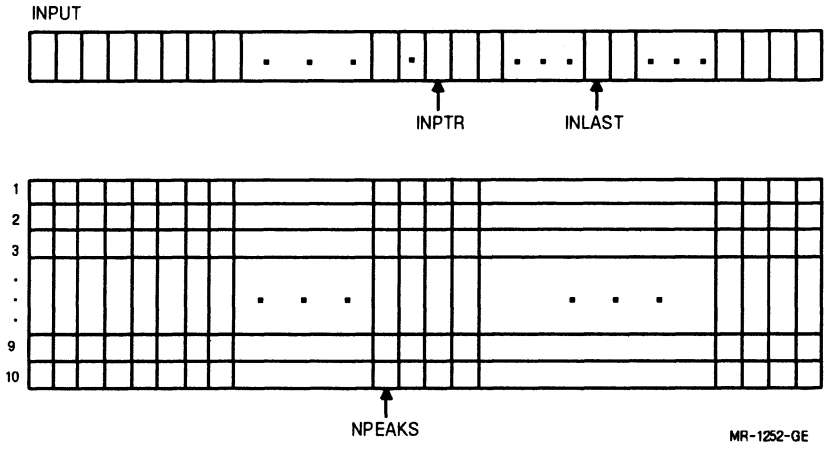
  — If condition 2 occurred, NPEAKS=-1 and INPTR equals the proper subscript value for reentry — one less than the subscript of the next element to be processed.

  If the routine is called again with either INPTR or NPEAKS equal to -1, the routine interprets the value as zero.

## B.6 Using the Peak-Processing Routine

You can use several inherent features of the peak-processing routine to process data produced in real time. Thus, you may use PEAK in conjunction with other routines that monitor and digitize real phenomena. The particular arguments that make possible this real-time application are INPTR, INLAST, and NPEAKS (see Section B.5). You can visualize the input and output arrays as a series of "pigeonholes," and INPTR and NPEAKS as pointers to the next available data element to be processed and the next slot for outputting the data, respectively (Figure B-11). INLAST is a pointer to the last INPUT element containing data.

**Figure B–11: INPTR, INLAST, and NPEAKS Point to Slots**




The routine returns when all data in the input buffer have been processed, that is, INPTR=INLAST, or the output array is filled, whichever occurs first. If all data in the input buffer have been processed, INPTR will equal -1 and NPEAKS will point to the last slot (subscript) in the output array that was filled. If, conversely, all slots in the output array have been filled, NPEAKS = -1 and INPTR points to the last element (subscript) in the input array that was processed. Neither is an error condition, and neither is more advantageous outside the context of your specific application.

These conditions give you great flexibility in handling routine input and output. When you have large quantities of data to process, you need not allocate space for all data at once because the routine is designed to process a given data set in sequential parts. In fact, all data need not be known before processing begins, as is true in real-time processing. Data can be asynchronously collected into one buffer at the same time that a previously collected buffer is processed.

Handling of output is also flexible. It might, for example, be printed or stored after each return from the routine, or it might be further processed only when the output buffer was filled, that is, NPEAKS = -1. You can choose the procedure that is most convenient for you.

Further flexibility is introduced by the fact that all arguments in the calling statement expect ITABLE can be changed between successive calls to the routine to reflect the origin of the remaining input data and where the output is to be stored. ITABLE must not be tampered with during the intervals between calls for a given data stream because it contains the current information needed to resume processing at the point where processing was stopped on the previous call.

The routine is position-independent and reentrant. Although these features are of interest mainly at the system level, they do result in additional advantages at the user level. Perhaps most significant is the possibility of processing several data streams simultaneously. All pertinent information concerning the history of a data stream is contained in the ITABLE array rather than in the code for the routine. Imaginative use of the arguments in the routine call should make the routine functionally compatible with any application that uses the peak-processing algorithm.

## B.7 Sample Program Using the PEAK Routine

The sample program presented in this section processes a waveform — the sum of four Gaussian curves — shown in Figure B–12.

This sample program is idealized in several respects. Normally, you do not know that the input array is empty upon return from the routine, or that the output array has sufficient room for all output data. You must therefore provide for these possibilities by checking INPTR and NPEAKS. Also, no provision is made for error checking because the input and output are known and the program has been debugged with respect to these types of errors. In practice, ITABLE(6) should always be checked. This program is used to illustrate the minimal requirements for implementation, and how the routine and its arguments affect a given set of data.

The data are input as four 256-point parts; the routine processes each part as it is received, placing the results in the output array. In this case, the output array is large enough to contain the complete set of processed data. Upon return from the routine, the input array is always empty (INPTR = -1), and the output array is never filled (NPEAKS$\neq$-1).

## Example B–1: Sample Program Using the PEAK Routine

```
C Define array variables and their size:

        DIMENSION INPUT(256),OUTPUT(10,3),EMU(4),SIGMA(4),SIZE(4)
        DIMENSION ITABLE(79),VTYPE(2,2)

C VTYPE is used to print a word describing how the peak ended (TYPE).

        DATA VTYPE/'  VA','LLEY','BASE','LINE'/

C Arrays EMU, SIGMA, and SIZE are used to produce the waveform to
C be processed, which is the sum of four Gaussian curves.

        DATA EMU/20.,70.,600.,1000./
        DATA SIGMA/20.,10.,200.,100./
        DATA SIZE/950.,400.,300.,200./

C Data statements initializing the variable input parameters to the
C algorithm (ITABLE) and the arguments for the call to PEAK.

        DATA ITABLE/1,2,3,1,1,63*0/
        DATA INLAST,INPTR,IDIMO,NPEAKS/256,0,3,0/

C Section producing values that represent the waveform; as X increases,
C the next 256 values are calculated and PEAK is called.  Four waveform
C segments are produced.

        X=0.
        DO 3 K=1,4
        DO 1 I=1,256
        A=0.
        X=X+1
        DO 2 J=1,4
      2 A=A+SIZE(J)*EXP(-.5*((X-EMU(J)/SIGMA(J))**2)
      1 INPUT(I)=A

C Call to the PEAK routine:

        CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)

C Loop for each of four sections of waveform.  All elements of INPUT
C array are processed (INPTR equals -1) but OUTPUT array still has room
C (NPEAKS less than or equal to IDIMO).

      3 CONTINUE
```

Example B–1 Cont'd. on next page

## Example B–1 (Cont.): Sample Program Using the PEAK Routine

```
C This section displays the results on the terminal screen:

      TYPE 900
  900 FORMAT (1H1,T24,'PEAK Example 1'//)
      TYPE 1000
 1000 FORMAT ('PEAK NO.',8X,'AREA',4X,'P HEIGHT',6X,'P TIME',4X,
     A 'L HEIGHT',6X,'L TIME',/,11X,'HALF WIDTH',4X,'T HEIGHT',6X,
     B 'T TIME',8X,'TYPE',8X,'RATE',//)
      DO 4 L=1,NPEAKS
      KK=OUTPUT(9,L)+1
    4 TYPE 2000,(L,(OUTPUT(I,L),I=1,8),(VTYPE(K,KK),K=1,2),OUTPUT(10,L))
 2000 FORMAT(I9,5F12.0,/,9X,3F12.0,4X,2A4,F12.0)
      END
```

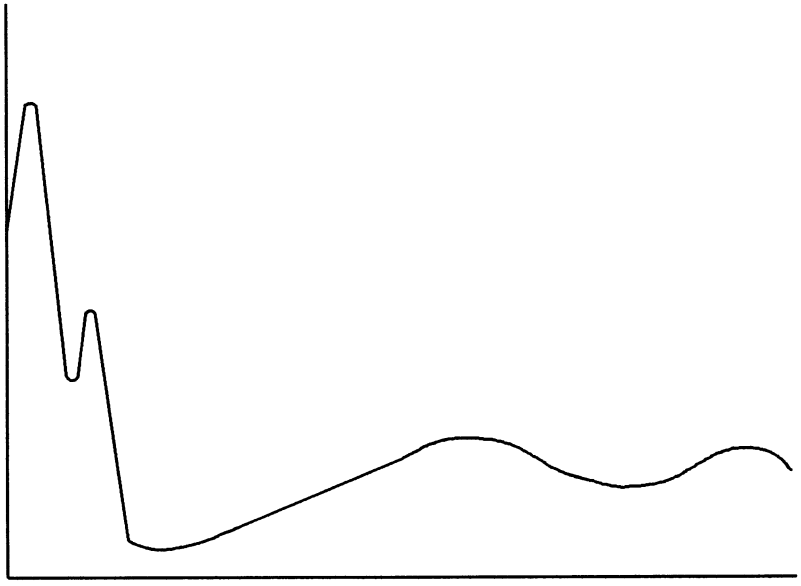
This program produces the following terminal output with the digital filter enabled:

```
                        PEAK Example 1

PEAK NO.          AREA     P HEIGHT     P TIME     L HEIGHT     L TIME
             HALF WIDTH     T HEIGHT     T TIME         TYPE       RATE

       1         35795.         951.         19.         692.         4.
                    12.         345.         53.     BASELINE         1.
       2         11803.         451.         68.         343.        54.
                     7.          41.         93.     BASELINE         1.
       3        134928.         299.        596.          13.       106.
                   124.         200.        845.       VALLEY         1.
```

This program produces the following terminal output with the No Filter (NOFLT$) option enabled:

```
                        PEAK Example 1

PEAK NO.          AREA     P HEIGHT     P TIME     L HEIGHT     L TIME
             HALF WIDTH     T HEIGHT     T TIME         TYPE       RATE

       1         38147.         953.         19.         608.         1.
                    14.         342.         54.     BASELINE         1.
       2         11835.         454.         68.         342.        54.
                     7.          41.         93.     BASELINE         1.
       3        132652.         300.        597.          14.       106.
                   117.         201.        831.       VALLEY         1.
```

**Figure B–12: Actual Plot of the Input Data in Example C-1**



MR-1253-GE

# Index


## A

Analog-to-digital data translation • 6–38
Autocorrelation function • 6–11
  definition • 2–7
  mathematical equation • 2–7
  references • 2–8

## C

Condition values
  LSP • 7–3
Correlation function • 6–11
  definition • 2–7
  mathematical equation • 2–7
  references • 2–8

## D

Data format translation • 1–3
Data translation
  analog-to-digital • 6–38
  digital-to-analog • 6–41
Digital filtering
  definition • 3–1
  LSP$APPLY_WINDOW_TABLE routine • 6–5
  LSP$BUILD_WINDOW_TABLE routine • 6–8
  LSP$SPECTRAL_WINDOWS routine • 6–62
  nonrecursive filtering • 3–3
  polynomials • 3–1
  references • 3–13
  spectral window filtering • 4–2
Digital-to-analog data translation • 6–41
Discrete Fourier transform
  definition • 2–3
  mathematical equation • 2–3

## E

Error messages
  checking routine call status • 7–2
  explanation and user action • 7–3
  LSP • 7–1
  symbolic status definition files • 7–2

## F

Fast Fourier transform
  in two dimensions • 2–5
  mathematical equation • 2–3
  of real-valued data • 6–20
  reduced-symmetric storage • 2–5
  references • 2–8
Forward Fourier transform
  definition • 2–1
  of complex-valued data • 6–14
  of complex-valued data in two dimensions • 6–17

## G

Gibbs Phenomenon • 3–6

## I

Interval histogram analysis
  with floating-point input • 6–45

Interval histogram analysis (cont'd.)
with integer input • 6–49
Inverse Fourier transform
definition • 2–2
mathematical equation • 2–3
of complex-valued data • 6–14
of complex-valued data in two dimensions • 6–17
of real-valued data • 6–20

## L

Laboratory signal-processing routines
error messages • 7–1
LSP$APPLY_SPECTRAL_WINDOWS_TABLE routine • 6–5
LSP$BUILD_WINDOW_TABLE routine • 6–8
LSP$CORRELATION routine • 6–11
LSP$FFT_COMPLEX routine • 6–14
LSP$FFT_COMPLEX_2D routine • 6–17
LSP$FFT_REAL routine • 6–20
LSP$FILTER_NONREC routine • 6–23
LSP$FILTER_POLY routine • 6–26
LSP$FILTER_POLY_1ST_DERIV routine • 6–29
LSP$FILTER_POLY_2ND_DERIV routine • 6–32
LSP$FILTER_POLY_3RD_DERIV routine • 6–35
LSP$FORMAT_TRANSLATE_ADC routine • 6–38
LSP$FORMAT_TRANSLATE_DAC routine • 6–41
LSP$HIST_F routine • 6–45
LSP$HIST_I routine • 6–49
LSP$PHASE_ANGLE routine • 6–53
LSP$PHASE_ANGLE_2D routine • 6–56
LSP$POWER_SPECTRUM routine • 6–59
LSP$SPECTRAL_WINDOWS routine • 6–62
LSP$THERMOCOUPLE_X routine • 6–65
LSP sample programs • 8–1

## N

Nonrecursive filtering • 6–23
bandpass • 3–3
bandstop • 3–3
highpass • 3–3
lowpass • 3–3
mathematical equation • 3–4
sample program • 3–6
Nyquist frequency • 3–3

## P

Periodogram
power spectrum • 4–2
Phase angle and amplitude spectra • 6–53, 6–56
Polynomial filtering
definition • 3–1
for smoothing • 6–26
with first-derivative output • 6–29
with second-derivative output • 6–32
with third-derivative output • 6–35
Power spectrum • 6–59
periodogram technique • 4–2

## S

Sample programs
using LSP routines • 8–1
Signal-processing routines
LSP$APPLY_SPECTRAL_WINDOWS_TABLE • 6–5
LSP$BUILD_WINDOW_TABLE • 6–8
LSP$CORRELATION • 6–11
LSP$FFT_COMPLEX • 6–14
LSP$FFT_COMPLEX_2D • 6–17
LSP$FFT_REAL • 6–20
LSP$FILTER_NONREC • 6–23
LSP$FILTER_POLY • 6–26
LSP$FILTER_POLY_1ST_DERIV • 6–29
LSP$FILTER_POLY_2ND_DERIV • 6–32
LSP$FILTER_POLY_3RD_DERIV • 6–35
LSP$FORMAT_TRANSLATE_ADC • 6–38
LSP$FORMAT_TRANSLATE_DAC • 6–41
LSP$HIST_F • 6–45
LSP$HIST_I • 6–49
LSP$PHASE_ANGLE • 6–53
LSP$PHASE_ANGLE_2D • 6–56
LSP$POWER_SPECTRUM • 6–59
LSP$SPECTRAL_WINDOWS • 6–62
LSP$THERMOCOUPLE_X • 6–65
Spectral window
symbolic status definition files • 4–14
Spectral window filtering
applying coefficient table • 6–5
building coefficient table • 6–8
LSP$SPECTRAL_WINDOWS routine • 6–62

Spectral window filtering (cont'd.)
LSP window algorithms • 4–4
LSP window routines • 4–4
periodogram technique • 4–2
sample program • 4–9
spectral window reference list • 4–14
window types illustration • 4–6
Spectral Window routine • 6–62

## T

Thermocouple conversion • 5–1
LSP$THERMOCOUPLE_X • 5–1
temperature and voltage ranges • 5–2
Thermocouple conversion routines • 5–1

## V

VAXlab signal-processing
analog-to-digital data translation • 1–1
calculating phase angles • 1–2
calculating the correlation function • 1–2
converting thermocouple voltages to temperatures • 1–2
determining power spectra • 1–2
digital filtering • 1–2
digital-to-analog data translation • 1–1
Gibbs Phenomenon • 3–6
interval histogram analysis • 1–2
number representations • 1–3
binary • 1–3
offset binary • 1–4
two's complement • 1–4
Nyquist frequency • 3–3
performing Fourier transform • 1–1
spectral window filtering • 1–2

## READER'S COMMENTS

Your comments and suggestions help us to improve the quality of our publications.

**For which tasks did you use this manual?** (Circle your responses.)

(a) Installation (c) Maintenance (e) Training
(b) Operation/use (d) Programming (f) Other (Please specify.) _____

**Did the manual meet your needs?** Yes ☐ No ☐ Why? _____

**Please rate the manual in the following categories.** (Circle your responses.)

| | Excellent | Good | Fair | Poor | Unacceptable |
|---|---|---|---|---|---|
| Accuracy (product works as described) | 5 | 4 | 3 | 2 | 1 |
| Clarity (easy to understand) | 5 | 4 | 3 | 2 | 1 |
| Completeness (enough information) | 5 | 4 | 3 | 2 | 1 |
| Organization (structure of subject matter) | 5 | 4 | 3 | 2 | 1 |
| Table of Contents, Index (ability to find topic) | 5 | 4 | 3 | 2 | 1 |
| Illustrations, examples (useful) | 5 | 4 | 3 | 2 | 1 |
| Overall ease of use | 5 | 4 | 3 | 2 | 1 |
| Page Layout (easy to find information) | 5 | 4 | 3 | 2 | 1 |
| Print Quality (easy to read) | 5 | 4 | 3 | 2 | 1 |

**What things did you like *most* about this manual?** _____

**What things did you like *least* about this manual?** _____

**Please list and describe any errors you found in the manual.**

| Page | Description/Location of Error |
|---|---|
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

**Additional comments or suggestions for improving this manual:** _____

Name _____ Job Title _____
Street _____ Company _____
City _____ Department _____
State/Country _____ Telephone Number _____
Postal (ZIP) Code _____ Date _____

Fold Here and Tape

Affix
Stamp
Here

**DIGITAL EQUIPMENT CORPORATION**
**CORPORATE USER PUBLICATIONS**
200 FOREST STREET MR01-2/L12
MARLBOROUGH, MA 01752-9101

Fold Here