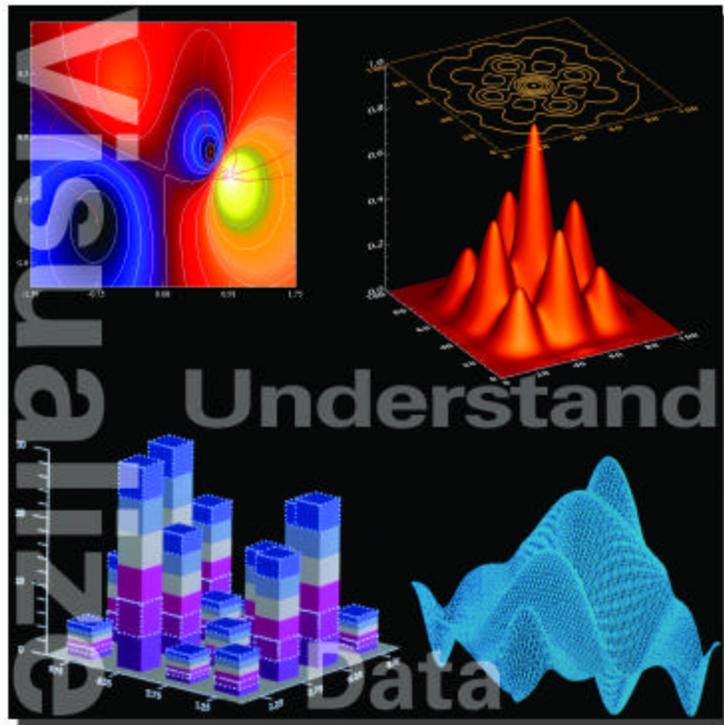




# P V - W A V E 7 . 5<sup>®</sup>



T u t o r i a l f o r W i n d o w s

HELPING CUSTOMERS **SOLVE** COMPLEX PROBLEMS

## Visual Numerics, Inc.

---

Visual Numerics, Inc.  
2500 Wilcrest Drive  
Suite 200  
Houston, Texas 77042-2579  
United States of America  
713-784-3131  
800-222-4675  
(FAX) 713-781-9260  
<http://www.vni.com>  
e-mail: [info@boulder.vni.com](mailto:info@boulder.vni.com)

Visual Numerics, Inc.  
7/F, #510, Sect. 5  
Chung Hsiao E. Rd.  
Taipei, Taiwan 110 ROC  
+886-2-727-2255  
(FAX) +886-2-727-6798  
e-mail: [info@vni.com.tw](mailto:info@vni.com.tw)

Visual Numerics S.A. de C.V.  
Cerrada de Berna 3, Tercer Piso  
Col. Juarez  
Mexico, D.F. C.P. 06600  
Mexico

Visual Numerics, Inc. (France) S.A.R.L.  
Tour Europe  
33 place des Corolles  
Cedex 07  
92049 PARIS LA DEFENSE  
FRANCE  
+33-1-46-93-94-20  
(FAX) +33-1-46-93-94-39  
e-mail: [info@vni-paris.fr](mailto:info@vni-paris.fr)

Visual Numerics International GmbH  
Zettachring 10  
D-70567 Stuttgart  
GERMANY  
+49-711-13287-0  
(FAX) +49-711-13287-99  
e-mail: [info@visual-numeric.de](mailto:info@visual-numeric.de)

Visual Numerics, Inc., Korea  
Rm. 801, Hanshin Bldg.  
136-1, Mapo-dong, Mapo-gu  
Seoul 121-050  
Korea

Visual Numerics International, Ltd.  
Suite 1  
Centennial Court  
East Hampstead Road  
Bracknell, Berkshire  
RG 12 1 YQ  
UNITED KINGDOM  
+01-344-458-700  
(FAX) +01-344-458-748  
e-mail: [info@vniuk.co.uk](mailto:info@vniuk.co.uk)

Visual Numerics Japan, Inc.  
Gobancho Hikari Building, 4th Floor  
14 Gobancho  
Chiyoda-Ku, Tokyo, 102  
JAPAN  
+81-3-5211-7760  
(FAX) +81-3-5211-7769  
e-mail: [vda-spirt@vnij.co.jp](mailto:vda-spirt@vnij.co.jp)

---

© 1990-2001 by Visual Numerics, Inc. An unpublished work. All rights reserved. Printed in the USA.

Information contained in this documentation is subject to change without notice.

IMSL, PV-WAVE, Visual Numerics and PV-WAVE Advantage are either trademarks or registered trademarks of Visual Numerics, Inc. in the United States and other countries.

The following are trademarks or registered trademarks of their respective owners: Microsoft, Windows, Windows 95, Windows NT, Fortran PowerStation, Excel, Microsoft Access, FoxPro, Visual C, Visual C++ — Microsoft Corporation; Motif — The Open Systems Foundation, Inc.; PostScript — Adobe Systems, Inc.; UNIX — X/Open Company, Limited; X Window System, X11 — Massachusetts Institute of Technology; RISC System/6000 and IBM — International Business Machines Corporation; Java, Sun — Sun Microsystems, Inc.; HPGL and PCL — Hewlett Packard Corporation; DEC, VAX, VMS, OpenVMS — Compaq Computer Corporation; Tektronix 4510 Rasterizer — Tektronix, Inc.; IRIX, TIFF — Silicon Graphics, Inc.; ORACLE — Oracle Corporation; SPARCstation — SPARC International, licensed exclusively to Sun Microsystems, Inc.; SYBASE — Sybase, Inc.; HyperHelp — Bristol Technology, Inc.; dBase — Borland International, Inc.; MIFF — E.I. du Pont de Nemours and Company; JPEG — Independent JPEG Group; PNG — Aladdin Enterprises; XWD — X Consortium. Other product names and companies mentioned herein may be the trademarks of their respective owners.

**IMPORTANT NOTICE: Use of this document is subject to the terms and conditions of a Visual Numerics Software License Agreement, including, without limitation, the Limited Warranty and Limitation of Liability.** If you do not accept the terms of the license agreement, you may not use this documentation and should promptly return the product for a full refund. Do not make illegal copies of this documentation. No part of this documentation may be stored in a retrieval system, reproduced or transmitted in any form or by any means without the express written consent of Visual Numerics, unless expressly permitted by applicable law.

---

---

# ***Table of Contents***

## **Preface ix**

### **Technical Support x**

FAX and E-mail Inquiries xi

Electronic Services xii

## ***Chapter 1: Introduction 1***

### **The PV-WAVE Software Family for Windows 2**

Introducing PV-WAVE Extreme Advantage™ and  
PV-WAVE Advantage™ 2

PV-WAVE:Signal Processing Toolkit 5

PV-WAVE:Image Processing Toolkit 5

### **What is Visual Data Analysis? 6**

### **Using This Tutorial 6**

### **Windows NT vs. Windows 95 and Windows 98 6**

### **Conventions Used in This Tutorial 7**

Terms 7

Type Styles 7

### **Installation 8**

### **Where to Obtain Additional Help 8**

## ***Chapter 2: About PV-WAVE 9***

### **Some Basic Rules and Concepts 10**

PV-WAVE Notation 10

Extensive Error Checking 10

General Information 11

Basic Data Types 11

Scalars, Arrays, and Structures 12

Comparison of Matrices and Arrays 12

## **Chapter 3: Experimenting with the Navigator 15**

**What is the Navigator? 15**

**Starting PV-WAVE 16**

Starting PV-WAVE Under Windows NT 16

Starting PV-WAVE Under Windows 95 16

**Starting the Navigator 17**

**The Online Help System 18**

**Manuals Online 18**

**Tutorial Exercises Using the Navigator 19**

### ***Displaying Time-Series Data 20***

**Importing ASCII Data 20**

**Creating XY Plots 23**

**Using PV-WAVE Commands 25**

**Reformatting Datasets 26**

**Displaying Data as an Image 27**

Resizing the Image 27

Displaying Profile Plots 29

Manipulating Data Across VDA Tools 30

Using Color 32

**Creating Three-Dimensional Surface Plots 33**

**Using Online Help 34**

Context-Sensitive Help 34

PV-WAVE Online Help 34

**Printing Your Results 35**

**Experimenting on Your Own 36**

**Ending the Session 36**

### ***Displaying Image Data 37***

**Importing 8-bit Image Data 37**

**Checking the Selected Variables List 38**

Displaying the Image	39
Displaying the Image as a 3D Surface Plot	40
Using Color	41
Rescaling Image Data	41
Resizing the Image	43
Smoothing the Image	44
Cropping the Image	45
Creating an Inverted Image	47
Editing Color	48
Experimenting on Your Own	50
Saving the Session	50
Ending the Session	50
<b><i>Animating Image Data</i></b>	<b>51</b>
Importing the Animation Data	51
Viewing Displacement as an Image	52
Shading and Rotating a Surface View	54
Building an Animation Sequence	56
Defining an Animation Array	56
Viewing Animation	58
Changing the Appearance of a Plot	59
Saving and Using a Template	61
Saving a Plot	62
Experimenting on Your Own	62
Exiting PV-WAVE	62

## **Chapter 4: Getting Started 63**

**Before You Begin ... 64**

**Starting and Exiting PV-WAVE 64**

Starting PV-WAVE Under Windows NT 64

Starting PV-WAVE Under Windows 95 64

Exiting PV-WAVE 64

**Brief Online Help Review 65**

**Previewing This Chapter 65**

**Creating and Labeling a Plot 66**

Creating the Data 66

Drawing the Plot 67

Using Keywords 68

Obtaining Information About Your Data 69

Adding Symbols 70

Adding Another Line to the Plot 71

Working with Colors 72

Recalling Previous Commands 74

Adding Labels to the Plot 75

Adding Today's Date to the Plot 77

Using System Variables to Retain Changes 78

Placing Labels Within the Plot 80

Resetting System Variables 81

**Creating Surfaces and Contours 82**

**Close the Windows 89**

## **Chapter 5: Basic Information for Any Session 91**

**Getting Help with PV-WAVE 92**

Using Online Documentation 92

Using the INFO Command 92

Looking at PV-WAVE Source Files 94

**Exiting, Interrupting, and Aborting the Software 94**

Exiting PV-WAVE 94

Interrupting or Aborting PV-WAVE 95

**Restoring "Lost" Variables 95**

**Saving Session Commands and Variables 96**

Using the SAVE Procedure 96

Using the RESTORE Procedure 97

**Using Data from Files 98**

**What Does “Program Area Full” Mean? 98**

***Chapter 6: Plotting Your Data 99***

**Previewing This Chapter 100**

**Reading Data from a File 100**

File Operations and Logical Unit Numbers 101

Accessing Data 102

**Using 2D Plotting Capabilities 103**

Processing and Plotting Signal Data 103

Using the SMOOTH Command 105

Frequency Domain Filtering 105

Displaying the Results 107

Velocity Field Plotting 110

More Information on 2D Plotting 111

**Contour and Surface Plotting 111**

Accessing a Data Set 111

Plotting with CONTOUR 113

Plotting with SURFACE 123

Displaying Data as a Shaded Surface 126

Displaying a TIFF File or a Logo 129

Showing Three Levels 130

Using Color to Represent Data 130

More Information on 3D Plotting 134

**Deleting Windows 134**

***Chapter 7: Array Processing Techniques 135***

**Previewing This Chapter 136**

**Displaying and Processing Arrays 136**

Reading an Array 136

Displaying an Array 137

Changing the Size of an Image	138
Contrast Enhancement	140
Smoothing and Sharpening	142
Other Image Manipulations	144
Extracting Profiles	147
More Information on Array Processing	147

## **Chapter 8: Using Color 149**

### **Using Color Tables 150**

About Color in Previous Chapters	150
Creating Your Own Colors	150

### **Using Color to Enhance Analysis 153**

Displaying an Image	153
Changing Colors in the Image	154

## **Chapter 9: Advanced Math and Stats 155**

### **Programs You Can Use 156**

### **Solving a Nonlinear System of Equations 157**

Running the Example Program	158
Example Program Listing: nonlin_system	158

### **Using a 2D B-Spline Interpolation Procedure 160**

Running the Example Program	161
Example Program Listing: heat	162

### **Creating and Plotting a Parametric Cubic Spline Interpolant 164**

Running the Example	165
Example Program Listing: para_spline	167

### **Estimating Area with a Random Number Generator 168**

Running the Example Program	168
Example Program Listing: monte	169

### **Plotting a Fourier Sine Series 171**

Running the Example	171
Example Program Listing: fourier	172

### **Plotting a Shaded Surface of a Scattered 2D Interpolant 174**

Running the Example	174
Example Program Listing: scattered	175
<b>Using the Lorenz Attractor</b>	<b>177</b>
Running the Example	178
Example Program Listing: lorenz	178
<b>Using Multiple Regression Capabilities</b>	<b>180</b>
Running the Example	180
Example Program Listing: mreg	181
<b>Chapter 10: Writing Programs</b>	<b>185</b>
<b>Creating an Executable Program File</b>	<b>186</b>
Example Program: quad	186
Example Program: Solve an Ordinary Differential Equation	188
<b>Creating and Using Batch Files</b>	<b>193</b>
About Batch Files	193
Using JOURNAL to Save Commands	193
Example Program: gam_bess.pro	194
Open a JOURNAL File	194
Enter Commands at the Command Line	195
<b>A Comparison of Matrices and Arrays</b>	<b>198</b>
Example Program: matrices.pro	199
Example Program: linear.pro	201
Running the Example Program	202
Example Program Listing: linear	202
<b>Chapter 11: Animating Data</b>	<b>205</b>
<b>Previewing This Chapter</b>	<b>206</b>
<b>Displaying a Series of Images</b>	<b>206</b>
<b>Animation As a Wire Frame Surface</b>	<b>208</b>
<b>Rotating the Images in an Animation</b>	<b>210</b>
<b>More Information on Animation</b>	<b>210</b>

**Index** 1



# Preface

Welcome to the tutorial for PV-WAVE on Windows!

The first two chapters provide information about this tutorial and PV-WAVE. Chapters 3 through 11 provide sequenced, hands-on, step-by-step examples that demonstrate a wide range of applications and enable you to learn the fundamentals of using PV-WAVE products.

The chapters in this tutorial are organized as follows:

**Chapter 1, *Introduction*** — Describes the PV-WAVE Family of Products and explains how to use this tutorial.

**Chapter 2, *About PV-WAVE*** — Discusses the features of PV-WAVE and some basic concepts that are helpful to know before you begin.

**Chapter 3, *Experimenting with the Navigator*** — Prepares you to use the Navigator and VDA Tools of PV-WAVE:Visual Exploration. This technology provides the easiest way for you to interact with PV-WAVE. It features an intuitive graphical interface and context sensitive online help.

**Chapter 4, *Getting Started*** — Introduces you to the PV-WAVE command line interface, beginning with a simple plot and continuing with a more detailed exercise in the use of some basic functions and keywords.

**Chapter 5, *Basic Information for Any Session*** — Discusses some of the basics of working with PV-WAVE, such as saving and restoring sessions, using online documentation, and getting help.

**Chapter 6, *Plotting Your Data*** — Provides you with experience using plotting and graphics keywords to display several types of 2D and 3D plots, including contour and surface plots.

[Chapter 7, \*Array Processing Techniques\*](#) — Shows you how to read in an array of bytes, how to display this data as an image, and how to use PV-WAVE array processing commands.

[Chapter 8, \*Using Color\*](#) — Shows you how you can use color to enhance image analysis and to produce customized displays of color in your plots.

[Chapter 9, \*Advanced Math and Stats\*](#) — Demonstrates the power of PV-WAVE:IMSL Mathematics and PV-WAVE:IMSL Statistics. Incorporates code examples of mathematical and statistical programs you can run and use to pattern your own programs.

[Chapter 10, \*Writing Programs\*](#) — Focuses on methods that will help you write your own PV-WAVE programs.

[Chapter 11, \*Animating Data\*](#) — Shows you several methods for animating data.

[Index](#)

---

## ***Technical Support***

If you have problems installing, unlocking, or running your software, contact Visual Numerics Technical Support by calling:

<b>Office Location</b>	<b>Phone Number</b>
Corporate Headquarters Houston, Texas	713-784-3131
Boulder, Colorado	303-939-8920
France	+33-1-46-93-94-20
Germany	+49-711-13287-0
Japan	+81-3-5211-7760
Korea	+82-2-3273-2633
Mexico	+52-5-514-9730
Taiwan	+886-2-727-2255
United Kingdom	+44-1-344-458-700

Users outside the U.S., France, Germany, Japan, Korea, Mexico, Taiwan, and the U.K. can contact their local agents.

Please be prepared to provide the following information when you call for consultation during Visual Numerics business hours:

- Your license number, a six-digit number that can be found on the packing slip accompanying this order. (If you are evaluating the software, just mention that you are from an evaluation site.)
- The name and version number of the product. For example, PV-WAVE 7.0.
- The type of system on which the software is being run. For example, SPARCstation, IBM RS/6000, HP 9000 Series 700.
- The operating system and version number. For example, HP-UX 10.2 or IRIX 6.5.
- A detailed description of the problem.

## **FAX and E-mail Inquiries**

Contact Visual Numerics Technical Support staff by sending a FAX to:

<b>Office Location</b>	<b>FAX Number</b>
Corporate Headquarters	713-781-9260
Boulder, Colorado	303-245-5301
France	+33-1-46-93-94-39
Germany	+49-711-13287-99
Japan	+81-3-5211-7769
Korea	+82-2-3273-2634
Mexico	+52-5-514-4873
Taiwan	+886-2-727-6798
United Kingdom	+44-1-344-458-748

or by sending E-mail to:

<b>Office Location</b>	<b>E-mail Address</b>
Boulder, Colorado	support@boulder.vni.com
France	support@vni-paris.fr

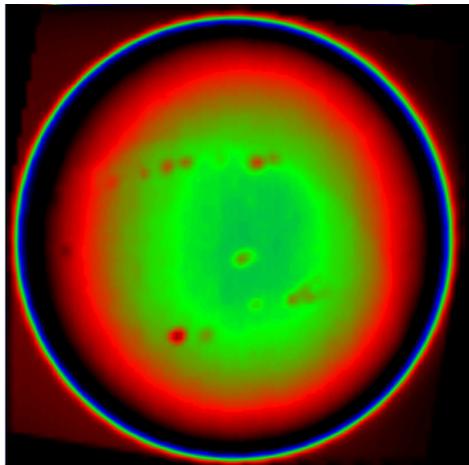
Germany	<code>support@visual-numerics.de</code>
Japan	<code>vda-sprt@vnij.co.jp</code>
Korea	<code>support@vni.co.kr</code>
Taiwan	<code>support@vni.com.tw</code>
United Kingdom	<code>support@vniuk.co.uk</code>

## **Electronic Services**

General e-mail	<code>info@boulder.vni.com</code>
Support e-mail	<code>support@boulder.vni.com</code>
World Wide Web	<code>http://www.vni.com</code>
Anonymous FTP	<code>ftp.boulder.vni.com</code>
FTP Using URL	<code>ftp://ftp.boulder.vni.com/VNI/</code>
<b>PV-WAVE</b> Mailing List:	<code>Majordomo@boulder.vni.com</code>
To subscribe include:	<code>subscribe pv-wave YourEmailAddress</code>
To post messages	<code>pv-wave@boulder.vni.com</code>

## Introduction

Welcome to the *PV-WAVE<sup>®</sup> Tutorial*. This tutorial helps you begin using PV-WAVE Foundation and the companion technologies — PV-WAVE: Visual Exploration, PV-WAVE:IMSL<sup>®</sup> Mathematics, PV-WAVE:IMSL<sup>®</sup> Statistics. The logical approach used in the tutorial gets you started in a focused and productive way, so that you can have immediate results.



**Figure 1-1** Image of sun's intensity taken from a diode array.

*In 325 B.C, Aristotle concluded in his treatise, *On the Soul*,  
“... thought is impossible without an image.”*

---

## **The PV-WAVE Software Family for Windows**

*“If only I’d had PV-WAVE ...I wish I’d had something like PV-WAVE back when I did this sort of work for a living. I would have gotten so much more done. It’s enough to make me maudlin about the weeks and months spent writing FORTRAN and C routines to do what now can be done with PV-WAVE in almost no time at all.”*

— Barry Sheen, *RS/Magazine*

### **Introducing PV-WAVE Extreme Advantage™ and PV-WAVE Advantage™**

Both PV-WAVE Extreme Advantage and PV-WAVE Advantage provide application developers with powerful functionality that supports a rapid application development environment (RADE) and provide the fundamental components of application development.

These components include an interactive language, reusable objects, powerful portability and robust graphic and numeric routines.

With PV-WAVE Extreme Advantage or PV-WAVE Advantage, you can make the best decisions, build the most robust applications, and solve the most complex problems. Leading the industry in ease-of-use and the ability to customize applications, these PV-WAVE packages enable you to identify hidden data trends, improve the quality of your analysis, reduce application development time, and become more productive.

#### ***PV-WAVE Extreme Advantage™***

PV-WAVE Extreme Advantage includes:

- PV-WAVE Foundation
- PV-WAVE Visual Exploration
- PV-WAVE:IMSL Mathematics
- PV-WAVE:IMSL Statistics
- PV-WAVE:Signal Processing Toolkit
- PV-WAVE:Image Processing Toolkit

## ***PV-WAVE Advantage™***

PV-WAVE Advantage includes:

- PV-WAVE Foundation
- PV-WAVE Visual Exploration
- PV-WAVE:IMSL Mathematics
- PV-WAVE:IMSL Statistics

## ***PV-WAVE Foundation***

The cornerstone of PV-WAVE Foundation is its interactive programming language. PV-WAVE's array-oriented and highly developed 4GL reduces coding requirements up to 80% and eliminates the need for compiling and linking. It supports variables and collections of variables, and all the same language constructs of FORTRAN and C. Its powerful functionality includes global variables, an event-driven interpreter, resource file support, and dialog box alerts to make your programming more efficient.

PV-WAVE Foundation features:

- Industry standard I/O
- GUI-based debugger
- Flexible data management functions
- Backwards compatibility
- Cross-platform compatibility

## ***PV-WAVE:Visual Exploration***

PV-WAVE Visual Exploration is an interactive environment for efficient visual data analysis and rapid application development. PV-WAVE Visual Exploration consists of multiple high-level application components, called VDA Tools, and a pre-developed GUI called the Navigator.

VDA Tools are reusable objects that provide the developer with a powerful modular framework to easily customize applications. VDA Tools allow you to perform specific functions such as generating plots, images, and contours, importing and exporting data, creating tables, generating code, and many other common procedures in an easy-to-use interactive environment. VDA Tools can also be pieced together to quickly create sophisticated, custom interfaces.

The Navigator is an intuitive interface that provides quick, individual access to the VDA Tools. Built entirely from VDA Tools, the Navigator can also be customized and extended for specific application requirements.

### ***PV-WAVE:IMSL Mathematics***

PV-WAVE:IMSL Mathematics provides mathematical routines that save implementation time and provide accurate results. The math libraries contain a full range of capabilities, with subroutines for engineering and scientific disciplines, and other fields requiring accurate, reliable mathematical computation.

The capabilities of PV-WAVE:IMSL Mathematics include:

- linear systems
- differential equations
- optimization
- basic matrix/vector operations
- interpolation and approximation
- eigenvalue systems analysis
- quadrature
- Bessel functions
- random number generation

### ***PV-WAVE:IMSL Statistics***

PV-WAVE:IMSL Statistics routines reduce development time, save development expense and allow you to build applications that are portable across multiple platforms. The statistics routines provide versatility in handling missing values, evaluating various data types and quantities, and generating printed results.

Some of the calculations performed using PV-WAVE:IMSL Statistics routines include:

- basic statistics
- factor and cluster analysis
- regression and random number generation
- tests of goodness-of-fit
- correlation

- cluster analysis
- nonparametric statistics
- time series analysis and forecasting
- variance analysis

Together, PV-WAVE:IMSL Mathematics and PV-WAVE:IMSL Statistics provide the utmost in numerical functionality and power because they

- are built upon the most widely used routines in the IMSL Fortran Numerical Libraries.
- include extensive online documentation with powerful search capabilities.
- include prepackaged functions that reduce application development.
- use a variable argument list structure that simplifies calling sequences.
- contain building blocks that eliminate the need to write code from scratch.

## **PV-WAVE:Signal Processing Toolkit**

Signal processing is widely used in engineering and scientific research and development for representing, transforming, and manipulating signals and the information they contain. The PV-WAVE:Signal Processing Toolkit is a collection of digital processing functions that work in conjunction with PV-WAVE Advantage. These functions are designed for easy use by the beginning signal processor, while providing the advanced signal processor with many options for solving difficult problems.

## **PV-WAVE:Image Processing Toolkit**

Included in the PV-WAVE:Image Processing Toolkit is an extensive set of filters, transforms, and image processing operators designed to meet the needs of even the most demanding image processing application. A robust graphical interface makes the PV-WAVE:Image Processing functionality easy to use giving the user easy access to:

- Image file import and export
- Image processing
- Histograms
- Profiles

- Contour Plots
- Surface Plots

---

## ***What is Visual Data Analysis?***

Visual Data Analysis (VDA) improves traditional data analysis by giving you a more active role in the analysis process. By emphasizing user interaction and visual representations of data, VDA enables you to control your data analysis in powerful new ways. The hallmarks of VDA are

- the ability to handle large, multidimensional data sets
- tools for fast data manipulation and subsetting
- quick graphical displays of intermediate results
- immediate user interaction
- advanced graphics tools for animating and displaying multidimensional data

By involving you visually in the analysis process — by allowing you see your data — VDA enables you to process large amounts of information quickly, giving you the opportunity to direct the discovery process from one moment to the next. With VDA you not only get more results from technical data, you get them faster.

---

## ***Using This Tutorial***

The *PV-WAVE Tutorial* assists you in discovering and exploring some of the powerful features of these products. This tutorial assumes that you have a working knowledge of your computer, its operating system, and a text editor.

---

## ***Windows NT vs. Windows 95 and Windows 98***

In general, PV-WAVE functions the same way under Windows NT, Windows 95, and Windows 98.

We will note anytime there is a difference between the way the software works on Windows NT vs. Windows 95 and Windows 98.

---

## Conventions Used in This Tutorial

This tutorial uses specific terms and typographical conventions; these conventions are explained in this section.

### Terms

**Enter** and **Type** mean type the indicated text at the prompt or into a text input field and then press the <Return> key.

The notation **MenuName=>Command** means select the function **Command** from the **MenuName** menu. For example, “select **File=>Print**” means select the **Print** command from the **File** menu.

**Choose**, **Select**, and **Click** mean move the pointer to the indicated position and press the left mouse button (MB1).

**Drag** means move the pointer to the indicated item, press the left mouse button (MB1) and hold it down while dragging the pointer to a specific location.

When you must press two keys simultaneously, the two key names are separated by a hyphen. For example: <Control>-D indicates that you should press the <Control> key and the <D> key at the same time.

**Pathname** refers to all directories and subdirectories in a path. **Filename** refers to the complete name of the file, including its extension.

### Type Styles

To assist you in recognizing functions, keywords, system variables, etc., the following conventions are used:

Typeface	Used to Indicate
All upper case (i.e., all capital letters)	Command and statement names, such as PLOT, FLTARR, FOR, WHILE, DO, CASE
Initial letter capitalized, default font	System variables, such as !Path, !P.Ticklen, !D.N_Colors
Initial letter capitalized and italics	Keywords, such as <i>X_Type</i> , <i>Max_Levels</i> , <i>Linestyle</i>
Courier (mono-spaced typewriter-style font)	Examples of PV-WAVE code
Lower case (no capitals) and italics	User-defined variables, e.g., <i>myvariable</i> , <i>myarray</i> , <i>x</i> , <i>y</i> , <i>z</i>

Unless otherwise noted, you can use upper or lower case as you prefer, because PV-WAVE is not case sensitive.

Text printed in italics in a command indicates that you must substitute the name yourself. For example,

```
WAVE> OPENR, 1, 'PathName/FileName'
```

means that you are to supply the full path and filename, such as

```
d:\users\home%VNI_DIR%\wave\data\mandril.img
```

---

## ***Installation***

When you are ready to install PV-WAVE products, refer to the installation instructions provided with the software. Once you have installed your PV-WAVE software, you may need to validate the installation by contacting Visual Numerics, Inc. Contact information is shown at the end of this chapter.

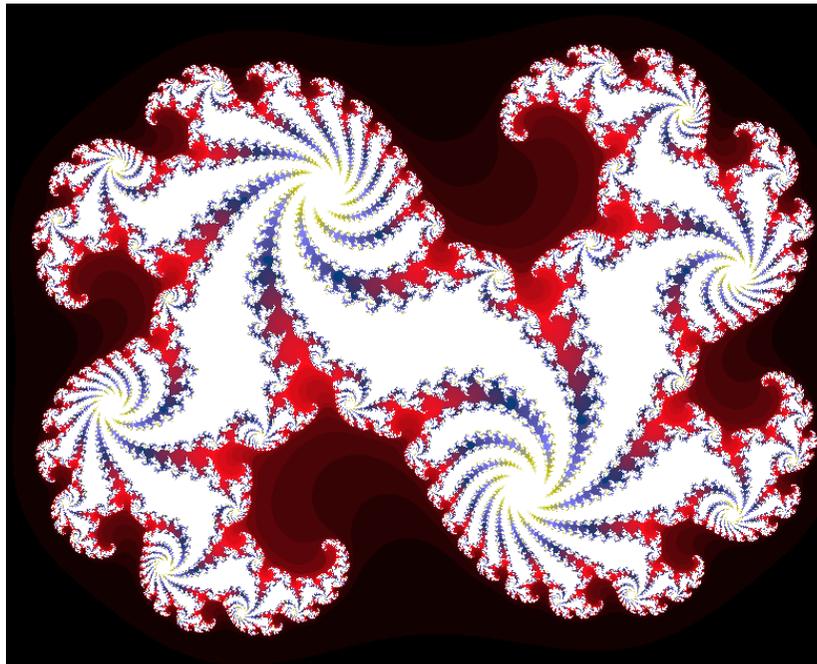
---

## ***Where to Obtain Additional Help***

Online help is available on all platforms. Just type HELP at the WAVE> prompt. The online help system contains detailed information on all PV-WAVE commands and on the PV-WAVE programming language. A complete set of online manuals for PV-WAVE and its add-on components is also available online as an optional installation on some platforms.

## *About PV-WAVE*

Using PV-WAVE's interactive programming environment, you can analyze and visualize your data in less time than with other programming tools.



**Figure 2-1** A Julia set image generated using PV-WAVE.

PV-WAVE provides an extensive set of data analysis functions for processing, analyzing and manipulating data. Programs written in PV-WAVE are shorter and execute faster than programs written in other scientific programming languages. With the seamless integration of IMSL's C/Math/Library and C/Stat/Library into PV-WAVE:IMSL Statistics and PV-WAVE:IMSL Mathematics, you can realize the strength, flexibility, and power of these mathematical and statistical libraries.

---

## ***Some Basic Rules and Concepts***

Knowledge of certain basic rules and concepts of PV-WAVE makes learning the programming language more rapid.

### **PV-WAVE Notation**

PV-WAVE provides a set of data types and operations to represent data with a natural and efficient notation. You can easily define and use structures containing aggregate data types. PV-WAVE variables, procedures, operators, and functions operate on scalar, vector, and array data with no change in notation or meaning.

PV-WAVE borrows much of its semantics from the programming language APL. The power and conciseness of PV-WAVE can be attributed to this APL influence. The main advantages over APL are syntax and control mechanisms plus visualization capabilities.

In the design of PV-WAVE, whenever there was a choice between brevity (and perhaps obscurity) and verbosity, the most readable alternative was selected.

Because scientists write their formulas using infix notation with parentheses, PV-WAVE has an expression syntax that resembles FORTRAN or BASIC, where operators are evaluated according to precedence and left-to-right sequence.

### **Extensive Error Checking**

As with any well-designed interactive language or system, extensive error checking and informative error messages are provided. The type of error and the associated variable are printed in an understandable format, without error codes or cryptic messages. You can stop a program that is running at any time and look at or change intermediate values. You can then resume the suspended program from the point of interruption.

## General Information

The following list contains basic concepts, rules, and tips that you will use over and over during any PV-WAVE session. More detail about each item is provided in subsequent lessons and in the appendix.

- ✓ Start online help by typing `HELP` at the `WAVE` prompt.
- ✓ Start online documentation by entering `HELP, /doc` at the `WAVE>` prompt.
- ✓ Obtain online information about saved variables, procedures, etc., by entering `INFO` at the `WAVE>` prompt.
- ✓ PV-WAVE is not case sensitive; upper and lower case letters are used in this manual to enable you to learn names and to distinguish functions from key-words more easily.
- ✓ Press the <Return> key to indicate you have completed the entry of a command and to execute the command.
- ✓ Use the “up” arrow ( $\uparrow$ ) key to recall the most recent line of input to PV-WAVE. (This works on most computers.) After you redisplay a line, you can edit it and then press the <Return> key to execute it. Recall up to 20 lines by pressing this key repeatedly.
- ✓ Commas (,) are used to separate one argument from another.
- ✓ Use the ampersand (&) to enter more than one command on a line.
- ✓ To continue a command from one line to another, append a dollar sign (\$) at the end of the line you wish to continue.
- ✓ The `RETALL` command is very useful when you encounter an error or interrupt. After an error, PV-WAVE stops at the end of the last procedure, which may not return you to the main program level. Your variables are available at the main program level. Entering the `RETALL` command returns you to the main program level. Alternatively, you may enter the `RETURN` command, which will return you to the next highest program level.
- ✓ A semicolon (;) is used to begin a comment in a PV-WAVE .pro file or program.

## Basic Data Types

Typing and binding of variables in PV-WAVE are dynamic, that is, the structure and type of data contained in a variable may change during a session. The basic data types that PV-WAVE variables may have are:

Byte	An 8-bit, unsigned integer ranging from 0 to 255. Pixels in images are commonly represented as byte data.
Integer	A 16-bit, signed integer ranging from $-32,768$ to $+32,767$ .
Longword	A 32-bit, signed integer ranging in value from approximately $-2 \times 10^9$ to $+2 \times 10^9$ .
Floating	A 32-bit floating point number in the range of $\pm 10^{34}$ on VAX and $\pm 10^{38}$ on machines supporting the IEEE standard, with approximately 6-7 decimal places of significance.
Double	A 64-bit double precision, floating point number in the range of $\pm 10^{34}$ on VAX and $\pm 10^{38}$ on machines supporting the IEEE standard, with approximately 13-14 decimal places of significance.
Complex	A real-imaginary pair using single-precision floating point numbers. Complex numbers are useful for signal processing and frequency domain filtering.
Double Complex	A real-imaginary pair using double-precision floating point numbers.
String	A sequence of alphanumeric characters, from 0 to 32,767 characters in length.

---

## Scalars, Arrays, and Structures

A scalar is a single instance of one of the eight data types or a single composite structure. An array is a simple structure containing multiple elements of the same data type. Array elements are addressed with subscripts and subscript ranges. A structure is an aggregation of the basic data types, other structures, and arrays.

PV-WAVE is array-oriented, that is, it handles an array as a single entity, rather than as separate numbers. As a result, you can perform mathematical operations on arrays in the same manner as on individual elements. For example, to multiply the variable  $a$  times 5, you enter " $a*5$ ". PV-WAVE performs the operation with ease, whether the variable  $a$  is a scalar, vector, or an array.

## Comparison of Matrices and Arrays

Matrices can be referenced differently from arrays by using the functions PM (Print Matrix), RM (Read Matrix), PMF (Print Matrix File), and RMF (Read Matrix File).

Matrices are used by mathematically-oriented functions and procedures. If  $\mathbf{A}$  is a matrix, then  $\mathbf{A}(i, j)$  refers to the  $i$ -th row,  $j$ -th column of  $\mathbf{A}$ . This conforms to standard mathematical notation. The elements in a matrix are stored columnwise, i.e., the elements of the 0-th column are first, followed by the elements of the 1-st column, and so forth.

Other functions and procedures, such as the image display function TV, use arrays. If  $\mathbf{A}$  is an array (an image is an array of pixels), then  $\mathbf{A}(i, j)$  refers to the pixel whose  $x$  coordinate is related to  $i$  and whose  $y$  coordinate is related to  $j$ . The elements in an array are stored row-wise, i.e., the elements of the 0-th row are first, followed by the elements of the 1-st row, etc.

PV-WAVE provides specific methods to read arrays and matrices. The PV-WAVE PRINT command returns values according to the array syntax, and the READ command is the corresponding command to read data. The PV-WAVE PM command (*Print Matrix*) and the RM command (*Read Matrix*) follow the matrix notation convention. You have the option of using the type of commands that emulate the notation with which you are most comfortable.



# *Experimenting with the Navigator*

---

## *What is the Navigator?*

The Navigator provides an easy-to-use graphical interface for performing visual data analysis. It consists of a collection of high-level components called VDA Tools. The Navigator lets you perform many of the tasks that normally require the expertise of an experienced PV-WAVE programmer.

The Navigator allows you to:

- Import and export data
- Preview ASCII data files
- Display 2D plots, images, histograms, surfaces, and contour plots
- Animate data
- View and subset tables of data
- Save and restore sessions
- View and select variables
- Modify image and plot colors

---

## **Starting PV-WAVE**

If PV-WAVE isn't already installed on your system, install it first. If you're installing PV-WAVE yourself, the installation information is included with the media you received from Visual Numerics, Inc. Along with the PV-WAVE software, the media contains several data files you will use to work through the examples.

Once PV-WAVE is installed, you're ready to begin.

### **Starting PV-WAVE Under Windows NT**

**Step 1** Click the PV-WAVE Console icon in the PV-WAVE Program Group.

After a brief pause, the PV-WAVE Console window appears displaying the prompt:

```
WAVE>
```

When you see this prompt, PV-WAVE is ready for you to enter commands.

### **Starting PV-WAVE Under Windows 95**

**Step 1** Use the **Start** button. Select **Start=>Programs=>PV-WAVE 6.0=>PV-WAVE Console**

After a brief pause, the PV-WAVE Console window appears displaying the prompt:

```
WAVE>
```

When you see this prompt, PV-WAVE is ready for you to enter commands.

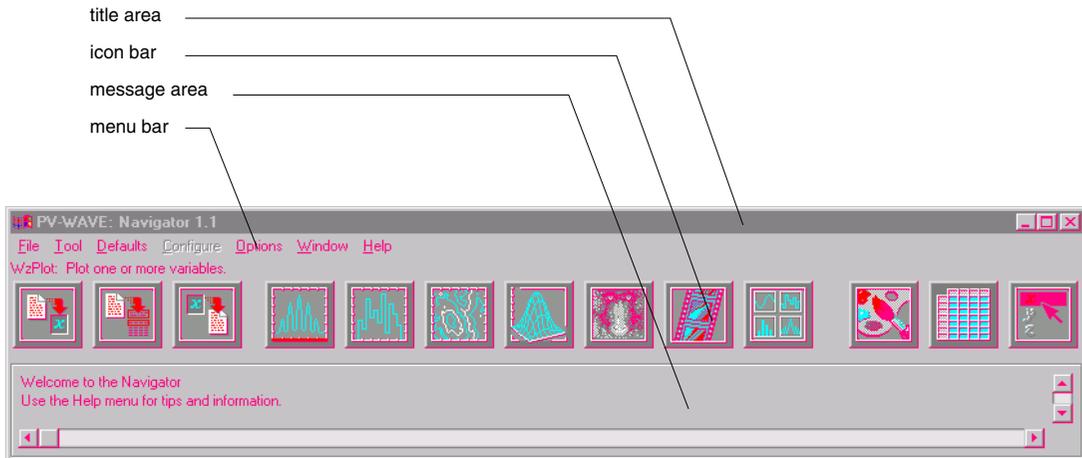
---

## Starting the Navigator

To start the Navigator, enter the following command at the PV-WAVE prompt:

```
WAVE> navigator
```

Now take a look at the Navigator window to get a feel for the point-and-click interface.



**Figure 3-1** The Navigator window consists of a title area, menu bar, icon bar, and message area.

---

**NOTE** All figures in this chapter were captured on the Windows NT version of PV-WAVE.

---

This window contains four parts:

- **Title Area** — You can move the window to a different location on the screen by pressing and holding down the left mouse button in the title area while you drag the window to its new location. The title area also contains the window manager menu (on the left) and buttons for minimizing or maximizing the window (on the right).
- **Menu Bar** — The menus available in the Navigator are listed below the title area. If you press the left mouse button and hold it down while the pointer is over a menu button, a pulldown menu appears. Try it.

The menus are pulldown style, which means you hold the mouse button down and release it over the desired menu selection. To close a menu without making a selection, drag the pointer out of the menu and release the mouse button.

- **Icon Bar** — When you click on an icon, the corresponding VDA Tool opens.

---

**TIP** To see the name of the VDA Tool associated with an icon, simply position the pointer over the icon. The name of the icon appears in the space just above the left-most icon.

---

- **Message Area** — Messages appear at the bottom of the window. They provide information that can be helpful when you're using the window.

At any time, you can exit the Navigator by selecting **File=>Close** from the Navigator window.

---

**NOTE** The notation **File=>Close** means “select the **Close** command from the **File** menu”. This notation is used throughout the tutorial.

---

Now we'll introduce you to the online help system.

---

## ***The Online Help System***

Online help is provided on all platforms. You can get information on any PV-WAVE routine and many aspects of the programming language by typing **HELP** at the **WAVE>** prompt. In addition, context sensitive help is provided for the VDA Tools and Navigator of PV-WAVE: Visual Exploration.

You will have many opportunities to explore the online help system as you work through this tutorial.

---

## ***Manuals Online***

A complete set of PV-WAVE documentation, equivalent to the hardcopy documentation set, is available as an optional installation.

If you have the online documentation installed on your system, you can start an interactive table of contents window by entering the following command at the PV-WAVE system prompt:

```
WAVE> HELP, /Documentation
```

The online documentation system includes a table of contents from which you can display pages from any manual. Hypertext links enable you to find additional information on most topics quickly. You can also print any of the online documentation files on a PostScript printer.

---

## ***Tutorial Exercises Using the Navigator***

To get you started quickly and to pave the way for using your own data, we're giving you three short Navigator exercises using datasets that we have provided for you.

The first example uses an ASCII dataset containing atmospheric temperature and air quality data. You start by importing data into PV-WAVE, and then you create simple graphics. You will see how to use the WzImage Tool to get a quick visual overview of the data, even though it was not originally obtained from an imaging device.

The second dataset was obtained from an imaging device. The data is in 8-bit image format and consists of a poor-quality image of a flame. You will use the WzImage Tool to modify the image by changing its size, cropping bad data along its edges, increasing the contrast, and adding color. You will see how easy it is to manipulate and enhance image data.

The third example contains data from shock wave measurements. Each data point represents the calculated pressure at a point in a simulated block of graphite epoxy composite. You'll look at surfaces representing pressures along slices through this volume and learn how to "animate" these surfaces to look at the progression of shock waves.

---

---

## ***Displaying Time-Series Data***

This first example uses an ASCII file containing three columns of data. Each column contains 8,760 floating-point values.

This dataset contains atmospheric data collected by environmental monitoring instruments taken in one hour intervals for an entire year, which resulted in 8,760 observations of temperature, carbon monoxide, and sulphur dioxide levels. Using the Navigator, you can analyze this data — in less time than it took to collect two sets of measurements.

In this session, you will learn how to do the following:

- Import ASCII data without elaborate conversion or programming.
- Compute and display a mathematical relationship between entire datasets.
- View your data using three different graphical methods: 2D line, image, and surface.
- View cross-sectional plots of image data.
- Experiment with different colortables.
- Use the online help system.
- Print your results.

---

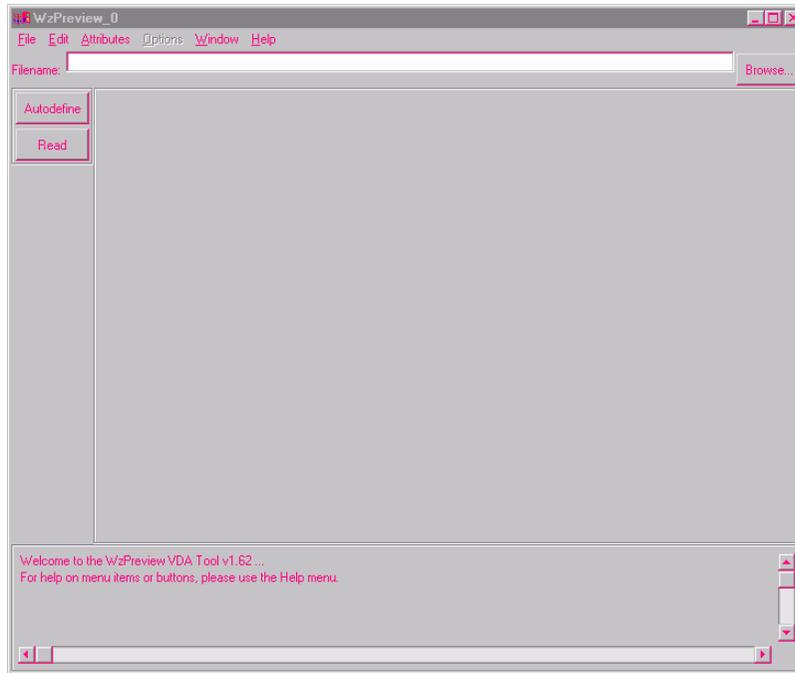
## ***Importing ASCII Data***

Start by importing the atmospheric data for the example. The data file consists of three columns:

- ✓ **Column 1** — Temperature (degrees Fahrenheit)
- ✓ **Column 2** — CO (carbon monoxide)
- ✓ **Column 3** — SO<sub>2</sub> (sulphur dioxide)

**Step 1** Click the WzPreview icon in the Navigator main window. (When you move the pointer over the icons, their names appear in the upper-left portion of the Navigator window, just above the leftmost icon.)

The WzPreview Tool appears.



**Figure 3-2** The WzPreview Tool is a VDA Tool used to read ASCII data from a file into variables.

**Step 2** Click the left mouse button in the Filename text field. This selects the field for receiving typed input.

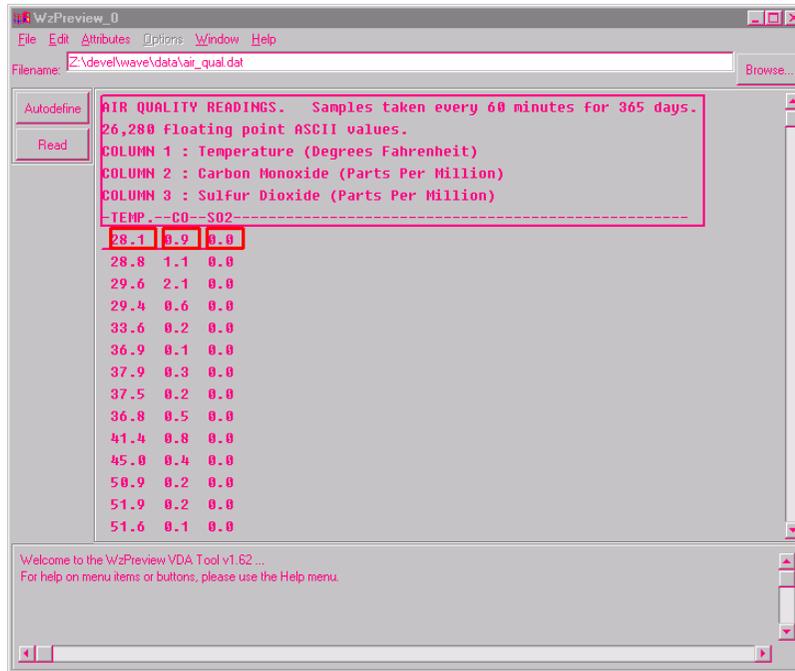
**Step 3** Enter the full path and filename of the first example data file as follows, and press <Return>:

```
<wavedir>\data\air_qual.dat
```

Where <wavedir> is the main PV=WAVE directory. This is the directory where PV=WAVE was installed. If you aren't sure which directory this is, see your system administrator.

(You can also use the BROWSE button to locate this file.)

In a few moments, the top of the `air_qual.dat` file appears in the display area of the WzPreview Tool. WzPreview has auto-defined and named the three columns of data it “sees” in the file. The width of each column is shown with a small rectangle, and the header information is differentiated from the data with a larger rectangle.



**Figure 3-3** The WzPreview Tool highlights portions of the ASCII data file that are to be read into variables.

**Step 4** Click the Read button in the control area of the WzPreview Tool. This function reads the columns of the data file into individual variables.

**Step 5** Cloze the WzPreview Tool by selecting **File=>Close**.

The WzPreview Tool has enabled you to read the data without programming or converting it. Indeed, the WzPreview Tool understood enough to read past the header information in the data file.

---

**TIP** Look in the message area — the rectangular text area along the bottom edge of the WzPreview Tool. A message informs you when the read operation is complete and tells you the names of the variables that have been created.

---

---

## Creating XY Plots

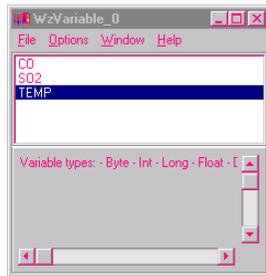
At this point, you have read ASCII data into variables with very little effort and with no programming at all.

The next step is to plot this data. Before doing this, bring up another VDA Tool called WzVariable. Think of WzVariable as a *data manager*. Use this tool to list and select the variables that you want to plot during a session.

**Step 1** Click on the WzVariable icon in the Navigator window. The WzVariable Tool appears.

All the variables you created with WzPreview are all listed in the WzVariable window.

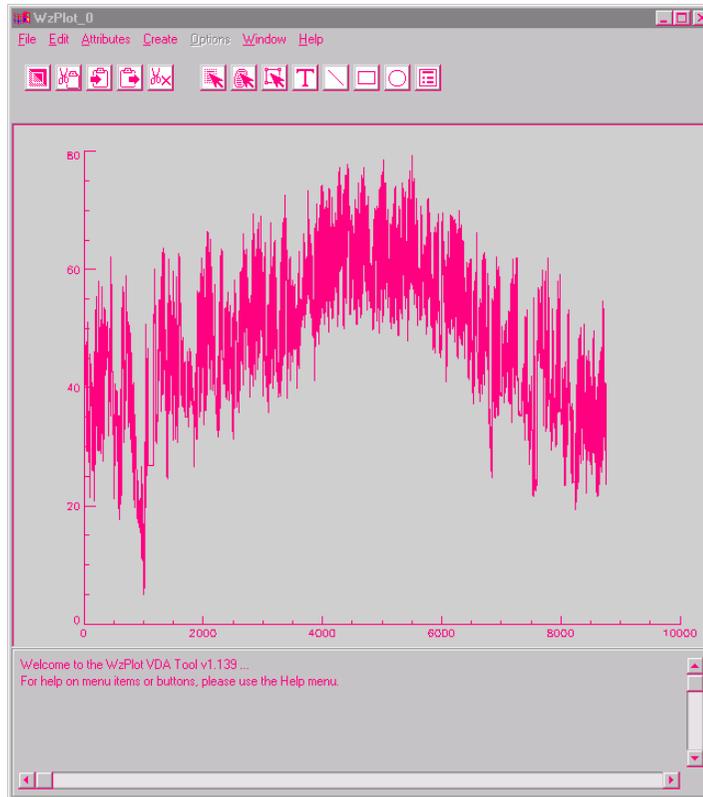
**Step 2** To get a first graphical view of your data, select TEMP from the WzVariable list by clicking on it with the left mouse button.



**Figure 3-4** The TEMP variable is selected in the WzVariable Tool.

**Step 3** Click the WzPlot icon in the Navigator window.

A 2D line plot of the data appears in the WzPlot Tool.



**Figure 3-5** The WzPlot Tool is used to plot 2D or 3D datasets.

WzPlot distributes the points evenly along the  $x$ -axis and plots their values along the  $y$ -axis. This example lets you see daily temperature variations at the observing station along with the annual variation. By looking for the place where the data drops nearly down to the bottom axis, you can spot a cold snap during the winter months.

At this point, you have meaningful graphic of the data without typing a single command.

Once you get a plot in the WzPlot Tool, you can modify it easily. Take a look at the menus available on the menu bar. Functions on the **Attributes** menu let you modify the appearance of the plot and the data. Functions on the **Create** menu let you add graphical elements such as lines and text to a graph. The button bar, just below the menu bar, contains functions for adding graphical elements, selecting data and graphics, and several editing functions.

We'll explain more about these capabilities later.

**Step 4** Exit the WzPlot Tool by selecting **File=>Close**.

---

## **Using PV-WAVE Commands**

The variables you have read in and plotted are normal PV-WAVE variables. They “exist” in PV-WAVE at the main program level. Therefore, you can perform operations on these variables from the command line using PV-WAVE functions and procedures.

Next, you will perform some simple operations on the variables you have created. Then, you will see how the processed variables can again be displayed in the WzPlot Tool.

With PV-WAVE, you can easily do mathematics on an entire array. Because of the way the numerical operators work in PV-WAVE, you can handle entire datasets with a single equation. For example, suppose you want to plot each day's ratio of SO<sub>2</sub> concentration to its temperature.

**Step 1** To compute this ratio, enter the following command at the WAVE> prompt. (This prompt appears in the window from which you started PV-WAVE. You might need to move other windows out of the way to see the WAVE> prompt window.)

```
WAVE> RATIO = SO2/TEMP
```

**Step 2** Display this newly created variable, RATIO, in the WzVariable Tool. To do this, select **Options=>Redisplay List** from the WzVariable Tool menu. The variable RATIO appears in the list. (If you closed the WzVariable Tool previously, click its icon to display it.)

**Step 3** Select RATIO by clicking the left mouse button on it.

**Step 4** Plot this variable the same way you plotted the TEMP variable previously. Simply click the WzPlot icon in the Navigator window.

**Step 5** When you have finished viewing this data, close the WzPlot Tool.

---

## Reformatting Datasets

You can reorganize data to analyze periodicity. The temperature dataset you are working with has two periods:

- ✓ **Diurnal** — A period equal to 24 hours
- ✓ **Annual** — A period equal to 365 days

You can derive some interesting graphics by converting the 8,760-point, 1D TEMP variable into a 24-by-365 2D variable. In the new variable, the first dimension now represents the hour of the day and the second dimension represents the day of the year.

**Step 1** Convert your data with the following command entered at the WAVE> prompt:

```
WAVE> TEMP_I = REFORM(TEMP, 24, 365)
```

**Step 2** Display this newly created variable, TEMP\_I, in the WzVariable Tool. To do this, select **Options=>Redisplay List** from the WzVariable Tool menu.

The variable TEMP\_I appears in the WzVariable list box.

**Step 3** Verify that this is a 2D (24-by-365) variable by double clicking on the variable name in the WzVariable Tool.

**Step 4** Click OK in the Variable Information dialog box to dismiss it.

Now you can look at this new 2D variable in some new ways. You'll use image processing techniques with this time-series data, all without any programming!

---

## Displaying Data as an Image

Viewing data as an image is a powerful technique for looking at a large amount of data at once, because an image is compact and it enables you to use color to differentiate the various values in the dataset.

**Step 1** If the `TEMP_I` variable is not already selected in the WzVariable Tool list, select it. Make sure all other variables are deselected.

**Step 2** Click on the WzImage icon in the Navigator window.

PV-WAVE displays an image of this 2D variable. Remember that in a WzImage view, each data value is represented by only one pixel.

### Resizing the Image

Even though the image does not fill the viewing area, you can resize it to make it larger and easier to view. You do this by adding additional values to the dataset.

The image you just displayed is long and narrow. To increase the size in the horizontal direction, you can simply increase the number of points in the array.

PV-WAVE provides several functions to help. Two functions, `CONGRID` and `REBIN`, will do the job. `CONGRID` interpolates a new set of points with the same overall shape as the old set. `REBIN` produces a similar result, faster than `CONGRID`, but requires that the new set be a factor or multiple of the old set.

Go ahead and use `REBIN` to increase the horizontal dimension by some multiple to get a larger number of points. The number 360 is a multiple of 24. Change the first dimension of 24 points to 360 (24 times 15) and leave the second at its original 365.

**Step 1** Enter the following command at the `WAVE>` prompt:

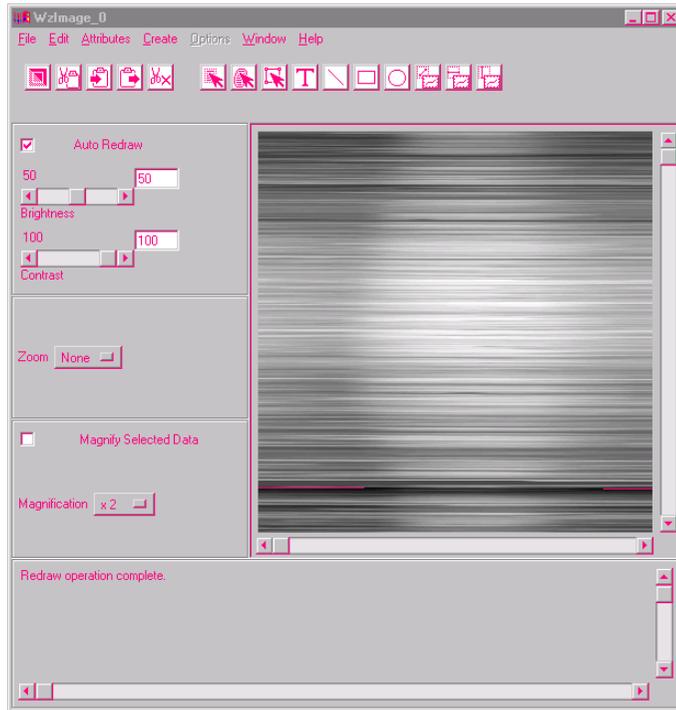
```
WAVE> TEMP_I = REBIN(TEMP_I, 24*15, 365)
```

PV-WAVE will evaluate the mathematical expression given as the second parameter, resulting in a first dimension of 360.

**Step 2** Now, just click the Redraw icon on the WzImage Tool button bar. (Redraw is the leftmost icon.)

The image is updated when you click `Redraw` because the variable you updated is the same one that is currently displayed in the VDA Tool.

**Step 3** Resize the WzImage window so that the entire image is displayed. Do this by pressing the left mouse button over the lower right corner of the border and dragging.



**Figure 3-6** A display of the 2D variable using the WzImage Tool.

WzImage automatically scales the numerical values of the dataset so that every shade of color (or every shade of gray) is utilized. The lightest points have a numerical value of 127, while the darkest have a value of 0 (zero).

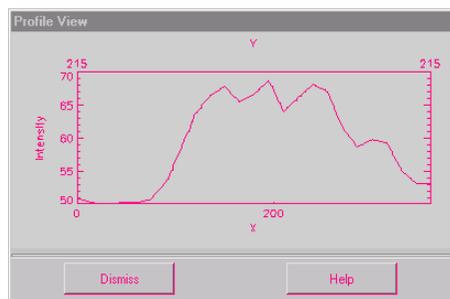
The origin of the image is at the lower left; thus, the beginning of the year is at the bottom, and the beginning of each day is at the left.

As you would expect, the temperature is at its minimum (the darkest colors) near the beginning of each day and at its maximum (the lightest colors) near the middle of each day. It's also easy to see that the average daily temperature reaches its minimum early in the year and its maximum in the middle of the year.

## Displaying Profile Plots

You can easily create profile plots. With the data you are using in this tutorial, a row profile plot represents temperature variations during a day. A column profile plot represents the temperature for each day of the year, measured at the same time each day. The profile, which is like a cross-section of the image, allows you to examine a particular portion of the image with more precision than you can in the image itself.

- Step 1** Select either the Row Profile button or the Column Profile button from the button bar (just below the menu bar on the WzImage Tool), or select **Column Profile** or **Row Profile** from the **Edit** menu to start a row or column plot. A blank viewing mode window appears, ready to display the row or column that you select.
- Step 2** Using the left mouse button, click the image at the point where you want to see a cross-section. Clicking the left mouse button displays either a row or column plot, depending on which profile button you chose. Remember that the plots are scaled to fit the available viewing area.
- Step 3** Press and drag the left mouse button and the profile is updated dynamically.



**Figure 3-7** An example of the x-y profile view of the 2D variable.

- Step 4** When you are through looking at profiles, click the Dismiss button in the Profile window.

## Manipulating Data Across VDA Tools

This example demonstrates how VDA Tools can share the same data. When you modify the data in one VDA Tool, the changes take effect for all VDA Tools sharing the same data.

**Step 1** First, in the WzVariable List, select the variable TEMP.

**Step 2** Now, click the WzPlot icon in the Navigator window. This brings up the same 2D plot you saw previously.

Remember the dip that occurs early in the sampling period, which we attributed to a cold snap? Let's assume that there wasn't a cold snap, but rather the data sampled during that period was erroneous.

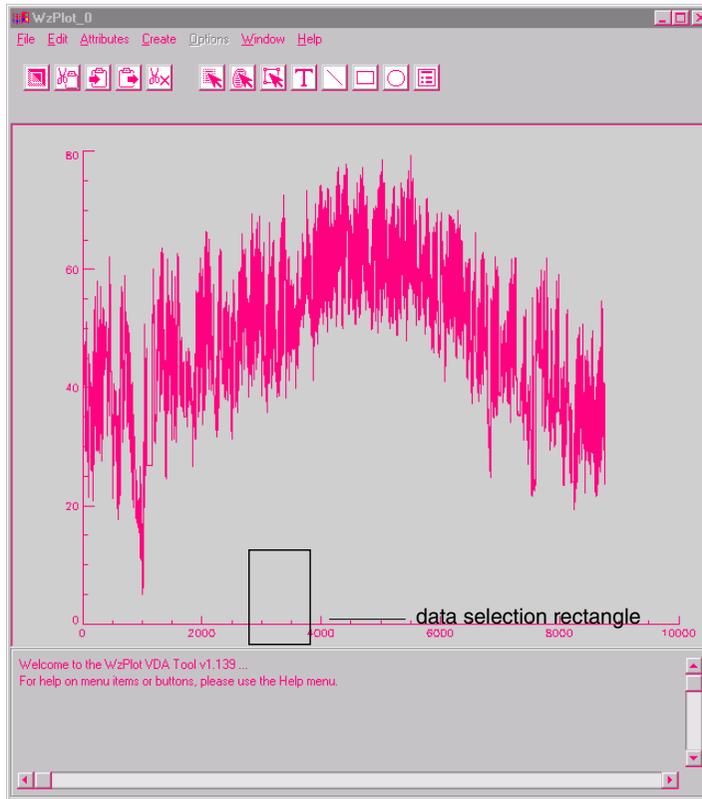
Let's correct the faulty data using the data selection feature and the WzTable Tool.

**Step 3** Click the WzTable icon in the Navigator window. WzTable opens and the individual data values from the TEMP variable are shown in the table cells.

**Step 4** Move the windows so that you can see both the WzTable Tool and WzPlot Tool at once.

**Step 5** In the WzPlot Tool, select **Edit=>Data Select**. This puts the Tool in data selection mode.

**Step 6** Select the downward spike that we previously attributed to a cold snap. To do this, press and drag the left mouse button to draw a rectangle around the data. Try to select the entire spike, approximately as shown in the following figure.



**Figure 3-8** An area of the 2D plot selected for analysis or data manipulation using the data selection mode.

**Step 7** Now, edit the variable TEMP directly by entering 32 in the Change Selected Values To field of the WzTable Tool.

**Step 8** Press <Return>.

The plot in WzPlot is automatically updated with the new values. Note that this operation has directly modified the variable TEMP. You could have edited each value in the table individually instead of setting them all to one value.

**Step 9** See what happens when you select **Edit=>Undo** from the WzTable menu bar.

We encourage you to experiment with other values and other selected ranges of data.

**Step 10** Close the WzPlot and WzTable windows.

## Using Color

PV-WAVE supplies you with a number of colortables that display values in color rather than shades of gray.

- Step 1** To try color, select the WzColorEdit icon from the Navigator window.
- Step 2** If necessary, move the WzColorEdit Tool away from WzImage Tool, so that you can see the image.
- Step 3** Select **ColorTable=>System** to bring up the list of System Color Tables.
- Step 4** Click on the Blue-Red item in the list and notice the change.
- Step 5** Click the Redraw icon in the WzImage Tool, or select **Edit=>Redraw** from the WzImage Tool menu.

The image now appears in colors better suited for the display of this temperature data. If you like, you can try some of the other system color tables available to you.

- Step 6** Click Dismiss when you are finished with the list of System Color Tables.
- Step 7** Close the WzColorEdit Tool by selecting **File=>Close** when you are done experimenting with colors.
- Step 8** Close the WzImage Tool.

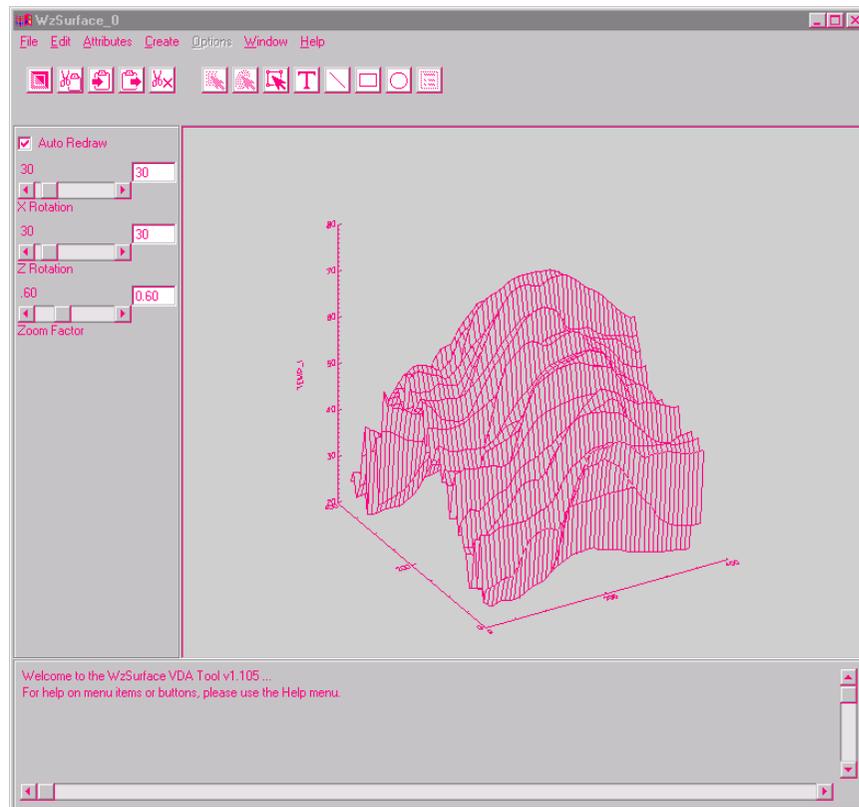
---

## Creating Three-Dimensional Surface Plots

You can display the data as a mesh surface plot.

**Step 1** If the TEMP\_I variable is not already selected in the WzVariable Tool list, select it. Make sure all other variables are deselected.

**Step 2** Click on the WzSurface icon in the Navigator window.



**Figure 3-9** The WzSurface Tool provides yet another way to look at the same temperature data you were viewing earlier as a 2D line and as an image.

---

## Using Online Help

Use online help anytime you want more detailed information about a particular Navigator feature. Online help also gives you fast access to reference information on all PV-WAVE functions and procedures and features of the programming language.

We encourage you to experiment with the help system as you progress through this tutorial.

### Context-Sensitive Help

When you are using the Navigator or VDA Tools, context-sensitive help is available to you at any time during the session. Context-sensitive help is available when you select **Help=>On Window** from the Navigator menu or from any VDA Tool menu. **On Window** displays a table of contents of information about the specific window.

**Step 1** Select **Help=>On Window** from the WzSurface Tool menu bar. A table of contents for the VDA Tool appears.

---

**TIP** You can leave the help window open during an entire session, and refer to it at any time.

---

In addition, all dialog boxes have a **Help** button that brings up specific reference information about that dialog box.

**Step 2** Select **Attributes=>Surface Attributes** from the WzSurface Tool menu bar. The Surface Attributes dialog box appears.

**Step 3** Click the **Help** button in the dialog box. Information on the dialog box is displayed in the help window.

**Step 4** Click the **Cancel** button to close the Surface Attributes dialog box.

### PV-WAVE Online Help

First, let's see what information is available on the REBIN command that you used during this lesson. REBIN is a PV-WAVE function.

**Step 1** At the `WAVE>` prompt, type:

```
WAVE> HELP , 'REBIN'
```

Reference information on the REBIN function is displayed in the help window.

**Step 2** Type the following command at the `WAVE>` prompt:

WAVE> HELP, 'REFORM'

The reference information on the REFORM function is displayed in the help window.

---

**TIP** If you want to learn more about the online help system, select **Help=>On Help** from the menu bar of any VDA Tool or the Navigator.

---

---

## ***Printing Your Results***

Obtaining a printed version of a VDA Tool window is often an important step, because it enables you to share your results with others. But if you do not wish to create a hardcopy version of the surface you see in the VDA Tool, you may skip this section.

- Step 1** To print a copy of the plot that you see in the WzSurface Tool, select **File=>Print**. The Printer Setup dialog box appears.
- Step 2** Type the appropriate print queue name in the text field. If you do not know the print queue name, ask your System Administrator, or just leave the text field empty to try the default queue.
- Step 3** Select the appropriate output format for your printer from the Printer Type option menu.
- Step 4** Select the Print button.
- Step 5** Click OK to dismiss the dialog box. (If you type the wrong print queue, or wish to change the print queue, you need to select **File=>Print Setup** to display the Printer Setup dialog box again.)

---

**TIP** If you wish to print subsequent graphics during this session, you can simply select **File=>Print**. This is because you only have to set up your printer options once during a session. The Printer Setup dialog box will not appear again unless you explicitly select **File=>Print Setup**.

---

- Step 6** Close the WzSurface Tool by selecting **File=>Close**.
- Step 7** Also close the WzVariable Tool.

Printing procedures will vary from site to site. If you have difficulty printing, contact your System Administrator to determine the proper printer queue names, output formats, and printer options.

---

## ***Experimenting on Your Own***

In this example, you have done some simple visual data analysis, mostly of temperature data. You may want to continue working with this data to explore the relationship of temperature to the recorded levels of CO and SO<sub>2</sub> and we encourage you to do so.

In this session, you have learned how to do the following:

- Import ASCII data without elaborate conversion or programming.
- Compute and display a mathematical relationship between entire datasets.
- View your data using three different graphical methods: 2D line, image, and surface.
- View cross-sectional plots of image data.
- Experiment with different colortables.
- Use the online help system.
- Print your results.

If you have an ASCII dataset of your own that would lend itself to similar visual data analysis, try it.

---

## ***Ending the Session***

You can exit the Navigator and PV-WAVE session or continue to the next tutorial.

When you exit the Navigator, all of the data you have imported or created through processing remains on the main program level of PV-WAVE. When you exit PV-WAVE, however, all the data you have imported or created will be lost unless you have saved it.

Of course, the original data in the imported file remains intact.

---

**NOTE** We'll discuss saving files in a later example, so don't bother to save your data now. You won't need the results from this example for any future session.

---

- To exit the Navigator, choose **File=>Close** from the Navigator window. To exit from PV-WAVE altogether, type EXIT at the WAVE> prompt.

---

---

## Displaying Image Data

The next dataset to experiment with consists of an image of a flame obtained with an imaging device. This dataset is a 128-by-128 2D array. The data is 8-bit binary, and thus has values ranging from 0 to 255.

We have purposely given you a dataset that contains a poor image so that you can see how easy it is to use the Navigator to enhance the image.

- Step 1** If it is not already running, start PV-WAVE. (See [Starting PV-WAVE on page 16](#) if you need detailed instructions.)
- Step 2** If it is not already running, start the Navigator by typing `navigator` at the `WAVE>` prompt.

---

---

## Importing 8-bit Image Data

Because the data type for this 8-bit data is byte, you do not import the data with the WzPreview Tool, as you did in the previous example. (Remember that the WzPreview Tool is designed for importing ASCII data.) For this type of data, you need to use the WzImport Tool.

- Step 1** Click the WzImport icon in the Navigator window. The WzImport Tool appears.
- Step 2** From the **File Type** option menu of the WzImport Tool, select **8 Bit Image**.
- Step 3** In the File Name text field, type the full path and file name of the image data file, as follows:

```
<wavedir>\data\flame.img
```

Where `<wavedir>` is the main PV-WAVE directory. This is the directory where PV-WAVE was installed. If you aren't sure which directory this is, see your system administrator.

- Step 4** Select the Read button.

A message appears in the message area indicating that the variable `IMG8` has been created.

- Step 5** Close the WzImport Tool. (Select **File=>Close**.)

---

## ***Checking the Selected Variables List***

To display a variable in a graphical VDA Tool, such as WzImage, the variable must be “selected”. One way to select a variable is to open the WzVariable Tool and click on the variable name in the list box. That was how you selected variables in the previous lesson.

Anytime you import data into a variable using WzImport, the newly created variable is automatically “selected”. In other words, you do not have to bring up WzVariable now (unless you want to).

You can always determine which variable is the currently selected variable, by doing the following:

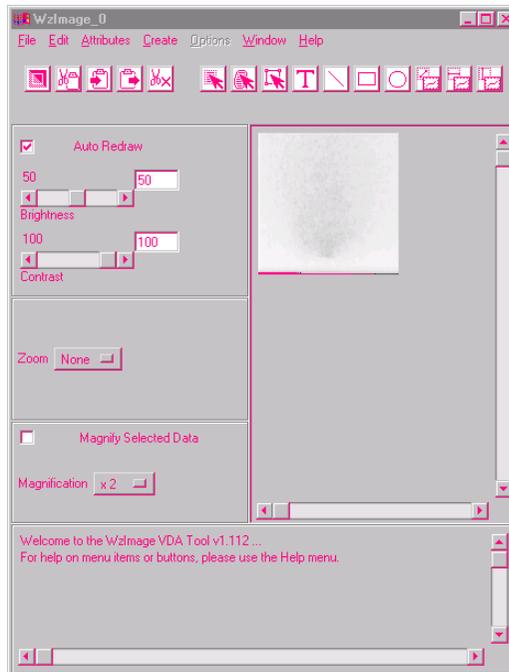
- Choose **Tool=>List Selected Variables** from the Navigator window menu. The currently selected variable or variables are then listed in the message area of the Navigator window. In this case, the variable IMG8, the variable you just read in, is listed there.

---

## Displaying the Image

Because the image variable is already selected, all you need to do is click the WzImage icon in the Navigator window.

The image is displayed in the WzImage Tool.



**Figure 3-10** The selected variable is displayed in the default WzImage Tool window.

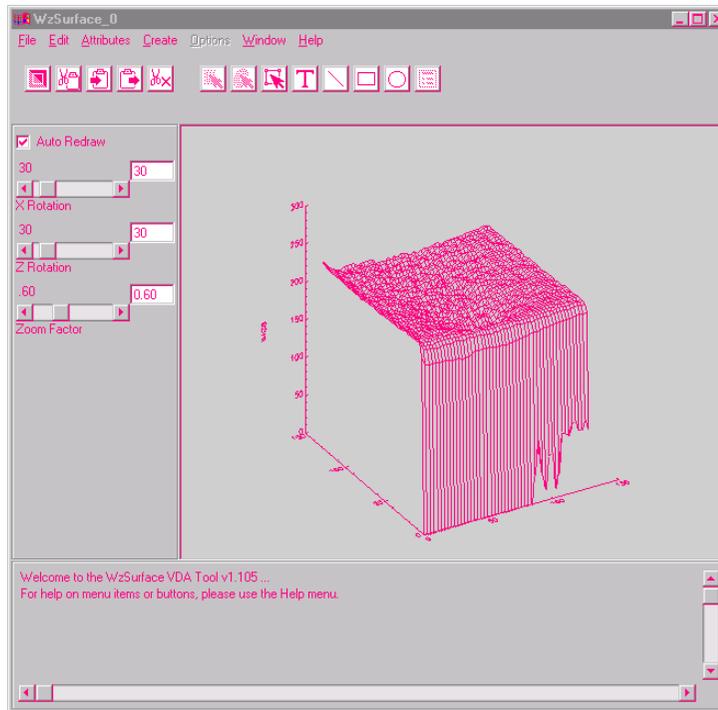
Note that the displayed image doesn't look very distinctive and is small compared with the available view area. That is because it is 128 x 128 pixels, while the view area can display a much larger area.

---

## Displaying the Image as a 3D Surface Plot

**Step 1** To display the image as a 3D surface plot, click the WzSurface icon in the Navigator window.

A 3D mesh surface of the flame is displayed in the WzSurface Tool.



**Figure 3-11** A 3D mesh surface of the image data.

You can also see that the edge of the surface contains spurious data, shown as a cliff. This has an undesirable effect on the display of the image since contrast in the image is scaled from the lowest to the highest valued point. It's like accidentally getting the sun in a photograph, which over-saturates the image. A little later you'll use PV-WAVE commands to remove this meaningless data.

**Step 2** Close the WzSurface Tool.

---

## ***Using Color***

Now try looking at the image of the flame with a different set of colors.

- Step 1** Click the WzColorEdit icon on the Navigator window. The WzColorEdit Tool appears.
- Step 2** If necessary, move the windows around so that you can see both the WzImage Tool and WzColorEdit.
- Step 3** In the WzColorEdit Tool, select **ColorTable=>System**. The System Color Tables list box appears.
- Step 4** To use color more effectively, select the RED TEMPERATURE color table and click the Dismiss button.

You will see the image in shades of red. It's still a poor quality image; however, the next few steps will improve it.

- Step 5** Close the WzColorEdit Tool. (For now, do not close the WzImage Tool. It will be interesting to compare this original image with the enhanced image you will create later)

---

## ***Rescaling Image Data***

You can eliminate much of the undesirable effect of the zeros at the edge of the image by using the PV-WAVE BYTSCL function.

BYTSCL scales your data so that all values below the specified minimum are scaled to 0, and all points with values above the specified maximum are scaled to the new maximum. When the resulting data is displayed, it scales all the points to values ranging from 0 to 255.

In the flame image, all real points fall between 200 and 255. By selecting 200 as a minimum, you set the spurious edge data to the same values as the darkest part of the flame. This will result in a marked improvement in image contrast, so go ahead and make the change.

- Step 1** Enter the following command at the WAVE> prompt:

```
WAVE> FLAME1 = BYTSCL(IMG8, Min=200)
```

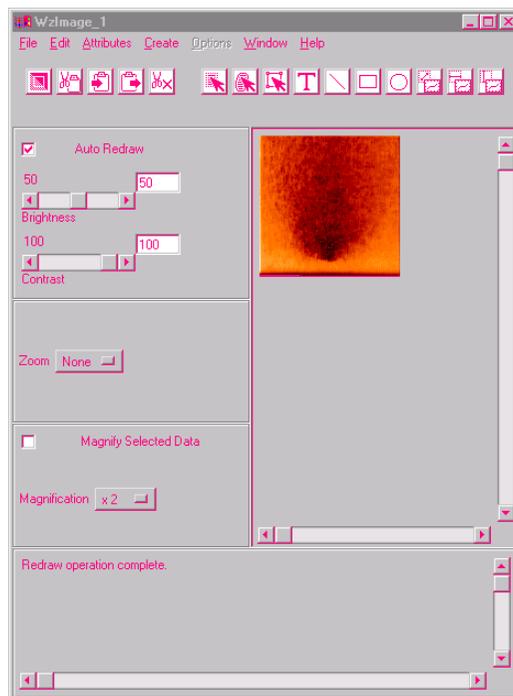
The variable FLAME1 is the byte-scaled data with the effects of the zeros along the edge removed. You will now select and view the data as an image to see the improvement.

Instead of using the Navigator, let's see how a VDA Tool can be started directly from the command line. All VDA Tools can be started in this manner, without going through the Navigator.

**Step 2** At the WAVE> prompt, enter the following command:

```
WAVE> WzImage, FLAME1
```

The rescaled image is displayed in the WzImage Tool.



**Figure 3-12** Rescaled data is displayed in the WzImage Tool.

---

## Resizing the Image

Use the REBIN function to increase the size of the image.

**Step 1** Enter the following command at the WAVE> prompt:

```
WAVE> FLAME2 = REBIN(FLAME1, 512, 512)
```

**Step 2** Open the WzVariable Tool.

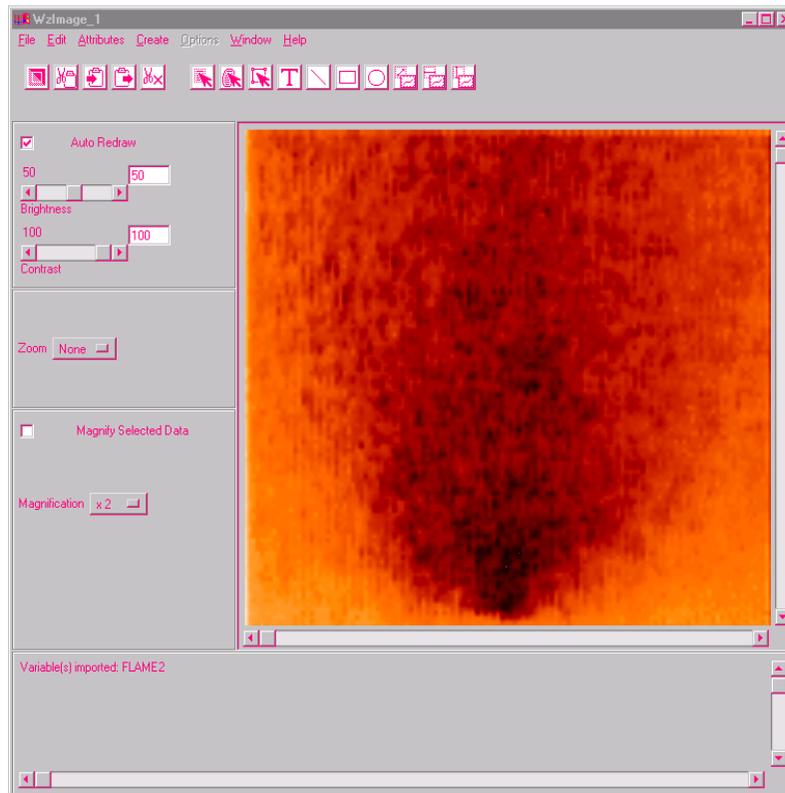
**Step 3** In the WzVariable Tool list, select the variable FLAME2.

**Step 4** Select **File=>Export Variables** from the WzVariable Tool menu bar. The Export Variables dialog box appears.

**Step 5** Select FLAME2 in the upper list box of the dialog box and WzImage\_1 in the lower one.

**Step 6** Click the OK button in the Export Variables dialog box.

**Step 7** Resize the WzImage window so the entire image fits in the display area.



**Figure 3-13** The resized image is displayed in the WzImage Tool.

The image now is much larger and fills the view area. As a result of resampling, the image has a blotchy appearance. Next, you will smooth the image.

---

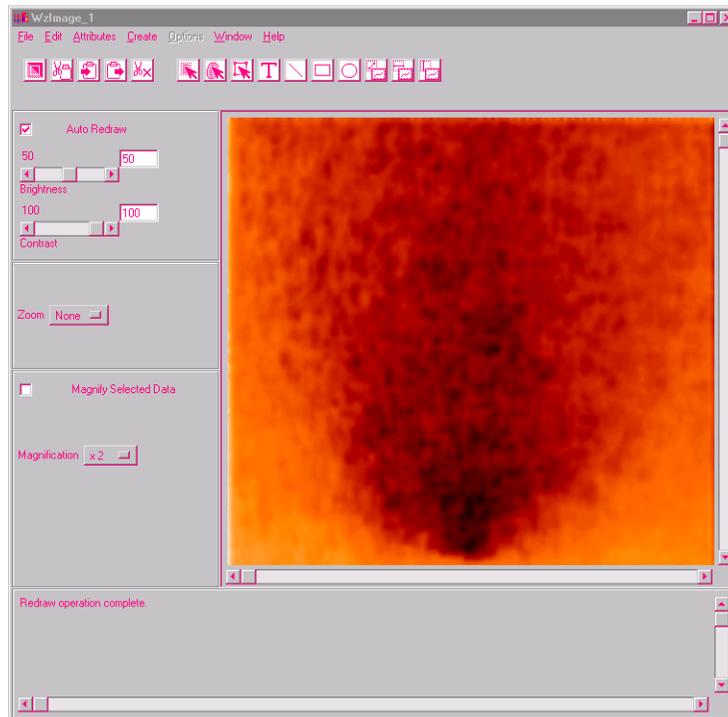
## Smoothing the Image

The smoothing algorithm works by changing the values based on nearby values. The PV-WAVE SMOOTH command takes a *width* parameter, which should be chosen as small as possible to still provide adequate smoothing. The larger the value of width, the more nearby values are considered in the smoothing algorithm. The effect is greater smoothing at the risk of losing detail. With this data, a good value to use for width is 7.

**Step 1** Enter the following command at the WAVE> prompt:

```
WAVE> FLAME2 = SMOOTH (FLAME2, 7)
```

**Step 2** Now view the smoothed image of FLAME2 by clicking on the Redraw icon in the button bar of the WzImage Tool.



**Figure 3-14** The smoothed image is displayed in the WzImage Tool.

The image is noticeably smoother. If you like, experiment with different values of width.

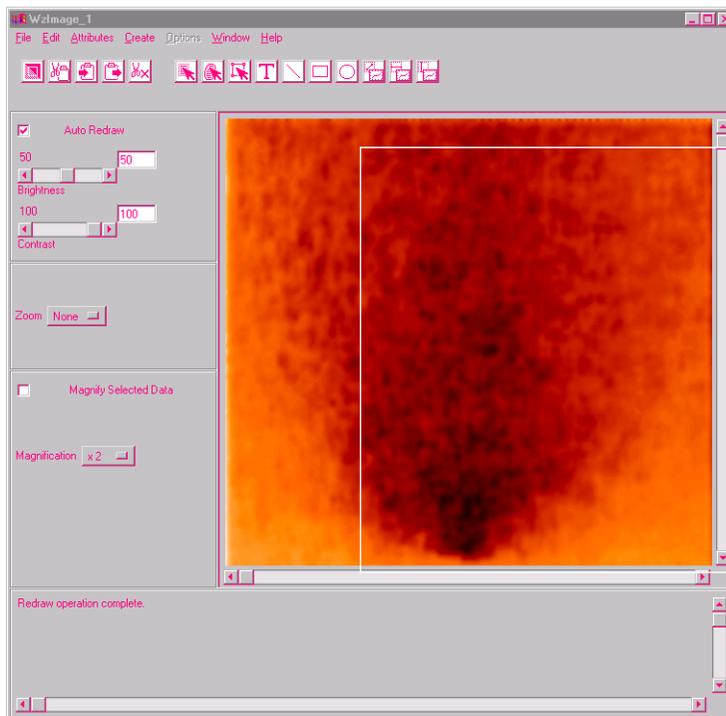
---

## Cropping the Image

Even though the spurious edge data is no longer affecting the quality of the flame image, you may want to remove this data by cropping the image. To graphically crop an image, you will use the Data Selection function.

**Step 1** Click the Data Selection button on the button bar of the WzImage Tool or choose **Edit=>Data Select**.

**Step 2** Press and drag the left mouse button to draw a rectangle around part of the image, approximately as shown in the following figure.



**Figure 3-15** Part of the image data is selected using the mouse.

**Step 3** Select **File=>Export Selected Data** in the WzImage Tool. The Export Selected Data dialog box appears. This dialog box contains two list boxes. The first list box contains the name of a variable that was created

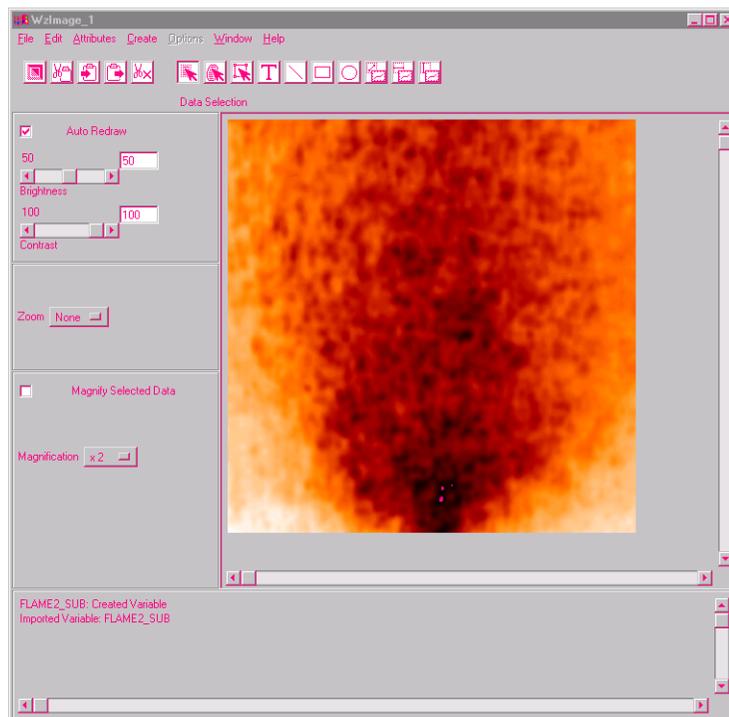
to hold the subsetted data. The second list box contains the names of the VDA Tools that are currently open.

**Step 4** Select the variable name FLAME2\_SUB in the first list box, and select WzImage\_1 in the second list box.

**Step 5** Click OK.

The subsetted variable is exported back to the original WzImage Tool. The exported variable is automatically displayed in the WzImage Tool.

Notice how cropping the image affects its appearance.



**Figure 3-16** The image contrast is improved by cropping spurious edge data.

---

## Creating an Inverted Image

You can create an inverted image by subtracting the value of each element in the variable from 255.

---

**NOTE** The number 255 is used because the data was byte scaled into the range 0-255. If, for example, the data was 4 in the range 0-127, then inversion would be achieved by subtracting your data from 127, and so on.

---

**Step 1** Enter the following command at the WAVE> prompt:

```
WAVE> FLAME3 = 255 - FLAME2_SUB
```

**Step 2** In the WzVariable Tool, select **Options=>Redisplay List**. The FLAME3 variable appears in the list.

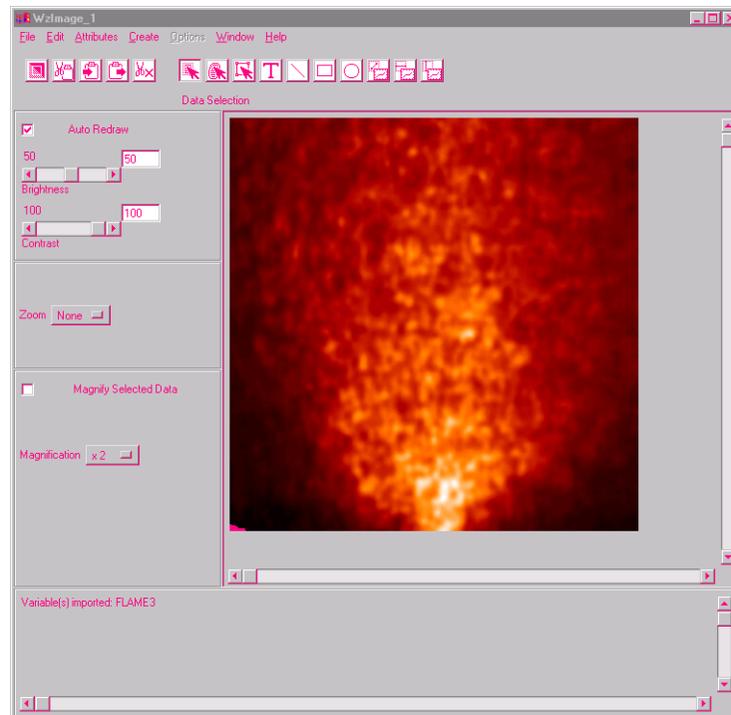
**Step 3** In the WzVariable Tool list, select the variable FLAME3.

**Step 4** Select **File=>Export Variables** from the WzVariable Tool menu bar.

**Step 5** Select FLAME3 in the upper list box and WzImage\_1 in the lower one.

**Step 6** Click OK.

You now see an inverted image of the flame.



**Figure 3-17** The flame image has been inverted.

---

## **Editing Color**

You can select your own combinations of red, green, and blue for a colortable to see different effects in the image display. In this example you will change the color of data with values 60 and 100.

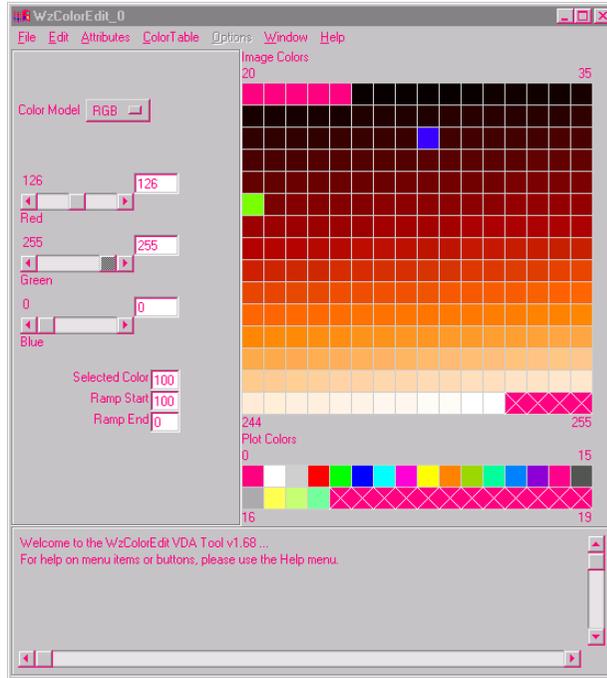
- Step 1** Click the WzColorEdit icon in the Navigator window. The WzColorEdit Tool appears.
- Step 2** In the Selected Color text field, type 60 and press <Return> to select colortable index 60.
- Step 3** Place the pointer on the slider portion of the scale labeled Blue. Press and hold down the left mouse button and drag the slider to the right to set this color index to a shade of blue. Release the left mouse button after adjusting the scale.
- Step 4** Click the Redraw button, or select **Edit=>Redraw**, in the WzImage Tool.

---

**NOTE** If your Windows system is set to 256 Colors display mode, you can see the changes in the color display of the image as you interact with the slider. If your display is set to 65536 Colors or True Color mode, you have to click Redraw (or select **Edit=>Redraw**) in the WzImage Tool to see the changes take effect.

---

- Step 5** Now do the same for the color index 100, but make it green.



**Figure 3-18** Editing colors using the WzColorEdit Tool.

Take a moment to look at the remarkable highlighting effects in the flame image. Try experimenting with other color changes.

**Step 6** Close the **WzColorEdit** Tool.

**Step 7** Finally, compare this enhanced, modified image with the original image displayed in window WzImage\_0.

---

## ***Experimenting on Your Own***

In this session, you have learned how to do the following:

- Read 8-bit image data from a file.
- Use PV-WAVE commands to improve the quality and contrast of an image.
- Interactively subset an image before performing further analysis on a region of interest.
- Change the appearance of an image by selecting a different colortable.

If you have any 8-bit (or byte) data of your own that would lend itself to visual display, try it.

---

## ***Saving the Session***

When you save a Navigator session, you can save all or some of the VDA Tools that are currently open. Once the session is saved, you can reopen it at any time, fully restoring the VDA Tools and data that was saved.

**Step 1** Select **File=>Save Session** from the Navigator menu bar. The Save Session dialog box appears.

**Step 2** In the Save Session dialog box, select the following items from the list using the left mouse button to select the first item and <Shift>-<left mouse button> to select the others: **WzImage\_0**, **WzImage\_1**, **WzVariable\_0**.

**Step 3** In the Session File Name text field, type: `flame.sav`

**Step 4** Click the OK button.

Now the session is saved. To restore the session at any time in the future, simply start PV-WAVE and the Navigator, then use the **File=>Restore Session** function from the Navigator menu bar.

**Step 5** Close the open VDA Tools—**WzImage\_0**, **WzImage\_1**, and **WzVariable\_0**.

---

## ***Ending the Session***

You can now either exit the PV-WAVE session or start a new tutorial without quitting.

- To exit the Navigator, select **File=>Close** on the Navigator window.
- To exit PV-WAVE type `EXIT` at the `WAVE>` prompt.

---

---

## ***Animating Image Data***

In this example, you'll get a chance to experiment with animation using the VDA Tool WzAnimate. You observe the animation in the same manner that you would view a movie — by the rapid display of a series of still graphics.

We chose an example that allows you to observe how a shock wave, radiating through a block of graphite composite, is reflected off the boundaries of the block. You'll also get some practice adding annotation to a plot.

**Step 1** If you are not presently running the Navigator, you will need to start it now. Follow the same procedure to start the Navigator as you followed in the previous two examples.

---

---

## ***Importing the Animation Data***

Start by importing some multidimensional shock wave data into PV-WAVE. The data for this exercise has been provided for you as a save file. The data was generated in this format using PV-WAVE commands.

**Step 1** To read the PV-WAVE save file, enter the following command at the PV-WAVE prompt:

```
WAVE> RESTORE, !Data_Dir+'shock_wave.sav'
```

Now the data is ready to use as a series of PV-WAVE variables.

**Step 2** Click on the WzVariable icon in the Navigator window. The WzVariable Tool appears.

**Step 3** Resize the WzVariable Tool by dragging the bottom right corner downward until the window is about six inches long. Resizing allows you to avoid scrolling to select variables.

Note that a series of variables beginning with “DSP” and “STR” appear in the WzVariable Tool list. All of these variables were imported from the same save file.

The variables represent data from 14 time slices taken at 1 millisecond intervals. The variable with the name beginning in DSP contains displacement data. These data values represent the magnitude of the displacement of molecules from their equilibrium position in a plane in the block. The variables with names beginning in STR contain stress data. These data values represent stress on points in the plane.

Each variable contains an array of 55-by-40 displacement or shock values. That is, each of the 55 slices has 40 points at which data was measured.

---

## Viewing Displacement as an Image

To begin, increase the size of the image DSP14 using the REBIN command and then view the image to see what it looks like before you animate it.

**Step 1** Enter the following command at the PV-WAVE prompt:

```
WAVE> DSP_MAG = REBIN(DSP14, 55*10, 40*10)
```

This increases the size by a factor of ten. Remember that with REBIN, any array you create must be a multiple or factor of the original array.

**Step 2** In the WzVariable Tool, select **Options=>Redisplay List**. The newly created variable now appears in the list.

**Step 3** Click on the DSP\_MAG variable in the WzVariable Tool. Now that variable is selected and ready to be displayed as an image.

**Step 4** Click the WzImage icon in the Navigator window.

The image is displayed in the WzImage Tool.

**Step 5** Click the WzColorEdit icon in the Navigator window.

**Step 6** Select **Colortable=>System**, and then select **Blue-Red** from the color-table list.

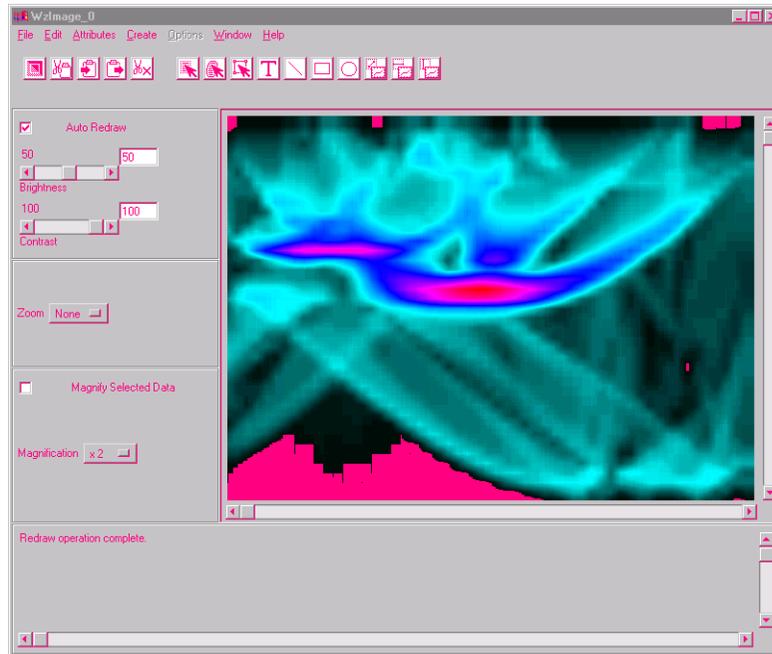
**Step 7** Click the Redraw button, or select **Edit=>Redraw**, in the WzImage Tool.

You'll notice that the color of the image has changed to blue and red.

---

**NOTE** If your Windows system is set to 256 Colors display mode, you can see the changes in the color display of the image as you interact with the slider. If your display is set to 65536 Colors or True Color mode, you have to click Redraw (or select **Edit=>Redraw**) in the WzImage Tool to see the changes take effect.

---



**Figure 3-19** Using the WzImage Tool to display displacement data.

**Step 8** Click on Dismiss in the System Color Table list.

**Step 9** Close the WzColorEdit Tool

**Step 10** Close the WzImage Tool.

---

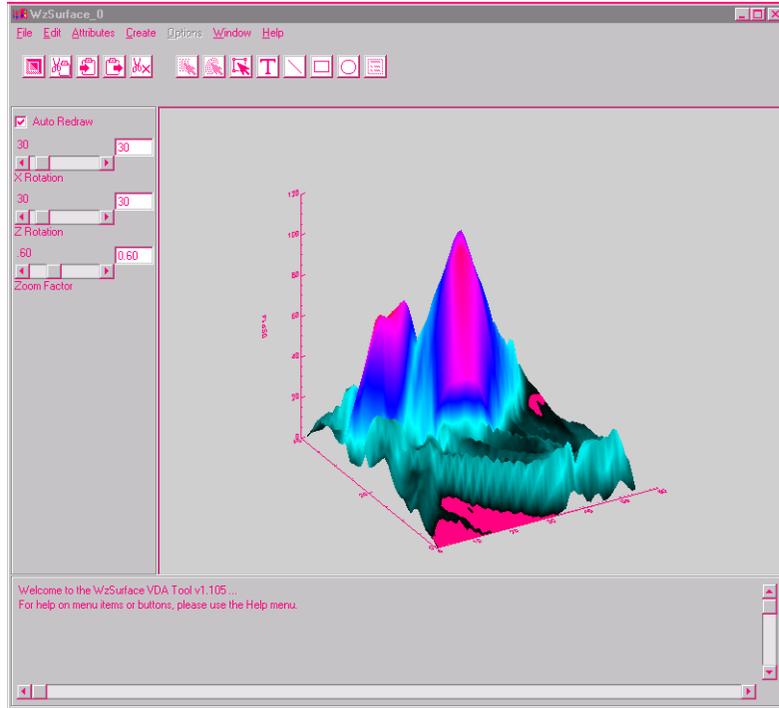
## ***Shading and Rotating a Surface View***

Now create a surface view using two variables. Use DSP14 for the surface, and shade it using STR14. By shading the surface, you create a view that contains more information than can be obtained from either variable alone.

- Step 1** In the WzVariable Tool list, select DSP14.
- Step 2** Click the WzSurface icon in the Navigator window.
- Step 3** Select **Attributes=>Surface Attributes** from the WzSurface menu bar. The Surface Attributes dialog box appears.

Now select your view attributes by interacting with some option menus.

- Step 4** Select None for the Surface attribute (in the Surface option menu in the dialog box).
- Step 5** Now select From Variable from the Shading option menu.
- Step 6** Type STR14 in the Shade Variable text input field.
- Step 7** Select the OK button. The **Surface** view window with the shaded surface appears.



**Figure 3-20** The default view displays a surface at a 30 degree angle of rotation about the z-axis, which is displayed as the vertical axis.

Your individual surface plot might be better viewed from a different angle. The WzSurface Tool lets you rotate to any view angle.

Practice changing the rotation by changing the angle to 120 degrees.

**Step 8** Select the value 30 next to the Z Rotation slider and change the value to 120. Press <Return> to confirm the rotation change.

**Step 9** When you finish viewing the surface, close the WzSurface Tool.

---

## Building an Animation Sequence

Next, you will create an animation sequence with the WzAnimate Tool. You individually create the frames that will be animated and store them in a variable. After the sequence is complete, you can animate the frames forward or backward. WzAnimate displays the views in rapid sequence, allowing you to get a dynamic, interactive view of your data.

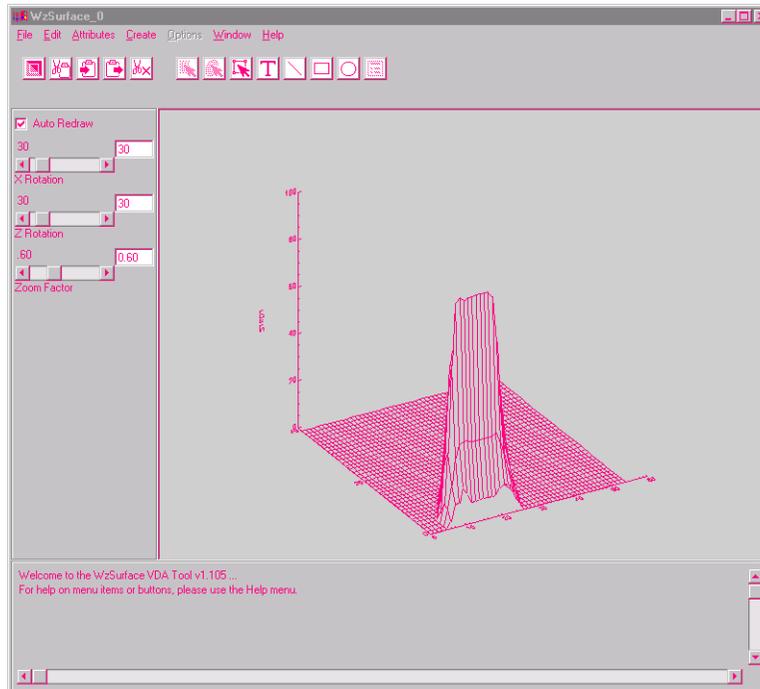
For this example, you're animating surface views of the data. The first step is to create an animation array.

### Defining an Animation Array

You will create the array at the same time that you store the first frame. Your animation sequence will consist of six “snapshots” of the shock wave.

**Step 1** Select STR01 in the WzVariable list.

**Step 2** Click the WzSurface icon in the Navigator window. A WzSurface Tool appears displaying the first set of stress data.



**Figure 3-21** A “snapshot” frame of the animated shock wave displacement data using the WzSurface Tool.

**Step 3** In the WzSurface Tool, select **File=>Export As Pixmap**. The Export Pixmap dialog box appears. Note that the name ANIMATE\_0 appears in the Pixmap Variable Name text field, and the Add to Pixmap Variable button is selected.

**Step 4** Click Apply.

At this point, the plot in the WzSurface display area is saved as a pixmap in the variable called ANIMATE\_0.

**Step 5** Select STR02 through STR06 in the WzVariable list. To select multiple elements of a list, click the left mouse button on the first element, and click <Shift>-<left mouse button> on each subsequent element.

**Step 6** Select **File=>Export Variables** in the WzVariable Tool.

**Step 7** In the Variables Export dialog box, select STR02 in the upper list box and make sure that WzSurface\_0 is selected in the lower list box.

**Step 8** Click Apply in the Variables Export dialog box. This exports the variable STR02 to the WzSurface Tool, which displays the variable in its view area.

**Step 9** Click the Apply button in the Export Pixmap dialog box. Now, you have stored a second pixmap in the variable ANIMATE\_0.

**Step 10** Repeat steps 7, 8, and 9 four more times, each time selecting the next variable, until STR03, STR04, STR05, and STR06 are stored in ANIMATE\_0.

**Step 11** When you have saved all variables, close the Export Pixmap dialog box by clicking the Cancel button.

**Step 12** Close the Variables Export dialog box by clicking the Cancel button.

**Step 13** Select **Options=>Redisplay List** in the WzVariable Tool. The variable ANIMATE\_0 appears in the list.

**Step 14** Double-click on the variable name ANIMATE\_0 in the WzVariable list.

The Variable Information dialog box confirms that this is a 3D variable (638-by-510-by-6). The first two dimensions hold the image data, and the third dimension represents the number of individual pixmaps in the animation sequence. For example, if the dimensions of a variable were 512-by-512-by-30, the variable, when viewed in the WzAnimate Tool, would produce a loop or cycle consisting of 30 frames.

**Step 15** Click OK to dismiss the Variable Information dialog box.

**Step 16** Close the WzSurface Tool.

You have now stored a six-frame animation sequence. You are ready to view the animation sequence.

---

## Viewing Animation

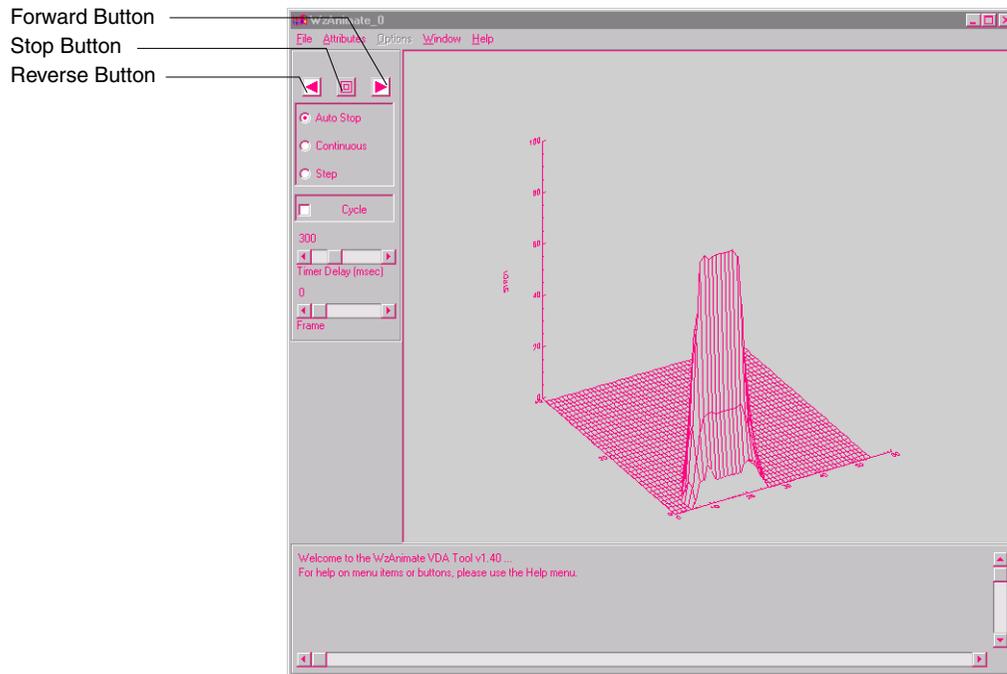
Now view the animation to see how the shock wave develops.

Since you did not define an animation array name, the array was given the default name, ANIMATE\_0.

**Step 1** In the WzVariable Tool, select ANIMATE\_0 from the list. Ensure that all other variables are deselected.

**Step 2** Click the WzAnimate icon in the Navigator window.

The WzAnimate Tool appears.



**Figure 3-22** Using the WzAnimate Tool to view successive data “snapshots”.

Now you’re ready to play the animation in a variety of ways:

**Step 3** Select the Continuous button in the controls area.

**Step 4** Select the Forward button to run the animation frames forward.

**Step 5** When you are finished viewing the animation, select the Stop button and then close the WzAnimate Tool.

---

## ***Changing the Appearance of a Plot***

After a plot is created, you can change its appearance and annotate it. For example, you may want to do this before showing the plot to someone else, before printing it for your files, or to prepare an illustration for a publication.

For this example, you can use a surface view of STR14. You'll change the colors of parts of the plot, select a font to use for a title you're giving the plot, and place the title on the plot.

- Step 1** Select STR14 from the WzVariable list. Make sure that all other variables are deselected.
- Step 2** Click the WzSurface icon in the Navigator window.
- Step 3** Select **Attributes=>Surface Attributes**. The Surface Attributes dialog box appears.

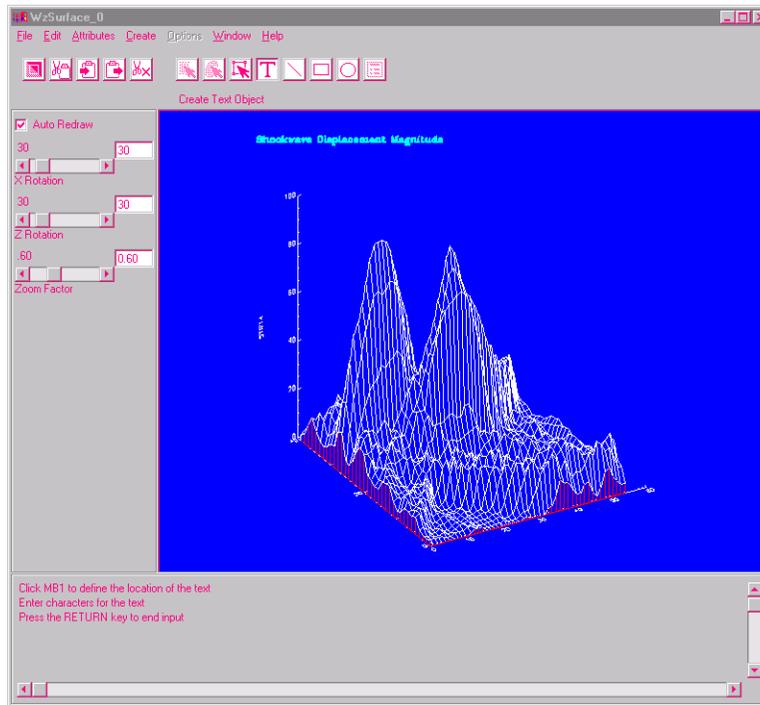
This dialog box offers a variety of options to enhance the appearance of a plot. We'll suggest a particular set of attributes, but you may want to experiment using your own attributes.

- Step 4** Click the Upper Color button and select the color white from the color bar, then click Dismiss. This choice will make the top of the surface white (color 1).
- Step 5** Click the Lower Color button and select the color red from the color bar, then click Dismiss. This choice will make the base (underside) of the surface red (color 3).
- Step 6** Select the Skirt checkbox to highlight the edge of the surface with a skirt.
- Step 7** Select the OK button of the Surface Attributes dialog box to apply the attributes and exit the dialog box.
- Step 8** Select **Attributes=>View Attributes**.
- Step 9** Click the Background Color button and select the color blue (color 5) from the color bar, then click Dismiss.
- Step 10** Click OK in the View Attributes dialog box.

Now add a title to your plot.

- Step 11** Select **Attributes=>Defaults=>Text Object**. The Default Text Attributes dialog box appears.
- Step 12** Click the Color button and select the color cyan (color 6) from the color bar, then click Dismiss.
- Step 13** Select Complex Roman from the Font option menu.
- Step 14** Enter 2 in the Size field and 2 in the Thickness field.
- Step 15** Click the OK button.
- Step 16** Select the Create Text Object icon on the WzSurface Tool button bar, or select **Create=>Text Object**.
- Step 17** Position the pointer near the upper-left corner of the plot and click the left mouse button. Then type *Shockwave Displacement Magnitude* in the String field and click OK.

The surface plot of STR14 now includes a title at the top of the plot.



**Figure 3-23** Changing the appearance of the plot using the Attributes menu.

---

## ***Saving and Using a Template***

Now that you have a surface plot set up with annotation, colors, and other attributes, you can save these setups as a template. Then, you can open other surface plots with the template, and those plots will contain the same color, text, and other setups.

This feature can be useful if you want to achieve a consistent look or style in your graphics output. For example, you could create a template with a custom legend in one corner and a company logo in another.

**Step 1** Select **File=>Save Template As** from the Navigator menu bar. The Save Template dialog box appears.

**Step 2** In the File Name field, enter the name of a file for the template. Give the filename a .tpl extension, for example STR14 .tpl.

**Step 3** Click OK.

Now that the template is saved, you can use it with other surface plots.

**Step 4** Close the WzSurface Tool.

**Step 5** Select **Defaults=>WzSurface** from the Navigator menu bar. The Default VDA Tool Attributes dialog box appears.

**Step 6** In the Template File Name field, enter STR14 .tpl (or whatever you named the template file). Or, you can use the BROWSE button to locate and select the file.

**Step 7** Click OK.

**Step 8** In the WzVariable Tool, select STR13.

**Step 9** Click the WzSurface icon in the Navigator window.

A new surface appears with all the same annotations and other setups that were saved in the template.

---

## ***Saving a Plot***

You can save a specific plot, along with all of its annotation and other attributes, for future use in another application, such as a desktop publishing system.

- Step 1** To save the plot, select **File=>Page Setup**. The Page Setup dialog box appears.
- Step 2** Select the Print to Metafile checkbox.
- Step 3** Click the Print button.
- Step 4** Enter a filename for the metafile and click OK. The plot is saved as an enhanced metafile.
- Step 5** Close the WzSurface Tool.

---

## ***Experimenting on Your Own***

By now, you have a good idea of how to use the Navigator and VDA Tools features of PV-WAVE: Visual Exploration.

VDA Tools provide a rich environment for doing visual data analysis. This tutorial covered only a portion of the VDA Tool capabilities. We encourage you to try out the following features. You can read more about them in the online help system:

- **Code generation** — Save the PV-WAVE code used to create a plot in a file. You can then use this code in other applications.
- **Copy and paste between VDA Tools** — You can copy graphical elements such as text, lines, and rectangles from one VDA Tool and paste them in another VDA Tool.
- **Histogram plots** — Try out the WzHistogram Tool. Use it to analyze quantitative trends in large datasets.
- **Contour plots** — The WzContour Tool displays 2D variables containing contour data.
- **Data export** — The WzExport Tool can be used to export data from a variable to a file.

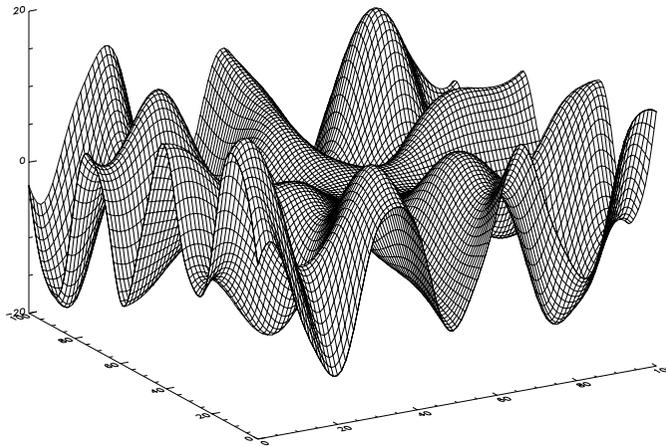
---

## ***Exiting PV-WAVE***

To exit PV-WAVE, type EXIT at the WAVE> prompt.

## *Getting Started*

In this chapter, you will learn some of the techniques that enable you to be more creative and more productive, in less time than ever before. First-time users of PV-WAVE are frequently delighted at how easily they can learn the command language, and also at how few commands are necessary to achieve complex results compared to other methods of programming.



**Figure 4-1** A surface plot that you will create in this chapter.

---

## ***Before You Begin ...***

Before you begin, make sure that PV-WAVE is installed as described in the installation instructions.

---

## ***Starting and Exiting PV-WAVE***

### **Starting PV-WAVE Under Windows NT**

**Step 1** Click the PV-WAVE Console icon in the PV-WAVE Program Group.

After a brief pause, the PV-WAVE Console window appears displaying the prompt:

```
WAVE>
```

When you see this prompt, PV-WAVE is ready for you to enter commands.

### **Starting PV-WAVE Under Windows 95**

**Step 1** Use the **Start** button. Select **Start=>Programs=>PV-WAVE 6.0=>PV-WAVE Console**

After a brief pause, the PV-WAVE Console window appears displaying the prompt:

```
WAVE>
```

When you see this prompt, PV-WAVE is ready for you to enter commands.

### **Exiting PV-WAVE**

**Step 1** Type the EXIT command to end the PV-WAVE session:

```
WAVE> EXIT
```

The PV-WAVE Console window closes.

---

## ***Brief Online Help Review***

PV-WAVE's online help system employs the standard Windows online help interface. If you are already familiar with Windows Help, you can probably skip this section. If you would like to learn more about Windows Help than is covered here, refer to your Microsoft Windows documentation.

### ***Help from the Command Line***

At the WAVE> prompt, enter:

```
WAVE>  HELP
```

This command starts PV-WAVE's online documentation system with the main Help Table of Contents displayed by default.

You can also display help on a particular PV-WAVE command. For example, for help on the REBIN command, you can type:

```
WAVE>  HELP, 'REBIN'
```

### ***VDA Tools Help***

Context sensitive help is provided with all VDA Tools. Each VDA Tool has a menu bar with a Help menu. The Help menu contains the following functions:

- **On Window** — Displays the Help viewer with the Table of Contents for information on the VDA Tool.
- **PV-WAVE Help** — Brings up the Table of Contents for PV-WAVE online help. This is the full PV-WAVE reference.
- **On Help** — Displays detailed information on how to use the Help system.
- **On Version** — Displays the PV-WAVE version number and information on electronic services.

Help is also available from VDA Tool dialog boxes. Most dialog boxes have a Help button in the lower right-hand corner. When you click this button, the online help viewer appears displaying information on the dialog box.

---

## ***Previewing This Chapter***

To preview the steps you will take and the plots you will create in this chapter, run the batch file named below.

**Step 1** Start PV-WAVE as explained previously in this chapter.

**Step 2** Use the PV-WAVE CD command to move to the subdirectory that contains sample code for the tutorial. At the WAVE> prompt, enter:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code'
```

---

**NOTE** Commands typed at the WAVE> prompt are not case sensitive; however, mixed case is often used in this tutorial for greater readability.

---

**Step 3** Run the chapter preview by entering the following command at the WAVE> prompt:

```
WAVE> @lesson_3
```

The batch file `lesson_3` runs a series of PV-WAVE programs that demonstrate the plots you will produce in this chapter. The preview takes just a few minutes to complete.

---

## ***Creating and Labeling a Plot***

Line plots are the basic expression of many mathematical equations. The PLOT procedure is a rapid method of displaying data that has been manipulated with one or more mathematical operations and routines.

---

**TIP** For best results, position the Console window in the lower-right corner of the screen.

---

### **Creating the Data**

Before you can create a plot, you need data for the plot. You will create data by using the FINDGEN function, which returns a single-precision floating-point array with the specified dimensions. Each element of the array is set to the value of its one-dimensional subscript.

**Step 1** Create an 801-element vector to hold the values for  $x$ . Divide each element of the vector by 5, then subtract 80 from each element to scale the values from  $-80$  to  $80$ . Type:

```
WAVE> x = FINDGEN(801)/5 -80
```

**Step 2** Set  $y$  equal to  $x\sin(x)$ :

```
WAVE> y = x*SIN(x)
```

## Drawing the Plot

By typing PLOT, followed by the name of the variable you wish to plot, you can easily display a line plot with PV-WAVE. The PLOT command automatically draws  $x$ - and  $y$ -axes and places tick marks on them.

---

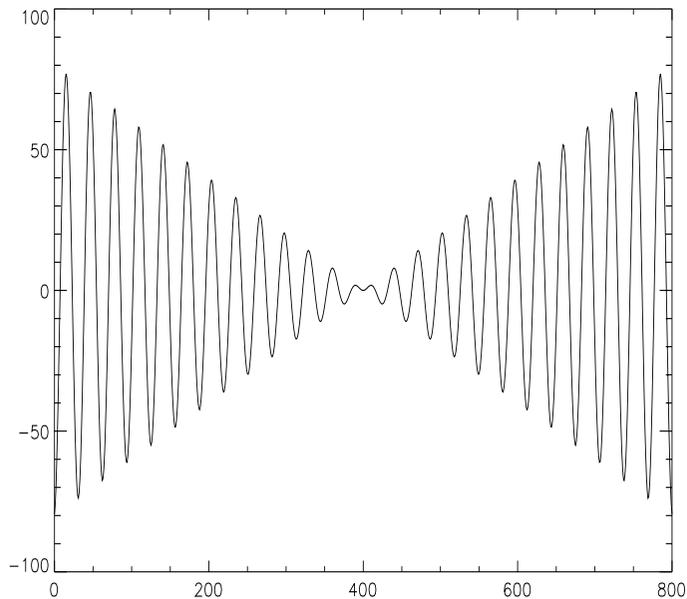
**NOTE** Use a comma (,) to separate arguments in a command.

---

**Step 3** Type the command:

```
WAVE> PLOT, y
```

A plot of the indices versus function appears.



**Figure 4-2** The generated plot of the indices versus function.

The plot above is shown with black lines and white background, which is the reverse of what appears on your monitor. Reverse printing is often used in this tutorial to improve readability.

## Using Keywords

Keywords are an easy means to add optional parameters to commands, providing variety, power, and flexibility. For example, you can create windows with the WINDOW command. You can modify the size, position, title bar text, and colors used in a window by using keywords for the WINDOW command.

---

**NOTE** PV-WAVE assigns an index number between 0 and 31 to a window if you do not. The initial default window index number, printed across the title bar, is 0. The default window size is 640 pixels wide by 512 pixels in height.

---

A single keyword, such as *Title*, may apply to many functions, such as PLOT, SURFACE, CONTOUR, WINDOW, etc.

**Step 1** To change the size of the window, use the *XSize* and *YSize* keywords. Add a title by using the *Title* keyword.

```
WAVE> WINDOW, 0, XSize = 875, $  
    ──▶ YSize = 800, Title = 'PV-WAVE'
```

---

**NOTE** The dollar sign (\$) is used to continue a command on the next line. Whether you need to use the dollar sign (\$) depends on the width of your window. When you have room on the line, you do not need to break the command. Widen the Home window to accommodate more text, if you like.

---

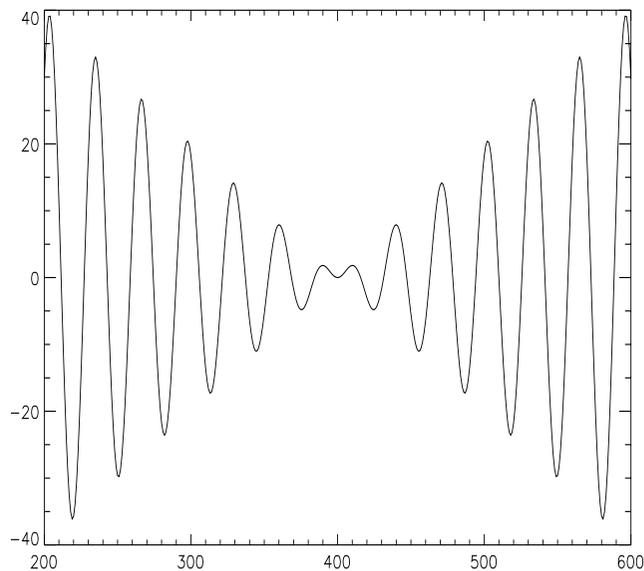
The original window is replaced by a new, empty, larger window that has the title “PV-WAVE”.

The previous plot depicted all of the data. You can explicitly define the data range using the *XRange* and *YRange* keywords.

**Step 2** Specify the data range:

```
WAVE> PLOT, y, XRange = [200, 600], $  
    ──▶ YRange = [-40, 40]
```

The new plot contains a single line with *x* values ranging from 200 to 600.



**Figure 4-3** The plot modified using specific keywords.

## Obtaining Information About Your Data

You can use the `INFO` and the `PRINT` commands to display information about variables.

**Step 1** To determine the type and dimensions of your variables, enter:

```
WAVE> INFO
```

`PV-WAVE` returns a list of your variables and of saved procedures and functions. Note that  $x$  and  $y$  are listed as floating-point arrays and the dimensions of each are given. If you ran the chapter preview, several saved variables, procedures, and functions also are listed.

You can print any `PV-WAVE` data variables to the screen; they are displayed as ASCII characters.

**Step 2** Print each value of the array variable  $x$  to the screen:

```
WAVE> PRINT, x
```

All values of  $x$  are displayed in the Home window.

**Step 3** Print the values in *y*:

```
WAVE> PRINT, y
```

To print a particular value, specify the position of the value in the array.

**Step 4** Print the 604th value of *y*, enter:

```
WAVE> PRINT, y(603)
```

---

**NOTE** Arrays are indexed from zero, so this must be taken into account.

---

**Step 5** To print the range of third through the sixteenth values, enter:

```
WAVE> PRINT, y(2:15)
```

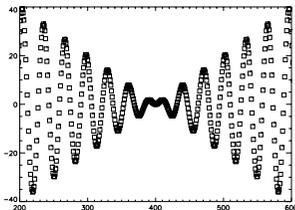
## Adding Symbols

PV-WAVE offers eight different types of plotting symbols that you can use to mark the points.

**Step 1** For square symbols, type:

```
WAVE> PLOT, y, XRange = [200, 600], $  
    ↵ YRange = [-40, 40], PSym = 6
```

The points are plotted, using squares for the symbols.



The *PSym* keyword tells PV-WAVE which type plotting symbol to use. In this case, a value of 6 produces squares.

Use the up arrow (↑) to recall previously typed commands. Once a command is recalled to the command line, you can edit it or add to it and then execute it.

Setting *PSym* equal to 6 did not draw the connections between the points. Assigning a negative value to the symbol number associated with *PSym*, i.e., adding a

single character, the minus sign ( $-$ ), is all that is necessary to specify that points be connected with lines.

**Step 2** To connect the symbols, and to use a triangle symbol, type:

```
WAVE> PLOT, y, XRange = [200, 600], $
      ┆ YRange = [-40, 40], PSym = -5
```

The triangles representing data points are connected by a solid line.

## Adding Another Line to the Plot

You can plot more than one line on a single plot. The OPLLOT command overplots data on the same axes as an existing plot. More than 75 keywords can be used with this function.

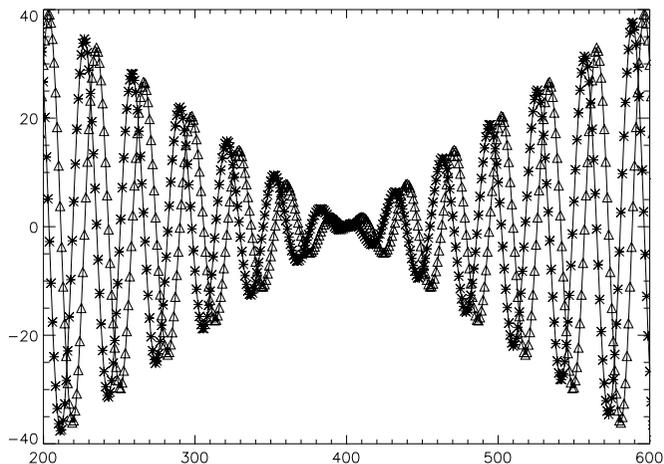
**Step 1** First, create another data set,  $w$ , which is a cosine variation of  $y$ .

```
WAVE> w = x * COS (x)
```

**Step 2** Plot  $w$  over the existing plot of  $y$ .

```
WAVE> OPLLOT, w, PSym = -2
```

The new line is plotted with the same axis range as the first line. The “asterisk” symbol is specified with the keyword  $PSym = -2$ .



**Figure 4-4** Using symbols to differentiate multiple plots in a display.

## Working with Colors

Your plot is in black and white because the default color table is a gray-scale. There are 16 ready-made color tables from which to choose. The default, Black and White Linear, is color table number 0. The other numbers are in the range 1–15.

**Step 1** You can determine the color indices and manipulate colors in the color table. Type:

```
WAVE> LOADCT, 5
```

The *Standard Gamma II* color table is loaded.

To specify a specific color, you can refer to the color's index number.

**Step 2** Display the numbered colors.

```
WAVE> COLOR_PALETTE
```

The COLOR\_PALETTE window appears, displaying alternate colors (of a possible 128-color palette) with the index printed on each. Keep the COLOR\_PALETTE window open for later use.

---

**TIP** To exit the COLOR\_PALETTE window, use the window Control menu or use the command to delete the window that is stated later in this chapter.

---

**Step 3** To see how many colors are available, enter:

```
WAVE> INFO, !D.N_Colors
```

The number of available colors is returned.

The total number of available colors will vary because other programs allocate colors also, but PV-WAVE distributes the colors of a color table across the available range. !D.N\_Colors is a system variable that is also useful in writing code for colors, as shown in the next example.

**System variables** are a special class of predefined variables, each having a predefined type and structure that cannot be changed. Their names always begin with an exclamation mark (!).

**Step 4** Now, plot the lines in the color that has an index number equal to the total available minus 40 (yellow) and make the background red (index 46). Type:

```
WAVE> PLOT, y, XRange = [200, 600], $
```

```

    ► YRange = [-40, 40], PSym = -5, $
    ► Color = !D.N_Colors - 40, Background=46

```

The keyword **Color** specifies the number of the color to use for drawing. **!D.N\_Colors** contains the total number of colors currently available. **Background** specifies the color number of the background.

Yellow is used to plot the lines on the blue background.

What if only approximately 100 colors are available? The previous commands might not produce the desired colors. In some cases, it may be better to specify a percentage of the total number of available colors.

The default linestyle, 0, is a solid line. Five other styles of dotted and dashed lines are available. For example, linestyle 4 specifies a line composed of long-short-short dashes. The following table lists the available linestyles and their index numbers:

Index	Linestyle
0	Solid
1	Short dashes
2	Long dashes
3	Long-short dashes
4	Long-short-short dashes
5	Long dashes

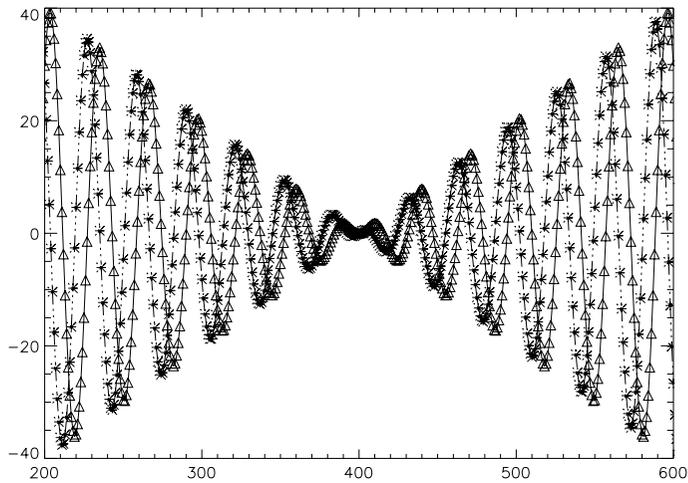
**Step 5** Overplot the other set of data in orange by specifying a color index that is 60 percent of the total number and draw it in a different linestyle.

```

WAVE> O PLOT, w, XRange = [200, 600], $
    ► YRange = [-40, 40], PSym = -2, $
    ► Color = !D.N_Colors*.6, LineStyle = 4

```

The orange and yellow lines are plotted against a black background.



**Figure 4-5** Using symbols, linestyles, and colors to differentiate multiple plots in a display.

Many more color techniques will be explored in later chapters.

## Recalling Previous Commands

To recall the line you typed in step 1, or any of the last 20 lines of text you typed at the `WAVE>` prompt, use the following:

**Step 1** To display your last line of input, press the up ( $\uparrow$ ) arrow key once. `PV-WAVE` returns the last line you typed.

**Step 2** Press the up ( $\uparrow$ ) arrow key again to recall previous lines.

You can edit, erase, copy and re-enter any of these lines.

**Step 3** To review as many as the last 20 lines you input, type:

```
WAVE> INFO, /Recall_Commands
```

Up to 20 lines are displayed from the input buffer.

Using this feature, you can easily correct errors and you rarely need to retype an entire command.

**Step 4** Press the down ( $\downarrow$ ) arrow key until there is no longer any text at the prompt.

## Adding Labels to the Plot

You can place labels on the  $x$ - and  $y$ -axes, as well as a title and subtitle on your plot. You may use the default font, Simplex Roman, or select from any of 15 other software fonts.

### *Using Software Fonts*

PV-WAVE can produce text output using either *software* or *hardware* fonts. Software fonts, sometimes called vector-drawn fonts or Hershey fonts, are internal to PV-WAVE and are drawn with line vectors. Hardware fonts are built into specific output devices, such as PostScript printers and window systems. PV-WAVE simply sends the characters to the graphics device, which displays them using these fonts.

In this chapter, you will work with software fonts. For more information on using fonts, see the *PV-WAVE User's Guide*.

The software fonts can be rotated in plots. The font character sets include serif, non-serif, script, regular and oblique English, as well as Greek, math and special symbols, and several others.

**Step 1** If you have not already done so, use PV-WAVE's CD command to move to the subdirectory that contains sample code for the tutorial. At the WAVE> prompt, enter:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code'
```

**Step 2** The code subdirectory contains a program to display all the fonts. To see what each font looks like, enter:

```
WAVE> @showfonts
```

Each character of each of the fonts is displayed alongside its corresponding keyboard key. Select Close from the Control menu of the graphics window after all the fonts have been displayed.

Use the Hershey triplex Roman font for the titles by placing “!17” before the text in the quoted string. In order to avoid having an unwanted space at the beginning of your title, do not add a space between the font number (17) and the first character of the title, which for the next plot is *P*.

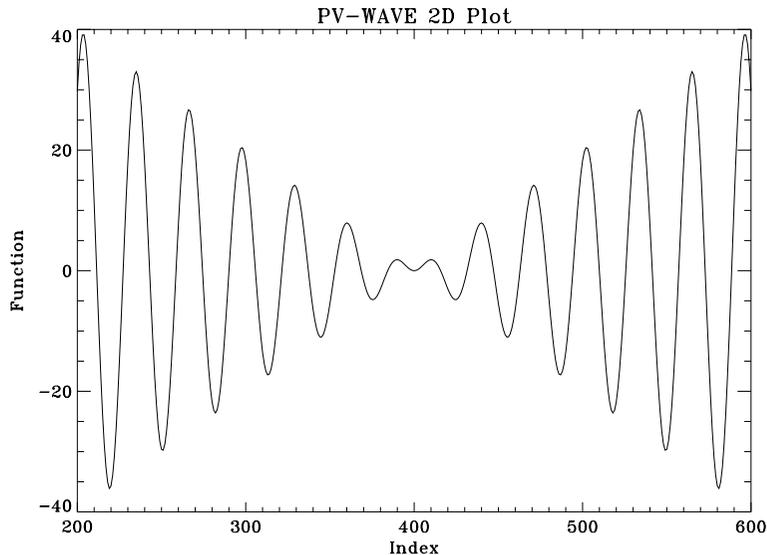
## Adding Titles

By default, the *Title* is automatically made to be 1.25 times as large as the titles on the *x*- and *y*-axes.

**Step 1** Add titles to the *x*- and *y*-axes and a title to the top of the plot.

```
WAVE> PLOT, y, XRange = [200, 600], $
      ► YRange = [-40, 40], $
      ► Color = !D.N_Colors - 40, Back = 46, $
      ► XTitle = 'Index', YTitle='Function', $
      ► Title = '!17PV-WAVE 2D Plot'
```

The designated titles are added to your plot.



**Figure 4-6** Annotating the plot using specific keywords.

Plot annotation — titles, subtitles, and axis labels — are drawn in a specific order. Text characteristics such as font and text size are inherited from items that are drawn first. For example, a plot subtitle inherits its characteristics from the main title, and so on. The order of plot annotation is:

1. Main title
2. Subtitle
3. X axis numbers
4. X axis title
5. Y axis numbers
6. Y axis title
7. Z axis numbers
8. Z axis title

Font specification remains in effect until another font is specified. If you wanted to have all titles in font 17 and all axis numbers in the default font (3), you would need to enter:

```
XTitle = '!17Index!3', $  
YTitle = '!17Function!3', $  
Title = '!17PV-WAVE 2D Plot!3', $
```

Adding the “!3” at the end of the text string drawn last (the *Title* string in this case) causes the default font to be used thereafter, in this and in subsequent plots.

## Adding Today’s Date to the Plot

Information relating to dates and times is essential to many types of data processing. PV-WAVE has many ways to incorporate and manipulate calendar-based information.

**Step 1** Create the variable *a* to hold today’s date by using the TODAY function.

```
WAVE> a = TODAY()
```

**Step 2** Print the contents of *a*.

```
WAVE> DT_PRINT, a
```

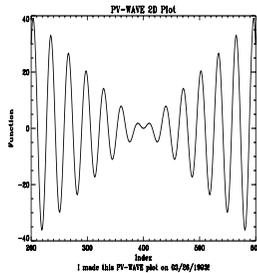
DT\_PRINT prints the contents of a date/time variable in a readable format. Compare DT\_PRINT with the PRINT procedure, which can also be used to print the contents of a variable.

**Step 3** Convert the variable to a character string and specify that it prints only the day, month, and year.

```
WAVE> DT_TO_STR, a, b, Date_Fmt = 1
```

The keyword `Date_Fmt` specifies the format for the resulting string variable. Format 1 is “mm/dd/yyyy”. For example, May 1, 1993 is represented by 05/01/1993.

**Step 4** Add a subtitle that incorporates today’s date into a sentence.



```
WAVE> PLOT, y, XRange = [200, 600], $
      ► YRange = [-40, 40], $
      ► Color = !D.N_Colors - 40, Back = 46, $
      ► XTitle = 'Index', YTitle='Function', $
      ► Title = 'PV-WAVE 2D Plot', $
      ► SubTitle='I made this PV-WAVE plot ' $
      ► + 'on ' + b + '!!!'
```

Today’s date is placed within the character string in the subtitle.

---

**NOTE** The plus (+) character is used to concatenate strings. For formatting purposes, it is sometimes necessary to use the plus (+) character to break a string and continue it on the next line.

---

---

**TIP** The font is Hershey Triplex Roman, font 17, which was inherited from the last plot. To place an exclamation mark (!) character in a text string, precede it with another exclamation mark, as shown in this example.

---

## Using System Variables to Retain Changes

While keywords can be used to make changes to a plot, you may want to make the changes apply to all plots. The keywords affect only the command with which they are associated; their effect does not carry over into subsequent plots. The effects of

system variables, on the other hand, remain until they are explicitly changed with a similar command. Many system variable fields have plotting keyword counterparts.

The tick marks along the  $x$ - and  $y$ -axes are small by default, to ensure that they will not be too large for small windows. The default value is .02, which represents a percentage of the size of the plot area. A value of .5 produces a grid, and negative values place the tick marks outside the plot region.

**Step 1** Make the tick marks smaller:

```
WAVE> !P.TickLen = .01
```

The bottom of the previous plot's subtitle touched the frame of your window. You can re-position the plot and associated annotation in the window by using the !Region system variable, in which you set the fractional distances for the  $XMin$ ,  $YMin$ ,  $XMax$ , and  $YMax$  parameters.

**Step 2** To re-position the plot so that there is greater margin around the labels, type:

```
WAVE> !P.Region = [.0, .04, .98, .95]
```

**Step 3** To double the size of the characters, type:

```
WAVE> !P.CharSize = 2
```

**Step 4** Increase the size of the  $x$ - and  $y$ -axis characters:

```
WAVE> !X.CharSize = .75
```

```
WAVE> !Y.CharSize = .75
```

---

**NOTE** **CharSize** indicates the desired multiplier for the default size (1.0) of your characters. **!P** is a system variable of type structure, and *Thick*, *CharSize*, *TickLen*, *Region* are fields in this structure.

---

Specifying a value of .75 makes the  $x$ - and  $y$ -axis characters 75 percent as large as those in other text strings, but they will be 150 percent the default value of 1 because 75 percent of 2 is 1.5.

**Step 5** To change the thickness of plotted lines to twice the normal value, type:

```
WAVE> !P.Thick = 2
```

**Step 6** To see the effects of the changes you just made, type:

```

WAVE> PLOT, y, XRange = [200, 600], $
    ┌─▶ YRange = [-40, 40], $
    ┌─▶ Color = !D.N_Colors - 40, Back = 46, $
    ┌─▶ XTitle = 'Index', YTitle='Function', $
    ┌─▶ Title = 'PV-WAVE 2D Plot', $
    ┌─▶ SubTitle='I made this PV-WAVE plot ' $
    ┌─▶ + 'on ' + b + '!!'

WAVE> OPLOT, w, XRange = [200, 600], $
    ┌─▶ YRange = [-40, 40], $
    ┌─▶ Color = !D.N_Colors*.6, $
    ┌─▶ LineStyle = 4

```

The line thickness and character sizes are increased in the plot, and the tick marks are shorter.

## Placing Labels Within the Plot

The plot would be more informative if it had a legend, such as “Sine” and “Cosine”, to distinguish the two lines. The XYOUTS procedure can draw text at any designated position in the window or currently selected graphics device.

**Step 1** Place a text string at a position (in data coordinates) that begins at  $x = 300$  and  $y = -30$ :

```

WAVE> XYOUTS, 300, -30, '!17Sine', $
    ┌─▶ CharSize = 1

```

The word “Sine” is placed at the specified coordinates.

**Step 2** Overplot a horizontal line from  $x = 350$  to  $x = 500$  to represent the first function:

```

WAVE> OPLOT, [350, 500], [-30, -30], $
    ┌─▶ LineStyle = 0, Thick = 2, $
    ┌─▶ Color = !D.N_Colors - 40

```

A yellow line is drawn next to “Sine.”

**Step 3** Place a text string at a position that begins at  $x = 300$  and  $y = -35$ .

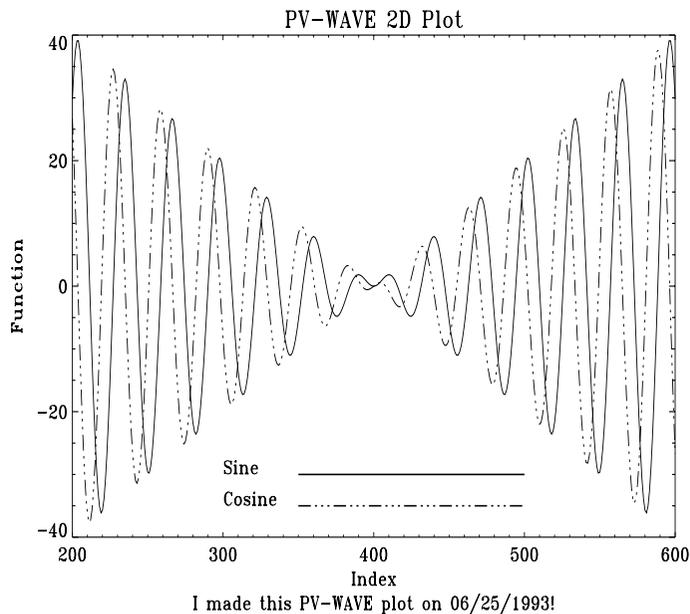
```
WAVE> XYOUTS, 300, -35, '!17Cosine', $
      CharSize = 1
```

The word “Cosine” is placed at the specified coordinates.

**Step 4** Overplot a horizontal line from  $x = 350$  to  $x = 500$  to represent the second function.

```
WAVE> OPLOT, [350, 500], [-35, -35], $
      LineStyle = 4, Thick = 2, $
      Color = !D.N_Colors*.6
```

An orange line is drawn next to “Cosine.”



**Figure 4-7** Placing labels inside a plot using keywords.

## Resetting System Variables

Most system variables that are set with numeric values can be set back to the default value by setting them to zero or one. The `prep_plot` procedure in the directory:

```
<maindir>\docs\tutorial\code
```

(where <maindir> is the main directory where PV-WAVE is installed) resets system variables to their defaults. The `prep_plot` routine also does several other things, such as setting the device to WIN32 if it is not currently set to WIN32 and reloading the default color table.

The following steps demonstrate how to return a system variable to its default value without affecting any others (which you can do when you do not want to return all system variables to the default values.) Simply reassign the variables to their default values.

**Step 1** Return the system variables to the default values. Type:

```
WAVE> !P.TickLen = 0.02
WAVE> !P.Region = [0.0, 0.0, 1.0, 1.0]
WAVE> !P.CharSize = 1
WAVE> !X.CharSize = 1
WAVE> !Y.CharSize = 1
WAVE> !P.Thick = 1
```

Subsequent plots will be displayed with the default values.

---

## ***Creating Surfaces and Contours***

In this exercise, you will create a 3D surface of a function and display it in several different ways. You will create seven windows, each showing a different view of the data.

To demonstrate that PV-WAVE can multiply arrays as easily as integers, this example plots the function:

$$f(x, y) = x\sin(y) + y\cos(x) - \sin(.25xy)$$

**Step 1** Create an array  $x$  and set  $x$  equal to  $y$ . Store the result in the variable  $z$ . Enter:

```
WAVE> x = FINDGEN(101)/5 -10
```

**Step 2** Set  $y$  equal to  $x$ :

```
WAVE> y = x
```

**Step 3** Next, create an empty 101-by-101-element array called  $z$  by using the `FLTARR` function:

```
WAVE> z = FLTARR(101, 101)
```

**Step 4** Fill the array with the values:

```
WAVE> FOR i = 0, 100 DO BEGIN & $
    ┌─▶ z(i, *) = x(i)*SIN(y) + $
    ┌─▶ y*COS(x(i)) - SIN(0.25*x(i)*y)
```

---

**NOTE** The **ampersand** (&) allows you to enter two commands on the same line.

---

Next, resize your current window, which is window 0.

**Step 5** To resize the window, type:

```
WAVE> WINDOW, 0, XSize = 400, YSize = 400
```

The window is resized to 400-by-400 pixels.

**Step 6** Draw a surface. Type:

```
WAVE> SURFACE, z
```

A wire-frame surface of the data is displayed. (See [Figure 4-1](#) on page 63.)

The current color table displays the lines in white. **LOADCT** tells PV-WAVE to load a specified color table.

**Step 7** Change to the *Blue-Green-Red-Yellow* color table by typing:

```
WAVE> LOADCT, 4
```

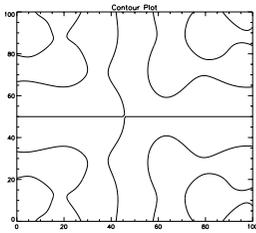
The lines change to yellow.

**Step 8** Create a new window:

```
WAVE> WINDOW, 1, XSize = 350, YSize = 350
```

The window appears in the lower-right corner of the screen.

**Step 9** Draw a contour plot of  $z$  and add a title.



```
WAVE> CONTOUR, z, Title = '!3Contour Plot'
```

A 2D contour of  $z$  is displayed with a title above it.

**Step 10** Create a new window:

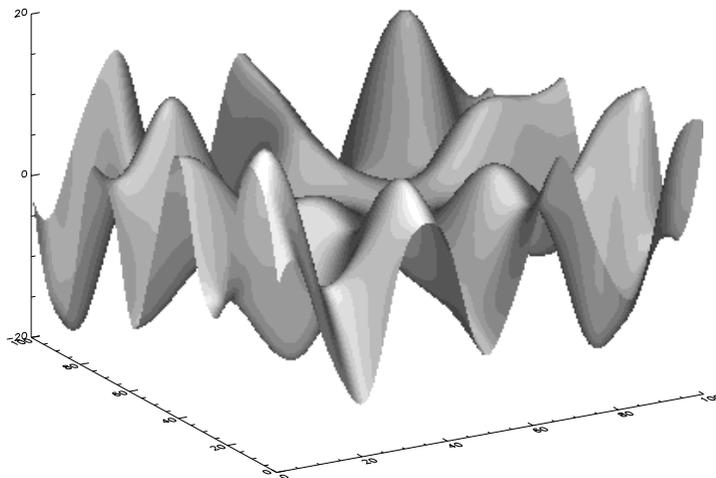
```
WAVE> WINDOW, 2, XSize = 400, YSize = 400
```

The new window appears in the upper-left corner of the screen.

**Step 11** Draw a shaded surface of  $z$ :

```
WAVE> SHADE_SURF, z
```

The data is displayed as a light-source shaded surface.



**Figure 4-8** The resulting light-source shaded surface.

**Step 12** Create a new window.

```
WAVE> WINDOW, 3, XSize = 350, YSize = 350
```

A window appears in the lower-left corner of the screen.

Display the same data set as an image. The range of data values in a 101-by-101 image is better visualized by replicating the data to fit into your 350-by-350 window (using **CONGRID**). You can simultaneously scale the values in the image to the number of available colors by using **BYTSCL**.

---

**NOTE** By default, **BYTSCL** scales the values in the image into the range of 0 – 255. The *Top* keyword can be used to constrain the upper limit of this range. **CONGRID** re-samples the image to a specified dimension. In this case, it makes a 350-by-350 pixel replication of a 101-by-101 image.

---

**Step 13** Scale the data by typing:

```
WAVE> z_img = BYTSCL(CONGRID(z, 350, 350), $
      top = !D.N_Colors-1)
```

---

**TIP** A quick way to display the image is to use the **TV** procedure. The **TV** procedure is only one of several procedures that can be used to display images. **TV** displays images on the image display without scaling the intensity.

---

**Step 14** Display the data:

```
WAVE> TV, z_img
```

PV-WAVE displays the image.

Notice that the pixel replication makes a jagged image. You can smooth it using image processing and re-display it in the same window.

**SMOOTH** returns a copy of the array smoothed with a boxcar average of the specified width. The result is of the same size and dimension as the array.

**Step 15** To smooth and re-display the image, type:

```
WAVE> smooth_z = SMOOTH(z_img, 5)
```

```
WAVE> TV, smooth_z
```

A smoothed version of the image is displayed.

---

**NOTE Standard Library** procedures are executable text files that you can read, copy, and edit. All have the `.pro` extension and reside in the subdirectory `<maindir>\wave\lib\std` subdirectory, where `<maindir>` is the main directory where PV-WAVE is installed.

---

You can combine graphics and images to get even more information from your data. Use the Standard Library procedure, `IMAGE_CONT`, to display a contour over the image you made.

**Step 16** Open a new window, placing it to the right of window number 3 by typing:

```
WAVE> WINDOW, 4, XSize = 350, YSize = 350,$
      ┌─▶ XPos = 360, YPos = 0
```

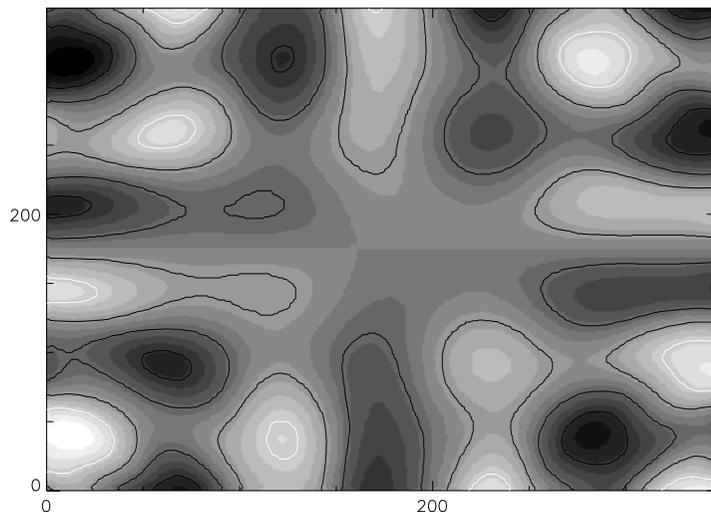
**XPos** and **YPos** specify the  $x$  and  $y$  positions of the lower-left corner of the window, specified in device coordinates.

Window 4 appears to the right of window 3.

**Step 17** Display the image with a contour plot over it:

```
WAVE> IMAGE_CONT, smooth_z
```

The image is drawn in window 4 with a contour plot over it.



**Figure 4-9** The image plot overlaid with a contour plot.

A row profile is a plot of the pixel intensities or values in a particular row of the image. The PROFILES procedure enables you to dynamically select a row or column in a displayed array and plot a profile of the selection.

**Step 18** Enter:

```
WAVE> PROFILES, smooth_z
```

A window titled *Profiles* appears, and the instructions for its use are printed in the PV-WAVE window.

**Step 19** Experiment with PROFILES by selecting points on the image in window 4, and then close the PROFILES window by clicking the right mouse button while the pointer is in the image window (window 4).

To make a plot suitable for printing, you can extract a row or column and plot it. For window 5, you will take a row profile of the image and plot it.

**Step 20** Create window 5 and place it above window 4 by typing:

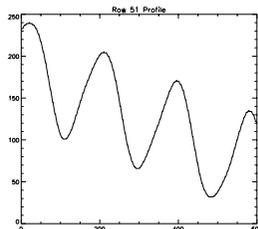
```
WAVE> WINDOW, 5, XSize = 350, YSize = 350, $
      ► Xpos = 390, YPos = 380
```

Window 5 appears above window 4.

**Step 21** Extract a row, such as row 51, out of the image, plot it in a new window and add a title. Type:

```
WAVE> PLOT, smooth_z(*, 50), Title = $
      ► 'Row 51 Profile'
```

A labeled plot of row 51 appears.



In the last window, you show  $z$  three ways in a display that combines a SURFACE plot, a CONTOUR plot, and color-scaled pixels in a single window.

**Step 22** Create window 6:

```
WAVE> WINDOW, 6, XSize = 600, YSize = 600, $
    ┆ XPos = 300, YPos = 140
```

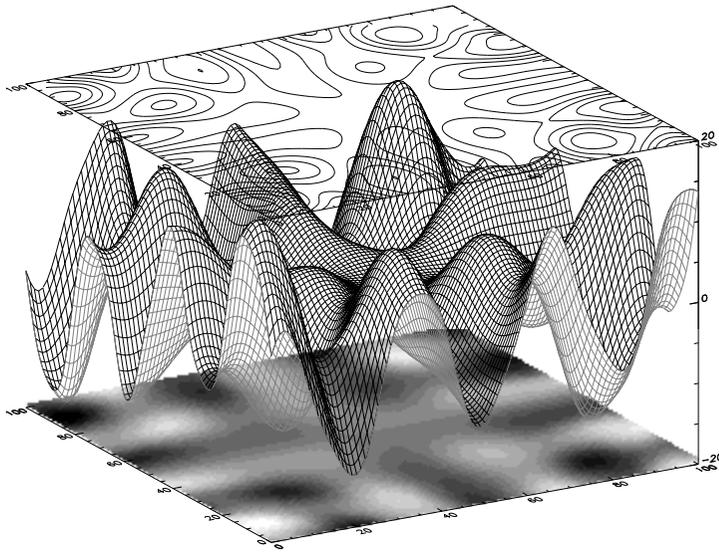
**Step 23** Display the three-part image:

```
WAVE> SHOW3, z
```

The array *z* is displayed as a surface, a contour, and as an image.

**Step 24** Consider the effects of a different color table:

```
WAVE> LOADCT, 5
```



**Figure 4-10** A plot showing 3 views of the data simultaneously: color-scaled pixels, a mesh surface plot, and a contour plot.

---

**TIP** Drag window 6 to a new position if you do not want it covering some other particular window.

---

---

## ***Close the Windows***

The WDELETE procedure deletes a specified window. You could close all the windows with the following commands:

```
WAVE> WDELETE, 0 & WDELETE, 1 & WDELETE, 2 & $
      ► WDELETE, 3 & WDELETE, 4 & WDELETE, 5 & $
      ► WDELETE, 6 & WDELETE, 31
```

It is much simpler to delete all open windows with a command that states “while any window with a window index number greater than or equal to zero is open, do WDELETE.”

**Step 25** Close all open windows. Enter:

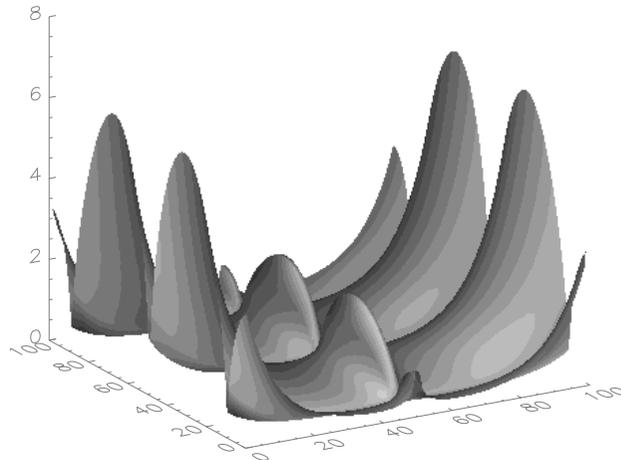
```
WAVE> WHILE (!D.WINDOW GE 0) DO WDELETE, $
      ► !D.WINDOW
```

This chapter will prove useful regardless of how you plan to use PV-WAVE. In the next chapter, you will learn more fundamental concepts that will prove useful, regardless of how you choose to use PV-WAVE in the future.



## ***Basic Information for Any Session***

This chapter presents basic information that will find useful during any PV-WAVE session.



**Figure 5-1** PV-WAVE *Advantage* functions Gamma and BessJ are used to define a shaded surface. The procedure for creating this plot is presented in [Chapter 9, \*Advanced Math and Stats\*](#).

---

## ***Getting Help with PV-WAVE***

You can, at any time, find assistance within PV-WAVE.

PV-WAVE is equipped with extensive online help facilities that include:

- Windows Help
- The INFO command

### **Using Online Documentation**

PV-WAVE's online help system provides detailed information on the following language and graphics topics:

- ✓ Alphabetical Listing of Routines
- ✓ Functional Listing of Routines
- ✓ Graphics and plotting keywords
- ✓ Device keywords
- ✓ System variables
- ✓ Operators

If you purchase the optional Math Toolkit and Statistics Toolkit, additional online help is provided with information on the functions and procedures of the IMSL Mathematics and Statistical Library routines.

### **Using the INFO Command**

The INFO procedure gives information about the PV-WAVE session.

#### ***Getting Information About the Session***

Typing INFO, with no additional parameters, displays an overview of the current session, including one-line descriptions of all variables and the names of all compiled procedures and functions.

**Step 1** To obtain general information about the session, enter:

```
WAVE> INFO
```

PV-WAVE returns information about variables, saved procedures and functions, area used for code and data, etc.

## ***Getting Information on Specific Variables***

To display information about a variable's type and structure, add a comma (,) and the name of the variable. This makes the variable an argument to the INFO command, as is demonstrated next.

If you started PV-WAVE at the beginning of this chapter, you have no user-defined variables.

**Step 1** Define a variable to use as an example. Create a 6-element array named *vector*. Type:

```
WAVE> vector = [1, 2, 3, 4, 5, 6]
```

**Step 2** Display information on *vector*. Enter:

```
WAVE> INFO, vector
```

PV-WAVE returns:

```
vector      int = Array(6)
```

indicating *vector* is a 6-element integer array.

**Step 3** View both *vector* and the tripled value of each element, type:

```
WAVE> PRINT, vector, 3*vector
```

PV-WAVE returns

```
1  2  3  4  5  6
```

```
3  6  9 12 15 18
```

## ***Information About System Variables***

You can rapidly determine the settings of your system variables.

**Step 1** To print the values of system variables, type:

```
WAVE> INFO, /System_Variables
```

The values of the system variables are returned to the screen.

**Step 2** To print the value of a single variable, you can use the PRINT command. To print the value of the !Pi system variable, type:

```
WAVE> PRINT, !Pi
```

```
3.14159
```

## ***Changing a System Variable***

You can change the values of system variables within PV-WAVE. For example:

**Step 1** Change the WAVE prompt. Type:

```
WAVE> !Prompt = "Windows WAVE> "
```

Your prompt is now Windows WAVE>.

When you change a system variable, the change will remain in effect during the session unless it is explicitly changed.

## **Looking at PV-WAVE Source Files**

The following directories contain many examples of programs and applications written in PV-WAVE command language (where <maindir> is the name of the main product directory):

- <maindir>\docs\tutorial\code
- <maindir>\wave\lib\std
- <maindir>\wave\lib\user

As you begin to develop your own applications, you are free to examine these files and whenever possible use the code in your programs.

---

## ***Exiting, Interrupting, and Aborting the Software***

### **Exiting PV-WAVE**

Entering an EXIT or QUIT command at the WAVE> prompt causes PV-WAVE to exit unconditionally, and you are returned to the operating system prompt.

---

**CAUTION** When you exit unconditionally, variable assignments are lost; however, data that is buffered for open output files is flushed to these files before exiting is complete. Customizations you make to PV-WAVE, such as changing the font used in the windows, or enabling MDI graphics, are saved in the file WAVE.INI and are restored the next time you start the software.

---

## Interrupting or Aborting PV-WAVE

This section describes individual characters that can be entered in conjunction with the <Control> key to interrupt or stop PV-WAVE. These characters are summarized in the following table.

### Control Characters Used for Interrupting or Stopping PV-WAVE

Character	Action
<Control>-C	Keyboard interrupt; enter .CON to continue.
<Control>-D	Signifies EOF; causes PV-WAVE to exit.
<Control>-Break	Abort.

---

**NOTE** The control characters listed in the previous table are only recognized in the Console window. Control characters are not recognized in any auxiliary windows, such as graphics windows. (Exception: <Control>-C is a standard Windows accelerator, so it is recognized in any window where a Copy to Clipboard operation is meaningful.)

---

---

## Restoring “Lost” Variables

When a procedure or function is interrupted, control of PV-WAVE may stop in the procedure or function instead of returning to the main program level. This transfer of control to the procedure or function level can make variables defined at the main program level temporarily unavailable. Use the RETALL command to return to the main program level.

Entering the command RETALL (*Return All*) returns PV-WAVE’s context to the main program level, causing the variables to “reappear”. Alternately, you can type the command RETURN whenever you wish to return PV-WAVE’s context to the next highest level (in the case of nested procedures).

---

## ***Saving Session Commands and Variables***

You can save all or only some of the commands and variables you use in PV-WAVE. By saving your variables, you are prepared to exit and later restore sessions. By saving sequences of commands, you can create batch files that may be easily called both alone and in other procedures.

---

**CAUTION** Creating a new save file causes any existing file with the same name to be lost. Use the *Filename* keyword with the SAVE procedure to avoid destroying files that you want to keep.

---

### **Using the SAVE Procedure**

The SAVE procedure saves user-generated variables, system variables, and compiled user-written procedures and functions in a file, using an efficient binary format, for later recovery. Information about option choices you have made, such as colors, fonts, journaling, and so forth, is *not* saved in the session file; it is saved in the WAVE.INI file.

The variables are saved in special data files, to which you may add the extension *.dat*.

The command syntax is

```
SAVE, FileName = 'filename', /Variables
```

where *filename* is the name of the file to be saved.

Type the full pathname if you want to save the file to a different directory. If you do not specify a filename, the file is saved as *wavesave.dat*.

#### ***Example***

If you started PV-WAVE at the beginning of this chapter, the only variable you have defined is *vector*.

**Step 1** To save *vector* so that you can restore it and to receive information about what you saved, type:

```
WAVE> SAVE, FileName = 'vector.dat', $  
    ►► /Variables, /Verbose
```

PV-WAVE returns something similar to

```
% SAVE: Portable (XDR) SAVE/RESTORE file.
```

```
% SAVE: Saved variable: VECTOR
```

The **Verbose** keyword instructs PV-WAVE to print informative messages about saved objects.

## Using the RESTORE Procedure

The RESTORE procedure restores the objects previously saved in a save file when you entered the SAVE procedure at the WAVE> prompt.

If a filename is not supplied in the call to RESTORE, the filename `wave-save.dat` is used. In addition, you can use keywords with RESTORE.

### Example

**Step 1** To demonstrate that your session information was saved, first type:

```
WAVE> EXIT
```

**Step 2** Now restart PV-WAVE by clicking on the PV-WAVE icon in the Win32 Applications window.

**Step 3** To restore your saved session, type:

```
WAVE> RESTORE, 'vector.dat', /Verbose
```

The session information is restored and PV-WAVE returns:

```
% RESTORE: Portable (XDR) SAVE/RESTORE file.
```

```
% RESTORE: Save file written by yourID@your  
computer and the date
```

```
% RESTORE: Restored variable: VECTOR.
```

**Step 4** To see *vector*, type:

```
WAVE> PRINT, vector
```

PV-WAVE returns:

```
1  2  3  4  5  6
```

---

## Using Data from Files

Unlike many data visualization products, which require data in specific formats, PV-WAVE is able to read and write data files in virtually any data format, using generalized I/O routines, and thus provides unparalleled access to collected data. PV-WAVE can read both free-formatted and user-formatted data, as well as unformatted or binary data.

XDR (External Data Representation) binary data is supported to encourage data portability across machine architectures. Procedures also are available to read and write 8-bit TIFF files, Device Independent Bitmaps (DIB files), and Windows metafiles (WMF files).

This tutorial provides a number of examples of reading data. However, for complete information about how to read your data into PV-WAVE and how to write data to files, see the *PV-WAVE Programmer's Guide*.

---

## What Does “Program Area Full” Mean?

Eventually, you may see a message from PV-WAVE telling you the program area is full. You can use the .SIZE system variable to change the amount of space allowed for the code area.

---

**NOTE** When a PV-WAVE program is compiled (using .RUN, for example) output is directed to two areas: the **code area** and the **data area**. The code area holds internal instruction codes and the data area holds symbols for variables, common blocks, and keywords.

---

When you type the INFO command at the WAVE prompt, you will see information about the code area used and the data area used. To increase the amount of space allowed for code area, try entering the following:

```
.SIZE 32000
```

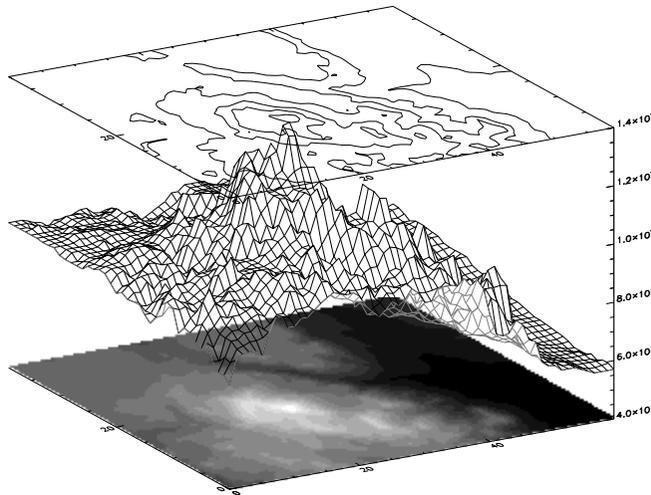
When you re-enter the INFO command, you will see the new size and the percent used.

Three functions exist that enable you to delete variables, processes, and functions; they are DELVAR, DELPROC, and DELFUNC. The latter two will accept the *All* keyword. All three should be used with caution and only after invoking the INFO command to determine what you currently have saved. You can read more about these functions in online Help.

For detailed information on the system limits, and particularly on the code and data areas, see the *PV-WAVE Programmer's Guide*.

## *Plotting Your Data*

PV-WAVE makes data plotting easy. Plots, both 2D and 3D, can be displayed with a single command, such as PLOT or SHADE\_SURF, and multiple plots can be viewed at the same time. In this chapter, you gain experience using some of PV-WAVE's plotting commands.



**Figure 6-1** A 3D plot of the Pikes Peak, Colorado region using the SHOW3 procedure.

---

## Previewing This Chapter

To preview the steps you will take and the plots you will create in this chapter, follow these steps:

**Step 1** Use the CD command to move to the PV-WAVE subdirectory that contains code for the tutorial. At the WAVE> prompt, enter:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code'
```

**Step 2** Run the chapter preview by entering the following command:

```
WAVE> @lesson_5
```

The batch file runs the programs, demonstrating the plots produced in this chapter.

---

## Reading Data from a File

One of PV-WAVE's most powerful features is its ability to read data files in virtually any file format. Your data can come, for example, from spreadsheets, from your SQL database, from other computer programs, from numerical simulations, and from physical sensors and other data collection devices. Whether your data is row-oriented or column-oriented, formatted or unformatted, PV-WAVE's general file reading routines can handle it.

Data files fall into two general categories: formatted files, which are usually human-readable ASCII files; and unformatted files, which are machine-readable binary files.

Binary files require less storage space than ASCII files and, for this reason, are often used to store large data sets containing image data. The disadvantage of binary files is that they are often not portable from one computer architecture to another. For example, you would not generally be able to read a binary file created on a computer running Windows NT if you were working on a computer using a UNIX operating system.

On the other hand, formatted files are almost always portable from one machine to another.

## File Operations and Logical Unit Numbers

The OPEN commands listed below are used to open files for input or output. Each opened file is assigned a Logical Unit Number (LUN). Other file reading and writing operations refer to the LUN rather than the filename.

```
OPENR, lun, 'filename'
```

Open a file for Reading.

```
OPENW, lun, 'filename'
```

Open a file for Writing.

```
OPENU, lun, 'filename'
```

Open a file for Updating.

There are 128 logical unit numbers, or LUNs, available for you to use. LUNs numbered 1 to 99 can be used directly in an OPEN command like those described above. LUNs numbered 100 to 128 are managed by the GET\_LUN and FREE\_LUN procedures. Once a LUN in the range of 1 to 99 is assigned to a file, it cannot be reassigned until you close the file or you exit PV-WAVE (which automatically closes the file).

To close a file and release the LUN, use the CLOSE command

```
CLOSE, lun
```

where LUN is the logical unit number assigned to the file. For example, the following OPEN command opens the file named *numbers.dat* and associates the file with logical unit number 1:

```
OPENR, 1, 'numbers.dat'
```

The command:

```
CLOSE, 1
```

closes the file and releases logical unit number 1.

You can also use one of the Data Connect commands that start with the letters “DC\_”. These commands, which enable you to read data without having to specify a LUN, are discussed in the *Contour and Surface Plotting* section of this chapter. When you use a DC command, you do not need to explicitly open and close the file.

Data input and output is discussed in detail in the *PV-WAVE Programmer's Guide*.

## Accessing Data

The unformatted binary data file used for this example, `pv_wave.spd`, is a digitized speech sample of the name “PV-WAVE”. First, define the initial variables describing the data, and then open a data set to display.

**Step 1** Specify the size of the sample and store it in a variable called *original*:

```
WAVE> sig_len = 7519
```

```
WAVE> original = INTARR(sig_len)
```

**Step 2** Create the string variable *sig\_file* to contain the name and path to the data:

```
WAVE> sig_file = !Data_Dir + 'pv_wave.spd'
```

Normally, binary data is not portable between different machine architectures because of the way different machines represent binary data. XDR (External Data Representation, developed by Sun Microsystems, Inc.) converts between the machine’s internal and a standard external binary representation for data. XDR is supported to encourage data portability across machine architectures. The *Xdr* keyword is used to access XDR files. `OPENW` and `OPENU` normally open files for both input and output. However, XDR files can be open in only one direction at a time.

**Step 3** Open the file and read the data into the variable named *original*:

```
WAVE> OPENR, 1, sig_file, /xdr
```

```
WAVE> READU, 1, original
```

**Step 4** Display information about the file. Enter:

```
WAVE> INFO, /Files
```

This displays information about all open files, including the logical unit number assigned to each, filenames, and attributes such as read/write access.

**Step 5** Close the file:

```
WAVE> CLOSE, 1
```

**Step 6** Verify that the file is closed by re-entering:

```
WAVE> INFO, /Files
```

**Step 7** Load a color table that will display the data in color:

```
WAVE> LOADCT, 12
```

**Step 8** Convert the array to float:

```
WAVE> original = FLOAT(original)
```

---

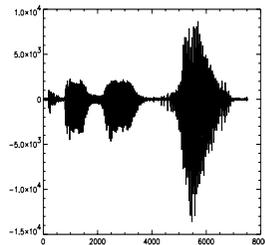
## Using 2D Plotting Capabilities

Perhaps one of the most common, and certainly one of the most useful, ways to display data is with the simple linear plot. In these exercises, you not only plot lines, you also use some of PV-WAVE's signal processing capabilities.

### Processing and Plotting Signal Data

Signal processing is used primarily to refine and clarify "real-time" sampled data. This example demonstrates some of the basic functions used in signal processing.

**Step 1** To view a quick plot of this data set, enter:



```
WAVE> PLOT, original
```

The *original* data is plotted.

**Step 2** Now add some uniformly-distributed random noise to this data set; store it in a new variable by entering:

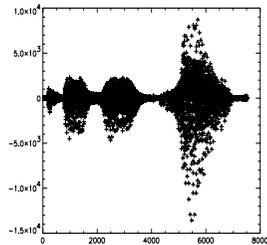
```
WAVE> noisy = original + (RANDOMU(seed, $
    sig_len)*1000) - 500
```

The RANDOMU function creates an array of uniformly distributed random values. The original data set plus the noise is stored in a new variable called *noisy*.

**Step 3** Plot *noisy* and specify a color for drawing it:

```
WAVE> PLOT, noisy, Color = 100
```

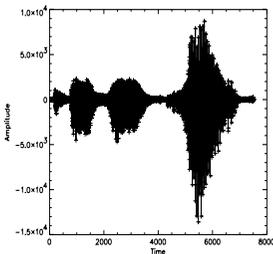
The line is now jagged.



**Step 4** Display the *original* data set and the *noisy* version simultaneously by entering the following commands:

```
WAVE> PLOT, original, XTitle = 'Time', $  
      YTitle = 'Amplitude'
```

The *original* data is plotted again, this time with titles.



```
WAVE> OPLOT, noisy, Color = 100, PSym = 1
```

The data in *noisy* is plotted as points, using crosses to mark the points.

The *XTitle* and *YTitle* keywords are used to create the *x* and *y* axis titles. The *OPLOT* command is used to plot the *noisy* data set over the existing plot of *original* without erasing.

You can use the *noisy* data set to demonstrate some of PV-WAVE's signal processing abilities.

## Using the SMOOTH Command

A simple way to smooth the *noisy* data set is to use the SMOOTH routine, which returns a copy of an array smoothed with a boxcar average of a specified width.

**Step 1** Create a new variable to hold the smoothed data set and display it by entering the following commands:

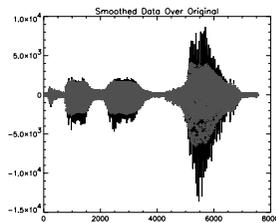
```
WAVE> smoothed = SMOOTH(noisy, 5)
```

**Step 2** Plot the smoothed data in a different color:

```
WAVE> PLOT, smoothed, PSym = 1, Color = 80
```

**Step 3** Compare the smoothed data to the original by overplotting it on the original data and add a title:

```
WAVE> PLOT, original, $  
      Title = "Smoothed Data Over Original"
```



```
WAVE> OPLOT, smoothed, PSym = 1, Color = 80
```

The *Title* keyword draws the title text centered above the plot, and the smoothed line appears less jagged than the original.

Notice that while SMOOTH did a fairly good job of removing noise spikes, the resulting amplitudes “taper off” as the frequency increases.

## Frequency Domain Filtering

Perhaps a better way to eliminate noise in the *noisy* data set is to use Fourier transform filtering techniques. Noise is basically unwanted high-frequency content in sampled data. Applying a lowpass filter to the noisy data allows low-frequency components to remain unchanged while high-frequency components are smoothed or attenuated.

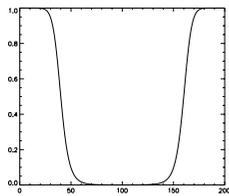
**Step 1** Construct a filter function by entering the following commands:

```
WAVE> Y = FINDGEN(200)
WAVE> Y(101:199) = - Reverse(Y(1:99))
WAVE> Filter = 1. / (1 + (Y/40)^10)
```

**Step 2** Draw a plot of the filter using a colored line twice as thick as the default:

```
WAVE> PLOT, Filter, Color=100, Thick = 2.0
```

A plot of the filter function appears.

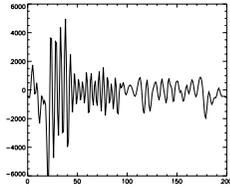


This filter is a fifth-order Butterworth filter with a cutoff of 40 cycles per total sampling period. First, the `FINDGEN` function creates a floating-point array with each element set to the value of its subscript and stores it in the variable `y`. The second command makes the last 99 elements of `y` a “mirror image” of the first 99 elements. The last command creates a variable called *Filter* that holds the filter function based on `y`.

To filter data in the frequency domain, multiply the Fourier transform of the data by the frequency response of a filter and then apply an inverse Fourier transform to return the data to the spatial domain.

**Step 3** Now you can use a lowpass filter on the *noisy* data set and store the filtered data in the variable *lowpass* by entering:

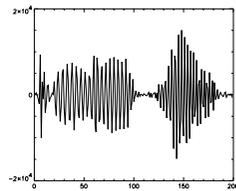
```
WAVE> lowpass = FFT(FFT(noisy, 1) $
    * Filter, -1)
WAVE> PLOT, lowpass, Color = 60
```



The variable *lowpass* is plotted.

**Step 4** Use the same filter function as a highpass filter (allowing only the high-frequency or noise components through) by entering:

```
WAVE> highpass = $
      FFT(FFT(noisy, 1)*(1.0 - Filter), -1)
WAVE> PLOT, highpass, Color = 100
```



The high-frequency components are plotted.

## Displaying the Results

Now look at all of the results at the same time. You can split the plotting window into six sections, and make each section display a different plot. The system variable !P.Multi tells PV-WAVE how many plots to put on a single page.

**Step 1** Make the window a larger rectangle:

```
WAVE> WINDOW, 0, XSize = 700, YSize = 800
```

**Step 2** To allow two columns and three rows of plots, enter:

```
WAVE> !P.Multi = [0, 2, 3]
```

Now you can see all of your plots at the same time.

**Step 3** Draw the original data set by entering:

```
WAVE> PLOT, original, $
    Title = "Original 'PV-WAVE' Data"
```

The plot is displayed in the upper-left corner of the window.

**Step 4** Now display the noisy version by entering:

```
WAVE> PLOT, noisy, Title = 'Noisy Data', $
    Color = 100
```

**Step 5** Display the filter function by entering:

```
WAVE> PLOT, SHIFT(Filter, 100), $
    Title = 'Filter Function', $
    Color = 80, Thick = 2.0
```

---

**NOTE** The SHIFT function was used here to show the filter's peak as centered.

---

**Step 6** Display the lowpass filtered data set by entering:

```
WAVE> PLOT, lowpass, $
    Title = 'LowPass Filtered', Color = 60
```

**Step 7** Display a plot of just the high-frequency noise by entering:

```
WAVE> PLOT, highpass, $
    Title = 'HighPass Filtered', Color=100
```

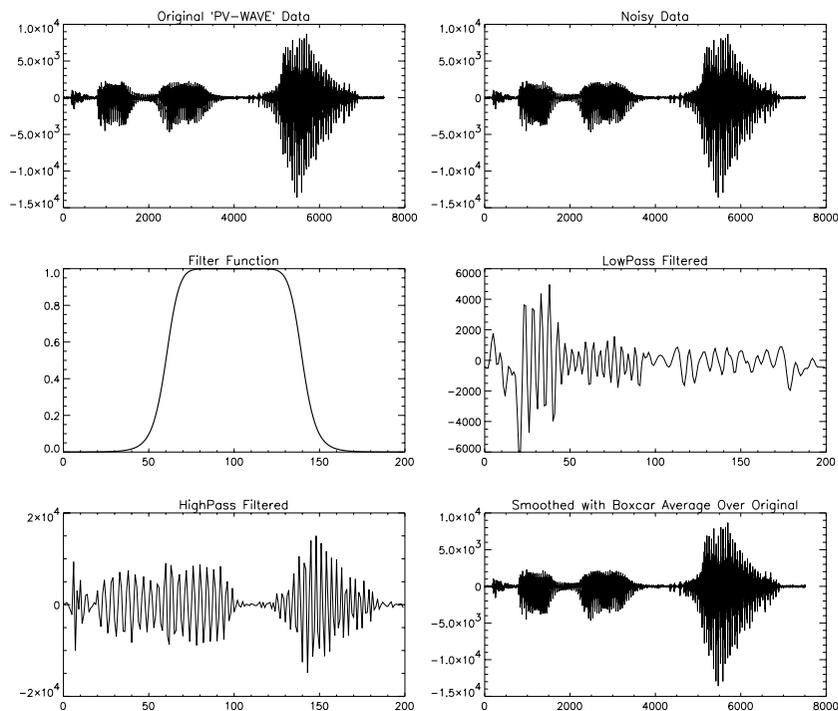
**Step 8** Finally, display the data set that you used the SMOOTH function on so it can be compared with the lowpass filtered data. Enter:

```

WAVE> PLOT, original, Title = $
      ► 'Smoothed with Boxcar Average ' + $
      ► 'Over Original'

WAVE> OPLOTT, smoothed, Color = 80

```



**Figure 6-2** The final plot is displayed in the lower right corner of the window.

**Step 9** Before continuing, reset the plotting window to display a single image by entering the command:

```
WAVE> !P.Multi = 0
```

**Step 10** Remove the large window by entering the command:

```
WAVE> WDELETE, 0
```

The window closes.

## Velocity Field Plotting

An example of a more complicated plotting routine written in the PV-WAVE command language is the VEL routine from the PV-WAVE Standard Library. The Standard Library contains many useful functions and procedures written with the PV-WAVE language. The VEL routine plots velocity vectors given arrays of  $x$  and  $y$  velocity values.

**Step 1** Create a dummy set of  $x$  and  $y$  velocities to visualize by entering:

```
WAVE> VX = original#FINDGEN(20)
```

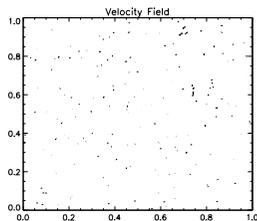
```
WAVE> VY = noisy#FINDGEN(20)
```

The PV-WAVE matrix multiplication operator (#) is used to make the needed dummy arrays.

**Step 2** Now use the VEL command to plot the velocity vectors by entering:

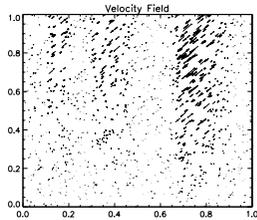
```
WAVE> VEL, VX, VY
```

The information is plotted, with the length of each arrow following the velocity field and being proportional to the field strength.



For most plots, the default appearance is adequate. Nevertheless, you can make the plot more informative by adding a few keywords to the command line.

**Step 3** Specify the length of the lines to be five times as long as the default value (0.1) and specify the number of points to be plotted as six times the default value (200):



```
WAVE> VEL, VX, VY, Length = .5, $
      NVecs = 1200
```

Twelve hundred points are plotted with longer lines than in the previous example.

## More Information on 2D Plotting

Using just a few commands, you have performed some powerful signal processing and plotting tasks. PV-WAVE has many more plotting abilities than the ones you learned in this chapter.

## Contour and Surface Plotting

PV-WAVE provides many techniques, including contour plots, wire-mesh surfaces, and shaded surfaces, for visualizing 2D arrays. This chapter demonstrates a few of these commands.

### Accessing a Data Set

First, you need a 2D data set to visualize. This example is a freely formatted file of the Pikes Peak, Colorado, region. The data set contains information about latitude, longitude, elevation, and snow depth, with 2400 single values per variable.

#### *Using the DC\_READ\_\* Functions to Read Files*

DC\_READ\_\* functions simplify the process of reading many kinds of files. These functions automate many of the steps required to read data by other methods:

- ✓ opening the file
- ✓ assigning it a logical unit number (LUN)
- ✓ closing the file

These functions include the following:

- `DC_READ_FREE` — Reads freely-formatted ASCII files. Furthermore, relieves you of the task of composing a format string that describes the organization of the data in the input file.
- `DC_READ_FIXED` — Reads fixed-formatted ASCII data using a `PV-WAVE` format that you specify.
- `DC_READ_TIFF` — Reads a Tagged Image File Format (TIFF) file.
- `DC_READ_DIB` — Reads a Device Independent Bitmap file.
- `DC_READ_8_BIT` — Reads an 8-bit image file.
- `DC_READ_24_BIT` — Reads a 24-bit image file.

**Step 1** Open the data file, using *status* as the value returned by `DC_READ_FREE`. Enter:

```
WAVE> status = $
    ► DC_READ_FREE(!Data_Dir + $
    ► 'wd_pike_elev.dat', $
    ► lat, lon, elev, depth, /Column)
```

**Step 2** Use the `INFO` command to verify that the new variables *lat*, *lon*, *elev*, and *depth* were created. This command also tells you about the size and data type of the arrays:

```
WAVE> INFO
```

### ***Creating Arrays to Contain the Data***

**Step 1** Create four 2D arrays to contain the data. Use the `REFORM` function to reformat the 1D arrays without changing the values numerically:

```
WAVE> lat = REFORM(lat, 60, 40)
WAVE> lon = REFORM(lon, 60, 40)
WAVE> elev = REFORM(elev, 60, 40)
WAVE> depth = REFORM(depth, 60, 40)
```

**Step 2** Enter the `INFO` command again to see how the arrays have been changed:

```
WAVE> INFO
```

## Plotting with CONTOUR

One way to view a 2D array is as a contour plot. The large number of keywords available make it possible to create almost any kind of contour plot.

### *Creating Basic Contour Plots*

**Step 1** Create a simple contour plot of the elevation by entering:

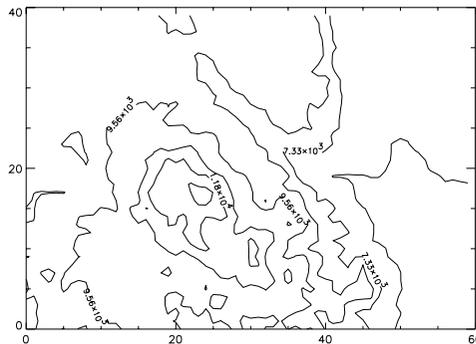
```
WAVE> CONTOUR, elev
```

The contour lines are displayed.

That command was very simple, but the resulting plot wasn't as informative as it could be.

**Step 2** Create a customized CONTOUR plot with more elevations and labels by entering:

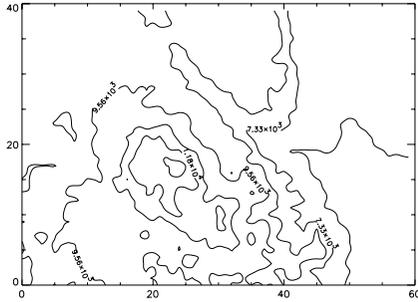
```
WAVE> CONTOUR, elev, NLevels = 8, /Follow
```



**Figure 6-3** The *NLevels* keyword told CONTOUR to plot eight equally-spaced elevation levels. The *Follow* keyword put the labels on the contours.

**Step 3** Smooth the contours using the *Spline* keyword:

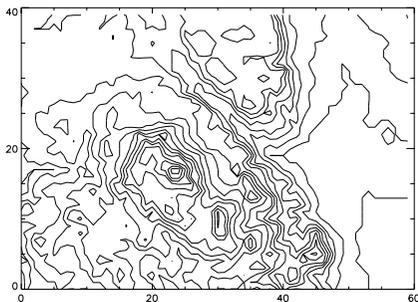
```
WAVE> CONTOUR, elev, NLevels = 8, /Spline
```



**Figure 6-4** The *Spline* keyword also produced labels on the contours.

**Step 4** Add more contour levels to get a better picture of the area:

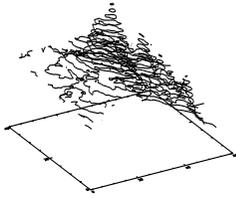
```
WAVE> CONTOUR, elev, NLevels = 24
```



**Figure 6-5** The contour plot with more levels specified using the *NLevels* keyword.

**Step 5** The same contour plot can be rendered from a 3D perspective. First, set the default 3D viewing angle by entering:

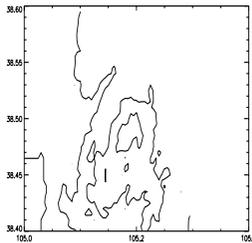
```
WAVE> SURFR
```



**Step 6** By using the *T3d* keyword in the next call to `CONTOUR`, the contours will be drawn as seen from a 3D perspective. Enter:

```
WAVE> CONTOUR, elev, NLevels = 24, /T3d
```

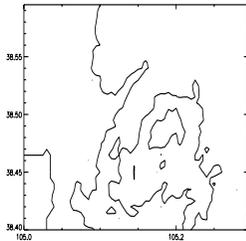
The 3D perspective of the contour is plotted.



**Step 7** Take advantage of the latitude and longitude data by displaying it along the *x*- and *y*-axes:

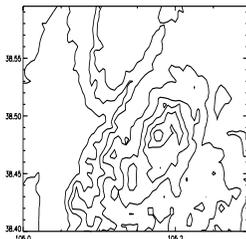
```
WAVE> CONTOUR, elev, lon, lat
```

**Step 8** Make the axes fit the data range by adding the *X*- and *YStyle* keywords:



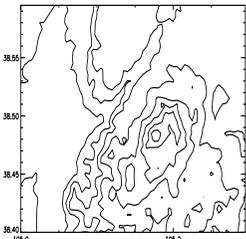
```
WAVE> CONTOUR, elev, lon, lat, XStyle=1, $
      ► YStyle = 1
```

**Step 9** Add more contour levels:



```
WAVE> CONTOUR, elev, lon, lat, XStyle=1, $
      ► YStyle = 1, NLevels = 10
```

**Step 10** Double the thickness of the contour lines using the *Thick* keyword:



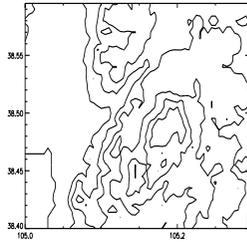
```
WAVE> CONTOUR, elev, lon, lat, XStyle=1, $
      ► YStyle = 1, NLevels = 10, Thick = 2.0
```

The thicker lines add more definition to the contours.

**Step 11** Create a variable to specify the contour levels to display:

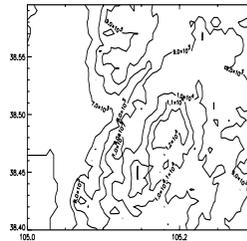
```
WAVE> level = [5000, 6000, 7000, 8000, $
      ► 9000, 10000, 11000, 12000]
```

**Step 12** Display the specified contour levels.



```
WAVE> CONTOUR, elev, lon, lat, XStyle=1, $
      ► YStyle = 1, Levels = level
```

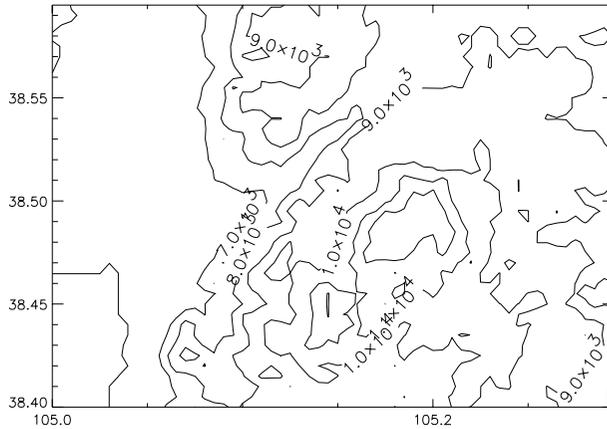
**Step 13** Add labels that display the assigned values to the contour levels:



```
WAVE> CONTOUR, elev, lon, lat, XStyle = 1, $
      ► YStyle = 1, Levels = level, $
      ► C_Label = [1, 1, 1, 1, 1, 1, 1, 1]
```

**Step 14** Increase the size of the characters on the contour labels by 25%:

```
WAVE> CONTOUR, elev, lon, lat, XStyle = 1, $
      ► YStyle = 1, Levels = level, $
      ► C_Label = [1, 1, 1, 1, 1, 1, 1, 1], $
      ► C_Charsize = 1.25
```



**Figure 6-6** The custom contour labels are larger than the tick labels.

### ***Using Color in Contour Plots***

The `TEK_COLOR` command loads 32 predefined, unique, highly saturated colors into the bottom 32 indices of the color table. When the `TEK_COLOR` color table is in effect, you will be able to easily differentiate the different data sets in a line plot.

Chapter 7, *Using Color*, discusses additional color manipulation tools. Both *Chapter 6* and *Chapter 7* discuss methods for using color to enhance data analysis.

**Step 1** Load a color table that specifies the first 32 distinct colors in the lower end of the color table:

```
WAVE> TEK_COLOR
```

**Step 2** Bring up the `COLOR_PALETTE` window. This window displays the colors in the color table, and enables you to see the 32 colors (0-31) in the `TEK_COLOR` palette. The `COLOR_PALETTE` window is useful as a reference when you are choosing plot colors.

```
WAVE> COLOR_PALETTE
```

**Step 3** Create a variable that specifies the indices for the colors of the contour levels:

```
WAVE> color1 = [11, 10, 1, 5, 8, 19, 7, 3, 31]
```

---

**TIP** The color table indices specified in the array must be in the range {0 ... 31} to take advantage of the bright colors created by the TEK\_COLOR procedure. Color table indices above 31 are not affected by the TEK\_COLOR procedure, and will remain as defined by the previously loaded color table.

---

**Step 4** Now display the contour levels in colors, smooth the contours, and add titles to the  $x$ - and  $y$ -axes:

```
WAVE> CONTOUR, elev, lon, lat, XStyle = 1, $
      ► YStyle = 1, Levels = level, $
      ► C_Label = [1, 1, 1, 1, 1, 1, 1, 1], $
      ► C_Charsize = 1.25, C_Colors = color1, $
      ► /Spline, XTitle = 'Longitude', $
      ► YTitle = 'Latitude', Thick = 2
```

The color further defines the contour lines.

### ***Creating 3D Contour Plots***

You can create a 3D contour plot of the elevation using colors for the contour lines.

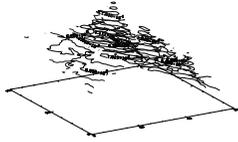
**Step 1** Use the color palette in TEK\_COLOR to choose the colors, then create a variable to specify the color indices for the lines:

```
WAVE> TEK_COLOR
WAVE> color2 = [17, 23, 16, 2, 3, 5, 7, 1, $
      ► 4, 31, 8, 9, 10, 11, 13, 15, 30, 19, $
      ► 20, 21]
```

**Step 2** Use the SURFR procedure to duplicate the rotation, translation and scaling of the SURFACE procedure:

```
WAVE> SURFR
```

**Step 3** Create the contour plot in 3D space:



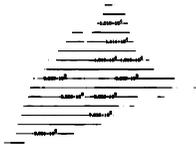
```
WAVE> CONTOUR, elev, NLevels = 20, $
      ► C_Colors = color2, /Follow, /T3d
```

The lines are displayed in the specified colors.

**Step 4** To better see how SURFR works, change the rotation and then re-draw the plot:

```
WAVE> SURFR, AX = 0, AZ = 210
WAVE> CONTOUR, elev, NLevels = 20, $
      ► C_Colors = color2, /Follow, /T3d
```

The x-y plane appears parallel to the bottom of the window.



### ***Filling the Contours with Color Using CONTOURFILL***

You can shade enclosed contours generated by the CONTOUR procedure with the CONTOURFILL procedure. The basic syntax for this procedure is:

```
CONTOURFILL, 'filename', z, x, y
```

**Step 1** Use the CD command to move to a directory where you have write privileges:

```
WAVE> CD, 'pathname'
```

**Step 2** Write the contouring information to a file:

```
WAVE> CONTOUR, elev, XStyle=1, YStyle=1, $
      ┌─> Levels = level, $
      ┌─> Path_FileName = 'peak_elev.dat'
```

The information is saved to the specified file.

**Step 3** Now plot the shaded areas, using the colors you specified earlier:

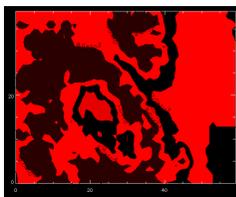
```
WAVE> CONTOURFILL, 'peak_elev.dat', elev, $
      ┌─> Color_Index = color1
```

Now you are ready to display the contour lines. Using the *NoErase* keyword, you can draw them on top of the current plot. To be sure they will show up, draw the lines in black.

**Step 4** Create a variable that specifies the color black:

```
WAVE> color3 = 0
```

**Step 5** Now draw the contour lines on top of the colors:



```
WAVE> CONTOUR, elev, XStyle=1, YStyle=1, $
      ┌─> Levels = level, $
      ┌─> C_Label = [1, 1, 1, 1, 1, 1, 1, 1], $
      ┌─> C_Charsize = 1.25, C_Colors = color3,$
      ┌─> /NoErase
```

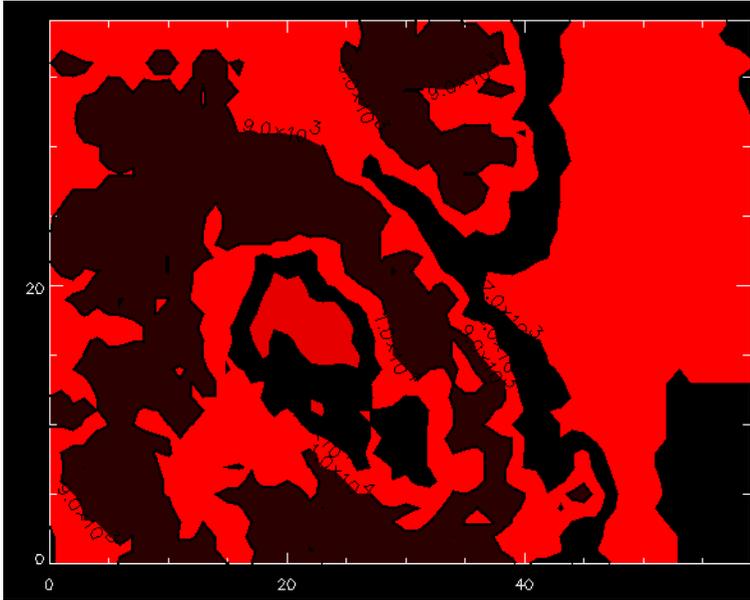
The black lines clearly define the edges of the filled contours.

**Step 6** You can make the lines show up better by doubling the thickness of each. Use the *C\_Thick* keyword, which is specifically for the contour lines:

```
WAVE> CONTOUR, elev, XStyle=1, YStyle=1, $
      ┌─> Levels = level, $
```

```
▣▣▣ C_Label = [1, 1, 1, 1, 1, 1, 1, 1], $
▣▣▣ C_Charsize = 1.25, C_Colors = color3,$
▣▣▣ /NoErase, C_Thick = 2.
```

The black contour lines appear twice as thick.



---

**NOTE** For more information on contouring, see the *PV-WAVE User's Guide*.

---

**TIP** In addition to the CONTOUR procedure, you can create contour plots with the CONTOUR2 procedure. CONTOUR2 is particularly useful for plotting irregularly gridded or “scattered” data. Furthermore, CONTOUR2 includes a *Fill* keyword that allows you to produce filled contours without using the CONTOURFILL method. For detailed information and examples of CONTOUR2, see the *PV-WAVE Reference*.

---

## Plotting with SURFACE

**Step 1** To view the array *elev* as a 3D, wire-frame surface, just enter the command:

```
WAVE> SURFACE, elev
```

A wire-frame representation of the array is displayed.

The SURFACE command can be used to view your data from any angle:

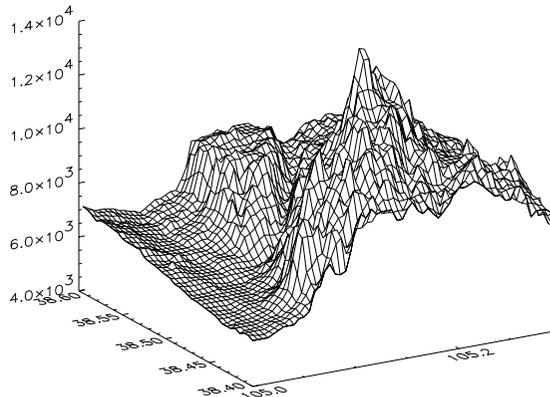
**Step 2** Add the latitude and longitude information:

```
WAVE> SURFACE, elev, lon, lat
```

The *x*- and *y*-axes of the new plot contain the latitude and longitude information.

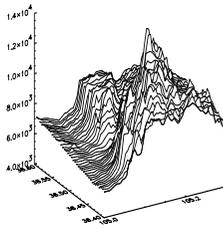
**Step 3** Make the axes fit the data and increase the character size:

```
WAVE> SURFACE, elev, lon, lat, XStyle = 1, $
      CharSize = 1.75
```



**Figure 6-7** The increased character size for the axes text defining the tick marks.

**Step 4** Make the peaks easier to see by drawing only the horizontal lines. Specify that the bottom of the surface be drawn in yellow (color number 7):



```
WAVE> SURFACE, elev, lon, lat, XStyle = 1, $
      CharSize = 1.75, /Horizontal, Bottom = 7
```

The new plot demonstrates that plotting half as many lines as the wire frame mesh provides ample information.

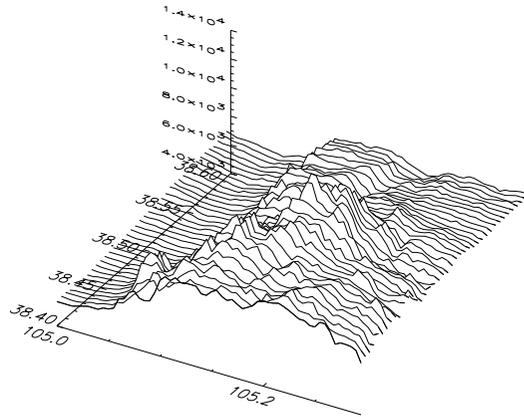
---

**NOTE** The default viewing angle is quite suitable to get a general idea of the surface of this data, but what if you needed to rotate the plot to see the other side and most of the peaks. *AX* and *AZ* are plotting keywords that are used to control the *SURFACE* command. The keyword *AX* specifies the angle of rotation of the surface (in degrees towards the viewer) about the *x*-axis. The *AZ* keyword specifies the rotation of the surface in degrees counterclockwise around the *z*-axis.

---

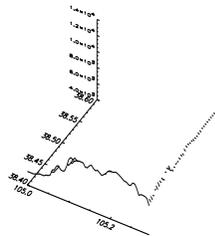
**Step 5** Use the *AX* keyword to rotate the plot 60 degrees along the *x*-axis and use the *AZ* keyword to rotate it  $-30$  degrees along the *z*-axis:

```
WAVE> SURFACE, elev, lon, lat, XStyle = 1, $
      CharSize = 1.75, /Horizontal, Bottom = 7, $
      AZ = -30, AX = 60
```



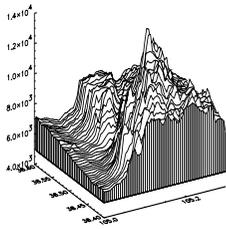
**Figure 6-8** The surface is displayed rotated about the x- and z-axes.

**Step 6** Now, draw only the yellow lines on the bottom of the surface :



```
WAVE> SURFACE, elev, lon, lat, XStyle = 1, $
      ||| CharSize = 1.75, /Horizontal, Bottom = 7, $
      ||| AZ = -30, AX = 60, /Lower_Only
```

**Step 7** Draw the surface in the default viewing angle, adding a yellow skirt that begins at 5000 feet :



```
WAVE> SURFACE, elev, lon, lat, XStyle = 1, $
      CharSize = 1.75, /Horizontal, Bottom = 7, $
      Skirt = 5000
```

## Displaying Data as a Shaded Surface

You can also view a 2D array as a light-source shaded surface.

**Step 1** First, enlarge the current window:

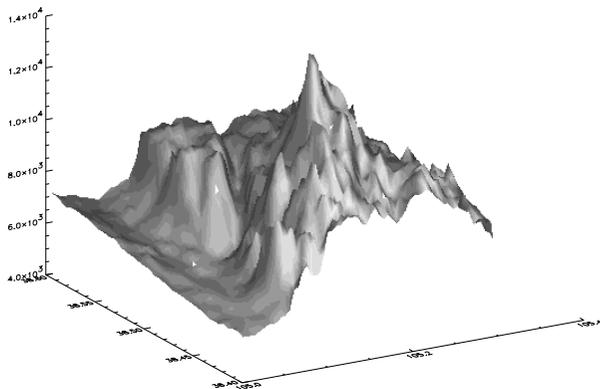
```
WAVE> WINDOW, 0, XSize = 800, YSize = 800
```

**Step 2** Load one of the pre-defined color tables by entering:

```
WAVE> LOADCT, 5
```

**Step 3** To view the light-source shaded surface, simply enter the command:

```
WAVE> SHADE_SURF, elev, lon, lat
```



**Figure 6-9** A shaded surface of the array is displayed.

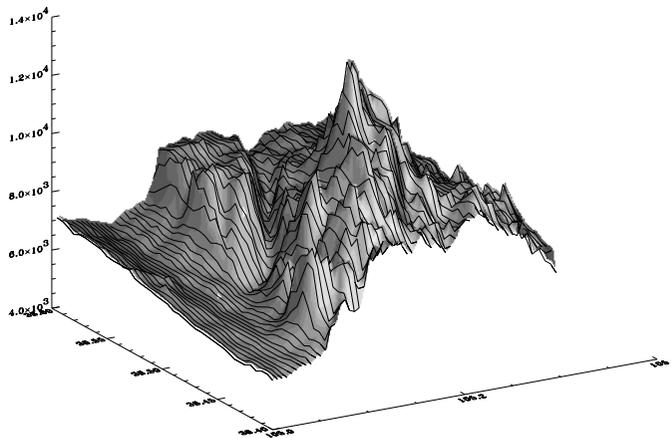
**Step 4** Outline the peaks in black using wire frame SURFACE command with the *Horizontal* keyword:

```
WAVE> SURFACE, elev, lon, lat, XStyle = 4, $
      ►►► YStyle = 4, ZStyle = 4, Color = 0, $
      ►►► /Horizontal, /NoErase
```

---

**NOTE** The *NoErase* keyword allows the SURFACE plot to be drawn over the existing SHADE\_SURF plot. The *XStyle*, *YStyle*, and *ZStyle* keywords are used to select different styles of axes. Here, SURFACE is told not to draw the *x*-, *y*-, and *z*-axes because they were already drawn by the SHADE\_SURF command.

---



**Figure 6-10** The contour lines are plotted on top of the surface.

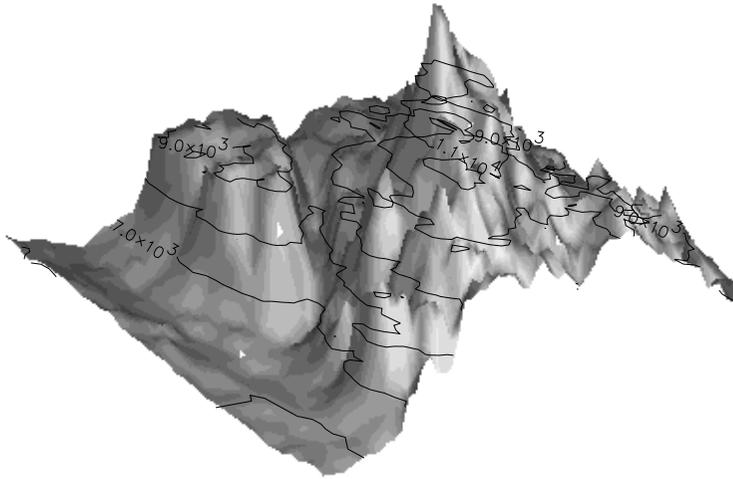
You can plot the contour lines on the shaded surface, but you will need to save the 3D transformation matrix using the *Save* keyword.

**Step 5** Draw the shaded surface without axes and save the transformation:

```
WAVE> SHADE_SURF, elev, lon, lat, $
      ►►► XStyle = 4, YStyle = 4, ZStyle = 4, /Save
```

**Step 6** Now draw the contour lines on the current plot:

```
WAVE> CONTOUR, elev, lon, lat, XStyle = 4, $
      ►►► YStyle = 4, ZStyle = 4, /NoErase, $
      ►►► Levels = level, /Follow, /T3d
```



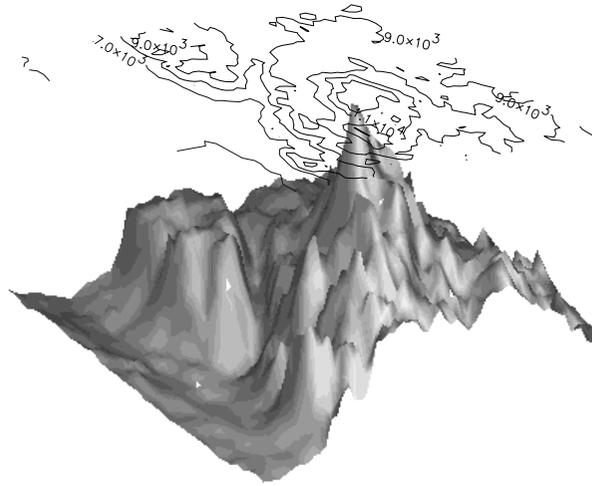
**Figure 6-11** The white contour lines follow the surface.

**Step 7** Now place colored contour lines above the shaded surface. When you loaded color table 5, you overrode the TEK\_COLOR palette, so you need to load it again:

```
WAVE> TEK_COLOR
```

**Step 8** Draw the contour lines in colors. Place them above the shaded surface by specifying displacement along the z-axis (*ZValue*) of 1.0:

```
WAVE> SHADE_SURF, elev, lon, lat, $
    ──▶ XStyle = 4, YStyle = 4, ZStyle = 4, /Save
WAVE> CONTOUR, elev, lon, lat, XStyle = 4, $
    ──▶ YStyle = 4, ZStyle = 4, /NoErase, $
    ──▶ Levels = level, C_Colors = color2, $
    ──▶ /Follow, /T3d, ZValue = 1.0
```



**Figure 6-12** The colored contour lines are drawn at the top of the box.

## Displaying a TIFF File or a Logo

What if you wanted your logo to appear on your plots? PV-WAVE has several tools for reading images in different formats. TIFF (Tagged Image File Format) files are very common and very useful because they are easily transported across platforms. This example shows how to place the Visual Numerics, Inc., logo on your plot.

You may not know the dimensions of the image, but `DC_READ_TIFF` will take care of this problem if you ask it to output them using the keywords *imagewidth* and *imagelength*.

**Step 1** Read the file into the variable *vni\_s*:

```
WAVE> status = $
      DC_READ_TIFF(!Data_Dir + $
      'vni_small.tif', vni_s, $
      Imagewidth = XSize, Imagelength = YSize)
```

You can display the image using the `TV` procedure. The image is displayed by default from the bottom-up, which makes it upside-down and backwards. By specifying the *Order* to have the value 1, the image will display properly.

**Step 2** Display the image:

```
WAVE> TV, vni_s, Order = 1
```

The image is displayed at the bottom of the window.

Now you are ready to place the logo on a plot.

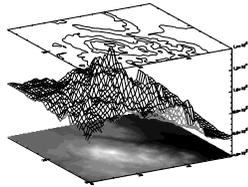
## Showing Three Levels

With a little effort, you were able to display a contour plot above a shaded surface. What if you wanted to do that *and* place an image below the surface? The SHOW3 procedure allows you to do this with one command.

**Step 1** Try it by entering:

```
WAVE> SHOW3, elev, /Interp
```

The image, wire-frame surface and contour are displayed.



To specify the position of the logo, enter a numeric value after the variable name. The numbers that can be used vary, depending on image and screen size, but the value 0 always places the image in the upper left corner.

**Step 2** Add the Visual Numerics logo:

```
WAVE> TV, vni_s, 0, Order = 1
```

The image is displayed in the upper-left corner of the plot.

## Using Color to Represent Data

You can specify a different array for the shading colors.

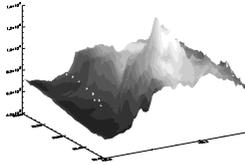
**Step 1** Use TEK\_COLOR again to load 32 distinct colors at the beginning of the current color table:

```
WAVE> TEK_COLOR
```

**Step 2** Find the BYTSCL values of the depth and allow the first 32 distinct values to be excluded:

```
WAVE> shade = BYTSCL(depth, Top=95) + 32b
```

**Step 3** Shade the elevation data plot with snow depth values:



```
WAVE> SHADE_SURF, elev, lon, lat, $  
      XStyle = 1, Shades = shade, Color = 1
```

---

**NOTE** The BYTSCL function scales and converts an array to byte data type. The *Shades* keyword specifies the same dimensions as the shade parameter, which contains the shading color indices.

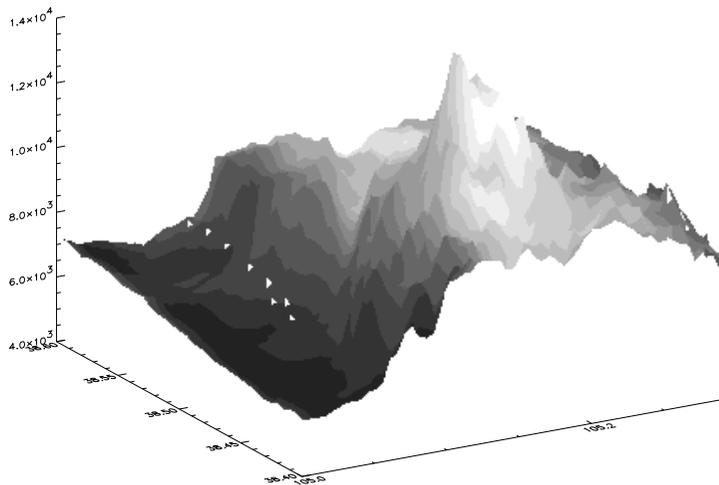
---

The different shading colors correspond to snow depth.

The axes' labels were drawn in white because this was specified by assigning the value of 1 (white in the TEK\_COLOR palette) to the *Color* keyword.

**Step 4** Display the Visual Numerics logo on the plot:

```
WAVE> TV, vni_s, 0, Order = 1
```



**Figure 6-13** The logo is placed in the upper-left corner.

**Step 5** Create a shaded surface that has the shading information provided by the elevation of each point:

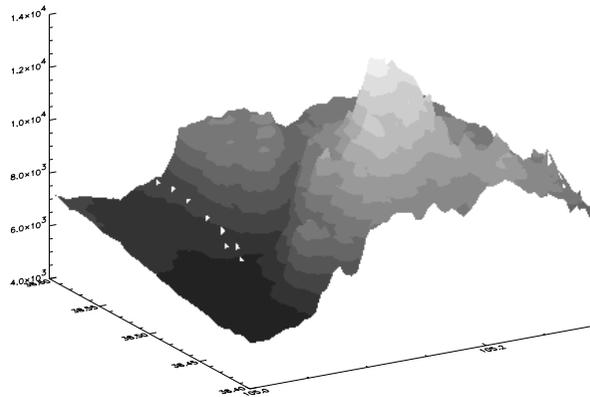
```
WAVE> shade = BYTSCL(elev, Top = 95) + 32b
```

```
WAVE> SHADE_SURF, elev, lon, lat, $
```

```
    XStyle = 1, Shades = shade, Color = 1
```

```
WAVE> TV, vni_s, 0, Order = 1
```

Visual Numerics.

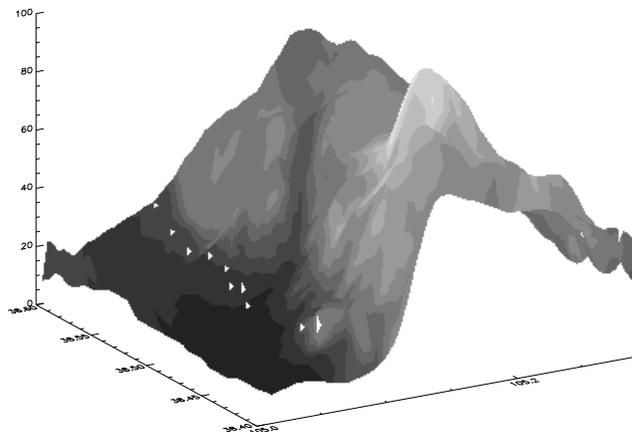


**Figure 6-14** Now different shading colors on the plot correspond to different elevations.

**Step 6** Shade the snow depth data plot with elevation:

```
WAVE> SHADE_SURF, depth, lon, lat, $  
      ► XStyle=1, Shades = shade, Color = 1  
WAVE> TV, vni_s, 0, Order = 1
```

Visual Numerics.



**Figure 6-15** The shading colors correspond to the elevation data.

## More Information on 3D Plotting

The SURFACE, CONTOUR, and SHADE\_SURF commands have many more keywords that can be used to create even more complex, customized plots. For more information see the *PV-WAVE User's Guide*.

---

### ***Deleting Windows***

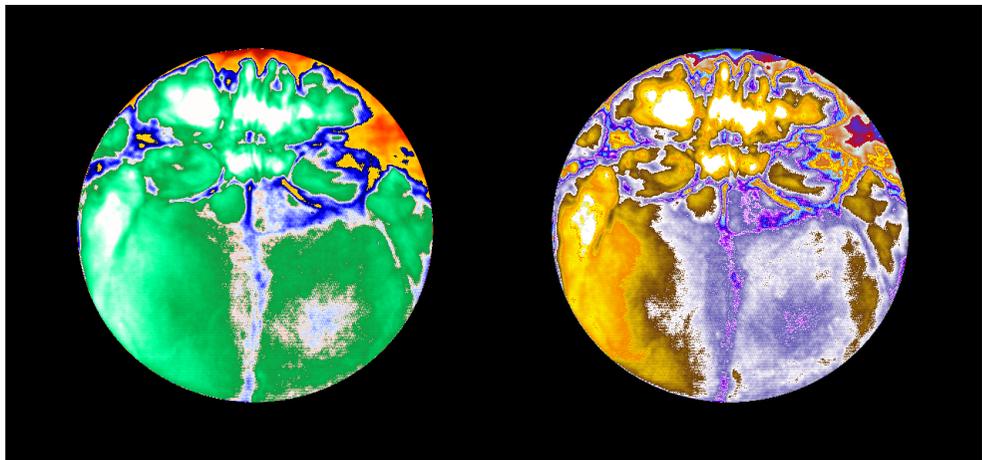
Delete the open graphics windows by entering the following command:

```
WAVE> WHILE (!D.Window GE 0) DO WDELETE,$
      !D.Window
```

All the windows close.

## *Array Processing Techniques*

PV-WAVE is an ideal tool for working with arrays because of its interactive capabilities, uniform notation, and array-oriented operators and functions. This chapter demonstrates some of PV-WAVE's capabilities when working with arrays and also focuses on the ability to display and process your data as an image.



**Figure 7-1** An image of a section of a human brain made using a computer-aided tomography scan. Emphasis on different structures is created in these two views by adjusting the color table.

---

## Previewing This Chapter

This chapter demonstrates some basic image processing techniques using PV-WAVE. To preview the steps you will take and the plots you will create in this chapter, do the following:

**Step 1** Use the CD command to move to the subdirectory that contains code for the tutorial. At the WAVE> prompt, enter:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code'
```

**Step 2** Run the chapter preview by entering the following command at the WAVE> prompt:

```
WAVE> @lesson_6
```

The batch file runs the program, demonstrating the plots you will produce in this chapter.

---

## Displaying and Processing Arrays

Images, which are 2D arrays, are easily visualized in PV-WAVE and can be processed just like any other array. PV-WAVE also contains many procedures and functions specifically designed for image display and processing.

### Reading an Array

First, import an image to be processed. Reading data files into PV-WAVE is easy if you know the format in which the data is stored. Often, images are stored as arrays of bytes.

**Step 1** Open the file for reading by entering:

```
WAVE> OPENR, 1, $
    ──▶ FILEPATH('head.img', SubDir = 'data')
```

---

**NOTE** The OPENR command opens the file named in quotes and assigns it to the specified logical unit number. Here you assign the file `head.img` to unit number 1. Unit numbers can range from 1 to 128. The FILEPATH function, used as an argument to OPENR, returns the full path for the file `head.img` located in the PV-WAVE data directory.

---

**Step 2** The image you read is a 512-by-512-element array of bytes, so you will create a 512-element square array variable called `head_LS` by entering:

```
WAVE> head_LS = BYTARR(512, 512)
```

**Step 3** Read the image into the variable `head_LS` and close the file `head.img` by entering:

```
WAVE> READU, 1, head_LS
```

```
WAVE> CLOSE, 1
```

## Displaying an Array

The default window size is 640-by-512 pixels.

**Step 1** Since the image is 512-by-512, create a custom-sized window for the display:

```
WAVE> WINDOW, XSize = 512, YSize = 512
```

The window appears in the upper right corner.

---

**NOTE** Many of the keyword names can be abbreviated. In most cases, they can be abbreviated to the smallest unique set of letters. A slash (/) can be used to set a keyword equal to 1 or True.

---

**Step 2** Now return to the default color table, *B-W Linear*, and display the image by entering:

```
WAVE> LOADCT, 0
```

```
WAVE> TV, head_LS
```

The image appears in the window.

---

**NOTE** The TV command writes an array to the display as an image without scaling. TVSCL scales the values of the image array into the range of available colors.

---

**Step 3** To display the image with its values scaled to use the entire color table, enter:

```
WAVE> TVSCL, head_LS
```

The image uses the entire set of color table values.

## Changing the Size of an Image

Two commands commonly used to expand or shrink image sizes are CONGRID and REBIN.

### **CONGRID**

- ✓ Uses nearest neighbor interpolation
- ✓ Faster than REBIN
- ✓ Can produce final dimensions of arbitrary size
- ✓ Uses bilinear interpolation when the *Interp* keyword is specified

### **REBIN**

- ✓ Uses bilinear interpolation
- ✓ Takes longer than CONGRID
- ✓ Produces higher quality results
- ✓ Produces final dimensions that are integral multiples or factors of the original

**Step 1** Use the CONGRID function to create an enlarged image:

```
WAVE> head_LSbig = CONGRID(head_LS, 768, $
    ──▶ 768)
```

**Step 2** Now you need a larger window:

```
WAVE> WINDOW, XSize = 768, YSize = 768
```

```
WAVE> TVSCL, head_LSbig
```

Work with smaller images now so that you can display six windows simultaneously.

**Step 3** Make the image smaller and display it in a smaller window. Enter:

```
WAVE> b = CONGRID(head_LS, 384, 384)
```

```
WAVE> WINDOW, 0, XSize = 384, YSize = 384
```

```
WAVE> TVSCL, b
```

**Step 4** Load an *rgb*-range color table and display the image:

```
WAVE> LOADCT, 15 & TVSCL, b
```

The TV and TVSCL commands can also accept array expressions as arguments.

**Step 5** For example, enter the command:

```
WAVE> TV, b*1.5
```

Each element of *b* is multiplied by 1.5 and the result is sent to the display. The data in variable *b* remains unchanged.

**Step 6** Multiply each element by 2:

```
WAVE> TV, b*2
```

Each element of *b* is multiplied by 2.

## Contrast Enhancement

*Thresholding* is one of the simplest contrast enhancements that can be performed on an image. Thresholding produces a two-level mapping from all of the possible intensities into black and white.

---

**NOTE** The PV-WAVE relational operators, EQ, NE, GE, GT, LE, and LT, return a value of 1 if the relation is true and 0 if the relation is false. When applied to images, the relation is evaluated for each pixel and an image of 1s and 0s is created. The relational operators are:

---

- ✓ EQ — Equal
  - ✓ GE — Greater than or equal to
  - ✓ GT — Greater than
  - ✓ LE — Less than or equal to
  - ✓ LT — Less than
  - ✓ NE — Not equal
- 

**Step 1** Add a new window:

```
WAVE> WINDOW, 1, XSize = 384, YSize = 384
```

**Step 2** To display the pixels in the image *b* that have values greater than 110 as white and all others as black, enter:

```
WAVE> TVSCL, b GT 110
```

**Step 3** Similarly, you can display the pixels that have values less than 110 as white by entering the command:

```
WAVE> TVSCL, b LT 110
```

---

**NOTE** The LT and GT operators display the pixels in either black or white, whereas the < and > operators display the pixels in the full range of brightness.

---

Another way to enhance the contrast of an image is to scale a subrange of pixel values to fill the entire range of displayed brightness. The maximum operator > (greater than), returns a result equal to the larger of its two parameters.

**Step 4** Use the maximum operator to scale pixels with a value of 110 or greater into the full range of displayed brightness. Type:

```
WAVE> TVSCL, b > 110
```

This highlights certain aspects of the neural tissue that didn't show before.

**Step 5** To scale pixels with a value less than 110 into the full range of brightness, use the minimum operator < (less than). Enter:

```
WAVE> TVSCL, b < 110
```

This highlights the air spaces, soft tissue, and vascular areas.

Similarly, you can set the minimum brightness to 40, set the maximum brightness to 160, scale the image and display it by entering:

```
WAVE> TVSCL, b > 40 < 160
```

The neural portion is very bright.

**Step 6** Although this command illustrates the use of the minimum and maximum operators, you can execute the same function more efficiently with the command:

```
WAVE> c = BYTSCL(b, Min = 40, Max = 160, $
    ┌─► Top = !D.N_Colors)
```

```
WAVE> TVSCL, c
```

Notice that this image looks exactly like the previous one.

**Step 7** Add another window:

```
WAVE> WINDOW, 2, XSize = 384, YSize = 384
```

In many images, the pixels have values that are only a small subrange of the possible values. By spreading the distribution so that each range of pixel values contains an approximately equal number of members, the information content of the display is maximized. The HIST\_EQUAL function performs this redistribution on an array.

**Step 8** Create and retain a histogram-equalized image in a new variable called *h*:

```
WAVE> h = HIST_EQUAL(c)
```

**Step 9** Display *h* by entering:

```
WAVE> TVSCL, h
```

The displayed image appears much brighter, but with less detail than *c*.

## Smoothing and Sharpening

The SMOOTH function is typically used to:

- Remove ripples, spikes or high frequency noise
- Blur an image or set of data so that only the general trends can be seen
- Isolate the lower spatial frequency components
- Soften sharp transitions from one color to another in a color table

Images can be rapidly smoothed to soften edges or compensate for random noise in an image using the SMOOTH function. This function performs an equally-weighted smoothing using a square neighborhood of a specified odd width, which is called *boxcar averaging*; the MEDIAN function finds the median value of an array or applies a median filter of a specified width.

MEDIAN smoothing replaces each point with the median of the 1D or 2D neighborhood of the given width, effectively eliminating high and low values without blurring any edges.

**Step 1** Display a median smoothed image of *c* with a 1D neighborhood of 3:

```
WAVE> TVSCL, MEDIAN(c, 3)
```

SMOOTH uses a boxcar average. If the first boxcar width you try does not give you the results you expected, try a different width. Boxcar widths should always be odd numbers.

**Step 2** Display the image smoothed using a 3-by-3 neighborhood by entering:

```
WAVE> TVSCL, SMOOTH(c, 3)
```

**Step 3** Try a wider neighborhood:

```
WAVE> TVSCL, SMOOTH(c, 7)
```

This image looks a bit blurred and contains only the low frequency components of the original image.

**Step 4** Now display the image smoothed using a 5-by-5 neighborhood by entering:

```
WAVE> TVSCL, SMOOTH(c, 5)
```

This image looks less blurred.

**Step 5** Add a new window:

```
WAVE> WINDOW, 3, XSize = 384, YSize = 384
```

---

**NOTE** Often, an image needs to be sharpened so that edges or high spatial frequency components of the image are enhanced. One way to sharpen an image is to subtract a smoothed image containing only low-frequency components from the original image. This technique is called *unsharp masking*.

---

**Step 6** To unsharp mask and save as a new variable, enter:

```
WAVE> smoothed = FIX(c - SMOOTH(c, 5))
```

Recall that  $c$  was created by displaying only values in the range 40–160. A smoothed version of the image  $c$  is subtracted from the original  $c$ , scales the result and saves it as a variable called *smoothed*. The `FIX` command converts byte data to integer data.

**Step 7** Display the image, enter:

```
WAVE> TVSCL, smoothed
```

**Step 8** It is also possible to subtract a smoothed version of the original image ( $b$ ) from  $c$ . Enter:

```
WAVE> smoothed = FIX(c - SMOOTH(b, 5))
```

**Step 9** Display the image, enter:

```
WAVE> TVSCL, smoothed
```

**Step 10** Add a new window:

```
WAVE> WINDOW, 4, XSize = 384, YSize = 384, $  
    ┌─▶ XPos = 385, YPos = 350
```

**Step 11** Re-display the original image and compare it to the other images:

```
WAVE> TVSCL, b
```

---

**NOTE** PV-WAVE has other built-in sharpening functions that use differentiation to sharpen images. The ROBERTS and SOBEL functions are edge enhancement generators. The ROBERTS function returns the Roberts gradient of an image.

---

**Step 12** Try the ROBERTS function by entering:

```
WAVE> r = ROBERTS (c)
```

```
WAVE> TVSCL, r
```

**Step 13** The *prism* color table, color table number 6, shows the results better. Load it by entering:

```
WAVE> LOADCT, 6
```

```
WAVE> TVSCL, r
```

The convoluted tissue is displayed primarily in red.

---

**NOTE** Another commonly-used gradient operator is the SOBEL operator. The SOBEL function operates over a 3-by-3 region, making it less sensitive to noise than some other methods.

---

**Step 14** Display a SOBEL-sharpened version of the image:

```
WAVE> so = SOBEL (c)
```

```
WAVE> TVSCL, so
```

## Other Image Manipulations

Sections of images are easily displayed by using subarrays.

**Step 1** Create a new window:

```
WAVE> WINDOW, 5, XSize = 400, YSize = 400, $
```

```
    ► XPos = 385, YPos = 0
```

**Step 2** Create a new array that contains just the 100-by-100 pixels of image *b* with the cerebellum in it and display it by entering:

```
WAVE> d = b (210:309, 130:229)
```

```
WAVE> TVSCL, d
```

The image appears in the lower-left corner of the new window.

**Step 3** Load the standard gamma-II color table:

```
WAVE> LOADCT, 5
```

The displayed image is small, so you can “magnify” the image to an arbitrary size using bilinear interpolation with the REBIN command.

---

**NOTE** REBIN allows you to scale each dimension by an integer factor.

---

**Step 4** Make each dimension of *d* four times its current size and display the result by entering:

```
WAVE> e = REBIN(d, 400, 400)
```

```
WAVE> TVSCL, e
```

The image has, in effect, been magnified four times.

**Step 5** To make this image the same size as the others while still using bilinear interpolation, use the CONGRID function with the *Interp* keyword:

```
WAVE> f = CONGRID(d, 384, 384, /Interp)
```

**Step 6** Change the size of the window:

```
WAVE> WINDOW, 5, XSize = 384, YSize = 384, $  
    XPos = 385, YPos = 0
```

**Step 7** Display *f*:

```
WAVE> TVSCL, f
```

---

**NOTE** Simple rotation in multiples of 90 degrees can be accomplished with the ROTATE function.

---

**Step 8** Display the magnified image rotated by 90 degrees by entering:

```
WAVE> g = ROTATE(f, 1)
```

The second parameter of ROTATE is an integer from 1 to 8 that specifies which one of the eight possible combinations of rotation and axis reversal to use.

**Step 9** Display *g*:

```
WAVE> TVSCL, g
```

The image is rotated 90 degrees counterclockwise.

### ***Using PV-WAVE Color Tables***

The predefined color tables offer a variety of ways to view the data. Try the following color tables and compare the results.

---

**TIP** Use the up arrow ( $\uparrow$ ) key to recall commands, then edit the color table number to rapidly cycle through the color tables.

---

```
WAVE> LOADCT, 0
WAVE> LOADCT, 1
WAVE> LOADCT, 2
WAVE> LOADCT, 3
WAVE> LOADCT, 4
WAVE> LOADCT, 5
WAVE> LOADCT, 6
WAVE> LOADCT, 7
WAVE> LOADCT, 8
WAVE> LOADCT, 10
WAVE> LOADCT, 11
WAVE> LOADCT, 12
WAVE> LOADCT, 14
WAVE> LOADCT, 15
```

The WDELETE function enables you clear the screen by removing the unnecessary windows you created in this chapter:

```
WAVE> WDELETE, 0 & WDELETE, 1 & $  
      ┌─▶ WDELETE, 2 & WDELETE, 3 & WDELETE, 4 &  
      ┌─▶ WDELETE, 5
```

All the windows close.

## Extracting Profiles

Another useful image processing tool is the Standard Library routine `PROFILES`. This routine draws row or column profiles of an image, and it allows you to simultaneously view an image and an  $x$ - $y$  plot of the pixel brightness in any row or column of the image.

**Step 1** Redisplay the rotated image.

```
WAVE> TVSCL, g
```

**Step 2** Use the `PROFILES` routine with the rotated image that you just displayed by entering:

```
WAVE> PROFILES, g
```

A new window for displaying the profiles appears.

**Step 3** Move the pointer into the graphics window (the one containing the image  $g$ ) to display the profiles of different rows and columns.

**Step 4** While the pointer is in the graphics window, press the left mouse button to switch between displaying row and column profiles.

**Step 5** Exit the `PROFILES` routine by clicking the right mouse button while the pointer is in the graphics window.

The `PROFILES` window closes.

**Step 6** Now delete window 5:

```
WAVE> WDELETE, 5
```

## More Information on Array Processing

For more information on image display and image processing, see the *PV-WAVE User's Guide*.



## *Using Color*

PV-WAVE provides numerous ways to enhance your data through the use of color. Working with color can enhance your data and provide insight into the subtleties of your data. This chapter shows you how to use PV-WAVE's predefined color tables, how to customize color usage, and ways to use color to improve your results.



**Figure 8-1** An image of New York City, New York. In the upper left corner, the Brooklyn, Manhattan, and Williamsburg bridges can be distinguished to the right of the tip of lower Manhattan.

---

## Using Color Tables

PV-WAVE provides 16 pre-defined color tables that enable you to apply a large variety of color values to your illustrations. Color tables map the data values written to the screen to different colors and intensities.

### About Color in Previous Chapters

You have already seen a number of ways to handle color in previous chapters. In Chapter 3, you began using color tables, `COLOR_PALETTE` and `!D.N_Colors` to work with a wide range of colors.

If only a few distinct color values are necessary, as in the case of line plots, the Standard Library procedure `TEK_COLOR` may be all you need. The `TEK_COLOR` procedure, which you used in Chapter 5, defines and loads 32 distinct colors. You assign and then reference the assigned colors by using the *Color* keyword or the *C\_Colors* keyword (for contour plots).

**Step 1** For example, type:

```
WAVE> PLOT, RANDOMN(seed, 10)

WAVE> TEK_COLOR

WAVE> COLOR_PALETTE

WAVE> OPLOT, RANDOMN(seed, 10), Color = 4

WAVE> OPLOT, RANDOMN(seed, 10), Color = 6
```

In *Chapter 6*, you loaded all the pre-defined color tables and saw how changing a color table can greatly affect a displayed image.

This chapter shows you how to create custom color tables, as well as some techniques for creatively managing color.

### Creating Your Own Colors

The PV-WAVE Standard Library offers numerous routines that enable you to create and save your own color tables. The type of color table you use depends on your system's capabilities. PV-WAVE accepts color specification in the RGB (Red, Green, and Blue), HSV (Hue, Saturation, and Value), or HLS (Hue, Lightness, and Saturation) color systems. Most devices capable of displaying color use the RGB

color system. Algorithms exist in PV-WAVE to convert colors from one system to another.

For detailed information about color systems see the *PV-WAVE User's Guide*.

**Step 1** To see the number of colors currently available enter:

```
WAVE> PRINT, !D.N_Colors
```

PV-WAVE returns the number of colors that have not been reserved and are available to the device to which you are printing (the monitor).

### ***Creating Custom Color in a Display***

PV-WAVE maintains its own internal color table, which is read and written by the TVLCT procedure. If you want to create a few of your own colors or know exactly what values of red, green and blue your colors should be, you can explicitly define these vectors.

In this exercise, you create a graph with the axes drawn in white, and then successively add red, green, blue and yellow lines. As there are 5 distinct colors, plus one color for the background, a 6-element color table is created.

Usually, color index 0 represents black (0, 0, 0). You could choose color indices as follows:

- 1 = white (1, 1, 1)
- 2 = red (1, 0, 0)
- 3 = green (0, 1, 0)
- 4 = blue (0, 0, 1)
- 5 = yellow (1, 1, 0)

The display must have at least 3 bits per pixel to represent 6 colors simultaneously, and an 8-bit color table is assumed.

**Step 2** Create a data set. Type:

```
WAVE> x = SIN ( ( FINDGEN (200) /35 ) ^2.5 )
```

```
WAVE> y = (x - 0.5)
```

```
WAVE> z = (x - 0.75)
```

```
WAVE> r = (x - 1.0)
```

**Step 3** Specify the components for each color. Type:

```
WAVE> red = [0, 1, 1, 0, 0, 1]
```

```
WAVE> green = [0, 1, 0, 1, 0, 1]
```

```
WAVE> blue = [0, 1, 0, 0, 1, 0]
```

**Step 4** Load the first 6 elements of the color table. Type:

```
WAVE> TVLCT, 255*red, 255*green, 255*blue
```

**Step 5** Draw the axes in white, color index 1. Type:

```
WAVE> PLOT, Color = 1, /NoData, x
```

$x$ - and  $y$ -axes are drawn in white on a black background.

---

**NOTE** `NoData` is used with the `PLOT` command to specify that the axes, titles, and annotation be drawn without plotting data points.

---

**Step 6** Draw in red. Type:

```
WAVE> OPLOT, Color = 2, x
```

The variable  $x$  is drawn in red.

**Step 7** Draw in green. Type:

```
WAVE> OPLOT, Color = 3, y
```

The variable  $y$  is drawn in green.

**Step 8** Draw in blue. Type:

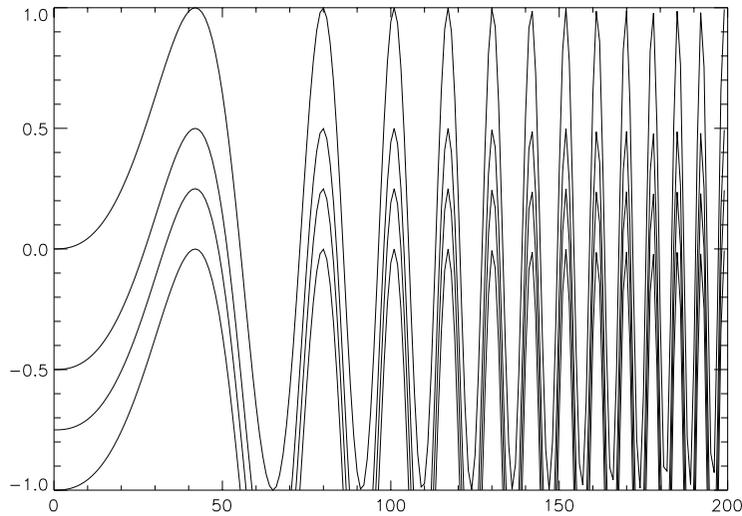
```
WAVE> OPLOT, Color = 4, z
```

The variable  $z$  is drawn in blue.

**Step 9** Draw in yellow. Type:

```
WAVE> OPLOT, Color = 5, r
```

The variable  $r$  is drawn in yellow.



**Figure 8-2** The four variables are plotted with their identifying colors.

---

## Using Color to Enhance Analysis

By using various color tables and changing the parameters, you can produce striking results in terms of image analysis.

### Displaying an Image

You will use an image of New York City for your display.

**Step 1** Open the file `nyc.dat` for reading. Type:

```
WAVE> OPENR, 1, FILEPATH('nyc.dat', $
    SubDir = 'Data')
```

You need an array 1024-by-1024 to hold all the data.

**Step 2** Create the variable `nyc` by entering:

```
WAVE> nyc = BYTARR(1024, 1024)
```

**Step 3** Now read the New York City data. Type:

```
WAVE> READU, 1, nyc
```

**Step 4** Close the file by typing:

```
WAVE> CLOSE, 1
```

The image is too large to display all of it on most screens.

**Step 5** Create a smaller version of the image.

```
WAVE> small = REBIN(nyc, 512, 512)
```

**Step 6** Display the data in your large window by typing:

```
WAVE> TV, small, Order = 1
```

An image of New York City appears in the window.

**Step 7** Load a color table to add a new dimension. Type:

```
WAVE> LOADCT, 4
```

The city appears in color.

## Changing Colors in the Image

Now you are ready to see the effects of changing colors in your image.

You can select any of the color tables from the PV-WAVE command line, as you have done in previous exercises. The default color table, 0, is *B-W Linear*, a gray-scale color table. The other color tables are numbered 1-15.

**Step 1** Load the *Red Temperature* color table. At the PV-WAVE prompt, enter:

```
WAVE> LOADCT, 3
```

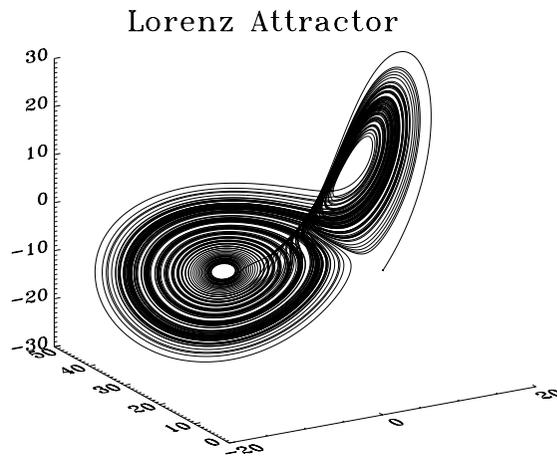
**Step 2** Use LOADCT to load several other color tables.

### ***More Information on Color***

From these exercises, you can see that color offers you great potential for enhancing image analysis. For more information, refer to the *PV-WAVE User's Guide*.

## *Advanced Math and Stats*

PV-WAVE:IMSL Mathematics and PV-WAVE:IMSL Statistics provide some of the most powerful mathematics and statistics capabilities available in software today. These products are separate options that you can purchase which greatly expand the analysis capabilities of PV-WAVE.



**Figure 9-1** Trajectory solution of an ordinary differential equation (ODE). This graphic is generated in this chapter, beginning on page [177](#).

---

## Programs You Can Use

This chapter contains examples of two types of programs:

- programs that use functions from PV-WAVE:IMSL Mathematics and
- programs that use PV-WAVE:IMSL Statistics functions.

To run a program that contains Mathematics functions, you must have installed PV-WAVE:IMSL Mathematics. Likewise, example programs that use functions from PV-WAVE:IMSL Statistics require that product to be installed.

To start PV-WAVE:IMSL Mathematics, enter at the WAVE> prompt:

```
WAVE> @math_startup
```

To start PV-WAVE:IMSL Statistics, enter at the WAVE> prompt:

```
WAVE> @stat_startup
```

The programs in this chapter can be found in the following directory, where <maindir> is the main directory where PV-WAVE is installed:

```
<maindir>
```

You can run them, study them, and use any as a pattern for your own programs.

This chapter demonstrates a variety of mathematics and statistics functions available in PV-WAVE:IMSL Mathematics and PV-WAVE:IMSL Statistics along with sample programs. Whether you examine the sample programs as they are listed here in the “hardcopy” or look at them online using a text editor, we recommend that you look carefully at the underlying code for each example.

For detailed information on the mathematics and statistics functions presented in this chapter, see the *PV-WAVE:IMSL Mathematics Reference* and the *PV-WAVE:IMSL Statistics Reference*. You can also refer to the online Help system.

---

## Solving a Nonlinear System of Equations

The example program `nonlin_system` shows the use of the PV-WAVE:IMSL Mathematics procedure ZEROSYS to solve a system of  $n$  nonlinear equations  $f(x) = 0$ .

---

**NOTE** ZEROSYS uses different keywords to solve this problem using different algorithms. Most PV-WAVE procedures implement various keywords so that you obtain added functionality with each procedure. The fact that the keywords are optional makes programming in PV-WAVE less cumbersome.

---

In this example, the following nonlinear system of equations is solved:

$$x_0^2 - 81(x_1 + 0.1)^2 + \sin(x_2) + 1.06 = 0$$

$$3x_0 - \cos(x_1 x_2) - \frac{1}{2} = 0$$

$$e^{-x_0 x_1} + 20x_2 + \frac{10\pi - 3}{3} = 0$$

ZEROSYS solves the problems several times, using a different keyword each time after the first. Initially, the equation is solved by providing ZEROSYS with only the function defining the system and the size of the problem. Next, the keyword *XGuess* is added. *XGuess* is the array with  $n$  components containing the initial guess. The default is that all components of *XGuess* are zero.

The third time, the system is solved by using the keyword, *Double*, so that the problem is solved in double precision. In the final solution the system is solved using double precision and by specifying the analytic Jacobian matrix with the keyword, *Jacobian*.

## Running the Example Program

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed and initialized to run this example (@math\_startup).

---

**Step 1** If you are not in the advantage subdirectory, go to it by entering:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Run the sample program. Enter:

```
WAVE> nonlin_system
```

### Example Program Listing: nonlin\_system

The following is a complete listing of the program nonlin\_system. This code can also be found online in the following file, where <maindir> is the main directory where PV-WAVE is installed:

```
<maindir>\docs\tutorial\code\advan-  
tage\nonlin_system.pro
```

```
; nonlin_system.pro  
; This example illustrates finding the zero of a system of  
; nonlinear equations  
;  
; Define the system of equations  
FUNCTION NONLIN_FCN, x  
; f will be the same type and size of x  
f = x  
f(0) = 3*x(0) - cos(x(1)*x(2)) - .5  
f(1) = x(0)*x(0) - 81*(x(1)+0.1)*(x(1)+0.1) + sin(x(2)) + 1.06  
f(2) = exp(-x(0)*x(1)) + 20*x(2) + (10*pi - 3)/3  
RETURN, f  
END  
; Define the Jacobian of the system  
FUNCTION NONLIN_JACOBIAN, x  
; df will be the same type and size of x#x  
df = x#x  
df(0, *) = [3, x(2)*SIN(x(1)*x(2)), x(1)*SIN(x(1)*x(2))]
```

```

df(1, *) = [2*x(0), -162*(x(1)+0.1), COS(x(2))]
df(2, *) = [-x(1)*EXP(-x(0)*x(1)), -x(0)*EXP(-x(0)*x(1)), 20]
RETURN, TRANSPOSE(df)
END

; Main Program
PRO NONLIN_SYSTEM
; Call PV-WAVE procedure ZEROSYS to find the zeros
;of the nonlinear system.
;The system will be solved several times showing the usage of
;PV WAVE keywords for ZEROSYS
;
; Solve the system using no keywords
result = ZEROSYS('nonlin_fcn', 3)
PRINT, 'Solution'
PRINT, 'Computed solution for x: ', result
PRINT, 'Error in solution:          ', ABS(NONLIN_FCN(result))
;
; Solve the system providing an initial guess
result = ZEROSYS('nonlin_fcn', 3, XGuess = [1, 1, 1])
PRINT, ' '
PRINT, 'Solution with Initial Guess (1, 1, 1)'
PRINT, 'Computed solution for x: ', result
PRINT, 'Error in solution:          ', ABS(NONLIN_FCN(result))
;
; Solve the system using double precision
result = ZEROSYS('NONLIN_FCN',3, /Double)
PRINT, ' '
PRINT, 'Solution with double precision'
PRINT, 'Computed solution for x: ', result
PRINT, 'Error in solution:          ', ABS(NONLIN_FCN(result))
;
; Solve the system using double precision and analytic Jacobian
result = ZEROSYS('NONLIN_FCN', 3, /Double, $
  jacobian = 'NONLIN_JACOBIAN')
PRINT, ' '
PRINT, 'Solution with double precision and analytic Jacobian'
PRINT, 'Computed solution for x: ', result
PRINT, 'Error in solution:          ', ABS(NONLIN_FCN(result))
END

```

---

## Using a 2D B-Spline Interpolation Procedure

The example program `heat` uses the PV-WAVE:IMSL Mathematics 2D B-spline interpolation procedure. The data file `heat.dat` (obtained from a *Farmer's Almanac*) contains limited information detailing the heat index for a given Fahrenheit temperature and relative humidity. Interpolation procedures must be used to obtain values not present in the file `heat.dat`.

The data file `heat.dat` is in the following directory, where `<maindir>` is the main directory where PV-WAVE is installed:

```
<maindir>\wave\data\heat.dat
```

A listing of this file shows the following data:

```
999 70 75 80 85 90 95 100
0 64 69 73 78 83 87 91
10 65 70 75 80 85 90 95
20 66 72 77 82 87 93 99
30 67 73 78 84 90 96 104
40 68 74 79 86 93 101 110
50 69 75 81 88 96 107 120
60 70 76 82 90 100 114 132
70 70 77 85 93 106 124 144
80 71 78 86 97 113 136 156
90 71 79 88 102 122 148 169
100 72 80 91 108 131 161 181
```

The first row represents temperature and the first column represents the relative humidity. The other entries correspond to the heat index for a given temperature and humidity value.

---

**NOTE** The **heat index** is a value representing the “perceived” temperature when the relationship between relative humidity and air temperature is taken into account.

---

The function `HEAT` accepts the parameter *humidity* and returns and interprets the value for the heat index. The default splines produced are of order 4, or cubic splines. The procedure `HEATPLOT` uses the function `HEAT` to produce a plot.

## Running the Example Program

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

**Step 1** If you are not in the `advantage` directory, enter:

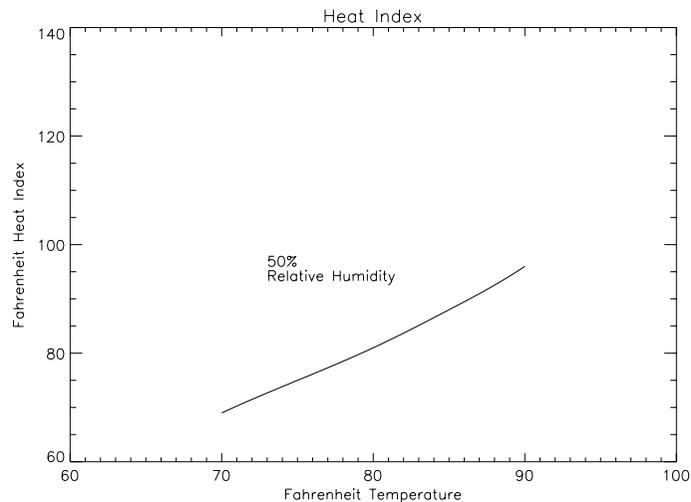
```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Compile the `prep_plot` procedure. It is used to reset plotting parameters and system variables:

```
WAVE> .RUN prep_plot
```

**Step 3** Run the sample program `heat.pro` without arguments. Enter:

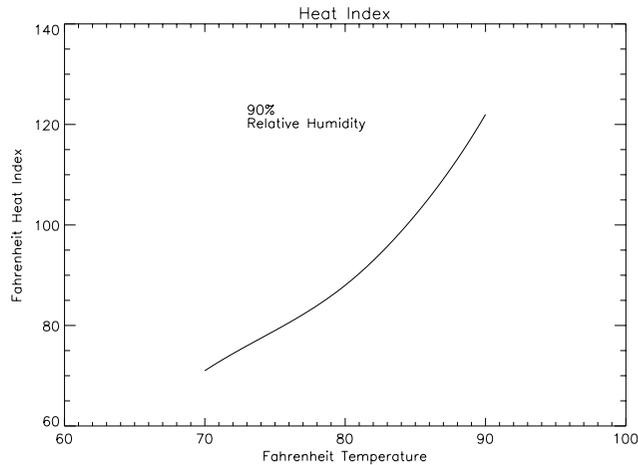
```
WAVE> heat
```



**Figure 9-2** PV-WAVE plots the heat index vs. temperature, using the default 50 percent humidity

**Step 4** Use a value of 90 for percent humidity. Enter:

WAVE> heat, 90



**Figure 9-3** The plot shows the heat index for the entire range of temperatures at 90 percent humidity.

## Example Program Listing: heat

The following is a complete listing of the program `heat`. This code can also be found online in the following file, where `<maindir>` is the main directory where PV-WAVE is installed:

`<maindir>\docs\tutorial\code\advantage\heat.pro`

```
; heat.pro
PRO READ_HEAT_DATA
; Read the file heat.dat and parse the data file into
; appropriate variables
COMMON heat_data, temperatures, humidities, heat_values
OPENR, unit, !Data_Dir + 'heat.dat', /Get_LUN
RMF, unit, heat_data, 12, 8
FREE_LUN, unit
temperatures = heat_data(0, 1:7)
humidities = heat_data(1:11, 0)
heat_values = heat_data(1:11, 1:7)
END
;
FUNCTION CREATE_SPLINE, x, y
```

```

; Calculate a spline function based on interpolated values
; using the BSINTERP and SPVALUE functions
COMMON heat_data, temperatures, humidities, heat_values
heatindex = BSINTERP(humidities, REFORM(temperatures, 7), $
heat_values)
z = FLTARR(100)
FOR i = 0, 99 DO z(i) = SPVALUE(x(i), y(i), heatindex)
RETURN, z
END
;
PRO PLOT_HEAT_DATA, humidity
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
prep_plot
WINDOW, 0, XSize = 875, YSize = 700, $
XPos = 3, YPos = 30, $
Title = 'PV-WAVE'
LOADCT, 4, /Silent
; Plot the temperature verses the heat index for the
; specified relative humidity
COMMON heat_data, temperatures, humidities, heat_values
; Plot over temperature range of 70-90 degrees
y = 70 + FINDGEN(100)/99*20
; Create the plot without data values
PLOT, y, XRange = [60, 100], YRange = [60, 140], $
/NoData, Title = 'Heat Index', $
XTitle = 'Fahrenheit Temperature', $
YTitle = 'Fahrenheit Heat Index', $
Thick = 3, CharSize = 3
; Create a vector of the Spline calculation
x = MAKE_ARRAY(100, Value = humidity)
z = CREATE_SPLINE(x, y)
;
; Overplot new data on the plot
OPLOT, y, z
;
; Add annotation
XYOUTS, y(10) + 1, z(99), $
STRTRIM(STRING(humidity), 2) + $
'!CRelative Humidity', Size = 2
WAIT, 8
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
END
;
PRO HEAT, humidity
; This is the main program loop. The procedure

```

```

;is called with the relative humidity,
;for which the heat index should be
;calculated. For example, to plot the
;heat index curve for a relative humidity
;of 50%, call the procedure like this
;from within PV-WAVE:
;WAVE> .run heat.pro
;WAVE> heat, 60
; Note: There is little error checking in this program
;
IF (N_Params() EQ 0) THEN humidity = 50
READ_HEAT_DATA
PLOT_HEAT_DATA, humidity
prep_plot
END

```

---

## ***Creating and Plotting a Parametric Cubic Spline Interpolant***

The example program `para_spline` illustrates use of the PV-WAVE:IMSL Mathematics cubic spline interpolation and evaluation procedures in order to create a parametric cubic spline interpolant and plot from a set of  $(x, y)$  data points. In this example, a parameter,  $t$ , is introduced, and two interpolants are produced,  $x(t)$ , and  $y(t)$ . The parameter  $t$  is assigned equally-spaced values on the interval  $[0, 1]$ .

The two interpolants are evaluated, producing the plot of  $(x(t), y(t))$ . You control the number of spline evaluation points to use in order to achieve the desired smoothness.

Demonstrate the use of the program `para_spline.pro` by plotting the parametric cubic spline defined by the following data set:

$$\left( r(\theta) \cos\left(\theta + \frac{\pi}{2}\right), r(\theta) \sin\left(\theta + \frac{\pi}{2}\right) \right)$$

where,

$$\theta = 0, \dots, 20\pi$$

$$r(\theta) = e^{\cos(\theta)} - 2\cos(4\theta) + \sin^5\left(\frac{\theta}{12}\right)$$

## Running the Example

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

**Step 1** If you are not in the `advantage` directory, go to it by entering:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Compile the `prep_plot` procedure. It is used to reset plotting parameters and system variables:

```
WAVE> .RUN prep_plot
```

**Step 3** Enter:

```
WAVE> TwenPi = 20*!Pi
```

```
WAVE> Theta = TwenPi*FINDGEN(200)/199
```

```
WAVE> r = EXP(COS(Theta)) - $  
      2*COS(Theta*4) + (SIN(Theta/12))^5
```

```
WAVE> x = r*COS(Theta+!Pi/2)
```

```
WAVE> y = r*SIN(Theta+!Pi/2)
```

```
WAVE> para_spline, x, y, Npts = 200
```

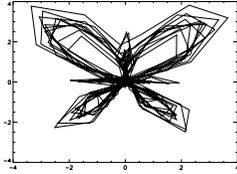
The plot shows 200 connected points.

---

**NOTE** `!Pi` is the PV-WAVE system variable that contains the value of Pi.

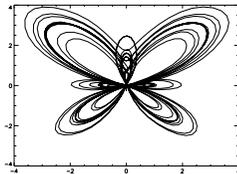
---

```
WAVE> WDELETE
```



```
WAVE> para_spline, x, y, Npts = 1000
```

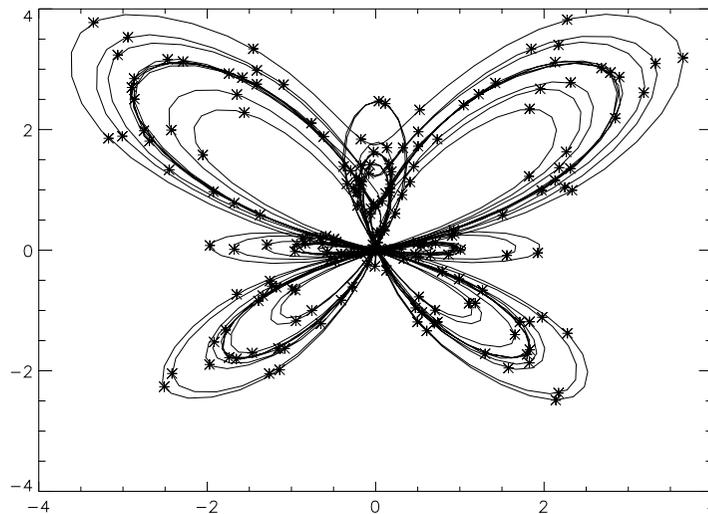
The plot shows 1,000 connected points.



```
WAVE> Oplot, x, y, PSym = 2
```

Asterisks are drawn over the points.

```
WAVE> WDELETE
```



**Figure 9-4** The plot shows asterisks are drawn over 1000 the points.

## Example Program Listing: para\_spline

The following is a complete listing of the program `para_spline`. This code can also be found online in the following file, where `<maindir>` is the main directory where PV-WAVE is installed:

```
<maindir>\docs\tutorial\code\advantage\para_spline.pro
```

```
; para_spline.pro
; Usage: para_spline, xdata, fdata, npts = npts
; npts is an integer value indicating the number of times to
; evaluate the resulting interpolant. This number affects the
; smoothness of the resulting plot.
;
PRO para_spline, x, y, npts = npts
;
prep_plot
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
;
nxy = N_ELEMENTS(x)
IF NOT(KEYWORD_SET(npts)) THEN npts = 5*nxy
;
t = FINDGEN(nxy)/(nxy -1)
;
xspline = CSINTERP(t, x)
yspline = CSINTERP(t, y)
t = FINDGEN(npts)/(npts -1)
;
LOADCT, 4, /Silent
WINDOW, 0, XSize = 875, YSize = 700, $
XPos = 3, YPos = 30, $
Title = 'PV-WAVE'
PLOT, SPVALUE(t, xspline), SPVALUE(t, yspline)
END
;
```

---

## Estimating Area with a Random Number Generator

The example program `monte` uses the random number generator procedure, `RANDOM`, to estimate the area of a quarter-circle. `RANDOM` is available in both `PV-WAVE:IMSL Mathematics` and `PV-WAVE:IMSL Statistics`.

Ordered pairs,  $(x, y)$ , are generated inside a unit square. The area of the inscribed quarter circle is  $\pi/4$ . The `monte.pro` file uses the `RANDOM` function to carry out a Monte Carlo simulation to estimate the area of a quarter circle of radius one.

You provide, as input, the number of Monte Carlo trials to run. The trials produce random values. The resulting area estimate, error and relative error are displayed. A scatter plot is shown to visually indicate how well the simulation approximated the desired area.

### Running the Example Program

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have either the `PV-WAVE:IMSL Mathematics` or `Statistics` installed to run this example.

---

**Step 1** If you are not in the `advantage` directory, go to it by entering:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Run the sample program using 5000 trials. Enter:

```
WAVE> monte, 5000
```

The program returns something similar to:

```
Exact area of unit quarter circle = 0.785398
Estimate of area                   = 0.787000
Absolute Error                      = 0.00160182
Relative Error                      = 0.00203949
```

and then plots the data.

**Step 3** Try different values, such as 10,000 and 50,000, and see the results.

## Example Program Listing: monte

The following is a complete listing of the program monte. This code can also be found online in the following file, where <maindir> is the main directory where PV-WAVE is installed:

```
<maindir>\docs\tutorial\code\advantage\monte.pro
```

```
; monte.pro
; This program uses the PV WAVE random number
; generator procedure, Random, to carry out a Monte Carlo
; simulation to estimate the area of a quarter circle of
; radius one
;
; USAGE: monte, ntrial
; where ntrial is the number of random ordered pairs to
; generate
;
PRO monte, ntrial
;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
;
; Generate ntrial ordered pairs
x = RANDOM(ntrial)
y = RANDOM(ntrial)

; Determine the distance between the origin and each point
z = x^2 + y^2
;
; Keep all (x, y) points within the quarter circle
x = x(WHERE(z LE 1, Count))
y = y(WHERE(z LE 1, Count))
;
; Calculate the area (Probability of an (x, y)
; point in the quarter circle)
prob = FLOAT(Count)/FLOAT(ntrial)
; Exact area is pi/4
exact = !Pi/4
;
PRINT, 'Exact area of unit quarter circle', exact
PRINT, 'Estimate of area          =      ', prob
PRINT, 'Absolute Error            =      ', abs(prob-exact)
PRINT, 'Relative Error            =      ', abs(prob-exact)/exact
```

```

;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
TEK_COLOR
WINDOW, 0, XSize = 875, YSize = 700, $
XPos = 3, YPos = 30, $
Title = 'PV-WAVE Monte Carlo Example'
;

; Plot the x-y axis and title
PLOT, y, x, /NoData, XRange = [0, 1.2], $
YRange = [0, 1.2], Color = 0, $
Title = '!17Area Estimate of Unit Quarter '$
+ 'Circle Exact = !4p!17/4!9N!17 0.7853', $
CharSize = 1.5, Back = 1.0
;

; Plot the (x, y) values within the quarter circle
OPLOT, y, x, PSym = 2, Color = 2
XPoint = .60
YPoint = .90 & WAIT, 10
;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
END

```

---

## Plotting a Fourier Sine Series

The example program `fourier` illustrates a 2D function plot. The Fourier Sine series for  $f(x) = 1$  for  $x$  on the interval  $[0, 1]$  is given by:

$$\sim f(x) \sum_{n=1}^{\infty} \frac{4}{(2n+1)\pi} \sin((2n+1)\pi x)$$

The procedure, `fourier`, plots the  $n$ th partial sums of the Fourier Series. It uses the `RANDOM` function, which is available in both `PV-WAVE:IMSL Mathematics` and `PV-WAVE:IMSL Statistics`.

### Running the Example

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have either `PV-WAVE:IMSL Mathematics` or `PV-WAVE:IMSL Statistics` installed to run this example.

---

**Step 1** If you are not in the `advantage` directory, go to it by entering:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Create a window for this size plot:

```
WAVE> WINDOW, 0, XSize = 875, YSize = 700, $
    ► XPos = 30, YPos = 5, $
    ► Title = 'PV-WAVE Fourier ' + 'Sine Series'
```

**Step 3** Run the `fourier.pro` program. Enter:

```
WAVE> FOR i = 2,20,2 DO BEGIN fourier, i & $
    ► WAIT, 1
```

PV-WAVE plots the series by twos, pausing for one second between each series.

WAVE> WDELETE

## Example Program Listing: fourier

The following is a complete listing of the fourier program. This code can also be found online in the following file, where <maindir> is the main directory where PV-WAVE is installed:

<maindir>\docs\tutorial\code\advantage\fourier.pro

```
; fourier.pro
; This program evaluates the Nth
; Partial sums to the Fourier Sine series for f(x) = 1
; USAGE: fourier, npartial
; npartial = number of partial sums to evaluate and plot
;
PRO fourier, npartial
;
; Read in the Visual Numerics logo
status = DC_READ_TIFF (!Data_Dir + 'vni_smal.tif', $
    vni_s, ImageWidth = XSize, ImageLength = YSize)
;
; Set initial sum vector to zero
sum = MAKE_ARRAY(200, value = 0)
;
; Use 200 points on the interval [0, 1]
x = FINDGEN(200)/199
;
red = RANDOM(150) & red(0) = 0 & red(149) = 0
green = RANDOM(150) & green(0) = 0 & green(149) = 1
blue = RANDOM(150) & blue(0) = 0 & blue(149) = 0
TVLCT, 255*red, 255*green, 255*blue
;
TEK_COLOR
PLOT, x, /NoData, XRange = [0, 1], $
    YRange = [0, 1.5], YMargin = [4, 7], $
    Back = 0, color = 5, $
    Title = '!17Nth Partial Sums for ' $
    + 'Sine Series of f(x) = 1 !C N = ' $
    + string(npartial), CharSize = 2
; Plot the N partial sums
FOR i = 0, npartial -1 DO BEGIN
temp = !Pi*(2*i +1)
```

```
sum = sum + 4/temp*sin(temp*x)
OPLOT, x, sum, Color = i
ENDFOR
TV, vni_s, 0, Order = 1
END
```

---

## Plotting a Shaded Surface of a Scattered 2D Interpolant

The program `scattered` uses the PV-WAVE:IMSL Mathematics procedure, `SCAT2DINTERP`. From a file, the scattered  $x$ ,  $y$  values and the associated  $z$  value are read for each pair. The resulting interpolant is plotted as a shaded surface.

### Running the Example

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

**Step 1** If you are not in the `advantage` directory, go to it by entering:

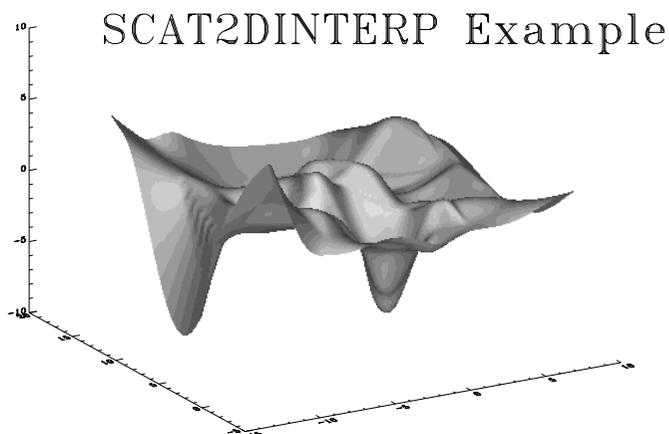
```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Compile the `prep_plot` procedure. It is used to reset plotting parameters and system variables:

```
WAVE> .RUN prep_plot
```

**Step 3** Run the sample program. Enter:

```
WAVE> scattered
```



**Figure 9-5** The shaded surface of the interpolant appears.

## Example Program Listing: scattered

The following is a complete listing of the scattered program. This code can also be found online in the following file, where <maindir> is the main directory where PV-WAVE is installed:

<maindir>\docs\tutorial\code\advantage\scattered.pro

```
; scattered.pro
PRO scattered
prep_plot
CLOSE, 1
OPENR, 1, !Data_Dir + 'scat.dat'
; Read x, y and function values in matrix A
RMF, 1, A, 50, 3
CLOSE, 1
; Create array to contain the x, y values
x = FLTARR(2, 50)
x(0, *) = A(*, 0)
x(1, *) = A(*, 1)
; Create the desired x grid points
XGrid = -12 + 20*FINDGEN(100)/99
; Create the desired y grid points
YGrid = -3 + 19*FINDGEN(100)/99
; Array containing the function values
fdata = A(*, 2)
; Call SCAT2DINTERP to produce interpolant
result = SCAT2DINTERP(x, fdata, XGrid, YGrid)
;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
WINDOW, 0, XSize = 875, YSize = 700, $
    XPos = 3, YPos = 30, $
    Title = 'PV-WAVE Scattered 2D Interpolant Example'
LOADCT, 15, /Silent
;
; Plot the interpolant as a shaded surface
SHADE_SURF, result, XGrid, YGrid
;
ss = '!17SCAT2DINTERP Example'
XYOUTS, .2, .81, ss, Size = 3, Color = 248, /Normal, /NoClip
;

; Read in the Visual Numerics logo
status = DC_READ_TIFF (!Data_Dir + 'vni_smal.tif', $
```

```
        vni_s, ImageWidth = XSize, ImageLength = YSize)
TV, vni_s, 0, Order = 1 & WAIT, 8
;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
prep_plot
END
```

---

## Using the Lorenz Attractor

The example program `lorenz` is an example of use of the PV-WAVE:IMSL Mathematics function ODE to solve a system of ordinary differential equations. The example used is the Lorenz attractor.

The Lorenz model is an autonomous set of dissipative first-order differential equations. The system can be represented as:

$$\frac{d\vec{y}}{dt} = F(\vec{y})$$

where the derivative components of  $\vec{y}$  are:

$$\frac{dy_0}{dt} = \sigma(y_0 - y_1)$$

$$\frac{dy_1}{dt} = -y_0 y_2 + r y_0 - y_1$$

$$\frac{dy_2}{dt} = -y_0 y_1 - b y_2$$

The dissipative nature of the equations means that for an arbitrary volume element  $V$ , enclosed by the surface  $S$  evolving from the solution of the differential equation, the volume contracts in time, or  $dV/dt < 0$ .

For the solution, choose  $\sigma = 10$ ,  $r = 28$ , and  $b = 8/3$ . Allow for a maximum of 20,000 time steps, and ask for the return of the solution for 50,000 points in the interval  $t = [0, 150]$ .

The initial condition is  $y(0, 0, 0) = (0, 0.01, 0)$ .

## Running the Example

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

**Step 1** If you are not in the advantage directory, go to it by entering:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Compile the `prep_plot` procedure. It is used to reset plotting parameters and system variables:

```
WAVE> .RUN prep_plot
```

**Step 3** Run `lorenz.pro`. Enter:

```
WAVE> lorenz
```

The connected points are shown in [Figure 9-1](#) on page 155.

## Example Program Listing: lorenz

The following is a complete listing of the program `lorenz`. This code can also be found online in the following file, where `<maindir>` is the main directory where PV-WAVE is installed:

```
<maindir>\docs\tutorial\code\advantage\lorenz.pro
```

```
; lorenz.pro
; Function used by ODE
FUNCTION f, t, y
COMMON CONSTANTS, a, b, r
    yp = y
    yp(0) = a*(y(1) - y(0))
    yp(1) = -y(0)*y(2) + r*y(0) - y(1)
    yp(2) = y(0)*y(1) - b*y(2)
RETURN , yp
END

; Main driver
PRO lorenz
prep_plot
```

```

COMMON CONSTANTS, a, b, r
    a = 10
    b = 8./3
    r = 28
ntime = 50000
time_range = 150
max_steps = 20000
t = FINDGEN(ntime)/(ntime-1)*time_range
y = [.0, .01, 0]
y = ODE(t, y, 'f', Max_Steps = max_steps)
;
; Plot the results:
;   1. Make a big window
;   2. Set up a transformation matrix using SURFACE
;   3. Plot the data with PLOTS
;   4. Use XYOUTS to place title
;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
WINDOW, 0, XSize = 875, YSize = 700, $
    XPos = 3, YPos = 30, $
    Title = 'PV-WAVE Demonstrates the Lorenz Attractor'
LOADCT, 4, /Silent
;
SURFACE, FLTARR(2, 2), /NoData, /Save, CharSize = 2, $
    XRange = [MIN(y(0, *)), MAX(y(0, *))], $
    ZRange = [MIN(y(1, *)), MAX(y(1, *))], $
    YRange = [MIN(y(2, *)), MAX(y(2, *))].
; Setting PSym = 0 plots data as connected lines
PLOTS, y(0, *), y(2, *), y(1, *), PSym = 0, /T3d
ss = '!17Lorenz Attractor'
XYOUTS, .2, .8, ss, Size = 3, Color = 227, /Normal, /NoClip
WAIT, 8
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
prep_plot
END

```

---

## Using Multiple Regression Capabilities

The example program `mreg` is designed as an introduction to the multiple regression capabilities that are built into PV-WAVE:IMSL Statistics. For each data set, the demo plots the data and fits a least-squares surface. The model used for all data sets is

$$y = b_0 + b_1*x_1 + b_2*x_2 + b_3*x_1*x_2 + b_4*x_1^2 + b_5*x_2^2 + \text{error}$$

### Running the Example

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have PV-WAVE:IMSL Statistics installed to run this example.

---

**Step 1** If you are not in the `advantage` directory, go to it by entering:

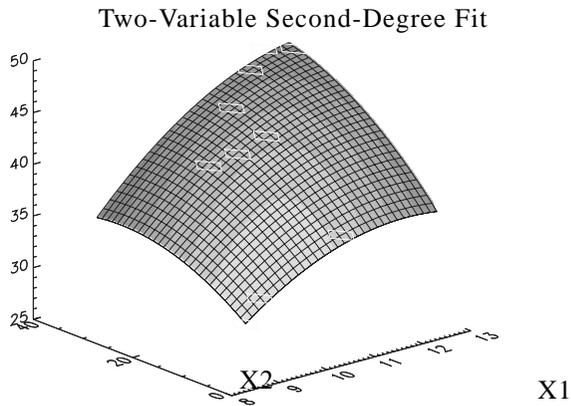
```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Compile the `prep_plot` procedure. It is used to reset plotting parameters and system variables:

```
WAVE> .RUN prep_plot
```

**Step 3** Run `mreg.pro`. Enter:

```
WAVE> mreg
```



**Figure 9-6** The multiple regression plot appears. The complete program is listed in the next section.

### Example Program Listing: mreg

The following is a complete listing of the program `mreg`. This code can also be found online in the following file, where `<maindir>` is the main directory where `PV-WAVE` is installed:

`<maindir>\docs\tutorial\code\advantage\mreg.pro`

```

; mreg.pro
PRO mreg
;
x1 = FLTARR(10, 5)
x1(*, 0) = [8.5, 8.9, 10.6, 10.2, $
           9.8, 10.8, 11.6, 12.0, $
           12.5, 10.9]
x1(*, 1) = [2.0, 3.0, 3.0, 20.0, $
           22.0, 20.0, 31.0, 32.0, $
           31.0, 28.0]
x1(*, 2) = x1(*, 0)*x1(*, 1)
x1(*, 3) = x1(*, 0)*x1(*, 0)
x1(*, 4) = x1(*, 1)*x1(*, 1)
y = [30.9, 32.7, 36.7, 41.9, $
     40.9, 42.9, 46.3, 47.6, $
     47.2, 44.0]
ax = 30

```

```

az = 30
color_points = .98*!D.N_Colors
color_grid   = .12*!D.N_Colors
color_title  = .98*!D.N_Colors
x1(*, 2) = x1(*, 0)*x1(*, 1)
x1(*, 3) = x1(*, 0)*x1(*, 0)
x1(*, 4) = x1(*, 1)*x1(*, 1)
;
; Setup vectors for surface plot. These will be nxgrid x nygrid elements each, evenly spaced over the range of the data in x1(*, 0) and x1(*, 1)
nxgrid = 30
nygrid = 30
ax1 = MIN(x1(*, 0)) + (MAX(x1(*, $
    0))- MIN(x1(*, $
    0)))*FINDGEN(NXGrid)/(nxgrid-1)
ax2 = MIN(x1(*, 1)) + (MAX(x1(*, $
    1))- MIN(x1(*, $
    1)))*FINDGEN(nxgrid)/(nxgrid-1)
;
; Compute regression coefficients
coefs = MULTIREGRESS(x1, y, residual = Resid)
one_solution = 1
;
; Create two-dimensional array of evaluations of the regression
; model at points in grid established by ax1 and ax2
z = FLTARR(nxgrid, nygrid)
FOR i = 0, nxgrid-1 DO BEGIN
    FOR j = 0, nygrid-1 DO BEGIN
        z(i,j) = coefs(0) + $
            coefs(1)*ax1(i) $
            + coefs(2)*ax2(j) $
            + coefs(3)*ax1(i)*ax2(j) $
            + coefs(4)*ax1(i)^2 $
            +coefs(5)*ax2(j)^2
    ENDFOR
ENDFOR
;
!P.TickLen = .03
!P.CharSize = 2
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
WINDOW, 0, XSize = 875, YSize = 700, $
    XPos = 3, YPos = 30, $
    Title = 'PV-Wave Regression Example'
LOADCT, 2, /Silent

```

```

;
SHADE_SURF, z, ax1, ax2, /SAVE, $
  XRange = [MIN(ax1), MAX(ax1)], $
  YRange = [MIN(ax2), MAX(ax2)], $
  ZRange = [MIN(z) -2, MAX(z)+2], $
  AX = 30, AZ = 40
SURFACE, z, ax1, ax2, /NoErase, $
  XRange = [MIN(ax1), MAX(ax1)], $
  YRange = [MIN(ax2), MAX(ax2)], $
  ZRange = [MIN(z) -2, MAX(z)+2], $
  AX = 30, AZ = 40, $
  Color = color_grid
PLOTS, x1(*, 0), x1(*, 1), y, /T3D, $
  PSym = 4, SymSize = 5, $
  Color = color_points, $
  Thick = 2
XYOUTS, 130, 720, /Device, CharSize = 3.0, $
  '!17Two-Variable Second-Degree Fit', $
  Color = color_title
XYOUTS, 610, 90, /Device, CharSize = 2.0, $
  'X1', Color = color_title
XYOUTS, 215, 90, /Device, CharSize = 2.0, $
  'X2', Color = color_title & WAIT, 10
WHILE (!D.Window GE 0) Do WDelete, !D.Window
prep_plot
END

```



## *Writing Programs*

One of the best methods to learn how to use a programming language is to study examples and then copy and modify some short programs containing relevant code. This chapter not only provides building blocks for more detailed programs, it also gives you rapid, hands-on experience in using some of the powerful mathematical and statistical routines in PV-WAVE:IMSL Mathematics and PV-WAVE:IMSL Statistics.

The exercises in this chapter show you how to execute sample programs. All the programs referenced in this chapter are located in the following directory, where `<maindir>` is the main directory where PV-WAVE is installed.

```
<maindir>\docs\tutorial\code\advantage
```

As with the previous chapter, all the examples are written in ASCII text and have a `.pro` extension. The examples can be printed, edited, and read.

---

## Creating an Executable Program File

It is easy to create an executable PV-WAVE program. Executable programs can be created and run directly from the WAVE> prompt. Or, you can write a program using a text editor, save the program in a file, and then run it.

Using a text editor, you can easily compose and edit the file. When you create the file from within PV-WAVE, each line is compiled as you enter it.

For detailed information on creating and running programs directly from the WAVE> prompt, see *PV-WAVE User's Guide*. The *Tutorial* does not cover this topic; instead, the *Tutorial* focuses on how to create programs using a text editor.

### Example Program: quad

This program, which you will save as `quad`, uses the PV-WAVE:IMSL Mathematics quadrature function, `INTFCN`, to evaluate the following integral:

$$\int_0^{\pi} \ln(\sin(x)) dx = -\pi \ln(2)$$

Note that the integrand is singular at both end points. The PV-WAVE:IMSL Mathematics function `INTFCN` can handle endpoint singularities, and also allows you to specify internal singularities as well. First, the value of the integral is computed and compared to the exact value. This information is returned in PV-WAVE when the program is run. Next, the function  $\ln \sin(x)$  in the interval  $x = [0.01, \pi - 0.01]$  is plotted in a PV-WAVE window.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

**Step 1** Type the following program into any text editor, paying careful attention to the punctuation marks and special characters.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

```
FUNCTION f, x
  RETURN, ALOG ( SIN ( x ) )
```

```

END
PRO QUAD
  Exact = -!Pi * ALog(2)
  Result = INTFCN("f", 0, !Pi)
  PRINT, 'Computed answer = ', result
  PRINT, 'Exact answer = ', exact
  PRINT, 'Relative error = ', $
    ABS((exact - result)/exact)
  x = .01 + FINDGEN(101)/100*(!Pi - .02)
  y = f(x)
  PLOT, x, y, Title='!17Graph of ln(SIN(x))', $
    XTitle = 'x', YTitle = 'y', $
    CharSize = 2, Thick = 2
END

```

**Step 2** Save the file you just typed as an ASCII text file and name it `quad.pro`.

**Step 3** Compile the program. At the `WAVE>` prompt, enter:

```
WAVE> .RUN quad.pro.
```

If you did not make any typing mistakes, and the program compiled successfully, you will see two messages stating that PV-WAVE compiled the module *F* and the module *quad*.

If you made any errors when you typed the file, you will probably receive an error message that will aid you in locating the error.

If you made any errors, return to the text editor and correct them, re-save the file, and recompile.

**Step 4** To run the `quad.pro` program, type:

```
WAVE> quad
```

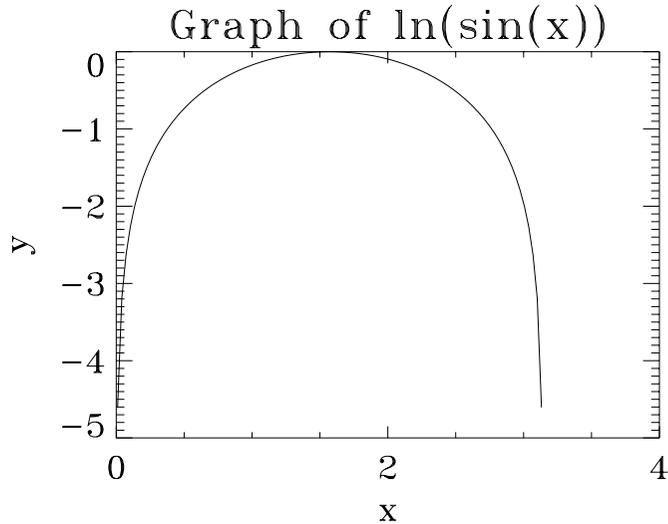
The program runs, delivering the information you requested it to print:

```
Computed answer = -2.17759
```

```
Exact answer = -2.17759
```

```
Relative error = 1.09488e-07
```

PV-WAVE then plots the result, labeling it as specified by the *Title* keyword.



**Figure 10-1** The plot obtained by running the program, `quad.pro`.

---

**NOTE** To run any previously compiled program, simply type the name of the program at the `WAVE>` prompt. To obtain a list of names of compiled programs, enter the `INFO` command at the `WAVE>` prompt. This command returns a listing of all saved (compiled) procedures and functions.

---

**Step 5** Close the window. Enter:

```
WAVE> wdelete
```

### Example Program: Solve an Ordinary Differential Equation

This example uses the `PV-WAVE:IMSL` Mathematics function `ODE` to solve a simple, non-linear, one-dimensional, ordinary differential equation.

The simplest population model assumes that population growth is directly proportional to the size of the population:

$$\frac{dy}{dt} = \epsilon y$$

where  $t$  is the time in years,  $y$  is the population at time  $t$ , and  $\varepsilon > 0$  is the growth rate. The solution to this equation is:

$$y(t) = y_0 e^{\varepsilon t}$$

where  $y(0) = y_0$ . This gives an exponentially growing population.

However, eventually, resources will limit population growth, so that the growth rate is not constant but a function of the population. Thus, for small populations, growth will occur due to availability of resources, and for large populations the limitations on resources will produce an inhibitory effect on population. A simple function meeting these criteria is achieved by setting the growth rate to be:

$$\varepsilon - \sigma y$$

where  $\varepsilon, \sigma > 0$ . The ODE then becomes

$$\frac{dy}{dt} = \varepsilon y - \sigma y^2$$

The exact solution is given by:

$$y(t) = \frac{\varepsilon}{\sigma + \frac{\varepsilon - \sigma y_0}{y_0} e^{-\varepsilon t}}$$

As  $t \rightarrow \infty$ , then  $y \rightarrow \varepsilon/\sigma$ .

For this example, choose  $\varepsilon = 0.07$  and  $\sigma = 0.001$ , and solve the ODE for various values of initial conditions. The resulting solutions are plotted. Note that the line  $y = 0.07/0.001 = 70$  is the limiting curve, regardless of the initial population value. For each initial condition, the population at time  $t = 40$  is given.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

**TIP** If you prefer to just execute the program and not type it, enter the following three commands (otherwise, begin with Step 1 below):

---

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

```
WAVE> .run population_ode
```

```
WAVE> population_ode
```

**Step 1** Enter the following program using a text editor:

---

**NOTE** Comments are preceded by semicolons (;). The plus character (+) is needed when two character strings are concatenated into one. Character strings can be entered only one line at a time; therefore, any string longer than one line must be broken into two by using a plus sign to combine them. The plus sign can be placed at the end of the first line or at the beginning of the second.

---

```
; Define a function to evaluate the right-hand side of the ODE
FUNCTION ode_1, t, y
  epsilon = 0.07
  sigma = 0.001
  RETURN, epsilon*y - sigma*y*y
END
;
```

```

; Main program
PRO POP_ODE
;
; The variable t represents time in years
t = [0, 40]
t = FINDGEN(41)
; Set up a matrix to hold the ODE solution
; corresponding to runs with 10
; different initial solutions
yprime = FLTARR(10, 41)
WINDOW, 0, XSize = 875, YSize = 800, $
XPos = 3, YPos = 30, $
Title = 'PV-WAVE ODE Integrator'
; Set up the axes for a plot of the solutions
PLOT, yprime(0, *), t, YRange = [0, 100], $
XRange = [0, 40], /NoData, Title = '!17ODE Population Model', $
XTitle= 'Time t in Years', YTitle= 'Population y in millions', $
CharSize = 2
; For each solution print the initial
;condition and final value
PM, 'Initial Condition y(0) Population at t = 40'
; Loop over 10 different initial conditions
FOR i = 10, 100, 10 DO BEGIN
    yinitial = [i]
    index = i/10 -1
; Call PV-WAVE function ODE
yprime(index,*) = ODE(t, yinitial, 'ode_1')
; Print initial and final solutions
PRINT, yinitial(0), '      ', yprime(index, 40)
; Plot solutions
OPLOT, t, yprime(index, *)
ENDFOR
END

```

**Step 2** Save the file as `pop_ode.pro`.

Now you are ready to compile and run the program.

**Step 3** Compile the `prep_plot` procedure. It is used to reset plotting parameters and system variables:

```
WAVE> .RUN prep_plot
```

**Step 4** Compile the program. Type:

```
WAVE> .run pop_ode
```

The program is compiled.

---

**NOTE** A runtime error may stop the program at a level below the main level. To return to the main program level, type RETALL at the WAVE> prompt.

---

If you made a typing error, you will receive an error message. In this case, go back to your text editor and correct the mistake; save the program; then run the program as in the previous step. You do not need to close (exit) PV-WAVE.

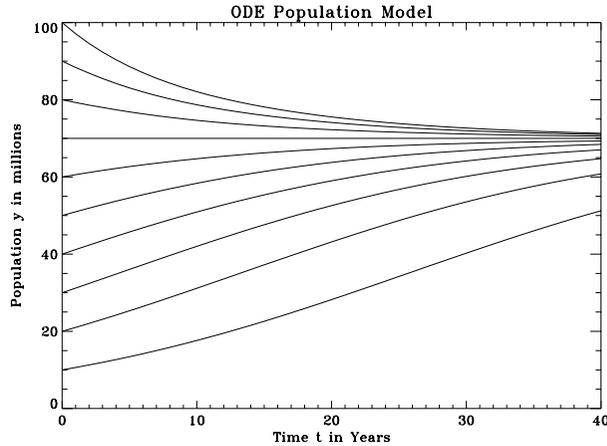
**Step 5** Now you can use your program. At the WAVE> prompt, type:

```
WAVE> pop_ode
```

The program returns the values for the *Initial Condition* and for the *Population* at each condition:

	Initial Condition $y(0)$	Population
at $t=40$		
10	51.2446	
20	60.8025	
30	64.7429	
40	67.0317	
50	68.4430	
60	69.3272	
70	70.0000	
80	70.5622	
90	70.9890	
100	71.3117	

then it plots the values:



**Figure 10-2** The plotted solution to the ordinary differential equation with population data.

---

## Creating and Using Batch Files

Earlier in this chapter, you created a `.pro` file using a text editor. In this section, you will create an executable `.pro` file by using the `JOURNAL` command to record and save in a file everything you enter at the `WAVE>` prompt. The file that is created is called a *batch file*.

### About Batch Files

Unlike `PV-WAVE` program files, batch files are interpreted as though they were entered from the keyboard. In other words, each line of a batch file is independently compiled and executed before proceeding to the next line. Batch files can be created with any text editor or with the `PV-WAVE JOURNAL` facility. As you will see, batch files are executed from the `WAVE>` prompt with the `@` command.

### Using `JOURNAL` to Save Commands

The `JOURNAL` procedure enables you to save to a file all the text you enter at the `PV-WAVE` prompt. You can use this command to create a file that contains a record of an interactive `PV-WAVE` session. In this way, you can create a complete description of your `PV-WAVE` session. These recorded commands constitute a batch file that can then be run to “replay” the session.

Files created by using the JOURNAL command will, like any batch file, be executed a line at a time, as opposed to programs, which are first compiled in their entirety by PV-WAVE, then run as a complete program.

The syntax for the JOURNAL command is:

```
JOURNAL, 'filename'
```

where *filename* is the name of the journal file to be created.

If no filename is specified, the JOURNAL file will be named `wavesave.pro`. To close the journal file, enter JOURNAL again or exit PV-WAVE.

## Example Program: gam\_bess.pro

You will use JOURNAL to create and save a program called `gam_bess.pro`. This program illustrates the use of two functions from PV-WAVE:IMSL Mathematics. GAMMA and BESSJ are used to define a surface to be plotted. In this example, you plot the function:

$$f(x, y) = \sqrt{\Gamma(0.01 + |x|)J_0(y)J_1(y)}$$

in the interval,  $x = [-6, 6]$ ,  $y = [-3, 3]$  and use SHADE\_SURF to plot a shaded surface.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

## Open a JOURNAL File

**Step 1** If you are not in one, move to a directory where you have write permission. For example:

```
WAVE> CD, 'pathname'
```

**Step 2** Open a JOURNAL file:

```
WAVE> JOURNAL, 'gam_bess.pro'
```

## Enter Commands at the Command Line

The commands you enter are being recorded directly in the JOURNAL file `gam_bess.pro`.

**Step 1** Generate 100  $x$  values in the interval  $[-6, 6]$ . Type:

```
WAVE> x = -6 + FINDGEN(100)/99*12
```

**Step 2** Generate 100  $y$  values in the interval  $[-3, 3]$ . Type:

```
WAVE> y = -3 + FINDGEN(100)/99*6
```

**Step 3** Create a matrix to hold the  $z$  values. Type:

```
WAVE> z = FLTARR(100, 100)
```

**Step 4** Create a larger window. Type:

```
WAVE> WINDOW, 0, XSize = 875, YSize = 800
```

An empty window appears.

**Step 5** Load color table 6, *Prism*. Type:

```
WAVE> LOADCT, 6
```

PV-WAVE returns a message telling you the color table has been loaded.

**Step 6** Define the surface in the 2D matrix  $z$ , using GAMMA and BESSJ. Type:

```
WAVE> FOR I = 0, 99 DO z(*,i) = $
    ┌─▶ SQRT(ABS(GAMMA(.01 + $
    ┌─▶ ABS(x))*BESSJ(0, y(i))* $
    ┌─▶ BESSJ(1, y(i))))
```

---

**NOTE** A space is required before a \$ when: (a) an equal sign (=) appears immediately before the \$ or (b) when the equal sign (=) falls at the beginning of the following line.

---

**Step 7** Plot a shaded surface of  $x$ ,  $y$ , and  $z$ . Type:

```
WAVE> SHADE_SURF, z, CharSize = 3, $
    ┌─▶ Color = 127
```

A shaded surface of the result is displayed.

Add a line displaying the mathematical function. You can use the *Title* keyword to place the function above the plot, but you also have the ability to display the information across the plot by using XYOUTS.

A string that will display the following function requires text commands capable of producing the characters in the correct graphic relationship:

$$f(x, y) = \sqrt{\Gamma(0.01 + |x|)J_0(y)J_1(y)}$$

For a complete discussion of the text commands used in the next step, refer to the *PV-WAVE User's Guide*.

**Step 8** First, create a string to produce the equation. Type:

```
WAVE> SS = '!18f!6(x, y) = '$
      ► + '(!7C!6(0.01 + !9!!!6x!9!!!6)' $
      ► + 'J!S!i0!r (y)J!s!i1!r (y))!e(1/2)'
```

---

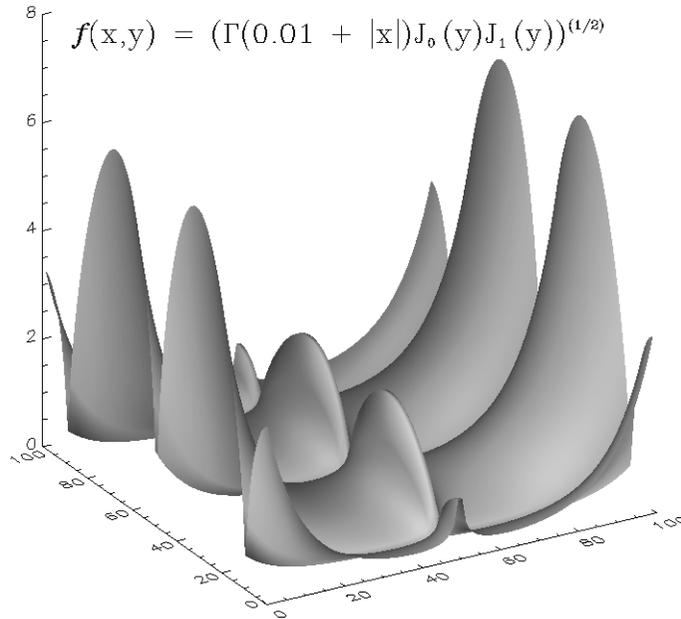
**NOTE** The plus sign (+) is the concatenation character used for text strings in PV-WAVE.

---

**Step 9** Output the string over the plot. Type:

```
WAVE> XYOUTS, .2, .81, SS, Size = 3, $
      ► Color = 80, /Normal, /NoClip
```

The string is displayed across the plot.



**Figure 10-3** The resulting shaded surface plot.

### ***Close the JOURNAL File***

**Step 1** To close and save the JOURNAL file type:

```
WAVE> JOURNAL
```

The gam\_bess.pro file is saved in your current directory.

### ***Running the Batch File***

Now that you have created a batch file, you can run it. You may wish to view the contents first.

**Step 1** View the file with any text editor on your system.

**Step 2** Type:

```
WAVE> @gam_bess.pro
```

---

**NOTE** The @ command is a special command used to execute batch files. As the file is read, each line is compiled and executed one after another, displaying a shaded surface as before.

---

**Step 3** Close the window. Enter:

```
WAVE> WDELETE
```

---

## ***A Comparison of Matrices and Arrays***

Matrices can be referenced differently from arrays by using the functions PM (Print Matrix), RM (Read Matrix), PMF (print matrix file), and RMF (Read Matrix File).

Matrices are used by mathematically-oriented functions and procedures, such as EIG (compute eigenvalues). If  $A$  is a matrix, then  $A(i, j)$  refers to the  $i$ -th row,  $j$ -th column of  $A$ . This conforms to standard mathematical notation. The elements in a matrix are stored columnwise, i.e., the elements of the 0-th column are first, followed by the elements of the 1-st row, etc. A matrix can be converted to an array by taking its transpose.

Other functions and procedures, such as the image display function TV, use arrays. If  $A$  is an array (an image is an array of pixels), then  $A(i, j)$  refers to the pixel whose  $x$  coordinate is related to  $i$  and whose  $y$  coordinate is related to  $j$ . The elements in an array are stored row-wise, i.e., the elements of the 0-th row are first, followed by the elements of the 1-st row, etc. If the array is an image, then a row corresponds to a scan line. Images are stored in scan line order. The main reason for this is to allow the  $x$  subscript to appear first when subscripting images, as is the convention.

PV-WAVE provides specific methods to read arrays and matrices. The PV-WAVE PRINT command returns values according to the array syntax, and the READ command is the corresponding command to read data. The PV-WAVE PM command (*Print Matrix*) and the RM command (*Read Matrix*) follow the matrix notation convention. Both matrices and arrays are read and printed in column-major order unless you use PM, RM, PMF, or RMF, which specify row-major order.

You have the option of using the type of commands that emulate the notation with which you are most comfortable. By being consistent in your choice of commands to read and write your matrix data, you will always find that the PV-WAVE commands work the way you expect.

## Example Program: matrices.pro

The first part of this exercise demonstrates the use of the RM and PM commands; you can best see how these commands work by actually typing the nine short lines within PV-WAVE. This example solves a simple linear system using the PV-WAVE:IMSL Mathematics LUSOL function.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

In this example, you are prompted to enter the 2-by-2 matrix of coefficients,  $A$ , and the 2-by-1 right-hand side of the vector,  $b$ . The systems,  $Ax = b$ , and  $A^T x = b$  are then solved.

**Step 1** Tell PV-WAVE to keep track of your commands under the name `matrices.pro`. Enter:

```
WAVE> JOURNAL, 'matrices.pro'
```

**Step 2** Type the program:

```
WAVE> PM, 'Enter a 2x2 matrix of coeffs, a:'
```

```
WAVE> RM, a, 2, 2
```

PV-WAVE returns:

```
row 0:
```

**Step 3** Input any two numbers, separated by a comma (or a space), such as “3,5” (or 3 5). Then you are prompted to input the values for the next row (row 1). Again, enter two numbers (such as 4, 6), separating them with a comma or a space.

**Step 4** Enter:

```
WAVE> PM, 'Enter the 2x1 vector for the ' $
      + 'right hand side, b:'
```

```
WAVE> RM, b, 2, 1
```

PV-WAVE returns:

```
row 0:
```

**Step 5** Enter:

row 0: 8

row 1: 7

**Step 6** Continue. Enter:

```
WAVE> PM, 'a = ', a, 'b = ', b
```

```
WAVE> x = LUSOL(b, a)
```

```
WAVE> PM, 'x = ', x, 'a#x', a#x
```

```
WAVE> y = LUSOL(b, a, /Transpose)
```

```
WAVE> PM, 'y = ', y, 'TRANSPOSE(a)#y = ', $
```

```
    ┆▶ TRANSPOSE(a)#y
```

**Step 7** Stop the journaling session. Enter:

```
WAVE> JOURNAL
```

The file `matrices.pro` saved in the current directory.

**Step 8** Now run your program. Enter:

```
WAVE> @matrices
```

The program prompts you for input, then returns the values specified in matrix algebra notation. The solution is found only if  $A$  is nonsingular.

## Example Program: linear.pro

This colorful example, `linear.pro`, illustrates use of the PV-WAVE:IMSL Mathematics procedures designed to solve general linear system problems. In this example, you solve the 10-by-10 linear system,  $Ax = b$ , where  $A$  is the tri-diagonal matrix,

$$A = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 \end{bmatrix}$$

and

$$b_i = \sin\left(\frac{2\pi i}{9}\right) \quad i = 0. \dots 9$$

The linear system is solved using the PV-WAVE:IMSL Mathematics function, LUSOL, and the quantity  $\max\{|Ax-b|\}$ , the largest residual, is returned. The solution,  $x$ , is plotted, then the system is solved 100 more times with a different right-hand side vector,  $b$ . Each  $b(i)$  is perturbed by an additive factor of at most 1. The corresponding solution vectors are plotted.

As you examine the code, notice that, in the course of solving the problem 100 times with LUSOL, the factorization of the matrix  $A$  is calculated needlessly each time. A more efficient method would be to use the PV-WAVE:IMSL Mathematics function LUFAC to compute the factorization one time. Then, using the given factorization, use LUSOL each time to only solve the system.

The WAIT command is used within the procedure `linear.pro` to slow the plotting so that you can see each line being plotted.

## Running the Example Program

This section explains how to run the example program; see the next section for a complete listing of the program.

---

**NOTE** You must have PV-WAVE:IMSL Mathematics installed to run this example.

---

**Step 1** Go to the directory containing `linear.pro`. Enter:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code\advantage'
```

**Step 2** Compile the `prep_plot` procedure. It is used to reset plotting parameters and system variables:

```
WAVE> .RUN prep_plot
```

**Step 3** To run the program, enter:

```
WAVE> linear
```

PV-WAVE returns:

```
Maximum residual error = 2.08616e-07
```

and plots the values:

### Example Program Listing: linear

The following is a complete listing of the program `linear`. This code can also be found online in the following file, where `<maindir>` is the main directory where PV-WAVE is installed:

```
<maindir>\docs\tutorial\code\advantage\linear.pro
```

```
; linear.pro
; This example illustrates usage of the PV WAVE
; procedures designed to solve real
; general linear system problems
;
PRO linear
prep_plot
;
; Solve the linear system Ax = b
; A is a tri-diagonal matrix with 2's along the diagonal and
```

```

; 1's along the sub- and super-diagonal
A = FLTARR(10, 10)
FOR i = 0, 9 DO A(i, i) = -2.
FOR i = 0, 8 DO A(i + 1, i) = 1.
FOR i = 0, 8 DO A(i, i + 1) = 1.
;
; Generate values for the right-hand side
b = SIN(2.*!Pi*FINDGEN(10)/9.)
;
; Solve the linear system
x = LUSOL(b, A)
;
; As a measure of accuracy, print the maximum residual
PM, 'Maximum residual error = ', MAX(ABS(A#x - b))
;
; Load 150 colors to use
red = RANDOM(150) & red(0) = 0 & red(149) = 0
green = RANDOM(150) & green(0) = 0 & green(149) = 1
blue = RANDOM(150) & blue(0) = 0 & blue(149) = 0
;
TVLCT, 255*red, 255*green, 255*blue
;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
WINDOW, 0, XSize = 875, YSize = 700, $
XPos = 3, YPos = 30, $
Title = 'PV-WAVE General Linear Systems ' $
+ 'Problem Example'
;
; Plot the solution vector using a black background, green text
PLOT, x, YRange = [-4, 4], Back = 0, Color = 149
WAIT, 3
;
; Solve the system with 100 different right-hand sides
; The PV WAVE random number generator is used to
; perturb each component of b
; by at most +-1.
FOR i = 1, 100 DO OPLOT, $
LUSOL(b + .2*(RANDOM(10) -.5), A), $
Color = i
;
; A more efficient way is to save the factorization of
; the matrix, A.
WAIT, 3
PLOT, x, YRange = [-4, 4], Back = 0, Color = 149
WAIT, 3

```

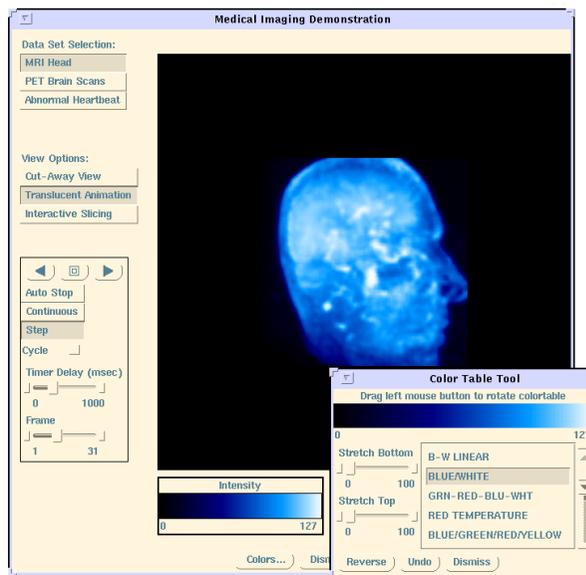
```

;
; Compute the factorization
LUFAC, A, pvt, fac
FOR i = 1, 100 DO BEGIN
OPLOT, LUSOL(b + .2*(RANDOM(10) -.5), $
Pivot = pvt, Factor = fac), $
Color = i, Thick = 2
WAIT, .2
ENDFOR
; Read in the Visual Numerics logo
status = DC_READ_TIFF (!Data_Dir + 'vni_smal.tif', $
vni_s, ImageWidth = XSize, ImageLength = YSize)
TV, vni_s, 0, Order = 1 & WAIT, 10
;
WHILE (!D.Window GE 0) DO WDELETE, !D.Window
prep_plot
END

```

## Animating Data

PV-WAVE can help you visualize your data dynamically by using animation.



**Figure 11-1** In this example, MRI data of a human head is displayed as a series of translucent images, which produces animation that simulates full-circle rotations.

An *animation* is a series of still frames shown sequentially to create the illusion of motion. In PV-WAVE, a series of frames can be represented by a 3D array (for example, a 3D array could hold forty, 300-by-300 pixel images). This chapter shows you how to create arrays of images and play them back as animated sequences.

---

## Previewing This Chapter

To preview some of the programs you will create in this chapter, run the batch file named below.

**Step 1** Use the CD command to move to the subdirectory that contains code for the tutorial. At the WAVE> prompt, enter:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code'
```

**Step 2** Run the chapter preview by entering the following command at the WAVE> prompt:

```
WAVE> @lesson_11
```

The batch file `lesson_11` runs the programs. The instructions for running and quitting the animations are printed in the PV-WAVE window. After a brief pause, the last window is closed.

---

**NOTE** You must type “q” in the PV-WAVE console window to stop each animation. Typing “q” in the graphics window has no effect.

---

---

## Displaying a Series of Images

Create an animation that shows a series of images that represent an abnormal heart-beat. First, read in the images to be displayed. The file `abnorm.dat` holds a series of 16 images of a human heart stored as 64-by-64 byte arrays.

**Step 1** Open the file and prepare it for reading-by-entering the following commands at the WAVE> prompt:

```
WAVE> OPENR, 1, !Data_dir + 'abnorm.dat'
```

This command opens the file `abnorm.dat` for reading.

Define a variable, *h*, as a 64-by-64-by-16 byte array.

**Step 2** Create a variable *h* to hold the images. Enter:

```
WAVE> h = BYTARR(64, 64, 16)
```

**Step 3** Read the images into the variable *h*, using the command, READU, which reads unformatted, or byte, data. Enter:

```
WAVE> READU, 1, h
```

**Step 4** Close the file abnorm.dat by entering:

```
WAVE> CLOSE, 1
```

**Step 5** Load an appropriate color table, *Red Temperature*, by entering:

```
WAVE> LOADCT, 3
```

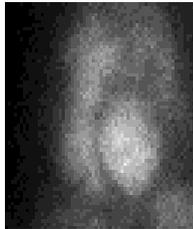
**Step 6** Open a window that will fit the final image. The final image size will be 256-by-256 bytes.

```
WAVE> WINDOW, 1, XSize = 256, YSize = 256
```

**Step 7** Display the first image in the array *h* by entering:

```
WAVE> TVSCL, h(*, *, 0)
```

The small image appears in the lower left corner of the window



The asterisks (\*) in the first two subscript positions tell PV-WAVE to use all of the elements in those positions. Hence, the TV command displays a 64-by-64 byte image.

**Step 8** The image is rather small, so resize each image in the array with bilinear interpolation by entering:

```
WAVE> h = REBIN(h, 256, 256, 16)
```

**Step 9** Redisplay the image:

```
WAVE> TVSCL, h(*, *, 0)
```

Each image in *h* is four times its previous size.

**Step 10** Now use the Standard Library procedure `MOVIE` to display each image in array `h` one after another. Enter:

```
WAVE> MOVIE, h, Order = 0
```

The animation appears.

**Step 11** Press `<s>` to slow the animation, `<f>` to speed it up, and `<q>` to return to the `PV-WAVE` command line.

---

## Animation As a Wire Frame Surface

The data in `abnom.dat` also can be displayed as a series of surface plots.

**Step 1** Create a new array `s` to hold the heartbeat data by entering:

```
WAVE> s = REBIN(h, 32, 32, 16)
```

The variable `s` now holds sixteen 32-by-32 byte versions of the heartbeat images.

`SURFACE` plots are often more legible when made from a resized version of the data set with fewer data points in it.

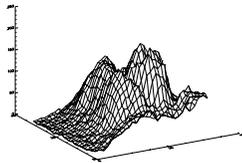
**Step 2** Now create a new window 300-by-300 pixels wide in which to display the images.

```
WAVE> WINDOW, 2, XSize = 300, YSize = 300, $
      Title = 'PV-WAVE Animation'
```

**Step 3** Display the first image in `s` as a wire-mesh surface by entering:

```
WAVE> SURFACE, s(*, *, 0)
```

The surface is drawn in white.



**Step 4** To make the surface plots appear in color, first load 32 distinct colors:

```
WAVE> TEK_COLOR
```

Create a series of surface plots, one for each image in the original data set.

**Step 5** First, create a 3D array *frames* to hold all of the images by entering:

```
WAVE> frames = BYTARR(300, 300, 16)
```

The variable *frames* will hold sixteen, 300-by-300 byte images.

The next command draws each frame of the animation. A surface plot is drawn in the window and then the TVRD command is used to read the image from the plotting window into the *frames* array. The FOR loop is used to increment the array indices. The lines below are actually a single PV-WAVE command.

**Step 6** Draw each animation frame. Enter:

```
WAVE> FOR I=0, 15 DO BEGIN SURFACE, $
    Color = 10, $
    s(*, *, I), ZRange = [0, 250] & $
    frames(0,0,I)=TVRD(0, 0, !D.X_VSize, $
    !D.Y_VSize) & END
```

---

**NOTE** The dollar sign (\$) works as a continuation character in PV-WAVE and the ampersand (&) allows multiple commands in the same line.

---

A series of surface plots is drawn in the window.

---

**NOTE** The *ZRange* keyword is used to keep the surface height at the same scale for each plot. The TVRD function returns the contents of the specified rectangular portion of a displayed image. The system variables, !D.X\_VSize and !D.Y\_VSize, report the size of the current window and each is updated when the window is resized.

---

**Step 7** Now display the new animation by entering

```
WAVE> MOVIE, frames, Order = 0
```

**Step 8** Press <s> to slow the animation, <f> to speed it up, and <q> to return to the PV-WAVE command line.

---

## Rotating the Images in an Animation

The file `nist.dat`, contains data representing the magnitude displacement caused by a shock wave. The next example, `animate.pro`, provides the general framework for creating and then rotating 30 increments of six degrees.

**Step 1** If you have not done so already, use the `CD` command to move to the sub-directory that contains code for the tutorial. At the `WAVE>` prompt, enter:

```
WAVE> CD, GETENV('vni_dir') + '\docs\tutorial\code'
```

If you want to look at the code for this example, use a text editor to open the file `animate.pro`.

The `REBIN` function is used to reduce the 50-by-50 array to 25-by-25. The angle of rotation is assigned by specifying the variable *angle* equal to  $-i * 6.0$ . The surfaces are read as images into the variable *frames* and *frames* is viewed both forward and backward to create a smooth loop.

**Step 1** Compile and run the procedure.

```
WAVE> .RUN animate
```

```
WAVE> animate
```

**Step 2** To quit the program, place the pointer in the plotting window and click the left mouse button.

---

## More Information on Animation

With just a few commands, you have created a number of different types of animation. For more information on animation, see the *PV-WAVE User's Guide* and in the online Help system.

# Index

## Symbols

! (exclamation mark) 78  
# (matrix multiplication operator) 110  
\$ (continuation character) 11, 68  
& (ampersand) 11, 83  
+ (plus character) 78, 190  
, (comma) 11, 67  
/(slash) 137  
; (semicolon) 11

## A

abnorm.dat 206  
aborting, PV-WAVE session 95  
ampersand (&) 11, 83  
animation 51, 205  
annotation, with hardware fonts 75  
arrays  
    compared to matrices 198  
    creating 66  
    definition 12  
    processing techniques 135  
ASCII characters 69  
attributes, of window 94  
ax keyword 120, 124  
az keyword 120, 124

## B

background keyword 73  
batch files 96, 193  
binary data 98  
bottom keyword 124  
boxcar averaging 142  
buffer 74  
Butterworth filter 106

BYTARR 209  
Byte 12  
BYTSCL 85, 131

## C

c\_charsize keyword 117  
c\_colors keyword 119  
c\_label keyword 117, 119  
case sensitivity  
    in PV-WAVE 11  
character string, concatenating 78, 190  
characters  
    hardware drawn 75  
    vector-drawn 75  
charsize keyword 79  
CLOSE 101  
color  
    create custom 151  
    in plots 118  
    keyword 73, 104  
    manipulation tools 118  
    table 72, 146  
    total number varies 72  
    used to enhance images 153  
    using 149  
COLOR\_PALETTE 72, 150  
column keyword 112  
column profiles 147  
comma (,) 11, 67  
Complex 12  
concatenating character strings 190  
CONGRID 138, 145  
continuation character (\$) 11, 68  
CONTOUR 113

- contour plot
  - 3D 119
  - creating 83, 113
  - filled 120
- CONTOUR2 procedure 122
- contrast enhancement 140
- control characters
  - list of those that stop or interrupt PV-WAVE 95
- Control-Break
  - aborting PV-WAVE 95
- Control-C
  - interrupting PV-WAVE ??-95
- COS 83
- creating data 66

## D

- d.n\_colors system variable 72, 73, 141
- data
  - binary 98
  - creating 66
- data\_dir system variable 112
- date/time
  - functionality 77
- DC\_READ\_24\_BIT 112
- DC\_READ\_8\_BIT 112
- DC\_READ\_DIB 112
- DC\_READ\_FIXED 112
- DC\_READ\_FREE 112
- DC\_READ\_TIFF 112, 129
- default
  - viewing angle 124
  - window index number 68
- DELFUNC 98
- DELPROC 98
- DELVAR 98
- differential equations 191
- display
  - latitude and longitude data 115
  - multiple plots in a window 107
  - subarrays 144
  - TIFF file or logo 129
- documentation
  - online 65
- dollar sign (\$) 68
- Double 12
- Double Complex 12

## E

- empty output buffer 94
- EQ operator 140
- exclamation mark (!) 78
- executive commands
  - .SIZE 98
- exiting unconditionally 94
- external data representation (xdr) 98, 102

## F

- FFT
  - creates lowpass filter 106
- FILEPATH 136, 153
- files
  - batch 193
  - closing 101
  - keyword 102
  - opening 101
  - reading 102
- filled contour plots 120
- FINDGEN 66, 106, 151
- FIX 143
- FLOAT 103
- Floating 12
- flushing, output buffer 94
- follow
  - keyword 113, 127
  - method of contouring 113
- fonts
  - character sets 75
  - Hershey character sets 75
  - specification 77
  - vector-drawn 75
- Fourier transform filtering 105
- fourier.pro 171
- FREE\_LUN 101

## G

- gamma\_bessel.pro 194
- GET\_LUN 101
- GT operator 140

## H

- hardware fonts
  - See fonts
- head.img 136
- heat.pro 160
- HELP
  - context sensitive 34
  - starting online documentation 11
- help
  - online 65
- Hershey fonts 75
- HIST\_EQUAL 141
- HLS 150
- horizontal keyword 124
- HSV 150

## I

- image processing tools 27
- IMAGE\_CONT 86
- index number, of window 68
- INFO 69, 92, 98
- INTARR 102
- Integer 12
- interp keyword 130, 138, 145

## J

- JOURNAL 193, 199

## K

- keyword
  - names and abbreviations 137
  - set to 1 with slash 137
  - used by many functions 68
  - using 68

## L

- labels 75
- LE operator 140
- length keyword 111
- levels keyword 117
- linear systems
  - problems 201
- linear.pro 201

- linestyle keyword 73
- LOADCT 72, 83, 103, 154
- loading
  - a color table 83
  - data from a file 102
  - PV-WAVE save session file 97
  - See also opening files
- logical unit number (LUN) 111
- logo, displaying 129
- Longword 12
- lorenz.pro 178
- lower\_only keyword 125
- LT operator 140
- LUFAC 201
- LUSOL 201

## M

- matrices 12, 198
- matrices.pro 199
- matrix multiplication operator (#) 110
- maximum operator 141
- MEDIAN 142
- monte.pro 168
- MOVIE 208

## N

- Navigator
  - introduction to 15
  - starting the 17
- NE operator 140
- nist.dat 210
- nlevels keyword 113
- nodata keyword 152, 191
- noerase keyword 127
- nonlin\_system.pro 157
- nvecs keyword 111
- nyc.dat 153

## O

- ODE 177, 191
- online documentation 65
- OPENR 101, 136
- OPENW 102
- operators
  - infix notation 10
- OPLOT 104, 152

- options
  - font 94
- order keyword 129
- order, for drawing titles and numbers 76
- ordinary differential equations 177, 191
- overplotting 71, 80

## P

- p.charsize system variable 79
- p.multi system variable 107
- p.region system variable 79
- p.thick system variable 79
- p.ticklen system variable 79
- para\_spline.pro 164
- PLOT 67, 103, 152
- plots
  - 2D 23
  - 3D contour 119
  - contour 83, 113
  - filled contours 120
  - linear 103
  - profile 29
  - shaded surface 84, 126
  - surface 83, 123
  - vector 110
- plotting symbols 70
- plus character (+) 78, 190
- PM 198
- PMF 198
- previewing ASCII data 20
- PRINT 198
- printing 35
- profile plots 29
- PROFILES 87, 147
- program
  - area full 98
- prompt 94
- psym keyword 70, 104
- pv\_wave.spd 102
- PV-WAVE session
  - aborting 95
  - exiting 94
  - files are overwritten 96
  - restoring 97
  - saving 96

## Q

- quit PV-WAVE session 64

## R

- RANDOMN 150
- RANDOMU 103
- READ 198
- reading a file
  - DC\_READ\_24\_BIT 112
  - DC\_READ\_8\_BIT 112
  - DC\_READ\_DIB 112
  - DC\_READ\_FIXED 112
  - DC\_READ\_FREE 112
  - DC\_READ\_TIFF 112
  - READ 198
  - READU 102
- REBIN 138, 145, 154, 207
- recall\_commands keyword 74
- recalling commands 74
- REFORM 112
- regression 180
- RESTORE 97
- restoring a PV-WAVE session 97
- RETAIL 192
- RGB 150
- RM 198
- RMF 198
- ROBERTS 144
- ROTATE 145
- rotating the images in an animation 210
- row profile 87

## S

- SAVE 96
- save keyword 127
- saving
  - a PV-WAVE session 96
- semicolon (;) 11
- SHADE\_SURF 126
- shades keyword 131
- shading
  - plots 84, 126
- sharpening 144
- SHIFT 108
- SHOW3 88, 130

- showfonts batch file 75
- SIN 83, 151
- slash (/) 137
- SMOOTH 85, 105, 142
- smoothing 142
- SOBEL 144
- spline keyword 113
- Standard Library 86, 110
- storing session information 96
- String 12
- strings, concatenating 190
- subdir keyword 136
- subtitle keyword 78
- SURFACE 123
- surface plots 33, 83, 123
- SURFR 114, 119
- symbols 70
- system variables 72, 94

## T

- T3D 115, 127
- TEK\_COLOR 118, 150
- text
  - hardware-drawn 75
  - vector-drawn 75
- thick keyword 106, 119
- Thresholding 140
- TIFF 129
- title keyword 68, 77, 108
- TODAY 77
- transformation, 3D 127
- TRANSDUCE 200
- TV 85, 137, 207
- TVSCL 138

## U

- unsharp masking 143
- up arrow (  
(up arrow) 74

## V

- variables
  - getting information on 93
  - system 72, 94
- VDA 6
- VDA Tools, definition of 15

- vector
  - plot 110
  - variable 66
- vector-drawn text
  - See text
- VEL 110
- view setup
  - using Options menu 94
- Visual Data Analysis 6
- vni\_small.tif 129

## W

- WAIT 201
- wavesave.dat 96
- wavesave.pro 194
- wd\_pike\_elev.dat 112
- WDELETE 89, 109, 146
- WINDOW 68, 137
- window
  - creating 68
  - deleting 89, 109
  - resizing 83
  - setting size 107
- Windows
  - control characters 95

## X

- x.charsize system variable 79
- xdr keyword 102
- xguess keyword 157
- xpos keyword 86
- xrange keyword 68
- xsize keyword 68, 107, 137
- xstyle keyword 116
- xtitle keyword 77, 104
- XYOUTS 80, 196

## Y

- y.charsize system variable 79
- ypos keyword 86
- yrange keyword 68
- ysize keyword 68, 107, 137
- ystyle keyword 116
- ytitle keyword 77, 104

# Z

ZEROSYS [157](#)  
zstyle keyword [127](#)