Laboratory Subroutines Programmer's Reference Manual

AA–C984C–TC

digital software

Laboratory Subroutines Programmer's Reference Manual

AA–C984C–TC

October 1981

This document describes the Laboratory Subroutines Package available to users of FORTRAN/RT–11 and RSX–11M FORTRAN IV and FORTRAN 77.

This manual replaces the *Laboratory Subroutines Programmer's Reference Manual*, order number AA–C984B–TC.

OPERATING SYSTEM:	RT–11 Version 4.0 RSX–11M Version 3.2
SOFTWARE:	FORTRAN IV Version 2.5 FORTRAN 77 Version 4.0 LSP Version 1.2

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid–Atlantic Region
Technical Documentation Center
Cotton Road
Nashua, New Hampshire 03060
Telephone: (800)258–1710
NH residents: (603)884–6660

Central Region
Technical Documentation Center
1050 East Remington Road
Schaumburg, Illinois 60195
Telephone: (312)640–5612

Western Region
Technical Documentation Center
2525 Augustine Drive
Santa Clara, California 95051
Telephone: (408)984–0200

digital equipment corporation • marlboro, massachusetts

First Printing, February 1978
Revised, June 1980
Revised, October 1981

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright ©, 1978, 1980, 1981, Digital Equipment Corporation.
All Rights Reserved.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DECnet	IAS
DECUS	DECsystem–10	MASSBUS
DECSYSTEM–20	PDT	PDP
DECwriter	RSTS	UNIBUS
DIBOL	RSX	VAX
EduSystem	VMS	VT
digital	MINC–11	MINC–23
		MINC/DECLAB23

Contents

Page

Preface

Chapter 1 Introduction to the Laboratory Subroutines

1.1 The Laboratory Subroutines Package 1–1
1.2 The Laboratory Subroutines Package Distribution Kit 1–3

Chapter 2 The Peak-Processing (PEAK) Subroutine

2.1 Definition of Basic Terms and Conventions 2–1
2.2 The Peak-Processing Algorithm: Processing Raw Data 2–2
2.2.1 Averaging of Input Data 2–2
2.2.2 Use of the Digital Filter 2–3
2.2.3 Trend Detection — Application of the Gate Factor 2–3
2.2.4 Calculation of Area under the Peak 2–4
2.2.5 Algorithm Definition of the Width of a Peak 2–4
2.2.6 Algorithmic Detection of the Baseline 2–6
2.2.7 Flow Charts for the PEAK Subroutine 2–7
2.3 How to Call the Peak-Processing Subroutine 2–17
2.4 Using the Peak-Processing Subroutine 2–20
2.5 Modifying the Subroutine — Using Options 2–22
2.5.1 EIS (Extended Instruction Set) 2–22
2.5.2 EAE (Extended Arithmetic Element) 2–22
2.5.3 AUTOG$ (Autogaining) 2–22
2.5.4 DPP$ (Double Precision Integers) 2–24
2.5.5 NOFLT$ (No Filter) 2–24
2.6 Examples Using the PEAK Subroutine 2–24

Chapter 3 The Envelope-Processing (NVELOP) Subroutine

Page

3.1 Definition of Basic Terms and Conventions 3–2
3.2 The Envelope-Processing Algorithm 3–2
3.2.1 The Baseline Value 3–3
3.2.2 Trend Detection — Application of the Gate Factor 3–3
3.2.3 The Width of a Peak 3–3
3.2.4 Calculating the Area of a Peak 3–4
3.2.5 Calculating the Centroid of a Peak 3–4
3.2.6 Flow Charts for the NVELOP Subroutine 3–4
3.3 How to Call the Envelope-Processing Subroutine 3–16
3.4 Using the Envelope-Processing Subroutine 3–20
3.5 Modifying the Subroutine — Using Options 3–22
3.5.1 EIS (Extended Instruction Set) 3–22
3.5.2 EAE (Extended Arithmetic Element) 3–22
3.6 Examples Using the NVELOP Subroutine 3–22

Chapter 4 The Interval Histogramming (HISTI) Subroutine

4.1 Definition of Basic Terms and Conventions 4–1
4.2 Your Input to the Subroutine: Its Characteristics 4–2
4.2.1 The Relation between Data and Categories 4–2
4.2.2 Describing the Categories 4–3
4.2.3 Overflow and Underflow Counts 4–3
4.2.4 How the Subroutine Interprets Single-Precision Numbers . . . 4–4
4.3 How to Call the Interval Histogramming Subroutine 4–5
4.4 Input and Output — Using the Subroutine 4–7
4.5 Modifying the Subroutine — Using Options 4–7
4.5.1 EIS (Extended Instruction Set) 4–8
4.5.2 EAE (Extended Arithmetic Element) 4–8
4.5.3 DPH$ (Double-Precision Integers) 4–8
4.5.4 FREQ$ (Frequency Histogram) 4–8
4.6 Examples of Input/Output Variation Using the HISTI Subroutine 4–10

Chapter 5 The Interval Histogramming with Reference Points (RHISTI) Subroutine

5.1 Definitions of Basic Terms and Conventions 5–1
5.2 Your Input to the Subroutine: Its Characteristics 5–2
5.2.1 The Reference Points — Their Significance 5–2
5.2.2 The Relation between Data and Categories 5–2
5.2.3 How the Subroutine Interprets Single-Precision Numbers . . . 5–3
5.3 How to Present Your Problem to the Subroutine 5–5
5.3.1 Number of Events to be Processed Following Each Reference Point 5–5
5.3.2 Describing the Categories for the Interval Histogram 5–5
5.3.3 Overflow and Underflow Count 5–6
5.3.4 Frequency Histogram 5–6

Page

5.4 How to Call the RHISTI Subroutine 5–7
5.5 Input and Output — Using the Subroutine 5–9
5.6 Modifying the Subroutine — Using Options 5–10
5.6.1 EIS (Extended Instruction Set) 5–10
5.6.2 EAE (Extended Arithmetic Element) 5–10
5.6.3 DPR$ (Double-Precision Integers) 5–10
5.7 Examples of Input/Output Variation Using the RHISTI Subroutine 5–11

Chapter 6 The Fast Fourier Transform (FFT) Subroutine

6.1 An Introduction to Fourier Transforms 6–1
6.1.1 Mathematical Definition of the Fourier Transform (CFT) . . . 6–2
6.1.2 Mathematical Definition of the Discrete Fourier Transform (DFT) 6–2
6.2 Comparing the Continuous and Discrete Fourier Transform 6–3
6.2.1 Comparison of FFT and CFT Input 6–4
6.2.2 Comparison of FFT and CFT Output 6–5
6.2.3 Similarities and Discrepancies between FFT and CFT Results 6–7
6.3 Scaling the Results of the FFT Subroutine 6–8
6.3.1 Internal Subroutine Scaling Procedure 6–9
6.3.2 Relational Scaling Procedure 6–9
6.4 Useful Properties of the FFT 6–12
6.5 How to Call the FFT Subroutine 6–13
6.5.1 Interpreting the Results of the Output Arrays 6–14
6.6 Modifying the Subroutine — Using Options 6–16
6.6.1 EIS (Extended Instruction Set) 6–16
6.6.2 EAE (Extended Arithmetic Element) 6–16
6.6.3 F.MAXN (Maximum I/O Array Size) 6–16
6.7 Example Using the FFT Subroutine 6–17

Chapter 7 The Phase Angle and Amplitude Spectra (PHAMPL) Subroutine

7.1 How to Call PHAMPL 7–2
7.2 Other Routines Used by PHAMPL 7–2
7.3 Modifying the Subroutine — Using Options 7–2
7.3.1 EIS (Extended Instruction Set) 7–3
7.3.2 EAE (Extended Arithmetic Element) 7–3
7.3.3 F4P$ (FORTRAN 77 Compiler) 7–3
7.4 Examples Using the PHAMPL Subroutine 7–3

Chapter 8 The Power Spectrum (POWRSP) Subroutine

8.1 How to Call POWRSP 8–1
8.2 Modifying the Subroutine — Using Options 8–2

Page

8.2.1 EIS (Extended Instruction Set) 8–2
8.2.2 EAE (Extended Arithmetic Element) 8–2
8.3 Examples Using the POWRSP Subroutine 8–2

Chapter 9 The Correlation Function (CORREL) Subroutine

9.1 Using the Correlation Function Subroutine 9–2
9.2 Discrete Evaluation of CORREL 9–3
9.3 Calculating the Correlation Coefficients 9–4
9.4 How to Call CORREL 9–5
9.5 Other Routines Used 9–6
9.6 Modifying the Subroutine — Using Options 9–7
9.6.1 EIS (Extended Instruction Set) 9–7
9.6.2 EAE (Extended Arithmetic Element) 9–7
9.6.3 FFT Options 9–7
9.7 Examples Using the CORREL Subroutine 9–7

Appendix A Installing, Verifying, and Using LSP Under RT–11

A.1 Installation Requirements A–1
A.2 Installing the Laboratory Subroutines Software A–2
A.2.1 Copying the Distribution Volume A–2
A.2.1.1 Copying with Three or More Mass Storage Devices A–3
A.2.1.2 Copying only Two Mass Storage Devices A–4
A.2.2 File Protection A–5
A.2.3 Making Corrections A–5
A.2.4 Selecting the Form of Subroutine to Use A–5
A.2.4.1 Using Distributed Object Files A–5
A.2.4.2 Creating Customized Object Files — LSPMAK, the Interactive Build Procedure A–6
A.3 Verifying the Laboratory Subroutines Software A–9
A.3.1 Verifying the Distributed LSP Object Files A–9
A.3.2 Verifying the Customized Object Files A–10
A.4 Storing the Laboratory Subroutines A–10
A.5 Creating a Program that Calls the Laboratory Subroutines A–10
A.6 Using Libraries A–11

Appendix B Installing, Verifying, and Using LSP Under RSX–11M/M–PLUS

B.1 Installation Requirements B–1
B.2 Installing the Laboratory Subroutines Software B–2
B.2.1 Copying the Distribution Volume B–2
B.2.1.1 Copying a FILES–11 Distribution Volume with Three or More Mass Storage Devices B–3
B.2.1.2 Copying a FILES–11 Distribution Volume with Only Two Mass Storage Devices B–5
B.2.1.3 Copying a DOS–11 Distribution Volume B–7

Page

B.2.2 Making Corrections B–10
B.2.3 Selecting the Form of Subroutine to Use B–10
B.2.3.1 Using Distributed Object Files B–10
B.2.3.2 Creating Customized Object Files — LSPMAK, the Interactive Build Procedure. B–10
B.3 Verifying the Laboratory Subroutines Software. B–14
B.3.1 Verifying the Distributed LSP Object Files B–14
B.3.2 Verifying the Customized Object Files B–15
B.4 Storing the Laboratory Subroutines B–15
B.5 Creating a Program that Calls the Laboratory Subroutines B–15
B.6 Using Libraries B–17

Appendix C Sample of the Interactive Build Procedure for RT–11, LSPMAK.SAV

Appendix D Sample of the Interactive Build Procedure for RSX–11M, LSPMAK.TSK

Figures

2–1 Flow of the PEAK Subroutine 2–1
2–2 Calculation of True Peak Width 2–5
2–3 Calculation of Estimated Peak Width 2–5
2–4 Flow Chart for Peak-Processing; Initialization, Data Averaging, and Application of Digital Filter 2–9
2–5 Flow Chart for Peak-Processing; Calculation of Peak Width and Search for Baseline 2–10
2–6 Flow Chart for Peak-Processing; Area Calculation 2–11
2–7 Flow Chart for Peak-Processing; Determining the Baseline 2–12
2–8 NEXTPT Subroutine — Peak-Processing 2–13
2–9 RITOUT Subroutine — Peak-Processing 2–14
2–10 INPTR, INLAST, and NPEAKS Point to Slots 2–21
2–11 Actual Plot of the Input Data 2–24
3–1 Envelopes of Data (may contain more than one peak). 3–1
3–2 Flow Chart for Envelope Processing; Data Entry 3–6
3–3 Flow Chart for Envelope Processing; Initialization, Calculation of Peak Width, and Finding End of Envelope. 3–7
3–4 Flow Chart for Envelope Processing; Count of Reject Points and Reading Additional Values where Envelope Begins 3–8
3–5 Flow Chart for Envelope Processing; New Minimum and Crest Detection 3–9
3–6 Flow Chart for Envelope Processing; New Maximum; Peak Begins at Valley 3–10
3–7 NEXTPT Subroutine — Envelope Processing 3–11
3–8 RITOUT Subroutine — Envelope Processing 3–12
3–9 INPTR, NPEAKS, and INLAST Point to Slots 3–21
3–10 Actual Plot of Input Data, Showing Threshold 3–23
3–11 Plot of Input Data, Showing New Threshold Value 3–30
3–12 Plot of Input Data, Showing Threshold Value from Example 2 and Assumed Baseline Offset 3–34

		Page
4–1	Interrelation between DATA/CATEGORY and INTEGER/INTERVAL Concepts	4–2
4–2	Relation between FORTRAN Integers and Unsigned Binary Values	4–4
5–1	Interrelation between DATA/CATEGORY and INTEGER/INTERVAL Concepts	5–3
5–2	Relation between FORTRAN Integers and Unsigned Binary Values	5–4
6–1	The Fourier Transform	6–1
6–2	The Relationship Between FFT Input and CFT Input	6–5
6–3	Output from the FFT	6–6
6–4	G(n · df) Returned by the FFT	6–7
6–5	Total Range of the Frequency Domain	6–14
6–6	Five Elements in the IREAL Output Array	6–16

Tables

2–1	Switch Settings for Significant Events in Peak Definition	2–7
2–2	Definition of Symbols	2–8
2–3	Definition of Peak Events	2–15
3–1	Definitions of Symbols Used	3–5
3–2a	Envelope-Processing Algorithm	3–14
3–2b	How to Compile and Prepare Data for Envelope-Processing Subroutine	3–16

Index

Preface

MANUAL OBJECTIVES AND READER ASSUMPTIONS

The *Laboratory Subroutines Programmer's Reference Manual* describes the Laboratory Subroutines Package (LSP), a set of eight data-processing subroutines to be used in a laboratory environment.

To use this manual, you should be a laboratory-oriented programmer familiar with the FORTRAN IV or FORTRAN 77 programming language. You should also be familiar with either the RT–11 operating system (Single Job (SJ) or Foreground/Background (F/B) monitor) or the RSX–11M or RSX–11M–PLUS operating system. In addition, you must understand the mathematics involved in explaining the subroutine algorithms. However, although the laboratory subroutines are written in the MACRO programming language, you do not need to be familiar with MACRO to use them.

MANUAL STRUCTURE

The *Laboratory Subroutines Programmer's Reference Manual* contains nine chapters and four appendixes.

Chapter 1 introduces the Laboratory Subroutines software and explains how to use the manual.

Chapter 2 describes the peak-processing subroutine, PEAK.

Chapter 3 details the envelope-processing subroutine, NVELOP.

Chapter 4 discusses the interval histogramming subroutine, HISTI.

Chapter 5 describes the interval histogramming with reference points subroutine, RHISTI.

Chapter 6 details the fast Fourier transform subroutine, FFT.

Chapter 7 discusses the phase angle and amplitude spectra subroutine, PHAMPL.

Chapter 8 describes the power spectrum subroutine, POWRSP.

Chapter 9 details the correlation function subroutine, CORREL.

Appendix A explains how to install, verify, and use LSP under the RT–11 operating system.

Appendix B explains how to install, verify, and use LSP under the RSX–11M operating system.

Appendix C contains an example of the interactive build procedure for RT–11, LSPMAK.SAV. The appendix also shows the output from the procedure.

Appendix D contains an example of the interactive build procedure for RSX–11M, LSPMAK.TSK. The appendix also shows the output from the procedure.

RELATED DOCUMENTS

The following documents provide more information about the RT–11, RSX–11M, or RSX–11M–PLUS operating systems and the FORTRAN IV and FORTRAN 77 programming languages:

Reference	Order Number
PDP–11 FORTRAN 77 User's Guide	AA–1884D–TC
IAS/RSX–11 FORTRAN IV User's Guide	AA–1936E–TC
PDP–11 FORTRAN Language Reference Manual	AA–1855D–TC
RSX–11M/M–PLUS MCR Operations Manual	AA–H263A–TC
RSX–11M/M–PLUS Task Builder Manual	AA–H266A–TC
RSX–11 Utilities Manual	AA–H268A–TC
RT–11 Programmer's Reference Manual	AA–H378A–TC
RT–11/RSTS/E FORTRAN IV User's Guide	AA–5749B–TC
RT–11 Software Support Manual	AA–H379A–TC
RT–11 System User's Guide	AA–5279B–TC
RT–11 System Message Manual	AA–5284C–TC

DOCUMENTATION CONVENTIONS

The following conventions are used in this manual.

- In programming examples, all information the computer prints appears in black. All commands and responses you type appear in red.
- RET means you must press the RETURN key on your terminal.

- You produce certain characters by typing a combination of keys together. For example, hold down the CTRL key and type the letter C to produce the CTRL C character. Combinations such as this are represented by CTRL/C.

- Many commands in this manual contain the expression dvn:. When you execute the commands, specify a device and unit number in place of dvn:. If you do not include a unit number, the system uses unit 0 as a default.

  For a list of devices and their abbreviations, see Chapter 3 of the *RT–11 System User's Guide* or Chapter 2 of the *RSX–11M/M–PLUS MCR Operations Manual*.

- In descriptions of commands or file names, capital letters represent actual commands, file names, or file types. You must type these exactly as they appear. Lower case letters mean that you must supply a name.

- The term RSX–11M/M–PLUS means either or both the RSX–11M or RSX–11M–PLUS operating systems.

- Brackets represent optional elements in a specification. When you use an option, do not type the brackets in the command line.

NOTE

Under RSX–11M/RSX–11M–PLUS, brackets are also a part of the User File Directory (UFD) portion of file specifications, that is [group, member]. When you type this portion of a file specification, brackets are required syntax elements. You must type the brackets in the command line.

Chapter 1
Introduction to the Laboratory Subroutines

The *Laboratory Subroutines Programmer's Reference Manual* accompanies the Laboratory Subroutines Package (LSP), a set of eight laboratory, data-processing subroutines.

This manual describes the eight subroutines and explains how to use them with your FORTRAN programs. It contains nine chapters and four appendixes.

You can use any chapter in this manual independently. Each chapter is a self-contained document that deals with all the essential aspects of a particular subroutine. Each chapter outlines the algorithm and logic of a subroutine. Each chapter describes the subroutine's FORTRAN call, arguments, and the options that can be used with the subroutine. Each chapter has a special reference divider that precedes the chapter and that summarizes the information contained in the chapter.

In addition, each chapter presents one or more example programs that use the subroutines. Where necessary, certain chapters also present flowcharts and glossaries to explain subroutine operation in greater detail.

The appendixes tell you how to install, verify, and use the Laboratory Subroutines with your FORTRAN programs under the RT–11, RSX–11M/M–PLUS operating systems.

1.1 The Laboratory Subroutines Package

The Laboratory Subroutines Package consists of eight subroutines that you can call from any FORTRAN IV program running under the RT–11 operating system and from any FORTRAN IV or FORTRAN 77 program running under the RSX–11M operating system. The eight subroutines perform a variety of standard tasks commonly encountered in the laboratory.

NOTE

FORTRAN 77 V4.0 is the next version of FORTRAN IV–PLUS V3.0. FORTRAN 77 is so named because it adheres to the 1977 ANSI subset standard for FORTRAN programming languages. Because FORTRAN 77 is compatible with FORTRAN IV–PLUS V3.0, your FORTRAN IV–PLUS V3.0 programs can run under FORTRAN 77.

The eight LSP subroutines are:

1. The peak-processing subroutine, PEAK, detects peaks in waveform data.
2. The envelope-processing subroutine, NVELOP, detects peaks in discontinuous segments (envelopes) of waveform data.
3. The interval histogramming subroutine, HISTI, counts the number of elements in a data stream that fall into one or more predefined categories.
4. The interval histogramming with reference points subroutine, RHISTI, counts the number of elements in a data stream marked with reference points that fall into one or more numerical intervals.
5. The fast Fourier transform subroutine, FFT, numerically approximates the analytical or continuous Fourier transform.
6. The phase angle and amplitude spectra subroutine, PHAMPL, converts complex numerical values to phase angles and amplitudes.
7. The power spectrum subroutine, POWRSP, determines the power spectrum (the relationship between power and signal frequency) in a set of Fourier coefficients.
8. The correlation function subroutine, CORREL, provides a discrete method of performing the correlation function.

Each subroutine has hardware and software options you can use to extend the capabilities of the subroutine. The chapter dealing with each subroutine explains which options the subroutine can use, what the options do, and when and how you should use the options. The Laboratory Subroutines Package also contains a simple, interactive procedure that lets you build the subroutines in order to create a customized version of your LSP software. "Building" consists of assembling the subroutines with the options you choose enabled. The procedure does the following things:

1. Lets you select which subroutines you want to assemble and which options you want to use.
2. Creates a file that sets the switches to enable options you choose.
3. Creates a file that builds each subroutine you requested with the options you chose enabled.

4. Creates a file that tests the subroutines you built to make sure your software works properly.

All of the laboratory subroutines are written in MACRO assembly language, but you do not have to know MACRO to use them. You can invoke any of the laboratory subroutines with a FORTRAN call statement as you would invoke any FORTRAN subroutine.

The specific FORTRAN call format for a Laboratory Subroutine is outlined in the chapter describing that subroutine. In all calls you make to any of the Laboratory Subroutines, make sure you state all of the required arguments explicitly. There are no default values for any of the arguments. If you omit an argument, accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results and no data is processed.

1.2 The Laboratory Subroutines Package Distribution Kit

The distribution kit for the Laboratory Subroutines Package consists of a mass-storage volume containing the Laboratory Subroutines Package, this manual, a software product description (SPD), and other forms.

The subroutines are supplied to you on the distribution volume as both source files and object files. If you decide to enable options, the interactive build procedure creates customized versions of the subroutines from the source files. If you decide not to enable any options, you can use the procedure to build the subroutines without any options, or you can, in some cases, use the distributed object files by linking or task building them to your FORTRAN programs. The object files were built with no switches set, and therefore, with no options enabled. To determine if you can use the distributed object files, see Appendix A if you are using RT–11, or Appendix B if you are using RSX–11M.

In addition, the distribution volume contains example programs that call the subroutines. Example programs exist on the distribution volume as FORTRAN source files. Example program file names have the form:

EXnsub.FOR or EXnsub.FTN

where: n stands for the number of the subroutine example program

sub stands for the first three letters of the subroutine's file name

For instance:

EX3RHI.FOR or EX3RHI.FTN

is the third example program for the RHISTI subroutine.

Text of the example programs appears in each chapter along with the terminal output that results when you run the programs.

Files on the distribution volume are as follows:

Peak-processing subroutine files:

For RT–11	For RSX–11M
FPEAK.MAC	FPEAK.MAC
FPEAK.OBJ	FPEAK.OBJ
EX1FPE.FOR	EX1FPE.FTN
EX2FPE.FOR	EX2FPE.FTN
EX3FPE.FOR	EX3FPE.FTN
EX4FPE.FOR	EX4FPE.FTN

Envelope-processing subroutine files:

For RT–11	For RSX–11M
FNVLOP.MAC	FNVLOP.MAC
FNVLOP.OBJ	FNVLOP.OBJ
EX1FNV.FOR	EX1FNV.FTN
EX2FNV.FOR	EX1FNV.FTN
EX3FNV.FOR	EX3FNV.FTN

Interval histogramming subroutine files:

For RT–11	For RSX–11M
HISTI.MAC	HISTI.MAC
HISTI.OBJ	HISTI.OBJ
EX1HIS.FOR	EX1HIS.FTN
EX2HIS.FOR	EX2HIS.FTN
EX3HIS.FOR	EX3HIS.FTN
EX4HIS.FOR	EX4HIS.FTN

Interval histogramming with reference points subroutine files:

For RT–11	For RSX–11M
RHISTI.MAC	RHISTI.MAC
RHISTI.OBJ	RHISTI.OBJ
EX1RHI.FOR	EX1RHI.FTN
EX2RHI.FOR	EX2RHI.FTN
EX3RHI.FOR	EX3RHI.FTN

Fast Fourier transform subroutine files:

For RT–11	For RSX–11M
F4FFT.MAC	F4FFT.MAC
F4FFT.OBJ	F4FFT.OBJ
EX1F4F.FOR	EX1F4F.FTN

Phase angle and amplitude spectra subroutine files:

For RT–11	For RSX–11M
PHAMPL.MAC	PHAMPL.MAC
PHAMPL.OBJ	PHAMPL.OBJ
EX1PHA.FOR	EX1PHA.FTN
EX2PHA.FOR	EX2PHA.FTN

Power spectrum subroutine files:

For RT–11	For RSX–11M
POWRSP.MAC	POWRSP.MAC
POWRSP.OBJ	POWRSP.OBJ
EX1POW.FOR	EX1POW.FTN
EX2POW.FOR	EX2POW.FTN

Correlation function files:

For RT–11	For RSX–11M
CORREL.MAC	CORREL.MAC
CORREL.OBJ	CORREL.OBJ
EX1COR.FOR	EX1COR.FTN

Verification procedure files:

For RT–11	For RSX–11M
LSPVER.COM	LSPVER.CMD

Interactive build procedure files:

For RT–11	For RSX–11M
LSPMAK.SAV	LSPMAK.TSK

Files generated by the interactive build procedure:

For RT–11	For RSX–11M
LSPCND.MAC	LSPCND.MAC
LSPBLD.COM	LSPBLD.CMD
LSPVER.COM	LSPVER.CMD

For instructions on installing, verifying, and using the Laboratory Subroutines, see Appendix A if you use RT–11 or Appendix B if you use RSX–11M/M–PLUS. Each appendix includes a description of the interactive build procedure, LSPMAK, and instructions for using it.

PEAK–PROCESSING (PEAK) SUBROUTINE

FORMAT:

CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)

Where:

ITABLE is a 68-element integer array (a 79-element array if AUTOG$ or DPP$ is enabled).

ITABLE(1)	=	original point density
ITABLE(2)	=	baseline test factor
ITABLE(3)	=	gate parameter
ITABLE(4)	=	minimum increase indicator
ITABLE(5)	=	output data type
ITABLE(6)	=	error indicator
ITABLE(7)	=	reentry pointer
ITABLE(8)	=	input type indicator (if AUTOG$ or DPP$ is enabled)

INPUT is an integer array containing input data.

INLAST is an integer variable specifying subscript of last data element in INPUT.

INPTR is an integer variable specifying subscript of last element in INPUT processed.

OUTPUT is a double-subscripted array used to store output data.

OUTPUT(1,N)	=	area, Nth peak
OUTPUT(2,N)	=	crest height, Nth peak
OUTPUT(3,N)	=	crest time, Nth peak
OUTPUT(4,N)	=	leading minimum height, Nth peak
OUTPUT(5,N)	=	leading minimum time, Nth peak
OUTPUT(6,N)	=	width, Nth peak
OUTPUT(7,N)	=	trailing minimum height, Nth peak
OUTPUT(8,N)	=	trailing minimum time, Nth peak
OUTPUT(9,N)	=	ending indicator, Nth peak
OUTPUT(10,N)	=	current number of input points averaged

IDIMO is an integer variable specifying number of peak data sets that can be stored in OUTPUT.

NPEAKS is an integer variable specifying number of peak data sets already stored in OUTPUT.

FILE NAMES:

FPEAK.MAC (source file); FPEAK.OBJ (object file)

OPTIONS:

- EIS (Extended Instruction Set — KE11–E)
- EAE (Extended Arithmetic Element — KE11)
- AUTOG$ (Autogaining)
- DPP$ (Double Precision Integer Input)
- NOFLT$ (No Filter)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If you use the digital filter and enable the following options:

	NONE	EIS	EAE
NONE	1033	905	946
AUTOG$	1237	1097	1150
DPP$	1219	1079	1132

If you enable the No Filter (NOFLT$) option and the following options:

	NONE	EIS	EAE
NONE	957	831	870
AUTOG$	1118	982	1031
DPP$	1100	964	1013

TYPICAL EXECUTION SPEED:

With PDP–11/34 and EIS enabled: 1000 Points/second.

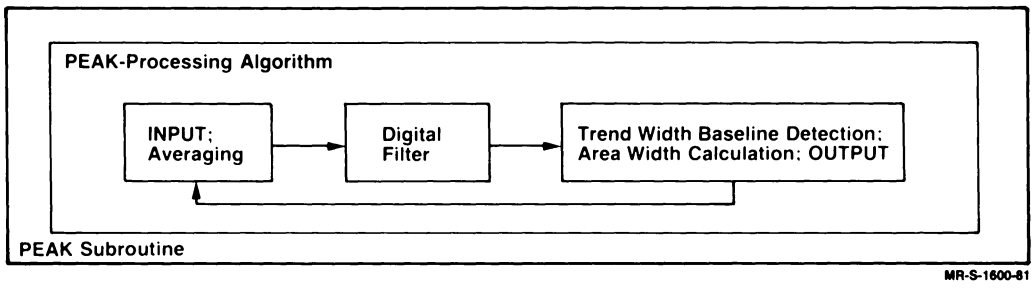
With PDP–11/03 and EIS enabled: 450 Points/second.

Chapter 2
The Peak-Processing (PEAK) Subroutine

The peak-processing subroutine 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.

Input to the subroutine 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 2–1). You can change specified algorithm parameters to enhance detectability of directional trends and baselines for a given set of data.

Figure 2–1: Flow of the PEAK Subroutine



Output from the subroutine 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 subroutine further reports how each peak ends — on a baseline or at a valley.

2.1 Definition of Basic Terms and Conventions

It is important to understand how some of the terms and conventions describing the PEAK subroutine are used throughout this chapter.

- The term data (input) stream describes all values presented to the subroutine 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 that direction exceeds the number in the opposite direction.

2.2 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 subroutine 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 subroutine parameters are aimed at removing these distortions so that only the real (dominant) trends in the data are visible.

2.2.1 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 subroutine 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. Obviously you will want to 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 subroutine 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.

2.2.2 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.

$$Y_0 = \left[-(Y_{-3} + Y_{+3}) + 4(Y_{-2} + Y_{+2}) + 11(Y_{-1} + Y_{+1}) + 14Y_0\right] / 42$$

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, $Y_{-1} = Y_0$. Prerequisites 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.
- 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.

2.2.3 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

[1] When the half-width-at-half-height measurement of a peak exceeds 25, the algorithm doubles the number of points averaged.

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 change "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, and conversely, if the height is greater than the sum of the local 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 to a very low value. Between crests the local minimum and maximum may 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.

2.2.4 Calculation of Area under the Peak

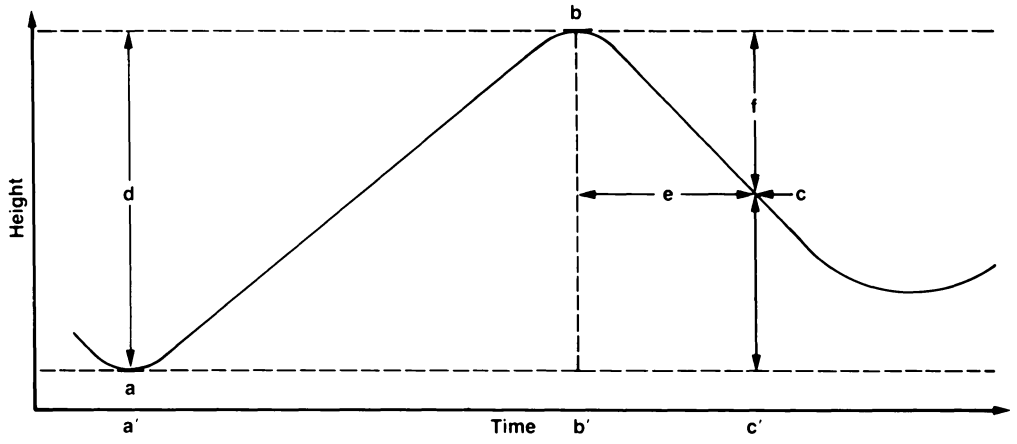
Two peak characteristics that are not entirely dependent on points of dominant trend change are area and width. The area under the peak, or integral, is calculated by taking the sum of the area increments corresponding to each filtered point and half the area increment at the first and last points of a peak. The area increment at each filtered point is the product of its height times the number of points currently being averaged.

2.2.5 Algorithm Definition of the Width of a Peak

Calculation of the 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 2–2).

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*, and it is half the difference between time of crest occurrence and time at which the increasing trend is observed (Figure 2–3).

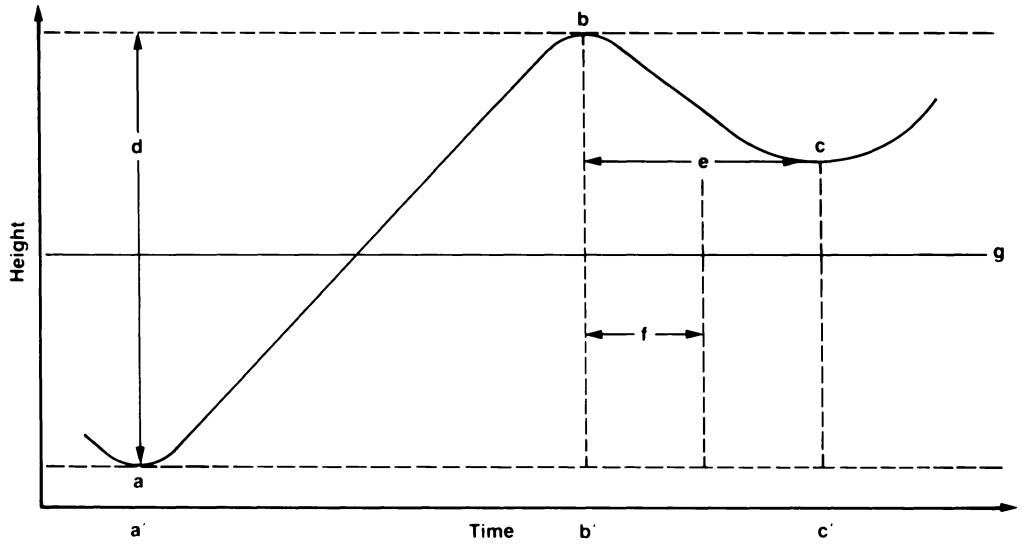
Figure 2–2: Calculation of True Peak Width



a = Leading Minimum Height
b = Crest Height
c = Point Whose Value is ½(a + b)
d = b–a
e = Peak Width = Time c'–Time b'
f = d/2

MR-S-1601-81

Figure 2–3: 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-S-1602-81

2.2.6 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 it does not occur, we say that the peak has ended at a valley. The problem thus becomes one of detecting when, or if, the baseline is reached.

Normally we assume that when the algorithm is initiated, input starts from a quiescent state; therefore we may 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, the slopes of successive tangent lines from the current baseline point to each new filtered point are calculated. 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 remain 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]

[1] Data taken during this period indicates that there is no peak-producing activity.

2.2.7 Flow Charts for the PEAK Subroutine

The series of flow charts presented as Figures 2–4 through 2–9 gives detailed logic for the PEAK subroutine. Supplementary information is presented in Tables 2–1 through 2–3. Table 2–1 lists the combinations of switch/indicator settings that characterize significant events during peak detection. Table 2–2 defines the symbols used in the flow charts and accompanying explanation. Table 2–3 reviews and summarizes flow-charted events as they apply to three possible peak configurations:

- A peak starting on the baseline and ending on a "new" baseline
- 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 2–1: Switch Settings for Significant Events in Peak Definition

Significant Event Switch BS	Current Trend Indicators		What is Happening with Relation to Peak Processing
	Decreasing DI	Increasing II	
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 conditions)
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 2–2: Definition of Symbols

BH	Current Baseline height
BHT	Time of current baseline height
BS	Baseline switch: 0 Peak starts on baseline 1 Looking for width 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: IPD = PD if number of points is to be increased 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
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]

(Continued on next page)

Table 2–2: Definition of Symbols (Cont.)

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 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 2–4: Flow Chart for Peak-Processing; Initialization, Data Averaging, and Application of Digital Filter

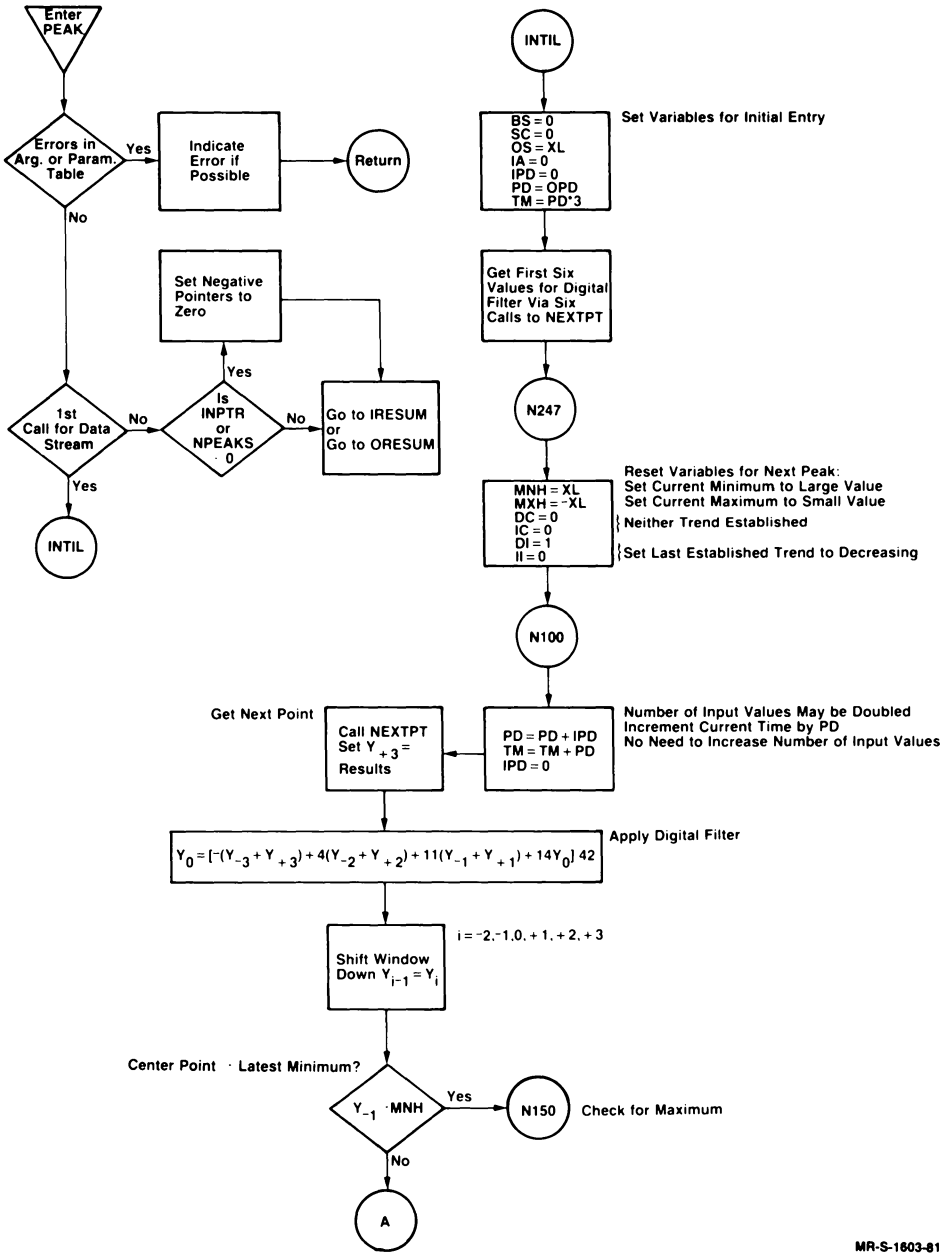


Figure 2–5: Flow Chart for Peak-Processing; Calculation of Peak Width and Search for Baseline

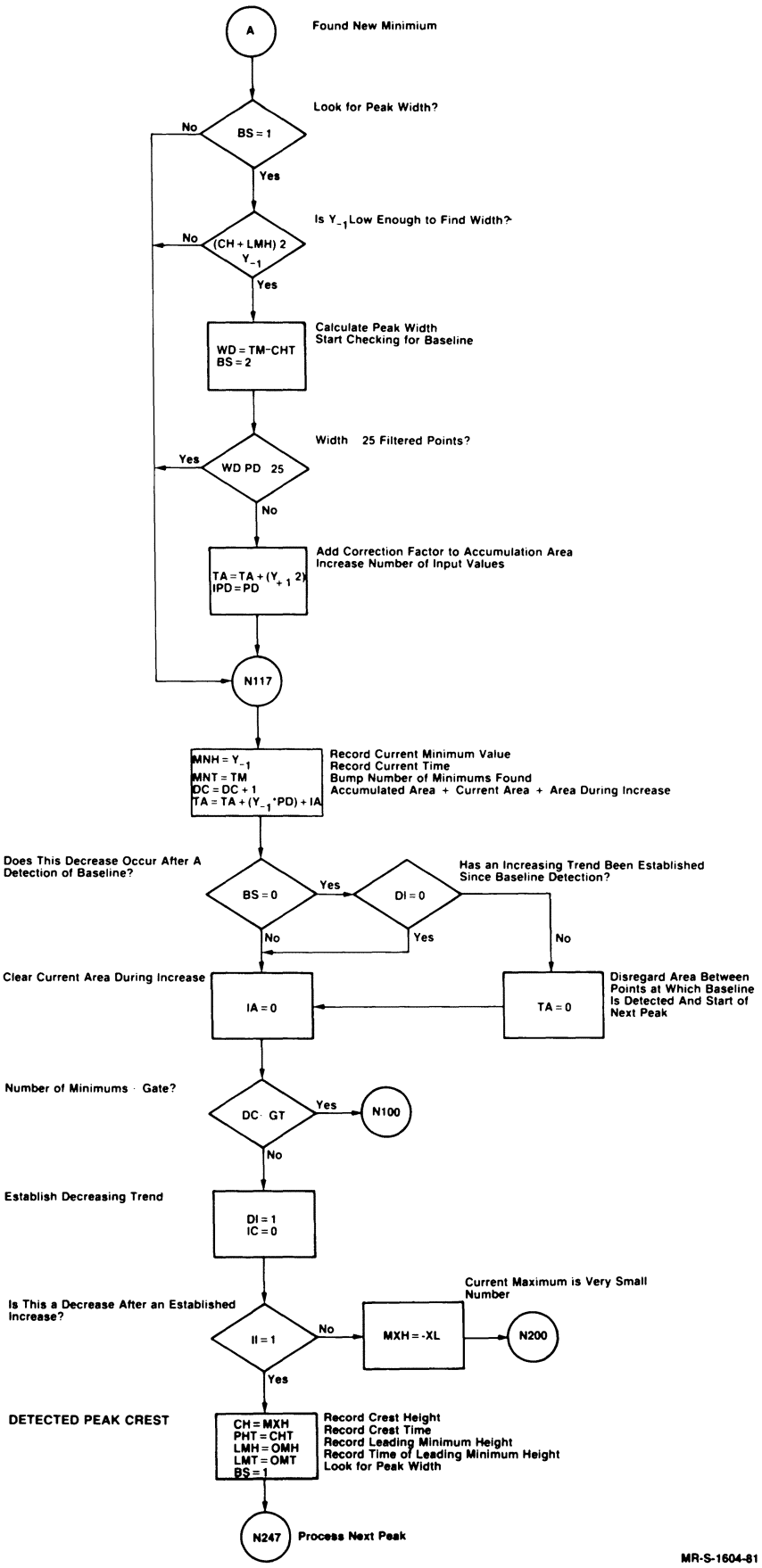


Figure 2–6: Flow Chart for Peak-Processing; Area Calculation

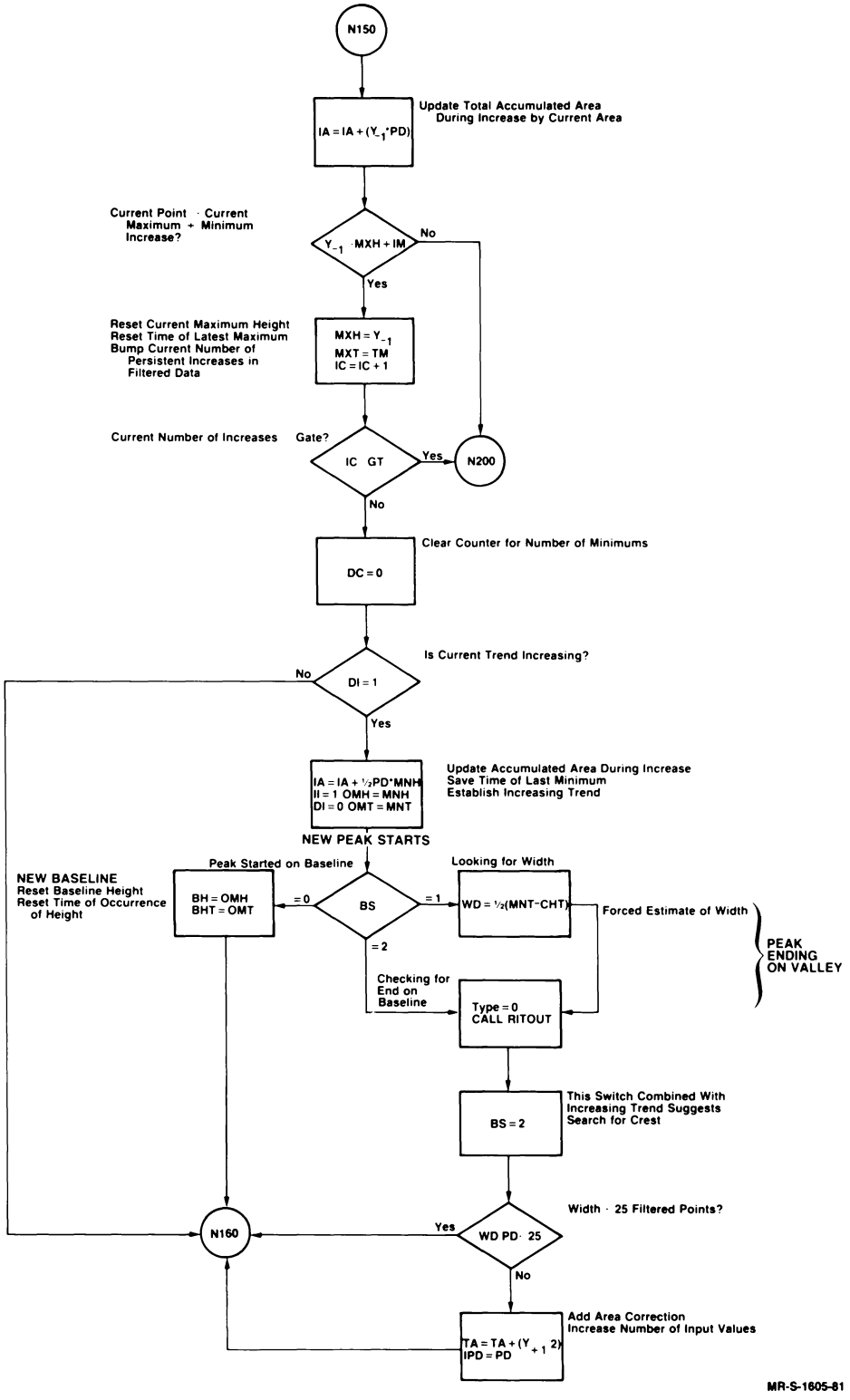


Figure 2–7: Flow Chart for Peak-Processing; Determining the Baseline

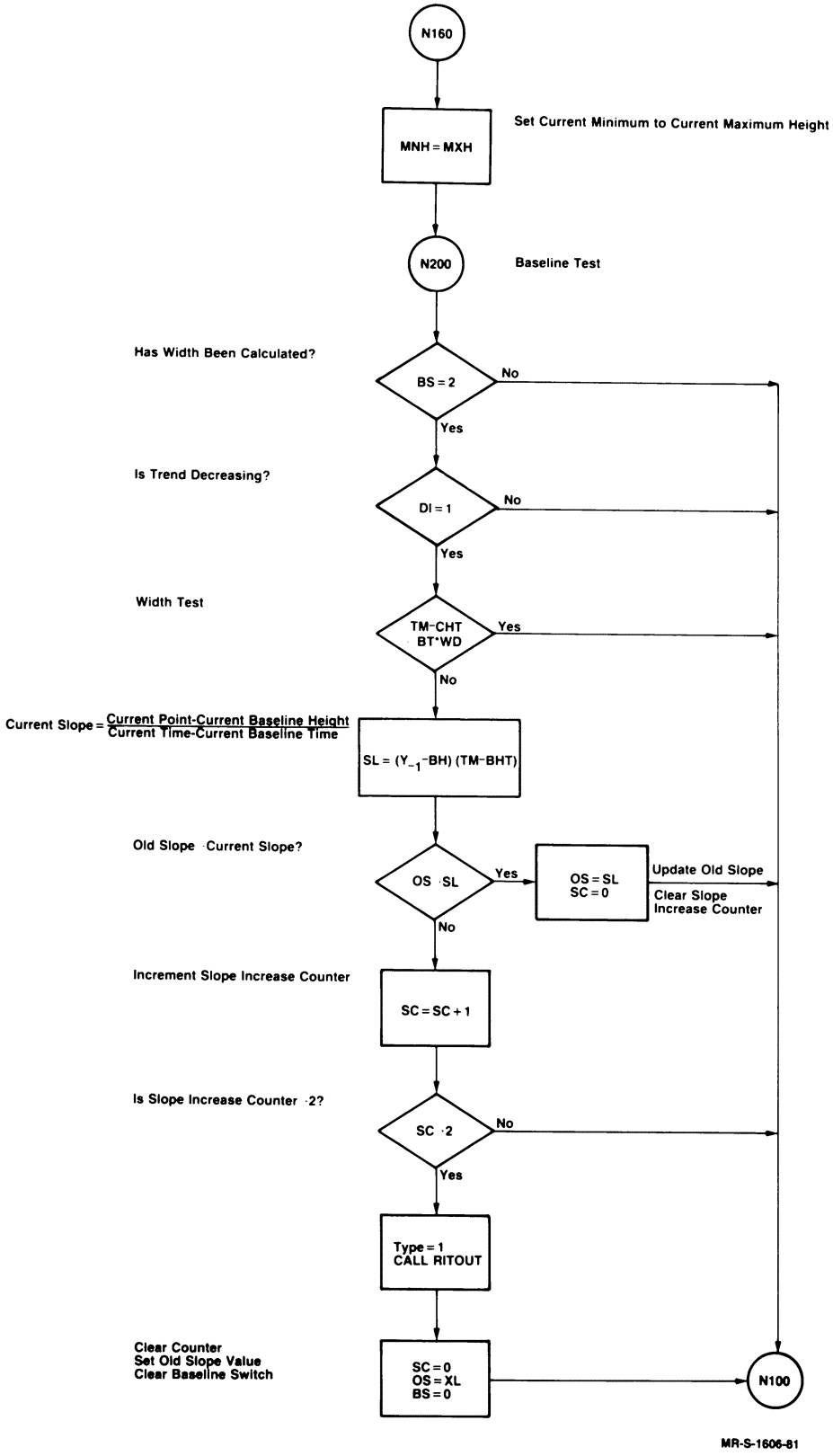


Figure 2–8: NEXTPT Subroutine — Peak-Processing

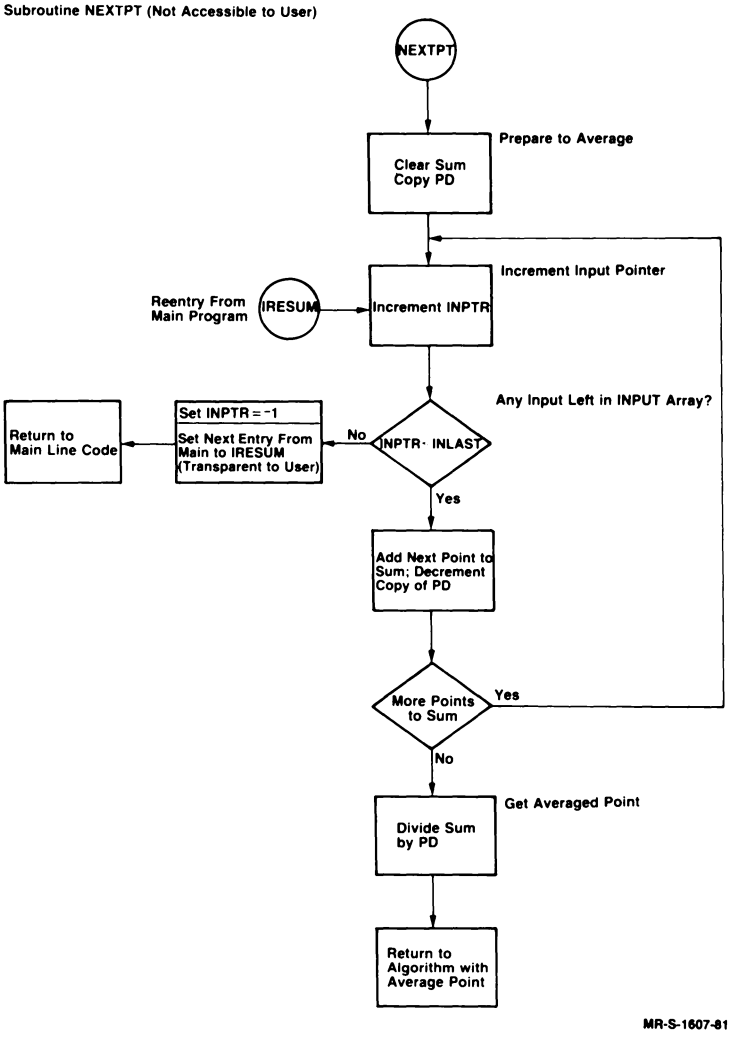


Figure 2–9: RITOUT Subroutine — Peak-Processing

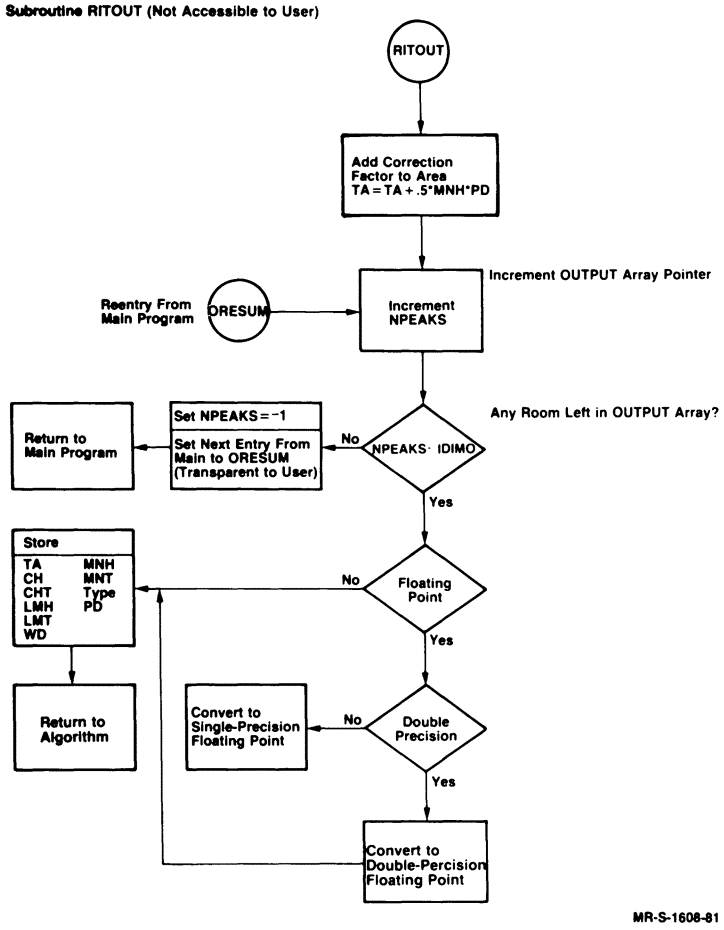
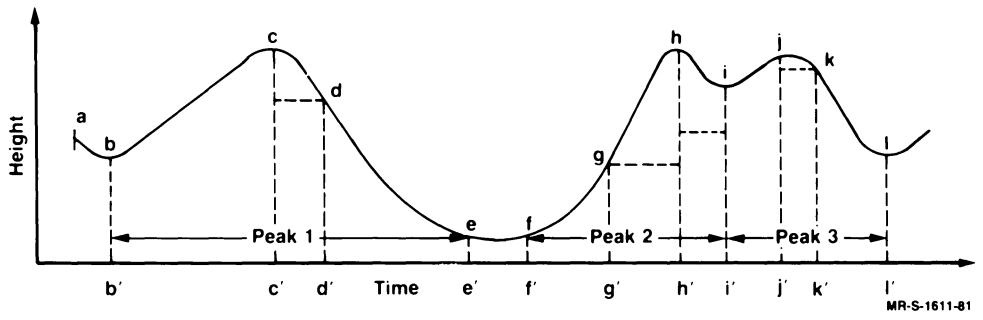


Table 2–3: Definition of Peak Events



Point/Section of Curve	Description	Flow Chart Reference(s)
a	Input begins	Flowchart begins
a-b	Decreasing trend in data after baseline detection	BS = 0,DI = 1,II = 0 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 OMT = b' BHT = b'
b-c	Increasing trend in data; change in established trend will indicate crest detection	BS = 0,DI = 0,II = 1 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 LMT = OMT CHT = c'
c-d	Decreasing trend in data after crest detection and before width calculation	BS = 1,DI = 1,II = 0 DC>GT,IC<GT
d	Point where width is calculated; d = (b + c)/2	WD = d'-c'
d-e	Decreasing trend in data after width is calculated and before baseline is detected	BS = 2,DI = 1,II = 0 DC>GT,IC<GT
e	Baseline detected; Peak 1 ends at this point, which is recorded as trailing minimum	MNH = e MNT = e' 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 DC>GT,IC<GT

(Continued on next page)

Table 2–3: Definition of Peak Events (Cont.)

Point/Section of Curve	Description	Flow Chart Reference(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 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; 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 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 LMT = OMT CHT = h′
h-i	Decreasing trend in data after crest detection and before width calculation (see c-d)	BS = 1,DI = 1,II = 0 DC>GT,IC<GT
i	Increasing trend established before width of Peak 2 calculated; forced estimation of width of peak 2 as (i′-n′)/2; Peak 2 ends at valley with i as trailing minimum for Peak 2; Peak 3 begins with i as leading minimum; baseline data remain unchanged	WD = (MNT–CHT)/2 MNH = i BH = f OMH = i MNT = i′ BHT = f′ OMT = i′ Type = 0
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 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 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 DC>GT,IC<GT
k	Point where width is calculated; k = (j + i)/2	WD = k′-j′

(Continued on next page)

Table 2–3: Definition of Peak Events (Cont.)

Point/Section of Curve	Description	Flow Chart Reference(s)
k-l	Decreasing trend in data after width is calculated and before baseline is detected (see d-e)	BS = 2,DI = 1,II = 0 DC>GT,IC<GT
l	Increasing trend established 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 MNT = 1′ BHT = f′ OMT = 1′ Type = 0
END OF PEAK 3/START OF PEAK 4		
	Peak 4 not shown in illustration	

2.3 How to Call the Peak-Processing Subroutine

The symbolic name for the peak-processing subroutine 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 may 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, either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results, and no data is processed. The arguments are described in the following discussion.

ITABLE is an integer array used to store intermediate results and other information required by the algorithm; its dimension is normally 68.[1] You must set the values of the following array elements to transmit variable parameters and other information to the subroutine.

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 so chosen that the number of averaged data points on the first peak is about 100.

[1] See Section 2.5 for the options you can use with the peak-processing subroutine.

ITABLE(2) The baseline test (BT) factor (Section 2.2.6); on a peak whose width is WD, baseline detection begins at time[2] 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 2.2.3. 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 2.2.3). In general, suggested values can range from 1 to 5.

ITABLE(5) The data type of the output array:

=0	output type is double-precision integer
=1	output type is single-precision floating-point
=-1	output type is double-precision floating-point

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 subroutine 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 to it the remaining area of the peak.

[2] Refer to conventions defined in Section 2.1 of this chapter.

ITABLE(7) This element must be set to zero before the initial call is made to the subroutine for each new stream of data. When the subroutine processes a data stream in "parts" (Section 2.4), it uses ITABLE(7) for reentry to process each subsequent part. This element should thus not be altered by a user until all parts have been processed.

ITABLE(8) Elements used exclusively by the subroutine while the data stream is being processed.

ITABLE(68)

INPUT is an integer array containing the raw data to be processed. Note that all data must be positive and in the range 0 through 32767($2^{15} - 1$).[1]

INLAST is an integer variable having the value of the subscript of the last element of INPUT containing data.

INPTR is an integer variable having the value of the subscript of the last element processed by PEAK. We 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 ten. The second dimension specifies the number of sets of peak data that can be stored by the algorithm while processing the input data and is defined by IDIMO.

The data type of the output array is optional and can be any of those specified by ITABLE(5).

The ten 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

[1] See Section 2.5 for the options you can use with the peak-processing subroutine.

OUTPUT(9,N) Indicator of how peak ended:

=0 ended on valley
=1 ended on a baseline

OUTPUT(10,N) *Current* number of input data points being averaged

IDIMO is an integer variable that transmits to the subroutine 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 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, NPEAKS should be set to zero.

You must set the value of this argument before calling the subroutine; however, the subroutine can change the value before returning.

NOTE

PEAK returns (assuming there are no errors) after either of the following events:

1. All input data elements have been processed.
2. 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 subroutine has set NPEAKS to the proper value for the next subroutine 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.

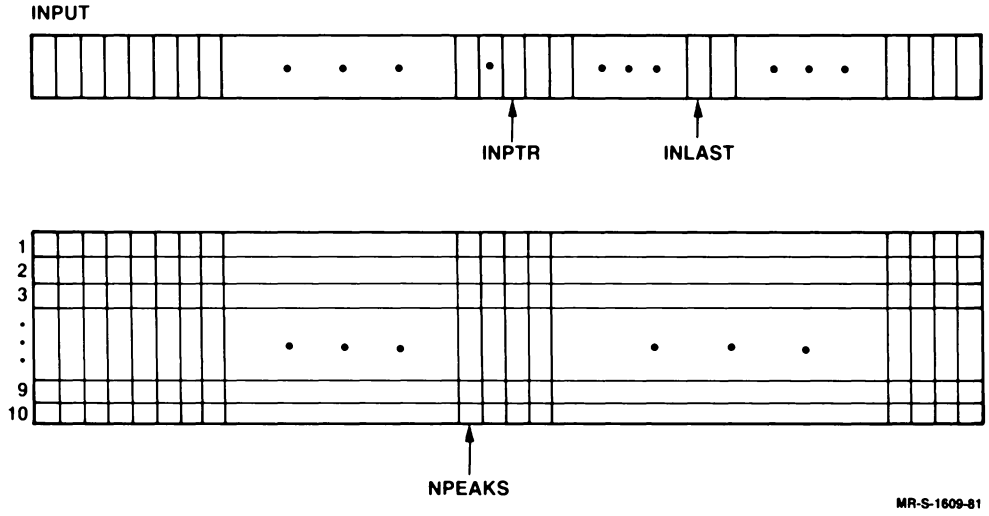
   If the subroutine is called again with either INPTR or NPEAKS equal to -1, the subroutine interprets the value as zero.

2.4 Using the Peak-Processing Subroutine

You can use several inherent features of the peak-processing subroutine 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 2.3). Let us 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 data, respectively (Figure 2–10). INLAST is a pointer to the last INPUT element containing data.

Figure 2–10: INPTR, INLAST, and NPEAKS Point to Slots



The subroutine 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 subroutine input and output. When you have large quantities of data to process, you need not allocate space for all data at once because the subroutine 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 subroutine, or it might be further processed only when the output buffer was filled, that is, NPEAKS = -1. You may choose the procedure that is most convenient for you.

Further flexibility is introduced by the fact that all arguments in the calling statement except ITABLE can be changed between successive calls to the subroutine 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 subroutine 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 subroutine. Imaginative use of the arguments in the subroutine call should make the subroutine functionally compatible with any application that uses the peak-processing algorithm.

2.5 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the peak-processing subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

2.5.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS (KE11–E) hardware or any other floating-point option available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 139 words if AUTOG$ or DPP$ is enabled or by approximately 129 words if AUTOG$ or DPP$ is not enabled.

2.5.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE (KE11) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 87 words.

2.5.3 AUTOG$ (Autogaining)

Enable the autogaining option only if you have a bipolar 12-bit A/D converter supplied by DIGITAL that has four, program-selectable gain values: 1, 4, 16, 64. The converter has optional features that allow the dynamic range to collapse under software control as the analog signal being monitored approaches zero volts. This feature effectively increases the resolution of the converted value as the analog signal becomes weaker.

Autogaining allows you to magnify the analog signal by a factor of one of these gain values, so that the resulting converted value becomes as significant as possible without causing an overflow. As an example, once a signal decreases (in absolute value) to a voltage 1/4 the maximum convertible voltage at a gain of 1, the gain is increased to 4. Thus the full digital range of the converter is implemented. As the real voltage of the analog signal continues to decrease in absolute value, the software increases the gain appropriately.

It should be pointed out, however, that while the resolution of the converted values increases, some analog signals differing exactly by magnifications of 4, 16, or 64 continue to be represented by the same 12-bit number. To distinguish values converted at different hardware gain settings, the autogaining software sets two additional bits in words that contain the converted values. These are bits 12 and 13, which indicate the gain value for a particular datum, as follows:

Bit 13	Bit 12	Gain Value Used for Conversion
0	0	1
0	1	4
1	0	16
1	1	64

When the peak-processing subroutine is assembled to process data collected using the autogaining algorithm, input data must still be positive; negative values are set to zero. A characteristic of the bipolar converter used in autogaining is to represent converted values in the range of zero to maximum positive voltage as 4000_8 to 7777_8, and those in the range of zero to maximum negative voltage as 3777_8 to 0000_8.

To deal with all converted values in their proper relation, the subroutine normalizes them before processing begins; all values are effectively represented as if sampled at a gain of 64. For example, all values sampled at a gain of 1 are multiplied by 64; those sampled at a gain of 4 are multiplied by 16, and so on. Consequently the resulting heights and areas are increased by a factor of 64 over those produced by sampling and processing the same analog signal with a gain of 1 (no gain).

Multiplying a 12-bit number by 64 results in an 18-bit number, which is a double-precision word. Consequently:

- The size of the required code increases by approximately 195 words.
- You must increase the dimension of ITABLE, the first argument in the call to PEAK, from 68 to 79.
- You must set another element in ITABLE:

  ITABLE(8) = 0 input data not autogained
  = 1 input data autogained

  By using this parameter, input that is not autogained can still be processed.
- You should select ITABLE(3) and ITABLE(4) with an awareness that all data processed can be as much as 64 times more sensitive than if collected with a gain of 1. For instance, if two values differ by one when collected at a gain of 1, they could be interpreted as differing by as much as 64 when their values have been normalized.

2.5.4 DPP$ (Double Precision Integers)

If the upper range of data points to be processed exceeds 32767 but is less than 33554431 decimal (2^{25}–1), you must enable this option. If you enable this option, all input data can be double precision integer, that is, the second argument in the FORTRAN call statement, INPUT, can be an INTEGER*4 array (or equivalent). Consequently:

- The size of the required code increases by approximately 195 words.
- You must increase the dimension of ITABLE, the first argument in the call to PEAK, from 68 to 79.
- You must set another element in ITABLE:

ITABLE(8) = 0 input data single-precision
= 1 input data double-precision

2.5.5 NOFLT$ (No Filter)

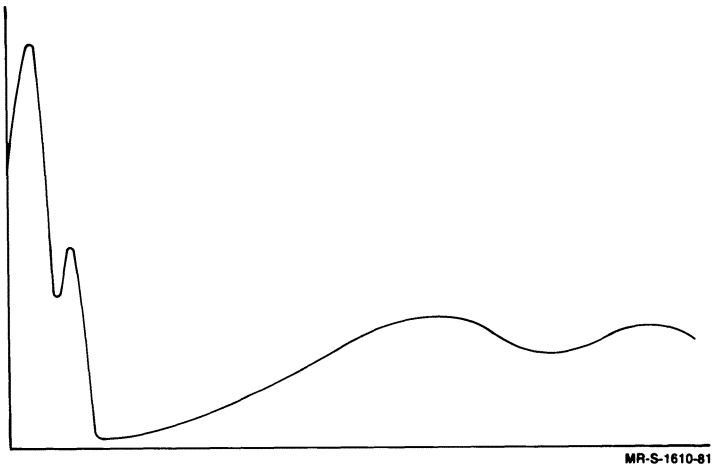
This option disables the software digital filter that the subroutine normally uses. Enable this option if you want to average and process data points without filtering them, or if you want to apply your own filter to the raw data before calling the PEAK subroutine.

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

2.6 Examples Using the PEAK Subroutine

The four examples presented here process the same waveform — the sum of four Gaussian curves — represented by the identical 1024 points. The variable input parameters are likewise the same for both examples, and the resulting output is printed on the terminal. Figure 2–11 is a graphic representation of the input data.

Figure 2–11: Actual Plot of the Input Data



Example 1 is idealized in several respects. Normally you will not know that the input array will be empty upon return from the subroutine or that the output array had 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 type of example was chosen to illustrate 1) minimal requirements for implementation and 2) how the subroutine and its parameters affect a given set of data.

In Example 1 the data are input as four 256-point parts; the subroutine processes each part as it is received, placing the results in the output array, which is large enough to accommodate the complete set of processed data. Upon return from the subroutine, the input array is always empty (INPTR = -1) and the output array is never filled (NPEAKS ≠ -1).

PEAK Example #1

```
(1)       DIMENSION INPUT(256),OUTPUT(10,3),EMU(4),SIGMA(4),SIZE(4)
          DIMENSION ITABLE(68),VTYPE(2,2)

(2)       DATA VTYPE/'  VA','LLEY','BASE','LINE'/

(3)       DATA EMU/20.,70.,600.,1000./
          DATA SIGMA/20.,10.,200.,100./
          DATA SIZE/950.,400.,300.,200./

(4)       DATA ITABLE/1,2,3,1,1,63*0/
          DATA INLAST,INPTR,IDIMO,NPEAKS/256,0,3,0/

(5)       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

(6)       CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)

(7)     3 CONTINUE

          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,
(8)     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
```

① Define array variables and their size.

② VTYPE is used to print a word describing how the peak ended (TYPE).

③ Arrays EMU, SIGMA, and SIZE are used to produce the waveform to be processed, which is the sum of four Gaussian curves.

④ Data statements initializing the variable input parameters to the algorithm (ITABLE) and the arguments for the call to PEAK.

⑤ Section producing values that represent the waveform: as X increases, the next 256 values are calculated and PEAK is called. Four waveform segments are produced.

⑥ Each time 256 values are produced, PEAK is called.

ITABLE is not affected by the program but is used by the subroutine.

INPUT contains the input data; actual values change each time PEAK is called.

INLAST is the subscript of the last element in INPUT that contains data; always 256 in this example.

INPTR is either 0 (initially) or -1; subroutine looks for data to start in the first element in the INPUT array.

OUTPUT is the array where data for each peak is stored; space is allocated to accommodate all data produced; argument remains unchanged by program.

IDIMO specifies the number of sets of peak data that can be stored before the OUTPUT array is filled.

NPEAKS specifies the number of sets of peak data produced thus far; because results are known, no check is made for a full condition with respect to the output array.

⑦ Loop for each of four sections of waveform. All elements of INPUT array are processed (INPTR=-1), but OUTPUT array still has room (NPEAKS≤IDIMO).

⑧ This section types the results on the terminal.

Terminal output with 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.

Terminal output with No Filter option enabled:

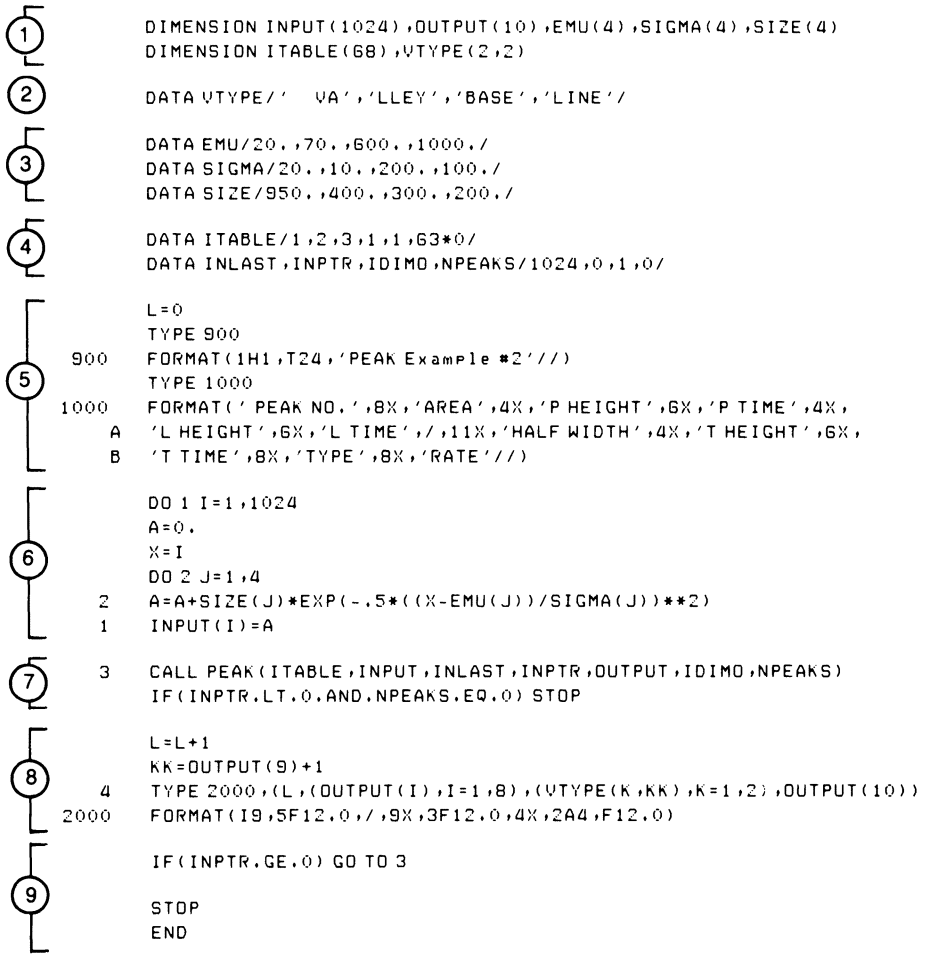
PEAK Example #1

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.

Example 2 is also idealized because the input and output are known and the program has been debugged. Therefore no error checking is done. The variable input parameters (ITABLE) are set to the same values and in the same manner as in Example 1.

All input for this example is presented to the subroutine in one large array. However the output array is large enough for only one set of peak data. Thus each time the subroutine returns to the main program (except for the last return), the output array is full (NPEAKS = -1) but the input array has not been completely processed (INPTR ≠ -1,<INLAST). On return from the subroutine, the data in the output array must be further processed (in this example printed on the terminal) before another call is made to the subroutine to process input data.

PEAK Example #2



```
      DIMENSION INPUT(1024),OUTPUT(10),EMU(4),SIGMA(4),SIZE(4)
      DIMENSION ITABLE(68),VTYPE(2,2)

      DATA VTYPE/'  VA','LLEY','BASE','LINE'/

      DATA EMU/20.,70.,600.,1000./
      DATA SIGMA/20.,10.,200.,100./
      DATA SIZE/950.,400.,300.,200./

      DATA ITABLE/1,2,3,1,1,63*0/
      DATA INLAST,INPTR,IDIMO,NPEAKS/1024,0,1,0/

      L=0
      TYPE 900
  900 FORMAT(1H1,T24,'PEAK Example #2'//)
      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 1 I=1,1024
      A=0.
      X=I
      DO 2 J=1,4
    2 A=A+SIZE(J)*EXP(-.5*((X-EMU(J))/SIGMA(J))**2)
    1 INPUT(I)=A

    3 CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)
      IF(INPTR.LT.0.AND.NPEAKS.EQ.0) STOP

      L=L+1
      KK=OUTPUT(9)+1
    4 TYPE 2000,(L,(OUTPUT(I),I=1,8),(VTYPE(K,KK),K=1,2),OUTPUT(10))
 2000 FORMAT(I9,5F12.0,/,9X,3F12.0,4X,2A4,F12.0)

      IF(INPTR.GE.0) GO TO 3

      STOP
      END
```

① Define array variables and their size.

② VTYPE is used to print a word describing how the peak ended (TYPE).

③ Arrays EMU, SIGMA, and SIZE are used to produce the waveform to be processed, which is the sum of four Gaussian curves.

④ Data statements initializing the variable input parameters to the algorithm (ITABLE) and the arguments for the call to PEAK.

⑤ Type headings for peak output.

⑥ Produce 1024 values representing the waveform.

⑦ Call the PEAK subroutine to continue processing the input array. If the input array is empty and no new peak data are in the output array, exit.

ITABLE is not affected by the program but is used by the subroutine.

INPUT contains data to be processed, some of which may already have been processed.

INLAST is the subscript of the last element in INPUT containing data; it is always 1024.

INPTR is always equal to the subscript of the last element in the input array that was processed. Initially it is zero, but in subsequent calls it points to the last element in the input array that was processed when the output array was filled. The PEAK subroutine manages this element.

OUTPUT is array where data for each peak is stored. Space is available for only one set of peak data. Therefore each time an additional set of peak data is available, the subroutine returns to the main program so that more space can be made available to store data.

IDIMO specifies the number of sets of peak data that can be stored before the OUTPUT array is filled; it is set to one.

NPEAKS is zero (initially), 1 (if one peak was found but the input was exhausted), or -1 if two sets of peak data are ready to be reported.

⑧ Print peak data.

⑨ If input data is not completely processed, call PEAK again to continue processing.

Terminal output with digital filter enabled:

PEAK Example #2

PEAK NO.	AREA HALF WIDTH	P HEIGHT T HEIGHT	P TIME T TIME	L HEIGHT TYPE	L TIME 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.

Terminal output with No Filter option enabled:

PEAK Example #2

PEAK NO.	AREA HALF WIDTH	P HEIGHT T HEIGHT	P TIME T TIME	L HEIGHT TYPE	L TIME 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.

PEAK Example #3

Example 3 is almost identical to Example 1. But, because Example 3 processes autogained data, the peak-processing subroutine had to be built with AUTOG$ enabled. Three changes were made to the source code of example 1:

1. In Section 1, the size of ITABLE was increased to 79 elements.
2. In Section 4, elements 6 and 7 of ITABLE have been set to 0 and element 8 has been set to 1 to specify autogained data.
3. In Section 5, each element in INPUT has had 4000 octal added to it to simulate bipolar zero and 30000 octal added to it to simulate the highest gain value possible and to eliminate the need for normalization.

PEAK Example #3

```
(1)         DIMENSION INPUT(256),OUTPUT(10,3),EMU(4),SIGMA(4),SIZE(4)
            DIMENSION ITABLE(79),VTYPE(2,2)

(2)         DATA VTYPE/'  VA','LLEY','BASE','LINE'/

(3)         DATA EMU/20.,70.,600.,1000./
            DATA SIGMA/20.,10.,200.,100./
            DATA SIZE/950.,400.,300.,200./

(4)         DATA ITABLE/1,2,3,1,1,2*0,1,71*0/
            DATA INLAST,INPTR,IDIMO,NPEAKS/256,0,3,0/

            X=0.
            DO 3 K=1,4
            DO 1 I=1,256
(5)         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+"34000

(6)         CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)

(7)       3 CONTINUE

            TYPE 900
        900 FORMAT(1H1,T24,'PEAK Example #3'//)
            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,
(8)       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
```

① Define array variables and their size.

② VTYPE is used to print a word describing how the peak ended (TYPE).

③ Arrays EMU, SIGMA, and SIZE are used to produce the waveform to be processed, which is the sum of four Gaussian curves.

④ Data statements initializing the variable input parameters to the algorithm (ITABLE) and the arguments for the call to PEAK.

⑤ Section producing values that represent the waveform: as X increases, the next 256 values are calculated and PEAK is called. Four waveform segments are produced.

⑥ Each time 256 values are produced, PEAK is called.

ITABLE is not affected by the program but is used by the subroutine.

INPUT contains the input data; actual values change each time PEAK is called.

INLAST is the subscript of the last element in INPUT containing data; always 256 in this example.

INPTR is either 0 (initially) or -1; subroutine looks for data to start in the first element in the INPUT array.

OUTPUT is array where data for each peak is stored; space is allocated to accommodate all data produced; argument remains unchanged by program.

IDIMO specifies the number of sets of peak data that can be stored before the OUTPUT array is filled.

NPEAKS specifies the number of sets of peak data produced thus far; because results are known, no check is made for a full condition with respect to the output array.

⑦ Loop for each of four sections of waveform. All elements of INPUT array are processed (INPTR=-1), but OUTPUT array still has room (NPEAKS≤IDIMO).

⑧ This section types the results on the terminal.

Terminal output with digital filter enabled:

PEAK Example #3

PEAK NO.	AREA HALF WIDTH	P HEIGHT T HEIGHT	P TIME T TIME	L HEIGHT TYPE	L TIME 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.

Terminal output with No Filter option enabled:

PEAK Example #3

PEAK NO.	AREA HALF WIDTH	P HEIGHT T HEIGHT	P TIME T TIME	L HEIGHT TYPE	L TIME 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.

Example 4 is almost identical to Example 3. But because Example 4 processes double precision input data, the peak-processsing subroutine had to be built with DPP$ enabled. Four sections of Example 3 were changed:

1. In Section 1, INPUT was changed to a double precision integer array and some additional elements were defined for the purpose of generating the input data.
2. In Section 3, the values of the parameters used to produce the input waveform for PEAK were changed to yield double precision values.
3. In Section 4, ITABLE(5) was changed to -1 so that the output would be given in double precision, floating-point values.
4. In Section 5, the algorithm to calculate the input to PEAK was changed to produce double precision integer values.

PEAK Example #4

```
      REAL*8 OUTPUT(10,3),EMU(4),SIGMA(4),SIZE(4),A,B,X
      INTEGER TEMP(512)
      DIMENSION ITABLE(79),VTYPE(2,2)
      INTEGER*4 INPUT(256)
      EQUIVALENCE (TEMP,INPUT)

      DATA B/65536.D0/
      DATA VTYPE/'  VA','LLEY','BASE','LINE'/

      DATA EMU/20.D0,70.D0,600.D0,1000.D0/
      DATA SIGMA/20.D0,10.D0,200.D0,100.D0/
      DATA SIZE/950.D3,400.D3,300.D3,200.D3/

      DATA ITABLE/1,2,3,1,-1,2*0,1,71*0/
      DATA INLAST,INPTR,IDIMO,NPEAKS/256,0,3,0/

      X=0.D0
      DO 3 K=1,4
      DO 1 II=2,512,2
      A=0.D0
      X=X+1.D0
      DO 2 J=1,4
    2 A=A+SIZE(J)*DEXP(-.5D0*((X-EMU(J))/SIGMA(J))**2)
      TEMP(II)=A/B
      C=A-B*TEMP(II)
      IF(C.GE.32768.D0) TEMP(II-1)=C-B
      IF(C.LT.32768.D0) TEMP(II-1)=C
    1 CONTINUE

      CALL PEAK(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)

    3 CONTINUE

      TYPE 900
  900 FORMAT(1H1,T24,'PEAK Example #4'//)
      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
```

① Define array variables and their size.

② VTYPE is used to print a word describing how the peak ended (TYPE).

③ Arrays EMU, SIGMA, and SIZE are used to produce the waveform to be processed, which is the sum of four Gaussian curves.

④ Data statements initializing the variable input parameters to the algorithm (ITABLE) and the arguments for the call to PEAK.

⑤ Section producing double precision integer values that represent the waveform: as X increases, the next 256 values are calculated and PEAK is called. Four waveform segments are produced.

⑥ Each time 256 values are produced, PEAK is called.

ITABLE is not affected by the program but is used by the subroutine.

INPUT contains the double precision input data; actual values change each time PEAK is called.

INLAST is the subscript of the last element in INPUT containing data; always 256 in this example.

INPTR is either 0 (initially) or -1; subroutine looks for data to start in the first element in the INPUT array.

OUTPUT is array where data for each peak is stored; space is allocated to accommodate all data produced; argument remains unchanged by program.

IDIMO specifies the number of sets of peak data that can be stored before the OUTPUT array is filled.

NPEAKS specifies the number of sets of peak data produced thus far; because results are known, no check is made for a full condition with respect to the output array.

⑦ Loop for each of four sections of waveform. All elements of INPUT array are processed (INPTR=-1), but OUTPUT array still has room (NPEAKS≤IDIMO).

⑧ This section types the results on the terminal.

Terminal output with digital filter enabled:

PEAK Example #4

PEAK NO.	AREA	P HEIGHT	P TIME	L HEIGHT	L TIME
	HALF WIDTH	T HEIGHT	T TIME	TYPE	RATE
1	32158219.	952958.	20.	693113.	4.
	11.	483071.	44.	BASELINE	1.
2	10732416.	452258.	68.	344193.	54.
	7.	189827.	83.	BASELINE	1.
3	134705883.	300062.	600.	14923.	107.
	119.	201630.	839.	VALLEY	1.

Terminal output with No Filter option enabled:

PEAK Example #4

PEAK NO.	AREA	P HEIGHT	P TIME	L HEIGHT	L TIME
	HALF WIDTH	T HEIGHT	T TIME	TYPE	RATE
1	35898677.	954478.	20.	608373.	1.
	13.	398766.	48.	BASELINE	1.
2	10745955.	454131.	68.	342397.	54.
	7.	189097.	83.	BASELINE	1.
3	134706424.	300068.	600.	14877.	107.
	119.	201624.	839.	VALLEY	1.

ENVELOPE–PROCESSING (NVELOP) Subroutine

FORMAT:

CALL NVELOP(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)

Where:

ITABLE is a 51-element integer array.

ITABLE(1)	=	number of additional values supplied
ITABLE(2)	=	gate parameter
ITABLE(3)	=	baseline value indicator
ITABLE(4)	=	output data type
ITABLE(5)	=	error indicator
ITABLE(6)	=	reentry pointer

INPUT is an integer array containing input data.

INLAST is an integer variable specifying subscript of last data element in INPUT processed.

INPTR is an integer variable specifying subscript of last element in INPUT processed.

OUTPUT is a double-subscripted array used to store output data.

OUTPUT(1,N)	=	ending indicator, Nth peak
OUTPUT(2,N)	=	area, Nth peak
OUTPUT(3,N)	=	centroid, Nth peak
OUTPUT(4,N)	=	width, Nth peak
OUTPUT(5,N)	=	crest time, Nth peak
OUTPUT(6,N)	=	crest height, Nth peak
OUTPUT(7,N)	=	leading minimum time, Nth peak

The following elements appear if ITABLE(1) = 1 or 2.

OUTPUT(8,N)	=	first additional value preceding current envelope
OUTPUT(9,N)	=	second additional value preceding current envelope

IDIMO is an integer variable specifying number of peak data sets that can be stored in OUTPUT.

NPEAKS is an integer variable specifying number of peak data sets already stored in OUTPUT.

FILE NAMES:

FNVLOP.MAC (source file); FNVLOP.OBJ (object file)

OPTIONS:

- EIS (Extended Instruction Set — KE11-E)
- EAE (Extended Arithmetic Element — KE11)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If the following options are enabled:

NONE	EIS	EAE
669	620	644

TYPICAL EXECUTION SPEED:

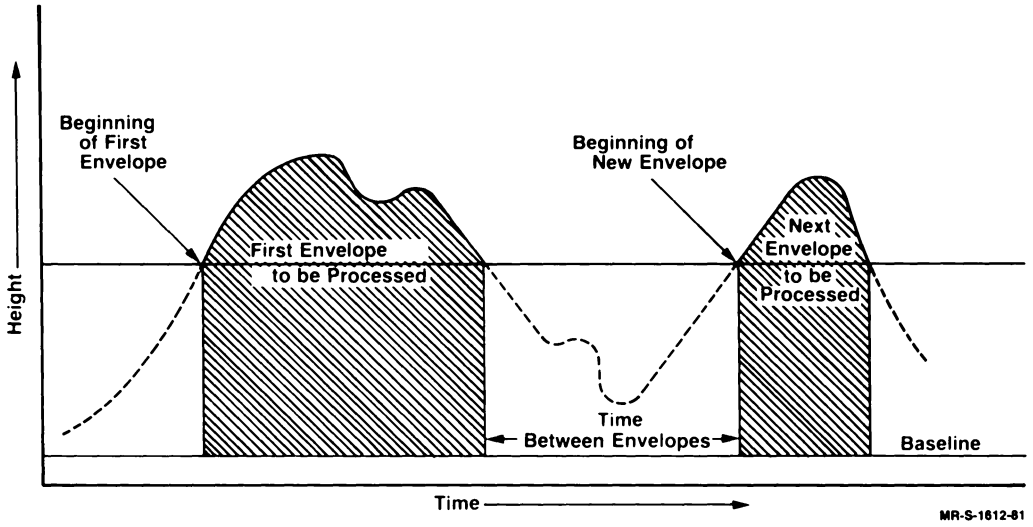
With PDP–11/34 and EIS enabled: 4300 Points/second.

With PDP–11/03 and EIS enabled: 2000 Points/second.

Chapter 3
The Envelope-Processing (NVELOP) Subroutine

This subroutine detects significant fluctuations, called peaks, in sets of data. These data represent discontinuous segments, or envelopes, of a waveform (Figure 3–1). The envelopes are chosen arbitrarily, but they should be presented sequentially to the original waveform.

Figure 3–1: Envelopes of Data (may contain more than one peak)



Each envelope is represented by a series of discrete positive integers corresponding to values of a waveform at evenly spaced intervals. For the result of applying the algorithm for this subroutine to be correlated with the overall waveform, you must supply the distances (elapsed times) between envelopes. You can also supply as many as two additional data values for related functions at the critical point where a new set of envelope data starts.

The subroutine reports definitive characteristics for all peaks found; output is the area, width, and centroid for each peak, as well as the height and time of its crest. The time at which each peak starts and where it ends — at envelope termination or at a valley — is also reported. The additional values supplied at the start of a new envelope are reported with each peak detected in that envelope.

3.1 Definition of Basic Terms and Conventions

It is important to understand how some of the terms and conventions describing the NVELOP subroutine are used throughout this chapter.

- The term *data stream* describes all values presented to the subroutine for processing.
- Data values that represent the peak being processed are referred to as *heights*, for example, crest height is the maximum data value for a given peak.
- The duration axis of the waveform is the *time* axis, for example, the time at which an envelope begins is the leading time of a peak. Time is measured as the number of input values processed since input began.
- Point-to-point changes are *local* changes, as contrasted with overall changes during the course of a waveform, which are *trends*.
- Changes are *persistent* in one direction if the number of changes in that direction exceeds the number in the opposite direction.
- "Noise" is a generic term for all distortion-producing components in the input data.

3.2 The Envelope-Processing Algorithm

The envelope-processing algorithm detects increasing and decreasing trends (which may define peaks) in a set of data representing a segment of a waveform. Output from the subroutine is directly related to the times at which the envelope begins and ends and the points where *changes* in the increasing/decreasing trends take place. We define the point where an envelope begins or where a subsequent increasing trend appears as the *leading minimum* of a peak. A point where the data first show a decreasing trend preceded by an increasing trend is called the *crest* of a peak. Each peak ends either at the end of an envelope or at the start of a new peak (at a valley).

A functional description of how the algorithm applies to each envelope of data follows; details of how the envelope is defined are presented in Section 3.3, which describes the input stream.

3.2.1 The Baseline Value

A baseline value for the waveform data can be either the first element in the input data stream or the third parameter in the input parameter table (Section 3.3). Setting this baseline value is intended to be a mechanism for eliminating the "constant" noise level inherent in input data; it should therefore be less than any of the waveform data values to be processed.

This value is subtracted from each data value before the trend-detection portion of the algorithm is applied. If the result of this subtraction is negative, it is treated as a zero.

3.2.2 Trend Detection — Application of the Gate Factor

Data points may exhibit slight point-to-point fluctuations unrelated to the dominant trend of the data. You may eliminate much of this undesirable fluctuation by selecting an appropriate gate factor. The gate factor specifies a valid directional trend in terms of the number of either persistent or consecutive changes in direction over a series of input data points.

At points where a peak begins or where a crest is detected, neither an increasing nor a decreasing directional trend has yet been established. The next current trend is the first direction in which the data change "gate" number of times.

At intermediate points current trends are already established. Changes in directional trend at these points can be established only if the number of consecutive local changes in the new direction is equal to the gate factor.

A local change is defined in terms of the relation between a given data point and the local minimum and maximum. If the current height is less than the local minimum, the change is downward, and conversely, if the height is greater than the local maximum, the change is upward. If the height is between the local minimum and maximum, no change is indicated (although the area and centroid are updated).

At points of trend change, such as the beginning of a peak or its crest, the local minimum is set to a very high value and the maximum to a very low value. Between points of trend change 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 the peak and its crest. This test is the heart of the algorithm.

3.2.3 The Width of a Peak

Peak width is measured as elapsed time between its start and the end of the peak or the end of the envelope. Time is measured in units of distance between input data values; therefore peak width is the count of the number of input data values that describe the peak.

Operation of the gate factor makes it imperative that at least "gate" number of data values be recorded on either side of the crest before a peak is detected; that is, the width of each peak must be at least twice the gate parameter. The single exception is a peak that ends because the envelope terminates; the width of such a peak must be at least six[1] for it to be reported. The gate factor is not involved, and the output data are reported even if no crest has been detected.

3.2.4 Calculating the Area of a Peak

The area under a peak is found simply by summing the corrected points on the peak. Thus if the width is N,X_i represents the *i*th input data point on the peak, and BAS is the baseline value

$$\text{AREA} = \sum_{i=1}^{N} X_i - \text{BAS}$$

3.2.5 Calculating the Centroid of a Peak

The centroid is the time axis component of the two-dimensional center of mass of a peak. It is calculated as the sum of the products of the current time and corrected points, divided by the area of the peak. Thus if T_i represents time of input of the *i*th data point, and other symbols are as defined in Section 3.2.4, then

$$\text{CENTROID} = \sum_{i=1}^{N} T_i \cdot (X_i - \text{BAS}) / \text{AREA}$$

The centroid is the best approximation of the real position of the peak with respect to the entire waveform.

3.2.6 Flow Charts for the NVELOP Subroutine

The series of flow charts in Figures 3–2 through 3–8 gives detailed logic for the NVELOP subroutine. Supplementary information is presented in Tables 3–1 and 3–2. Table 3–1 defines the symbols used in the flow charts and accompanying explanations; Table 3–2a reviews and summarizes flowcharted events and associated waveform phenomena as they relate to data on possible peak configurations:

- A peak starting and ending at the boundaries of an envelope
- A peak starting at the beginning of an envelope and ending at a valley
- A peak starting at a valley and ending at the termination of an envelope

[1] An arbitrarily chosen number that is twice the nominal gate factor; it is part of the algorithm and therefore may not be changed without modifying the code.

Table 3–2b presents significant data-collection events on your part as they relate to a waveform like that presented in Table 3–2a.

Note that the flow charts and accompanying illustrations and examples assume that you impose an arbitrary threshold value to delimit data envelopes to be processed.

Table 3–1: Definitions of Symbols Used

BAS	Baseline height[1]	NEW	Next input value (height)
CH	Crest height[2]	NOFC	Number of extra values recorded at beginning of new envelope[1]
CI	Crest-found indicator	PA	Partial area accumulated during increase
CT	Time of crest height[2]	PC	Partial centroid accumulated during increase
DC	Counter for number of persistent decreases in height	TA	Total area summation[2]
DI	Trend indicator (decreasing) 1 = decreasing 0 = not decreasing	TC	Total centroid summation[2,3]
GT	Number of persistent increases or decreases to be considered a valid change in trend[1]	TM	Local time
IC	Counter for number of persistent increases in height	TT	True time
II	Trend indicator (increasing) 1 = increasing 0 = not increasing	T_0	Time of start of envelope
LMT	Time of start of peak[2]	WD	Width of peak[2]
MN	Current minimum height	XL	Large value
		X1	First extra value recorded[2,1]
MX	Current maximum height	X2	Second extra value recorded[2,1]
MXT	Time of current maximum height	Type	0 Peak ends at end of envelope 1 Peak ends at valley[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 the peak is detected.

Figure 3–2: Flow Chart for Envelope Processing; Data Entry

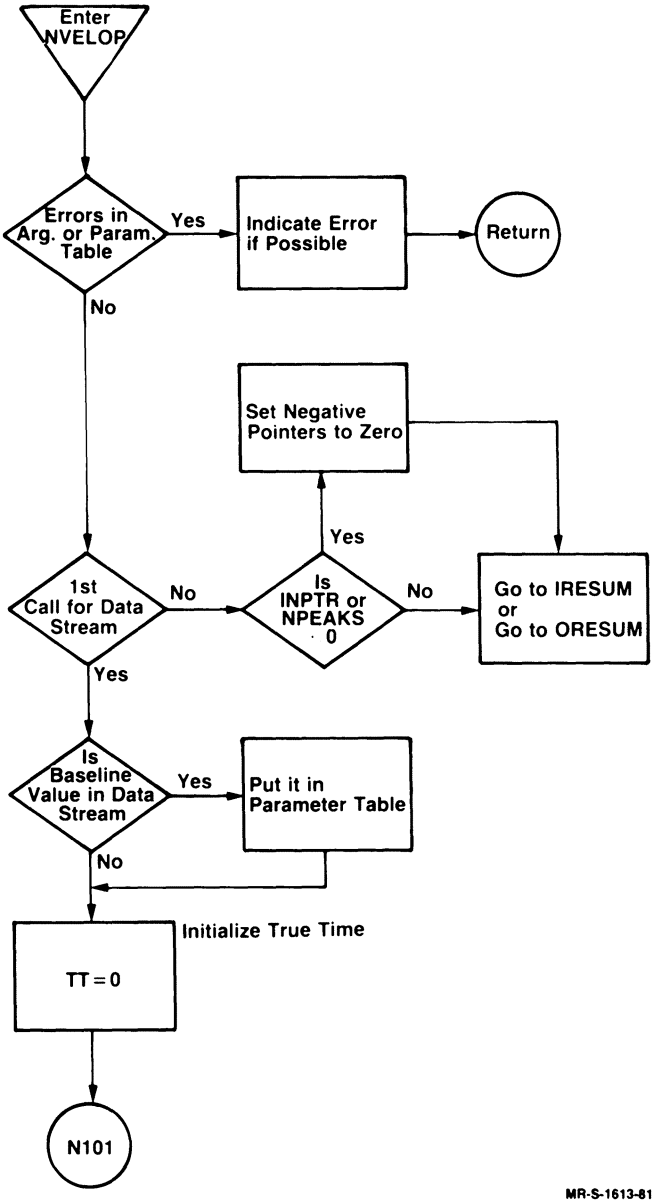


Figure 3–3: Flow Chart for Envelope Processing; Initialization, Calculation of Peak Width, and Finding End of Envelope

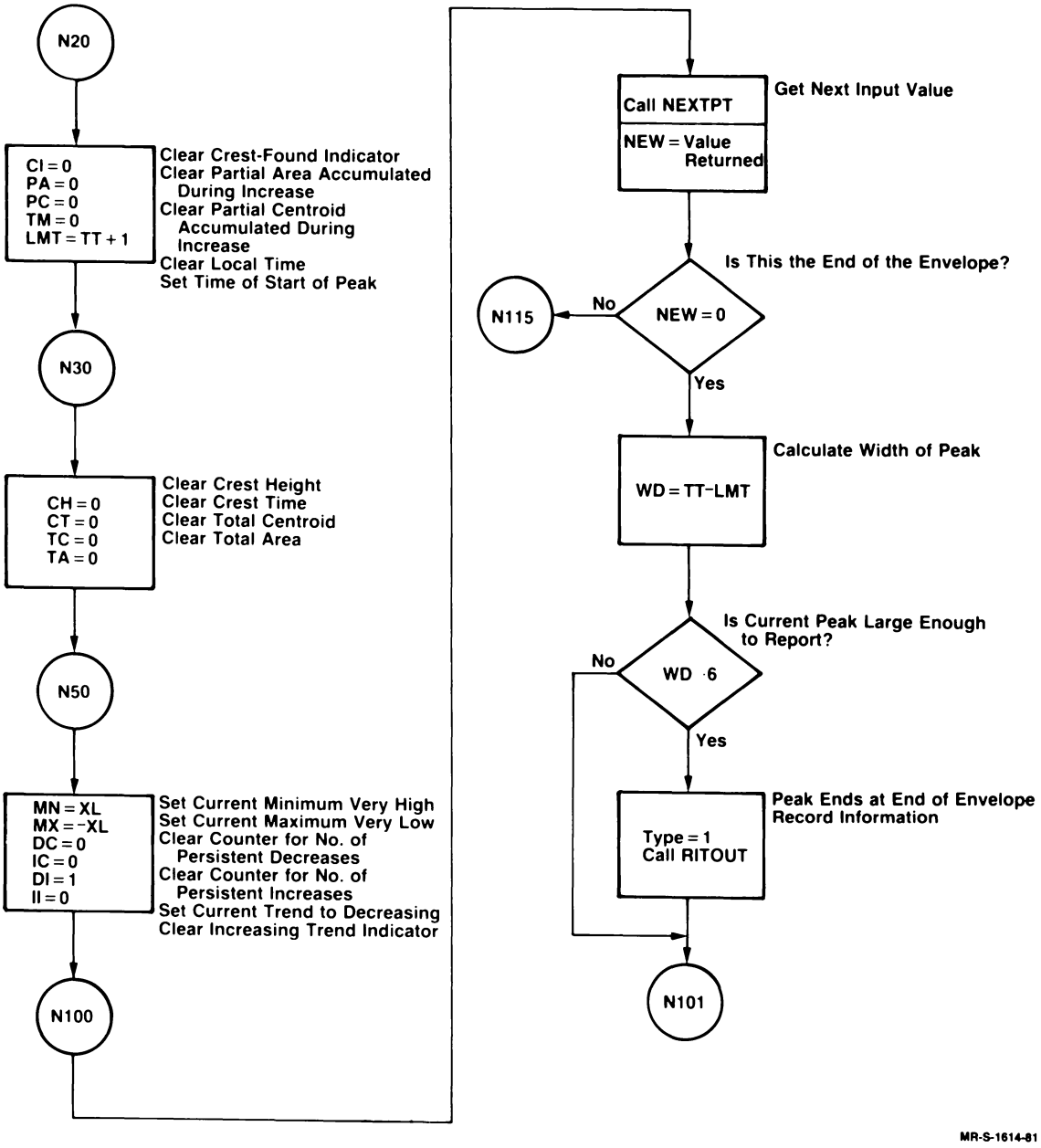


Figure 3–4: Flow Chart for Envelope Processing; Count of Reject Points and Reading Additional Values where Envelope Begins

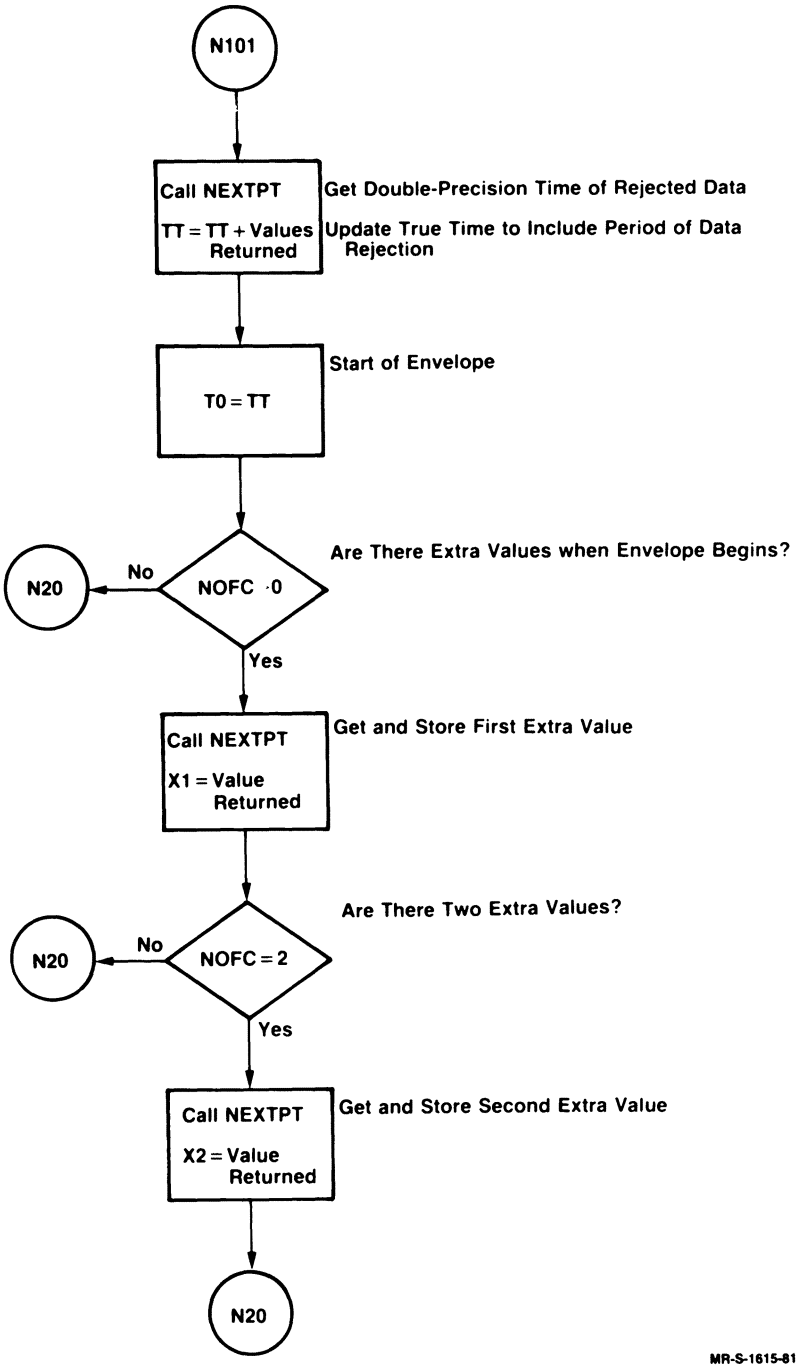


Figure 3–5: Flow Chart for Envelope Processing; New Minimum and Crest Detection

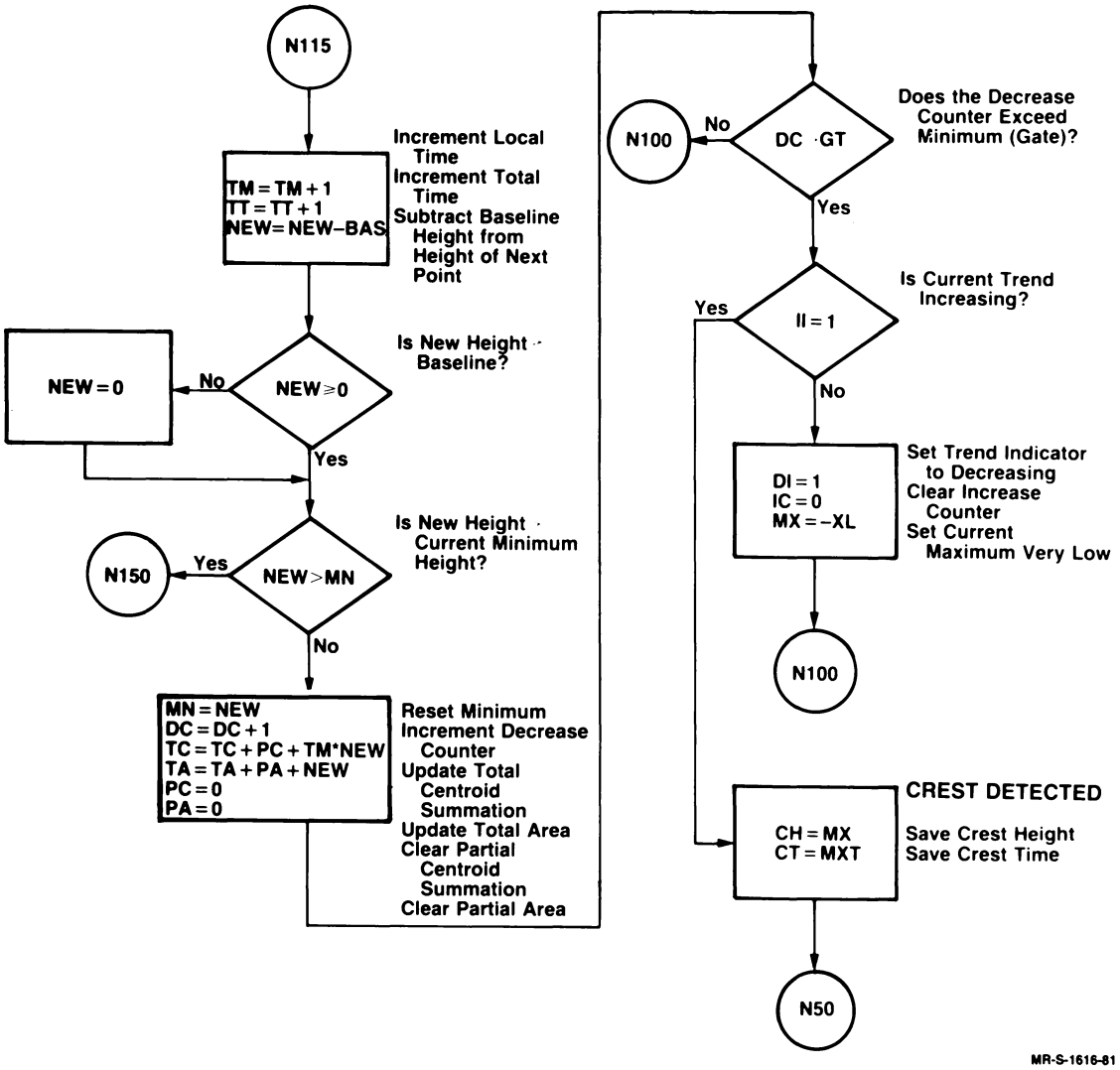


Figure 3–6: Flow Chart for Envelope Processing; New Maximum; Peak Begins at Valley

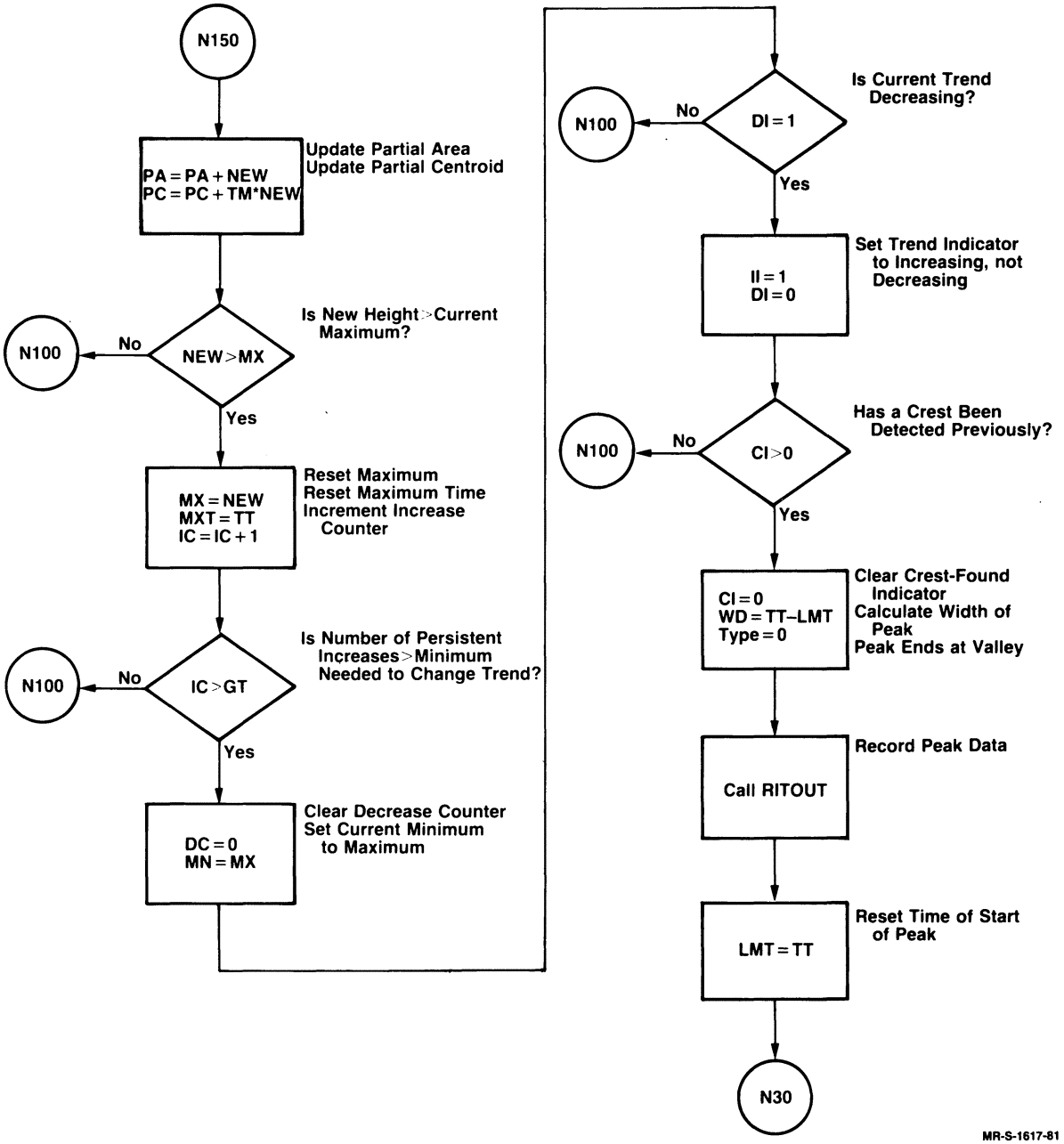


Figure 3–7: NEXTPT Subroutine — Envelope Processing

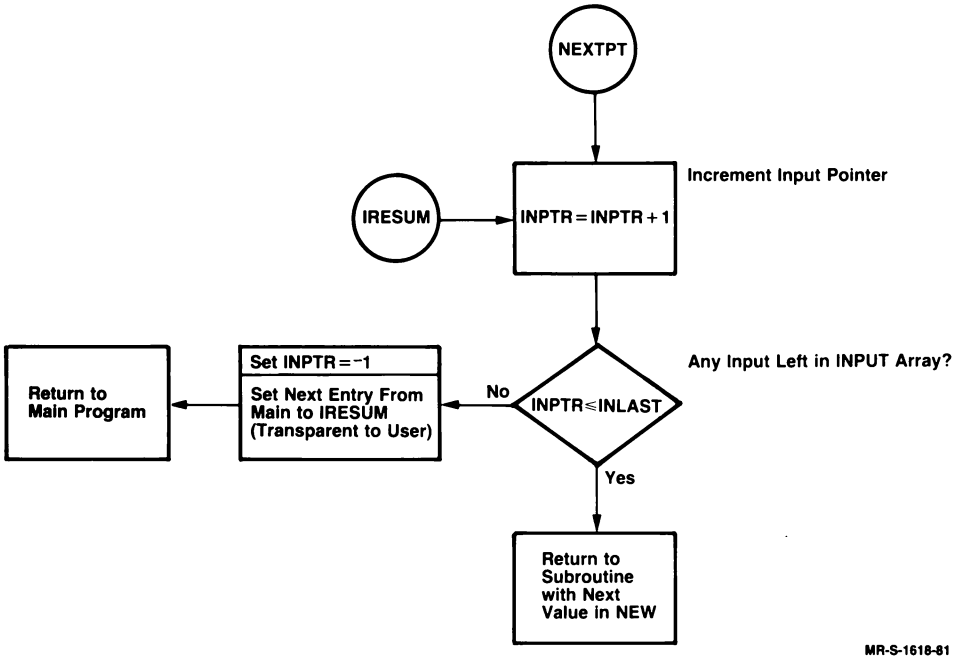


Figure 3–8: RITOUT Subroutine — Envelope Processing

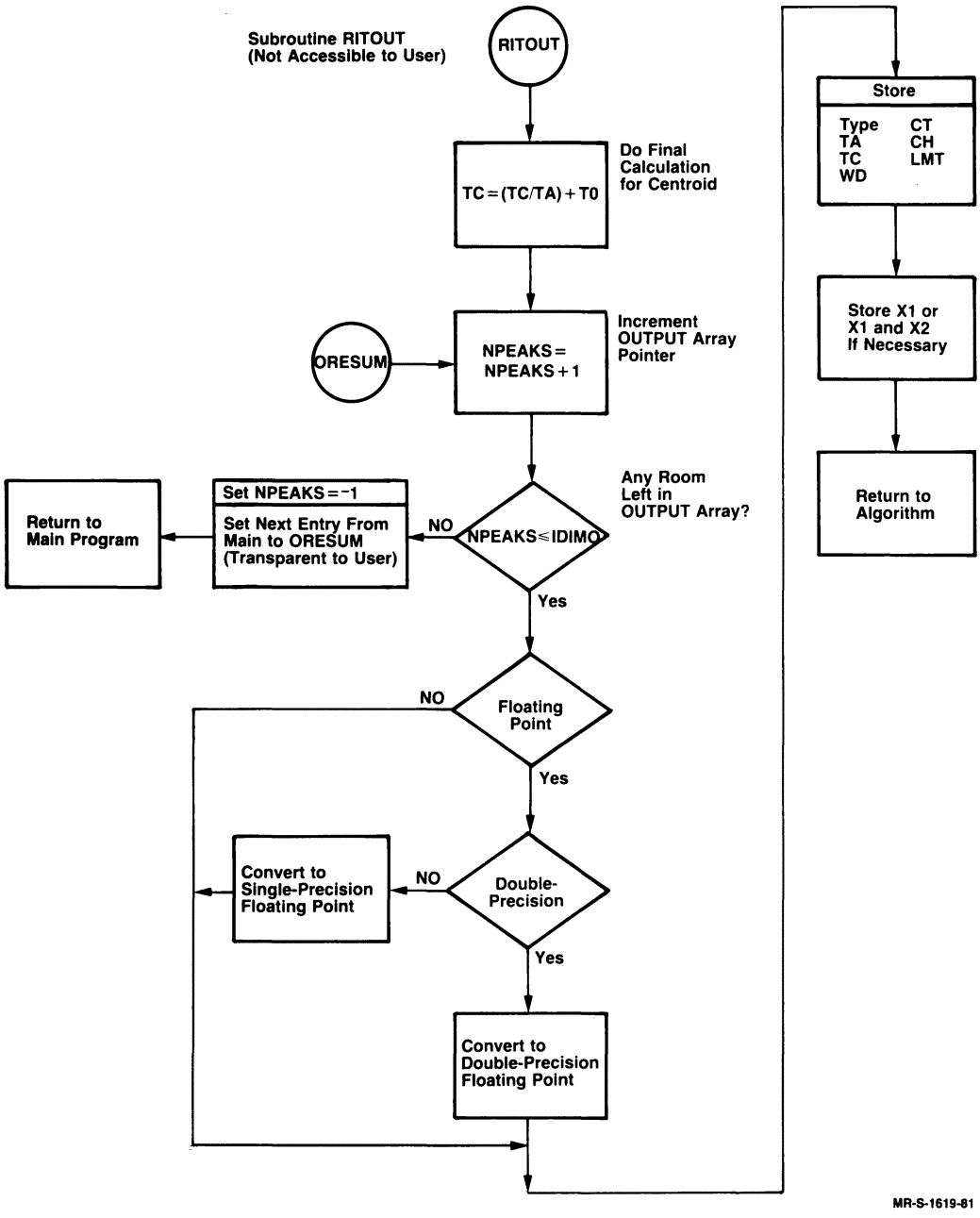
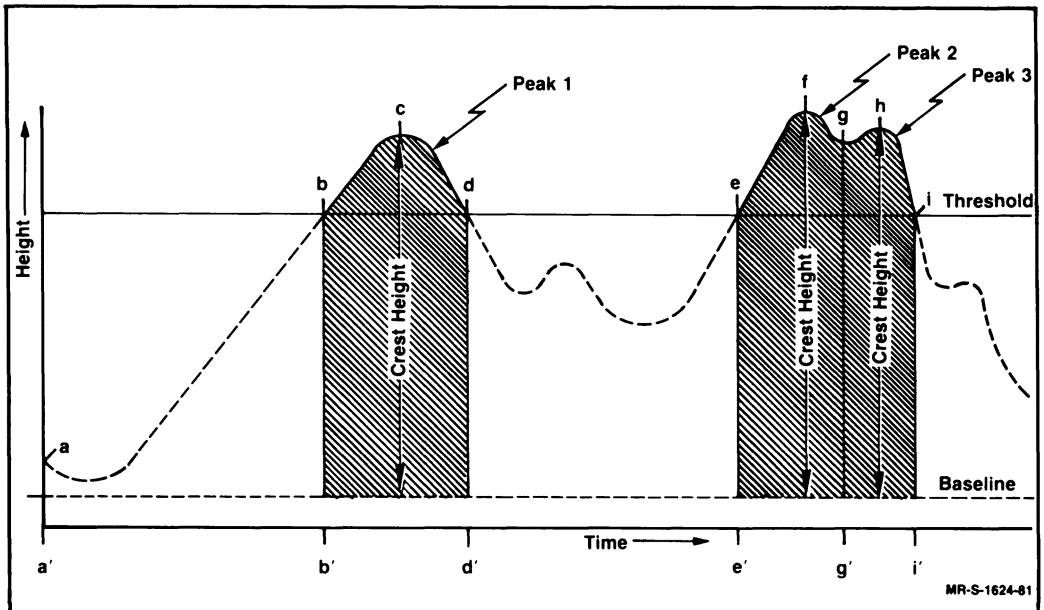


Table 3–2a: Envelope-Processing Algorithm



Point/Section of Curve	Description of Event	Flow Chart Reference
b	New envelope begins; look for	
	1) Elapsed time since last data point processed (or beginning of waveform) and update current total time	
	2) Values other than waveform data, supplied when new envelope begins; save values to be output with data for each peak detected during current envelope	TT = TT + time since last envelope = b′
	New peak begins; save time of start	LMT = TT + 1 = b′ + 1
b-c	Increasing trend in data; change in established trend will indicate crest detection	DI = 0,II = 1 DC<GT,IC>GT
c	Decreasing trend established; detect and record crest height and time	CH = c CT = c′
c-d	Decreasing trend continues; change in established trend will indicate end of peak and start of next peak	DI = 1,II = 0 DC>GT,IC<GT
d	Envelope of data ends; peak ends on threshold; calculate width and centroid; enter peak data into output array	WD = TT–LMT = d′-b′ Type = 0

NOTE

The NVELOP subroutine must receive information describing the length of interval d′-e′. Table 3–2b details this information.

(Continued on next page)

Table 3–2a: Envelope-Processing Algorithm (Cont.)

Point/Section of Curve	Description of Event	Flow Chart Reference
e	New envelope begins; look for 1) Elapsed time since last data point processed and update current total time	
	2) Values other than waveform data, supplied when new envelope begins; save values to be output with data for each peak detected during current envelope	TT = TT + time since last envelope = TT = (e'-d')
	New peak begins; save time of start	LMT = TT + 1 = e' + 1
e-f	Increasing trend in data; change in established trend will indicate crest detection	DI = 0,II = 1 DC<GT,IC>GT
f	Decreasing trend established; detect and record crest height and time	CH = f CT = f'
f-g	Decreasing trend continues; change in established trend will indicate end of peak and start of next peak	DI = 1,II = 0 DC>GT,IC<GT
g	Increasing trend established; current peak ends; calculate width and centroid and enter peak data in output array Next peak begins; save time of start	WD = TT–LMT = g'-e' Type = 1 LMT = TT = g'
g-h	Increasing trend in data; change in established trend will indicate crest detection	DI = 0,II = 1 DC<GT,IC>GT
h	Decreasing trend established; detect and record crest height and time	CH = h CT = h'
h-i	Decreasing trend continues; change in established trend will indicate end of current peak and start of next peak	DI = 1,II = 0 DC>GT,IC<GT
i	Envelope of data ends; peak ends on threshold; calculate width and centroid and enter peak data into output array	WD = TT–LMT = i'-g' Type = 0

Table 3–2b: How to Compile and Prepare Data for Envelope-Processing Subroutine

Pertinent Times	Description of Event	Data Supplied to the Subroutine
a′	Start of data stream; data below threshold	None (unless baseline value entered)
a′-b′	Monitor data and test against threshold value; count number of data samples below threshold (rejected)	None
b′	Data break threshold; ascertain values of auxiliary functions to be input	Supply reject count (double-precision) and additional values (if required)
b′-d′	Data above threshold	Place each value in data stream
d′	Data fall below threshold	Place zero in data stream
d′-e′	Monitor data and test against threshold value; count number of data samples below threshold (rejected)	None
c′	Data break threshold; ascertain values of auxiliary functions to be input	Supply reject count (double-precision) and additional values (if required)
e′-i′	Data above threshold	Place each value in data stream
i′	Data fall below threshold	Place zero in data stream

3.3 How to Call the Envelope-Processing Subroutine

The symbolic name for the envelope-processing subroutine is NVELOP, and the general format for the FORTRAN call is:

```
CALL NVELOP(ITABLE,INPUT,INLAST,INPTR,OUTPUT,IDIMO,NPEAKS)
```

For reference, argument names in the call to NVELOP have been assigned arbitrarily. You may 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, either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results and no data is processed. The arguments are described in the following discussion.

ITABLE is an integer array of 51 elements and is used by the subroutine 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 subroutine.

ITABLE(1) Number of additional values (no more than two) that you supply in the data stream each time a set of data for a new envelope begins. These values are presumably related to, but not representative of, the waveform being processed.

ITABLE(2) This parameter defines 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 3.2.2. In general, suggested values may range from 2 to 5.

ITABLE(3) The parameter that indicates the baseline value BAS (Section 3.2.4) to be used with the waveform data:

If ITABLE(3)≥0, it is used as the baseline value.
If ITABLE(3)<0, the first element in the data stream is the baseline value; when this value is detected in the data stream, the subroutine inserts it into ITABLE(3).

ITABLE(4) The element that defines the data type of the output array:

=0	Output is double-precision integer
=1	Output is single-precision real
=-1	Output is double-precision real

ITABLE(5) The element used by the subroutine to indicate errors in the calling sequence or input parameters:

=0	Indicates no error
=N	Indicates ITABLE(N) is in error, for example, ITABLE(1)>2 ITABLE(2)<0
=-N	Indicates the Nth argument is in error or missing; for example, INPTR>INLAST (see the following discussion)

ITABLE(6) Element that must be set to zero before the initial call to the subroutine to process a new waveform. When a data stream is processed in parts (see Section 3.4), the subroutine uses ITABLE(6) for reentry to process each subsequent part. For this reason you should not alter this element until all parts have been processed.

ITABLE(7) Elements used exclusively by the subroutine while the data stream is being processed.

ITABLE(51)

INPUT is an integer array containing the raw data to be processed by the subroutine. There may be as many as five different types of data for a single waveform:

Baseline Value If ITABLE(3) is nonnegative, this value is omitted from the data stream. If ITABLE(3) is negative, the first element in the input stream is taken to be the baseline value to be used; if the baseline value is negative, it is treated as a zero (Section 3.2.1).

Reject Count

This double-precision integer provides the elapsed time, in terms of data samples, between envelopes of data. Actually it is the count of data values rejected by the subroutine since the last value of the previous envelope. Because the array is single-precision integer, two elements are needed for each reject count; the least significant part (low order) is entered first, followed immediately by the most significant part (high order) in the next element. For example, if the reject count is R, and the subscript for the preceding element is N:

$$\text{INPUT}(N + 2) = R / 2^{16}$$

$$\text{INPUT}(N + 1) = R - (\text{INPUT}(N+2)) \cdot 2^{16}$$

Keep in mind that R is real, INPUT is integer, and that fractional truncation takes place when a real number is equated to an integer.

Auxiliary Values

If ITABLE(1) is zero, no auxiliary values will appear in the data stream. Otherwise as many as two values, not representative of the waveform, may be supplied and associated with each set of data corresponding to an envelope to be processed. These values may be any single-precision integers, and there must be as many as specified in ITABLE(1).

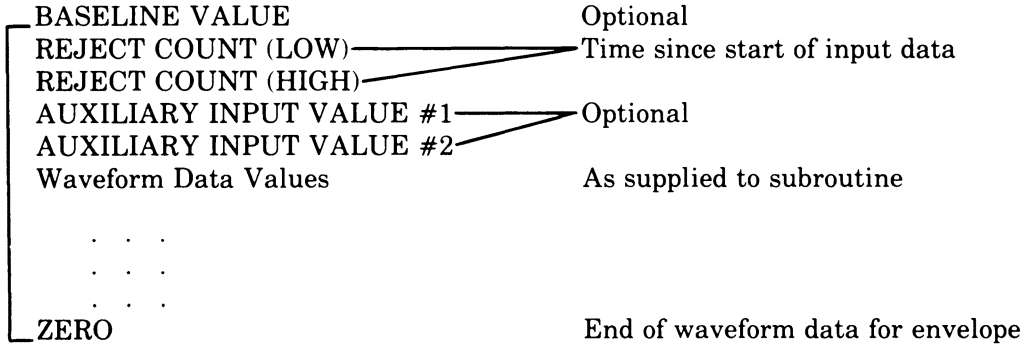
Waveform Data Values

Heights of the waveform at evenly-spaced intervals must be supplied for each envelope of data to be processed; heights that do not exceed the baseline value are treated as equal to the baseline value and greater than zero.

Zero Values

Because all waveform data to be processed must be greater than zero, detection of a zero value has special significance in that it indicates the end of an envelope (if it is obviously not a baseline value).

The subroutine distinguishes between these various kinds of data in the input stream on the basis of the order in which they are expected to appear:



The subset of input values designated by the bracket is repeated for each envelope of data to be processed until the data stream ends.

INLAST is an integer variable that equals the value of the subscript of the last array element that contains data.

INPTR is an integer variable that points to the value of the subscript of the last element processed by the subroutine. We may also think of it as having a value one less than the subscript of the next datum in the input array 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 this argument before calling the subroutine; however, the subroutine changes the value before returning.

OUTPUT is a double-subscripted array used to store the results of applying the envelope-processing algorithm. The first dimension specifies the number of data elements to be output for each peak detected. There are always at least seven. In addition the ITABLE(1) auxiliary values provided with each envelope are reported for each peak found in that envelope. The second dimension specifies the number of sets of peak data that can be stored by the algorithm while processing the input data. Thus the first dimension is 7 + ITABLE(1), and the second is defined by IDIMO. Possible data elements reported for each peak are:

OUTPUT(1,N)	Indicator of how peak ended: = 0 ended at termination of envelope = 1 ended at a valley
OUTPUT(2,N)	Area of Nth peak
OUTPUT(3,N)	Centroid of Nth peak
OUTPUT(4,N)	Width of Nth peak
OUTPUT(5,N)	Position of Nth peak (time of crest height)
OUTPUT(6,N)	Height of Nth crest
OUTPUT(7,N)	Starting position of Nth peak (time of leading minimum height)

The following elements are optional:

OUTPUT(8,N)	First additional value at start of envelope (if ITABLE(1)>0)
OUTPUT(9,N)	Second additional value at start of envelope (if ITABLE(1) = 2)

IDIMO is an integer variable that transmits to the subroutine 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 variable giving the number of sets of peak data stored in the output array. We may 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 envelope data to be stored, NPEAKS should be set to zero.

You must set the value of this argument before calling the subroutine; however, the subroutine can change the value before returning.

NOTE

NVELOP returns (assuming there are no errors) after either of the following events:

1. All input data elements have been processed.
2. 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 subroutine has set NPEAKS to the proper value for the next subroutine 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 subroutine is called again with either INPTR or NPEAKS equal to -1, the subroutine interprets the value as zero.

3.4 Using the Envelope-Processing Subroutine

You can use several inherent features of the envelope-processing subroutine to process data produced in real time. Thus, you can use NVELOP 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 3.3).

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 data, respectively (Figure 3–9). INLAST is a pointer to the last INPUT element containing data.

The subroutine 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.

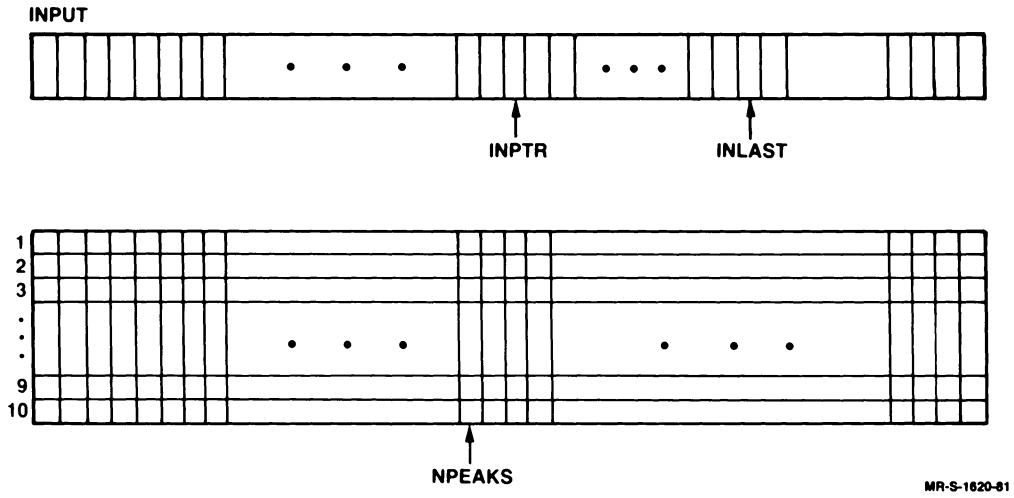
These conditions give you great flexibility in handling subroutine input and output. When you have large quantities of data to process, you need not allocate space for all data at once because the subroutine is designed to process a given data set in sequential parts. In fact, all data need not be known before processing begins, as illustrated by 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 upon each return from the subroutine or it might be further processed only when the output buffer was filled, that is, NPEAKS=-1. You may choose the procedure that is most convenient for you.

Furthermore, you can change all arguments in the calling statement except ITABLE between successive calls to the subroutine to reflect the origin of the remaining input data and where the output is to be stored. You must not tamper with ITABLE during the intervals between calls for a given data stream. ITABLE contains the current information needed to resume processing at the point where processing was stopped on the previous call.

The subroutine 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 that several data streams can be processed simultaneously. All pertinent information concerning the history of a data stream is contained in the ITABLE array rather than in the code for the subroutine. Imaginative use of the arguments in the subroutine call should make the subroutine functionally compatible with any application that uses the envelope-processing algorithm.

Figure 3–9: INPTR, NPEAKS, and INLAST Point to Slots



3.5 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the envelope-processing subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

3.5.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS(KE11–E) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 100 words.

3.5.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE(KE11) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 50 words.

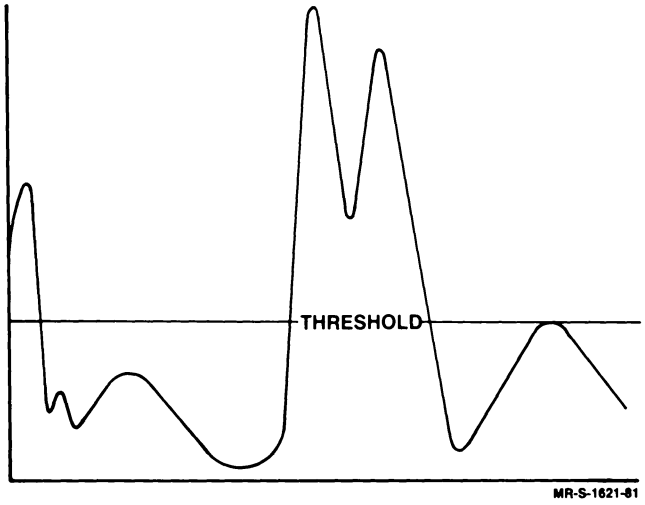
3.6 Examples Using the NVELOP Subroutine

The three examples presented here all use the same code and process the same 1024 data points — the sum of six Gaussian waveforms. The code compares the composite waveform against a preset threshold value (Figure 3–10) and supplies the results to the NVELOP subroutine. Each time the input array is filled (256 elements), the subroutine is called to process the data and place the results in the output array. Upon return from the subroutine to the main program, the input array is always empty (INPTR = -1), and the output array has never overflowed (NPEAKS ≠ -1).

The code for these examples is idealized in several respects. Normally you will not know that the input array will be empty upon return from the subroutine or that the output array had 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. In practice, ITABLE(5) should always be checked.

This type of example was chosen to illustrate 1) minimal requirements for implementation and 2) how the subroutine and its parameters affect a given set of data.

Figure 3–10: Actual Plot of Input Data, Showing Threshold



NVELOP Example #1

NVELOP Example #1

```
(1)     DIMENSION INPUT(256),OUTPUT(7,3),EMU(6),SIGMA(6),SIZE(6)
        DIMENSION ITABLE(51),VTYPE(3,2)

(2)     DATA VTYPE/'    ',' VA','LLEY','   T','HRES','HOLD'/

(3)     DATA EMU/20.,75.,200.,500.,600.,900./
        DATA SIGMA/20.,10.,75.,30.,35.,100./
        DATA SIZE/500.,100.,200.,800.,700.,300./
        DATA IT,IC,X/300,0,0./

(4)     DATA ITABLE/0,3,0,1,47*0/
        DATA J,INPTR,IDIMO,NPEAKS/0,0,3,0/

(5)     DO 3 I=1,1024
        A=0.
        X=X+1.
        DO 2 JJ=1,6
      2 A=A+SIZE(JJ)*EXP(-.5*((X-EMU(JJ))/SIGMA(JJ))**2)

(6)     IF(A.LE.IT) GO TO 4
        IF(IC.EQ.0) GO TO 6
        J=J+1
        INPUT(J)=IC
        ASSIGN 21 TO IRET
        GO TO 30
     21 J=J+1
        INPUT(J)=0
        ASSIGN 22 TO IRET
        GO TO 30
     22 IC=0
      6 J=J+1
        IF(I.NE.1) GO TO 7
        INPUT(1)=0
        INPUT(2)=0
        J=3
      7 INPUT(J)=A
        ASSIGN 3 TO IRET
        GO TO 30
      4 IF(J.EQ.0.OR.IC.NE.0) GO TO 20
        J=J+1
        INPUT(J)=0
        ASSIGN 20 TO IRET
        GO TO 30
     20 IC=IC+1
      3 CONTINUE
        IF(J.EQ.0) GO TO 10
        ASSIGN 10 TO IRET

(7)  30 IF(J.NE.256.AND.I.LT.1024) GO TO IRET
        CALL NVELOP(ITABLE,INPUT,J,INPTR,OUTPUT,IDIMO,NPEAKS)
        J=0
        GO TO IRET

(8)  10 TYPE 900
     900 FORMAT(1H1,T16,'NVELOP Example #1',//)
        TYPE 1002
    1002 FORMAT(' PEAK NO.',8X,'TYPE',8X,'AREA',4X,'CENTROID',5X,
       1 'P WIDTH',/,15X,'P TIME',4X,'P HEIGHT',3X,'LEAD TIME')
        DO 11 L=1,NPEAKS
        KK=1+OUTPUT(1,L)
      11 TYPE 1003,L,(VTYPE(K,KK),K=1,3),(OUTPUT(I,L),I=2,7)
    1003 FORMAT(I9,3A4,3F12.0,/,9X,3F12.0)
        END
```

(1) Define array variables and their size.

(2) VTYPE is used to print a word describing how the peak ended (TYPE).

(3) Parameters external to the NVELOP subroutine are initialized. Arrays EMU, SIGMA, and SIZE are used to produce the waveform to be processed, which is the sum of six Gaussian curves. IT is the threshold value; IC and X are used for preparation of the data.

(4) Data statements initializing the variable input parameters to the algorithm (ITABLE) and the arguments for the call to NVELOP.

(5) Section producing values that represent the waveform as a function of X.

(6) Raw data are prepared for processing by comparing them to the threshold value (IT), then either incrementing the reject count (IC), if the datum is below threshold, or entering the datum in the input array if above threshold.

(7) Each time 256 input values are produced, or when all 1024 raw data points have been produced, NVELOP is called.

ITABLE is not affected by the program but is used by the subroutine.

INPUT contains the input data; actual values change each time NVELOP is called.

J is the subscript of the last element in INPUT that contains data; always 256 except on last call.

INPTR is either 0 (initially) or −1; subroutine looks for data to start in the first element in the INPUT array.

OUTPUT is the array where data for each peak are stored; space is allocated to accommodate all data produced; argument remains unchanged by program.

IDIMO specifies the number of sets of peak data that can be stored before the OUTPUT array is filled.

NPEAKS specifies the number of sets of peak data produced thus far; because results are known, no check is made for a full condition with respect to the output array.

After NVELOP is called, all elements of the INPUT array are processed (INPTR = −1).

(8) This section types the results on the terminal.

Terminal Output

NVELOP Example #1

PEAK NO.	TYPE P TIME	AREA P HEIGHT	CENTROID LEAD TIME	P WIDTH
1	THRESHOLD 20.	17882. 511.	20. 1.	40.
2	VALLEY 501.	57935. 812.	504. 458.	96.
3	THRESHOLD 599.	53617. 706.	597. 554.	92.

NVELOP Example #2

Example 2 processes the same raw data points but the threshold value is lower (Figure 3–11). More of the data points are now above threshold than in Example 1, and more peaks are recorded. Thus we have enlarged the output array so that we can use the same code as in Example 1. Note the following changes in the source code.

In Section 1, the size of OUTPUT is increased:

```
DIMENSION INPUT(256),OUTPUT(7,6),EMU(6),SIGMA(6),SIZE(6)
```

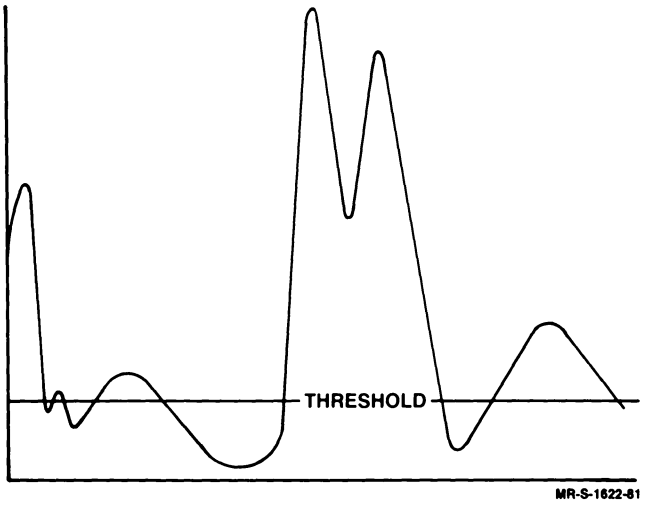
In Section 2, the threshold value (IT) is changed:

```
DATA IT,IC,X/150,0,0./
```

In Section 3, the value of IDIMO is changed to indicate that OUTPUT has more room:

```
DATA J,INPTR,IDIMO,NPEAKS/0,0,6,0/
```

Figure 3–11: Plot of Input Data, Showing New Threshold Value



Note that Peaks 1, 4, and 5 correspond to those in Example 1. Peak heights are not affected by the lowered threshold value, but because more data are "exposed", almost every other data value is changed.

Terminal Output

NVELOP Example #2

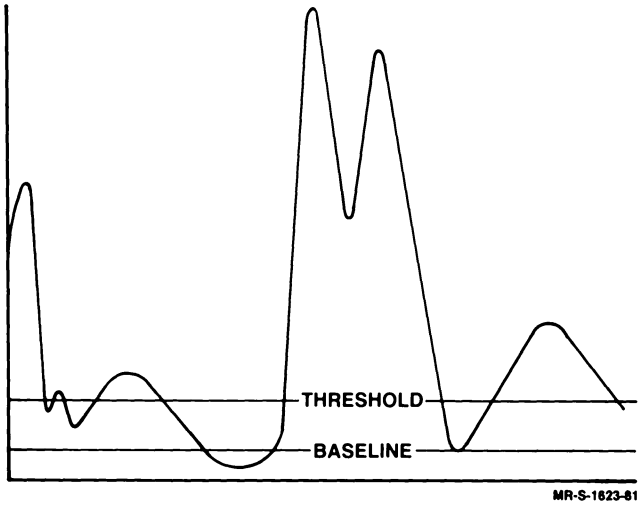
PEAK NO.	TYPE P TIME	AREA P HEIGHT	CENTROID LEAD TIME	P WIDTH
1	THRESHOLD	20890.	24.	54.
	20.	511.	1.	
2	THRESHOLD	1730.	74.	10.
	74.	161.	69.	
3	THRESHOLD	20576.	200.	112.
	193.	199.	144.	
4	VALLEY	60566.	502.	108.
	501.	812.	446.	
5	THRESHOLD	57374.	601.	109.
	599.	706.	554.	
6	THRESHOLD	57046.	900.	234.
	892.	299.	783.	

NVELOP Example #3

In Example 3 it is assumed that there is a nonzero baseline offset of 50 input units in the same raw data processed in Examples 1 and 2 (Figure 3–12). Using the code as modified for Example 2, we may eliminate the assumed baseline offset by setting the third value in the ITABLE array to 50 (Section 4):

```
DATA ITABLE/0,3,50,1,47*0/
```

Figure 3–12: Plot of Input Data, Showing Threshold Value from Example 2 and Assumed Baseline Offset



Because the only change introduced to Example 2 is the baseline correction, the only output values that change are the heights, and indirectly, the areas.

Terminal Output

NVELOP Example #3

PEAK NO.	TYPE	AREA	CENTROID	P WIDTH
	P TIME	P HEIGHT	LEAD TIME	
1	THRESHOLD	18140.	24.	54.
	20.	461.	1.	
2	THRESHOLD	1180.	74.	10.
	74.	111.	69.	
3	THRESHOLD	14926.	200.	112.
	193.	149.	144.	
4	VALLEY	55266.	502.	108.
	501.	762.	446.	
5	THRESHOLD	51774.	600.	109.
	599.	656.	554.	
6	THRESHOLD	45296.	900.	234.
	892.	249.	783.	

INTERVAL HISTOGRAMMING (HISTI) SUBROUTINE

FORMAT:

CALL HISTI(ITABLE,INPUT,IHGRAM)

Where:

ITABLE is an integer array of at least 10 elements.

ITABLE(1)	=	first interval lower limit
ITABLE(2)	=	specified interval length
ITABLE(3)	=	specified number of contiguous intervals considered
ITABLE(4)	=	total number of input array elements containing data
ITABLE(5)	=	initialization flag
ITABLE(6)	=	underflow count
ITABLE(7)	=	overflow count
ITABLE(8)	=	number of IHGRAM elements exceeding largest possible single-precision integer
ITABLE(9)	=	error indicator
ITABLE(10)	=	temporary internal storage element

INPUT is an integer array containing input data.

IHGRAM is an integer array used to store output data.

FILE NAMES:

HISTI.MAC (source file); HISTI.OBJ (object file)

OPTIONS:

- EIS (Extended Instruction Set — KE11–E)
- EAE (Extended Arithmetic Element — KE11)
- DPH$ (Double-Precision Integers)
- FREQ$ (Frequency Histogram)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If the following options are enabled:

	NONE	EIS	EAE
NONE	218	87	131
DPH$	287	164	200
FREQ$	279	158	192
DPH$ AND FREQ$	368	245	281

TYPICAL EXECUTION SPEED:

With PDP–11/34 and EIS enabled: 20000 Points/second.

With PDP–11/03 and EIS enabled: 6500 Points/second.

Chapter 4
The Interval Histogramming (HISTI) Subroutine

The interval histogramming subroutine counts the number of data elements that fall into one or more predefined categories or data types. Sets of such counts are often presented graphically as bar-graphs or histograms. In the context of this subroutine, a category is a defined numeric interval, and a set of categories for a given application must be representable as a contiguous group of intervals of equal length. Data to be processed must be represented as integers.

Results are presented as an array in which each output element corresponds to a specific category. The output element reports the number of data elements that fall into that category. Reported separately is a count of the number of input data elements that do not belong in any of the predefined categories. In addition, the subroutine can optionally produce a frequency histogram.

4.1 Definition of Basic Terms and Conventions

It is important to understand how some of the terms and conventions describing the HISTI subroutine are used throughout this chapter.

- The term *data stream* (or *input data stream*) describes all data to be processed to produce one histogram. Note that the entire data stream need not be processed at once; it may be processed in sequential parts.
- *Interval* describes a subset of integers. If N is taken to be its length, the interval is defined in terms of its lower boundary point and the next N–1 integers in ascending order.
- *Category* is a unique classification of data.

- Two areas that are outside the total range of interest are: those values that are smaller than the minimum, or *underflow* values, and those larger than the maximum, or *overflow* values.
- *Event* means something that generates a valid data element.
- *Continuum* means a continuous entity, parts of which can be distinguished from neighboring parts only by arbitrary division.

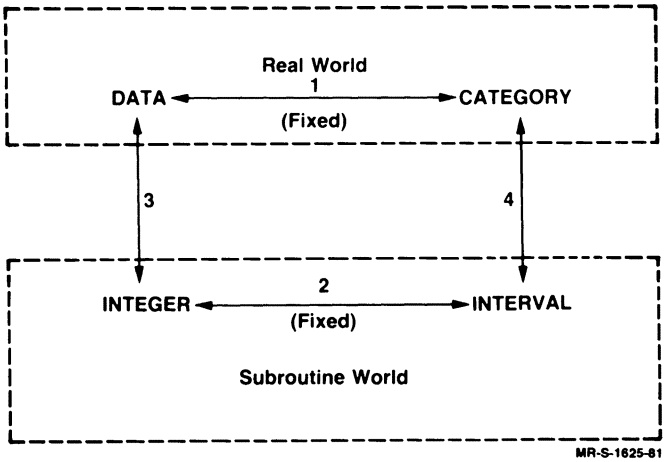
4.2 Your Input to the Subroutine: Its Characteristics

4.2.1 The Relation between Data and Categories

Input to the subroutine is an array of integers that are related in some way to the actual data they represent. If the data are numerical, such as measures of height, temperature, or time, this relation is immediately meaningful and obvious. However, the relation may be purely arbitrary, as when the data deal with an abstract condition that is represented by an integer for convenience in processing; for example, if balls of different colors are being counted, an integer might be assigned to represent one color and distinguish it from other colors.

Data categories are also numerical; each is represented by an interval of integers. And like the data/integer relation, the relation between a category and the interval representing it may be meaningful or completely arbitrary. Values assigned to these interrelated entities (Figure 4–1) must be mutually consistent. For instance, an integer representing a data element must be in the numeric interval corresponding to the particular category to which that data element belongs. Figure 4–1 illustrates that relations 1 and 2 are fixed, and that once relation 3 or 4 is chosen, the remaining relation is no longer completely arbitrary.

Figure 4–1: Interrelation between DATA/CATEGORY and INTEGER/INTERVAL Concepts



4.2.2 Describing the Categories

The numeric intervals representing the categories of interest to the subroutine are defined by an integer array of parameter values that you set. The first three values in the parameter table define the intervals (see Section 4.3):

- The first element of the parameter table defines the lower numeric limit of the first interval. Data values smaller than this limit do not fall into any predefined category and are reported separately (see Section 4.2.3).
- The second element of the parameter table defines the numeric length of each interval. Therefore the first interval spans integers greater than or equal to the first element but smaller than the sum of the first and second elements. Symbolically, if the first element is I and the second is J, the first interval spans all integers, $K1_i$, for which

  $I \leq K1_i < I + J$

  The second interval spans all integers, $K2_i$, for which

  $I + J \leq K2_i < I + 2 \cdot J$

  and so on. Note that there are J elements in each interval.
- The third element of the parameter table tells the subroutine how many intervals — starting with the value of the first element — to consider. This element also specifies the minimum dimension of the output array because each interval has a corresponding output array element. The implication of this value is that the last interval of interest spans all integers, KN_i, for which

  $I + (N - 1) \cdot J \leq KN_i < I + N \cdot J$

  where N is the integer value of the third element and I and J are the integer values of the first and second elements, respectively.

4.2.3 Overflow and Underflow Counts

Because the intervals of interest may not span the entire range of legal integer input, it is possible that the input data stream may contain values that do not fall within any specified category. The subroutine counts the number of data values outside the upper and lower limits of the specified categories. The number of data values that are smaller than the minimum specified in the first element of the parameter table is called the underflow count and is reported in the sixth element of the parameter array.[1] The number of values that exceed the maximum value in the interval of interest is called the overflow count and is reported in the seventh element of the parameter table.

[1] See Section 4.3, the subroutine call.

4.2.4 How the Subroutine Interprets Single-Precision Numbers

Acceptable input and output values for the HISTI subroutine can be single-precision integers in the range of 0 to 65535; single-precision positive integers in FORTRAN have a range of only 0 to 32767. This apparent conflict is actually a matter of interpretation and decimal representation of the negative integers in FORTRAN; it is easily resolved, as we shall explain.

In theory a 16-bit binary number can represent a decimal number as large as 65535. In FORTRAN this range is divided into two parts: values from 0 to 32767 are interpreted and used as positive integers; values in the other half of the range are interpreted and used as negative integers. In FORTRAN negative numbers ranging from -32768 to -1 correspond directly to binary numbers ranging from 32768 to 65535.

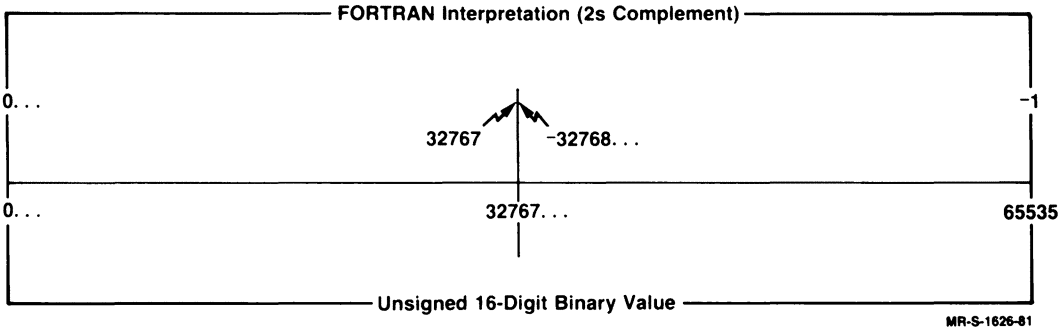
The HISTI subroutine does not process or report values less than zero. Therefore all input and output values are treated as *unsigned* so that we can use the full positive range of the 16-bit word.

You may well ask "How does this interpretation of the data by the subroutine affect my particular application?" The answer is that:

1. All input values and all resulting output data less than 32768 (2^{15}) are treated in exactly the same way by the subroutine and by FORTRAN (Figure 4–2).

2. If either your input or output data contain values greater than 32767 (but not greater than 65535), they are interpreted as negative by FORTRAN but not by the subroutine. For example, if the subroutine outputs a single-precision integer value of 40,000, FORTRAN interprets it as -25536.

Figure 4–2: Relation between FORTRAN Integers and Unsigned Binary Values



We shall now present a simple mechanism for converting an integer that FORTRAN interprets as negative to a *floating-point number* equal to the unsigned interpretation of the value. This conversion may not always be necessary; some operations, such as addition and subtraction, are unaffected by the signs of the operands. The unsigned results of such an operation would be correct so long as the results were not less than 0 or greater than 2^{16}. However, operations such as division and multiplication, or printing and typing, are affected by the FORTRAN interpretation of unsigned integers larger than 32767 as negative numbers.

The following two methods can be used to convert unsigned single-precision integers to floating-point numbers:

```
R = (1 - N/IABS(N)) / 2 * 65536. + N
```

or

```
R = N
IF (N.LT.0) R = 65536. + N
```

where R is the floating-point variable equivalent to the unsigned single-precision integer stored in N, and IABS(N) is a function available from the FORTRAN library.

To convert a floating-point number less than 65536 to an unsigned integer you can use one of the following equations:

```
N = R - 65536 * IFIX (R/32768.)
```

or

```
N = R - 65536.
IF (R.LT.32768.) N = R
```

where R is again the floating-point variable, N the unsigned integer stored in N, and IFIX is a function available from the FORTRAN library.

4.3 How to Call the Interval Histogramming Subroutine

The symbolic name for the interval histogramming subroutine is HISTI, and the general format for the FORTRAN call[1] is:

```
CALL HISTI(ITABLE,INPUT,IHGRAM)
```

For reference, argument names in the call to HISTI have been assigned arbitrarily. You may 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, either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results and no data is processed. The arguments are described in the following discussion.

[1] Format for the distributed version; it may change if the option FREQ$ is enabled (see Section 4.5.4).

ITABLE is an integer array of at least 10 elements, used to:

- Transmit information to the subroutine from the user (ITABLE(1) through ITABLE(5))
- Return information from the subroutine to the user (ITABLE(6) through ITABLE(9))
- Store information on an interim basis (by the subroutine) (ITABLE(10))

You must set the array elements that transmit information to the subroutine.

ITABLE(1) The lower limit of the first interval (see Section 4.2.2)

ITABLE(2) The specified interval length (see Section 4.2.2)

ITABLE(3) The total number of contiguous intervals to be considered (see Section 4.2.2)

ITABLE(4) The total number of array elements containing data (starting with the first element in the input array (INPUT))

ITABLE(5) A value that must be set to zero before the initial call to the subroutine; it signals the subroutine to initialize the output array and other output elements in the parameter table. On subsequent calls to the subroutine, if a different segment of the same data stream is being processed, operation continues, and there is no reinitialization.

The next group of elements is used to report information not included in the actual histogram.

ITABLE(6) The underflow, or count of the number of input data values that are smaller than ITABLE(1)

ITABLE(7) The overflow, or count of the number of input data values that exceeds the upper limit of the last interval; the number of data values exceeding

ITABLE(1)+ITABLE(2) · ITABLE(3)

are reported here.

ITABLE(8) The number of output array elements that have exceeded the largest possible single-precision integer (65535). In FORTRAN this integer is 32767; however, this subroutine can report integers as large as 65535 by treating all data as unsigned positive integers (see Section 4.2.4).

ITABLE(9) An element used to report error conditions:

=0	Indicates no errors
=1	Indicates that ITABLE(1)+ITABLE(2)· ITABLE(3)>65535
=N	Indicates ITABLE(N) is in error, for example, ITABLE(2)=0

The following element is used by the subroutine for its own operation:

ITABLE(10) An element used for internal storage on a temporary basis.

INPUT is an integer array containing the data to be processed. All data are treated as positive and unsigned (see Section 4.2.4). The number of array elements to be processed is ITABLE(4), and processing always begins with the first element of the array. Note, however, that all data need not be placed in the array at one time. Instead, one array of data can be processed, the array refilled with new data, and the subroutine called again to process the array a second time. It is possible to continue in this piecemeal fashion until all data have been processed. In real-time applications such a processing cycle becomes an ongoing function.

IHGRAM is an integer array to store the results of processing; each array element is devoted exclusively to one of the numerical intervals that represent a single category. The order of the array elements corresponds to the numeric order of the intervals, that is, the Nth element of the output array will have a value equal to the number of data elements in the input stream that belong in the interval.

ITABLE(1)+(N-1)·ITABLE(2) to ITABLE(1)+N·ITABLE(2)-1

4.4 Input and Output — Using the Subroutine

As previously stated, all data for a particular histogram need not be available, or even known, before processing begins. Initialization takes place (all counters are set to zero) only when ITABLE(5) equals zero. Parameter table elements are also checked for correctness when ITABLE(5) equals zero. Before the subroutine returns, it automatically changes ITABLE(5) so that if a subsequent call is made to process new data for the same histogram, processing continues as though no interruption had taken place. Thus an entire set of data for one histogram may be processed at one time, memory space permitting; or, if the user wishes, the data may be processed one segment at a time. The value assigned to ITABLE(4) should indicate the number of elements in the input array to be processed for a specific call.

The subroutine 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 in parallel. All pertinent information concerning the history of a data stream is contained in the ITABLE array rather than in the code for the subroutine. Imaginative use of the arguments in the subroutine call should make the subroutine functionally compatible with any application that uses interval histogramming.

4.5 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the interval histogramming subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

4.5.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS (KE11–E) hardware or any other floating-point option available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 121 words if DPH$ is not enabled, and by approximately 123 words if DPH$ is enabled.

4.5.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE (KE11) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 87 words.

4.5.3 DPH$ (Double-Precision Integers)

If the data values to be histogrammed exceed 65535 (2^{16}–1), this option must be enabled. If you are using a copy of the subroutine with the option enabled, you must provide your data to the subroutine as follows:

- All input data must be double-precision integer, that is, the second argument in the FORTRAN CALL statement, INPUT, must be an INTEGER*4 array (or equivalent).
- Because this option expands the possible range of input values by a factor of 65536 (2^{16}), the range of variables used to define the exact range of interest must also be expanded by 2^{16}. Therefore the first argument in the FORTRAN CALL statement, ITABLE, must also be an INTEGER*4 array or the equivalent. ITABLE(1) and ITABLE(2) may then select the range of interest.

Enabling this option has no effect on output because the range of input values does not affect the size of output values.

Enabling DPH$ alone increases the memory requirement by 69 words.

4.5.4 FREQ$ (Frequency Histogram)

A frequency (or zeroth) histogram usually has meaning only when input data elements represent contiguous measurements of a continuum and are input sequentially. The frequency histogram then tells, for each contiguous interval along the continuum, how often an event occurred that produced one data element for the subroutine.

For example, suppose we are studying the distribution of discarded beer cans on the side of a road. Let the input data be the distance between cans. The *interval* histogram would then describe the number of times cans were found within certain preset intervals, that is, how many were 5 feet apart or less, how many were between 6 and 10 feet apart, and so on. The *frequency* histogram would describe the distribution over sections of the road; how many beer cans (events) were found in the first quarter mile, how many in the second, and so on.

Another example might be to determine how often two events occur within specified time intervals: for instance, how often do the two events occur within 2 seconds, how often between 2 and 4 seconds, and so on. The frequency histogram would show how many events occurred within each subsequent time slice.

Mathematically, if the interval length in the continuum is chosen to be K, and if an input, which represents contiguous measurements of that continuum, is taken to be N_i, the first element of the frequency histogram is M_1, where:

$$\sum_{i=1}^{M_1} N_i \quad < K \leq \quad \sum_{i=1}^{M_1+1} N_i$$

The second element would be M_2, where:

$$\sum_{i=1}^{M_1+M_2} N_i \quad < 2 \cdot K \leq \quad \sum_{i=1}^{M_1+M_2+1} N_i$$

and so on.

If a frequency histogram is to be produced by the subroutine, this option must be enabled. When it is enabled, the length of the ITABLE array (Section 4.3) must be increased from 10 to 17, and additional parameters must be defined before implementation of the subroutine. The first nine parameters are defined exactly as if FREQ$ were not enabled, but ITABLE(10) through ITABLE(17) must be defined as follows:

ITABLE(10) Indicator of whether to produce a frequency histogram:

- =0 Do not produce a frequency histogram
- =1 Produce a frequency histogram

ITABLE(11) Length (dimension) of array where results of the frequency histogram are to be stored; it should be large enough to hold all input data for one call.

ITABLE(12) Length of the interval of the continuum to be used in constructing the frequency histogram; corresponds to value of K in the preceding equations.

ITABLE(13) Number of frequency histogram elements already calculated; will be set to 0 by the subroutine for the initial call to HISTI, when ITABLE(5) is 0. ITABLE(3) is updated continuously as input is processed by the subroutine. Because elements of this histogram are calculated sequentially, data from the storage array can be extracted each time the subroutine has processed all data in the input array and returned to the calling routine. If elements are extracted, ITABLE (13) can then be adjusted to reflect where the next element of the frequency histogram is to be stored.

ITABLE(14) Current partial count for the next entry in the frequency histogram; as soon as the number of input elements satisfies the requirements for completing the current interval of the frequency histogram, ITABLE(13) is incremented by 1, and the next entry is placed in the histogram.

ITABLE(15) Elements used exclusively by the subroutine for internal storage.

ITABLE(17)

NOTE

If you enable FREQ$ when you build the subroutine, you must still dimension ITABLE to 17 even if ITABLE(1) = 0. The subroutine uses ITABLE(15) through ITABLE(17).

The subroutine also needs another array to store the frequency histogram data. If ITABLE(10) = 1, the FORTRAN CALL statement takes the form:

```
CALL HISTI(ITABLE,INPUT,IHGRAM,IFREQ)
```

IFREQ is the array to contain the frequency histogram data. This integer array must be at least as long as specified by ITABLE(11); upon return, ITABLE(13) will notify the user how many elements in this array contain data.

If you remove data from the array before a subsequent call to the subroutine, you may adjust ITABLE(13) before reentry to reflect the additional storage space now available.

If the number of array elements available for storage is not large enough, that is, if ITABLE(13) becomes larger than ITABLE(11), the subroutine stops computing the frequency histogram (but continues to process data for the interval histogram). You can take corrective action by making ITABLE(13) smaller than ITABLE(11) when the subroutine returns. Note that ITABLE(13) also indicates the number of elements lost in the frequency histogram calculation (ITABLE(13)-ITABLE(11)).

If FREQ$ is enabled, memory requirements are increased by approximately 58 words if DPH$ is not enabled and by approximately 78 words if DPH$ is enabled.

4.6 Examples of Input/Output Variation Using the HISTI Subroutine

The four examples presented here process equivalent sets of data in similar ways but focus on different capabilities of the HISTI subroutine. Example 1 is based on the distributed version of the subroutine. A set of 10,000 random integers ranging from 15 to 225 is processed, 500 at a time. Each sequential set of input points is stored in alternate halves of the input array to simulate real-time applications, which conserve processing time by collecting and processing data in parallel.

NOTE

If you use FORTRAN 77 and you want to duplicate the terminal output for the example programs, replace the standard random-number generator in F4POTS with F4PRAN.OBJ. Terminal output for the example programs is based on the FORTRAN IV random-number generator. The FORTRAN 77 random-number generator is different from that for FORTRAN IV and will not produce the same output. See Section B.1.

Categories of interest comprise 40 intervals of five integers each; the minimum value for the first interval is 20. When processing is complete, the following output is printed:

- Total count of data items belonging to each category, that is, the interval histogram (IHGRAM)
- The number of data items smaller than the total interval of interest, that is, the underflow count (ITABLE(6))
- The number of data items larger than the total interval of interest, that is, the overflow count (ITABLE(7))
- The total number of output counters that have exceeded 65535.

HISTI Example #1

HISTI Example #1

```
      DIMENSION ITABLE(10),INPUT(1000),IHGRAM(40)
      EQUIVALENCE (ITABLE(1),IS),(ITABLE(2),IW)

      DATA ITABLE/20,5,40,500,6*0/
      DATA I1,I2/0,0/

      DO 2 J=2,21

      N=500*MOD(J,2)+1

      DO 1 I=N,N+499
    1 INPUT(I)=RAN(I1,I2)*210+15

      CALL HISTI(ITABLE,INPUT(N),IHGRAM)
      IF(ITABLE(9).EQ.0) GO TO 2
      TYPE 1000,ITABLE(9)
 1000 FORMAT(' ERROR INDICATOR = ',I3)
      STOP

    2 CONTINUE

      TYPE 900
  900 FORMAT(1H1,T30,'HISTI Example #1',//)
      TYPE 2000
 2000 FORMAT(24X,'RESULTING INTERVAL HISTOGRAM'//,
     1 4('     INTERVAL COUNT')/)
      TYPE 3000,((N-1)*IW+IS,N*IW+IS,IHGRAM(N),N=1,40)
 3000 FORMAT(4(I7,'-',I4,I6))
      TYPE 4000,(ITABLE(I),I=6,8)
 4000 FORMAT(//,' UNDERFLOW COUNT = ',I3,/,' OVERFLOW COUNT = ',I3,/,
     1 ' NO. OF COUNTERS WHICH OVERFLOWED = ',I2,//)
      CALL EXIT
      END
```

① Define array elements and their sizes; use equivalence for convenience.

② Set parameter table elements: there are 40 intervals of interest, starting at 20, and comprising 5 elements each; input array contains 500 data elements; set ITABLE(5) to zero to signal initialization. Initialize random number generation variables.

③ Provide 20 sets of data (500 points each) for processing.

④ Determine which half of the input array is to receive the next set of input data.

⑤ Calculate next set of random numbers for processing.

⑥ Process next set of input values:

ITABLE contains parameter table
INPUT contains input for processing, starting at subscript N
IHGRAM is array where interval histogram is stored

Check for error on return.

⑦ End of data generation and processing loop

⑧ Print output:

Interval histogram (IHGRAM)
Underflow (ITABLE(6)); overflow (ITABLE(7)); counter overflow count (ITABLE(8))

Terminal Output

```
                    HISTI Example #1

                RESULTING INTERVAL HISTOGRAM

  INTERVAL  COUNT     INTERVAL  COUNT     INTERVAL  COUNT     INTERVAL  COUNT

   20-  25   218       25-  30   234       30-  35   226       35-  40   242
   40-  45   248       45-  50   255       50-  55   216       55-  60   228
   60-  65   244       65-  70   226       70-  75   245       75-  80   242
   80-  85   271       85-  90   235       90-  95   217       95- 100   256
  100- 105   251      105- 110   246      110- 115   229      115- 120   233
  120- 125   237      125- 130   217      130- 135   242      135- 140   225
  140- 145   232      145- 150   224      150- 155   258      155- 160   249
  160- 165   241      165- 170   231      170- 175   268      175- 180   255
  180- 185   240      185- 190   231      190- 195   232      195- 200   228
  200- 205   240      205- 210   219      210- 215   250      215- 220   239

UNDERFLOW COUNT = 249
OVERFLOW COUNT = 231
NO. OF COUNTERS WHICH OVERFLOWED = 0
```

Example 2 is identical to Example 1 except that a frequency histogram is produced in addition to the results output in Example 1. The distributed version of the object file for the HISTI subroutine *cannot be used* to produce a frequency histogram. The subroutine must be built from the source file with the FREQ$ option enabled in order to produce a frequency histogram (see Section 4.5.4 and Section 1.1).

Once FREQ$ has been enabled and the correct form of the subroutine is available, the parameter table must be expanded to include definitions for the interval size of the frequency histogram and the size of the array in which the frequency histogram data are to be stored. The name of the array to be used to store the frequency histogram data (IFREQ) must also be added to the FORTRAN CALL statement.

Upon completion of processing, the output includes the elements of the frequency histogram in addition to the values output in Example 1.

HISTI Example #2

```
(1)       DIMENSION ITABLE(17),INPUT(1000),IHGRAM(40),IFREQ(120)
          EQUIVALENCE (ITABLE(1),IS),(ITABLE(2),IW)

(2)       DATA ITABLE/20,5,40,500,5*0,1,120,10000,5*0/
          DATA I1,I2/0,0/

(3)       DO 2 J=2,21

(4)       N=500*MOD(J,2)+1

(5)       DO 1 I=N,N+499
        1 INPUT(I)=RAN(I1,I2)*210+15

          CALL HISTI(ITABLE,INPUT(N),IHGRAM,IFREQ)
          IF(ITABLE(9).EQ.0) GO TO 2
(6)       TYPE 1000,ITABLE(9)
     1000 FORMAT(' ERROR INDICATOR = ',I3)
          STOP

(7)     2 CONTINUE

          TYPE 900
      900 FORMAT(1H1,T30,'HISTI Example #2',//)
          TYPE 2000
     2000 FORMAT(24X,'RESULTING INTERVAL HISTOGRAM'//,
         1 4('     INTERVAL COUNT')/)
          TYPE 3000,((N-1)*IW+IS,N*IW+IS,IHGRAM(N),N=1,40)
     3000 FORMAT(4(I7,'-',I4,I6))
          TYPE 4000,(ITABLE(I),I=6,8)
(8)  4000 FORMAT(//,' UNDERFLOW COUNT = ',I3,/,' OVERFLOW COUNT = ',I3,/,
         1 ' NO. OF COUNTERS WHICH OVERFLOWED = ',I2)
          TYPE 5000
     5000 FORMAT(///20X,'CORRESPONDING FREQUENCY HISTOGRAM'//,
         1 5('    ENTRY COUNT')/)
          I=ITABLE(13)
          IF(ITABLE(13).GT.ITABLE(11)) I=ITABLE(11)
          TYPE 7000,(N,IFREQ(N),N=1,I)
     7000 FORMAT(5(I8,I6))
          CALL EXIT
          END
```

① Define array elements and their sizes; use equivalence for convenience

② Set parameter table elements: there are 40 intervals of interest, starting at 20, and comprising 5 elements each; input array contains 500 elements; set ITABLE(5) to zero to signal initialization; indicate frequency histogram should be produced, set number of elements in frequency histogram array, and specify length of frequency histogram interval.
Initialize random number generation variables.

③ Provide 20 sets of data (500 points each) for processing.

④ Determine which half of the input array is to receive the next set of input data.

⑤ Determine next set of random numbers for processing.

⑥ Process next set of input values:

ITABLE contains parameter table
INPUT contains input for processing, starting at subscript N
IHGRAM is array where interval histogram is stored
IFREQ is array where frequency histogram is stored

Check for error on return.
Note that ITABLE(13) is assumed to be less than ITABLE(11), which is set at 120; therefore ITABLE(13) is not checked.

⑦ End of data generation and processing loop

⑧ Print output:

Interval histogram (IHGRAM)
Underflow (ITABLE(6)); overflow (ITABLE(7)); counter overflow count (ITABLE(8))
Frequency histogram (IFREQ(I), where I ranges from 1 to ITABLE(13))

Terminal Output

HISTI Example #2

RESULTING INTERVAL HISTOGRAM

INTERVAL	COUNT	INTERVAL	COUNT	INTERVAL	COUNT	INTERVAL	COUNT
20- 25	218	25- 30	234	30- 35	226	35- 40	242
40- 45	248	45- 50	255	50- 55	216	55- 60	228
60- 65	244	65- 70	226	70- 75	245	75- 80	242
80- 85	271	85- 90	235	90- 95	217	95- 100	256
100- 105	251	105- 110	246	110- 115	229	115- 120	233
120- 125	237	125- 130	217	130- 135	242	135- 140	225
140- 145	232	145- 150	224	150- 155	258	155- 160	249
160- 165	241	165- 170	231	170- 175	268	175- 180	255
180- 185	240	185- 190	231	190- 195	232	195- 200	228
200- 205	240	205- 210	219	210- 215	250	215- 220	239

UNDERFLOW COUNT = 249
OVERFLOW COUNT = 231
NO. OF COUNTERS WHICH OVERFLOWED = 0

CORRESPONDING FREQUENCY HISTOGRAM

ENTRY	COUNT	ENTRY	COUNT	ENTRY	COUNT	ENTRY	COUNT	ENTRY	COUNT
1	81	2	76	3	89	4	80	5	81
6	82	7	88	8	75	9	84	10	78
11	85	12	86	13	85	14	82	15	92
16	79	17	88	18	85	19	87	20	86
21	86	22	82	23	87	24	82	25	87
26	85	27	78	28	82	29	88	30	82
31	86	32	84	33	80	34	91	35	79
36	79	37	87	38	87	39	88	40	85
41	90	42	79	43	90	44	84	45	84
46	78	47	84	48	82	49	84	50	82
51	81	52	79	53	86	54	78	55	82
56	85	57	77	58	89	59	80	60	91
61	83	62	86	63	83	64	82	65	76
66	85	67	93	68	77	69	85	70	93
71	83	72	78	73	90	74	88	75	82
76	88	77	83	78	77	79	88	80	80
81	81	82	90	83	85	84	88	85	83
86	83	87	83	88	88	89	89	90	80
91	85	92	82	93	77	94	87	95	77
96	79	97	91	98	82	99	86	100	83
101	84	102	86	103	78	104	79	105	87
106	77	107	77	108	79	109	82	110	82
111	87	112	82	113	79	114	92	115	76
116	85	117	84	118	94	119	77		

The processing scheme used in Example 3 is identical to that for Example 1. One major variation in Example 3 is that each input data element is increased by a factor of 65536 (2^{16}) and must therefore be treated as a double-precision integer. Because the distributed version of the object file for the HISTI subroutine cannot process double-precision integers, the source file must be built with the DPH$ option enabled (see Section 1.1).

Example 3 illustrates how the proper version of the subroutine processes double-precision integers. Input values are exactly 2^{16} (65536) times larger than those values processed in the previous examples, and output is analogous to that in Example 1.

Note that when double-precision integers are processed, both the input array and certain of the parameter table values must be double-precision integer arrays (data type INTEGER*4). Because the 1st, 2nd, and 12th parameters of ITABLE are all increased by 2^{16}, categories and intervals of interest are increased so that their relation to the input remains the same as in Example 1. Thus output from Example 3 is identical to that from Example 1.

Because this example uses FORTRAN IV, double-precision arrays ITABLD and INPUTD are equivalenced to single-precision arrays ITABLE and INPUT of the same word length. As a result values for the double-precision elements can be set via their single-precision equivalents.

HISTI Example #3

```
      DIMENSION ITABLE(20),INPUT(2000),IHGRAM(40)
      INTEGER*4 ITABLD(10),INPUTD(1000)
      EQUIVALENCE (ITABLE(2),IS),(ITABLE(4),IW),(ITABLE,ITABLD),
     1            (INPUT,INPUTD)

      DATA ITABLE/0,20,0,5,40,0,500,13*0/
      DATA INPUT/2000*0/
      DATA I1,I2/0,0/

      DO 2 J=2,21

      N=500*MOD(J,2)+1

      DO 1 I=N*2,(N+499)*2,2
    1 INPUT(I)=RAN(I1,I2)*210+15

      CALL HISTI(ITABLD,INPUTD(N),IHGRAM)
      IF(ITABLE(17).EQ.0) GO TO 2
      TYPE 1000,ITABLE(17)
 1000 FORMAT(' ERROR INDICATOR = ',I3)
      STOP

    2 CONTINUE

      TYPE 900
  900 FORMAT(1H1,T30,'HISTI Example #3',//)
      TYPE 2000
 2000 FORMAT(24X,'RESULTING INTERVAL HISTOGRAM'//,
     1 4('     INTERVAL',6X),/,4('      X 2**16 COUNT')/)
      TYPE 3000,((N-1)*IW+IS,N*IW+IS,IHGRAM(N),N=1,40)
 3000 FORMAT(4(I7,'-',I4,I6))
      TYPE 4000,(ITABLE(I),I=11,15,2)
 4000 FORMAT(//,' UNDERFLOW COUNT = ',I3,/,' OVERFLOW COUNT = ',I3,/,
     1 ' NO. OF COUNTERS WHICH OVERFLOWED = ',I2,)
      CALL EXIT
      END
```

① Define array elements and their sizes; use equivalence for convenience; note that ITABLD and INPUTD are double-precision.

② Set parameter table elements: there are 40 intervals of interest, starting at $20 \cdot 2^{16} = 1{,}310{,}720$, and comprising $5 \cdot 2^{16} = 327{,}680$ elements each; input array contains 500 elements; set ITABLD(5) to zero to signal initialization.

Clear high and low parts of double-precision input.
Initialize random number generation variables.

③ Provide 20 sets of data (500 double-precision points each) for processing.

④ Determine which half of the input array is to receive the next set of input data.

⑤ Determine next set of random numbers for processing; place random number range from 15 to 225 in the high order part of the double-precision integer array ITABLD via ITABLE, effectively producing random numbers in the range of $15 \cdot 2^{16}$ to $225 \cdot 2^{16}$.

⑥ Process next set of input values:

ITABLD contains parameter table
INPUTD contains input for processing, starting at subscript N
IHGRAM is array where interval histogram is stored

Check for error on return.

⑦ End of data generation and processing loop

⑧ Print output:

Interval histogram (IHGRAM)
Underflow (ITABLD(6)); overflow (ITABLD(7)); counter overflow count (ITABLD(8))

HISTI Example #4

```
(1)     DIMENSION ITABLE(34),INPUT(2000),IHGRAM(40),IFREQ(120)
        INTEGER*4 ITABLD(17),INPUTD(1000)
        EQUIVALENCE (ITABLE(2),IS),(ITABLE(4),IW),(ITABLE,ITABLD),
       1            (INPUT,INPUTD)

(2)     DATA ITABLE/0,20,0,5,40,0,500,11*0,1,0,120,0,0,10000,10*0/
        DATA INPUT/2000*0/
        DATA I1,I2/0,0/

(3)     DO 2 J=2,21

(4)     N=500*MOD(J,2)+1

(5)     DO 1 I=N*2,(N+499)*2,2
      1 INPUT(I)=RAN(I1,I2)*210+15

(6)     CALL HISTI(ITABLD,INPUTD(N),IHGRAM,IFREQ)
        IF(ITABLE(17).EQ.0) GO TO 2
        TYPE 1000,ITABLE(17)
 1000   FORMAT(' ERROR INDICATOR = ',I3)
        STOP

(7)     2 CONTINUE

        TYPE 900
 (8)900 FORMAT(1H1,T30,'HISTI Example #4',//)
        TYPE 2000
 2000   FORMAT(24X,'RESULTING INTERVAL HISTOGRAM'//,
     1  4('     INTERVAL',6X),/,4('      X 2**16 COUNT')/)
        TYPE 3000,((N-1)*IW+IS,N*IW+IS,IHGRAM(N),N=1,40)
 3000   FORMAT(4(I7,'-',I4,I6))
        TYPE 4000,(ITABLE(I),I=11,15,2)
 4000   FORMAT(//,' UNDERFLOW COUNT = ',I3,/,' OVERFLOW COUNT = ',I3,/,
     1  ' NO. OF COUNTERS WHICH OVERFLOWED = ',I2)
        TYPE 5000
 5000   FORMAT(///20X,'CORRESPONDING FREQUENCY HISTOGRAM'//,
     1  5('    ENTRY COUNT')/)
        I=ITABLE(25)
        IF(ITABLE(25).GT.ITABLE(21)) I=ITABLE(21)
        TYPE 7000,(N,IFREQ(N),N=1,I)
 7000   FORMAT(5(I8,I6))
        CALL EXIT
        END
```

① Define array elements and their sizes; use equivalence for convenience; note that ITABLD and INPUTD are double-precision.

② Set parameter table elements: there are 40 intervals of interest, starting at $20 \cdot 2^{16}$ = 1,310,720, and comprising $5 \cdot 2^{16}$ = 327,680 elements each; input array contains 500 elements; set ITABLD(5) to zero to signal initialization; indicate frequency histogram should be produced, set number of elements in frequency histogram array, and specify length of frequency histogram interval ($10000 \cdot 2^{16}$ = 655,360,000).
Clear high and low parts of double-precision input.
Initialize random number generation variables.

③ Provide 20 sets of data (500 double-precision points each) for processing.

④ Determine which half of the input array is to receive the next set of input data.

⑤ Determine next set of random numbers for processing; place random number range from 15 to 225 in the high order part of the double-precision integer array ITABLD via ITABLE, effectively producing random numbers in the range of $15 \cdot 2^{16}$ to $225 \cdot 2^{16}$.

⑥ Process next set of input values:

ITABLD contains parameter table
INPUTD contains input for processing, starting at subscript N
IHGRAM is array where interval histogram is stored
IFREQ is array where frequency histogram is stored

Check for error on return.
Note that ITABLD(13) is assumed to be less than ITABLD(11), which is set at 120; therefore ITABLD(13) is not checked.

⑦ End of data generation and processing loop

⑧ Print output:

Interval histogram (IHGRAM)
Underflow (ITABLD(6)); overflow (ITABLD(7)); counter overflow count (ITABLD(8))
Frequency histogram (IFREQ(I), where I ranges from 1 to ITABLD(13) = ITABLE(25))

Terminal Output

```
                    HISTI Example #4

                RESULTING INTERVAL HISTOGRAM

 INTERVAL          INTERVAL          INTERVAL          INTERVAL
  X 2**16 COUNT     X 2**16 COUNT     X 2**16 COUNT     X 2**16 COUNT

  20-  25   218     25-  30   234     30-  35   226     35-  40   242
  40-  45   248     45-  50   255     50-  55   216     55-  60   228
  60-  65   244     65-  70   226     70-  75   245     75-  80   242
  80-  85   271     85-  90   235     90-  95   217     95- 100   256
 100- 105   251    105- 110   246    110- 115   229    115- 120   233
 120- 125   237    125- 130   217    130- 135   242    135- 140   225
 140- 145   232    145- 150   224    150- 155   258    155- 160   249
 160- 165   241    165- 170   231    170- 175   268    175- 180   255
 180- 185   240    185- 190   231    190- 195   232    195- 200   228
 200- 205   240    205- 210   219    210- 215   250    215- 220   239

UNDERFLOW COUNT = 249
OVERFLOW COUNT = 231
NO. OF COUNTERS WHICH OVERFLOWED = 0

                CORRESPONDING FREQUENCY HISTOGRAM

ENTRY COUNT   ENTRY COUNT   ENTRY COUNT   ENTRY COUNT   ENTRY COUNT

    1    81       2    76       3    89       4    80       5    81
    6    82       7    88       8    75       9    84      10    78
   11    85      12    86      13    85      14    82      15    92
   16    79      17    88      18    85      19    87      20    86
   21    86      22    82      23    87      24    82      25    87
   26    85      27    78      28    82      29    88      30    82
   31    86      32    84      33    80      34    91      35    79
   36    79      37    87      38    87      39    88      40    85
   41    90      42    79      43    90      44    84      45    84
   46    78      47    84      48    82      49    84      50    82
   51    81      52    79      53    86      54    78      55    82
   56    85      57    77      58    89      59    80      60    91
   61    83      62    86      63    83      64    82      65    76
   66    85      67    93      68    77      69    85      70    93
   71    83      72    78      73    90      74    88      75    82
   76    88      77    83      78    77      79    88      80    80
   81    81      82    90      83    85      84    88      85    83
   86    83      87    83      88    88      89    89      90    80
   91    85      92    82      93    77      94    87      95    77
   96    79      97    91      98    82      99    86     100    83
  101    84     102    86     103    78     104    79     105    87
  106    77     107    77     108    79     109    82     110    82
  111    87     112    82     113    79     114    92     115    76
  116    85     117    84     118    94     119    77
```

INTERNAL HISTOGRAMMING WITH REFERENCE POINTS (RHISTI) SUBROUTINE

FORMAT:

CALL RHISTI(ITABLE,INPUT,IHGRAM[,IFREQ])

Where:

ITABLE is an integer array of at least 16 elements.

ITABLE(1)	=	first interval lower limit
ITABLE(2)	=	specified interval length
ITABLE(3)	=	total number of contiguous intervals considered
ITABLE(4)	=	total number of array elements containing data
ITABLE(5)	=	number of data values to be processed after each reference is detected
ITABLE(6)	=	initialization flag
ITABLE(7)	=	number of reference points detected
ITABLE(8)	=	underflow count
ITABLE(9)	=	overflow count
ITABLE(10)	=	number of output array elements exceeding largest possible single-precision integer
ITABLE(11)	=	error flag
ITABLE(12)	=	frequency histogram indicator 0 = no frequency histogram 1 = frequency histogram
ITABLE(13)	=	number of array elements used to store frequency histogram ITABLE(12) = 1)
ITABLE(14)	=	number of frequency histogram elements already calculated (if ITABLE(12) = 1)
ITABLE(15)	=	current partial count for next entry into the frequency histogram (if ITABLE(12) = 1)
ITABLE(16)	=	internal storage

INPUT is an integer array containing input data.

IHGRAM is an integer array used to store output data.

IFREQ is an integer array used to store frequency histogram data (required only when ITABLE(12) = 1).

FILE NAMES:

RHISTI.MAC (source file); RHISTI.OBJ (object file)

OPTIONS:

- EIS (Extended Instruction Set — KE11-E)
- EAE (Extended Arithmetic Element — KE11)
- DPR$ (Double-Precision Integers)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If the following options are enabled:

	NONE	EIS	EAE
NONE	286	165	199
DPR$	360	237	273

TYPICAL EXECUTION SPEED:

With PDP–11/34 and EIS enabled: 15000 Points/second.

With PDP–11/03 and EIS enabled: 5000 Points/second.

Chapter 5
The Interval Histogramming with Reference Points (RHISTI) Subroutine

The interval histogramming with reference points subroutine counts the number of data elements that fall into one or more predefined categories, or data types. Sets of such counts are often presented graphically as bar-graphs or histograms. The subroutine interprets preset numerical intervals as categories; a set of categories for a given application must be representable as a contiguous group of intervals of equal length.

Data to be processed must be represented by unsigned integers not equal to zero. Data elements equal to zero are interpreted by the subroutine as reference points. Processing of input data begins only after the first reference point is detected. Additional reference points are significant in that they partition the data stream.

Interval histogramming results are presented as an array in which each output element is associated with a specific category. A count is also kept of input elements that do not belong in any predefined category; this count is reported separately.

If a frequency histogram is also produced, each element of its output array contains the total number of data elements that appear in the data stream in the segment described by each pair of reference points.

5.1 Definitions of Basic Terms and Conventions

It is important to understand how some of the terms and conventions describing the RHISTI subroutine are used throughout this chapter.

- *Data stream* (or *input data stream*) describes all data to be processed to produce one histogram. Note that the entire data stream need not be processed at once; it may be processed in sequential parts.

- *Interval* describes a subset of integers. If N is taken to be its length, the interval is defined in terms of its lower boundary point and the next N-1 integers in ascending order.
- *Category* is a unique classification of data.
- Two areas that are outside the total range of interest are: those values that are smaller than the minimum, or *underflow* values, and those larger than the maximum, or *overflow* values.
- *Event* means something that generates a valid data element.

5.2 Your Input to the Subroutine: Its Characteristics

Your input to the subroutine should have the form of a stream of integers, divided into like segments by a set of reference points. This section deals with how this data stream is seen in the real world and in the subroutine world; it also explains how the subroutine interprets the single-precision integers in the input stream.

5.2.1 The Reference Points — Their Significance

Reference points serve two purposes:

- They indicate the point in the input data stream where the interval histogram should begin, that is, at the first reference point.
- They denote boundary points in the input stream which are used in the production of a frequency histogram (see Section 5.3.4).

  Reference points are normally used to signify events that subdivide the input data stream into related partitions. For example, reference points can signify ticks of a clock at a preset rate during a sampling process. Or, if you are monitoring responses to a series of stimuli, the occurrence of each stimulus could be recorded as a reference point, and the data characterize the response being observed.

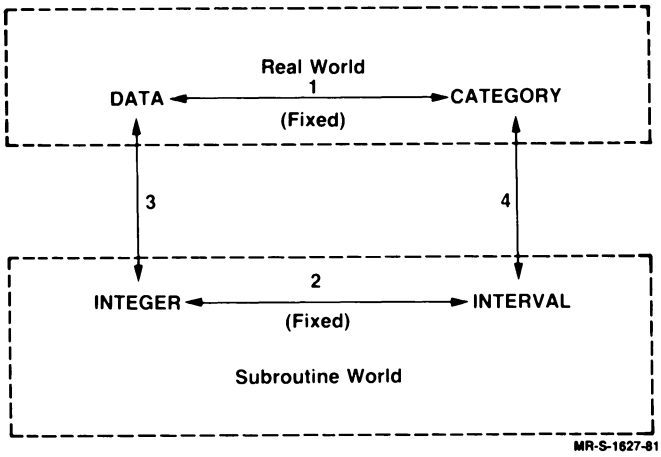
5.2.2 The Relation between Data and Categories

Data to be histogrammed are nonzero integers related in some way to the actual data they represent. If the data are numerical, such as measures of height, temperature, or time, this relation is immediately meaningful and obvious. However, the relation may be purely arbitrary, as when the data deal with an abstract condition that is represented by an integer to be processed by the subroutine; for example, if balls of different colors are being counted, an integer must be assigned to represent one color to distinguish it from other colors.

Data categories are also numerical; each is represented by an interval of integers. And like the data/integer relation, the relation between a category and the interval representing it may be obvious or completely arbitrary. Values assigned to these interrelated entities (Figure 5–1) must be

mutually consistent; for instance, an integer representing a data element must be in the numeric interval corresponding to the particular category to which that data element belongs. Figure 5–1 illustrates that relations 1 and 2 are fixed, and that once relation 3 or 4 is chosen, the remaining relation is no longer completely arbitrary.

Figure 5–1: Interrelation between DATA/CATEGORY and INTEGER/INTERVAL Concepts



5.2.3 How the Subroutine Interprets Single-Precision Numbers

Acceptable input and output values for the RHISTI subroutine are single-precision integers in the range of 0 to 65535; single-precision positive integers in FORTRAN have a range of 0 to 32767. This apparent conflict is actually a matter of interpretation and decimal representation of the negative numbers in FORTRAN; it is easily resolved, as we shall explain.

In theory a 16-bit binary number can represent a decimal number as large as 65535. In FORTRAN this range is divided into two parts: values from 0 to 32767 are interpreted and used as positive integers; values in the other half of the range are interpreted and used as negative integers. In FORTRAN negative numbers ranging from -32768 to -1 correspond directly to binary numbers ranging in decimal value from 32768 to 65535.

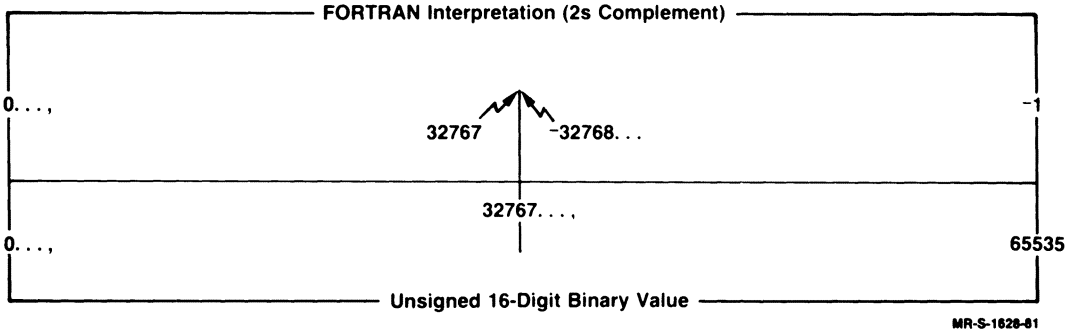
The RHISTI subroutine does not process or report values less than zero. Therefore all input and output values are treated as *unsigned* so that we can use the full positive range of the 16-bit word.

You may well ask "How does this interpretation of the data by the subroutine affect my particular application?" The answer is:

1. All input values and all resulting output data less than 32768 (2^{15}) are treated in exactly the same way by the subroutine and by FORTRAN (Figure 5–2).

2. If either your input data or the resulting output contain values greater than 32767 (but not greater than 65535), they are interpreted as negative by FORTRAN but not by the subroutine. For example, if the subroutine outputs a single-precision integer value of 40,000, FORTRAN interprets it as -25536.

Figure 5–2: Relation between FORTRAN Integers and Unsigned Binary Values



We shall now present a simple mechanism for converting a value that FORTRAN interprets as negative to a *floating-point number* equal to the unsigned interpretation of the value. This conversion may not always be necessary; some operations, such as addition and subtraction, are unaffected by the signs of the operands. The unsigned results of such an operation would be correct so long as the results were not less than 0 or greater than 65535. However, operations such as division and multiplication, or printing and typing, are affected by the FORTRAN interpretation of unsigned numbers larger than 32767 as negative numbers. The following two methods can be used to convert unsigned single-precision numbers to floating-point numbers:

```
R = (1 - (N/IABS(N))/2) * 65536. + N
```

or

```
R = N
IF (N.LT.0) R = 65536. + N
```

where R is the floating-point variable equivalent to the unsigned single-precision integer stored in N, and IABS(N) is a function available from the FORTRAN library.

To convert a floating-point number less than 2^{16} to an unsigned integer, you can use one of the following equations:

```
N = R - 65536. * IFIX (R/32768.)
```

or

```
N = R - 65536.
IF (R.LT.32768.) N = R
```

where R is again the floating-point variable, N the unsigned integer stored in N, and IFIX is a function available from the FORTRAN library.

5.3 How to Present Your Problem to the Subroutine

You can control the type and scope of the histogram(s) output by this subroutine by setting the appropriate parameter table elements. You must specify the number of data elements to be processed following each reference point and define your categories of interest so that the resulting interval histogram meets your requirements. You can also specify whether a corresponding frequency histogram is to be produced. An array of parameter values is used to pass your processing requirements to the subroutine (Section 5.4).

5.3.1 Number of Events to be Processed Following Each Reference Point

The number of events to be counted into the *interval* histogram following detection of each new reference point is specified by the fifth element of the parameter array (ITABLE(5)), Section 5.4. If this element is set to zero, all data elements following each reference point are counted into the interval histogram. Regardless of the value of ITABLE(5), all data elements following each reference point are counted into the *frequency* histogram, if one is produced.

This particular feature of the RHISTI subroutine is useful for applications in which statistical information involves a fixed number of data elements related to each reference point. In such applications the reference point often represents an event that causes, or provides the stimulus for, the data to be histogrammed. Such data elements often represent measures of time or space relative to the causing event.

For example, suppose you wish to study the arrival times of the first four pieces of fire-fighting apparatus at the scene of a fire after the alarm sounds. The sounding of the alarm is the stimulus, and is represented by a reference point in the data stream; the data elements represent the elapsed time between the sounding of the alarm and the arrival of the pieces of equipment. Thus, if ITABLE(5) is set to 4 and the data are input chronologically, the arrival times of the first four pieces of equipment are counted into the interval histogram. A corresponding frequency histogram would record the total number of pieces of equipment responding to the alarm.

5.3.2 Describing the Categories for the Interval Histogram

The numerical intervals representing the categories of interest to the subroutines are defined by means of a table of parameter values set by the user. The first three values in the parameter table define these intervals (see Section 5.3):

- The first element of the parameter table defines the lower limit of the first interval. Data values smaller than this limit do not fall into any predefined category and are reported separately (see Section 5.2.3).

- The second element of the parameter table defines the numerical length of each interval. Therefore the first interval spans values greater than or equal to the first element but smaller than the sum of the first and second elements. Symbolically, if the first element is I and the second is J, the first interval spans all data, $K1_i$, for which

  $I \leq K1_i < I + J$

  The second interval spans all values, $K2_i$, for which

  $I + J \leq K2_i < I + 2 \cdot J$

  and so on. *Note that there are J values in each interval.*

- The third element of the parameter table tells the subroutine how many intervals — starting with the value of the first element — to consider. This element also specifies the minimum dimension of the output array because each interval has a corresponding output array element. The implication of this value is that the last interval of interest spans all values, KN_i, for which

  $I + (N - 1) \cdot J \leq KN_i < I + N \cdot J$

  where N is the value of the third element and I and J are the values of the first and second elements, respectively.

5.3.3 Overflow and Underflow Count

Because the intervals of interest may not span the entire range of legal integer input, it is possible that the input data stream may contain values that do not fall within any specified category. The subroutine counts the number of data values outside the upper and lower limits of the specified categories. The number of data values that are smaller than the minimum specified in the first element of the parameter table (Section 5.3) is called the underflow count and is reported in the eighth element of the parameter table. The number of values that exceed the maximum value in the interval of interest is called the overflow count and is reported in the ninth element of the parameter table.

5.3.4 Frequency Histogram

In addition to the other processing characteristics described, an optional frequency or zeroth histogram can also be produced. It takes the form of an additional output array that indicates the level of activity between each pair of reference points. Specifically the *number* of input elements following the nth reference point and preceding the $n+1$th reference point are reported as the nth element of the frequency histogram array.

5.4 How to Call the RHISTI Subroutine

The symbolic name for the interval histogramming with reference points subroutine is RHISTI, and the general format of the FORTRAN call is:

```
CALL RHISTI(ITABLE,INPUT,IHGRAM[,IFREQ])
```

For reference, argument names in the call to RHISTI have been assigned arbitrarily. You may 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 either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results and no data is processed. The arguments are described in the following discussion.

ITABLE is an integer array of at least 16 elements used to:

- Transmit information from the user to the subroutine (ITABLE(1) through ITABLE(6))
- Return information from the subroutine to the user (ITABLE(7) through ITABLE(11))
- Transmit/report information to/from the subroutine on the optional frequency histogram (ITABLE(12) through ITABLE(15))
- Store information on an interim basis (by the subroutine) (ITABLE(16), ITABLE(17))

You must set the array elements that transmit information to the subroutine.

ITABLE(1)	The lower limit of the first interval (see Section 5.3.2)
ITABLE(2)	The specified interval length (see Section 5.3.2)
ITABLE(3)	The total number of contiguous intervals to be considered (see Section 5.3.2)
ITABLE(4)	The total number of array elements containing data (starting with the first element of the input array (INPUT))
ITABLE(5)	The number of data values to be processed after each reference point is detected; if set to zero, all values are processed (see Section 5.3.1)
ITABLE(6)	A value that must be set to zero before the initial call to the subroutine; it signals the subroutine to initialize the output array and other output elements in the parameter table. On subsequent calls to the subroutine, if a different segment of the same set of data is being processed, operation continues and there is no reinitialization.

The next group of elements is used to report information not included in the actual histogram:

ITABLE(7)	The number of reference points detected
ITABLE(8)	The underflow, or the count of the number of input data values that are smaller than ITABLE(1)

ITABLE(9) The overflow, or the count of the number of input data values that exceed the upper limit of the last interval; data values exceeding

ITABLE(1) + (ITABLE(2)·ITABLE(3))

are counted.

ITABLE(10) The number of output array elements that have exceeded the largest possible single-precision number (65535) (or overflow/underflow count) (see Section 5.2.3)

ITABLE(11) An element used to report error conditions:

= 0	Indicates no errors
= 1	Indicates that ITABLE(1) + ITABLE(2)· ITABLE(3) > 65535
= N	Indicates the ITABLE(N) is in error, for example, ITABLE(2) = 0
= -N	Indicates incorrect number of arguments in call

The next group of elements is used to transmit/report information concerning the optional frequency histogram. Note that you must set *ITABLE(12)*, and that ITABLE must be dimensioned to 17 regardless of whether a frequency histogram is produced.

ITABLE(12) Indicator of whether a frequency histogram should be produced:

= 0	Do not produce a frequency histogram
= 1	Produce a frequency histogram

If ITABLE(12) = 1, the fourth argument in the call statement must be present (IFREQ).

ITABLE(13) The number of elements in the array used to store the frequency histogram (the dimension of IFREQ in the CALL statement)

ITABLE(14) Number of frequency histogram elements already calculated; will be set by the subroutine to 0 for the initial call to RHISTI, when ITABLE(6) is 0. ITABLE(14) is updated continuously as input is processed by the subroutine. Because elements of this histogram are calculated sequentially, data from the storage array may be extracted each time the subroutine has processed all data in the input array and returned to the calling routine. If elements are extracted, ITABLE(14) may then be adjusted to reflect where the next element of the frequency histogram is to be stored.

ITABLE(15) The current partial count for the next (ITABLE(14) + 1) entry in the frequency histogram; each time another reference point is detected, ITABLE(14) is updated and the next entry is placed in the histogram.

The next elements are used only by the subroutine:

ITABLE(16), Elements used exclusively by the subroutine for internal storage

ITABLE(17)

INPUT is an integer array containing the data to be processed. All data are treated as positive and unsigned (see Section 5.2.3). All input elements equal to zero are interpreted as reference points (see Section 5.2.1). The number of array elements to be processed is ITABLE(4), and processing always begins with the first element of the array. Note, however, that all data need not be placed in the array at one time. Instead, one array of data can be processed, the array refilled with new data, and the subroutine called again to process the array of data a second time. It is possible to continue in this piecemeal fashion until all data have been processed. In real-time applications such a processing cycle becomes an ongoing function.

IHGRAM is an integer array to store the results of interval histogram processing; each array element is devoted exclusively to one of the numerical intervals that represents a single category. The order of the array elements corresponds to the numeric order of the intervals, that is, the Nth element of the output array will have a value equal to the number of data elements in the input stream that belong in the interval.

[ITABLE(1) + (N–1)·ITABLE(2)] to [ITABLE(1) + N·ITABLE(2) – 1]

IFFREQ is an array required only when ITABLE(12) = 1 and is used to store the frequency histogram data. This integer array must be at least as long as specified by ITABLE(13). Upon return, ITABLE(14) notifies the user how many elements in the array contain data.

If you remove data from the array before a subsequent call to the subroutine, you can adjust ITABLE(14) before reentry to reflect the additional storage space now available.

If the number of array elements available for storage is not large enough, that is, if ITABLE(14) becomes larger than ITABLE(13), the subroutine stops computing the frequency histogram (but continues to process interval histogram data). If this condition arises, frequency histogram data may be lost. To correct this problem, make ITABLE(14) smaller than ITABLE(13). ITABLE(14) indicates the number elements lost (ITABLE(14)–ITABLE(13)).

5.5 Input and Output — Using the Subroutine

As previously stated, all data for a particular histogram need not be available, or even known, before processing begins. Initialization takes place (all counters are set to zero) only when ITABLE(6) equals zero. Parameter table elements are also checked for correctness when ITABLE(6) equals zero.

Before the subroutine returns, it automatically changes ITABLE(6) so that if a subsequent call is made to process new data for the same histogram, processing continues as though no interruption had taken place. Thus an entire set of data for one histogram can be processed at one time, memory space permitting; or, if the user wishes, the data can be processed one segment at a time. The value assigned to ITABLE(4) should indicate the number of elements in the input array to be processed for a specific call.

The subroutine 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 that several data streams can be processed simultaneously. All pertinent information concerning the history of a data stream is contained in the ITABLE array rather than in the code for the subroutine. Imaginative use of the arguments in the subroutine call should make the subroutine functionally compatible with any application that uses interval histogramming of data relative to reference points.

5.6 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the interval histogramming with reference points subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

5.6.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS (KE11–E) hardware or any other floating-point option available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 121 words if DPR$ is not enabled, and by approximately 123 words if DPR$ is enabled.

5.6.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE (KE11) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 87 words.

5.6.3 DPR$ (Double-Precision Integers)

If the range of data values to be histogrammed exceeds 65535 (2^{16}–1), this conditional option must be enabled. Note that if you are using a copy of the subroutine in which the symbol DPR$ has been enabled, you must provide your data to the subroutine as follows:

- All input data must be double-precision integer, that is, the second argument in the FORTRAN CALL statement, INPUT, must be an INTEGER*4 array (or equivalent).

- Because this option expands the possible range of input values by a factor of 2^{16}, the range of variables used to define the exact range of interest must also be expanded by 2^{16}. Therefore the first argument in the FORTRAN CALL statement, ITABLE, must also be an INTEGER*4 array or the equivalent. ITABLE(1) and ITABLE(2) can then select the range of interest.

Enabling this option has no effect on the output because the range of input values does not affect the size of output values.

If DPR$ is enabled, the size of the subroutine increases by approximately 74 words.

5.7 Examples of Input/Output Variation Using the RHISTI Subroutine

The three examples presented here process equivalent sets of data in similar ways but focus on different capabilities of the RHISTI subroutine.

Example 1 is based on the distributed version of the subroutine. A set of 10,000 random integers ranging from 15 to 225 is processed, 500 at a time. Each sequential set of input points is stored in alternate halves of the INPUT array to simulate real-time applications, which conserve processing time by collecting and processing data in parallel.

NOTE

If you use FORTRAN 77 and you want to duplicate the terminal output for the example programs, replace the standard random-number generator in F4POTS with F4PRAN.OBJ. Terminal output for the example programs is based on the FORTRAN IV random-number generator. The FORTRAN 77 random-number generator is different from that for FORTRAN IV and will not produce the same output. See Section B.1.

Categories of interest comprise 40 intervals of five integers each; the minimum value for the first interval is 20. When processing is complete, the following output is typed on the terminal:

- The total count of data items belonging to each category, that is, the interval histogram (IHGRAM)
- The total number of reference points detected (ITABLE(7))
- The number of data items smaller than the total interval of interest, that is, the underflow count (ITABLE(8))
- The number of data items larger than the total interval of interest, that is, the overflow count (ITABLE(9))
- The total number of output counters that have exceeded 65535 (ITABLE(10))

RHISTI Example #1

RHISTI Example #1

```
(1)     DIMENSION ITABLE(17),INPUT(1000),IHGRAM(40)
        EQUIVALENCE (ITABLE(1),IS),(ITABLE(2),IW)

(2)     DATA ITABLE/20,5,40,500,13*0/
        DATA I1,I2/0,0/

(3)     DO 2 J=2,21

(4)     N=500*MOD(J,2)+1

(5)     DO 1 I=N,N+499
        K=RAN(I1,I2)*210+15
        M=K/80
        IF(M*80.EQ.K) K=0
    1   INPUT(I)=K

(6)     CALL RHISTI(ITABLE,INPUT(N),IHGRAM)
        IF(ITABLE(11).EQ.0) GO TO 2
        TYPE 1000,ITABLE(11)
 1000   FORMAT(' ERROR INDICATOR = ',I3)
        STOP

(7)   2 CONTINUE

(8)     TYPE 900
  900   FORMAT(1H1,T30,'RHISTI Example #1',//)
        TYPE 2000
 2000   FORMAT(28X,'RESULTING HISTOGRAM'//,
     1  4('     INTERVAL COUNT')/)
        TYPE 3000,((N-1)*IW+IS,N*IW+IS,IHGRAM(N),N=1,40)
 3000   FORMAT(4(I7,'-',I4,I6))
        TYPE 4000,(ITABLE(I),I=7,10)
 4000   FORMAT(//,' NO. OF REFERENCE POINTS = ',I3,/,
     1  ' UNDERFLOW COUNT = ',I3,/,' OVERFLOW COUNT = ',I3,/,
     2  ' NO. OF COUNTERS WHICH OVERFLOWED = ',I2,//)
        CALL EXIT
        END
```

① Define array elements and their sizes; use equivalence of convenience.

② Set parameter table elements: there are 40 intervals of interest, starting at 20, and comprising five elements each; input array contains 500 data elements; set ITABLE(6) to zero to signal initialization (no frequency histogram is produced); set number of data values to be processed after each reference point to zero to signify that all values are to be processed.
Initialize random number generation variables.

③ Provide 20 sets of data (500 points each) for processing.

④ Determine which half of the input array is to receive the next set of input data.

⑤ Determine next set of random numbers for processing; produce some zero values (reference points).

⑥ Process next set of input values:

ITABLE contains parameter table
INPUT contains input for processing, starting at subscript N
IHGRAM is array where interval histogram is stored

⑦ End of data generation and processing loop

⑧ Type output:

Interval histogram (IHGRAM)
Number of reference points detected (ITABLE(7))
Underflow (ITABLE(8)); overflow (ITABLE(9)); counter overflow count (ITABLE(10))

Terminal Output

```
                    RHISTI Example #1

                  RESULTING HISTOGRAM

  INTERVAL COUNT    INTERVAL COUNT    INTERVAL COUNT    INTERVAL COUNT

   20-  25   214     25-  30   232     30-  35   224     35-  40   239
   40-  45   248     45-  50   253     50-  55   215     55-  60   228
   60-  65   243     65-  70   226     70-  75   242     75-  80   239
   80-  85   199     85-  90   234     90-  95   216     95- 100   253
  100- 105   248    105- 110   245    110- 115   225    115- 120   231
  120- 125   235    125- 130   212    130- 135   240    135- 140   225
  140- 145   229    145- 150   220    150- 155   254    155- 160   247
  160- 165   191    165- 170   229    170- 175   264    175- 180   250
  180- 185   237    185- 190   225    190- 195   226    195- 200   225
  200- 205   237    205- 210   213    210- 215   247    215- 220   236

NO. OF REFERENCE POINTS = 114
UNDERFLOW COUNT = 240
OVERFLOW COUNT = 228
NO. OF COUNTERS WHICH OVERFLOWED = 0
```

RHISTI Example #2

Example 2 is identical to Example 1 except that a frequency histogram is produced in addition to the results output in Example 1.

RHISTI Example #2

```
(1)       DIMENSION ITABLE(17),INPUT(1000),IHGRAM(40),IFREQ(120)
          EQUIVALENCE (ITABLE(1),IS),(ITABLE(2),IW)

(2)       DATA ITABLE/20,5,40,500,7*0,1,120,4*0/
          DATA I1,I2/0,0/

(3)       DO 2 J=2,21

(4)       N=500*MOD(J,2)+1

          DO 1 I=N,N+499
          K=RAN(I1,I2)*210+15
(5)       M=K/80
          IF(M*80.EQ.K) K=0
        1 INPUT(I)=K

          CALL RHISTI(ITABLE,INPUT(N),IHGRAM,IFREQ)
          IF(ITABLE(11).EQ.0) GO TO 2
(6)       TYPE 1000,ITABLE(11)
     1000 FORMAT(' ERROR INDICATOR = ',I3)
          STOP

(7)     2 CONTINUE

          TYPE 900
      900 FORMAT(1H1,T30,'RHISTI Example #2',//)
          TYPE 2000
     2000 FORMAT(28X,'RESULTING HISTOGRAM'//,
         1 4('     INTERVAL COUNT')/)
          TYPE 3000,((N-1)*IW+IS,N*IW+IS,IHGRAM(N),N=1,40)
     3000 FORMAT(4(I7,'-',I4,I6))
          TYPE 4000,(ITABLE(I),I=7,10)
     4000 FORMAT(//,' NO. OF REFERENCE POINTS = ',I3,/,
(8)      1 ' UNDERFLOW COUNT = ',I3,/,' OVERFLOW COUNT = ',I3,/,
         2 ' NO. OF COUNTERS WHICH OVERFLOWED = ',I2,//)
          TYPE 5000
     5000 FORMAT(///20X,'CORRESPONDING FREQUENCY HISTOGRAM'//,
         1 5('    ENTRY COUNT')/)
          I=ITABLE(14)
          IF(ITABLE(14).GT.ITABLE(13)) I=ITABLE(13)
          TYPE 7000,(N,IFREQ(N),N=1,I)
     7000 FORMAT(5(I8,I6))
          CALL EXIT
          END
```

① Define array elements and their sizes; use equivalence for convenience.

② Set parameter table elements: there are 40 intervals of interest, starting at 20, and comprising five elements each; input array contains 500 elements; set number of data values to be processed after each reference point to zero to signify that all values are to be processed; set ITABLE(6) to zero to signal initialization; indicate that frequency histogram should be produced and set number of elements in frequency histogram array.
Initialize random number generation variables.

③ Provide 20 sets of data (500 points each) for processing.

④ Determine which half of the input array is to receive the next set of input data.

⑤ Determine next set of random numbers for processing; produce some zero values (reference points).

⑥ Process next set of input values:

ITABLE contains parameter table
INPUT contains input for processing, starting at subscript N
IHGRAM is array where interval histogram is stored
IFREQ is array where frequency histogram is stored

Check for error on return.

⑦ End of data generation and processing loop

⑧ Type output:

Interval histogram (IHGRAM)
Number of reference points detected (ITABLE(7))
Underflow (ITABLE(8)); overflow (ITABLE(9)); counter overflow count (ITABLE(10))
Frequency histogram (IFREQ(N), where N ranges from 1 to ITABLE(14))

Terminal Output

RHISTI Example #2

RESULTING HISTOGRAM

INTERVAL	COUNT	INTERVAL	COUNT	INTERVAL	COUNT	INTERVAL	COUNT
20- 25	214	25- 30	232	30- 35	224	35- 40	239
40- 45	248	45- 50	253	50- 55	215	55- 60	228
60- 65	243	65- 70	226	70- 75	242	75- 80	239
80- 85	199	85- 90	234	90- 95	216	95- 100	253
100- 105	248	105- 110	245	110- 115	225	115- 120	231
120- 125	235	125- 130	212	130- 135	240	135- 140	225
140- 145	229	145- 150	220	150- 155	254	155- 160	247
160- 165	191	165- 170	229	170- 175	264	175- 180	250
180- 185	237	185- 190	225	190- 195	226	195- 200	225
200- 205	237	205- 210	213	210- 215	247	215- 220	236

NO. OF REFERENCE POINTS = 114
UNDERFLOW COUNT = 240
OVERFLOW COUNT = 228
NO. OF COUNTERS WHICH OVERFLOWED = 0

CORRESPONDING FREQUENCY HISTOGRAM

ENTRY	COUNT	ENTRY	COUNT	ENTRY	COUNT	ENTRY	COUNT	ENTRY	COUNT
1	19	2	119	3	80	4	206	5	31
6	46	7	2	8	124	9	24	10	40
11	91	12	130	13	17	14	72	15	433
16	161	17	101	18	44	19	9	20	13
21	95	22	51	23	29	24	96	25	11
26	73	27	126	28	125	29	75	30	11
31	323	32	67	33	52	34	28	35	224
36	68	37	37	38	42	39	36	40	44
41	21	42	22	43	33	44	10	45	1
46	34	47	88	48	209	49	84	50	10
51	47	52	228	53	16	54	50	55	47
56	192	57	47	58	89	59	37	60	6
61	164	62	69	63	18	64	90	65	98
66	87	67	133	68	158	69	72	70	10
71	8	72	29	73	151	74	36	75	134
76	149	77	18	78	376	79	25	80	81
81	68	82	18	83	103	84	245	85	92
86	3	87	2	88	147	89	61	90	139
91	15	92	46	93	13	94	181	95	38
96	113	97	15	98	85	99	306	100	42
101	9	102	166	103	70	104	275	105	5
106	155	107	450	108	13	109	82	110	60
111	77	112	26	113	67				

Example 3 is identical to Example 2 except that each input data element is increased by a factor of 65536(2^{16}) and must therefore be represented as a double-precision integer. The distributed version of the subroutine cannot process double-precision integers. The subroutine must be built from the source file with DPR$ enabled in order to produce an object file capable of processing double-precision input (see Section 5.6.3 and Section 1.1).

Example 3 illustrates how the proper version of the subroutine processes double-precision integers. Input values are exactly 65536 times those processed in previous examples, and output is analogous to that in Example 2.

Note that when double-precision integers are processed, both the input array and certain of the parameter table values must be double-precision integer (data type INTEGER*4). Because the 1st and 2nd parameters of ITABLE are increased by 65536, categories of interest are increased so that their relation to input remains the same as in Example 2. Thus output from Example 3 is identical to that from Example 2.

Because this example uses FORTRAN IV, double-precision arrays ITABLD and INPUTD are equivalent to single-precision arrays ITABLE and INPUT of the same length; thus values for the double-precision elements can be set via their single-precision equivalents.

RHISTI Example #3

```
(1)       DIMENSION ITABLE(34),INPUT(2000),IHGRAM(40),IFREQ(120)
          INTEGER*4 ITABLD(17),INPUTD(1000)
          EQUIVALENCE (ITABLE(2),IS),(ITABLE(4),IW),(ITABLE,ITABLD),
         1            (INPUT,INPUTD)

(2)       DATA ITABLE/0,20,0,5,40,0,500,15*0,1,0,120,9*0/
          DATA INPUT/2000*0/
          DATA I1,I2/0,0/

(3)       DO 2 J=2,21

(4)       N=500*MOD(J,2)+1

(5)       DO 1 I=N*2,(N+499)*2,2
          K=RAN(I1,I2)*210+15
          M=K/80
          IF(M*80.EQ.K) K=0
         1 INPUT(I)=K

(6)       CALL RHISTI(ITABLD,INPUTD(N),IHGRAM,IFREQ)
          IF(ITABLE(21).EQ.0) GO TO 2
          TYPE 1000,ITABLE(21)
     1000 FORMAT(' ERROR INDICATOR = ',I3)
          STOP

(7)      2 CONTINUE

(8)       TYPE 900
          900 FORMAT(1H1,T30,'RHISTI Example #3',//)
          TYPE 2000
     2000 FORMAT(28X,'RESULTING HISTOGRAM'//,
         1 4(' INTERVAL',6X),/,4('  X 2**16 COUNT')/)
          TYPE 3000,((N-1)*IW+IS,N*IW+IS,IHGRAM(N),N=1,40)
     3000 FORMAT(4(I7,'-',I4,I6))
          TYPE 4000,(ITABLE(I),I=13,19,2)
     4000 FORMAT(//,' NO. OF REFERENCE POINTS = ',I3,/,
         1 ' UNDERFLOW COUNT = ',I3,/,' OVERFLOW COUNT = ',I3,/,
         2 ' NO. OF COUNTERS WHICH OVERFLOWED = ',I2,//)
          TYPE 5000
     5000 FORMAT(///20X,'CORRESPONDING FREQUENCY HISTOGRAM'//,
         1 5(' ENTRY COUNT')/)
          I=ITABLE(27)
          IF(ITABLE(27).GT.ITABLE(25)) I=ITABLE(25)
          TYPE 7000,(N,IFREQ(N),N=1,I)
     7000 FORMAT(5(I8,I6))
          CALL EXIT
          END
```

① Define array elements and their sizes; use equivalence for convenience; note that ITABLD and INPUTD are double-precision.

② Set parameter table elements: there are 40 intervals of interest, starting at $20 \cdot 2^{16} = 1{,}310{,}720$, and comprising $5 \cdot 2^{16} = 327{,}680$ elements each; input array contains 500 elements; set number of data values to be processed after each reference point; set ITABLE(6) to zero to signal initialization; indicate that frequency histogram should be produced, and set number of elements in frequency histogram array.
Clear high and low parts of double-precision input.
Initialize random number generation variables.

③ Provide 20 sets of data (500 double-precision points each) for processing.

④ Determine which half of the input array is to receive the next set of input data.

⑤ Determine next set of random numbers for processing; place random number range from 15 to 255 in high order part of the double-precision integer array ITABLD via ITABLE, effectively producing random numbers in the range of $15 \cdot 2^{16}$ to $225 \cdot 2^{16}$; produce random zero values (reference points).

⑥ Process next set of input values:

ITABLD contains parameter table
INPUT contains input for processing, starting at subscript N
IHGRAM is array where interval histogram is stored
IFREQ is array where frequency histogram is stored

Check for error on return.

⑦ End of data generation and processing loop

⑧ Type output:

Interval histogram (IHGRAM)
Number of reference points detected (ITABLD(7))
Underflow (ITABLD(8)); overflow (ITABLD(9)); counter overflow count (ITABLD(10))
Frequency histogram (IFREQ(N), where N ranges from 1 to ITABLD(13) = ITABLE(25))

Terminal Output

```
                    RHISTI Example #3

                    RESULTING HISTOGRAM

INTERVAL          INTERVAL          INTERVAL          INTERVAL
 X 2**16 COUNT     X 2**16 COUNT     X 2**16 COUNT     X 2**16 COUNT

 20-  25   214     25-  30   232     30-  35   224     35-  40   239
 40-  45   248     45-  50   253     50-  55   215     55-  60   228
 60-  65   243     65-  70   226     70-  75   242     75-  80   239
 80-  85   199     85-  90   234     90-  95   216     95- 100   253
100- 105   248    105- 110   245    110- 115   225    115- 120   231
120- 125   235    125- 130   212    130- 135   240    135- 140   225
140- 145   229    145- 150   220    150- 155   254    155- 160   247
160- 165   191    165- 170   229    170- 175   264    175- 180   250
180- 185   237    185- 190   225    190- 195   226    195- 200   225
200- 205   237    205- 210   213    210- 215   247    215- 220   236

NO. OF REFERENCE POINTS = 114
UNDERFLOW COUNT = 240
OVERFLOW COUNT = 228
NO. OF COUNTERS WHICH OVERFLOWED = 0

                    CORRESPONDING FREQUENCY HISTOGRAM

ENTRY COUNT   ENTRY COUNT   ENTRY COUNT   ENTRY COUNT   ENTRY COUNT

    1    19       2   119       3    80       4   206       5    31
    6    46       7     2       8   124       9    24      10    40
   11    91      12   130      13    17      14    72      15   433
   16   161      17   101      18    44      19     9      20    13
   21    95      22    51      23    29      24    96      25    11
   26    73      27   126      28   125      29    75      30    11
   31   323      32    67      33    52      34    28      35   224
   36    68      37    37      38    42      39    36      40    44
   41    21      42    22      43    33      44    10      45     1
   46    34      47    88      48   209      49    84      50    10
   51    47      52   228      53    16      54    50      55    47
   56   192      57    47      58    89      59    37      60     6
   61   164      62    69      63    18      64    90      65    98
   66    87      67   133      68   158      69    72      70    10
   71     8      72    29      73   151      74    36      75   134
   76   149      77    18      78   376      79    25      80    81
   81    68      82    18      83   103      84   245      85    92
   86     3      87     2      88   147      89    61      90   139
   91    15      92    46      93    13      94   181      95    38
   96   113      97    15      98    85      99   306     100    42
  101     9     102   166     103    70     104   275     105     5
  106   155     107   450     108    13     109    82     110    60
  111    77     112    26     113    67
```

FAST FOURIER TRANSFORM (FFT) SUBROUTINE

FORMAT:

CALL FFT(IERROR,N,IREAL,IMAG,INVRS,ISCALE)

Where:

IERROR is an integer variable used to report errors.

N is an integer variable specifying the number of elements to be transformed.

IREAL is an integer array containing the real portion of input or output data.

IMAG is an integer array containing the imaginary portion of the input or output data.

INVRS is an integer variable indicating whether FFT is to perform a forward or inverse transform.

0 = a forward transform
1 = an inverse transform

ISCALE is an integer variable set by FFT to indicate scaling factor (number of times results have been divided by 2).

FILE NAMES:

F4FFT.MAC (source file); F4FFT.OBJ (object file)

OPTIONS:

- EIS (Extended Instruction Set — KE11-E)
- EAE (Extended Arithmetic Element — KE11)
- F.MAXN (Maximum Array Size)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If the following options are enabled:

	NONE	EIS	EAE
F.MAXN=1024	798	662	668
F.MAXN=2048	1004	918	924
F.MAXN=4096	1516	1430	1436
F.MAXN=8192	2540	2454	2460

TYPICAL EXECUTION SPEED:

With PDP–11/34 and EIS enabled: 900 Points/second.

With PDP–11/03 and EIS enabled: 320 Points/second.

Chapter 6
The Fast Fourier Transform (FFT) Subroutine

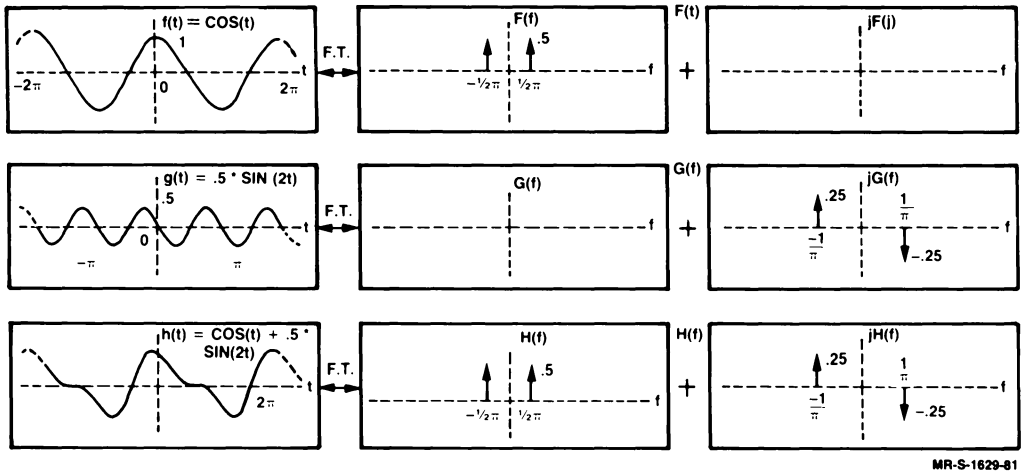
The fast Fourier transforms (FFT) subroutine provides an efficient means of numerically approximating analytical or continuous Fourier transforms.

This chapter discusses the fast Fourier transform (FFT) subroutine, describes the subroutine's FORTRAN call, and outlines the requirements for the subroutine's use. Before discussing the FFT subroutine, however, this chapter presents an introduction to Fourier transforms.

6.1 An Introduction to Fourier Transforms

The Fourier transform converts functions in the time domain to expressions in the frequency domain. Figure 6–1 shows the effect of the Fourier transform.

Figure 6–1: The Fourier Transform



The Fourier transform separates a function of time into a set of sinusoids that are represented by a complex-valued frequency function. Each non-zero value of the frequency function indicates that a sinusoid at that frequency is a component of the original time function. The value of the frequency function for a given frequency is equal to the amplitude of the associated sinusoid.

The real components of the complex-valued frequency function indicate sinusoids that are even functions, that is, are functions having the property

$$f(t) = f(-t)$$

Conversely, the imaginary components of the complex-valued frequency function indicate sinusoids that are odd functions; that is, they are functions having the property

$$f(t) = -f(-t)$$

There is also an inverse Fourier transform. The inverse Fourier transform converts a function in the frequency domain to an expression in the time domain.

6.1.1 Mathematical Definition of the Fourier Transform (CFT)

The analytical expression of the Fourier transform is called the continuous Fourier transform (CFT). Mathematically, the CFT is represented as:

$$H(f) = \int_{-\infty}^{\infty} h(t) \cdot e^{-j \cdot 2 \cdot \pi \cdot f \cdot t} \, dt$$

The inverse CFT function is represented mathematically as

$$h(t) = \int_{-\infty}^{\infty} H(f) \cdot e^{j \cdot 2 \cdot \pi \cdot t \cdot f} \, df$$

Where:

	j	is $\sqrt{-1}$.
	h(t)	is the time function to be transformed.
	H(f)	is the Fourier transform of h(t).
	t	is time.
	f	is frequency.

6.1.2 Mathematical Definition of the Discrete Fourier Transform (DFT)

A digital computer cannot perform the integration indicated by the expression for the CFT. A digital computer only can deal with discrete data

points. Thus, the FFT subroutine must use a method known as the discrete Fourier transform (DFT) to approximate the CFT at discrete frequencies.

The DFT does not process a continuous function. Instead, it processes discrete points that give only an approximation of the continuous function.

The DFT is represented mathematically as

$$H\left(\frac{n}{N \cdot dt}\right) = \sum_{k=0}^{N-1} h(k \cdot dt) \cdot e^{-j \cdot 2 \cdot \pi \cdot k \cdot n/N} \text{ for } n = 0,1,2,...,N-1$$

The mathematical expression of the inverse DFT is

$$h(k \cdot dt) = \frac{1}{N} \cdot \sum_{n=0}^{N-1} H\left(\frac{n}{N \cdot dt}\right) \cdot e^{j \cdot 2 \cdot \pi \cdot n \cdot k \, N} \text{ for } k = 0,1,2,...,N-1$$

Where (in addition to those terms explained for the CFT in Section 6.1.1): N is the number of sample input values from h(t) or H(f).

Although the FFT subroutine uses the DFT algorithm as a model, it also takes advantage of certain computational shortcuts to reduce the time required to evaluate the resulting values. Because of this computational time reduction, the shortcut method used by the subroutine is known as the fast Fourier transform (FFT).

The results of FFT algorithm differ from the results of DFT algorithm only by a known scale factor. (See Section 6.3 for a description of the scale factor.)

6.2 Comparing the Continuous and Discrete Fourier Transform

In general, largely because of different inputs, the results of the FFT and the DFT only *resemble* the expected results of the CFT. Only under certain special conditions do the FFT and the DFT produce the same results as the CFT for corresponding frequencies.

The following sections describe the special conditions in which the FFT and the CFT produce identical output. They also explain, without formal proof, how and why some discrepancies arise between the CFT and the FFT.

NOTE

These sections do not provide an exhaustive explanation of the discrete approximations to the Fourier transform, the discrepancies that generally result, or the methods needed to minimize these discrepancies. For such a description, see a good, general text such as *The Fast Fourier Transform* by E. Oran Brigham.

6.2.1 Comparison of FFT and CFT Input

The input to the DFT or the FFT for a function differs significantly from the input to the CFT for the same function. Input to the CFT is typically a continuous function defined for all values of time. Input to the FFT for the "same" function must be a finite set of discrete samples of the input function to the CFT.

This inherent difference between the actual input to the CFT and the actual input to the FFT is the most significant element causing discrepancies between the expected and the actual results of the FFT.

To determine what the expected results of the FFT should be, you should examine the results of the analytic CFT performed on the actual input to the FFT. Mathematically, an expression for the input to the FFT would be:

$$g(t) = h(t) \cdot \Delta_0(t) \cdot x(t)$$

Where:

- t is time.
- $g(t)$ is the input to the FFT.
- $\cdot$ is the multiplication operation.
- $h(t)$ is the original assumed input.
- $\Delta_0(t)$ is the sampling function.

  The mathematical representation of the sample function is

  $$\sum_{k=-\infty}^{\infty} \delta(t - k \cdot dt)$$

  where:
  - δ is the impulse or the Dirac delta function.
  - dt is the time interval between samples.
- $x(t)$ is the truncation function.

The truncation function defines the total length of the sampling period. If the function h(t) is to be sampled during a time duration, T_0, then:

$$x(t) = 1, \text{ for } \frac{-dt}{2} \leq t < T_0 - \frac{d}{2}$$

and

$x(t) = 0$ for all other values of t.

Figure 6–2 shows the relationship between FFT and CFT input. It also provides a graphic representation of the relationship of h, Δ_0, x and the resulting g (t).

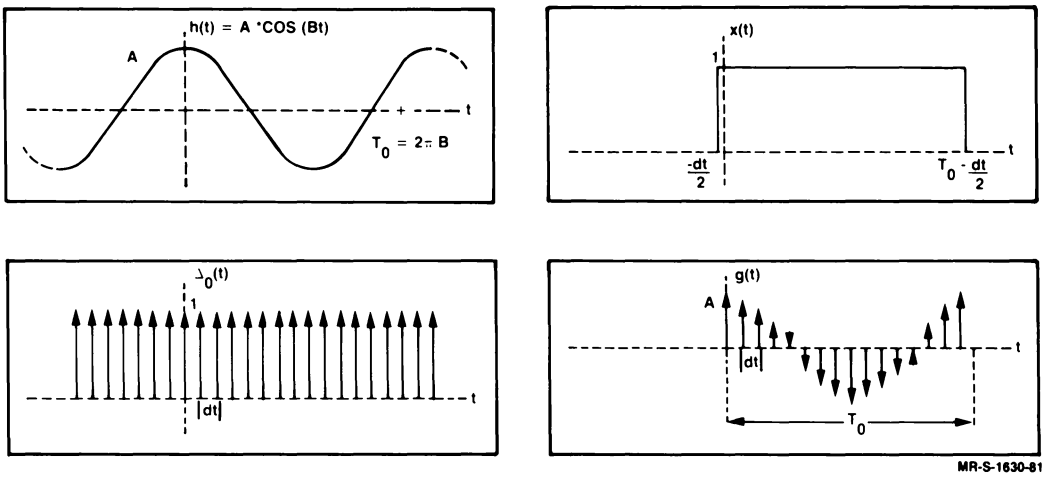
In Figure 6–2,

$h(t) = A \cdot COS(Bt)$

$T_0 = 2\pi / B$

$dt = T_0 / 16$

Figure 6–2: The Relationship Between FFT Input and CFT Input



6.2.2 Comparison of FFT and CFT Output

Given the mathematical expression for g(t), the actual CFT of g(t) can be derived analytically. Three facts allow this derivation.

1. The CFT of h(t) is assumed to be known.
2. The Fourier transforms of $\Delta_0(t)$ and x(t) are well known.
3. The Fourier transform of the product of two functions is demonstrably equal to the convolution of the transforms of the functions.

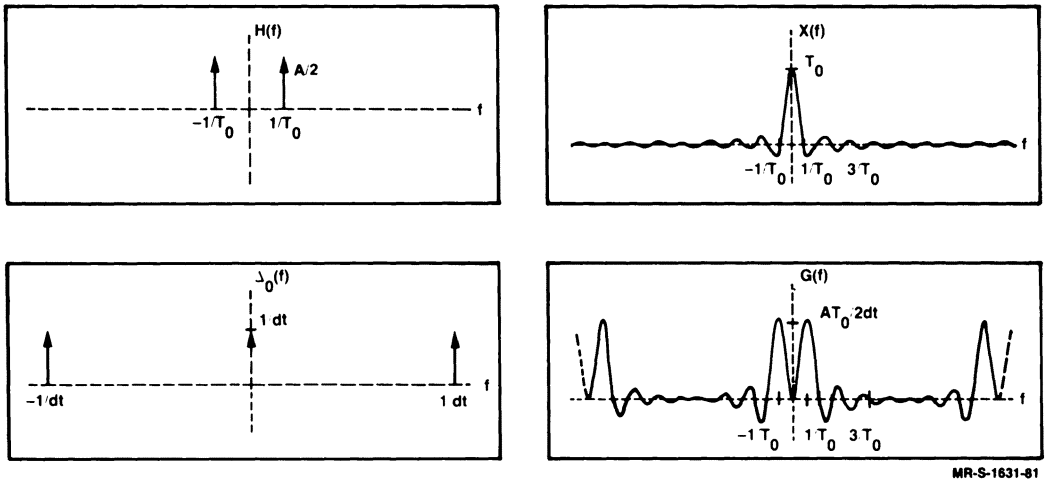
Thus, a mathematical expression for the CFT of the actual input to the FFT is

$G(f) = H(f) * \Delta_0(f) * X(f)$

Where:	$G(f)$	is the Fourier transform (CFT) of g (t).
	*	is the convolution function.
	$H(f)$	is the Fourier transform of h (t).
	$\Delta_0(f)$	is the Fourier transform of Δ_0 (t).
	$X(f)$	is the Fourier transform of x (t).
		Its mathematical representation is
		$[T_0 \cdot SIN(\pi \cdot T_0 \cdot f)] / (\pi \cdot T_0 \cdot f)$
	f	is frequency.

Figure 6–3 illustrates the transformed output of the input functions shown in Figure 6–2.

Figure 6–3: Output from the FFT



As Figure 6–3 shows, the expected output from the DFT or FFT — a CFT of the continuous function h(t) — bears little resemblance to the results that should be obtained from the CFT of the discrete samples of h(t) that are the actual input to the FFT.

The result of the CFT, G(f), does not appear to be good enough to make the FFT a useful algorithm. However, the actual result of the FFT is not G(f) but rather is a set of discrete samples of G(f).

When N input values are sampled with a time interval of dt, that is:

$T_0 = N \cdot dt$

the sample values of G (f) that result from the FFT are at frequencies of

$$0, \frac{1}{T_0}, \frac{2}{T_0}, \ldots, \frac{n-1}{T_0}$$

For convenience, $\frac{1}{T_0}$ is referred to as df.

Figure 6–4 shows G(n · df) returned by the fast Fourier transform.

As shown in Figure 6–4, the output from the FFT (indicated by the large dots) is:

G (n · df), for $0 < n < N - 1$

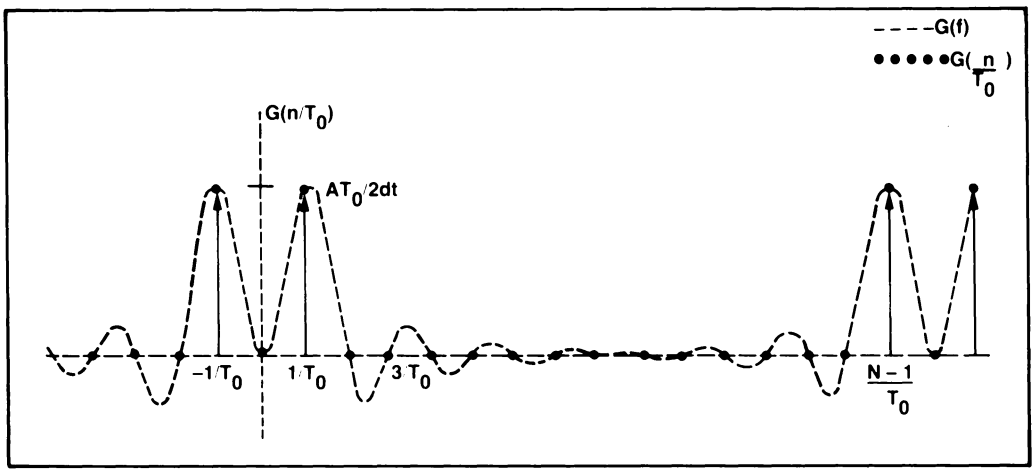
and the values yielded are

$$G(n \cdot df) = \frac{A \cdot T_0}{2 \cdot dt} \quad \text{for } n = 1 \text{ and } n = N - 1$$

and

G (n · df) = 0 for other n between 0 and N - 1

Figure 6–4: G(n • df) Returned by the FFT



Thus, the sampled results of G(f) returned by the FFT seem to correspond well with those of H(f), the expected result for frequencies in the range 0 to [(N - 1)/2 • df]. However, in the frequency range [N/2 • df] to [(N - 1) • df], even the sampled results of the FFT still do not render a close approximation of H(f).

Nevertheless, because of the periodicity of G(f) caused by the convolution of H(f) with X(f), this problem area can be interpreted as part of the desired solution. As Figure 6–3 shows, G(f) is periodic. The period of G(f) is equal to N • df. Thus, G(f) in the range [(-N)/2 • df] to -df is identical with G(f) in the range [N/2 • df] to [N•1 - df]. Therefore, the last N/2 values returned by the FFT can be interpreted as representing this negative frequency range. In this case, if the output is considered to be associated with the following frequency range:

- [N / 2 • df] to [(N - 1) / 2 • df]

or

- [1 / (2 • dt)] to 1 / (2 • dt)

the FFT gives a close approximation of H(f).

6.2.3 Similarities and Discrepancies between FFT and CFT Results

The case illustrated in Figure 6–3 is an unusual one. Normally, the actual results of the FFT do not match the expected results to within a scale factor. In fact, the example illustrated in Figure 6–3 represents the one class of functions and uses the one sampling technique for which the results of the FFT and the results of the CFT agree within a known scale factor.

Three conditions must be satisfied to get such agreement between the results of the FFT and the results of the CFT.

1. The function to be transformed must be periodic and band-limited; that is, the function's highest frequency component must be finite.

2. The function to be transformed must be truncated at *exactly* one non-zero integer multiple of the function's period.

3. The function to be transformed must be sampled at a rate greater than twice the function's highest frequency component.

All of these conditions are met in the example illustrated in Figure 6–3.

1. The expression, $h(t) = A \cdot COS(Bt)$ satisfies the first condition.

2. Because $T_0 = 2 \cdot \pi/B$, the expression x(t) satisfies the second condition.

3. Because $dt = (2 \cdot \pi)/(B \cdot 16)$ (implying a sampling rate of $(B \cdot 16)/(2 \cdot \pi)$, dt is greater than twice $B/2 \cdot \pi$, the highest frequency component of h(t)), and the expression $\Delta_0(t)$ satisfies the third condition.

If these three conditions are not met, discrepancies in the results begin to accumulate. For example, if the third requirement is not met — if the chosen value for dt is too large — then the period for G(f) is too short. This shortened period causes aliasing, that is, overlapping representations of the expected transform that produce false frequency contributions.

If the second requirement is not met, that is, if T or N is not chosen so that the original function is truncated at an exact integer multiple of the period of the h(t), the results of the FFT show a distorting effect known as leakage.

If T is an integer multiple of the period of h(t), it causes the form of X(f) to be such that the samples of G(f) returned by the FFT that are not related to H(f) coincide with the zero's of X(f) (see Figure 6–4). However, if T is not an integer multiple of the period of h(t), the results returned by the FFT are due in large part to the transform of the function x(t) and not just to the transform of the intended function h(t).

If the first requirement is not met, then either the second or the third condition cannot be met. This failure causes the associated discrepancies in the results.

In general, these discrepancies cannot be totally resolved. But you should recognize their causes and possible effects so that you can use this FFT subroutine effectively.

6.3 Scaling the Results of the FFT Subroutine

As mentioned in Section 6.1, the FFT subroutine uses a scaling procedure that is not used in the DFT algorithm.

There are two types of scaling procedure that are especially applicable to the FFT subroutine. The first type is peculiar to the subroutine itself and not to the FFT or DFT algorithms. The second type relates the actual subroutine results to those expected from the original CFT function.

The following sections describe both types of scaling procedure.

6.3.1 Internal Subroutine Scaling Procedure

The FFT subroutine uses internal scaling procedures that cause it to vary from the DFT algorithm in two ways.

1. To maximize speed, the FFT subroutine performs all operations in integer arithmetic. Thus the input and output of the subroutine are single-precision, integer values.

   However, the FFT subroutine generally cannot represent its output in the legal integer range (∓32767) of FORTRAN. Because of this limitation, the FFT subroutine must scale the output internally to fit the FORTRAN legal range.

   The fast Fourier transform subroutine indicates the scaling factor it uses in the argument ISCALE (see Section 6.5). To obtain the actual results of the FFT, you must scale all data by multiplying them by 2**ISCALE.

2. When performing an inverse transform, the FFT does not include the factor 1/N that appears in the expression for the inverse DFT. Thus, you must multiply the output of the inverse FFT by 1/N to duplicate the results of the inverse DFT.

6.3.2 Relational Scaling Procedure

The relational scaling procedure is relatively simple. Its rationale however is complicated and must be explained in some detail. Thus, a review of some of the information presented in previous sections is appropriate.

In Section 6.2.1, the continuous function of time, h(t) was set equal to A · COS(Bt). It was then stated that:

$$h(t) = A \cdot \cos(B \cdot t) \overset{CFT}{\longleftrightarrow} H(f) = \frac{A}{2}, \text{ for } \pm \frac{1}{T_0} = \pm \frac{B}{2\pi}$$

$$0, \text{ for } f \neq \pm \frac{1}{T_0}$$

Where: CFT is the continuous Fourier transform.

Then h(t) was sampled and truncated to produce g(t), the input to the DFT for the continuous function of time.

It was also shown that:

$$g(t) \overset{CFT}{\longleftrightarrow} G(f)$$

Although G(f) bears little resemblance to H(f), the DFT produces values G(f) that give a good approximation for the same value of H(f) when dt and T_0 are chosen correctly.

For example

$$g(n \cdot dt) \overset{DFT}{\longleftrightarrow} G(k \cdot df) \approx C \cdot H(k \cdot df)$$

Where: n is $0, \ldots, N-1$.

k is $0, \ldots, (N-1)/2, -N/2, \ldots, -1$.

df is $1/(N \cdot dt)$.

C is the scaling component.

$\cdot$ is the multiplication operation.

In the above example, because h(t) is periodic and band-limited, and because dt and T_0 were chosen correctly,

$$G(k \cdot df) = C \cdot H(k \cdot df)$$

In Section 6.2.2, it was shown that

$$G(k \cdot df) = \frac{A \cdot T_0}{2 \cdot dt}, \quad \text{for } k \cdot df = \pm \frac{1}{T_0} + \frac{n}{dt},\ n = 0, \pm 1, \pm 2, \pm 3, \ldots, \pm\infty$$

$$= 0, \quad \text{for } k \cdot df \neq \pm \frac{1}{T_0} + \frac{n}{dt}$$

From the above expression, and from H(f) given previously, it follows that

$$C = T_0 / dt$$

Furthermore, because

$$dt = T_0 / N$$

it follows that

$$C = N$$

Where: N is the total number of samples.

Thus, for the example we have

$$G(k \cdot df) = N \cdot H(k \cdot df)$$

or

$$1/N \cdot G(k \cdot df) = H(k \cdot df)$$

In general, if h(t) is periodic,

$$1/N \cdot G(k \cdot df) \approx H(k \cdot df)$$

Here it must be stressed that factor 1/N is not part of the DFT. Factor 1/N is a direct result of the approximation that resulted in g(t) from h(t). The DFT of g(t) is G(f), *not* $1/N \cdot G(f)$. To summarize, if

$$h(t) \overset{CFT}{\longleftrightarrow} H(f)$$

then

$$h(t) \approx g(n \cdot dt) \overset{DFT}{\longleftrightarrow} G(k \cdot df)$$

Where: $n = 0, \ldots N - 1$

$k = 0, \ldots (N - 1) / 2, -N / 2, \ldots, -1$

The description given above also applies to the output of the FFT subroutine for a forward transform (from time to frequency domain) because the results of the DFT and the results of the FFT subroutine are identical in the forward direction.

However, this description does not apply to the output of the FFT subroutine for an inverse transform. The output from the FFT subroutine and the output from the DFT differ by a factor of 1/N for the inverse transform. This difference is directly attributable to the DFT algorithm. (See Section 6.1.2.)

Therefore, if

$$g(n \cdot dt) \overset{DFT}{\longleftrightarrow} G(k \cdot df), n = 0, \ldots, N - 1$$

$$k = 0, \ldots, (N - 1) / 2, -N / 2, \ldots, -1$$

then

$$g(n \cdot dt) \overset{FFT}{\longrightarrow} G(k \cdot df)$$

but

$$(1 / N) \cdot g(n \cdot dt) \overset{FFT}{\longleftarrow} G(k \cdot df)$$

or

$$g(n \cdot dt) \overset{FFT}{\longleftarrow} (1 / N) \cdot G(k \cdot df)$$

But we have just determined that $1/N \cdot G(k \cdot df)$ is approximately $H(k \cdot df)$ and, under special circumstances, is exactly $H(k \cdot df)$. Thus

$$g(n \cdot dt) \overset{FFT}{\longleftarrow} (1 / N) \cdot G(k \cdot df) \approx H(k \cdot df)$$

or

$$g(n \cdot dt) \overset{CFT}{\leftsquigarrow} H(k \cdot df)$$

To summarize the scaling procedure that fast Fourier transform subroutine uses, we have

$$h(t) \approx g(n \cdot dt) \overset{FFT}{\longrightarrow} G(k \cdot df) \approx N \cdot H(k \cdot df)$$

and

$$h(t) \approx g(n \cdot dt) \overset{FFT}{\longleftarrow} (1 / N) \cdot G(k \cdot df) \approx H(k \cdot df)$$

Thus, if one understands the assumptions that must be made, the scaling of the output from the FFT subroutine can be performed as follows

$$h(t \cdot dt) \overset{CFT}{\rightsquigarrow} FFT\,(h(t)) \cdot 1/N \cdot 2^{ISCALE} \qquad ,t = k \cdot dt, k = 0,1, \ldots ,N-1$$

$$FFT(H(f)) \cdot 2^{ISCALE} \overset{CFT}{\leftsquigarrow} H(f), f = k \cdot df, k = 0, \ldots (N-1)/2, -N/2, \ldots , -1$$

Where:

h (t)	is the assumed function of time.
FFT (h(t))	is the output of the FFT subroutines for a forward transform.
$\overset{CFT}{\rightsquigarrow}$	is an approximate forward CFT.
N	is the number of samples.
H (f)	is the assumed frequency function to be transformed.
FFT (H(f))	is the output of the FFT subroutine on inverse transform.
$\overset{CFT}{\leftsquigarrow}$	is an approximate inverse CFT.
ISCALE	is the scaling indicator the subroutine returns to the calling program.

6.4 Useful Properties of the FFT

The fast Fourier transform has a number of useful properties. These properties are listed as follows: The fast Fourier transform has a number of useful properties. These properties are listed as follows:

- Linearity: $h(n) + g(n) \longleftrightarrow H(m) + G(m)$
- Symmetry: $1 / N\, H(m) \longleftrightarrow h(-n)$
- Time Shifting: $h(n - k) \longleftrightarrow H(m)\, e^{-j2\pi mk/N}$
- Frequency Shifting: $h(n)\, e^{j2\pi mk/N} \longleftrightarrow H(m - k)$
- Even Function: $h(n) \longleftrightarrow Re\,[H(m)]$
- Odd Function: $h(n) \longleftrightarrow jIm\,[H(m)]$
- Time Convolution: $h(n) * g(n) \longleftrightarrow H(m)\, G(m)$
- Frequency Convolution: $h(n)\, g(n) \longleftrightarrow \frac{1}{N} H(m) * G(m)$

6.5 How to Call the FFT Subroutine

The general format of the FORTRAN call to the FFT subroutine is:

```
CALL FFT(IERROR,N,IREAL,IMAG,INVRS,ISCALE)
```

For reference, argument names in the call to FFT have been assigned arbitrarily. You may 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, either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results and no data is processed. The arguments are described in the following discussion.

IERROR is an integer variable used to report error conditions. The values that IERROR can return and their meanings are given below.

0	=	No error
1	=	N is less than eight
2	=	N is greater than 8192 or F.MAXN
3	=	N is not a power of two
–n	=	Incorrect number of arguments in call

N is an integer variable that specifies the number of elements to be transformed. N must be a value that is a power of two between eight and whatever maximum array size you specified in the conditional assembly parameters F.MAXN. (See Section 6.6.3 for an explanation of F.MAXN.)

IREAL is an integer array N elements long that contains the real portion of the input data to be transformed. FFT returns the real results to this array, replacing the input data.

IMAG is an integer array N elements long that contains the imaginary portion of the input data to be transformed. FFT returns the imaginary results to this array, replacing the output data.

INVRS is an integer variable that indicates whether the subroutine is to perform a forward or an inverse transform. The two values that you can use in INVRS, and their meanings are given below.

INVRS = 0	The subroutine is to perform a forward transform
INVRS = 1	The subroutine is to perform an inverse transform

ISCALE is an argument set by the FFT subroutine. It is an integer variable that indicates the number of times the results of the FFT subroutine have been divided by two. The FFT subroutine sets the scaling factor as necessary to overflow.

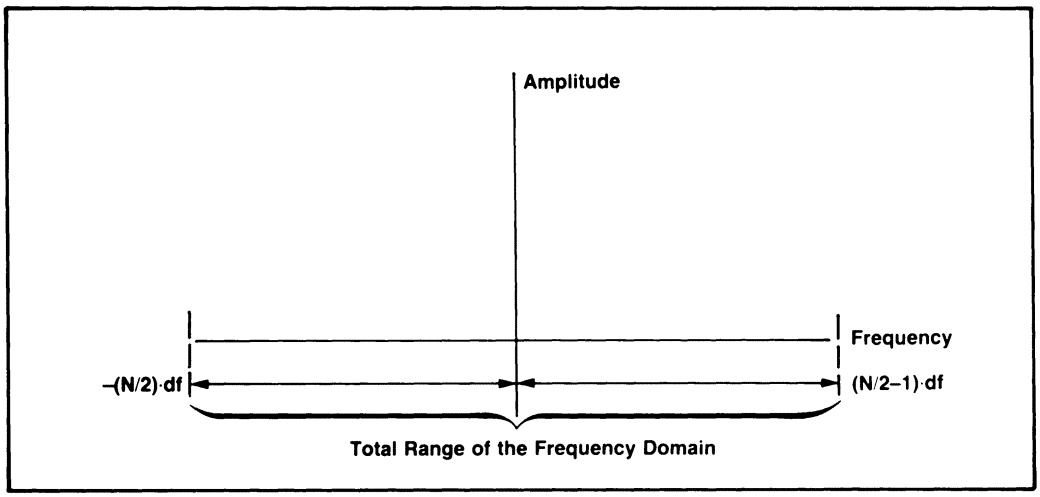
To obtain the unscaled results of the fast Fourier transform, multiply each output element in IREAL and IMAG by (real) 2^{ISCALE}.

6.5.1 Interpreting the Results of the Output Arrays

Each output element returned to IREAL and IMAG as a result of applying the FFT corresponds to the transformed function evaluated at a given frequency value. The frequency value at which the function is evaluated depends on the length of the period of the input function, T_0. Thus, frequency values are evaluated at evenly spaced intervals of $1/T_0$, which is referred to as df. For convenience, the following explanation discusses IREAL only, but applies equally to IMAG.

The total frequency domain consists of values in the range $-(N/2)\cdot(df)$ to $[(N/2)-1]\cdot(df)$. Figure 6–5 shows the total range of the frequency domain expressed as a graph where the x axis represents frequency and the y axis represents amplitude.

Figure 6–5: Total Range of the Frequency Domain



Although evaluated at evenly spaced intervals, values returned in the first half of IREAL represent values in the second half of the frequency domain; that is, they represent values in the range 0 to $[(N/2)-1]\cdot(df)$. Thus, values returned in IREAL beginning with the first element, or IREAL(1), can be expressed as:

IREAL(1) = G(f)
where f = 0

IREAL(2) = G(f)
where f = df or $1/T_0$

IREAL(3) = G(f)
where f = $2 \cdot df$

.
.
.

IREAL(N/2) = G(f)
where f = $\left(\frac{N}{2} - 1\right) \cdot df$

Values returned in the second half of IREAL represent values in the first half of the frequency domain; that is, they represent values in the range $-(N/2) \cdot (df)$ to $-(df)$. Thus, values returned in IREAL beginning with the first element in the second half of the array or IREAL(N/2 + 1) can be expressed as:

$$\text{IREAL}(N/2 + 1) = G(f) \quad \text{where } f = -\frac{N}{2} \cdot df$$

$$\text{IREAL}(N/2 + 2) = G(f) \quad \text{where } f = \left(-\frac{N}{2} + 1\right) \cdot df$$

$$\text{IREAL}(N/2 + 3) = G(f) \quad \text{where } f = \left(-\frac{N}{2} + 2\right) \cdot df$$

.

.

.

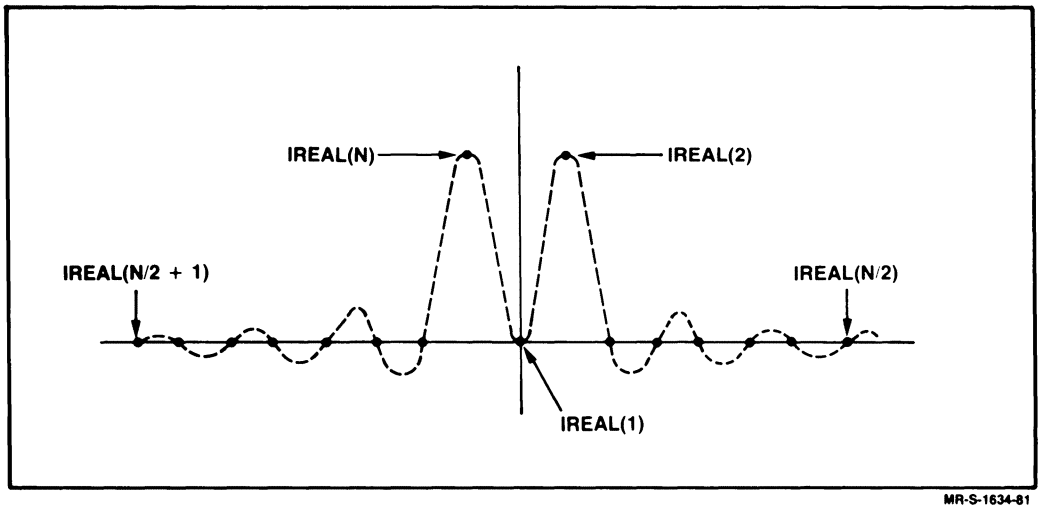
$$\text{IREAL}(N) = G(f) \quad \text{where } f = \left(-\frac{N}{2} + \frac{N}{2} - 1\right) \cdot df \text{ or } -df$$

For an explanation of why the FFT output works this way, see Section 6.2.2.

Figure 6–6 shows five elements of the IREAL output array, each of which represents a transformed function. The figure shows which values in the frequency domain each element (transformed function) corresponds to. The five elements are:

IREAL(1)	The first element in the output array
IREAL(2)	The second element in the output array
IREAL(N/2)	The last element in the first half of the output array
IREAL(N/2 + 1)	The first element in the second half of the output array
IREAL(N)	The last element in the second half of the output array. IREAL(N) is also the last element of the entire output array.

Figure 6–6: Five Elements in the IREAL Output Array



6.6 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the fast Fourier subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

6.6.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS (KE11–E) hardware or any other floating-point option available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 86 words.

6.6.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE (KE11) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 80 words.

6.6.3 F.MAXN (Maximum I/O Array Size)

The F.MAXN option specifies the maximum number of complex elements that the FFT subroutine can process at one time. The maximum number of complex elements is limited to those multiples of 1K (1024) that are powers of 2 in the range 1K to 8K (8192). In other words, F.MAXN is limited to the values 1024, 2048, 4096, and 8192.

F.MAXN allows you to control the size of the FFT subroutine. The FFT subroutine uses a quarter-sine-wave, look-up table that requires a number of elements directly proportional to the maximum size of the FFT subroutine input array.

F.MAXN is not enabled on the distributed object file. If you do not choose to enable F.MAXN, the FFT subroutine uses a default value of 1024 for F.MAXN.

If you want to use a larger maximum number of complex values, you should redefine F.MAXN as 2048, 4096, or 8192 in the distributed source file.

Redefining F.MAXN increases the memory requirements for the FFT subroutine. Specifically:

1. Changing F.MAXN from 1024 to 2048 increases the FFT subroutine size by 256 words.
2. Changing F.MAXN from 1024 to 4096 increases the FFT subroutine size by 768 words.
3. Changing F.MAXN from 1024 to 8192 increases the FFT subroutine size by 1792 words.

6.7 Example Using the FFT Subroutine

The following FORTRAN program example illustrates how the FFT subroutine produces both forward and inverse transforms of a specific set of input data.

FFT Example #1

FFT Example #1

```
(1)         DIMENSION IR(32),IM(32)
            DATA PI/3.141593/,IM/32*0/C/.01/
            DELX=2.*PI/32.
            X=0.

(2)         DO 1 I=1,32
            T=SIN(X)
            IR(I)=T*1000.+SIGN(C,T)
          1 X=X+DELX

(3)         TYPE 800
            TYPE 900
            TYPE 1001,IR
            TYPE 1002,IM

(4)         CALL FFT(IE,32,IR,IM,0,ISF)

(5)         IF(IE) 99,2,99
          2 IF(ISF.NE.0) TYPE 999,ISF

(6)         TYPE 1000
            TYPE 1001,IR
            TYPE 1002,IM

(7)         CALL FFT(IE,32,IR,IM,1,ISI)

(8)         IF(IE) 99,3,99
          3 IF(ISI.NE.0) TYPE 999,ISI

(9)         TYPE 2000
            TYPE 1001,IR
            TYPE 1002,IM

(10)        SCAL=2.**(ISF+ISI)

(11)        DO 4 I=1,32
            IR(I)=SCAL*(IR(I)/32)
          4 IM(I)=SCAL*(IM(I)/32)

(12)        TYPE 3000
            TYPE 1001,IR
            TYPE 1002,IM

            STOP

(13)     99 TYPE 998,IE

            STOP

(14)  800   FORMAT(1H1,T25,'FFT Example #1',//)
      900   FORMAT(' DATA TO BE TRANSFORMED - SINE WAVE SCALED BY 1000.')
      998   FORMAT(//' THE ERROR CODE RETURNED = ',I4)
      999   FORMAT(//' THE SCALE FACTOR RETURNED = ',I4)
     1000   FORMAT(//,' RESULTS FROM THE FORWARD FOURIER TRANSFORM OF 32',
         1  ' POINTS OF A SINE WAVE')
     1001   FORMAT(/' - REAL PART -'/,(8I8))
     1002   FORMAT(/' - IMAGINARY PART -'/,(8I8))
     2000   FORMAT(//,' RESULTS FROM THE INVERSE FOURIER TRANSFORM ',
         1  //,32X,'- BEFORE SCALING
         2  -')
     3000   FORMAT(//,32X,'- AFTER SCALING -')
            END
```

① Define array variables and their sizes; initialize parameters external to the subroutine, and set the value of array IM to zero.

② Compute 32 values of a sine wave scaled by 1000.

③ Print the real and imaginary parts of the numbers to be transformed.

④ Call the FFT subroutine to perform a forward transform on values stored in IR and IM.

⑤ If IE(RROR) is not equal to zero, print an error message; if ISCALE is not equal to zero, print the scaling factor.

⑥ Print the real and imaginary parts of the forward transform.

⑦ Call the FFT subroutine to perform an inverse transform on values stored in IR and IM.

⑧ If IE(RROR) is not equal to zero, print an error message; if ISCALE is not equal to zero, print the scaling factor.

⑨ Print the real and imaginary parts of inverse transforms (before scaling).

⑩ Compute the scaling factor.

⑪ Multiply the real and imaginary parts of the inverse transform of data by the scaling factor.

⑫ Print the real and imaginary parts of the inverse transform of data (after scaling).

⑬ Print error messages (error code returned).

⑭ Format statements

```
                        FFT Example #1

 DATA TO BE TRANSFORMED - SINE WAVE SCALED BY 1000.

- REAL PART -
     0      195      382      555      707      831      923      980
  1000      980      923      831      707      555      382      195
     0     -195     -382     -555     -707     -831     -923     -980
 -1000     -980     -923     -831     -707     -555     -382     -195

- IMAGINARY PART -
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0

RESULTS FROM THE FORWARD FOURIER TRANSFORM OF 32 POINTS OF A SINE WAVE

- REAL PART -
     0       -7        0       -6        0       -2        0       -2
     0        1        0        2        0        0        0        0
     0        1        0        2        0        2        0        2
     0        1        0        2        0        0        0        0

- IMAGINARY PART -
     0   -15981        0        2        0        1        0        0
     0       -5        0       -2        0       -1        0       -4
     0       -1        0        2        0        1        0        0
     0       -1        0       -2        0       -5        0    15984

RESULTS FROM THE INVERSE FOURIER TRANSFORM

                                  - BEFORE SCALING -

- REAL PART -
    -4     6217    12200    17749    22598    26578    29531    31356
 31972    31354    29527    26582    22604    17752    12223     6254
     4    -6217   -12200   -17749   -22598   -26578   -29531   -31356
-31972   -31354   -29527   -26582   -22604   -17752   -12223    -6254

- IMAGINARY PART -
   -12       -8       -9       -6       -6        4       -5       -2
    -4       -5       -6      -17        0      -14      -15      -20
    12        8        9        6        6       -4        5        2
     4        5        6       17        0       14       15       20

                                  - AFTER SCALING -

- REAL PART -
     0      194      381      554      706      830      922      979
   999      979      922      830      706      554      381      195
     0     -194     -381     -554     -706     -830     -922     -979
  -999     -979     -922     -830     -706     -554     -381     -195

- IMAGINARY PART -
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0
```

PHASE ANGLE AND AMPLITUDE SPECTRA (PHAMPL) SUBROUTINE

FORMAT:

CALL PHAMPL(N,IR,IM,PH,AM)

Where:

N is an integer variable that specifies the length of the input and output arrays.

IR is an integer array containing the real parts of the input data.

IM is an integer array containing the imaginary parts of the input data.

PH is a real array used to store phase angles.

AM is a real array used to store amplitudes.

FILE NAMES:

PHAMPL.MAC (source file); PHAMPL.OBJ (object file)

OTHER ROUTINES USED:

FLOAT, ATAN2, and SQRT from the FORTRAN library.

OPTIONS:

- EIS (Extended Instruction Set — KE11–E)
- EAE (Extended Arithmetic Element — KE11)
- F4P$ (FORTRAN 77 compiler)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If the following options are enabled:

NONE	EIS	EAE
182	138	157

TYPICAL EXECUTION SPEED:

With PDP–11/34 and EIS enabled: 180 Points/second.

With PDP–11/03 and EIS enabled: 70 Points/second.

Chapter 7
The Phase Angle and Amplitude Spectra (PHAMPL) Subroutine

The phase angle and amplitude spectra (PHAMPL) subroutine deals with complex values of the type produced by the fast Fourier transform (FFT) subroutine. (See Chapter 6 for more information on the FFT subroutine.) PHAMPL converts such complex values into phase angles and amplitudes.

The input to PHAMPL consists of two integer arrays. One integer array contains the real parts of the complex values to be converted. The other integer array contains the imaginary parts of the complex values to be converted.

The output from PHAMPL consists of two real arrays. One real array contains the phase angles of the complex values. The other real array contains the amplitudes of the complex values. All values in both real arrays are related through their subscripts.

Mathematically, the function of the subroutine can be described as:

$$A_i = \sqrt{IR^2_i + IM^2_i}$$

$$P_i = TAN^{-1}(IM_i/IR_i)$$

Where:

- P is the real array containing the related phase angles.
- A is the real array containing the related amplitudes.
- IR is the integer array containing the real parts of the complex values.
- IM is the integer array containing the imaginary parts of the complex values.

7.1 How to Call PHAMPL

The general format for the FORTRAN call is:

```
CALL PHAMPL(N,IR,IM,PH,AM)
```

For reference, argument names in the call to PHAMPL have been assigned arbitrarily. You may 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, either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results, and no data is processed. The arguments are described in the following discussion.

N is an integer that defines the length of the input and output arrays.

IR is an integer array containing the real parts of the complex values to be converted.

IM is an integer array containing the imaginary parts of the complex values to be converted.

PH is a real array in which the subroutine stores the phase angles from the conversion.

$$PH(I) = ATAN2(FLOAT(IM(I))/FLOAT(IR(I))$$

AM is a real array in which the subroutine stores the amplitudes resulting from the conversion.

$$AM(I) = SQRT(FLOAT(IR(I) \cdot IR(I) + IM(I) \cdot IM(I)))$$

7.2 Other Routines Used by PHAMPL

PHAMPL requires the FORTRAN library routines FLOAT, ATAN2, and SQRT. However, you do not have to take any special steps to access these library routines. Because the FORTRAN library is required by all FORTRAN programs, the necessary library routines are accessed automatically when you link or task-build the program that calls PHAMPL.

PHAMPL accesses the FORTRAN 77 library and the FORTRAN IV library differently. See Section 7.3.3 if you are using FORTRAN 77.

7.3 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the phase angle and amplitude spectra subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

7.3.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS (KE11–E) hardware or any other floating-point option available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 44 words.

7.3.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE (KE11) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 25 words.

7.3.3 F4P$ (FORTRAN 77 Compiler)

The distributed object file is intended for use with the FORTRAN IV compiler. Enable this option if you are using the FORTRAN 77 compiler.

7.4 Examples Using the PHAMPL Subroutine

The two examples presented here illustrate how to use PHAMPL to convert complex values to their corresponding phase angles and amplitudes. Example 1 uses random, complex numbers generated by a random-number generator. Example 2 uses output from the FFT subroutine.

NOTE

If you use FORTRAN 77 and you want to duplicate the terminal output for the example programs, replace the standard random-number generator in F4POTS with F4PRAN.OBJ. Terminal output for the example programs is based on the FORTRAN IV random-number generator. The FORTRAN 77 random-number generator is different from that for FORTRAN IV and will not produce the same output. See Section B.1.

PHAMPL Example #1

PHAMPL Example #1

```
(1)         DIMENSION IR(32),IM(32),P(32),A(32)
            DATA N,M/0,0/
            NEXT(V)=(RAN(N,M)*1000.)-500.

(2)         DO 10 I=1,32
            IR(I)=NEXT(X)
      10    IM(I)=NEXT(X)
            TYPE 900
            TYPE 1002,IR
            TYPE 1003,IM

(3)         DO 11 I=1,32
            X0=IR(I)
            X1=IM(I)
            P(I)=ATAN2(X1,X0)
      11    A(I)=SQRT(X1*X1+X0*X0)
            TYPE 1006,P
            TYPE 1007,A

(4)         CALL PHAMPL(32,IR,IM,P,A)
            TYPE 1004,P
            TYPE 1005,A

(5)   900   FORMAT(1H1,T20,'PHAMPL Example #1',//)
     1002   FORMAT(' RANDOM INPUT DATA'/' REAL PART',/,(4I15))
     1003   FORMAT(/,'     IMAGINARY PART',/,(4I15))
     1004   FORMAT(//,'     PHASE ANGLES FROM SUBROUTINE',/,(4F15.5))
     1005   FORMAT(/,' AMPLITUDES FROM SUBROUTINE',/,(4F15.5))
     1006   FORMAT(//,' PHASE ANGLES FROM DIRECT CALCULATIONS',/,(4F15.5))
     1007   FORMAT(/,' AMPLITUDES FROM DIRECT CALCULATIONS',/,(4F15.5))
            STOP
            END
```

① Define array variables and their sizes; initialize parameters external to PHAMPL subroutine; define real and imaginary values of 32 random numbers to be processed.

② Compute 32 random values and print real and imaginary parts.

③ Calculate phase angles and amplitudes for random input values and print results.

④ Call the PHAMPL subroutine to calculate phase angles and amplitudes for random input values and print results.

⑤ Format statements

```
                    PHAMPL Example #1

RANDOM INPUT DATA
REAL PART
          -499          -499          -487          -344
           302           322           338           -23
           148          -393          -106           -81
           158           426          -173           326
           -86            15           241           132
          -182             1          -194           346
           -45           147           360          -462
           448          -423            29          -180

 IMAGINARY PART
          -499          -496          -455            33
          -493           373          -329          -177
           490           226           326          -428
          -190           277           461          -191
           206           229           384           334
          -106           -29            99           178
           122          -218           130            47
           269            36          -151           283


PHASE ANGLES FROM DIRECT CALCULATIONS
       -2.35619      -2.35921      -2.39015       3.04596
       -1.02118       0.85865      -0.77191      -1.70002
        1.27747       2.61971       1.88517      -1.75784
       -0.87709       0.57654       1.92980      -0.52998
        1.96628       1.50539       1.01033       1.19443
       -2.61420      -1.53633       2.66973       0.47514
        1.92417      -0.97752       0.34654       3.04021
        0.54075       3.05669      -1.38105       2.13730

AMPLITUDES FROM DIRECT CALCULATIONS
      705.69257     703.57446     666.47882     345.57922
      578.14618     492.76059     471.68317     178.48810
      511.86325     453.34866     342.80023     435.59729
      247.11131     508.13876     492.39212     377.83197
      223.23082     229.49074     453.36188     359.13785
      210.61813      29.01724     217.80037     389.10153
      130.03461     262.93155     382.75317     464.38455
      522.55621     424.52914     153.75955     335.39380

 PHASE ANGLES FROM SUBROUTINE
       -2.35619      -2.35921      -2.39015       3.04596
       -1.02118       0.85865      -0.77191      -1.70002
        1.27747       2.61971       1.88517      -1.75784
       -0.87709       0.57654       1.92980      -0.52998
        1.96628       1.50539       1.01033       1.19443
       -2.61420      -1.53633       2.66973       0.47514
        1.92417      -0.97752       0.34654       3.04021
        0.54075       3.05669      -1.38105       2.13730

AMPLITUDES FROM SUBROUTINE
      705.69257     703.57446     666.47882     345.57922
      578.14618     492.76059     471.68317     178.48810
      511.86325     453.34866     342.80023     435.59729
      247.11131     508.13876     492.39212     377.83197
      223.23082     229.49074     453.36188     359.13785
      210.61813      29.01724     217.80037     389.10153
      130.03461     262.93155     382.75317     464.38455
      522.55621     424.52914     153.75955     335.39380
```

PHAMPL Example #2

PHAMPL Example #2

```
(1)     DIMENSION AMP(32),PHASE(32),IMAG(32)
        INTEGER REAL(32),SCALE,ERROR
        DATA N,I1,I2,T,A,B,C,D/32,0,0,0.,100.,200.,.5,2./
        DATA PI/3.141593/,IMAG/32*0/

(2)     DT=PI/8.
        CALL RANDU(I1,I2,X)
        DO 1 I=1,N
        REAL(I)=A*SIN(C*T)+B*COS(D*T)+(-1.**I)*15.*RAN(I1,I2)
      1 T=T+DT
        TYPE 900
        TYPE 1000
        TYPE 1001,REAL
        TYPE 1002,IMAG

(3)     CALL FFT(ERROR,N,REAL,IMAG,0,SCALE)
        IF(ERROR.NE.0) PRINT 3000,ERROR

(4)     CALL PHAMPL(N,REAL,IMAG,PHASE,AMP)
        TYPE 2000
        TYPE 1001,REAL
        TYPE 1002,IMAG
        TYPE 2001,PHASE
        TYPE 2002,AMP

        STOP

(5)   900 FORMAT(1H1,T22,'PHAMPL Example #2',//)
     1000 FORMAT(' INPUT DATA FOR FFT')
     1001 FORMAT(/' ***REAL PART***',/,(4I16))
     1002 FORMAT(' ***IMAGINARY PART***',/,(4I16))
     2000 FORMAT(//' RESULTS OF THE FFT')
     2001 FORMAT(/' ***PHASE ANGLES***',/,(4F16.5))
     2002 FORMAT(' ***AMPLITUDES***',/,(4F16.5))
     3000 FORMAT(///'ERROR CODE FROM FFT = ',I6,///)
        END
```

① Define array variables and their sizes; initialize variables external to the subroutine; specify input as integer.

② Generate 32 random values for input to the FFT subroutine and print the real and imaginary values:

Values = 100.*SIN(X/2.) + 200.*COS(2.*X) + NOISE

③ Call FFT subroutine to process these input values; print any error messages.

④ Call PHAMPL subroutine to process the output from the FFT subroutine; print the real and imaginary results of the FFT and the corresponding phase angles and amplitudes.

⑤ Format statements

Terminal Output

```
                    PHAMPL Example #2

INPUT DATA FOR FFT

***REAL PART***
              199             160              38             -86
             -129             -60              84             227
              299             227              79             -70
             -131             -93              33             151
              185             120             -49             -202
             -283             -230             -93             33
               95              29             -104             -229
             -285             -209             -42             115
***IMAGINARY PART***
                0               0                0               0
                0               0                0               0
                0               0                0               0
                0               0                0               0
                0               0                0               0
                0               0                0               0
                0               0                0               0
                0               0                0               0

RESULTS OF THE FFT

***REAL PART***
             -221              23              25              11
             3188               1              -9              16
                4              -7             -32               1
               21              29             -29              10
               13              13             -27              31
               22               3             -29             -12
                4              19              -6               5
             3189              19              27              30
***IMAGINARY PART***
                0           -1611             -39             -31
                8              -4             -14              35
               -5             -15              -2              -7
              -29              14             -20               7
                0             -13              15             -19
               28               6               2              11
                5             -37              14               1
               -7              32              40            1615

***PHASE ANGLES***
          3.14159        -1.55652        -1.00076        -1.22982
          0.00251        -1.32582        -2.14213         1.14202
         -0.89606        -2.00742        -3.07917        -1.42890
         -0.94405         0.44976        -2.53784         0.61073
          0.00000        -0.78540         2.63449        -0.54985
          0.90483         1.10715         3.07274         2.39965
          0.89606        -1.09640         1.97569         0.19740
         -0.00220         1.03499         0.97705         1.55222
***AMPLITUDES***
        221.00000      1611.16418        46.32494        32.89377
       3188.01001         4.12311        16.64332        38.48376
          6.40312        16.55295        32.06244         7.07107
         35.80503        32.20248        35.22783        12.20656
         13.00000        18.38478        30.88689        36.35932
         35.60899         6.70820        29.06888        16.27882
          6.40312        41.59327        15.23155         5.09902
       3189.00757        37.21559        48.25971      1615.27856
```

POWER SPECTRUM (POWRSP) SUBROUTINE

FORMAT:

CALL POWRSP(N,IR,IM,P)

Where:

N is an integer variable that specifies the length of the input arrays.

IR is an integer array containing the real parts of the input data.

IM is an integer array containing the imaginary parts of the input data.

P is a real array used to store power spectrum results.

FILE NAMES:

POWRSP.MAC (source file); POWRSP.OBJ (object file)

OPTIONS:

- EIS (Extended Instruction Set — KE11–E)
- EAE (Extended Arithmetic Element — KE11)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If the following options are enabled:

NONE	EIS	EAE
119	75	94

TYPICAL EXECUTIONS SPEED:

With PDP–11/34 and EIS enabled: 5500 Points/second.

With PDP–11/03 and EIS enabled: 2400 Points/second.

Chapter 8
The Power Spectrum (POWRSP) Subroutine

The power spectrum (POWRSP) subroutine deals with complex values of the type produced by the fast Fourier transform (FFT) subroutine. (See Chapter 6 for a detailed description of the FFT subroutine.) POWRSP computes the power spectrum — the relationship between power and signal frequency — of a set of Fourier coefficients by calculating the squares of the magnitudes of that set of Fourier coefficients.

The input to POWRSP consists of two integer arrays. One integer array contains the real parts of the Fourier coefficients. The other integer array contains the imaginary parts of the Fourier coefficients.

The output from POWRSP consists of one real array that contains the calculated power spectrum.

Mathematically, the function of the subroutine can be described as

$P_i = IR_i^2 + IM_i^2$

Where:

P	is the real array used to store the power spectrum.
IR	is the integer array containing the real parts of the Fourier coefficients.
IM	is the integer array containing the imaginary parts of the Fourier coefficients.

8.1 How to Call POWRSP

The general format for the FORTRAN call is:

```
CALL POWRSP(N,IR,IM,P)
```

For reference, argument names in the call to POWRSP have been assigned arbitrarily. You may 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, either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results and no data is processed. The arguments are described in the following discussion.

N is an integer that defines the length of the input and output arrays.

IR is an integer array containing the real parts of the complex Fourier coefficients.

IM is an integer array containing the imaginary parts of the complex Fourier coefficients.

P is a real array in which POWRSP stores the power spectrum results. Values for the array elements in P are computed as follows

$$P(I) = IR(I)*IR(I) + IM(I)*IM(I)$$

8.2 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the power spectrum subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

8.2.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS (KE11–E) hardware or any other floating-point option available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 44 words.

8.2.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 25 words.

8.3 Examples Using the POWRSP Subroutine

The following example FORTRAN program illustrates how to use POWRSP to calculate the power spectrum of complex Fourier coefficients. Example 1 uses random, complex numbers generated by a random-number generator. Example 2 uses output from the FFT subroutine.

NOTE

If you use FORTRAN 77 and you want to duplicate the terminal output for the example programs, replace the standard random-number generator in F4POTS with F4PRAN.OBJ. Terminal output for the example programs is based on the FORTRAN IV random-number generator. The FORTRAN 77 random-number generator is different from that for FORTRAN IV and will not produce the same output. See Section B.1.

POWRSP Example #1

POWRSP Example #1

```
      DIMENSION IR(32),IM(32),P(32)
      DATA N,M/0,0/
      NEXT(V)=(RAN(N,M)*500.)-250.

      DO 10 I=1,32
      IR(I)=NEXT(X)
10    IM(I)=NEXT(X)
      TYPE 900
      TYPE 1002,IR
      TYPE 1003,IM

      DO 11 I=1,32
      X0=IR(I)
      X1=IM(I)
      P(I)=X0*X0+X1*X1
11    CONTINUE
      TYPE 1004,P

      CALL POWRSP(32,IR,IM,P)
      TYPE 1005,P

 900  FORMAT(1H1,T20,'POWRSP Example #1',//)
1002  FORMAT(' RANDOM INPUT DATA'/' REAL PART',/,(4I15))
1003  FORMAT(/,'     IMAGINARY PART',/,(4I15))
1004  FORMAT(//,'     POWER FROM DIRECT CALCULATIONS',/,(4F15.5))
1005  FORMAT(//,'     POWER FROM SUBROUTINE',/,(4F15.5))
      STOP
      END
```

① Define array variables and their sizes; initialize parameters external to POWRSP subroutine; define real and imaginary values of 32 random numbers to be processed.

② Compute 32 random values and print real and imaginary parts.

③ Calculate power spectrum for random input values and print results.

④ Call the POWRSP subroutine to calculate power spectrum for random input values and print results.

⑤ Format statements

Terminal Output

```
                    POWRSP Example #1

RANDOM INPUT DATA
REAL PART
            -249            -249            -243            -172
             151             161             169             -11
              74            -196             -53             -40
              79             213             -86             163
             -43               7             120              66
             -91               0             -97             173
             -22              73             180            -231
             224            -211              14             -90

 IMAGINARY PART
            -249            -248            -227              16
            -246             186            -164             -88
             245             113             163            -214
             -95             138             230             -95
             103             114             192             167
             -53             -14              49              89
              61            -109              65              23
             134              18             -75             141


 POWER FROM DIRECT CALCULATIONS
   124002.00000    123505.00000    110578.00000    29840.00000
    83317.00000     60517.00000     55457.00000     7865.00000
    65501.00000     51185.00000     29378.00000    47396.00000
    15266.00000     64413.00000     60296.00000    35594.00000
    12458.00000     13045.00000     51264.00000    32245.00000
    11090.00000       196.00000     11810.00000    37850.00000
     4205.00000     17210.00000     36625.00000    53890.00000
    68132.00000     44845.00000      5821.00000    27981.00000

 POWER FROM SUBROUTINE
   124002.00000    123505.00000    110578.00000    29840.00000
    83317.00000     60517.00000     55457.00000     7865.00000
    65501.00000     51185.00000     29378.00000    47396.00000
    15266.00000     64413.00000     60296.00000    35594.00000
    12458.00000     13045.00000     51264.00000    32245.00000
    11090.00000       196.00000     11810.00000    37850.00000
     4205.00000     17210.00000     36625.00000    53890.00000
    68132.00000     44845.00000      5821.00000    27981.00000
```

POWRSP Example #2

POWRSP Example #2

```
      DIMENSION PSPECT(32),IMAG(32)
      INTEGER REAL(32),SCALE,ERROR
      DATA N,I1,I2,T,A,B,C,D/32,0,0,0.,100.,200.,.5,2./
      DATA PI/3.141593/,IMAG/32*0/

      DT=PI/8.
      CALL RANDU(I1,I2,X)
      DO 1 I=1,N
      REAL(I)=A*SIN(C*T)+B*COS(D*T)+(-1.**I)*15.*RAN(I1,I2)
    1 T=T+DT
      TYPE 900
      TYPE 1000
      TYPE 1001,REAL
      TYPE 1002,IMAG

      CALL FFT(ERROR,N,REAL,IMAG,0,SCALE)
      IF(ERROR.NE.0) PRINT 3000,ERROR

      CALL POWRSP(N,REAL,IMAG,PSPECT)
      TYPE 2000
      TYPE 1001,REAL
      TYPE 1002,IMAG
      TYPE 2001,PSPECT

      STOP

  900 FORMAT(1H1,T22,'POWRSP Example #2',//)
 1000 FORMAT(' INPUT DATA FOR FFT')
 1001 FORMAT(/' ***REAL PART***',/,(4I16))
 1002 FORMAT(' ***IMAGINARY PART***',/,(4I16))
 2000 FORMAT(//' RESULTS OF THE FFT')
 2001 FORMAT(//' POWER SPECTRUM RESULTS',/,(4F16.1))
 3000 FORMAT(///'ERROR CODE FROM FFT = ',I6,///)
      END
```

① Define array variables and their sizes; initialize variables external to the subroutine; specify input as integer.

② Generate 32 random values for input to the FFT subroutine and print real and imaginary values:

Values = 100.*SIN(X/2.) + 200.*COS(2.*X) + NOISE

③ Call FFT subroutine to process these input values; print any error messages.

④ Call POWRSP subroutine to process the output from the FFT subroutine; print the real and imaginary results of the FFT and the corresponding power spectrum.

⑤ Format statements

Terminal Output

```
                    POWRSP Example #2

INPUT DATA FOR FFT

***REAL PART***
             199             160              38             -86
            -129             -60              84             227
             299             227              79             -70
            -131             -93              33             151
             185             120             -49             -202
            -283             -230            -93              33
              95              29             -104            -229
            -285             -209            -42             115
***IMAGINARY PART***
               0               0               0               0
               0               0               0               0
               0               0               0               0
               0               0               0               0
               0               0               0               0
               0               0               0               0
               0               0               0               0
               0               0               0               0

RESULTS OF THE FFT

***REAL PART***
            -221              23              25              11
            3188               1              -9              16
               4              -7             -32               1
              21              29             -29              10
              13              13             -27              31
              22               3             -29             -12
               4              19              -6               5
            3189              19              27              30
***IMAGINARY PART***
               0           -1611             -39             -31
               8              -4             -14              35
              -5             -15              -2              -7
             -29              14             -20               7
               0             -13              15             -19
              28               6               2              11
               5             -37              14               1
              -7              32              40            1615

POWER SPECTRUM RESULTS
         48841.0       2595850.0          2146.0          1082.0
      10163408.0            17.0           277.0          1481.0
            41.0           274.0          1028.0            50.0
          1282.0          1037.0          1241.0           149.0
           169.0           338.0           954.0          1322.0
          1268.0            45.0           845.0           265.0
            41.0          1730.0           232.0            26.0
      10169770.0          1385.0          2329.0       2609125.0
```

CORRELATION FUNCTION (CORREL) SUBROUTINE

FORMAT:

CALL CORREL(IERROR,N,IA1,IA2,IFFTA,ISCALE)

Where:

IERROR	is an integer variable used to report errors.
N	is an integer variable used to specify the number of samples to be correlated.
IA1	is an integer array N elements long containing the samples of the first function to be correlated.
IA2	is an integer array N elements long containing samples of the second function to be correlated.
IFFTA	is an integer array used to hold the imaginary part(s) of the input to the FFT subroutine.
ISCALE	is an integer variable set by CORREL to indicate the scaling factor (number of times results have been divided by 2).

FILE NAMES:

CORREL.MAC (source file); CORREL.OBJ (object file)

OTHER ROUTINES USED:

Fast Fourier Transform (FFT) Subroutine

OPTIONS:

- EIS (Extended Instruction Set — KE11–E)
- EAE (Extended Arithmetic Element — KE11)

APPROXIMATE SIZE OF SUBROUTINE (IN WORDS):

If the following options are enabled:

NONE	EIS	EAE
266	240	246

TYPICAL EXECUTION SPEED:

With PDP–11/34 and EIS enabled:	420 Points/second (for auto-correlation) 280 Points/second (for cross-correlation)
With PDP–11/03 and EIS enabled:	150 Points/second (for auto-correlation). 100 Points/second (for cross-correlation).

Chapter 9
The Correlation Function (CORREL) Subroutine

The correlation function subroutine (CORREL) produces an estimate for the correlation function. The correlation function measures the similarity of two functions as one of the functions is shifted in time. The value of the correlation function for any time shift, "t," is the integral over all time of the product of the first function multiplied by the second function after it is shifted by an amount "t."

Mathematically, the correlation function can be described as:

$$R_{xy}(t) = \int_{-\infty}^{\infty} x(\beta)\, y(\beta + t)\, d\beta$$

Where: $x(\beta)$ and $y(\beta)$ are the two functions to be compared.

t is the time shift.

$R_{xy}(t)$ is the resulting correlation function.

When $x(\beta) = y(\beta)$, $R_{xy}(t)$ is called an auto-correlation function. When $x(\beta) \neq y(\beta)$, $R_{xy}(t)$ is called a cross-correlation function.

As this formula indicates, the correlation function yields useful information only when the functions being correlated have finite content:

$$\int_{-\infty}^{\infty} f^2(t)\, dt < \infty$$

But, because the functions to be correlated do not normally have finite content, the correlation function does not generally yield useful information. Under these circumstances, an estimate of the correlation function, called the average correlation function, must provide the desired information.

The mathematical formula for the average correlation function is:

$$R_{xy}(t) = \frac{1}{T} \int_0^T x(\beta)\, y(\beta + t)\, d\beta$$

Where: $R_{xy}(t)$ is the estimated average correlation function.

$x(\beta)$ and $y(\beta)$ are the two functions to be correlated.

T is a representative time interval.

It is this average correlation function that CORREL estimates.

The input to CORREL consists of two integer arrays. One integer array contains the samples of the first function to be correlated. The other integer array contains the samples of the second function to be correlated.

The output from CORREL (that is, the estimates of the average correlation function at various time shifts) is returned in the first input integer array.

9.1 Using the Correlation Function Subroutine

The discrete evaluation of the correlation function implies a large number of mathematical operations. Indeed, the number of mathematical operations increases linearly as the number of values of $R_{xy}(t)$ to be calculated increases.

When you use CORREL, however, this situation is improved. CORREL calls the fast Fourier transform (FFT) subroutine to process the input data, significantly decreasing the number of mathematical operations required to approximate the average correlation function. In fact, this method of evaluation becomes more efficient as the number of values of R_{xy} to be calculated increases.

See Chapter 6 for a full description of the FFT subroutine.

To use CORREL, the two functions you want to correlate should meet the following requirements:

1. They must have a Fourier transform.
2. They must be real (not complex) functions.
3. They must be sampled at the same evenly spaced intervals. (The number of samples must be a power of two.)
4. They must be periodic.
5. They must have a sample period that represents a non-zero, integer-multiple of their periods.

There are exceptions to these requirements. You can also use CORREL to correlate functions that are non-zero over only a finite-term interval.

To correlate such a function, first sample the function over the non-zero intervals. Then, double the number of sample values to be processed. Finally, set the extra samples to zero.

9.2 Discrete Evaluation of CORREL

The following is a brief summary, without proof, of the derivation of the technique used by CORREL.

Convolution, represented by the symbol *, is defined as

$$S_{xy}(\xi) = x(t) * y(t) = \int_{-\infty}^{\infty} x(t)\, y(\xi - t)\, dt$$

Correlation, represented by the symbol $\circledast$, is defined as

$$R_{xy}(\xi) = x(t) \circledast y(t) = \int_{-\infty}^{\infty} x(t)\, y(\xi + t)\, dt$$

These two functions are related as shown

$$x(-t) * y(t) = x(t) \circledast y(t)$$

since

$$x(-t) * y(t) = \int_{-\infty}^{\infty} x(-t)\, y(\xi - t)\, dt$$

Now, let $\beta = -t$, and

$$x(-t) * y(t) = -\int_{\infty}^{-\infty} x(\beta)\, y(\xi + \beta)\, d\beta$$

$$= \int_{-\infty}^{\infty} x(\beta)\, y(\xi + \beta)\, d\beta$$

$$= x(\beta) \circledast y(\beta) \text{ or } x(t) \circledast y(t)$$

It can also be shown that

$$x(t) * y(t) \overset{FT}{\longleftrightarrow} X(f)\, Y(f)$$

Where: $\overset{FT}{\longleftrightarrow}$ represents the Fourier transform and

$$x(t) \overset{FT}{\longleftrightarrow} X(f) \text{ and } y(t) \overset{FT}{\longleftrightarrow} Y(f)$$

Furthermore, it can be observed that if

$$x(t) \overset{FT}{\longleftrightarrow} X(f)$$

then

$$x(-t) \overset{FT}{\longleftrightarrow} X(-f) = X^*(f) \text{ (the conjugate of } X(f))$$

since the real of portion of X (f) is an even function and the imaginary portion is an odd function of f.

Finally, assuming all the preceeding statements are true, we have

$$R_{xy}(\xi) = x(t) \circledast y(t) = x(-t) * y(t) \overset{FT}{\longleftrightarrow} X^*(f)\, Y(f)$$

or

$$R_{xy}(\xi) = \overset{FT}{\longleftrightarrow} X^*(f)\, Y(f)$$

CORREL uses this final relationship to produce the correlation function. The specific steps are:

1. Transform x (t).

   $$x(t) \overset{FT}{\longrightarrow} X(f)$$

2. Transform y (t).

   $$y(t) \overset{FT}{\longrightarrow} Y(f)$$

3. Calculate the conjugate of X (f).

   $$X^*(f) = X(-f)$$

4. Multiply $X^*(f)$ by Y (f)

   $$X^*(f)\, Y(f)$$

5. Perform an inverse transform on the result of Step 4.

   $$R_{xy}(\xi) \overset{FT}{\longleftarrow} X^*(f)\, Y(f)$$

9.3 Calculating the Correlation Coefficients

The results obtained by CORREL provide an estimate for the discrete auto-correlation function or the discrete cross-correlation function. The normalized results of the auto-correlation function and of the cross-correlation function are known as correlation coefficients.

The following equation provides the algorithm by which you can derive the correlation coefficients from the results of the subroutine.

$$C_{xy}(n) = \frac{S_{xy}\, R_{xy}\,(n)}{\sqrt{S_{xx} \cdot R_{xx}(0) \cdot S_{yy} \cdot R_{yy}(0)}} \qquad \text{For } n = 0, 1, 2, \ldots, N-1$$

Where:

N	is the number of points.
$C_{xy}(n)$	are the calculated correlation coefficients in the range −1, 1 for the functions x(t), y(t).
$R_{xy}(n)$	is the raw integer result for the correlation function estimate returned by CORREL for the functions x(t), y(t).
$R_{xx}(0)$	is the value of the auto-correlation function estimate for x(t) for a zero displacement, that is, for n = 0.
$R_{yy}(0)$	is the value of the auto-correlation function estimate for y(t) for a zero displacement, that is, for n = 0.
S_{xy}, S_{xx}, S_{yy}	are the scale factors indicated by CORREL when calculating R_{xy}, R_{xx}, and R_{yy} respectively. (For an illustration, see the programming example in Section 9.7.)

For many applications, the scaled results of the correlation function provide as much, if not more, information than do the correlation coefficients.

9.4 How to Call CORREL

The general format for the FORTRAN call to CORREL is:

```
CALL CORREL(IERROR,N,IA1,IA2,IFFTA,ISCALE)
```

For reference, argument names in the call to CORREL have been assigned arbitrarily. You may 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, either accidentally or on purpose, or if you supply too many arguments, a FORTRAN error message results, and no data is processed. The arguments are described in the following discussion.

IERROR is an integer variable used to report error conditions.
Because CORREL calls the FFT subroutine, the values IERROR can return are the same as those returned by the IERROR argument in the FFT subroutine.

The values that IERROR can return, and their meanings are as follows:

0 = No error

1 = N is less than eight

2 = N is greater than F.MAXN (see Section 6.6.3 for a description of F.MAXN)

3 = N is not a power of two

-N = Incorrect number of arguments in call

N is an integer variable that specifies the number of samples in both functions to be correlated. N must be a value that is a power of two in the range of 8 to F.MAXN.

IA1 is an integer array at least N elements in length that contains the samples of the first function to be correlated. CORREL returns its results to this array, replacing the input values.

IA2 is an integer array at least N elements in length that contains the samples of the second function to be correlated.

If the array specified for IA1 has the same name and/or occupies the same location as the array specified for IA2 (in other words, if the two arrays are equivalenced), CORREL performs the average auto-correlation function. If IA1 has a different name and occupies a different location from the array specified for IA2, CORREL performs the average cross-correlation function.

IFFTA is an integer array used for the imaginary part(s) of the input to the FFT subroutine.

The length of IFFTA depends on whether CORREL is to perform an auto-correlation or a cross-correlation. If CORREL is to perform an auto-correlation, then IFFTA must be N elements long. If CORREL is to perform a cross-correlation, then IFFTA must be 2 · N elements long.

NOTE

The original values in IA1, IA2, and IFFTA are altered upon return from the subroutine.

ISCALE is an integer variable that is set by CORREL. It indicates the scaling factor, that is, the number of times the results of the correlation function have been divided by two. CORREL sets ISCALE as necessary to avoid overflow.

To obtain the unscaled results of the correlation function, you must multiply the real and imaginary output of the subroutine by 2^{ISCALE}.

9.5 Other Routines Used

CORREL uses the FFT subroutine. Thus, you must include both the CORREL object module and the FFT object module when you link or task-build a FORTRAN program that calls CORREL.

See Appendixes A and B for a detailed description of how to build a FORTRAN program.

9.6 Modifying the Subroutine — Using Options

The following sections explain which options you can use with the correlation function subroutine. If you want to use any of the options, you must enable them when you build the subroutine from the source file using the interactive build procedure (see Section 1.1).

9.6.1 EIS (Extended Instruction Set)

Enable this option if your installation has EIS (KE11–E) hardware or any other floating-point option available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 26 words.

9.6.2 EAE (Extended Arithmetic Element)

Enable this option if your installation has EAE (KE11) hardware available. Enabling this option increases the execution speed and decreases the memory requirements for the subroutine by approximately 20 words.

9.6.3 FFT Options

Because CORREL calls the FFT subroutine, the FFT options have a direct influence on CORREL's operation. Thus, you should take the FFT options into consideration whenever you modify CORREL.

See Section 6.6 for a description of the FFT options.

9.7 Example Using the CORREL Subroutine

The example presented here illustrates how to use CORREL to find both the average cross-correlation and the average auto-correlation functions of a specific set of input data.

The example program computes the unscaled and scaled average cross-correlation function of 32 sine values and 32 cosine values scaled by 1000. Then, the example program computes the value of the average auto-correlation function, with zero shift, of 32 sine values and 32 cosine values. Finally, the example program computes the average cross-correlation coefficients of the 32 sine and 32 cosine values.

CORREL Example #1

CORREL Example #1

```
      DIMENSION ICOS(32),ISIN(32),ICOS2(32),ISIN2(32)
      DIMENSION ISTORE(64),COR(32)
      COMPLEX*8 ERRMES(2,3)

      DATA ERRMES/'LESS THA','N EIGHT ','EXCEEDS ','F.MAXN   ',
     1'NOT A PO','WER OF 2'/
      DATA PI/3.14159/,T/0./C/.01/
      DT=PI/16.

      TYPE 900

      DO 1 I=1,32
      D=COS(T)
      E=SIN(T)
      ICOS(I)=D*1000.+SIGN(C,D)
      ISIN(I)=E*1000.+SIGN(C,E)
      ICOS2(I)=ICOS(I)
      ISIN2(I)=ISIN(I)
    1 T=T+DT

      CALL CORREL(IERROR,32,ISIN,ICOS,ISTORE,ISCALE)
      IF (IERROR.NE.0) GO TO 20
   10 TYPE 1000,ISIN
      FAC=2.**IABS(ISCALE)
      IF(ISCALE.LT.0) FAC=1./FAC
      DO 2 I=1,32
    2 COR(I)=FAC*ISIN(I)
      TYPE 1001,COR

      CALL CORREL(IERROR,32,ISIN2,ISIN2,ISTORE,ISCALE)
      IF(IERROR.NE.0) GO TO 20
      FAC=2.**IABS(ISCALE)
      IF(ISCALE.LT.0)FAC=1./FAC
      CSIN0=FAC*ISIN2(1)

      CALL CORREL(IERROR,32,ICOS2,ICOS2,ISTORE,ISCALE)
      IF(IERROR.NE.0) GO TO 20
      FAC=2.**IABS(ISCALE)
      IF(ISCALE.LT.0)FAC=1./FAC
      CCOS0=FAC*ICOS2(1)

      FAC=SQRT(CSIN0*CCOS0)
      DO 3 I=1,32
    3 COR(I)=COR(I)/FAC
      TYPE 1002,CSIN0,CCOS0,COR
      STOP

   20 TYPE 2000,(ERRMES(I,IERROR),I=1,2)
      STOP

  900 FORMAT(1H1,T25,'CORREL Example #1')
 1000 FORMAT(///' UNSCALED RESULTS OF CORRELATION OF SINE AND COSINE'/,
     1 /,(4I17))
 1001 FORMAT(///' SCALED RESULTS OF CORRELATION OF SINE AND COSINE'//,
     1 (1P4E17.5))
 1002 FORMAT(///' VALUE OF AUTO-CORRELATION OF SIN*1000 WITH ZERO SHIFT
     1 = ',1PE10.4,//,' VALUE OF AUTO-CORRELATION OF COS*1000 WITH ZERO
     2 SHIFT = ',1PE10.4,//,' CROSS-CORRELATION COEFFICIENTS FOR SIN*100
     3 0 BY COS*1000 ARE AS FOLLOWS:',//,(1P4E17.5))
 2000 FORMAT(///' ****ERROR**** ARRAY LENGTH ',4A4)
      END
```

① Dimension array variables and initialize the error message variable ERRMES.

② Initialize arrays for input to the subroutine. Sine and cosine values · 1000 for one period.

③ Call CORREL to produce the unscaled results of cross-correlation of sine values and cosine values. Print unscaled results of cross-correlation of sine and cosine values. If IERROR is not zero, print the proper error message. Compute the scaling factor. Print scaled results of cross-correlation of sine values and cosine values.

④ Call CORREL to produce auto-correlation of sine values scaled by 1000. If IERROR is not equal to zero, print the appropriate error message. Compute auto-correlation of sine values scaled by 1000.

⑤ Call CORREL to produce auto-correlation of cosine values scaled by 1000. If IERROR is not equal to zero, print the appropriate error message. Compute value of auto-correlation of cosine values scaled by 1000.

⑥ Compute the cross-correlation coefficients for scaled sine and cosine values.

⑦ Print error message (if error code returned).

⑧ Format statements

Terminal Output

```
                    CORREL Example #1

UNSCALED RESULTS OF CORRELATION OF SINE AND COSINE

           -5         6085        11939        17335
        22065        25943        28828        30605
        31213        30608        28833        25948
        22065        17337        11940         6089
            5        -6085       -11939       -17335
       -22065       -25943       -28828       -30605
       -31213       -30608       -28833       -25948
       -22065       -17337       -11940        -6089


SCALED RESULTS OF CORRELATION OF SINE AND COSINE

 -8.00000E+01  9.73600E+04  1.91024E+05  2.77360E+05
  3.53040E+05  4.15088E+05  4.61248E+05  4.89680E+05
  4.99408E+05  4.89728E+05  4.61328E+05  4.15168E+05
  3.53040E+05  2.77392E+05  1.91040E+05  9.74240E+04
  8.00000E+01 -9.73600E+04 -1.91024E+05 -2.77360E+05
 -3.53040E+05 -4.15088E+05 -4.61248E+05 -4.89680E+05
 -4.99408E+05 -4.89728E+05 -4.61328E+05 -4.15168E+05
 -3.53040E+05 -2.77392E+05 -1.91040E+05 -9.74240E+04

VALUE OF AUTO-CORRELATION OF SIN*1000 WITH ZERO SHIFT = 4.9930E+05

VALUE OF AUTO-CORRELATION OF COS*1000 WITH ZERO SHIFT = 4.9928E+05

CROSS-CORRELATION COEFFICIENTS FOR SIN*1000 BY COS*1000 ARE AS FOLLOWS:

 -1.60228E-04  1.94998E-01  3.82593E-01  5.55511E-01
  7.07087E-01  8.31360E-01  9.23811E-01  9.80757E-01
  1.00024E+00  9.80853E-01  9.23972E-01  8.31520E-01
  7.07087E-01  5.55575E-01  3.82625E-01  1.95126E-01
  1.60228E-04 -1.94998E-01 -3.82593E-01 -5.55511E-01
 -7.07087E-01 -8.31360E-01 -9.23811E-01 -9.80757E-01
 -1.00024E+00 -9.80853E-01 -9.23972E-01 -8.31520E-01
 -7.07087E-01 -5.55575E-01 -3.82625E-01 -1.95126E-01
```

Appendix A
Installing, Verifying, and Using LSP Under RT–11

This appendix explains how to install and verify the Laboratory Subroutines Package (LSP) software and how to create a FORTRAN IV program that calls the Laboratory Subroutines under the RT–11 operating system.

However, the appendix provides only general information about creating programs. It does not include detailed information about the FORTRAN IV programming language, the FORTRAN IV compiler, the MACRO assembler, or the RT–11 operating system. For additional information on these topics, refer to the appropriate RT–11 or FORTRAN IV documentation cited in this appendix.

A.1 Installation Requirements

Instructions in this appendix require that:

1. All files acted upon by system programs are on the default device, DK:, unless otherwise noted in this appendix.
2. You use the default files types which are:

.FOR	FORTRAN source files
.MAC	MACRO source files
.OBJ	object files and object library files
.SAV	image (executable) background files

3. The system MACRO library, SYSMAC.SML, and the MACRO assembler, MACRO.SAV, have already been built and reside on the system device, SY:.

4. The FORTRAN IV compiler, FORTRA.SAV, has already been built and resides on the system device, SY:.

5. The FORTRAN Object Time System, OTS.SAV, has been built and added to the system library, SYSLIB.OBJ, which resides on the system device, SY:.

6. The system utility programs, PIP.SAV, DUP.SAV, the RT–11 linker, LINK.SAV, and the RT–11 librarian, LIBR.SAV, have been installed and reside on the system device, SY:.

7. All necessary system programs and files are installed and reside on the system device, SY:.

A.2 Installing the Laboratory Subroutines Software

Installing the Laboratory Subroutines software consists of copying the LSP distribution volume and making any necessary corrections to the LSP software.

If you wish to enable any of the options described in this manual, you must also run the file LSPMAK.SAV which is part of your distributed LSP software. LSPMAK builds the subroutines by assembling them with the options enabled. You can also use LSPMAK to build the subroutines without any options, but if you do not wish to use options, you do not need to run LSPMAK. Section A.2.4.2 explains how to use LSPMAK.

A.2.1 Copying the Distribution Volume

Because all storage media can be adversely affected by environmental conditions, vandalism, and human error, you should keep several copies of any software that cannot be easily re-created. Copy the LSP distribution volume and then store it in a safe place. Use the distribution volume only when copying the LSP software. Use only copies for all other procedures described in this appendix.

The LSP software is distributed on a single volume that RT–11 can read. The procedure for copying the distribution volume varies depending on how many mass storage devices you have. If you have three or more mass storage devices, follow the procedure in Section A.2.1.1. If you have only two mass storage devices, follow the procedure in Section A.2.1.2.

NOTE

Instructions for copying one volume to another use the SQUEEZE command rather than the COPY command. The SQUEEZE command copies the protection code of files; the COPY command does not. For additional information on removing and restoring the protection code of files, see Section A.2.2.

A.2.1.1 Copying with Three or More Mass Storage Devices — To copy the LSP distribution volume with three or more mass storage devices, do the following:

1. Load the distribution volume.
2. Load a blank storage volume.
3. Format the blank storage volume if necessary. If you use RK05 disks, format each disk. If you use RX02 drives, format each diskette to be either single density or double density. Other storage media cannot be formatted.

   Use the FORMAT command to format your disks and diskettes. The following example formats an RK05 disk:

```
.FORMAT RK1:RET
RK1:/FORMAT-Are you sure?Y(RET)
?FORMAT-I-Formatting complete
```

   When you use the FORMAT command, diskettes on RX02 drives are formatted to be double density by default. If you want them to be single density, use the /SINGLEDENSITY switch with the FORMAT command. The following example formats a diskette to be single density:

```
.FORMAT/SINGLEDENSITY DY1:RET
DY1:/FORMAT-Are you sure?Y(RET)
?FORMAT-I-Formatting complete
```

   See the *RT–11 System User's Guide* for more information about the FORMAT command.
4. Initialize the blank storage volume using the /BADBLOCKS switch to search for and isolate bad blocks. Type:

```
.INITIALIZE/BADBLOCKS dvn:RET
dvn:/Initialize; Are you sure? Y(RET)
DUP-I-No bad blocks detected dvn:
```

   See the *RT–11 System User's Guide* for more information about the INITIALIZE command and what the system does if it finds bad blocks.
5. Use the SQUEEZE command with the /OUTPUT switch to copy files from the distribution volume to the blank storage volume. Using the SQUEEZE command with the /OUTPUT switch copies all files from the input (distribution) volume to the beginning of the output (storage) volume and consolidates all empty space on the output volume at the end of that volume. The following example copies all files from dv1: to dv2:

```
.SQUEEZE/OUTPUT:dv2: dv1:RET
```

   See the *RT–11 System User's Guide* for more information about the SQUEEZE command and copying files.

A.2.1.2 Copying with only Two Mass Storage Devices — If you have only two mass storage devices, you cannot copy the LSP distribution volume directly since the RT–11 system volume occupies one of your mass storage devices. To copy the distribution volume under these circumstances, do the following:

1. Load a blank storage volume.
2. Format the storage volume if necessary. If you use RK05 disks, format each disk. If you use RX02 drives, format each diskette to be either single density or double density. Other storage media cannot be formatted.

   Use the FORMAT command to format your disks and diskettes. The following example formats an RK05 disk:

```
.FORMAT RK1:RET
RK1:/FORMAT-Are you sure?YRET
?FORMAT-I-Formatting complete
```

   When you use the FORMAT command, diskettes on RX02 drives are formatted to be double density by default. If you want them to be single density, use the /SINGLEDENSITY switch with the FORMAT command. The following example formats a diskette to be single density:

```
.FORMAT/SINGLEDENSITY DY1:RET
DY1:/FORMAT-Are you sure?YRET
?FORMAT-I-Formatting complete
```

   See the *RT–11 System User's Guide* for more information about the FORMAT command.
3. Initialize the blank storage volume using the /BADBLOCKS switch to search for and isolate bad blocks. Type:

```
.INITIALIZE/BADBLOCKS dvn:RET
dvn:/Initialize; Are you sure? YRET
DUP-I-No bad blocks detected dvn:
```

   See the *RT–11 System User's Guide* for more information about the INITIALIZE command and what the system does if it finds bad blocks.
4. Use the SQUEEZE command with both the /WAIT and the /OUTPUT switches to copy files from the distribution volume to the blank storage volume. Using the /WAIT switch causes RT–11 to pause before copying files, allowing you to load a blank storage volume, if necessary, and to remove the RT–11 system volume from the system device. Then you can load the LSP distribution volume in the system device. RT–11 prompts you with messages telling you when to do each of these things. Answer YRET to each message when you are ready to continue.

   The /OUTPUT switch causes RT–11 to copy all files from the input (distribution) volume to the beginning of the output (storage) volume

and consolidates all empty space on the output volume at the end of that volume. After RT–11 finishes copying the files, it prompts you with a message telling you to replace the system volume in the system device. The following example copies all files from dv0: to dv1:

```
.SQUEEZE/WAIT/OUTPUT:dv1: dv0:RET
Mount output volume in dv1:; Continue? YRET
Mount input volume in dv0:; Continue? YRET
```

At this point, RT–11 copies the files and then types the following message on your terminal:

```
Mount system volume in dv0:; Continue? YRET
```

See the *RT–11 System User's Guide* for more information about the SQUEEZE command and copying files.

A.2.2 File Protection

Remember that all files in the LSP distribution volume have been protected. Never remove this protection. However, after copying the distribution volume, you may remove the protection of files from your copies, if you wish, by using the RENAME command with the /NOPROTECTION switch. To protect an unprotected file, use the /PROTECTION switch with the RENAME command.

See the *RT–11 System User's Guide* for more information about the RENAME command.

A.2.3 Making Corrections

From time to time, the *RT–11 Software Dispatch* publishes corrections that you must make to the LSP software. The dispatch also publishes instructions on how to make the corrections.

You only need to make corrections published in the dispatch for version 1.2 of the LSP software. Corrections that were published in the dispatch for previous versions of LSP have already been included in version 1.2.

Never make corrections to your distribution volume. Make corrections to your copies. After making any corrections, copy your software again.

A.2.4 Selecting the Form of Subroutine to Use

You receive the Laboratory Subroutines as both object files and MACRO source files. Read the following sections to determine which form of the subroutine to use.

A.2.4.1 Using Distributed Object Files — If you do not want to enable any of the options described in this manual, you can use the distributed object files by linking them to your FORTRAN programs. (See Section 1.2 for a list of

the LSP files.) Before you do so, however, test them to verify that they work correctly by running the program LSPVER.COM which is part of your distributed LSP software. See Section A.3 for information about how to run LSPVER.COM.

A.2.4.2 Creating Customized Object Files — LSPMAK, the Interactive Build Procedure — If you do want to enable any of the options described in this manual, use the interactive build procedure, LSPMAK.SAV, to create customized object files of the Laboratory Subroutines from the distributed source files. (See Section 1.2 for a list of the LSP files.)

LSPMAK.SAV is part of your distributed LSP software. It lets you specify the subroutines you want to assemble and the options you want to use. Options available include:

1. library option — lets you place the subroutines in a library which you create by supplying a library name.
2. hardware options — let you make use of EIS hardware (or any floating-point option) or EAE hardware if you have any.
3. subroutine options — let you specify which subroutines you want to assemble and which option you want to use for each subroutine.

When you run LSPMAK.SAV, it prompts you with questions. Most questions give the name of a subroutine or option. Answer "yes" if you want to build the subroutine or enable the option. Answer "no" if you do not. Some questions request information and tell you how to type your answers. Follow the instructions given when you respond. After you answer all the questions, LSPMAK creates three files on your output device as follows:

LSPCND.MAC — This file sets the switches to enable the options you requested.

LSPBLD.COM — This indirect-command file builds each subroutine you requested. Building consists of assembling each subroutine you specifed with the switches set to enable the options you chose. If you specified a library while running LSPMAK, LSPBLD creates that library and includes in it the subroutines you specified.

LSPVER.COM — This indirect-command file verifies that your LSP software is in good working order. It does this by running an example program that tests each subroutine you asked to build.

Before you attempt to run LSPMAK.SAV, assign logical device names to:

1. The device containing all the Laboratory Subroutines software. Assign this device the logical name "IN" for "input device." Type:

   ```
   .ASSIGN dvn: IN:RET
   ```

2. The device receiving the assembled object files. Assign this device the logical name "OU" for "output device." Type:

   ```
   .ASSIGN dvn: OU:RET
   ```

Now run the file LSPMAK.SAV. Type:

```
.RUN IN:LSPMAK(RET)
```

The build procedure runs and begins typing questions and information on your terminal. The following text shows all the questions and information that LSPMAK can type. However, when you run LSPMAK, not all the questions and information will appear since some of it depends on your responses to previous questions. A sample of LSPMAK.SAV and its output appears in Appendix C.

```
Laboratory Subroutines V1.2 Build Procedure for RT-11



Have you assigned devices for input (IN:) and output (OU:)? [Y/N]

NOTE: This procedure assumes that all distributed files for
      the Laboratory Subroutines package are on device "IN".

      This procedure directs all output and temporary files
      (except library files) to device "OU".


LIBRARY OPTION


Do you want to build a NEW library file from these subroutines?
        [Y/N]
Enter the specification (maximum of 14 characters) of the
desired library file. (example DK:LSPLIB.OBJ)
ENTER NAME:


As each subroutine is added to the library, its corresponding
object file is normally deleted from device "OU":
Is this acceptable? [Y/N]


HARDWARE OPTIONS


Does your machine have the EIS option (or any floating point option)?
        (the EIS option) [Y/N]

Does your machine have the EAE option?
        (the EAE option) [Y/N]


SUBROUTINE AND ALGORITHM OPTIONS


Do you want to build the subroutine "PEAK"? [Y/N]
Do you want to disable the software digital filter?
        (enable the NOFLT$ option) [Y/N]
```

```
Do you want to enable double precision input data processing by PEAK?
        (the DPP$ option) [Y/N]
Do you want to enable processing of coded A/D input data by PEAK?
        (the AUTOG$ option) [Y/N]


Do you want to build the subroutine "NVELOP"? [Y/N]


Do you want to build the subroutine "HISTI"? [Y/N]
Do you want to enable HISTI to produce a frequency histogram?
        (the FREQ$ option) [Y/N]
Do you want to enable double precision input data processing by HISTI?
        (the DPH$ option) [Y/N]


Do you want to build the subroutine "RHISTI"? [Y/N]
Do you want to enable double precision input data processing by RHISTI?
        (the DPR$ option) [Y/N]


Do you want to build the subroutine "FFT"? [Y/N]
What is the maximum length of any input array to be processed by FFT
        (the F.MAXN option)
              1024? [Y/N]
              2048? [Y/N]
              4096? [Y/N]
              8192? [Y/N]
***THE DEFAULT INPUT ARRAY LENGTH WILL BE USED (1024)!***


Do you want to build the subroutine "PHAMPL"? [Y/N]


Do you want to build the subroutine "POWRSP"? [Y/N]


NOTE: CORREL needs subroutine FFT to function!


Do you want to build the subroutine "CORREL"? [Y/N]

This first portion of the Laboratory Subroutines build
procedure is complete.

File LSPCND.MAC has been created on device "OU". This file
sets the switches required to enable the options that you
have requested.

File LSPBLD.COM has also been created on device "OU". You
should execute this indirect command file next. It will
build each subroutine requested on device "OU" and create
the library file specified.

Finally, the file LSPVER.COM has been created on device "OU".
You should execute this command file after using LSPBLD.COM
to build your customized Laboratory Subroutines object files.
This command file will verify that your software has been
successfully installed.

STOP --
```

Now run the indirect-command file LSPBLD.COM. This file builds the subroutines you requested on device "OU" and creates a library file on that device if you specified one. Type:

```
.@OU:LSPBLD
```

A.3 Verifying the Laboratory Subroutines Software

After installing the LSP software, test it to verify that you performed the installation procedure correctly, and that your LSP software was delivered in good working order. To do so, run the indirect command file, LSPVER.COM. Instructions for running LSPVER.COM follow. If you plan to use the distributed object files, follow the instructions in Section A.3.1. If you used LSPMAK, follow the instructions in Section A.3.2.

A.3.1 Verifying the Distributed LSP Object Files

To verify the distributed LSP object files, use the distributed version of LSPVER.COM. The distributed version of LSPVER.COM runs the first example program in each chapter of this manual. These programs call the distributed subroutine object files thus indicating whether they work correctly. Before attempting to run LSPVER, do the following:

1. Make sure that your system meets the requirements listed in Section A.1.
2. Assign logical device names. Assign the logical name "IN" to the device containing the Laboratory Subroutines software. Assign the logical name "OU" to the device which will be your output device.

   To assign an input device type:

   ```
   .ASSIGN dvn: IN:
   ```

   To assign an output device, type:

   ```
   .ASSIGN dvn: OU:
   ```

3. Copy all object files for the Laboratory Subroutines to your output device. See Section 1.2 for a list of the LSP files.
4. Now run LSPVER. Type:

   ```
   .@IN:LSPVER
   ```

LSPVER types the name and number of each example program it runs along with the output from that program on your terminal. Compare the output with that of the corresponding example program in the manual. If they do not agree, and if you are sure you have not neglected any of the requirements listed in Section A.1, you may have received a defective copy of your LSP software. Contact DIGITAL for more information.

A.3.2 Verifying the Customized Object Files

To verify your customized object files, use the indirect-command file LSPVER.COM that was created when you ran LSPMAK. LSPVER.COM runs the example program that calls the version of the subroutine with the options you enabled. To run LSPVER, do the following:

1. Make sure that the device assignments for "IN" and "OU" in Section A.2.4.2 are still in effect. If they are not, assign them so they are the same as when you ran LSPMAK.
2. Now run LSPVER. Type:

```
.@OU:LSPVER(RET)
```

LSPVER types the name and number of each example program it runs along with the output from that program on your terminal. Compare the output with that of the corresponding example program in the manual. If they do not agree, and if you are sure you have not neglected any of the requirements listed in Section A.1, you may have received a defective copy of your LSP software. Contact DIGITAL for more information.

A.4 Storing the Laboratory Subroutines

After determining that the Laboratory Subroutines you want to use are sound, copy them to your system volume or to the development volume where you store your FORTRAN programs, or put them in a library (see Section A.6).

A.5 Creating a Program that Calls the Laboratory Subroutines

To create a FORTRAN IV program that calls the Laboratory Subroutines, do the following:

1. Write and check your program.
2. Use one of the RT–11 editors, such as EDIT, to enter your program into a source file. The EDIT command with the /CREATE switch invokes EDIT and tells it to create a new file. Type:

```
.EDIT/CREATE prog.FOR(RET)
```

   where: prog is the name of your FORTRAN source program.

   For information about entering the text of your file, making changes, and displaying the file, see the *Introduction to RT–11* and the *RT–11 System User's Guide*.

NOTE

Always specify a file type when you use an RT–11 editor. RT–11 editors do not use default file types. In this case, give your source file the type .FOR since that is the default file type the FORTRAN IV compiler uses when it compiles your program.

3. Use the FORTRAN IV compiler to create an object file of your program. Type:

```
.FORTRAN prog
```

where: prog is the name of your FORTRAN source program.

4. Use the RT–11 linker to link the object file of your program with the object files of your Laboratory Subroutines. Type:

```
.LINK prog,sub1,sub2,...subn
```

where: prog is the name of your FORTRAN program.

sub1 is the name of the first Laboratory Subroutine object file you want to link.

subn is the name of the last Laboratory Subroutine object file you want to link.

To avoid linking individual subroutines to your program, you can place the subroutines in a library. See Section A.6.

5. Run the program. Type:

```
.RUN prog
```

where: prog is the name of your executable program.

For more information about compiling, linking, and running FORTRAN IV programs, see the *RT–11 System User's Guide*, and the *RT–11/RSTS/E FORTRAN IV User's Guide*.

A.6 Using Libraries

Once you decide which of the Laboratory Subroutines you will use most frequently, you can link them more easily to your programs by placing them in a library. When you place them in a library, you do not need to list each individual subroutine in the link command line. You only need to list the library name. When you place them in the system library, SYSLIB.OBJ, you do not even need to list the library name in the link command line. The RT–11 linker automatically searches the system library for a subroutine or function needed by a program.

For example, suppose your compiled program, MYPROG.OBJ calls the subroutines FPEAK.OBJ and F4FFT.OBJ. To link MYPROG, you have to type:

```
.LINK MYPROG,FPEAK,F4FFT
```

If you place the subroutines in a library called, for example, MYLIB.OBJ, you link MYPROG by typing:

```
.LINK MYPROG,MYLIB
```

If you add the subroutines to the system library, SYSLIB.OBJ, you link MYPROG by typing:

```
.LINK MYPROG
```

The RT–11 librarian program, LIBR.SAV lets you add object files or object file libraries to SYSLIB. When you do so, however, you must remove certain global symbols from SYSLIB's directory for the library to function properly. Remove the global symbol $OVRH each time you add files or libraries to SYSLIB. Also remove $ERRS and $ERRTB if you previously added the FORTRAN IV OTS library to SYSLIB.

To add individual object files or a library to a system library containing the distributed SYSLIB.OBJ file and the FORTRAN IV OTS library, use the LIBRARY command with the /REMOVE switch. The LIBRARY command invokes LIBR.SAV. The /REMOVE switch allows you to remove a global symbol from a library directory. The following example adds an object file residing on dv1: to SYSLIB on dv0: and removes the appropriate global symbols:

```
.LIBRARY/REMOVE dv0:SYSLIB dv1:file.OBJ
Global? $OVRH
Global? $ERRS
Global? $ERRTB
Global? (any other global previously removed from either
         library)
Global?
```

where: file is the name of the file or library you want to place in the library.

If you fail to remove any of the necessary global symbols when you execute this command, you will get a warning message for each symbol you failed to remove. An example of the warning message is:

```
?LIBR-W-Illegal insert of $OVRH
```

If you forget which global symbols to remove, create a dummy library file by typing a command such as the following:

```
.LIBRARY/CREATE dummy SYSLIB
```

where: dummy is the name of a dummy library file.

Executing this command will produce error messages that list all the global symbols you should remove. After noting the names of the global symbols, delete the dummy library file and type the correct command for adding files to SYSLIB.

For more information about the LIBRARY command and the RT–11 librarian program LIBR.SAV, see the *RT–11 System User's Guide*.

Appendix B
Installing, Verifying, and Using LSP Under RSX–11M/M–PLUS

This appendix explains how to install and verify the Laboratory Subroutines Package (LSP) software and how to create a FORTRAN program that calls the Laboratory Subroutines under the RSX–11M/M–PLUS operating systems.

However, the appendix provides only general information about creating programs. It does not include detailed information about the FORTRAN IV or FORTRAN 77 programming languages, the FORTRAN IV or FORTRAN 77 compilers, the MACRO assembler, or the RSX–11M/M–PLUS operating systems. For additional information on these topics, refer to the appropriate RSX–11M/M–PLUS, FORTRAN IV, or FORTRAN 77 documentation cited in this appendix.

B.1 Installation Requirements

Instructions in this appendix require that:

1. All files acted upon by system programs are on the default device, SY:, under your UIC, unless otherwise noted in this appendix.
2. You use the default file extensions which are:

.FTN	FORTRAN source files
.MAC	MACRO source files
.OBJ	object files
.OLB	object library files
.TSK	task-image (executable) files

3. The system MACRO libraries, RSXMAC.SML and EXEMC.MLB, have already been built and reside on the system device, SY:.
4. Either the FORTRAN IV or the FORTRAN 77 compiler has already been built and resides on the system device, SY:.

5. Either the FORTRAN Object Time System (FOROTS) or the FORTRAN 77 Object Time System (F4POTS) has been built and added to the system library, SYSLIB.OLB, which resides on the system device, SY:. Make sure that only the object time system you will use is added to the system library.

6. If you use FORTRAN 77 and you want to duplicate the terminal output for the example programs, replace the standard random-number generator in F4POTS with F4PRAN.OBJ. Terminal output for the example programs is based on the FORTRAN IV random-number generator. The FORTRAN 77 random-number generator is different from that for FORTRAN IV and will not produce the same output. To replace the standard random-number generator with F4PRAN.OBJ, set the UIC to [1,1] and use the following command:

```
>LBR SYSLIB=dvn:[g,m]F4PRAN/RP
```

   where: dvn: is the device where F4PRAN.OBJ resides

   [g,m] is the UIC that F4PRAN.OBJ is stored under

7. System utility programs, for example DSC, FLX, LBR, and the RSX–11M task-builder, TKB, have been installed and reside on the system device, SY:.

8. All necessary system programs and files are installed and reside on the system device, SY:.

9. All tasks and MCR functions are installed.

B.2 Installing the Laboratory Subroutines Software

Installing the Laboratory Subroutines software consists of copying the LSP distribution volume and making any necessary corrections to the LSP software.

If you wish to run any of the options described in this manual, you must also run the file LSPMAK.TSK which is part of your distributed LSP software. LSPMAK builds the subroutines by assembling them with the options enabled. You can also use LSPMAK to build the subroutines without any options. Section B.2.3.2 explains how to use LSPMAK.

B.2.1 Copying the Distribution Volume

Because all storage media can be adversely affected by environmental conditions, vandalism, and human error, it is a good idea to keep several copies of any software that cannot be easily recreated. Copy the LSP distribution volume and then store it in a safe place. Use the distribution volume only when copying the LSP software. Use only copies for all other procedures described in this appendix.

The LSP software is distributed on a single volume. Most of the LSP distribution volumes are in FILES–11 format which RSX–11M/M–PLUS can read. Magnetic tape distribution volumes, however, are in DOS–11 format which RSX–11M/M–PLUS cannot read.

Copy procedures vary depending on the type of distribution volume and the number of mass storage devices you have:

- If your distribution volume is a FILES–11 volume such as an RK05, RK06, RL01, or RL02 disk and you have three or more mass storage devices, follow the instructions in Section B.2.1.1.
- If your distribution volume is a FILES–11 volume and you have only two mass storage devices, follow the instructions in Section B.2.1.2.
- If your distribution volume is a DOS–11 formatted magnetic tape, follow the instructions in Section B.2.1.3.

B.2.1.1 Copying a FILES–11 Distribution Volume with Three or More Mass Storage Devices — To copy a FILES–11 distribution volume, do the following:

1. Load the distribution volume. Allocate the drive and mount the distribution volume. The volume label for the Laboratory Subroutines is LSP. The User Identification Code (UIC) is [200,200]. Type:

```
>ALL dun:RET
>MOU dun:LSP RET
```

2. Load a blank storage volume. Allocate the drive. Type:

```
>ALL dun:RET
```

3. If you use RSX–11M–PLUS, mount the blank storage volume foreign since it has not yet been initialized as a FILES–11 volume. Type:

```
>MOU dun:/FOR RET
```

4. Format the blank storage volume if necessary. If you use RK05 disks, format each disk. If you use RX02 drives, format each diskette to be either low (single) or high (double) density. Other storage media cannot be formatted.

   Use the RSX utility FMT to format your disks and diskettes. The following example formats an RK05 disk:

```
>FMT DK2:RET

** WARNING - DATA WILL BE LOST ON DK2: **

CONTINUE? [Y OR N] Y RET

START FORMATTING

OPERATION COMPLETE

>
```

When you use FMT, diskettes on RX02 drives are formatted to be low (single) density by default. If you want them to be high (double) density, use the /DENS=HIGH option. The following example formats an RX02 diskette to be double density:

```
>FMT DY1:/DENS=HIGH<RET>

** WARNING - DATA WILL BE LOST ON DY1: **

CONTINUE? [Y OR N] Y<RET>

START FORMATTING

OPERATION COMPLETE

>
```

See the *RSX–11 Utilities Manual* for more information about FMT.

5. Run the Bad Block Locator Utility (BAD) to search for bad blocks on the storage volume. The /LI switch causes BAD to list at your terminal the decimal number of any bad blocks found. Type:

```
>BAD dvn:/LI<RET>
```

If BAD finds bad blocks, it types a message for each one found. An example of the message is:

```
BAD -- dvn: BAD BLOCK FOUND - LBN = nnnnnn
```

LBN means Logical Block Number, and nnnnnn is the decimal number of the bad block found. When BAD finishes reading the storage volume it types a message telling how many bad blocks it found:

```
BAD -- dvn: TOTAL BAD BLOCKS= n
```

If BAD found none, it types 0. If your volume has bad blocks, isolate them using the /BAD=[AUTO] option with the INI command (see step 6).

If BAD types any other message, see Chapter 9 of the *RSX–11 Utilities Manual* to find out what the message means and what to do about it.

6. Initialize the blank storage volume so it will be in FILES–11 format. Use the MCR command INI with the /BAD=[AUTO] option to isolate bad blocks on the volume. Type:

```
>INI dvn:/BAD=[AUTO]<RET>

INI -- NO BAD BLOCK DATA FOUND
>
```

If you use RSX–11M–PLUS, INI does not type the above message; it just displays its prompt to indicate it found no bad blocks.

See the *RSX–11M/M–PLUS MCR Operations Manual* for more information about using the MCR command INI.

7. If you use RSX–11M–PLUS, your storage volume is still mounted foreign although it is now in FILES–11 format. Dismount it. Type:

```
>DMOU dvn:RET
```

8. For both RSX–11M and RSX–11M–PLUS, mount the volume as a FILES–11 volume. Type:

```
>MOU dvn:RET
```

9. Use the RSX Disc Save and Compress Utility (DSC) to copy files from the distribution volume to the storage volume. The following example copies files from a distribution volume on drive 1 to a storage volume on drive 2:

```
>DSC dv2:=dv1:RET
>
```

See the *RSX–11 Utilities Manual* for more information about using DSC to copy files.

B.2.1.2 Copying a FILES–11 Distribution Volume with only Two Mass Storage Devices — If you have only two mass storage devices, you can not copy the LSP distribution volume directly since the RSX–11M/M–PLUS system volume occupies one of your mass storage devices.

To copy the distribution volume under these circumstances, use the RSX utility program DSCS8. DSCS8 is a stand alone version of DSC. Once booted, DSCS8 no longer needs the system volume. You can unload the system volume thus freeing the system device. Then you can load the LSP distribution volume in the system device and begin copying files. The following procedure explains how to copy the distribution volume using DSCS8.

NOTE

Since this procedure requires you to remove the system volume, and halt the processor, make sure that no one else is using the system when you execute the procedure.

1. Load a blank storage volume. Allocate the drive it is on. Type:

```
>ALL dvn:RET
```

2. If you use RSX–11M–PLUS, mount the blank storage volume foreign since it has not yet been initialized as a FILES–11 volume. Type:

```
>MOU dvn:/FORRET
```

3. Format the blank storage volume if necessary. If you use RK05 disks, format each disk. If you use RX02 drives, format each diskette to be either low (single) or high (double) density. Other storage media can not be formatted.

Use the RSX utility FMT to format your disks and diskettes. The following example formats an RK05 disk:

```
>FMT DK1:RET

** WARNING - DATA WILL BE LOST ON DK1: **

CONTINUE? [Y OR N] YRET

START FORMATTING

OPERATION COMPLETE

>
```

When you use FMT, diskettes on RX02 drives are formatted to be low (single) density by default. If you want them to be high (double) density, use the /DENS=HIGH option. The following example formats an RX02 diskette to be double density:

```
>FMT DY1:/DENS=HIGHRET

** WARNING - DATA WILL BE LOST ON DY1: **

CONTINUE? [Y OR N] YRET

START FORMATTING

OPERATION COMPLETE

>
```

See the *RSX–11 Utilities Manual* for more information about FMT.

4. Run the Bad Block Locator Utility (BAD) to search for bad blocks on the storage volume. The /LI switch causes BAD to list at your terminal the decimal number of any bad blocks found. Type:

   ```
   >BAD dvn:/LIRET

   BAD -- NO BAD BLOCK DATA FOUND
   ```

   If BAD finds bad blocks, it types a message for each one found. An example of the message is:

   ```
   BAD -- dvn: BAD BLOCK FOUND - LBN = nnnnnn
   ```

   LBN means Logical Block Number, and nnnnnn is the decimal number of the bad block found. When BAD finishes reading the storage volume, it types a message telling how many bad blocks it found:

   ```
   BAD -- dvn: TOTAL BAD BLOCKS= n
   ```

   If BAD found none, it types 0. If your volume has bad blocks, isolate them using the /BAD=[AUTO] option with the INI command (see step 6).

   If BAD types any other message, see Chapter 9 of the *RSX–11 Utilities Manual* to find out what the message means and what to do about it.

5. Initialize the blank storage volume so it will be in FILES–11 format. Use the MCR command INI with the /BAD=[AUTO] option to isolate bad blocks on the volume. Type:

```
>INI dvn:/BAD=[AUTO]RET

INI -- NO BAD BLOCK DATA FOUND
>
```

If you use RSX–11M–PLUS, INI does not type the above message; it just displays its prompt to indicate it found no bad blocks.

See the *RSX–11M/M–PLUS MCR Operations Manual* for more information about using the MCR command INI.

6. Boot DSCS8. The UIC is [1,51]. Type:

```
>BOOT [1,51]DSCS8RET
RXS11S V2.2 BL26 DISC SAVE AND COMPRESS UTILITY V 3.2
```

DSCS8 displays its prompt. It is now booted and ready for input.

```
DSCS8>
```

7. Remove the RSX–11M/M–PLUS system volume and load a copy of the LSP distribution volume in the system device. Now instruct DSCS8 to begin copying files. The following example copies files from a distribution volume, on drive 0 to a storage volume on drive 1:

```
DSCS8>dv1:=dv0:RET
```

Copying now begins. When it completes, DSCS8 displays its prompt.

```
DSCS8>
```

8. Now halt the processor. Remove the LSP distribution volume from the system device and replace the system volume. Boot RSX–11M or RSX–11M–PLUS.

See the *RSX–11 Utilities Manual* for more information about using DSCS8 to copy files if you have only two mass storage devices. Information about DSCS8 (sometimes called "stand-alone DSC") is in the chapter on DSC.

B.2.1.3 Copying a DOS–11 Distribution Volume — To copy a DOS–11 distribution magnetic tape, do the following:

1. Load the distribution volume. Allocate the drive and mount the distribution volume. The User Identification Code (UIC) is [200,200]. Type:

```
>ALL dvn:RET
>MOU dvn:RET
```

2. Load a blank storage volume. Allocate the drive it is on. Type:

```
>ALL dvn:RET
```

3. If you use RSX–11M–PLUS, mount the blank storage volume foreign since it has not yet been initialized as a FILES–11 volume. Type:

```
>MOU dvn:/FORRET
```

4. Format the blank storage volume if necessary. If you use RK05 disks, format each disk. If you use RX02 drives, format each diskette to be either low (single) or high (double) density. Other storage media cannot be formatted.

   Use the RSX utility FMT to format your disks and diskettes. The following example formats an RK05 disk:

```
>FMT DK1:RET

** WARNING - DATA WILL BE LOST ON DK1: **

CONTINUE? [Y OR N] YRET

START FORMATTING

OPERATION COMPLETE

>
```

   When you use FMT, diskettes on RX02 drives are formatted to be low (single) density by default. If you want them to be high (double) density, use the /DENS=HIGH option. The following example formats an RX02 diskette to be double density:

```
>FMT DY1:/DENS=HIGHRET

** WARNING - DATA WILL BE LOST ON DY1: **

CONTINUE? [Y OR N] YRET

START FORMATTING

OPERATION COMPLETE

>
```

   See the *RSX–11 Utilities Manual* for more information about FMT.

5. Run the Bad Block Locator Utility (BAD) to search for bad blocks on the storage volume. The /LI switch causes BAD to list at your terminal the decimal number of any bad blocks found. Type:

```
>BAD dvn:/LIRET

BAD -- NO BAD BLOCK DATA FOUND
```

   If BAD finds bad blocks, it types a message for each one found. An example of the message is:

```
BAD -- dvn: BAD BLOCK FOUND - LBN = nnnnnn
```

   LBN means Logical Block Number, and nnnnnn is the decimal number of the bad block found. When BAD finishes reading the storage volume, it types a message telling how many bad blocks it found:

```
BAD -- dvn: TOTAL BAD BLOCKS= n
```

   If BAD found none, it types 0. If your volume has bad blocks, isolate them using the /BAD=[AUTO] option with the INI command (see step 6).

If BAD types any other message, see Chapter 9 of the *RSX–11 Utilities Manual* to find out what the message means and what to do about it.

6. Initialize the blank storage volume so it will be in FILES–11 format. Use the MCR command INI with the /BAD=[AUTO] option to isolate the bad blocks on the volume. Type:

```
>INI dvn:/BAD=[AUTO]RET

INI -- NO BAD BLOCK DATA FOUND
>
```

   If you use RSX–11M–PLUS, INI does not type the above message; it just displays its prompt to indicate it found no bad blocks.

   See the *RSX–11M/M–PLUS MCR Operations Manual* for more information about using the MCR command INI.

7. If you use RSX–11M–PLUS, your storage volume is still mounted foreign although it is now in FILES–11 format. Dismount it. Type:

```
>DMOU dvn:RET
```

8. For both RSX–11M and RSX–11M–PLUS, mount the volume as a FILES–11 volume. Type:

```
>MOU dvn:RET
```

9. Create a UIC of [200,200] on the storage volume. Use the MCR command UFD (User File Directory). Type:

```
>UFD dvn:[200,200]RET
```

10. Use the RSX utility FLX to copy files from the DOS–11 distribution tape to your FILES–11 storage volume. The /RS switch identifies the storage volume as a FILES–11 volume, and the /DO switch identifies the distribution volume as a DOS–11 volume. The wildcard construction, *.*, tells RSX–11M/M–PLUS that you wish to copy the most recent version of all files. (You cannot specify version numbers on a DOS–11 formatted volume.)

    The following example copies files from a DOS–11 distribution volume on drive 0 to a FILES–11 storage volume on drive 1:

```
>FLX dv1:[200,200]/RS=dv0:[200,200]*.*/DORET
>
```

    See the *RSX–11 Utilities Manual* for more information about FLX.

11. Use the RSX–11M utility program PIP to create a contiguous file from LSPMAK.TSK, the interactive build procedure. Type:

```
>PIP /CO=[200,200]LSPMAK.TSKRET
```

    See the *RSX–11 Utilities Manual* for more information about PIP.

B.2.2 Making Corrections

From time to time, the *RSX–11M/S–RSX–11M–PLUS Software Dispatch* publishes corrections you must make to the LSP software. The dispatch also publishes instructions on how to make the corrections.

You only need to make corrections published in the dispatch for version 1.2 of the LSP software. Corrections that were published in the dispatch for previous versions of LSP have already been included in version 1.2.

Never make corrections to your distribution volume. Make corrections to your copies. After making any corrections, copy your software again.

B.2.3 Selecting the Form of Subroutine to Use

You receive the Laboratory Subroutines as both object files and MACRO source files. Read Sections B.2.3.1 and B.2.3.2 to determine which form of the subroutine to use.

B.2.3.1 Using Distributed Object Files — You can use the distributed object files if:

1. You do not want to enable any of the options described in this manual.
2. You are using FORTRAN IV.
3. Your system does not have a hardware floating-point unit.

You use the distributed object files by task-building them to your FORTRAN programs. (See Section 1.2 for a list of the LSP files.) Before you do so, however, test them to verify that they work correctly by running the program LSPVER.CMD which is part of your distributed LSP software. See Section B.3 for information about how to run LSPVER.CMD.

B.2.3.2 Creating Customized Object Files — LSPMAK, the Interactive Build Procedure — If you want to enable any of the options described in this manual, use the interactive build procedure, LSPMAK.TSK, to create customized object files of the Laboratory Subroutines from the distributed source files. You can also use LSPMAK to build the subroutines without any options. (See Section 1.2 for a list of the LSP files.)

LSPMAK.TSK is part of your distributed LSP software. It lets you specify the subroutines you want to assemble and the options you want to use. Options available include:

1. library option — lets you place the subroutines in a library which you create by supplying a library name.
2. hardware options — let you make use of EIS hardware (or any floating-point option) or EAE hardware if you have any.
3. FORTRAN options — let you specify whether or not you will be using FORTRAN 77.

4. subroutine options — let you specify which subroutines you want to assemble and which options you want to use for each subroutine.

When you run LSPMAK.TSK, it prompts you with questions. Most questions give the name of a subroutine or option. Answer "yes" if you want to build the subroutine or enable the option. Answer "no" if you do not. Some questions request information and tell you how to type your answers. Follow the instructions given when you respond. After you answer all the questions, LSPMAK creates three files on your output device under the current UIC as follows:

LSPCND.MAC — This file sets the switches to enable the options you requested.

LSPBLD.CMD — This indirect-command file builds each subroutine you requested. Building consists of assembling each subroutine you specified with the switches set to enable the options you chose. If you specified a library while running LSPMAK, LSPBLD creates that library and includes in it the subroutines you specified.

LSPVER.CMD — This indirect-command file verifies that your LSP software is in good working order. It does this by running an example program that tests each subroutine you asked to build.

Before you attempt to run LSPMAK.TSK, assign logical device names to:

1. The device containing all the Laboratory Subroutines software. Assign this device the logical name "IN" for "input device." Type:

```
>ASN dvn:=IN:RET
```

2. The device receiving the assembled object files. Assign this device the logical name "OU" for "output device." Type:

```
>ASN dvn:=OU:RET
```

Now run the file LSPMAK.TSK. Type:

```
>RUN IN:[200,200]LSPMAKRET
```

The build procedure runs and begins typing questions and information on your terminal. The following text shows all the questions and information that LSPMAK can type. However, when you run LSPMAK, not all the questions and information will appear since some of it depends on your responses to previous questions. A sample of LSPMAK.TSK and its output appears in Appendix D.

```
Laboratory Subroutines V1.2 Build Procedure for RSX-11M


Have you assigned devices for input (IN:) and output (OU:)? [Y/N]
```

```
NOTE:  This procedure assumes that all distributed files for
       the Laboratory Subroutines package are on device "IN"
       under UIC [200,200].


       This procedure directs all output and temporary files
       (except library files) to device "OU" under the
       default UIC.


LIBRARY OPTION


Do you want to build a NEW library file from these subroutines?
        [Y/N]
Enter the specification (maximum of 30 characters) of the
desired library file. (example SY0:[1,1]LSPLIB.OLB)
ENTER NAME:


As each subroutine is added to the library, its corresponding
object file is normally deleted from device "OU".
Is this acceptable? [Y/N]


HARDWARE OPTIONS


Does your machine have the EIS option (or any floating point option)?
        (the EIS option) [Y/N]
Does your machine have the EAE option?
        (the EAE option) [Y/N]


FORTRAN OPTIONS


Will you be using FORTRAN 77? (the F4P$ option) [Y/N]

NOTE:  The software verification command file, OU:LSPVER.CMD,
       will use FORTRAN IV to compile the Laboratory Subroutines
       test programs. The task build procedure for these test
       programs will assume that the appropriate FORTRAN IV
       object time system library is added to the system library,
       SYSLIB.OLB.

       Also, since you have responded that your system does not
       have a floating-point unit, the task build procedure
       will assume that your FORTRAN IV object time system
       does not require a floating-point unit.

NOTE:  The software verification command file, OU:LSPVER.CMD, will
       use FORTRAN 77 to compile the Laboratory Subroutines
       test programs. The task build procedure for these test
       programs will assume that the appropriate FORTRAN 77
       object time system library is added to the system library,
       SYSLIB.OLB.
```

```
SUBROUTINE AND ALGORITHM OPTIONS


Do you want to build the subroutine "PEAK"? [Y/N]
Do you want to disable the software digital filter?
        (enable the NOFLT$ option) [Y/N]
Do you want to enable double precision input data processing by PEAK?
        (the DPP$ option) [Y/N]
Do you want to enable processing of coded A/D input data by PEAK?
        (the AUTOG$ option) [Y/N]


Do you want to build the subroutine "NVELOP"? [Y/N]


Do you want to build the subroutine "HISTI"? [Y/N]
Do you want to enable HISTI to produce a frequency histogram?
        (the FREQ$ option) [Y/N]
Do you want to enable double precision input data processing by HISTI?
        (the DPH$ option) [Y/N]


Do you want to build the subroutine "RHISTI"? [Y/N]
Do you want to enable double precision input data processing by RHISTI?
        (the DPR$ option) [Y/N]


Do you want to build the subroutine "FFT"? [Y/N]
What is the maximum length of any input array to be processed by FFT
        (the F.MAXN option):
                1024? [Y/N]
                2048? [Y/N]
                4096? [Y/N]
                8192? [Y/N]
***THE DEFAULT MAXIMUM INPUT ARRAY LENGTH WILL BE USED (1024)!***


Do you want to build the subroutine "PHAMPL"? [Y/N]


Do you want to build the subroutine "POWRSP"? [Y/N]


NOTE: CORREL needs subroutine FFT to function!


Do you want to build the subroutine "CORREL"? [Y/N]


This first portion of the Laboratory Subroutines build
procedure is complete.

File LSPCND.MAC has been created on device "OU". This file
sets the switches required to enable the options that you
have requested.

File LSPBLD.CMD has also been created on device "OU". You
should execute this indirect command file next. It will
build each subroutine requested on device "OU", and create
the library file specified.
```

```
Finally, the file LSPVER.CMD has been created on device "OU".
You should execute this command file after using LSPBLD.CMD
to build your customized Laboratory Subroutines object files.
This command file will verify that your software has been
successfully installed.

TTnn -- STOP
```

Now run the indirect-command file LSPBLD.CMD. This file builds the subroutines you requested on device "OU" and creates a library on that device if you specified one. Type:

```
>@OU:LSPBLD<RET>
```

B.3 Verifying the Laboratory Subroutines Software

After installing the LSP software, test it to verify that you performed the installation procedure correctly, and that your LSP software was delivered in good working order. To do so, run the indirect-command file, LSPVER.CMD. Instructions for running LSPVER.CMD follow. If you plan to use the distributed object files, follow the instructions in Section B.3.1. If you used LSPMAK, follow the instructions in Section B.3.2.

B.3.1 Verifying the Distributed LSP Object Files

To verify the distributed LSP object files, use the distributed version of LSPVER.CMD. The distributed version of LSPVER.CMD runs the first example program in each chapter of this manual. These programs call the distributed subroutine object files thus indicating whether they work correctly. Before attempting to run LSPVER, do the following:

1. Make sure that your system meets the requirements listed in Section B.1.
2. Assign logical device names. Assign the logical name "IN" to the device containing the Laboratory Subroutines software. Assign the logical name "OU" to the device which will be your output device.

   To assign an input device, type:

   ```
   >ASN dvn:=IN:<RET>
   ```

   To assign an output device, type:

   ```
   >ASN dvn:=OU:<RET>
   ```

3. Copy all of the object files for the Laboratory Subroutines to your output device. (See Section 1.2 for a list of the LSP files.)
4. Now run LSPVER. Type:

   ```
   >@IN:[200,200]LSPVER<RET>
   ```

LSPVER types the name and number of each example program it runs along with the output from that program on your terminal. Compare the output with that of the corresponding example in the manual. If they do not agree and you are sure you have not neglected any of the requirements listed in Section B.1, you may have received a defective copy of your LSP software. Contact DIGITAL for more information.

B.3.2 Verifying the Customized Object Files

To verify your customized object files, use the version of LSPVER.CMD that was created when you ran LSPMAK. This version of LSPVER.CMD runs the example program that calls the version of the subroutine with the options you enabled. To run LSPVER, do the following:

1. Make sure that the device assignments for "IN" and "OU" in Section B.2.3.2 are still in effect. If they are not, assign them so they are the same as when you ran the LSPMAK.
2. Now run LSPVER. Type:

```
>@OU:LSPVER<RET>
```

LSPVER types the name and number of each example program it runs along with the output from that program on your terminal. Compare the output with that of the corresponding example program in the manual. If they do not agree, and if you are sure you have not neglected any of the requirements listed in Section B.1, you may have received a defective copy of your LSP software. Contact DIGITAL for more information.

B.4 Storing the Laboratory Subroutines

After determining that the Laboratory Subroutines you want to use are sound, copy them to your system volume or to the development volume where you store your FORTRAN programs, or place them in a library (see Section B.6).

B.5 Creating a Program that Calls the Laboratory Subroutines

To create a FORTRAN program that calls the Laboratory Subroutines, do the following:

1. Write and check your program.
2. Use one of the RSX–11M/M–PLUS editors, such as EDI, to enter your program into a source file. Type:

```
>EDI prog.FTN<RET>
```

   where: prog is the name of your FORTRAN source program.

   For information about entering the text of your file, making changes and displaying the file, see the *RSX–11 Utilities Manual* and the *RSX–11M/M–PLUS Guide to Program Development*.

NOTE

Always specify a file extension when you use an editor. RSX–11M/M–PLUS editors do not use default file extensions. In this case, give your source file the extension .FTN since that is the default extension the FORTRAN IV or FORTRAN 77 compiler uses when it compiles your program.

3. Use the FORTRAN IV or FORTRAN 77 compiler to create an object file of your program.

   If you are using the FORTRAN IV compiler, Type:

   ```
   >FOR prog=prog<RET>
   ```

   If you are using the FORTRAN 77 compiler, Type:

   ```
   >F77 prog=prog<RET>
   ```

   where: prog is the name of your FORTRAN source program.

4. Use the RSX–11M/M–PLUS task builder to task-build the object file of your program with the object files of your Laboratory Subroutines. The /FP switch tells the task-builder to use the floating-point unit. Use the /FP switch if you use FORTRAN IV and your system has a floating-point unit. Always use the /FP switch if you use FORTRAN 77 since your system always has a floating-point unit. Type:

   ```
   >TKB prog[/FP]=prog,sub1,sub2,...,subn<RET>
   ```

   where: prog is the name of your FORTRAN program.

   sub1 is the name of the first Laboratory Subroutine object file you want to task build.

   subn is the name of the last Laboratory Subroutine object file you want to task build.

   You can specify as many object files as you wish for input. However, if your list of input files will use more than one line on your terminal, you should see the *RSX–11M/M–PLUS Task Builder Manual* for information about how to input files using multiple lines.

   To avoid linking individual subroutines to your programs, you can place the subroutines in a library. See Section B.6.

5. Run the task. Type:

   ```
   >RUN prog<RET>
   ```

   where: prog is the name of your executable program.

For more information about compiling, linking, and running FORTRAN programs, see the *RSX–11M/M–PLUS Guide to Program Development*, the *IAS/RSX–11 FORTRAN IV User's Guide*, and the *PDP–11 FORTRAN 77 User's Guide*.

B.6 Using Libraries

Once you decide which of the Laboratory Subroutines you will use most frequently, you can task-build them to your programs more easily by placing them in a library. When you place them in a library, you do not need to list each individual subroutine in the TKB command line. You only need to list the library name. When you place them in the system library, SYSLIB.OLB, you do not even need to list the library name in the TKB command line. The RSX–11M/M–PLUS task builder automatically searches the system library for a subroutine or function needed by a program.

For example, suppose your compiled program, MYPROG.OBJ calls the subroutines FPEAK.OBJ and F4FFT.OBJ. To task-build MYPROG, you have to type:

```
>TKB MYPROG[/FP]=MYPROG,FPEAK,F4FFT
```

If you place the subroutines in a library called, for example, MYLIB.OLB, you task-build MYPROG by typing:

```
>TKB MYPROG[/FP]=MYPROG,MYLIB/LB
```

You have to use the /LB switch to tell RSX–11M/M–PLUS that MYLIB is a library file. However, if you add the subroutines to the system library, SYSLIB.OLB, you task-build MYPROG by typing:

```
>TKB MYPROG[/FP]=MYPROG
```

Note that the interactive build procedure, LSPMAK, allows you to create a library of your own. If you use this option, LSPMAK places in the library those subroutines you tell it to build in response to later questions. If you do not use this option when you first run LSPMAK, but later decide you want to use a library, you can either rerun LSPMAK or use the RSX utility program LBR to create a new library or add to an existing one.

See the *RSX–11 Utilities Manual* for information about using LBR. The *RSX–11 Utilities Manual* also tells you how to use the system library.

Appendix C
Sample of the Interactive Build Procedure for RT–11, LSPMAK.SAV

The following example shows the interactive build procedure, LSPMAK.SAV with its output.

The example does the following:

1. Places the subroutines in a new library called LSPLIB.OBJ.
2. Enables the EIS option.
3. Builds the following subroutines:
   - PEAK — with the double-precision integers option (the DPP$ option) enabled.
   - FFT — with a maximum input array length of 8192.

```
.ASSIGN DMO: IN:
.ASSIGN DLO: OU:
.RUN IN:LSPMAK

Laboratory Subroutines V1.2 Build Procedure for RT-11


Have you assigned devices for input (IN:) and output (OU:)? [Y/N] Y

NOTE:  This procedure assumes that all distributed files for
       the Laboratory Subroutines package are on device "IN".

       This procedure directs all output and temporary files
       (except library files) to device "OU".


LIBRARY OPTION


Do you want to build a NEW library file from these subroutines?
       [Y/N] Y
Enter the specification (maximum of 14 characters) of the
desired library file. (example DK:LSPLIB.OBJ)
ENTER NAME:    DLO:LSPLIB.OBJ


As each subroutine is added to the library, its corresponding
object file is normally deleted from device "OU":
Is this acceptable? [Y/N] N


HARDWARE OPTIONS


Does your machine have the EIS option (or any floating point option)?
       (the EIS option) [Y/N] Y

SUBROUTINE AND ALGORITHM OPTIONS


Do you want to build the subroutine "PEAK"? [Y/N] Y
Do you want to disable the software digital filter?
       (enable the NOFLT$ option) [Y/N] N
Do you want to enable double precision input data processing by PEAK?
       (the DPP$ option) [Y/N] Y


Do you want to build the subroutine "NVELOP"? [Y/N] N


Do you want to build the subroutine "HISTI"? [Y/N] N


Do you want to build the subroutine "RHISTI"? [Y/N] N


Do you want to build the subroutine "FFT"? [Y/N] Y
What is the maximum length of any input array to be processed by FFT
       (the F.MAXN option)
             1024? [Y/N] N
             2048? [Y/N] N
```

```
4096? [Y/N] N
8192? [Y/N] Y

Do you want to build the subroutine "PHAMPL"? [Y/N] N

Do you want to build the subroutine "POWRSP"? [Y/N] N

Do you want to build the subroutine "CORREL"? [Y/N] N

This first portion of the Laboratory Subroutines build
procedure is complete.

File LSPCND.MAC has been created on device "OU". This file
sets the switches required to enable the options that you
have requested.

File LSPBLD.COM has also been created on device "OU". You
should execute this indirect command file next. It will
build each subroutine requested on device "OU" and create
the library file specified, DL0:LSPLIB.OBJ.

Finally, the file LSPVER.COM has been created on device "OU".
You should execute this command file after using LSPBLD.COM
to build your customized Laboratory Subroutines object files.
This command file will verify that your software has been
successfully installed.

STOP --

.

.@OU:LSPBLD

.

.MACRO/OBJECT:OU:FPEAK  OU:LSPCND+IN:FPEAK
ERRORS DETECTED:  0

.LIBRARY /CREATE DL:LSPLIB.OBJ OU:FPEAK

.MACRO/OBJECT:OU:F4FFT  OU:LSPCND+IN:F4FFT
ERRORS DETECTED:  0

.LIBRARY DL:LSPLIB.OBJ OU:F4FFT

.@OU:LSPVER

.

.FORTRAN/OBJECT:OU:LSPEX IN:EX4FPE
.MAIN.

.LINK/EXECUTE:OU:LSPEX OU:LSPEX,DL:LSPLIB.OBJ

.RUN OU:LSPEX
```

```
                        PEAK Example #4

PEAK NO.            AREA      P HEIGHT     P TIME     L HEIGHT     L TIME
                 HALF WIDTH   T HEIGHT     T TIME         TYPE       RATE

       1      32158219.       952958.        20.      693113.         4.
                    11.       483071.        44.     BASELINE         1.
       2      10732416.       452258.        68.      344193.        54.
                     7.       189827.        83.     BASELINE         1.
       3     134705883.       300062.       600.       14923.       107.
                   119.       201630.       839.       VALLEY         1.

STOP --

.DELETE/NOQUERY OU:LSPEX.*

.FORTRAN/OBJECT:OU:LSPEX IN:EX1F4F
.MAIN.

.LINK/EXECUTE:OU:LSPEX OU:LSPEX,DL:LSPLIB.OBJ

.RUN OU:LSPEX

                        FFT Example #1

 DATA TO BE TRANSFORMED - SINE WAVE SCALED BY 1000.

- REAL PART -
     0      195      382      555      707      831      923      980
  1000      980      923      831      707      555      382      195
     0     -195     -382     -555     -707     -831     -923     -980
 -1000     -980     -923     -831     -707     -555     -382     -195

- IMAGINARY PART -
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0
     0        0        0        0        0        0        0        0

RESULTS FROM THE FORWARD FOURIER TRANSFORM OF 32 POINTS OF A SINE WAVE

- REAL PART -
     0       -7        0       -6        0       -2        0       -2
     0        1        0        2        0        0        0        0
     0        1        0        2        0        2        0        2
     0        1        0        2        0        0        0        0

- IMAGINARY PART -
     0   -15981        0        2        0        1        0        0
     0       -5        0       -2        0       -1        0       -4
     0       -1        0        2        0        1        0        0
     0       -1        0       -2        0       -5        0    15984
```

```
RESULTS FROM THE INVERSE FOURIER TRANSFORM

                                - BEFORE SCALING -

- REAL PART -
    -4    6217   12200   17749   22598   26578   29531   31356
 31972   31354   29527   26582   22604   17752   12223    6254
     4   -6217  -12200  -17749  -22598  -26578  -29531  -31356
-31972  -31354  -29527  -26582  -22604  -17752  -12223   -6254

- IMAGINARY PART -
   -12      -8      -9      -6      -6       4      -5      -2
    -4      -5      -6     -17       0     -14     -15     -20
    12       8       9       6       6      -4       5       2
     4       5       6      17       0      14      15      20


                                - AFTER SCALING -

- REAL PART -
     0     194     381     554     706     830     922     979
   999     979     922     830     706     554     381     195
     0    -194    -381    -554    -706    -830    -922    -979
  -999    -979    -922    -830    -706    -554    -381    -195

- IMAGINARY PART -
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
     0       0       0       0       0       0       0       0
STOP --

.DELETE/NOQUERY OU:LSPEX.*

.
```

Appendix D
Sample of the Interactive Build Procedure for RSX–11M, LSPMAK.TSK

The following example shows the interactive build procedure, LSPMAK.TSK with its output.

The example does the following:

1. Places the subroutines in a new library called LSPLIB.OLB.
2. Enables the EIS option.
3. Enables the FORTRAN 77 option (the F4P$ option).
4. Builds the following subroutines:
   - PEAK — with the double-precision integers option (the DPP$ option) enabled.
   - FFT — with a maximum input array length of 8192.

```
>ASN DM1:=IN:RET
>ASN DR2:=OU:RET
>RUN IN:[200,200]LSPMAKRET

Laboratory Subroutines V1.2 Build Procedure for RSX-11M



Have you assigned devices for input (IN:) and output (OU:)? [Y/N] YRET

NOTE:  This procedure assumes that all distributed files for
       the Laboratory Subroutines package are on device "IN"
       under UIC [200,200].

       This procedure directs all output and temporary files
       (except library files) to device "OU" under the
       default UIC.


LIBRARY OPTION


Do you want to build a NEW library file from these subroutines?
        [Y/N] YRET
Enter the specification (maximum of 30 characters) of the
desired library file. (example SY0:[1,1]LSPLIB.OLB)
ENTER NAME:  DR2:[201,201]LSPLIB.OLBRET


As each subroutine is added to the library, its corresponding
object file is normally deleted from device "OU":.
Is this acceptable? [Y/N] NRET


HARDWARE OPTIONS


Does your machine have the EIS option (or any floating point option)?
        (the EIS option) [Y/N] YRET


FORTRAN OPTIONS


Will you be using FORTRAN 77? (the F4P$ option) [Y/N] YRET


NOTE:  The software verification command file, OU:LSPVER.CMD, will
       use FORTRAN 77 to compile the Laboratory Subroutines
       test programs. The task build procedure for these test
       programs will assume that the appropriate FORTRAN 77
       object time system library is added to the system library,
       SYSLIB.OLB.

SUBROUTINE AND ALGORITHM OPTIONS


Do you want to build the subroutine "PEAK"? [Y/N] YRET
Do you want to disable the software digital filter?
        (enable the NOFLT$ option) [Y/N] NRET
Do you want to enable double precision input data processing by PEAK?
        (the DPP$ option) [Y/N] YRET
```

```
Do you want to build the subroutine "NVELOP"? [Y/N] N<RET>

Do you want to build the subroutine "HISTI"? [Y/N] N<RET>

Do you want to build the subroutine "RHISTI"? [Y/N] N<RET>

Do you want to build the subroutine "FFT"? [Y/N] Y<RET>
What is the maximum length of any input array to be processed by FFT
        (the F.MAXN option)
              1024? [Y/N] N<RET>
              2048? [Y/N] N<RET>
              4096? [Y/N] N<RET>
              8192? [Y/N] Y<RET>

Do you want to build the subroutine "PHAMPL"? [Y/N] N<RET>

Do you want to build the subroutine "POWRSP"? [Y/N] N<RET>

Do you want to build the subroutine "CORREL"? [Y/N] N<RET>

This first portion of the Laboratory Subroutines build
procedure is complete.

File LSPCND.MAC has been created on device "OU". This file
sets the switches required to enable the options that you
have requested.

File LSPBLD.CMD has also been created on device "OU". You
should execute this indirect command file next. It will
build each subroutine requested on device "OU" and create
the library file specified, DR2:[201,201]LSPLIB.OLB.

Finally, the file LSPVER.CMD has been created on device "OU".
You should execute this command file after using LSPBLD.CMD
to build your customized Laboratory Subroutines object files.
This command file will verify that your software has been
successfully installed.

TT41 -- STOP
>


>@OU:LSPBLD<RET>
>MAC OU:FPEAK =OU:LSPCND,IN:[200,200]FPEAK
>LBR DR2:[201,201]LSPLIB.OLB/CR=OU:FPEAK
>MAC OU:F4FFT =OU:LSPCND,IN:[200,200]F4FFT
>LBR DR2:[201,201]LSPLIB.OLB=OU:F4FFT
>@ <EOF>
>@OU:LSPVER<RET>
>F77 OU:LSPEX=IN:[200,200]EX4FPE
>TKB OU:LSPEX/FP =OU:LSPEX,DR2:[201,201]LSPLIB.OLB/LB
>RUN OU:LSPEX
```

PEAK Example #4

PEAK NO.	AREA	P HEIGHT	P TIME	L HEIGHT	L TIME
	HALF WIDTH	T HEIGHT	T TIME	TYPE	RATE
1	32158219.	952958.	20.	693113.	4.
	11.	483071.	44.	BASELINE	1.
2	10732416.	452258.	68.	344193.	54.
	7.	189827.	83.	BASELINE	1.
3	134705883.	300062.	600.	14923.	107.
	119.	201630.	839.	VALLEY	1.

```
>PIP OU:LSPEX.*;*/DE
>F77 OU:LSPEX=IN:[200,200]EX1F4F
>TKB OU:LSPEX/FP =OU:LSPEX,DR2:[201,201]LSPLIB.OLB/LB
>RUN OU:LSPEX
```

FFT Example #1

DATA TO BE TRANSFORMED - SINE WAVE SCALED BY 1000.

- REAL PART -

0	195	382	555	707	831	923	980
1000	980	923	831	707	555	382	195
0	-195	-382	-555	-707	-831	-923	-980
-1000	-980	-923	-831	-707	-555	-382	-195

- IMAGINARY PART -

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

RESULTS FROM THE FORWARD FOURIER TRANSFORM OF 32 POINTS OF A SINE WAVE

- REAL PART -

0	-7	0	-6	0	-2	0	-2
0	1	0	2	0	0	0	0
0	1	0	2	0	2	0	2
0	1	0	2	0	0	0	0

- IMAGINARY PART -

0	-15981	0	2	0	1	0	0
0	-5	0	-2	0	-1	0	-4
0	-1	0	2	0	1	0	0
0	-1	0	-2	0	-5	0	15984

RESULTS FROM THE INVERSE FOURIER TRANSFORM

- BEFORE SCALING -

- REAL PART -

-4	6217	12200	17749	22598	26578	29531	31356
31972	31354	29527	26582	22604	17752	12223	6254
4	-6217	-12200	-17749	-22598	-26578	-29531	-31356
-31972	-31354	-29527	-26582	-22604	-17752	-12223	-6254

- IMAGINARY PART -

-12	-8	-9	-6	-6	4	-5	-2
-4	-5	-6	-17	0	-14	-15	-20
12	8	9	6	6	-4	5	2
4	5	6	17	0	14	15	20

```
                                - AFTER SCALING -

- REAL PART -
      0      194      381      554      706      830      922      979
    999      979      922      830      706      554      381      195
      0     -194     -381     -554     -706     -830     -922     -979
   -999     -979     -922     -830     -706     -554     -381     -195

- IMAGINARY PART -
      0        0        0        0        0        0        0        0
      0        0        0        0        0        0        0        0
      0        0        0        0        0        0        0        0
      0        0        0        0        0        0        0        0
TT41 -- STOP

>PIP OU:LSPEX.*;*/DE
>@ <EOF>
>
```

Index

Algorithm,
  envelope-processing, 3–1
  peak-processing, 2–1
  PHAMPL, 7–1
  POWRSP, 8–1
Aliasing, 6–8
ALL command, B–3, B–5, B–7
Amplitude spectra, 7–1
Amplitudes, 7–1 to 7–12
Angle,
  phase, 7–1 to 7–12
Approximate Fourier transform, 6–2, 6–13
Area,
  peak, 2–1, 2–4, 3–4
Arguments,
  CORREL, 9–5 to 9–6
  FFT, 6–13
  HISTI, 4–5 to 4–6
  NVELOP, 3–16 to 3–20
  PEAK, 2–17 to 2–20
  PHAMPL, 7–2
  POWRSP, 8–2
  RHISTI, 5–7, 5–8
ASN command, B–11, B–14
Assembler,
  RSX–11M MACRO, B–1
  RT–11 MACRO, A–1
ASSIGN command, A–6, A–9
ATAN2, 7–2
Auto-Correlation, 9–1, 9–2
AUTOG$ option, 2–22, A–8, B–3
Autogain, 2–22, A–8, B–3
Average correlation function, 9–1 to 9–5
Axis,
  definition of time, 2–2, 3–2

/BAD=[AUTO]switch, B–4, B–7, B–9
  BAD command, B–4, B–6, B–8
  BAD utility, B–4, B–6, B–8
  /BADBLOCKS switch, A–3, A–4
Bar-graph, 4–1, 5–1
Baseline
  definition of, 2–6
Baseline correction, 3–22
Baseline value,
  NVELOP, 3–3, 3–17, 3–18
BOOT command, B–7
Build procedure,
  Interactive, 1–3, 1–5, A–6 to A–9, B–10 to B–14, C–1, D–1

Category,
  data and, 4–2
  definition of, 4–1, 4–2, 5–2
Centroid,
  definition of, 3–4
CFT,
  comparing, 6–3
    DFT and, 6–3
    FFT and, 6–3
  definition of, 6–2
  input, 6–4
  output, 6–5
Change,
  definition of local, 2–1, 2–2
  local, 2–2, 2–4, 3–2
  trend, 2–3, 2–4, 3–2
Characteristics,
  Input,
    HISTI, 4–2 to 4–5
    RHISTI, 5–2 to 5–5
/CO switch, B–9
Coefficients,
  correlation, 9–1
  Fourier, 8–1
Command files,
  Indirect,
    LSPBLD, A–6, B–11
    LSPCND, A–6, B–11
    LSPMAK, A–6, B–11
    LSPVER, A–6, B–11

Commands,
RSX–11M,
ALL, B–3, B–5, B–7
ASN, B–11, B–14
BAD, B–4, B–6, B–8
BOOT, B–7
DMOU, B–5, B–9
DSC, B–5, B–7
DSCS8, B–7
EDI, B–15
FLX, B–9
FMT, B–3, B–4, B–6, B–8
FOR, B–16
F77, B–16
INI, B–4, B–7, B–9
LBR, B–2
MOU, B–3, B–5, B–7
PIP, B–9
RUN, B–11, B–16
TKB, B–16, B–17
UFD, B–9
RT–11,
ASSIGN, A–6, A–9
COPY, A–2
EDIT, A–10
FORMAT, A–3, A–4
FORTRAN, A–11
INITIALIZE, A–3, A–4
LIBRARY, A–12
LINK, A–11, A–12
RENAME, A–5
RUN, A–7, A–11
SQUEEZE, A–2, A–3, A–5
Comparing,
DFT and CFT, 6–4
FFT and CFT input, 6–4
FFT and CFT output, 6–5
Compiler,
FORTRAN IV, A–1, A–2, A–11, B–2, B–16
FORTRAN 77, B–2, B–16
Continuous Fourier transform, 6–2
Continuum,
definition of, 4–2
COPY command, A–2
Copying the distribution volume
under RSX–11M, B–2
under RT–11, A–2
Correction,
baseline, 3–22
Corrections,
making, A–5, B–5
CORREL,
arguments, 9–5 to 9–6
example program, 9–9
file, 1–5, 9–7
FORTRAN call, 9–5
input, 9–2
output, 9–2, 9–12
terminal output, 9–2, 9–12
using, 9–2
Correlation coefficients, 9–1
Correlation function,
average, 9–1 to 9–5
definition of, 9–1
description, 9–1 to 9–12
subroutine, 9–1 to 9–12
CORREL.MAC, 1–5
CORREL.OBJ, 1–5
Count,
HISTI overflow, 4–3, 4–16, 4–20
HISTI underflow, 4–3
NVELOP reject, 3–18
RHISTI overflow, 5–6
RHISTI underflow, 5–6
/CREATE switch, A–10
Creating,
customized object files, A–6, B–10
programs,
under RSX–11M, B–15
under RT–11, A–10
Crest,
peak, 2–1, 2–4, 2–6
PEAK, 2–1, 2–4, 2–6
Cross correlation, 9–1 to 9–12
Customized object files,
creating, A–6, B–10

Data,
averaging, 2–2
definition of, 2–1, 2–2
raw, 2–1, 2–2
Data and category, 4–2
Data/integer relation, 5–1, 5–2
Default,
file types,
RSX–11M, B–1, B–16
RT–11, A–1, A–10
Definition of,
baseline, 2–6
category, 4–1, 4–2, 5–2
centroid, 3–4
CFT, 6–2

Definition of, (Cont.)
continuum, 4–2
correlation function, 9–1
data, 2–1, 2–2
DFT, 6–2
event, 4–2, 5–2
FFT, 6–1
FORTRAN 77, 1–2
Fourier transforms, 6–1
interval, 5–1, 5–2
local change, 2–2, 2–4, 3–2
noise, 2–2, 3–2
overflow values, 4–2, 5–2
peak height, 2–1, 3–1, 3–2
reference points, 5–1, 5–2
time axis, 2–2, 3–2
trends, 2–3, 2–4, 3–2
underflow values, 4–2, 5–2
valley, 2–1, 2–6, 2–7, 3–2, 3–4
/DENS = HIGH switch, B–4, B–6, B–8
Description,
correlation function, 9–1
Device,
storage, A–2, B–3
DOS–11–formatted, B–3, B–5
FILES–11–formatted, B–3, B–7
DFT,
definition of, 6–2
inverse, 6–3
Digital filter,
software, 2–3
Discrete,
evaluation of CORREL, 9–3
Fourier transform, 6–2, 6–3
Distributed files,
object, 1–4, 1–5, A–5, B–10
source, 1–4, 1–5, A–6, B–10
Distribution volume,
contents of, 1–3 to 1–5
copying the,
under RSX–11M, B–2
under RT–11, A–2
DMOU command, B–5, B–9
/DO switch, B–9
DOS–11 formatted device, B–3, B–5
Double-precision integers, 2–24, 4–8, 5–17
DPH$ option, 4–8, A–8, B–13
DPP$ option, 2–24, A–8, B–12, C–2, D–2
DPR$ option, 5–17, A–8, B–13
DSC,
command, B–7
utility, B–5, B–7
DSCS8 utility, B–5
DUP utility, A–2
EAE option, 2–22, 3–22, 4–8, 5–10, 6–16, 7–3, 8–2, 9–7, A–7, B–12
EIS option, 2–22, 3–22, 4–8, 5–10, 6–16, 7–3, 8–2, 9–7, A–7, B–12, C–2, D–2
EDI command, B–15
EDIT command, A–10
Editor,
RSX–11M, B–15
RT–11, A–10
Envelope-processing,
algorithm, 3–1
subroutine, 3–1 to 3–32
terms and conventions, 3–2
Estimated peak width, 2–1, 2–5, 3–3
Evaluation of CORREL,
discrete, 9–3
Event,
definition of, 4–2, 5–2
Events,
NVELOP, 3–14
Example program file names, 1–3 to 1–5
Example programs,
CORREL, 9–9 to 9–12
FFT, 6–19 to 6–22
HISTI, 4–12 to 4–28
NVELOP, 3–22 to 3–28
PEAK, 2–24 to 2–40
PHAMPL, 7–5 to 7–12
POWRSP, 8–5 to 8–12
RHISTI, 5–13 to 5–24
EXEMC.MLB, B–1

F4FFT.MAC, 1–4
F4FFT.OBJ, 1–4
F4P$ option, B–12, D–1, D–2
F77 command, B–16
F4PRAN.OBJ, 4–11, 5–11, 7–3, 8–3, B–2
Factor,
PEAK gate, 2–3
scaling, 6–8 to 6–12
Fast Fourier transform subroutine, 6–1 to 6–22
FFT,
definition of, 6–1
arguments, 6–13
example program, 6–19 to 6–22
file, 1–4, 6–16
FORTRAN call, 6–13
input, 6–4
maximum I/O array size, 6–16
output, 6–5, 6–22
properties, 6–14
terminal output, 6–22
using, 6–1 to 6–22

File,
CORREL, 1–5, 9–7
FFT, 1–4, 6–16
HISTI, 1–4, 4–7
LSPBLD, A–6, B–11
LSPCND, A–6, B–11
LSPMAK, A–6, B–11
LSPVER, A–6, B–11
NVELOP, 1–4, 3–22
PEAK, 1–4, 2–22
PHAMPL, 1–5, 7–2
POWRSP, 1–5, 8–2
RHISTI, 1–4, 5–10
File names,
distribution kit, 1–3 to 1–5
example programs, 1–3 to 1–5
subroutine,
object, 1–3 to 1–5
source, 1–3 to 1–5
File protection, A–5
File types,
default,
RSX–11M, B–1, B–16
RT–11, A–1, A–10
Files,
indirect-command,
LSPBLD, A–6, B–11
LSPCND, A–6, B–11
LSPMAK, A–6, B–11
LSPVER, A–6, B–11
object, 1–3 to 1–5, A–5, B–10
source, 1–3 to 1–5, A–5, B–10
verifying, A–9 to A–10, B–14 to B–15
FILES–11–formatted device, B–3, B–7
Filter,
software digital, 2–3
FLOAT, 7–2
Flowchart,
NEXTPT, 2–13, 3–11
RITOUT, 2–14, 3–12
NVELOP, 3–6 to 3–10
PEAK, 2–9 to 2–12
FLX,
command, B–9
utility, B–2
F.MAXN option, 6–16, A–8, B–13, C–2, D–3
FMT utility, B–3, B–4, B–6, B–8
FNVLOP.MAC, 1–4
FNVLOP.OBJ, 1–4
FOR command, B–16
/FOR switch, B–3, B–7
FORMAT,
command, A–3, A–4
utility, A–3, A–4
FORTRAN command, A–11, B–16
FORTRAN IV,
compiler, A–1, A–2, A–11, B–2, B–16
library, 4–5, 7–2, A–12
object time system, A–2, B–2
random-number generator, 4–11, 5–11, 7–3, 8–3
FORTRAN 77,
compiler, B–2, B–9
definition of, 1–2
library, 4–5, 7–2
object time system, A–2, B–2
random-number generator, 4–11, 5–11, 7–3, 8–3, B–2
Forward transform, 6–1
Fourier coefficients, 8–1
Fourier transform subroutine,
fast, 6–1 to 6–22
/FP switch, B–16, B–17
FPEAK.MAC, 1–4
FPEAK.OBJ, 1–4
FREQ$ option, 4–8, A–8, B–13
Frequency functions, 4–9, 6–2
Frequency histogram, 4–1 to 4–4
Function,
average correlation, 9–1 to 9–5
definition of correlation, 9–1
Functions,
frequency, 4–9, 6–2

Gate factor,
NVELOP, 3–3
PEAK, 2–3
Graphs,
bar, 4–1, 5–1

Height,
definition of peak, 2–1, 3–1, 3–2
peak, 2–1, 3–1, 3–2
peak baseline, 3–1, 3–2
HISTI,
arguments, 4–5 to 4–6
example programs, 4–12 to 4–28
file, 1–4, 4–7
FORTRAN call, 4–5 to 4–7
input, 4–2
input characteristics, 4–2 to 4–5
output, 4–7
overflow count, 4–3
terminal output, 4–16, 4–20, 4–24, 4–28
terms and conventions, 4–1
underflow count, 4–3
using, 4–7
HISTI.MAC, 1–4

HISTI.OBJ, 1–4
Histogram,
  frequency, 4–1
  interval, 4–1
  zeroth, 5–6
Histogramming,
  interval, 4–1 to 4–28
Histogramming subroutine,
  interval, 4–1 to 4–28
Histograms, 4–1 to 4–28

Indicator,
  scaling, 6–9
Indirect-command files,
  LSPBLD, A–6, B–11
  LSPCND, A–6, B–11
  LSPMAK, A–6, B–11
  LSPVER, A–6, B–11
INI command, B–4, B–7, B–9
INITIALIZE command, A–3, A–4
Input,
  CORREL, 9–2
  FFT, 6–4
  HISTI, 4–2
  NVELOP, 3–17
  PEAK, 2–2
  PHAMPL, 7–1
  POWRSP, 8–1
  RHISTI, 5–9
Input characteristics,
  HISTI, 4–2 to 4–5
  RHISTI, 5–2 to 5–5
Input data averaging, 2–2
Installation,
  using RSX–11M, B–1
  using RT–11, A–1
Integer and interval, 4–2, 5–2
Integers,
  double-precision, 2–24, 4–8, 5–17
  single-precision, 2–24, 4–8, 5–17
  signed, 4–4, 5–3
  unsigned, 4–4, 5–3
Interactive build procedure, 1–3, 1–5, A–6 to A–9, B–10 to B–14, C–1, D–1
Internal scaling procedure, 6–9
Interval,
  definition of, 5–1, 5–2
  integer and, 4–2, 5–2
Interval histogramming, 4–1, 5–1
Interval histogramming subroutine, 4–1 to 4–28
Interval histogramming with reference points
  subroutine, 5–1 to 5–24
Inverse DFT, 6–3
Inverse Fourier transform, 6–2, 6–3

/LB switch, B–17
LBR utility, B–2, B–17
Laboratory Subroutines package,
  definition of, 1–1 to 1–5
Leakage, 6–8
/LI switch, B–4, B–6, B–8
LIBR utility, A–12
Libraries,
  FORTRAN IV, 4–5, 7–2, A–12
  FORTRAN 77, 4–5, 7–2
  System,
    MACRO, A–1, B–1
    object, A–1, A–11, A–12, B–1, B–17
  using, A–12, B–17
LIBRARY command, A–12
LINK command, A–11, A–12
Linker, A–2, A–11
Local change,
  definition of, 2–2
Local maximum, 2–4
Local minimum, 2–4
LSPBLD, A–6, B–11
LSPCND, A–6, B–11
LSPMAK, A–6, B–11
LSPVER, A–6, B–11

MACRO,
  assembler,
    RSX–11M, B–1
    RT–11, A–1
Making corrections, A–5, B–5
Maximum,
  local, 2–4
Minimum,
  local, 2–4
MOU command, B–3, B–5, B–7

NEXTPT,
  flowchart, 2–13, 3–11
  subroutine, 2–13, 3–11
NOFLT$ option, 2–24, A–7, B–13, C–2, D–2
Noise,
  definition of, 2–2, 3–2
/NOPROTECTION switch, A–5
Numbers,
  see Integers
NVELOP,
  arguments, 3–16 to 3–20
  baseline value, 3–3, 3–17, 3–18
  events, 3–14
  example programs, 3–25 to 3–32

NVELOP, (Cont.)
file, 1–4, 3–22
flowcharts, 3–6 to 3–10
FORTRAN call, 3–16
gate factor, 3–3
input, 3–17
options, 3–22
output, 3–19, 3–20
reject count, 3–18
symbol definition, 3–5
terms and conventions, 3–2
terminal output, 3–28, 3–30, 3–32
threshold value, 3–22, 3–23, 3–29, 3–31
using, 3–20 to 3–22
zero values, 3–18
NVELOP.MAC, 1–4
NVELOP.OBJ, 1–4

Object files,
list of, 1–3 to 1–5
creating customized, A–6, B–10
distributed, 1–4, 1–5, A–5, B–10
Operating system,
RSX–11M, 1–1 to 1–5, B–1, D–1
RT–11, 1–1 to 1–5, A–1, C–1
Options,
AUTOG$, A–22, A–8, B–3
DPH$, 4–8, A–8, B–13
DPP$, 2–24, A–8, B–13, C–2, D–2
DPR$, 5–17, A–8, B–13
EAE, 2–22, 3–22, 4–8, 5–10, 6–16, 7–3, 8–2, 9–7, A–7, B–12
EIS, 2–22, 3–22, 4–8, 5–10, 6–16, 7–3, 8–2, 9–7, A–7, B–12, C–2, D–2
F.MAXN, 6–16, A–8, B–13, C–2, D–3
F4P$, 7–3, B–12, D–1, D–2
FREQ$, 4–8, A–8, B–13
NOFLT$, 2–24, A–7, B–13, C–2, D–2
using, 1–1 to 1–3, 2–22, 3–22, 4–7, 5–10, 6–16, 7–2, 8–2, 9–7, A–5 to A–9, B–10 to B–14, C–1, D–1
Output,
CORREL, 9–2, 9–12
FFT, 6–5, 6–22
HISTI, 4–16, 4–20, 4–24, 4–28
NVELOP, 3–28, 3–30, 3–32
PEAK, 2–28, 2–32, 2–36, 2–40
PHAMPL, 7–12
POWRSP, 8–8, 8–12
RHISTI, 5–16, 5–20, 5–24
/OUTPUT switch, A–3, A–5
Overflow count,
HISTI, 4–3, 4–16, 4–20
RHISTI, 5–6

PEAK,
arguments, 2–17 to 2–20
autogain, 2–22, A–8, B–3
baseline, 2–6
crest, 2–1, 2–4, 2–6
digital filter, 2–3
example programs, 2–24 to 2–40
flowcharts, 2–9 to 2–12
file, 1–4, 2–22
FORTRAN call, 2–17
gate factor, 2–3
input, 2–2
local,
change, 2–2
maximum, 2–4
minimum, 2–4
output, 2–21
switch settings, 2–7
symbols, 2–8, 2–9
terminal output, 2–28, 2–32, 2–36, 2–40
terms and conventions, 2–1, 2–2
using, 2–20
peak,
area, 2–1, 2–4, 3–4
area calculation, 3–4
baseline, 2–6
centroid, 3–4
centroid calculation, 3–4
crest, 2–1, 2–4, 2–6
height,
definition of, 2–1, 3–1, 3–2
width, 2–1, 2–5, 3–3
Peak-processing,
algorithm, 2–1
subroutine, 1–2, 2–1 to 2–40
PHAMPL,
algorithm, 7–1
arguments, 7–2
example programs, 7–5 to 7–12
file, 1–5, 7–2
FORTRAN call, 7–2
input, 7–1
output, 7–1
terminal output, 7–12
using, 7–1
PHAMPL.MAC, 1–5
PHAMPL.OBJ, 1–5
Phase angle and amplitude spectra
subroutine, 7–1 to 7–12
Phase angles, 7–1
PIP,
command, B–9
utility, A–2, B–9

Points,
reference, 5–1, 5–2
Power spectrum subroutine, 8–1 to 8–12
POWRSP,
algorithm, 8–1
arguments, 8–2
example programs, 8–5 to 8–12
file, 1–5, 8–2
FORTRAN call, 8–1
input, 8–1
output, 8–1
terminal output, 8–8, 8–12
using, 8–1
POWRSP.MAC, 1–5
POWRSP.OBJ, 1–5
Procedure,
interactive build, 1–3, 1–5, A–6 to A–9, B–10 to B–14, C–1, D–1
internal scaling, 6–9
relational scaling, 6–9
Properties,
FFT, 6–14
Protection of files, A–5
/PROTECTION switch, A–5

Random-number generator, 4–11, 5–11, 7–3, 8–3, B–2
Raw data, 2–1, 2–2
Reference points, 5–1, 5–2
Reject count,
NVELOP, 3–18
Relation,
data/integer, 5–1, 5–2
integer/interval, 4–2, 5–2
of FFT to CORREL, 9–2
Relational scaling procedure, 6–9
/REMOVE switch, A–12
RENAME command, A–5
RHISTI,
arguments, 5–7 to 5–8
characteristics, 5–2 to 5–5
example programs, 5–13 to 5–24
file, 1–4, 5–10
FORTRAN call, 5–7
input, 5–9
output, 5–11
overflow count, 5–6
terminal output, 5–16, 5–20, 5–24
terms and conventions, 5–1 to 5–2
underflow count, 5–6
using, 5–5, 5–9
RHISTI.MAC, 1–5
RHISTI.OBJ, 1–5
RITOUT,
flowchart, 2–14, 3–12
subroutine, 2–14, 3–12
/RP switch, B–2
/RS switch, B–9
RSX–11M,
BAD utility, B–4, B–6, B–8
creating programs under, B–15
default file types, B–1, B–16
DSC utility, B–5, B–7
DSCS8 utility, B–5
editor, B–15
FLX utility, B–2
FMT utility, B–3, B–4, B–6, B–8
FORTRAN IV compiler, B–2, B–16
FORTRAN 77 compiler, B–2, B–16
LBR utility, B–2, B–17
MACRO assembler, B–1
operating system, 1–1 to 1–5, B–1, D–1
PIP utility, B–9
task-builder, B–1, B–2, B–16, B–17
RSXMAC.SML, B–1
RT–11,
creating programs under, A–10
default file types, A–1, A–10
DUP utility, A–2
editor, A–10
FORMAT utility, A–3, A–4
FORTRAN IV compiler, A–1, A–2, A–11
LIBR utility, A–12
linker, A–2, A–11
MACRO assembler, A–1
operating system, 1–1 to 1–5, A–1, C–1
PIP utility, A–2
RUN command, A–7, A–11, B–11, B–16, C–2, D–2

Scaling,
factor, 6–8 to 6–12
FFT results, 6–8 to 6–12
indicator, 6–9
Selecting the form of subroutine to use, A–5, B–10
Signed integers, 4–4, 5–3
/SINGLEDENSITY switch, A–3, A–4
Single-precision integers, 4–8, 5–17
Software digital filter, 2–3
Source files, 1–3 to 1–5, A–5, B–10
Spectra,
amplitude, 7–1
Spectra subroutine,
Phase angle and amplitude, 7–1 to 7–12

Spectrum subroutine,
power, 8–1 to 8–12
SQRT, 7–2
SQUEEZE command, A–2, A–3, A–5
Storage devices, A–2, B–3, B–5, B–7
Subroutine,
correlation function, 1–2, 9–1 to 9–12
envelope-processing, 1–2, 3–1 to 3–32
fast Fourier transform, 1–2, 6–1 to 6–22
interval histogramming, 1–2, 4–1 to 4–28
interval histogramming with reference points, 1–2, 5–1 to 5–24
NEXTPT, 2–13, 3–11
peak processing, 1–2, 2–1 to 2–40
phase angle and amplitude spectra, 1–2, 7–1 to 7–12
power spectrum, 1–2, 8–1 to 8–12
RITOUT, 2–14, 3–12
selecting the form of, A–5, B–10
Subroutine example program names, 1–4 to 1–5
Subroutine file names, 1–4 to 1–5
Switches,
/BAD=[AUTO], B–4, B–7, B–9
/BADBLOCKS, A–3, A–4
/CO, B–9
/CREATE, A–10
/DENS=HIGH, B–4, B–6, B–8
/DO, B–9
/FOR, B–3, B–7
/FP, B–16, B–17
/LB, B–17
/LI, B–4, B–6, B–8
/NOPROTECTION, A–5
/OUTPUT, A–3, A–5
/PROTECTION, A–5
/REMOVE, A–12
/RP, B–2
/RS, B–9
/SINGLEDENSITY, A–3, A–4
/WAIT, A–4, A–5
SYSLIB.OBJ, A–1, A–11
SYSLIB.OLB, B–2
SYSMAC.SML, A–1
System libraries,
MACRO, A–1, B–1
object, A–1, A–11, A–12, B–1, B–17
using, A–12, B–17

Task, B–16
Task-builder,
RSX–11M, B–1, B–2, B–16, B–17
Task-image file, B–1, B–2
Terminal output,
CORREL, 9–12
FFT, 6–22
HISTI, 4–16, 4–20, 4–24, 4–28
NVELOP, 3–28, 3–30, 3–32
PEAK, 2–28, 2–32, 2–36, 2–40
PHAMPL, 7–12
POWRSP, 8–8, 8–12
RHISTI, 5–16, 5–20, 5–24
Testing software
see verifying
Threshold value, 3–22, 3–23, 3–29, 3–31
Time axis,
definition of, 2–2, 3–2
TKB command, B–16, B–17
Transform,
approximate Fourier, 6–2, 6–13
forward, 6–1, 6–2
inverse, 6–2
Transform subroutine,
fast Fourier, 6–1 to 6–22
Transforms,
continuous Fourier, 6–2
discrete Fourier, 6–2, 6–3
inverse Fourier, 6–2
Trend,
change, 2–3, 2–4, 3–2
definition of, 2–3, 2–4, 3–2
True peak width, 2–5

UFD command, B–9
Underflow count,
HISTI, 4–3
RHISTI, 5–6
Underflow values, 4–2, 5–2
Unsigned integers, 4–4, 5–3
Using,
CORREL, 9–2
customized object files, A–6, B–10
distributed files, A–5, B–10
FFT, 6–1 to 6–22
HISTI, 4–7
Libraries, A–12, B–17
NVELOP, 3–20
options, 1–1 to 1–3, 2–22, 3–22, 4–7, 5–10, 6–16, 7–2, 8–2, 9–7, A–5 to A–9, B–10 to B–14, C–1, D–1
PEAK, 2–20
PHAMPL, 7–1
POWRSP, 8–1
RHISTI, 5–5, 5–9

Utilities,
  RSX–11M,
    BAD, B–4, B–6, B–8
    DSC, B–5, B–7
    DSCS8, B–5
    FLX, B–2
    FMT, B–3, B–4, B–6, B–8
    LBR, B–2, B–17
    PIP, B–9
  RT–11,
    DUP, A–2
    FORMAT, A–3, A–4
    LIBR, A–12
    PIP, A–2

Valley,
  definition of, 2–1, 2–6, 2–7, 3–2, 3–4
Value,
  NVELOP baseline, 3–3, 3–17, 3–18
  threshold, 3–22, 3–26, 3–27
Values,
  NVELOP zero, 3–18
Verifying,
  Laboratory Subroutines software, A–9 to A–10, B–14 to B–15

Width,
  peak, 2–1, 2–5, 3–3
/WAIT switch, A–4, A–5

Zero values,
  NVELOP, 3–18
Zeroth histogram, 5–6

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____ Telephone _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MR1-2/E37
MARLBOROUGH, MASSACHUSETTS 01752

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line

Printed in U.S.A.