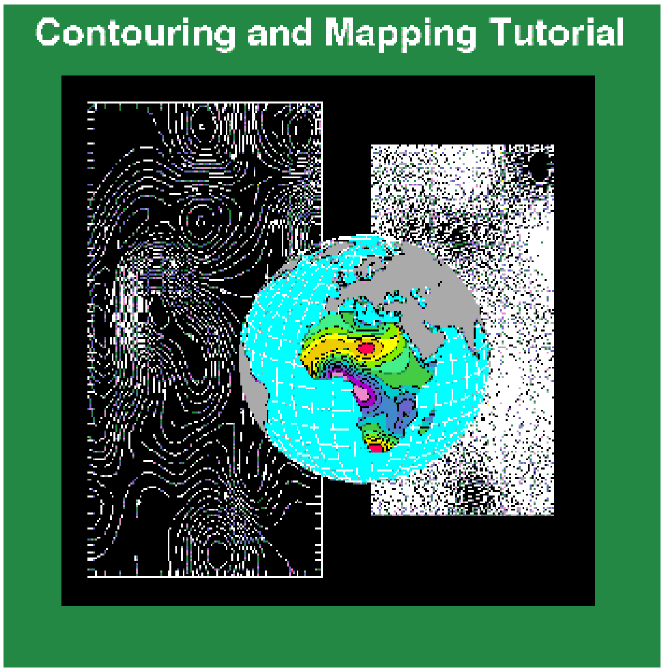
LLU Contouring and Mapping Tutorial

NCAR Graphics UNIX version 4.0.1



Preface and detailed table of contents

A short introduction

Using Areas - table of contents

Using Ezmap - table of contents

Using Conpack - table of contents

NCAR Graphics
Contouring and Mapping Tutorial

Version 2.0

May 1993

Copyright

Trademarks

Published by: National Center for Atmospheric Research, Scientific Computing Division, P.O. Box 3000, Boulder, CO 80307-3000. The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation.

Acknowledgments

Author

Adrianne Middleton-Link

Technical writer/editor

Brian Bevirt

Artist

Jacque Marshall

Document production

Christine Guzy, coordinator
Jacque Marshall

Technical reviewers

Ethan Alpert, SCD

Donna Beller, CGD

Michael Böttinger, Deutsches Klimarechenzentrum GMBH

Ginger Caldwell, SCD

Lee Carter, SCD

Diane Chace, Fleet Numerical Oceanography Center

Fred Clare, SCD

Dennis Colarelli, SCD

Nancy Dawson, SCD

Jim Eenigenburg, U.S. Forest Service

Keith Eggleston, Northeast Regional Climate Center

Gabor Fichtinger, Center for High Performance Computing, University of Texas at Austin

Christine Guzy, SCD

Mary Haley, SCD

Ken Hansen, SCD

Evelyn Hesse, Fleet Numerical Oceanography Center

Dave Kennison, SCD

Jeff Kuehn, SCD

Kirk Maasch, Quaternary Institute, University of Maine

Jacque Marshall, SCD

Pete Morreale, SCD

Tom Parker, SCD

Juli Rew, SCD

Tim Scheitlin, SCD

Dennis Shea, CGD

Susan Smith, SCD

Shirley Stephan, Lawrence Livermore National Laboratory

Paula Theophilus, SAIC

Michael Timlin, CIRES

The author especially wants to thank Dave Kennison of NCAR's Scientific Visualization staff. Dave has provided extensive technical support, explanations, and suggestions for this tutorial.

Thanks to Christine Guzy for pioneering many new techniques for electronically producing all the graphics in this book and demonstrating new ways to publish NCAR Graphics output. This work was made possible by ongoing technical support from Fred Clare, Dave Brown, John Clyne, and Dave Kennison of the Scientific Visualization Group. Thanks to Martha Lankton of NCAR Graphic Services for her help in printing this manual.

The instructional design principles used in this manual were planned by Adrianne Middleton-Link, Jeff Kuehn, and Brian Bevirt; the design was executed and refined by Brian Bevirt and Jacque Marshall. This particular design is derived primarily from seminars and books by Edmond H. Weiss, Ph.D.

Special thanks to Jacque Marshall for using Adobe Illustrator to create all the illustrations in this document that were not produced with the NCAR Graphics package. Jacque has also developed and and refined the templates for this design using Microsoft Word and FrameMaker.

Conventions used in this document

General

Bold type indicates filenames, pathnames, directories, UNIX command names, and UNIX command line options. The files that provide examples for this tutorial are sometimes specified by their exact filename, and other times the usage is more informal. Examples:

- Using **ccprwc.f**, set RWC=100, then set RWC=250, and compare the plots for differences.
- Using the **ccpcit** example, set Conpack so that every third contour is labeled.

Italic type indicates a variable where you provide the substitution. For example:

- The example uses "nice" values based on *n*/3*10*k*, where 1<=*n*<=11, and *k* is any integer.

Fortran

Fortran code segments and error messages are shown in Courier ("typewriter") font:

```
CALL GRIDAL (K-1, 0, N-1, 0, 1, 1, 5, 0., 0.)
ERROR    6 IN CPSPS2 - ERROR IN CALL TO MSSRF1
```

In the synopsis of Fortran code segments, lowercase letters represent a specific value that must be used in the code. These strings conform to the Fortran naming convention for declaring variable types as INTEGER or REAL. Examples:

```
CALL CPSETI ('PAI', n)
CALL CPSETR ('CIT', cit)
CALL CPGETI ('LIU', liu)
CALL CPSETI ('RWC', irwc)
```

When a line of Fortran code is longer than the width of the column, the continued line is indicated with a plus sign (+) in Fortran column 6. Example:

```
4       IF (IDLS .NE. 1 .AND. IDRS .NE. 1 .AND.
       + IDLS .NE. 3 .AND. IDRS .NE. 3 .AND.
       + IDLS .NE. 11 .AND. IDRS .NE. 11 .AND.
       + IDLS .NE. 79 .AND. IDRS .NE. 79 .AND.
       + IDLS .NE. 99 .AND. IDRS .NE. 99 .AND.
       + IDLS .NE. 104 .AND. IDRS .NE. 104 .AND.
       + IDLS .NE. 107 .AND. IDRS .NE. 107 .AND.
       + IDLS .NE. 163 .AND. IDRS .NE. 163) NPTS=0
5       RETURN
6       END
```

SCD documentation and consulting

Ordering information

For ordering information, please contact NCAR Graphics ordering information at (303) 497-1201 or ncarginf@ncar.ucar.edu or

University Corporation for Atmospheric Research
SCD/NCAR Graphics Orders
P.O. Box 3000
Boulder, CO 80307-3000 (USA)

How to get consulting help

If you have questions about using NCAR Graphics, contact your NCAR Graphics site representative. If you do not know who your site representative is and cannot find out from other users or the package installer at your site, you can call the NCAR Graphics Information Line at (303) 497-1201 to obtain the name of your representative. In its sole discretion, NCAR provides limited consulting support on usage and installation on an "as available" basis only. (Please refer to your Software License Agreement for a complete definition of "support.")

Table of contents

Using Ezmap

Using Conpack

NCAR Graphics Contouring and Mapping Tutorial

Intro 1. A short introduction

Purpose of this tutorial

This tutorial manual is a learning tool for beginning and intermediate users who need to prepare NCAR Graphics programs that fill areas, draw maps, and contour data fields. It answers most questions typically raised by people who are beginning to use the Conpack, Ezmap, and Areas utilities.

Note: This tutorial assumes that you already know how to write Fortran programs, that you are working on a UNIX system so you can view NCAR Graphics **man** pages and access the tutorial examples via the **ncargex** command, and that you are acquainted with some NCAR Graphics terminology.

We highly recommend that you study chapters 1 through 4 of the *NCAR Graphics Fundamentals* guide before attempting to use this manual.

Design of this tutorial

This tutorial manual is a step-by-step guide to performing specific programming tasks. This tutorial is designed to lead you through specific tasks to be accomplished by your NCAR Graphics program and to explain important aspects of the software.

Each module describes a single concept or task. Most modules appear on two facing pages, so the two-page spreads you see in this manual provide all the essential information you need to understand a single concept or accomplish a single task.

The structure of the modules is consistent; this helps you focus on the material being presented. The sequence of modules leads you from learning basic information about NCAR Graphics programs to performing increasingly more complex tasks.

On a larger scale, the chapters are organized by utility: Areas, then Ezmap, then Conpack. Each chapter leads you from basic information about one utility to more complex tasks you can perform with that utility. To help you understand how the utilities work together, the Ezmap chapter incorporates some material from the Areas chapter, then the Conpack chapter integrates material from both Ezmap and Areas.

Other resources to use with this manual

Other documentation is also available to help you use NCAR Graphics: the introductory user guide *NCAR Graphics Fundamentals*, a full set of **man** pages, and programmer documents from the package developers. Examples are also provided to show you NCAR Graphics capabilities and to help you understand and use the code that produces the examples. The online version of *Fundamentals* and the programmer documents are provided as PostScript files on the distribution medium. Please see your site installer for them.

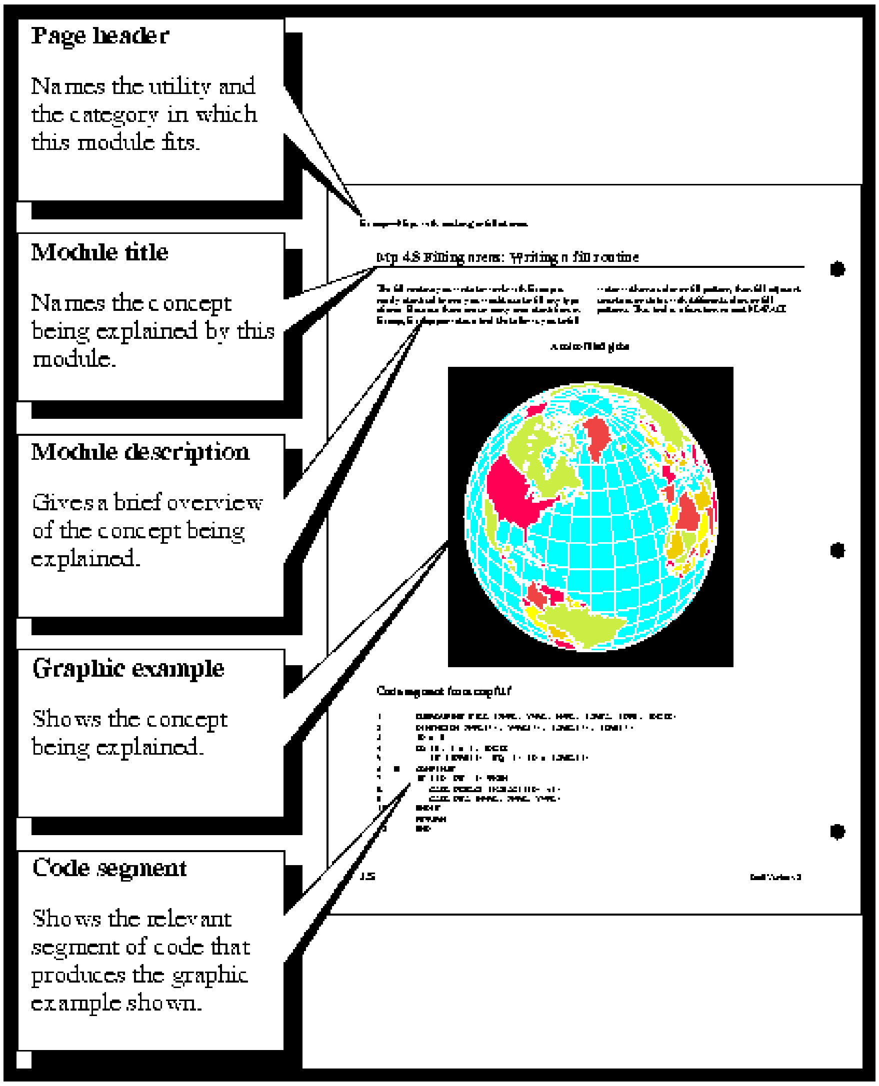
NCAR Graphics Fundamentals contains important information for all users: a glossary of NCAR Graphics terms, the structure of NCAR Graphics programs, strategies for learning NCAR Graphics, instructions for compiling, linking, and running programs, instructions for producing and editing output from NCAR Graphics programs, and guidelines for using the utilities.

The online **man** pages provide concise reference information for all user entry points, user-modifiable internal routines, parameters used in all utilities, and NCAR Graphics commands (such as **ncargex**). The overview **man** page **ncargintro** introduces all utilities in the package, and overview **man** pages for each utility introduce all routines in that utility.

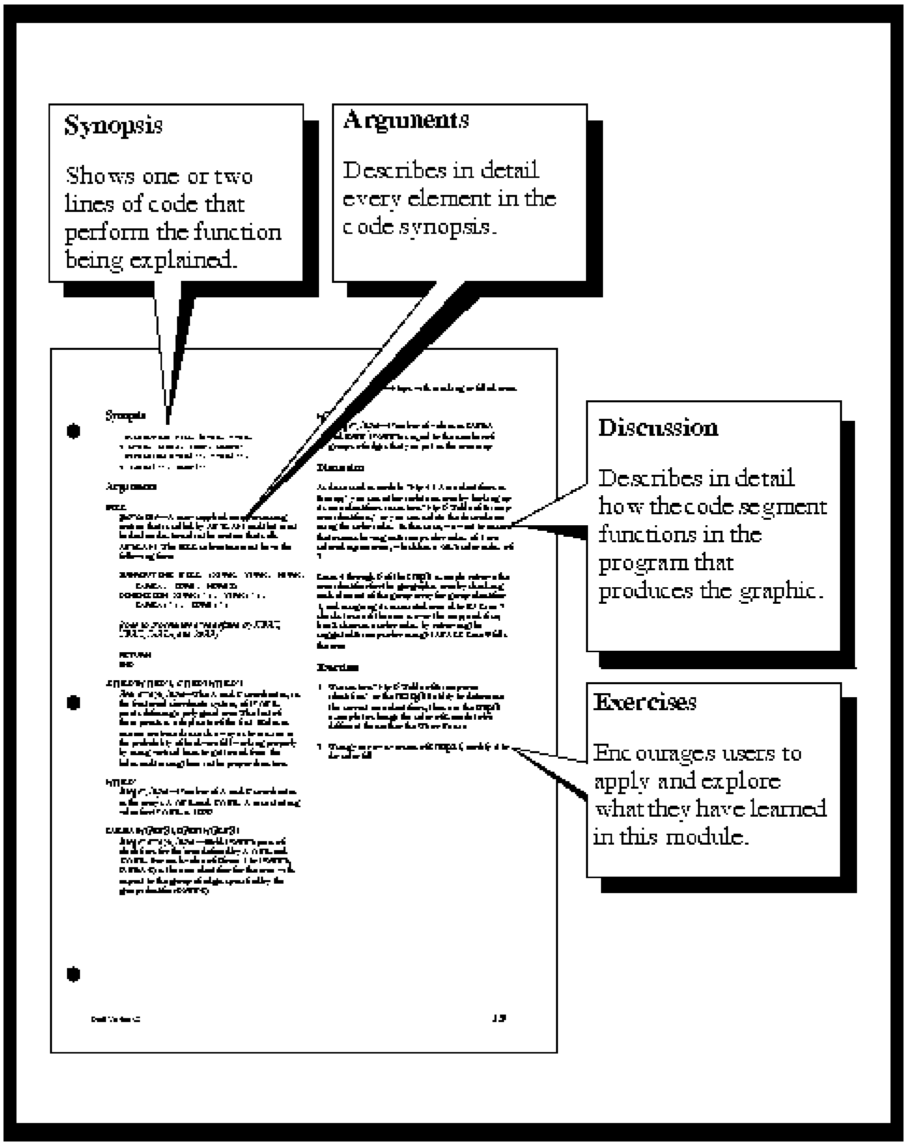
Programmer documents are the software developers' reference notes; they are provided for users who want to explore the information that developers use to maintain a utility's source code.

The **ncargex** command allows you to access more than 250 examples, and all of them are internally documented with comments. A brief introduction to **ncargex** is provided under the "Code segment" heading below, and procedures for using **ncargex** appear in Chapter 2 "Five quick steps for creating and viewing your plot" of *NCAR Graphics Fundamentals*. A pictorial index to all NCAR Graphics examples is provided as Appendix D of *Fundamentals*.

Intro 1.1 Understanding the layout of tutorial module



Page header
Names the utility and the category in which this module fits.
Module title
Names the concept being explained by this module.
Module description
Gives a brief overview of the concept being explained.
Graphic example
Shows the concept being explained.
Code segment
Shows the relevant segment of code that produces the graphic example shown.



Intro 1.2 Details about how information is presented in this manual

This tutorial is divided into chapters for each of the three utilities. Each chapter is divided into sections, and each section contains modules. The modules are organized as shown on the preceding two pages; most are laid out as two-page spreads so that all relevant information is available to you at a glance. Some modules fit on one page, and a few require four pages.

Reference guides throughout

Each chapter starts with a series of quick reference tables that show all user entry points and parameters for the utility. More detailed parameter descriptions appear as the final section of each chapter.

A functional outline for the utility is presented at the end of each chapter's first section and on the first page of each subsequent section. This outline shows the basic steps needed to run the utility. The steps to be discussed in that section are marked with asterisks to help you see how the concepts described in that section fit into any Fortran program using the utility.

Basic concepts explained first

If you must first understand any underlying concepts for a utility before you use it, the first section of the chapter discusses these concepts. (This includes definitions of terms and overviews of routines that control parameters.) If the utility provides a routine that allows you to quickly view a simple plot of your data, the first section also presents this.

The second section of a chapter may provide additional preliminary information. All subsequent sections then contain task-oriented modules that lead you through all steps necessary to use the utility.

Instructions for performing tasks

Each module explains a specific task, has a descriptive title, and is structured in a consistent way. The following describes the guidelines used in creating each part of a task-oriented module: this will be useful if you have a question about the way information is presented in any of these modules.

Module header, title, and introduction

The title of the module defines the concept or task that is discussed in it. The header above the title helps you see how this concept or task fits into a program that uses the utility. The text below the title briefly introduces the concept or task.

Module numbering indicates the chapter (with the two-letter utility code), the section (first number), and the module (second number, after the period). For example, module Cp 9.4 is the fourth module in the ninth section of the Conpack chapter. In some cases, multiple modules are needed to cover a group of related concepts; all such modules are grouped together with an additional number (for example, Cp 6.9.1 through Cp 6.9.4).

Graphic example

The graphic illustrates the task discussed in the module. In most modules, the graphic is produced by the example program named in the code segment.

All black-and-white graphics appear on your screen as white lines on a black background; in this manual, these examples are printed with black lines on a white background. This reversal is done so the graphics are easier to see in the manual. Most color graphics in this manual appear exactly as they do on your screen; in rare cases, a black background is changed to white for improved readability on the printed page.

The first instance of a color graphic is always printed in color. If a color graphic occurs twice in a four-page module, however, the second instance of the graphic appears in black and white.

Code segment

The code segment is an excerpt from the Fortran code that generates the graphic example; it shows how the task being explained by the module is accomplished. The line numbers before each line are only for references from the text; these numbers have nothing to do with the line numbers in the example code.

The heading "Code segment from *example_name*.f" gives you the name of the example file. To run the example and create the NCAR CGM (NCGM), type:

ncargex ***example_name***

Do not type the ".f" extension; "*example_name*.f" denotes the file that contains the Fortran code. The **ncargex** command generates local copies of the example source code, compiles and links the source file(s), runs the executable, and produces an NCGM that you can examine with a translator such as **idt** or **ictrans**. More information is available from the **man** pages and the *NCAR Graphics Fundamentals* guide.

All Fortran code is printed in this manual with the Courier (typewriter) font. This monospaced type font preserves the correct relationships of Fortran columns. Any Fortran code that continues onto the next line is printed with a plus sign (+) in column 6 of the continued line.

Synopsis

This line of code demonstrates the calling sequence of the routines or parameters discussed in the module. The synopsis may contain more than one line of code.

Arguments

This part of the module is a detailed description of every element in the synopsis. This section typically describes each argument to or parameter for the Fortran routine(s) in the synopsis.

Two important pieces of information follow the argument name: its Fortran type and its function in the program. When parameters are listed under the Arguments heading, only their Fortran type is given.

Fortran types include INTEGER, REAL, and CHARACTER, plus arrays of these types (the dimension of any array is always indicated). The argument's function in the program may be as *input* (the user supplies a value for the argument), *output* (the routine returns information to the user or the program in the argument), *workspace* (a scratch array for the routines to use in manipulating information, but that the user must declare), or as a *subroutine* (a special-purpose routine that the user typically writes or modifies to produce a desired result).

In parameter descriptions, uppercase letters show how the characters that comprise the parameter name are derived. Uppercase letters are also used in this way in the tables of parameters at the beginning and end of each chapter. This is not done for argument names, although the argument names are often derived in a similar way.

Wherever it is appropriate, specific values and ranges of values are explained in text that is indented from the text describing arguments and parameters.

Characters used as variables to represent a number (*n* or *m* for example) are printed in italics.

Discussion

This part of a module provides a line-by-line analysis of how the code segment functions in the program. The Discussion may also include other useful information such as common mistakes and how to avoid them, frequently seen error messages and the situations in which they occur, and general cautions.

All references to example names, filenames, and commands are printed with bold characters.

Exercises

These are directions that encourage users to apply and explore the concepts and tasks presented in this module. If you have trouble getting the desired results, please contact your site representative.

Using Areas

Table of contents

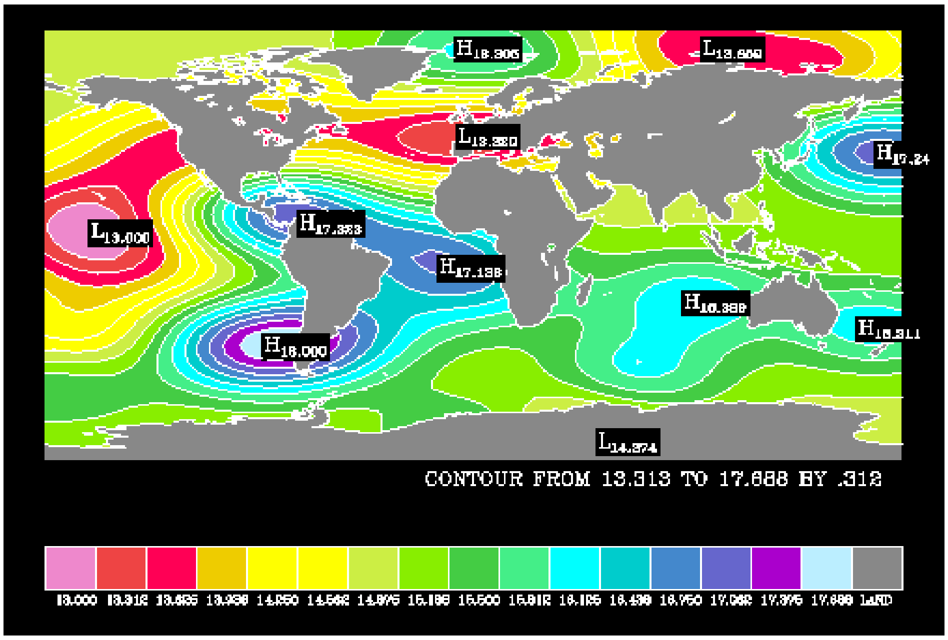
Using Areas

Ar 1. What is Areas?

Areas is designed as a support utility for other utilities. For this reason, most users need Areas only when using Conpack, Ezmap, or Vectors. However, Areas has a broad range of applications beyond the uses discussed in this chapter.

Used with a utility like Ezmap or Conpack that creates lines dividing the plotter frame into regions, Areas allows you to retrieve (or recover) outline and other information about each region. Also, Areas allows you to overlay two or more groups or sets of lines, and to recover the regions defined by the union of the groups as well as telling you where each region is located relative to each group of lines. These features allow you to do things like mask contour lines or latitude/longitude lines over the ocean, and color-fill or pattern-fill regions.

A common Areas application



Ar 1.1 Table of Areas user entry points

The Areas utility has nine user entry points. This table organizes the routines according to their functions.

Initialization and support routines

ARINAM

Initializes an area map. ARINAM must be called before using any other Areas routine that references that area map. See modules Ar 2.1, Mp 4.3, and Cp 5.3.

AREDAM

Adds edges or area boundaries to an area map. AREDAM must be called for each edge to be included in an area map. See module Ar 2.2.

ARPRAM

Processes an area map, removing nonclosed edges and cleaning up area identifiers, among other things. ARPRAM is called by the action and query subroutines if you don't call it explicitly. See modules Ar 2.3 through Ar 2.3.3.

Action routines

ARDRLN

Breaks a given polyline into pieces, each of which lies in a single area in an area map. A specified user routine is then used to decide what to do with each piece. See module Ar 3.2.

ARSCAM

Scans an area map, extracting the definitions of all areas formed by the map, and allows you to process each area separately. See modules Ar 3.4, Mp 4.7, and Cp 5.7.

Query routine

ARGTAI

Given a point in an area in an area map, ARGTAI returns group and area information about that area. See module Ar 3.1.

Parameter setting and retrieving routines

ARGETI

Retrieves an Areas integer parameter value. See module Ar 1.9.

ARSETI

Sets an Areas integer parameter value. See module Ar 1.9.

Debugging routine

ARDBPA

Produces a picture of that part of the area map that belongs to a specified group. See module Ar 4.2.

Ar 1.2 Table of Areas parameters

This table is a quick reference guide to all Areas parameters. More complete descriptions appear in section "Ar 5. Areas parameter descriptions."

The behavior of a typical routine in an NCAR Graphics utility is sometimes determined entirely by the routine's arguments, but frequently it is also affected by the value of one or more of the utility's "parameters." A parameter is a variable that controls the behavior of a utility; parameters are accessed via parameter-access routines that can set or retrieve the parameter value.

Instructions for setting and retrieving Areas parameters are provided in module "Ar 1.9 Areas parameters: What they do and how to use them."

Parameter	Brief description	Fortran type	Examples	Module
AT	Arithmetic Type flag	Integer		
DB	DeBug plots flag	Integer	cardb1	Ar 4.1
DC	Debug Colors index	Integer		
DI	Direction Indicator flag	Integer		
LC	Largest Coordinate	Integer		

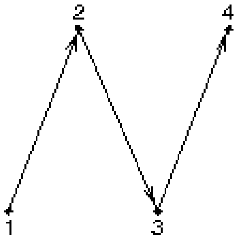
Ar 1.3 Areas definitions: Edges

The Areas definitions modules Ar 1.3 through Ar 1.7 provide the background you need to understand the discussions in subsequent Areas modules. Because technically complete definitions involve complex mathematics that are not needed by most users, the definitions provided here are intended to give an intuitive understanding of the material.

Three related concepts are very basic to Areas: edges, groups of edges, and group identifiers.

Examples of edges

```
  Figure 1  Figure 2
```



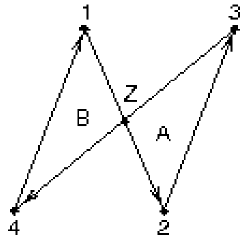
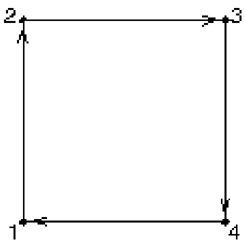
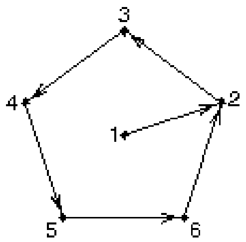


Figure 3 Figure 4





Discussion

An edge is a set of line segments connected so that a pen tracing the edge never needs to be lifted. Said another way, points specifying an edge must be sequentially ordered, as in Figure 1. Edges can be made of more than one line segment.

When one or more edges cross, Areas adds a vertex at each intersection so that regions such as A and B in Figure 2 can be addressed separately. When Figure 2 is drawn, it is defined by line segments (1,2), (2,3), (3,4), (4,1), but when the edges are processed by Areas, the point Z is added, and the edge segments defining the figure become (1,Z), (Z,2), (2,3), (3,Z), (Z,4), (4,1). These *edge segments* are among the building blocks of the Areas utility.

It is possible to define the same set of edge segments in many different ways. Looking at Figure 3, you can define the square as a single edge (1, 2, 3, 4, 1), or as two or more edges, like (1, 2, 3) and (3, 4, 1). In both cases, the square is defined by a set of edges: an *edge group*.

The most useful edge groups have no vertexes from which only one line segment emanates. In other words, useful edge groups won't have dangling line segments. This property implies that the plane is divided into discrete areas with well-defined boundaries. For example, the pentagon in Figure 4 divides the plotter frame into the area inside the pentagon and the area outside the pentagon. Notice that the line segment between points 1 and 2 in Figure 4 does have a vertex with precisely one line segment emanating from it. By default, Areas will remove dangling edges like this from the area map.

An area map is a linked list of edge segments, from which area and group information can be extracted. Area maps are discussed in module "Ar 1.6 Areas definitions: Areas, area identifiers, and area maps."

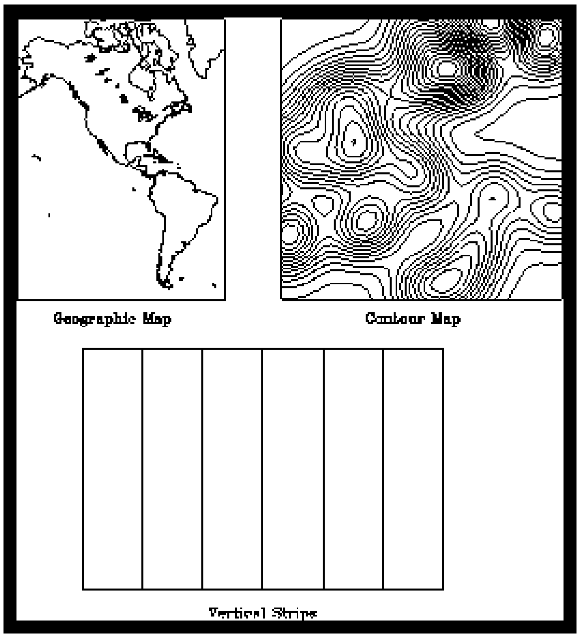
Ezmap produces dangling line segments. For instance, some small islands are crescent-shaped edges, so Areas will remove them from the area map.

Ar 1.4 Areas definitions: Groups of edges

A *group of edges* can be just about any set of edges that you want to group together. The most useful groups of edges are boundary lines from the Ezmap utility, contours from Conpack, and lines defining vertical strips. By defining more than one group of edges in an area map, it is possible to access areas defined by each group as well as areas defined by the union of the groups.

Note: Areas definition modules Ar 1.3 through Ar 1.7 provide the background needed for the discussions in subsequent Areas modules. Because technically complete definitions involve complex mathematics that are not needed by most users, the definitions provided here are intended to give an intuitive understanding of the material.

Three important edge groups



Discussion

The graphic in this module shows three sets or groups of edges. Areas allows you to individually assign any edge to an edge group. Groups are differentiated using a positive integer value called a *group identifier*. The module "Ar 2.2 Adding edges to an area map" discusses the details of assigning edges and group identifiers.

Although you can assign any identifier to a specific group with Areas, it is helpful to know the group identifiers for the groups that are created with the Ezmap and/or Conpack utilities. The following group identifiers are the defaults used with Ezmap and Conpack:

- 1 Regions defined by the Ezmap outline set
- 2 Vertical strips defined by Ezmap
- 3 Contour levels defined by Conpack

- 4 Vertical strips defined by Conpack

Vertical strips are strips dividing an Ezmap or Conpack plot into columns so that it will plot correctly on limited hardware devices, such as older terminals.

Using the graphic in this module, you could overlay the contour map over the geographic map; the map outlines would all be in group 1, and the contour lines would all be in group 3. By default, Ezmap produces a single vertical strip for the entire plot, so you would also have a group 2 defined that contains the perimeter of the geographic map. If you needed to overlay Conpack vertical strips as well, the Conpack vertical strips are placed in group 4 by default.

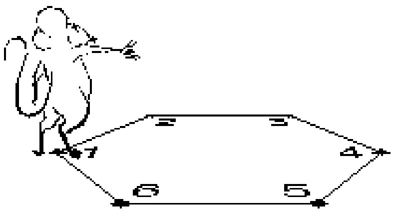
Ar 1.5 Areas definitions: Left and right

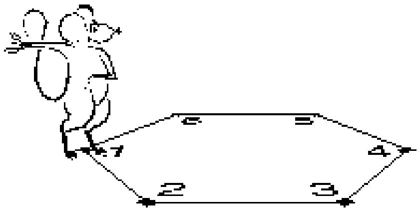
Each edge segment has a left side and a right side.

Note: Definitions in modules Ar 1.3 through Ar 1.7 provide the background needed for the discussions in subsequent Areas modules. Because technically complete definitions involve complex mathematics that are not needed by most users, the definitions provided here are intended to give an intuitive understanding of the material.

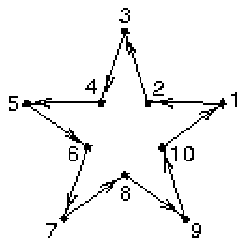
Left and right in Areas

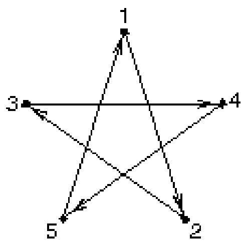
```
Figure 1a  Figure 1b
```






```
Figure 2  Figure 3
```





Discussion

Every edge segment has a left side and a right side that depends on the order of the points in that edge. The best way to understand which side is left and which is right is to imagine yourself standing on the first point on an edge, facing the second. Everything to your left is left of the edge segment, and everything to your right is right of the edge segment. Notice how in Figure 1, the area to the right of the edge depends on the order of the points in the edge.

Figure 2 shows a more complex edge. In this case, the area inside the star is to the left of the edge, and the area outside the star is to the right of the edge. Figure 3 illustrates an edge that crosses itself. Areas breaks the edge down into edge segments, and then in sequential order, determines what is right of each edge segment and what is left of each edge segment. The preprocess area map modules Ar 2.3 through Ar 2.3.3 discuss how area information is reconciled for edges like the one in Figure 3.

Conpack always traces contour lines in such a way that the function being contoured slopes downhill to the left and uphill to the right. For this reason, areas where the function value is below the contour line are to the left and areas where the function value is above the contour line are to the right. Since this is always the case, we refer to areas above or below a contour line rather than to the left or right of a contour line.

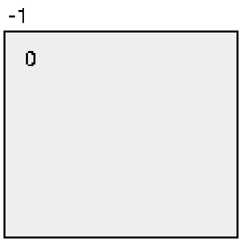
Ar 1.6 Areas definitions: Areas, area identifiers, and area maps

This module clarifies how Areas defines the term *area*; this module also defines the term *area map*.

Note: Definitions in modules Ar 1.3--Ar 1.7 offer an intuitive approach to understanding the material.

Examples of areas

Figure 1 Figure 2a



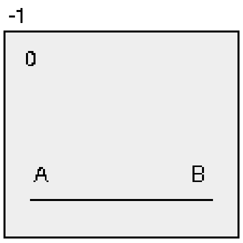
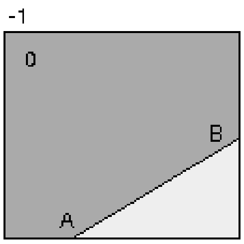


Figure 2b Figure 3a



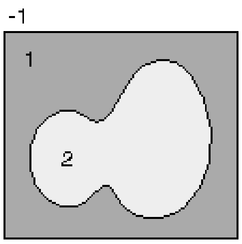
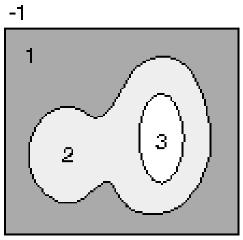
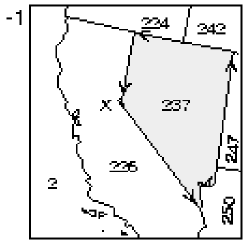


Figure 3b Figure 4





Discussion

An *area* defined by a set E of edge segments is a set S of points in the plane such that 1) if A and B are both members of S, there exists a continuous curve that connects A and B and that nowhere crosses any of the edge segments in E, and 2) if A is a member of S and there **does** exist a continuous curve that connects A to some other point B without crossing any of the edge segments in E, then B is also a member of S.

Let's look at some cases illustrating what this definition means. First, Areas defines a *perimeter* for the plotter frame so areas that would be otherwise unbounded can be identified and treated correctly by limited software and hardware. This perimeter is drawn around the limits of your plotter frame, but you can make any edge group inside your plotter frame into a perimeter. Figure 1 shows the default perimeter created by Areas.

First let's assume that we have two points, A and B, with an edge segment joining them. As shown in Figure 2a, the area defined by A and B is the entire plotter frame. When both A and B lie on different edges of the perimeter (Figure 2b), the edge segment (A, B) defines two areas, one to the left of the edge and one to the right of the edge.

A more complex case is that of a closed edge, or a loop (Figure 3a). In this case, what is outside the loop is an area, and what is inside the loop is an area. By adding one or more loops inside or outside the original, we now have areas within areas, or alternatively, holes within areas (Figure 3b).

Another common edge group that defines areas is a set of intersecting edges, for example, a map of the US state boundaries. As shown in Figure 4, you can think of an area as a region that is defined by starting at an endpoint X on an edge segment, going to the end of that edge segment, choosing the leftmost connected edge segment, going to the end of it, choosing the leftmost connected segment, and so on, until you return the first edge segment (Figure 4). To define another area, start at an edge segment and choose only rightmost edges.

Each edge segment has a left area identifier and a right area identifier associated with it. As an area is defined, Areas keeps track of the area identifiers associated with each edge segment; if these don't all match, Areas decides which one has precedence. This process is discussed in detail in the preprocess area map modules 2.3 through 2.3.3.

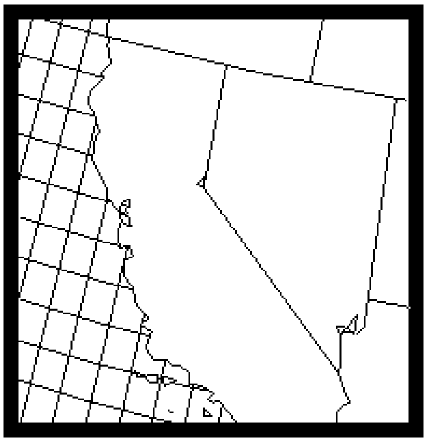
Generally, negative area identifiers are all treated as if they were -1; this denotes areas outside the perimeter. Zero area identifiers are used to denote areas that have not yet had an area identifier assigned, or areas where you don't know what area identifier has been assigned by some other edge segment in the area map. Positive area identifiers should be used to denote any area that you may want to address. The numbers in the various areas in Figures 1 through 4 show possible area identifiers for each area.

An *area map* is a set of edge segments in the plane. There are three integers associated with each edge segment in the area map: a group identifier, a left area identifier, and a right area identifier. An area map is represented by a linked list of information that is constructed by calls to routines in the Areas utility. In the Fortran code, the area map (called MAP in the examples) is stored in a large integer array that is passed from subroutine to subroutine.

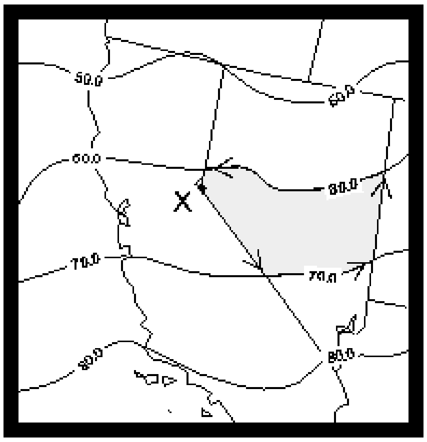
Ar 1.7 Areas definitions: Masks and areas defined by more than one edge group

There are two additional important concepts in Areas. A *mask* is a set of areas that allows you to do one thing in one set of areas and another in the masked areas. Masks are frequently used to hide latitude and longitude lines, protect areas from contours, change line or fill style and color, and so on. It is also important to understand how areas can be created from more than one edge group.

A mask over land



Areas defined by two groups of edges



Discussion

An area map can be used as a *mask* to hide parts of your plot from line-drawing, color-filling, and other routines so that you draw on or color only portions of your plot. The upper figure shows a plot where latitude and longitude lines are masked over the land masses so that they are only drawn over the ocean.

The previous module describes how to define an area by starting at an edge segment and tracing the leftmost edge segments until returning to the first edge segment. In that case, we traced out the state of Nevada as an area. In the lower figure, we overlay a contour edge group on the edge group defining the map of the western United States. Using the same approach as we used before, we now trace out the shaded area, the portion of a contour level over Nevada. Notice that as we do this, we choose the leftmost edge, regardless of which edge group it's in. In this case, the area has two area identifiers, one for the contour band that it's in and one for the state that it's in. As explained in module "Ar 1.4 Areas definitions: Groups of edges," contour levels are in edge group 3, and geographic maps are in edge group 1 by default.

Ar 1.8 What calls do I need to get my Areas plot?

Areas is one of the more complex utilities to use because it requires you to make at least three subroutine calls before a plot is drawn. It will probably also require you to write a subroutine or two to define exactly what is supposed to be done with specific areas after they are defined. For example, if you want to draw lines over a portion of your plot and color other portions of your plot, you would probably have to write two routines to do this: one to draw the lines in selected areas and one to color-fill other areas.

The functional outline below shows how Areas works.

Areas functional outline

```
--------------------------------------------------------------
 *  1.  Open GKS
    2.  Set parameters
 *  3.  Initialize area map
 *  4.  Add edges to area map
 *  5.  Preprocess area map
 *  6.  Do one or more of the following:
 *      Break a polyline into segments, each of which lies
 entirely in a single area, then process each segment
 individually
 *      Retrieve area identifiers for areas containing a given
 point
 *      Scan an area map to retrieve each subarea, then
 process each subarea individually
    7.  Debug plot
 *  8.  Call FRAME
 *  9.  Close GKS
--------------------------------------------------------------
```

* Steps needed to produce a plot with Areas

Ar 1.9 Areas parameters: What they do and how to use them

Areas allows the advanced user to adjust a few internal variables via its parameter setting and retrieving routines. All Areas parameters have integer values. These parameters are useful mostly for debugging Areas problems. As such, most users will not need the ARGETI and ARSETI routines, or the parameters that they address.

The first argument in a call to ARGETI or ARSETI is the parameter name. The second argument is either a variable in which the value of the named parameter is to be returned (via ARGETI) or an expression specifying the desired new value of the parameter (via ARSETI).

Synopsis

```
CALL ARGETI (PNAM, IVAL)
CALL ARSETI (PNAM, IVAL)
```

Routines

ARGETI

> Retrieves integer parameter information.

ARSETI

> Sets integer parameter information.

Arguments

PNAM

> *Character expression, Input or Output*---The name of the parameter that you want to set or retrieve. Two-character parameter names appear in examples and discussions in place of PNAM. The parameter names and their meanings are enclosed in single quotation marks in the Fortran code to ensure that they are treated as character expressions rather than real or integer variables. Only the first two characters within the quotation marks are examined.

IVAL

> *Integer, Input or Output*---An integer value for ARSETI, or an integer variable to hold the parameter value for ARGETI.

Discussion

Because most users will not need to adjust Areas parameters, this chapter discusses only the most frequently used parameter, DB. For an example of how to set an Areas parameter, please see module "Ar 4.1 Debugging options: Setting parameters."

Module "Ar 1.2 Table of Areas parameters" is a quick reference guide to all Areas parameters. Section "Ar 5. Areas parameter descriptions" provides more thorough descriptions of the Areas parameters.

Ar 2. Initializing Areas

Areas is one of several NCAR Graphics utilities that requires initialization before use. This section covers initializing an area map, adding edges to the area map, and preprocessing the area map.

Before you can use Areas, you must initialize an area map and add edges to it. Preprocessing area maps can then be done either directly or implicitly in the routines that produce results.

The last part of this section discusses preprocessing for users who want a deeper understanding of how Areas works.

Areas functional outline

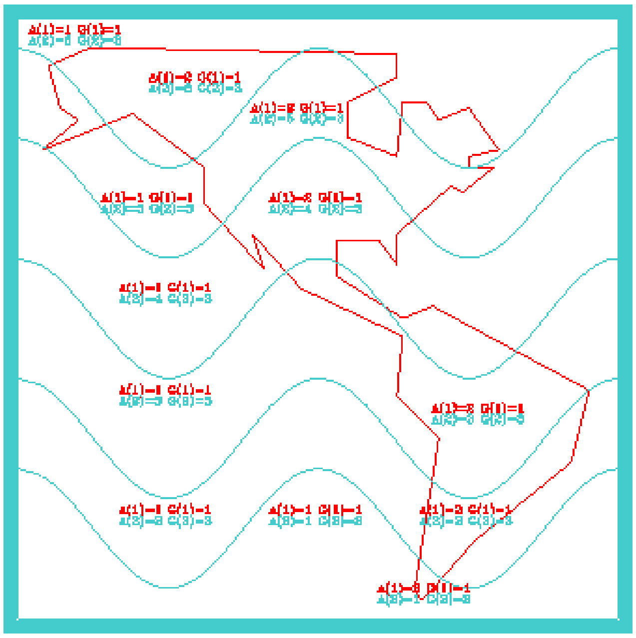
```
--------------------------------------------------------------
    1.  Open GKS
    2.  Set parameters
 *  3.  Initialize area map
 *  4.  Add edges to area map
 *  5.  Preprocess area map
    6.  Do one or more of the following:
        Break a polyline into segments, each of which lies
 entirely in a single area, then process each segment
 individually
        Retrieve area identifiers for areas containing a given
 point
        Scan an area map to retrieve each subarea, then
 process each subarea individually
    7.  Debug plot
    8.  Call FRAME
    9.  Close GKS
--------------------------------------------------------------
```

* Steps discussed in this section.

Ar 2.1 Set up area map

Before any other Areas subroutine can be used, Areas must be initialized using ARINAM.

Contours over the Americas



Code segment from carmap.f

```
      CALL COLOR
      CALL ARINAM (MAP, LMAP)
      CALL AREDAM (MAP, XGEO, YGEO, NMAP, 1, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR1, NPTS, 3, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR2, NPTS, 3, 3, 2)
      CALL AREDAM (MAP, XCNTR, YCNTR3, NPTS, 3, 4, 3)
      CALL AREDAM (MAP, XCNTR, YCNTR4, NPTS, 3, 5, 4)
      CALL AREDAM (MAP, XCNTR, YCNTR5, NPTS, 3, 6, 5)
```

Synopsis

```
CALL ARINAM (MAP, LMAP)
```

Arguments

MAP(LMAP)

> *Integer array, Output*---An integer array in which an area map is to be constructed. Each vertex for an edge segment in your area map requires ten words in the array MAP. Remember that the total number of vertexes includes those added at each intersection of edges, and those added when long edge segments are broken into shorter edge segments.

LMAP

> *Integer, Input*---Length of the MAP array. As part of the initialization process, the value of LMAP is stored in MAP(1); that way, it need not be given as an argument in calls to other routines that use the area map.

Discussion

Line 1 of the **carmap.f** code segment sets up a color table. Line 2 initializes a very large area map. In lines 3 through 8, edges are added to the area map using AREDAM, which is discussed in the next module. In the plot, notice that each area has 4 values printed in it. These values represent the group and area identifiers for each area. The next module describes how they are set.

If you set the area map too small, the error message:

```
ERROR 5 IN AREDAM - AREA-MAP ARRAY OVERFLOW
```

occurs when edges are added to the area map. There is no good way to predict exactly how large the area map should be before adding edges to it. Generally, if you are starting with a complex geographic map, start with an area map of 150,000 words. Most contour plots need between 50,000 and 100,000 words. Once you have your area map large enough, you can determine how much space was used by calculating:

```
MAP(1) - MAP(6) + MAP(5)
```

where MAP is your area map after all the edges have been added to it. Areas routines ARPRAM and ARSCAM use more space during execution than after execution, so be sure to calculate your area map size after your last call to an Areas routine, and make your area map slightly larger than the calculated size.

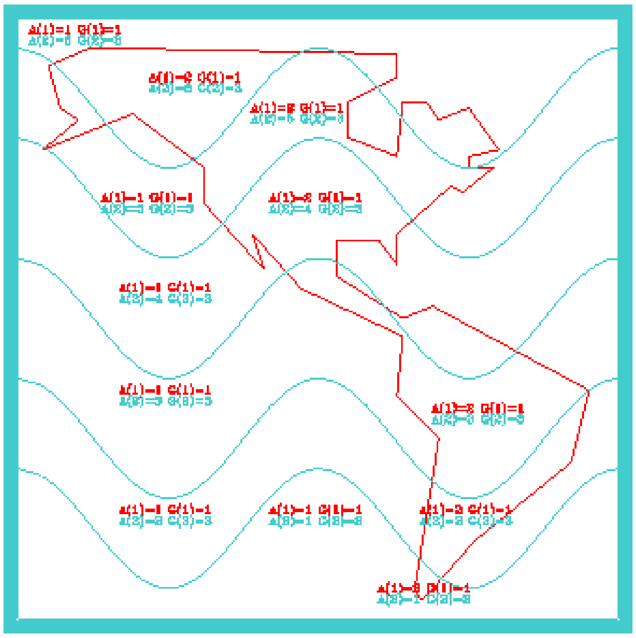
Exercises

1. Add code to **carmap.f** to determine how much of the area map is used, then reduce the size of the area map to some reasonable value.

Ar 2.2 Adding edges to an area map

Each edge is added to the area map with a call to AREDAM. When calling AREDAM directly, you must also assign group and area identifiers for each edge.

Contours over the Americas



Code segment from carmap.f

```
      CALL COLOR
      CALL ARINAM (MAP, MAPSIZ)
      CALL AREDAM (MAP, XGEO, YGEO, NMAP, 1, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR1, NPTS, 3, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR2, NPTS, 3, 3, 2)
      CALL AREDAM (MAP, XCNTR, YCNTR3, NPTS, 3, 4, 3)

      CALL AREDAM (MAP, XCNTR, YCNTR5, NPTS, 3, 6, 5)
```

Synopsis

```
CALL AREDAM (MAP, XCURVE, YCURVE, LCURVE, IGRP, IDLEFT, IDRGHT)
```

Arguments

MAP(LMAP)

> *Integer array, Workspace*---An array containing an area map initialized by a call to ARINAM. As part of initializing the area map, ARINAM stores the dimension of MAP in MAP(1); therefore, the dimension does not have to be given as an argument in calls to AREDAM.

XCURVE(LCURVE)

> *Real array, Input*---The X coordinates, in the user coordinate system, of the points defining an edge.

YCURVE(LCURVE)

> *Real array, Input*---The Y coordinates, in the user coordinate system, of the points defining an

edge.

LCURVE

Integer, Input---The number of coordinates in XCURVE and YCURVE.

IGRP

Integer, Input---Identifier of the group to which the edge belongs.

IDLEFT

Integer, Input---Identifier of the area that lies to the left of the edge.

IDRGHT

Integer, Input---Identifier of the area that lies to the right of the edge.

Discussion

Line 1 of the **carmap.f** code segment sets up a color table. Line 2 initializes the area map, and lines 3 through 8 add edges to the area map. Line 3 adds the continental outline in group 1 just as Ezmap does by default. Since the outline is drawn counterclockwise, the interior of the continent is to the left of each edge segment. The area identifier to the left of each edge segment is set to be 2, giving the interior of the continent an area identifier of 2. Similarly, the area to the right of each edge segment is given an area identifier of 1, giving the oceans area identifiers of 1.

Line 4 adds the bottom contour line. It is drawn from left to right, and it is placed in group 3 in the area map just as Conpack does by default. The area to the right of each edge segment in this contour line is given an area identifier of 1, giving the bottom contour level an area identifier of 1. The area to the left of each edge segment in the first contour line is given area identifier 2. When the second contour line is added to the area map in line 5, it is also drawn from left to right, and it is added to group 3 in the area map. The area to the right of each edge segment in this contour line is also given area identifier 2. This gives the second contour band the same area identifier for both contours so that when the area map is reconciled, the second contour level will be given area identifier 2. Lines 6 through 8 add the rest of the contour lines in a similar fashion, giving each contour band a different area identifier.

Conpack uses much the same principle as this example. Each contour line in a contour plot is put in the same group. Then area identifiers for the area above and the area below the contour are assigned, given that Conpack always draws a contour so that the area below the contour line is to the left, and the area above the contour line is to the right. Each contour level is given a single area identifier, by default starting with 1 and going to *n*, where *n* is the number of contour levels in the plot.

If you count vertexes in each edge group, you can see that the number of vertexes nearly doubles when you put both edge groups in the same area map. By putting both vertexes in the same area map, you gain control over each sub-area. However, you also increase your area map size, and you increase the CPU time needed to process each area. Since Areas is one of the most CPU-intensive utilities in NCAR Graphics, this can significantly increase the time needed to run your program. In many cases, you can produce results significantly faster by putting your geographic outlines in one area map, your contour lines in another area map, then using masking.

Exercises

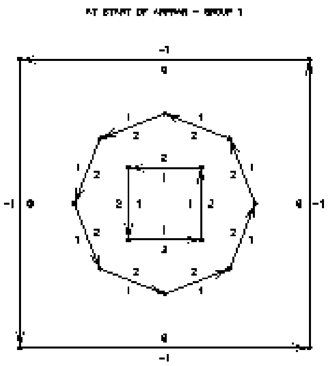
1. In the graphic, note that each area has two area identifiers and two group identifiers. Identifiers marked in red are for geographic information, and those in blue are for contour information. Be sure you understand how each area identifier is set.
2. Put each contour in a different edge group, and explain what happens.

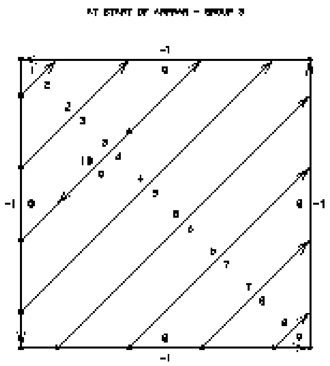
Ar 2.3 Preprocess area map: Overview

This group of four modules can deepen your grasp of Areas, but it may be skipped because area map processing is done automatically by the Areas action routines. You call ARPRAM to give information about your edge groups to Areas; this can significantly shorten processing time. If you don't call ARPRAM, other routines will call it for you. This module tells how to call ARPRAM and briefly describes the seven processing steps it does. The next three modules elaborate on these processing steps.

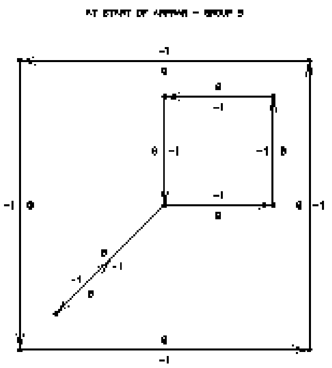
Three edge groups in example arex01

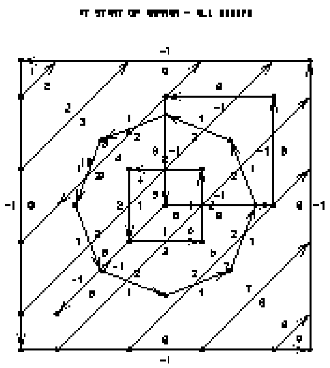
`Frame 1  Frame 2`





`Frame 3  Frame 4`





Synopsis

```
CALL ARPRAM (MAP, IF1, IF2, IF3)
```

Arguments

MAP(LMAP)

> *Integer array, Workspace*---An array containing an area map that has been initialized by a call to ARINAM and to which edges have been added by calls to AREDAM.
> **Note:** As part of initializing the area map, ARINAM stores the dimension of MAP in MAP(1); therefore, the dimension does not have to be given as an argument in calls to ARPRAM.

IF1

> *Integer, Input*---If you set IF1 nonzero, ARPRAM examines a pair of edge segments for intersection only if one of the pair has a left or right area identifier that is zero or negative. This would be appropriate for contour lines that intersect the perimeter, but that are known not to intersect each other.

IF2

> *Integer, Input*---If you set IF2 nonzero, ARPRAM does not check for dangling edges. This would be appropriate for contour lines that are known not to have any such edges. This is *not* appropriate for Ezmap boundary lines, because the Ezmap dataset contains some edge segments (small islands) that are formed by unclosed curves.

IF3

> *Integer, Input*---If you set IF3 nonzero, ARPRAM speeds up the process of adjusting area identifiers by omitting the consideration of "holes" in the areas examined. This is appropriate for contour lines as long as the left and right area identifiers provided at each contour level are consistent with those provided at the adjacent levels.

Discussion

Preprocessing an area map is a seven-step process that removes "dangling" edge segments (those that do not contribute to enclosing any area) and makes all of the area-identifier information in the area map consistent:

Step 1: ARPRAM shortens edge segments whose projections on the X axis are more than twice as long as the average. ARPRAM does this by interpolating points along their lengths. This improves efficiency when executing other parts of the algorithm.

Step 2: ARPRAM locates all intersections of edge segments and interpolates the intersection points along these edge segments. This step can take a lot of time. If you set IF1<>0, then pairs of edge segments are examined for intersections only if one of the pair has a left or a right area identifier that is zero or negative.

Step 3: ARPRAM removes coincident edge segments (those with identical endpoints). If two coincident edge segments belong to the same group, one of them is removed; if they belong to different groups, one of them is modified in such a way that it will appear to be present when looking for areas defined by edge segments in a particular group, but absent when looking for areas defined by all edge segments.

Step 4: ARPRAM searches for and removes "dangling" edges (those that do not contribute to enclosing any area). This step is skipped if IF2<>0.

Step 5: ARPRAM looks for holes in the areas defined by each edge group and draws connecting edge segments between the edge and the hole. This step is skipped if IF3<>0.

Step 6: ARPRAM adjusts area identifier information in the area map. It examines all the edge segments of each area in each group to see what area identifier should be assigned to the area, and then makes adjustments.

Step 7: Connecting lines that were inserted in step 5 are removed from the area map.

You can put edges in an area map, preprocess it, add more edges, preprocess it again, and so on.

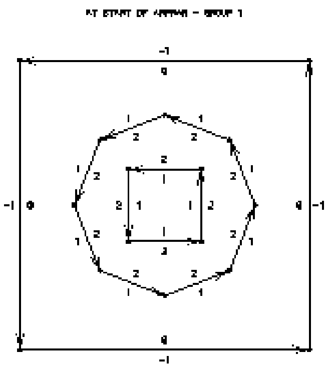
No discussion is given about the code that generates plots for the four modules in this preprocess area map group. Rather than analyzing the code, study the plots to learn how ARPRAM operates. Frames 1 through 3 from this code show the three edge groups that are added to the area map before processing, and frame 4 shows the entire area map before processing. The arrows show the direction of each edge segment, and the small numbers on each side of each edge segment are the left and right area identifiers for the segment.

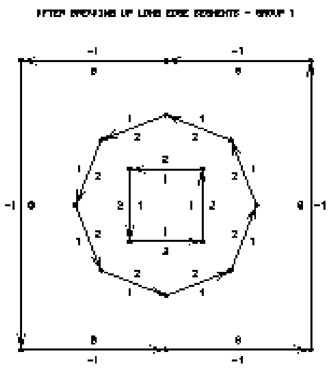
Ar 2.3.1 Preprocess area map: Breaking up edges

The first two steps that ARPRAM does break up edge segments into shorter lengths. Step 1 shortens long edges for faster processing, and step 2 finds all points where edges intersect and breaks edge segments at those points.

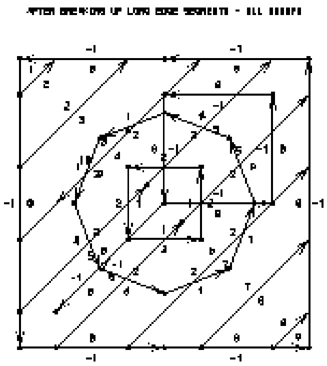
Shortened edge segments

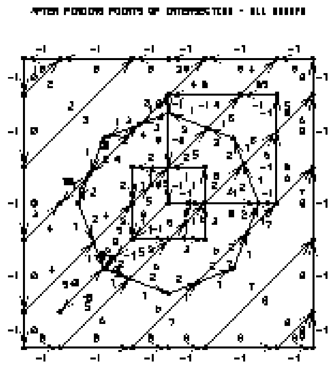
```
Frame 1  Frame 5
```





```
Frame 8  Frame 12
```





Arguments

In step 1, any edge segment whose projection on the X axis is longer than twice the average is replaced by two or more edge segments whose projections on the X axis are about the same as the average. Examine the edges in group 1 in frame 1 and note that the perimeter is drawn with unbroken top and bottom edge segments. In frame 5, however, you can see that those edge segments are divided in the middle by step 1. Step 1 is done on all edge groups in the area map to speed up later steps in processing the area map.

Step 2 breaks up edge segments at points of intersection with other edge segments. Compare frames 8 and 12; notice how Areas adds a vertex wherever an edge segment meets another edge segment, regardless of which groups the edge segments are in. It also shows that both edge segments are divided into shorter edge segments at that vertex.

More precisely, ARPRAM decides where to break edge segments if an edge segment (P,Q) is crossed by another edge segment at point I, where I is distinct from P and Q. ARPRAM then breaks (P,Q) into two edge segments, (P,I) and (I,Q). Clearly, this process can considerably increase the size of the area map. Also note that ARPRAM finds these points of intersection regardless of which group or groups the affected edge segments are in. Frame 12 shows the area map after this step is completed.

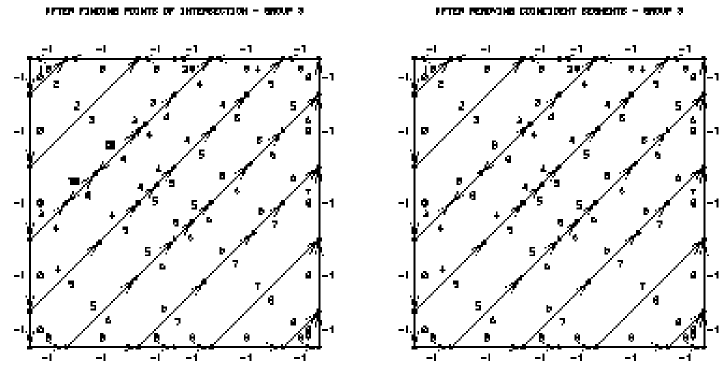
As discussed in the previous module, step 2 can be abbreviated by calling ARPRAM with IF1<>0. In this case, edge segments are examined for intersection only if one of the pair has a right or a left area identifier that is zero or negative. This is appropriate for contour lines, or if you know that you have equally "nice" edges that don't cross anywhere except at the perimeter.

Ar 2.3.2 Preprocess area map: Removing coincident and dangling edges

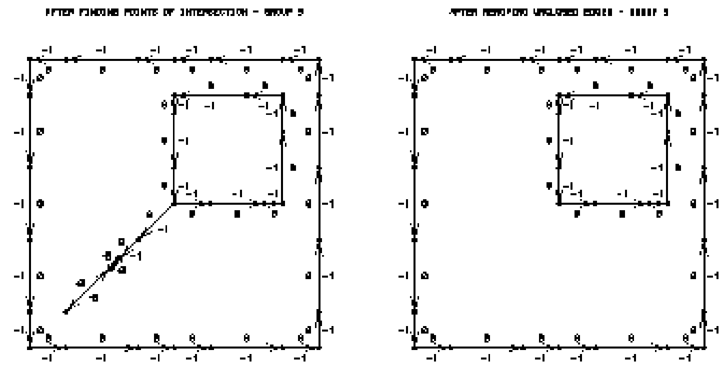
In its next two steps, ARPRAM removes coincident (overlapping) edges and dangling edges that don't contribute areas to the area map. Step 3 removes coincident edges, and step 4 removes dangling edges.

Removing coincident edges and dangling edges

`Frame 10  Frame 14`



Frame 11 Frame 19



Discussion

Step 3 searches the area map for coincident edge segments (edge segments with identical end points). If both edge segments are in the same edge group, then one of the edges is removed from the area map, and area identifier information is discarded (the area identifier is set to zero) unless it is self-consistent. For example, if you put edge segment (P,Q) into the area map with a right area identifier of 1, and later give (P’,Q’) with the same end points a right area identifier of 2, then the area map will contain a single edge segment (P,Q) with right area identifier 0. If the coincident edge segments are in different groups, then both will appear to be present when looking for areas defined by edges in the same group, and only one will appear to be present when looking for areas defined by all edges. Examine the third diagonal line from the upper left in frames 10 and 14. In frame 10, the overlapping left and right area identifiers show the coincident edges, and in frame 14, the area identifiers for the affected segments are set to zero.

This step ensures that areas are correctly traced when tracing an area boundary using the method of following rightmost or leftmost edge segments.

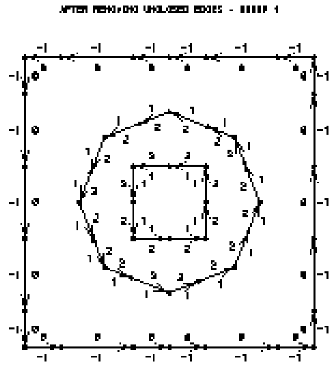
Step 4 removes dangling edge segments (edge segments that do not contribute to forming any areas). A dangling edge is any edge that has an endpoint with only one edge segment in an edge group emanating from it. Examples of dangling edges include a single line segment with endpoints not on the perimeter, short curves drawn by Ezmap to represent islands, and in frame 11, the tail on the square. Note that the tail is drawn as part of group 5; however, after step 4 is done for frame 19, the tail has been removed from the area map.

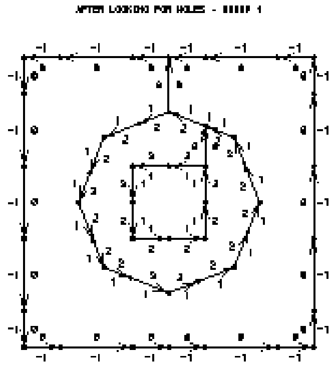
Ar 2.3.3 Preprocess area map: Reconciling area information

The last three steps that ARPRAM does are all used to reconcile area identifier information. ARPRAM adds temporary edge segments to the area map to remove any holes, then it reconciles area identifier information and removes the temporary edge segments.

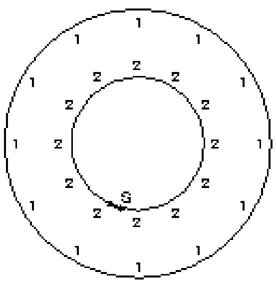
Temporary edge segments

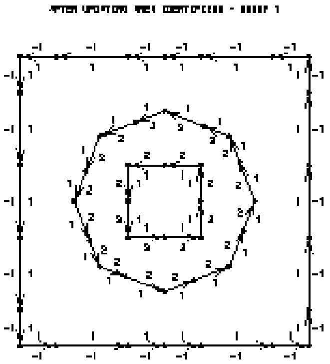
Frame 17 Frame 21





Simplified figure Frame 25





Discussion

The concept of a "hole" is hard to define precisely. Concentric circles are examples of areas with holes. Holes are shown in frame 17 on the preceding page: the square forms a hole in the area whose outer boundary is the octagon, and the octagon forms a hole in the area whose outer boundary is the perimeter. Closed contour lines are also examples of holes.

Holes are important because an area with a hole does not have a boundary that is a simple closed curve. Instead, the boundary consists of two or more simple closed curves; one of these forms the outer boundary of the area and the others form the holes. This complicates things for Areas because, in attempting to reconcile contradictory area identifier information, it can only look at one closed curve at a time; contradictory information on two different parts of the boundary of an area with holes would therefore not be reconciled.

In step 5, ARPRAM identifies holes and adds temporary edge segments to the area map to, in effect, remove the holes. These edge segments ensure that all the areas in the area map have boundaries that are simple closed curves. Thus, as ARPRAM traces the boundary of one area, it sees all of the area identifiers that have been provided by the user for that area.

Frames 17 and 21 show edge group 1 before and after step 5. Consider the area having the perimeter as an outer boundary and the octagon as a hole. Information provided with the perimeter says that its area identifier should be 0, but information provided with the octagon says that its area identifier should be 1. Step 5 inserts a vertical connecting line between the perimeter and the octagon, making it possible to trace the boundary of the area as a single simple closed curve, so that all area-identifier information can be considered at once and the contradiction can be seen and reconciled.

ARPRAM methodically traces all the area boundaries defined by an area map: it repeatedly selects an edge segment to start with and makes either leftmost turns only or rightmost turns only until it arrives back at the starting point. Edge segments seen while doing this are marked so that a given boundary is only traced once.

Holes are detected by keeping track of the total change in direction along each boundary as it is traced. Net changes in angular direction to the right (clockwise) are considered negative, and net changes in angular direction to the left (counterclockwise) are considered positive. The net change along a boundary will always be exactly +360 or -360 degrees. Either of the following two cases is sufficient to confirm the existence of a hole:

- If we always take rightmost turns and yet we trace a boundary in the counterclockwise direction (the total change in angular direction is +360 degrees), then what is to the left of our path is a hole in the area to the right.
- If we always take leftmost turns and yet we trace a boundary in the clockwise direction (the total change in angular direction is 360 degrees), then what is to the right of our path is a hole in the area to the left.

An example of a hole is given in the "Simplified figure" on the preceding page. If we start at S, and we proceed in the direction of the arrowhead, and we always take rightmost turns, the total change in direction is +360 degrees (counterclockwise). Thus, the area to the left is a hole in the area to the right.

If you set IF3 nonzero, ARPRAM will not look for holes. This should only be done when you can be absolutely sure that area identifier information for holes in an area does not contradict that for the outer boundary of the area.

In step 6, ARPRAM again traces all of the boundaries in the area map. This time, ARPRAM assumes that each such boundary defines precisely one area, and it reconciles the area identifier information given for that area with the edge segments defining that boundary. After ARPRAM has picked an area identifier for the area, it replaces all the area identifiers for those edge segments with the value selected. If any of the area identifiers examined is negative, a -1 will be used; if all of the area identifiers examined are zeroes, a 0 will be used; otherwise, the nonzero value most recently entered in the area map will be used.

In step 7, ARPRAM removes the temporary edge segments that were added in step 5.

In frame 25, note that the temporary edge segments added in step 5 have been removed and that the area identifiers on the inside of the perimeter now match the values on the outside of the octagon.

Ar 3. Producing results using Areas

After Areas is initialized, you can access and process each area in your area map individually. This flexibility allows you to do much more than the masking, filling, and line drawing discussed here. This section covers retrieving area ids and group ids, masking and drawing lines through an area map, writing a masking or line-drawing routine, filling areas, writing an area-fill routine, and describes how Areas interacts with contours and maps.

Areas functional outline

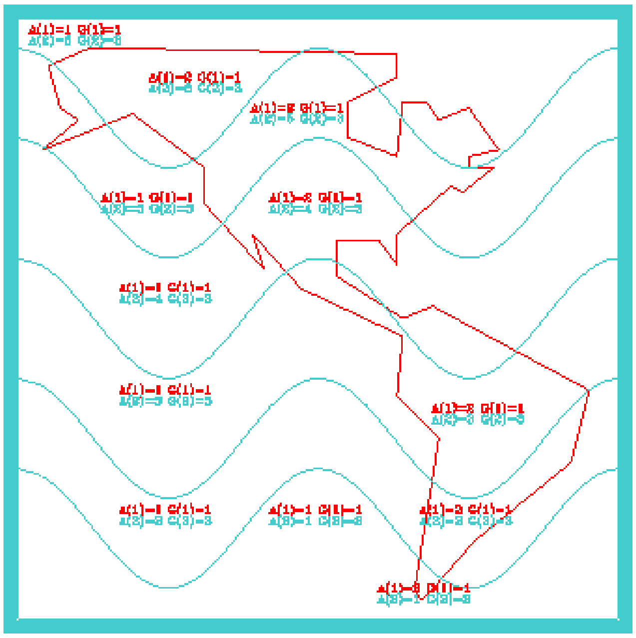
```
---------------------------------------------------------------
    1.  Open GKS
    2.  Set parameters
    3.  Initialize area map
    4.  Add edges to area map
    5.  Preprocess area map
 *  6.  Do one or more of the following:
 *      Break a polyline into segments, each of which lies
 entirely in a single area, then process each segment
 individually
 *      Retrieve area identifiers for areas containing a given
 point
 *      Scan an area map to retrieve each subarea, then
 process each subarea individually
    7.  Debug plot
    8.  Call FRAME
    9.  Close GKS
---------------------------------------------------------------
```

* Steps discussed in this section.

Ar 3.1 Obtaining area ids and group ids

In the preceding modules, you may have wondered how we got the information to print out the group and area identifier arrays for each area. ARGTAI takes a point in the user coordinate system and returns information about the area containing that point. This information can then be printed with a WRITE statement or plotted on your area map using Plotchar.

Labeling areas with group and area ids



Code segment from carmap.f

```
      DO 20, I=1, 12
         CALL ARGTAI (MAP, X(I), Y(I), IAREA, IGRP, NGRPS, NAI, 0)
         DO 30, J=1, 2
               WRITE (STRING, 11) J, IAREA(J), J, IGRP(J)
               IF (IGRP(J) .EQ. 1) THEN
                     CALL GSPLCI(1)
                     CALL PLCHHQ (X(I), Y(I), STRING, .01, 0., 0.)
               ENDIF
               IF (IGRP(J) .EQ. 3) THEN
                     CALL GSPLCI(6)
                     CALL PLCHHQ (X(I), Y(I) -.018, STRING, .01, 0., 0.)
               ENDIF
 30         CONTINUE
 20   CONTINUE
```

Synopsis

```
CALL ARGTAI (MAP, XCD, YCD, IAREA, IGRP, ISIZ, NGRPS, ICF)
```

Arguments

MAP(LMAP)

> *Integer array, Workspace*---An array containing an area map that has been initialized by a call to ARINAM and to which edges have been added by calls to AREDAM. If you did not preprocess the area map by calling ARPRAM, ARGTAI calls it before doing anything else.
> **Note:** As part of initializing the area map, ARINAM stores the dimension of MAP in MAP(1); therefore, the dimension does not have to be given as an argument in calls to ARGTAI.

XCD

Real, Input---The X coordinate, in the current user coordinate system, of a point about which you want to obtain information.

YCD

Real, Input---The Y coordinate, in the current user coordinate system, of a point about which you want to obtain information.

IAREA(ISIZ)

Integer array, Output---The array in which area identifiers for the area containing the specified point will be returned.

IGRP(ISIZ)

Integer array, Output---The array in which group identifiers for the area containing the specified point will be returned.

ISIZ

Integer, Input---Dimension of the arrays IAREA and IGRP. ISIZ>=NGRPS.

NGRPS

Integer, Output---The number of values returned in IAREA and IGRP. NGRPS equals the number of groups of edges that you put in the area map.

ICF

Integer, Input---Flag set nonzero by you to indicate that the definition of the user coordinate system has been changed since the last call to ARGTAI; in this case, calls to GETSET must be executed by ARGTAI. If you set the flag to zero, Areas assumes that the information retrieved previously is still correct and skips calls to GETSET. This option is valuable if you are planning to make thousands of calls to ARGTAI with the same area map: do the first one with ICF=1 and the rest with ICF=0.

Discussion

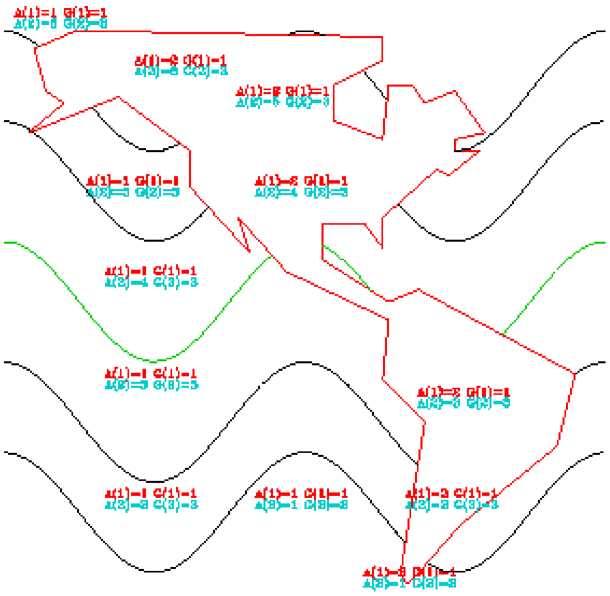
In the **carmap** example, the arrays X and Y contain the X and Y coordinates of points in the area map that are chosen to represent many of the areas. Line 1 of the **carmap.f** code segment sets up a loop so that we can find out area and group identifier information for each point. Line 2 retrieves the area and group identifier information. Since we know that geographic area identifiers are in group 1, lines 5 through 8 write them out in red, and lines 9 through 12 write the contour area identifier information out in blue.

ARGTAI has also been used to determine whether a specific point is over land or water. A user created a fine grid over the globe and used ARGTAI to determine if each point in the grid was over land or water. Then he used the resulting array to tell his model which points were over land and which points were over water.

Ar 3.2 Masking and drawing lines through an area map

ARDRLN takes a polyline (or curve) and an area map and breaks the polyline into pieces, each of which lies entirely within a single area defined by the area map. It calls user routine LPR once for each piece of the polyline. You can then program LPR to do whatever you want with the pieces. For example, you can change line color, dash pattern, or mask portions of the line that lie over selected areas. ARDRLN is most frequently used in conjunction with Ezmap to draw latitude and longitude lines over the oceans.

Controlling a line area by area



Code segment from carline.f

```
      EXTERNAL MASK
      CALL ARINAM (MAP, LMAP)
      CALL AREDAM (MAP, XGEO, YGEO, NGEO, 1, 2, 1)
      CALL ARDRLN (MAP, XCNTR, YCNTR1, NPTS, XWRK, YWRK, NWRK,
     + IAREA, IGRP, NGRPS, MASK)
      CALL ARDRLN (MAP, XCNTR, YCNTR2, NPTS, XWRK, YWRK, NWRK,
     + IAREA, IGRP, NGRPS, MASK)
      CALL ARDRLN (MAP, XCNTR, YCNTR3, NPTS, XWRK, YWRK, NWRK,
     + IAREA, IGRP, NGRPS, MASK)
      CALL ARDRLN (MAP, XCNTR, YCNTR4, NPTS, XWRK, YWRK, NWRK,
     + IAREA, IGRP, NGRPS, MASK)
      CALL ARDRLN (MAP, XCNTR, YCNTR5, NPTS, XWRK, YWRK, NWRK,
     + IAREA, IGRP, NGRPS, MASK)
```

Synopsis

```
      CALL ARDRLN (MAP, XCD, YCD, NCD, XCS, YCS, MCS, IAREA, IGRP,
     +             ISIZ, LPR)
```

Arguments

MAP(LMAP)

Integer array, Workspace---The area-map array.

XCD(NCD)

Real array, Input---The X coordinates, in the user coordinate system, of points defining the curve.

YCD(NCD)

Real array, Input---The Y coordinates, in the user coordinate system, of points defining the curve.

NCD

Integer, Input---Number of points for the curve.

XCS(MCS)

Real array, Workspace---Used in calls to LPR to hold the X coordinates of the curve segment contained in a given area.

YCS(MCS)

Real array, Workspace---Used in calls to LPR to hold the Y coordinates of the curve segment contained in a given area.

MCS

Integer, Input---Dimension of each of the arrays XCS and YCS.

IAREA(ISIZ)

Integer array, Workspace---The area identifier array used in calls to LPR.

IGRP(ISIZ)

Integer array, Workspace---The group identifier array used in calls to LPR.

ISIZ

Integer, Input---Dimension of the each of the arrays IAREA and IGRP. ISIZ>=*n*, where *n* is the number of groups in the area map.

LPR

Subroutine---A line-processing routine that you supply for ARDRLN to call. LPR must be declared external in the routine that calls ARDRLN. See the next module.

Discussion

ARDRLN overlays the curve that you give it onto the area map. It breaks the curve into pieces, one for each area that it crosses. LPR is then called with each piece of the curve. For the green contour in the **carline** example, ARDRLN breaks the contour into five pieces. The first piece is from the left edge to Mexico. ARDRLN calls the line-processing routine MASK, which checks to see if this piece is over water; if it is, ARDRLN draws it. ARDRLN then calls MASK with the second piece (over Mexico) and does not draw it because it is not over water. ARDRLN calls MASK with the third piece over the Gulf of Mexico and draws it. It does not draw the fourth piece over Central and South America, then it draws the fifth piece over the Atlantic Ocean. This shows how you can make a line-processing routine do something different with each piece of a curve.

Before executing the first call to LPR, ARDRLN calls GETSET to retrieve the current user-system mapping parameters, then executes the statement:

```
CALL SET (VPL, VPR, VPB, VPT, VPL, VPR, VPB, VPT, 1)
```

where VPL, VPR, VPB, and VPT are the viewport left, right, bottom, and top coordinates in NDCs.

This ensures correct results if the NDCs in XCS and YCS are used in calls to such routines as GPL and CURVE, and this allows clipping at the edges of the viewport. LPR may make its own SET call to achieve some other effect. Before returning control to the calling routine, ARDRLN calls SET again to restore the original mapping parameters.

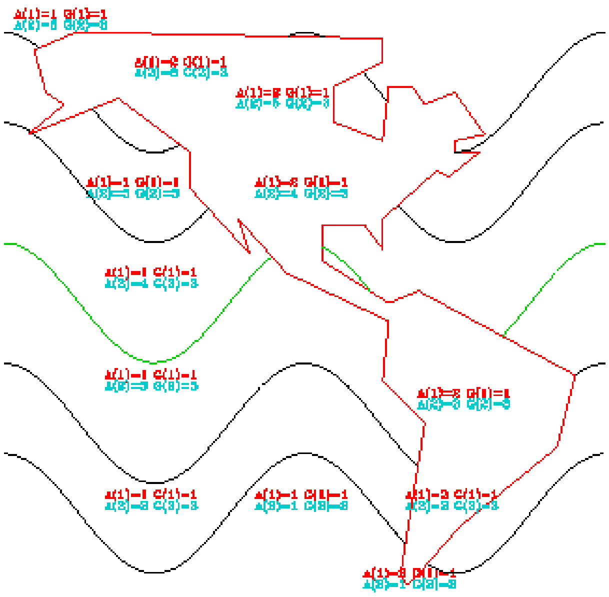
In the **carline** example, XGEO and YGEO are arrays containing the X and Y coordinate information for the "geographical" outline. XCNTR and YCNTR1-4 are the X and Y coordinate arrays for the "contour" lines. Line 1 declares our masking routine to be external so that Fortran doesn’t interpret the subroutine name as a variable. Line 2 initializes Areas. Line 3 adds the geographic outline to the area map in group 1 for consistency with Ezmap defaults. Lines 4 through 8 show examples of calling ARDRLN to mask the contour lines over land masses, where MASK is the actual line-drawing and masking routine.

When drawing lines with ARDRLN, it is best not to add the lines to the area map.

Ar 3.3 Writing a masking or line-drawing routine

By itself, Areas won’t actually do anything after it defines areas that can be acted on. You have to write one or two routines that actually do something because only you know what you want done inside the defined areas. The following subroutine is an example that checks which area the line is in and decides what color the line should be before drawing it. It also decides when not to draw the line.

Controlling a line, area by area



Code segment from carline.f

```
      SUBROUTINE MASK (XC, YC, MCS, IAREA, IGRP, NGRPS)
      INTEGER IAREA(NGRPS), IGRP(NGRPS), IDGEO
      REAL XC(MCS), YC(MCS)
      IDGEO = -1
      DO 10, I=1, NGRPS
         IF (IGRP(I) .EQ. 1) IDGEO = IAREA(I)
  10  CONTINUE
      IF (IDGEO .EQ. 1) THEN
         CALL CURVED (XC, YC, MCS)
      ENDIF
      RETURN
      END
```

Synopsis

```
SUBROUTINE LPR (XCS, YCS, NCS, IAREA, IGRP, ISIZ)
DIMENSION XCS(*), YCS(*), IAREA(*), IGRP(*)
```

Arguments

LPR

Subroutine---An area map line-processing routine you supply for ARDRLN to call. LPR must be declared external in the routine that calls ARDRLN.

```
SUBROUTINE LPR (XCS, YCS, NCS, IAREA, IGRP, ISIZ)
DIMENSION XCS(*), YCS(*), IAREA(*), IGRP(*)
```

(code to process polyline defined by XCS, YCS, IAREA, and IGRP)

```
RETURN
END
```

XCS(NCS), YCS(NCS)

Real arrays, Input---Hold the X and Y coordinates, in NDCs, of NCS points defining a piece of the original polyline that is completely contained in one of the areas defined by the area map.

NCS

Integer, Input---Number of X and Y coordinates in the arrays XCS and YCS.

IAREA(ISIZ), IGRP(ISIZ)

Integer arrays, Input---Hold ISIZ pairs of area identifiers and group identifiers, respectively, for the area in which the piece of the polyline lies. For each value of I from 1 to ISIZ, IAREA(I) is the area identifier for the area with respect to the group of edges specified by the group identifier IGRP(I).

ISIZ

Integer, Input---Number of values given in IAREA and IGRP. ISIZ>=*n*, where *n* is the number of groups of edges that you put in the area map.

Discussion

If the group identifier is 1, then we know the following:

- The area outside the perimeter has area identifier -1
- The areas over ocean have area identifier 1
- The areas over land have area identifier 2

Since we want to draw the contour lines only over ocean in this case, we first retrieve the area identifier for group 1 in lines 5 through 7 of the **carline.f** code segment. Because there is more than one group, it is necessary to check each group identifier individually in a DO loop. If the area identifier for group 1 is 1, then we draw the contour line segment in line 9.

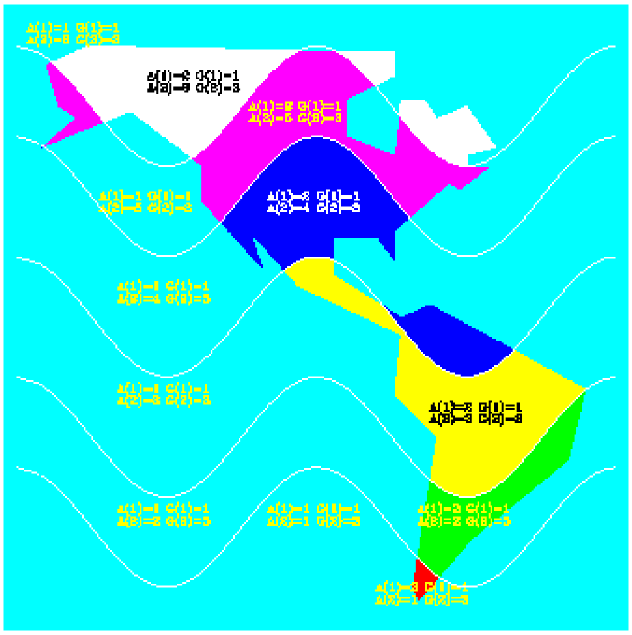
Exercises

1. Using the **carline** example, mask the areas over water and not land.
2. Draw a vertical line down the middle of the plot, and give a different color to the segment passing through each contour level.

Ar 3.4 Filling areas

Given an area map, ARSCAM scans it from left to right, extracting the definitions of all the areas in the area map. ARSCAM then processes each area using the user routine APR. Since you supply APR, you can use ARSCAM to do almost anything with the areas. ARSCAM is most frequently used to color-fill or pattern-fill regions.

Filling areas in an area map



Code segment from carfill.f

```
      EXTERNAL FILL
      CALL GSFAIS (1)
      CALL ARINAM (MAP, LMAP)
      CALL AREDAM (MAP, XGEO, YGEO, NMAP, 1, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR1, NPTS, 3, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR2, NPTS, 3, 3, 2)
      CALL AREDAM (MAP, XCNTR, YCNTR3, NPTS, 3, 4, 3)
      CALL AREDAM (MAP, XCNTR, YCNTR4, NPTS, 3, 5, 4)
      CALL AREDAM (MAP, XCNTR, YCNTR5, NPTS, 3, 6, 5)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS, FILL)
```

Synopsis

```
      CALL ARSCAM (MAP, XCS, YCS, MCS, IAREA, IGRP, ISIZ, APR)
```

Arguments

MAP(LMAP)

> *Integer array, Workspace*---An array containing an area map that has been initialized by a call to ARINAM and to which edges have been added by calls to AREDAM. If you did not preprocess the area map by calling ARPRAM, ARSCAM calls it before doing anything else.
> **Note:** As part of initializing the area map, ARINAM stores the dimension of MAP in MAP(1); therefore, the dimension does not have to be given as an argument in calls to ARSCAM.

XCS(MCS), YCS(MCS)

Real arrays, Workspace---Hold the X and Y coordinates defining the edge of a given area, for use by ARSCAM in calls to the area-processing routine APR.

MCS

Integer, Input---Dimension of each of the arrays XCS and YCS.

IAREA(ISIZ), IGRP(ISIZ)

Integer arrays, Workspace---Hold ISIZ pairs of area identifiers and group identifiers, respectively. Used by ARSCAM in calls to the area-processing routine APR.

ISIZ

Integer, Input---Dimension of each of the arrays IAREA and IGRP. ISIZ>=*n*, where *n* is the number of groups in the area map.

APR

Subroutine---An area-processing routine that is called by ARSCAM and that must be declared external in the routine that calls ARSCAM. You must supply the routine APR; it is described in detail in the next module.

Discussion

When ARSCAM is called by **carfill.f**, it locates the contour band at the bottom of the plot located over the Pacific Ocean, and passes that area to the area-processing routine FILL. Because the area is over the ocean, FILL colors the area blue; it then returns to ARSCAM. The next area ARSCAM locates and passes to FILL is the portion of the contour band directly above the first. FILL colors the area blue and returns to ARSCAM. After ARSCAM has passed all the contour bands over the Pacific to FILL, the next area it finds is over the western part of Alaska. ARSCAM passes this area to FILL, which recognizes that the area is in the fifth contour band and over land, so it fills the area with the color specified for the fifth contour band over land: magenta. Next, ARSCAM finds the area over eastern Alaska and northwest Canada and passes it to FILL. FILL recognizes that the area is in the sixth contour band and over land, so it fills it with white. ARSCAM then locates the area over central Canada and passes it to FILL. FILL recognizes that the area is in the fifth contour band and over land, so it fills it with magenta, and so on.

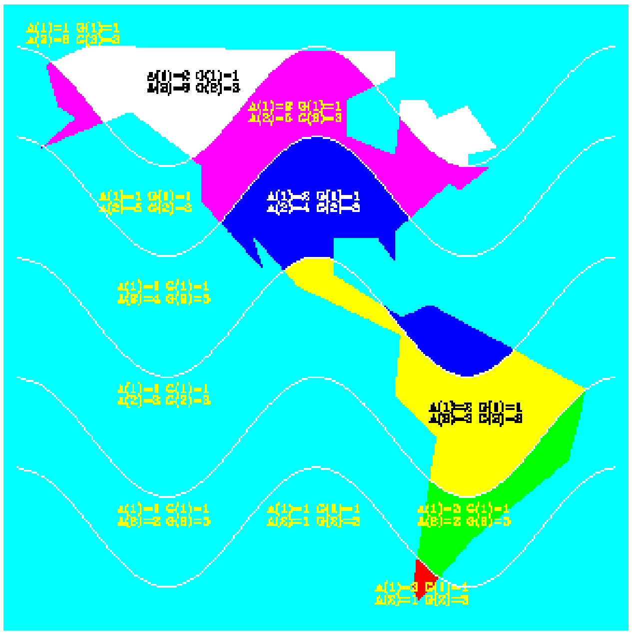
All regions having the same group and area identifiers are treated the same way. In this example, every separate piece of a single contour band over land is filled with the same color. This is why the color of western Alaska matches the color of central Canada. If you want to color a contour over Canada differently than the same contour over the US, you would have to add the political boundaries to your area map, then give Canada a different area identifier than the US. Fortunately, Ezmap can do this for you.

Line 1 of the **carfill.f** code segment declares our filling routine FILL to be external so that when we call it, Fortran doesn't interpret the subroutine name as a variable and then core dump. Line 2 sets the GKS fill style to be solid because the default is "hollow" fill. Line 3 initializes Areas. Lines 4 through 9 add "geographic" and "contour" lines to the area map using group 1 for geographic outlines and group 3 for contour lines; this ensures consistency with Conpack and Ezmap defaults. Line 10 does the area fill by calling ARSCAM, which examines the area map and feeds the appropriate information to the fill routine FILL. The next module describes the FILL routine.

Ar 3.5 Writing an area-fill routine

The Areas utility defines areas that can be acted on. You must decide what to do with these areas and then supply a routine that takes the appropriate action on the area or the line going through an area. Below is an example subroutine that supplies color and fill type for each area defined by an area map.

Filling areas in an area map



Code segment from carfill.f

```
      SUBROUTINE FILL (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)
      INTEGER IAREA(NGRPS), IGRP(NGRPS), IDGEO, IDCNTR
      REAL XWRK(NWRK), YWRK(NWRK)
      IDGEO = -1
      IDCNTR = -1
      DO 10, I=1, NGRPS
         IF (IGRP(I) .EQ. 1) IDGEO = IAREA(I)
         IF (IGRP(I) .EQ. 3) IDCNTR = IAREA(I)
   10 CONTINUE
      IF (IDGEO .EQ. 1) THEN
         CALL GSFACI(7)
         CALL GFA (NWRK, XWRK, YWRK)
      ELSE IF ((IDGEO .EQ. 2) .AND. (IDCNTR .GE. 1)) THEN
         CALL GSFACI(IDCNTR)
         CALL GFA (NWRK, XWRK, YWRK)
      ENDIF
      RETURN
```

```
18      END
```

Synopsis

```
SUBROUTINE APR (XCS, YCS, NCS, IAREA, IGRP, ISIZ)
DIMENSION XCS(*), YCS(*), IAREA(*), IGRP(*)
```

Arguments

APR

Subroutine---An area-processing routine that is called by ARSCAM and must be declared external in the routine that calls ARSCAM. You must supply the routine APR, and it must have the following form:

```
SUBROUTINE APR (XCS, YCS, NCS, IAREA, IGRP, ISIZ)
DIMENSION XCS(*), YCS(*), IAREA(*), IGRP(*)
```

(code to process the area defined by XCS, YCS, IAREA, and IGRP)

```
RETURN
END
```

XCS(NCS), YCS(NCS)

Real arrays, Input---The X and Y coordinates, in NDCs, of NCS points defining a polygonal area. The last of these points is a duplicate of the first. Holes in an area are traced in a way that maximizes the probability of hardware fill working properly.

NCS

Integer, Input---Number of X and Y coordinates in the arrays XCS and YCS.

IAREA(ISIZ), IGRP(ISIZ)

Integer arrays, Input---Hold ISIZ pairs of identifiers for the area defined by XCS and YCS. For each value of I from 1 to ISIZ, IAREA(I) is the area identifier for the area with respect to the group of edges specified by the group identifier IGRP(I).

ISIZ

Integer, Input---Number of values returned in IAREA and IGRP. ISIZ>=*n*, where *n* is the number of groups of edges that you put in the area map.

Discussion

In the **carfill** example, we need to distinguish between both geographic areas and contour levels since we'll only be filling contour levels over land. For this reason, lines 4 and 5 of the **carfill.f** code segment set our area identifier storage variables to a negative number; this avoids using information from a previous call to FILL. Lines 6 through 9 check every value in the group identifier array; if the group identifier is 1, then we store the associated area identifier in IDGEO. If the group identifier is 3, we store the associated area identifier in IDCNTR. Because of the way that Areas works, there is no way to guarantee that the first value in the area identifier array will be from your first group, the second value in the array will be from your second group, and so on.

After the area identifiers have been obtained, line 10 checks to see if we're over the ocean. If we are, then lines 11 and 12 fill the area in aquamarine. Line 13 checks to see if we're over land, and if so, then lines 14 and 15 fill the contour band with a color based on its area identifier.

APR can be used to do almost anything you want with each area in the area map. However, it is rarely used for anything except filling areas. If you are unable to produce color hardcopy output, you may want to write your fill routine so that it calls the Softfill utility.

Before executing the first call to APR, ARSCAM executes the statement:

```
      CALL SET (VPL, VPR, VPB, VPT, VPL, VPR, VPB, VPT, 1)
```

where VPL, VPR, VPB, and VPT are the viewport left, right, bottom, and top coordinates in NDCs. This ensures correct results if the NDCs in XCS and YCS are used in calls to such routines as GFA and FILL, and this allows clipping at the edges of the viewport. APR may make its own SET call to achieve some other effect. Before returning control to the calling routine, ARSCAM calls SET again to restore the original mapping parameters.

Exercises

1. Fill the contour bands over water in different colors, and fill the land areas in white.

Ar 4. Debugging Areas

Because Areas is used with various other packages, and because you may not always have a complete understanding of how edge groups and area identifiers are assigned in these different cases, it is often helpful to debug a problem by drawing edge segments and area identifiers for one or more groups of edges. This section describes Areas debugging tools that can help in these situations.

Areas functional outline

```
--------------------------------------------------------------
    1.  Open GKS
 *  2.  Set parameters
    3.  Initialize area map
    4.  Add edges to area map
    5.  Preprocess area map
    6.  Do one or more of the following:
        Break a polyline into segments, each of which lies
 entirely in a single area, then process each segment
 individually
        Retrieve area identifiers for areas containing a given
 point
        Scan an area map to retrieve each subarea, then
 process each subarea individually
```

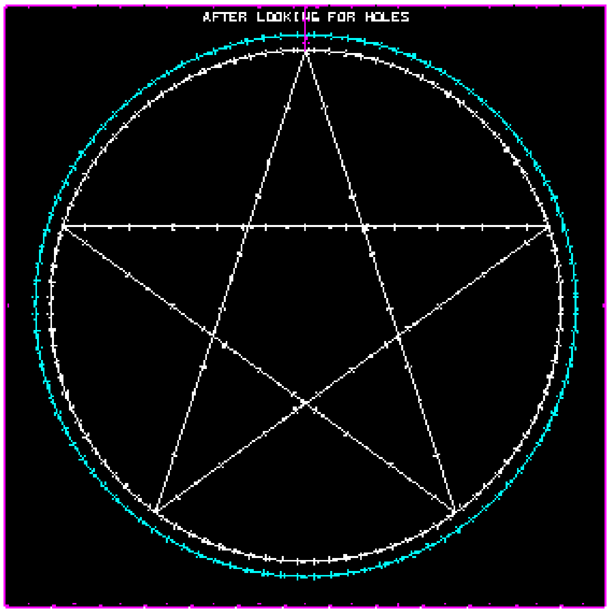
```
 *   7.   Debug plot
     8.   Call FRAME
     9.   Close GKS
---------------------------------------------------------------
```

* Steps discussed in this section.

Ar 4.1 Debugging options: Setting parameters

Areas currently has one debugging parameter, the debug plots flag DB. DB turns on debugging, and this directs ARPRAM to produce a series of plots that show all the edges belonging to a specified group in the area map at selected break points.

Checking area identifiers



Code segment from cardb1.f

```
        CALL ARINAM (MAP, MAPSIZ)
        CALL ARSETI ('DB - DEBUG PLOTS FLAG', 1)
        CALL AREDAM (MAP, X1, Y1, NPTS, 1, 1, 0)
        CALL AREDAM (MAP, X2, Y2, NPTS, 1, 2, 1)
        CALL AREDAM (MAP, X3, Y3, NVERT, 1, 3, 2)
        CALL ARSCAM (MAP, XC, YC, MCS, AREAID, GRPID, IDSIZE, FILL)
```

Synopsis

```
      CALL ARSETI ('DB', idb)
```

Arguments

DB

Integer---The DeBug plots flag is available to help you understand an area map at various stages of the Areas run.

0 Debugging is off. This is the default.

>=1 Debugging is on. At selected break points, ARPRAM produces plots showing all edge segments in the area map that have the group identifier specified by DB.

Discussion

Line 1 of the **cardb1.f** code segment initializes Areas. Line 2 turns on debugging. ARDBPA is then automatically called to produce a plot at each of seven different break points:

- The start of ARPRAM
- After breaking up long edge segments
- After finding points of intersection
- After removing coincident segments
- After removing unclosed edges
- After looking for holes
- After updating area identifiers

Lines 3 through 5 add edges to our area map. Line 6 forces a call to ARPRAM, during which the calls to ARDBPA occur, generating the seven frames listed above. This occurs before ARSCAM fills the areas, generating an eighth frame, which is the final plot.

In each of the debug plots, notice the dots on each side of each line segment. These are numbers that represent the left and right area identifiers for that edge. To view them, you need to use the zoom feature in **idt**.

The area identifiers are drawn very small so that each edge segment in an area map can be marked with both left and right area identifiers.

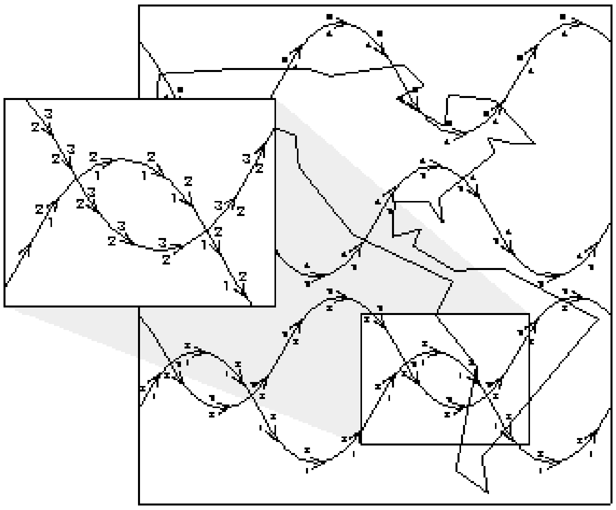
Exercises

1. Turn debugging on in the **carline** example, and see if you understand each step in the process.

Ar 4.2 Debugging options: Drawing parts of the area map

The routine ARDBPA produces a plot showing all the edges in an area map with a specified group identifier. In complex area maps, this allows you to debug a problem, group by group.

Area identifiers for crossing contours



Code segment from cardb2.f

```
      CALL ARINAM (MAP, LMAP)
      CALL AREDAM (MAP, XGEO, YGEO, NMAP, 1, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR1, NPTS, 3, 2, 1)
      CALL AREDAM (MAP, XCNTR, YCNTR2, NPTS, 3, 3, 2)
      CALL AREDAM (MAP, XCNTR, YCNTR3, NPTS, 3, 4, 3)
      CALL AREDAM (MAP, XCNTR, YCNTR4, NPTS, 3, 5, 4)
      CALL AREDAM (MAP, XCNTR, YCNTR5, NPTS, 3, 6, 5)
      CALL ARDBPA (MAP, 3, 'Crossing Contours')
```

Synopsis

```
CALL ARDBPA (MAP, IGRP, LABEL)
```

Arguments

MAP(LMAP)

> *Integer array, Workspace*---An array containing an area map that has been initialized by a call to ARINAM and to which edges have probably been added by calls to AREDAM.
> **Note:** As part of initializing the area map, ARINAM stores the dimension of MAP in MAP(1); therefore, the dimension does not have to be given as an argument in calls to ARDBPA.

IGRP

> *Integer, Input*---The group identifier of the group that you want to examine.

LABEL

> *Character expression, Input*---The label you want on the plot.

Discussion

Line 1 of the **cardb2.f** code segment initializes the area map. Line 2 puts our "Americas map" into group 1. Lines 3 through 7 add the contour lines to group 3. Notice that in this case, we have chosen contour lines that cross. Line 8 calls ARDBPA to examine edge group 3, which includes the crossing contours. Use the zoom feature in **idt** to resolve the small dots near the arrows on the contour lines into left and right area identifiers.

In the zoomed portion of the picture, notice how the left and right area identifiers are in conflict in the areas above and below the crossed contour lines. This situation occasionally arises when using smoothing in Conpack. When Areas is used in a situation like this, the contour bands may be filled in an undesired fashion, since the areas do not have well-defined identifiers.

By default, each edge in the plot appears in one of four different colors, depending on whether the area identifiers to the left and right are less than or equal to zero or greater than zero, as follows:

```
--------------------------------
Color    Left area   Right area
         identifier  identifier
--------------------------------
Magenta  <=0       <=0
Yellow   <=0       >0
Cyan     >0        <=0
White    >0        >0
--------------------------------
```

You can adjust these colors with the DC parameter.

In some cases, you may notice gray lines in your plot. This means that the same edge occurs in more than one group. In all but one of those groups, Areas negates the group identifier for the edge in the area map. This allows Areas to include the edge when it is looking at a particular group (as in ARPRAM), but omit it when it is looking at the union of all the groups (as in ARSCAM).

If ARDBPA is used for a complicated area map, the amount of output can be very large.

Exercises

1. Using the **cardb2** example and ARDPA, examine the area identifiers for edge group 1. Why is the Americas map drawn in cyan?

Ar 5. Areas parameter descriptions

Areas currently supports the following parameters.

AT

> *Integer*---Arithmetic Type flag. AT allows the user to specify the type of arithmetic to be used by Areas or to find out what type it used.
>
> **<=0** Areas decides what sort of arithmetic to use.
> **1** Real arithmetic is to be used.
> **2** Double precision arithmetic is to be used.

3 Multiple precision arithmetic is used; Areas should choose the base value.
>=4 Multiple precision arithmetic is used and AT specifies the base value. (For example, the value 100 would specify the use of base-100 multiple-precision integer arithmetic.)

Default: 0

Note: You should use a nonzero value of AT *only* on the recommendation of an NCAR consultant.

DB

Integer---DeBug plots flag. DB allows the user to study the area map.

0 Debugging turned off. This is the default.
>=1 Debugging is on. At selected break points, ARPRAM produces plots showing all edge segments in the area map that belong to the group with group identifier DB.

DC

Integer---Debug Colors index. DC tells ARDBPA to use color indices DC+1 through DC+5 for debugging colors.

By default, DC=100, so ARDBPA defines and uses color indices 101 through 106.

DI

Integer---Direction Indicator flag. DI tells the user the direction in which the edge of an area is given. For retrieval only, the value of DI cannot be set by the user. The two possible values of DI are:

1 Edge of the area is given in counterclockwise order (with the interior to the left).
2 Edge of the area is given in clockwise order (with the interior to the right).

Note: These values are only meaningful when the ARGETI call originates from the user-written routine (dummy argument "APR") that is called by ARSCAM. It gives you information about the area whose polygonal boundary is defined by the values of APR's arguments.

LC

Integer---Largest Coordinate. LC lets you specify the largest coordinate allowed in an area map. X and Y coordinates in an area map are represented by integers in the range from 0 to LC, inclusive; the default value of LC is 1000000.

The minimum value allowed for LC is 1000; attempting to set LC<1000 gives it the value 1000. The value of LC must not be greater than the largest integer on the machine on which Areas is running; its value must also be exactly representable as a real number on that machine.

Using Ezmap

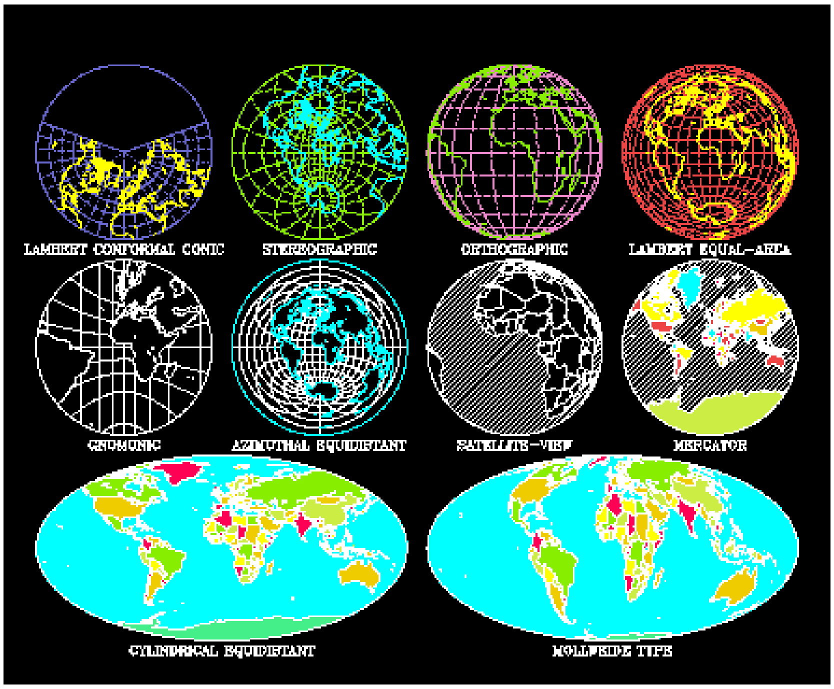
Table of contents

Using Ezmap

Mp 1. What is Ezmap?

Ezmap is a color mapping toolkit you can use to generate ten different projections of the globe with various continental outlines, political boundaries, and optional latitude and longitude lines. Ezmap was previously two utilities, Ezmap and Ezmapa.

Examples of Ezmap



Mp 1.1 Table of Ezmap user entry points

The Ezmap utility uses more than one routine to draw maps. The routines listed here cover all Ezmap tasks you may need to perform. This module organizes the routines according to their functions.

Single-call entry point

SUPMAP

Single-call map-drawing routine. See module Mp 1.7.

Map initialization routines

MAPINT

Initializes Ezmap. See modules Mp 3.1 and Mp 4.3.

MAPBLA

Adds the geographical map to an area map. See module Mp 4.3.

Latitude, longitude, and limb line drawing routines

MAPGRD

Draws latitude/longitude grid. See module Mp 3.2.

MAPGRM

Draws latitude/longitude grid with masking. See module Mp 4.5.

Labeling routine

MAPLBL

Labels map. See modules Mp 3.4 and Mp 4.4.

Map drawing and control routines

MAPLOT

Draws geographical and political outlines. See module Mp 3.5.

MAPDRW

Draws a complete map after user sets basic parameters. See module Mp 3.6.

MAPUSR

Changes line attributes for various parts of the map. See module Mp 2.7.

Positioning routines

MAPPOS

Positions map in frame. See module Mp 2.1.

MAPROJ

Sets global map projection. See module Mp 2.2.

MAPSET

Specifies rectangular portion of the u/v plane to draw. See module Mp 2.5.

MAPRS

Re-executes a call to SET. See module Mp 2.10.

Point and line routines

MAPTRA

Projects points onto a geographic map, returns a special value for points that are unprojectable and for points outside the perimeter. See module Mp 5.1.

MAPTRN

Projects points onto a geographic map, returns a special value for points that are unprojectable, but not for points outside the perimeter.

MAPEOD

Examines each outline-dataset segment. See module Mp 2.8.

MAPIT

Draws a line on a map, omitting nonvisible portions. See module Mp 5.3 .

MAPIQ

Terminates a line drawn by MAPIT. See module Mp 5.3.

MAPITA

Used to insert a line in an area map. See module Mp 5.5.

MAPIQA

Used to terminate insertion of a line in an area map. See module Mp 5.5.

MAPITM

Used to draw a masked line. See module Mp 5.6.

MAPIQM

Used to terminate drawing of a masked line. See module Mp 5.6 .

MAPGCI

Returns points interpolated on a great circle between two points. See module Mp 5.4.

Color function

MAPACI

Obtains a color index for a given area such that no two adjacent areas have the same color index. See module Mp 4.1.

Inverse transformation routine

MAPTRI

Computes the latitude and longitude of a point from its u/v coordinates. See module Mp 5.2.

Parameter routines

MAPSAV

Saves current state. See modules Mp 1.9 and Mp 2.10.

MAPRST

Restores state saved by MAPSAV. See modules Mp 1.9 and Mp 2.10.

MPSETC

Sets character parameters. See module Mp 1.9.

MPSETI

Sets integer parameters. See module Mp 1.9.

MPSETR

Sets real parameters. See module Mp 1.9.

MPGETC

Gets character parameters. See module Mp 1.9.

MPGETI

Gets integer parameters. See module Mp 1.9.

MPGETR

Gets real parameters. See module Mp 1.9.

Mp 1.2 Table of Ezmap parameters

This table is a quick reference guide to the examples in which Ezmap parameters are used and the modules in which Ezmap parameters are described. More complete descriptions appear in section "Mp 7. Ezmap parameter descriptions."

The behavior of a typical routine in an NCAR Graphics utility is sometimes determined entirely by the routine's arguments, but frequently it is also affected by the value of one or more of the utility's parameters. A "parameter" is a variable that controls the behavior of a utility; parameters are accessed via parameter-access routines that can set or retrieve the parameter value.

Instructions for setting and retrieving Ezmap parameters are provided in module "Mp 1.9 Ezmap parameters: What they do and how to use them."

Parameter	Brief description	Fortran type	Examples	Module
AR	ARea	Character	---	Mp 7.
C1	Color index 1 (for perimeter)	Integer	cmpclr	Mp 2.6
C2	Color index 2 (for grid)	Integer	cmpclr	Mp 2.6
C3	Color index 3 (for labels)	Integer	cmpclr	Mp 2.6
C4	Color index 4 (for limb line)	Integer	cmpclr	Mp 2.6
C5	Color index 5 (for continents)	Integer	cmpclr	Mp 2.6
C6	Color index 6 (for US states)	Integer	cmpclr	Mp 2.6
C7	Color index 7 (for countries)	Integer	cmpclr	Mp 2.6
DA	grid DAshline pattern	Integer	cmpdd	Mp 3.3
DD	Distance between Dots	Real	cmplot	Mp 3.5
DL	Dotted Line flag	Integer	---	Mp 7.
DO	Dotted Outline flag	Integer	cmplot	Mp 3.5
EL	ELliptical perimeter flag	Integer	cmpel	Mp 2.9
ER	ERror	Integer	---	Mp 7.
G1	Group 1 (group identifier for geo graphic entities and perimeter)	Integer	cmpgrp	Mp 4.2
G2	Group 2 (group identifier for vertical strips)	Integer	cmpgrp	Mp 4.2
GD	Grid Drawing resolution	Real	cmpgrd	Mp 3.2
GR	GRid spacing	Real	cmpgrd	Mp 3.2
IN	INitialization flag	Integer	---	Mp 7.

LA	meridian/pole LAbel flag	Integer	cmplbl, cmplab	Mp 3.4, Mp 4.4
LS	Label Size	Integer	cmplbl, cmplab	Mp 3.4, Mp 4.4
MV	Minimum Vector length	Real	cmplot	Mp 3.5
OU	OUtline data flag	Character	cmpou	Mp 2.4
P1	**P**LM**1**(1) value	Real	---	Mp 7.
P2	**P**LM**2**(1) value	Real	---	Mp 7.
P3	**P**LM**3**(1) value	Real	---	Mp 7.
P4	**P**LM**4**(1) value	Real	---	Mp 7.
P5	PLM1(2) value	Real	---	Mp 7.
P6	PLM2(2) value	Real	---	Mp 7.
P7	PLM3(2) value	Real	---	Mp 7.
P8	PLM4(2) value	Real	---	Mp 7.
PE	PErimeter flag	Integer	cmpel	Mp 2.9
PN	**P**LO**N** value	Real	---	Mp 7.
PR	PRojection specifier value	Character	---	Mp 7.
PT	**P**LA**T** value	Real	---	Mp 7.
RE	REsolution	Integer	cmplot	Mp 3.5
RO	ROtation	Real	---	Mp 7.
SA	SAtellite view distance	Real	cmpsat	Mp 2.3
S1	Satellite viewing angle	Real	cmpsat	Mp 2.3
S2	Satellite view projection	Real	cmpsat	Mp 2.3
SR	lat/lon Search Radius	Real	---	Mp 7.
VS	Vertical-Stripping parameter	Integer	cmpgrp	Mp 4.2
XL	XLOW value	Real	---	Mp 7.
XR	XROW value	Real	---	Mp 7.
YB	YBOW value	Real	---	Mp 7.
YT	YTOW value	Real	---	Mp 7.

Mp 1.3 Table of Ezmap map projections

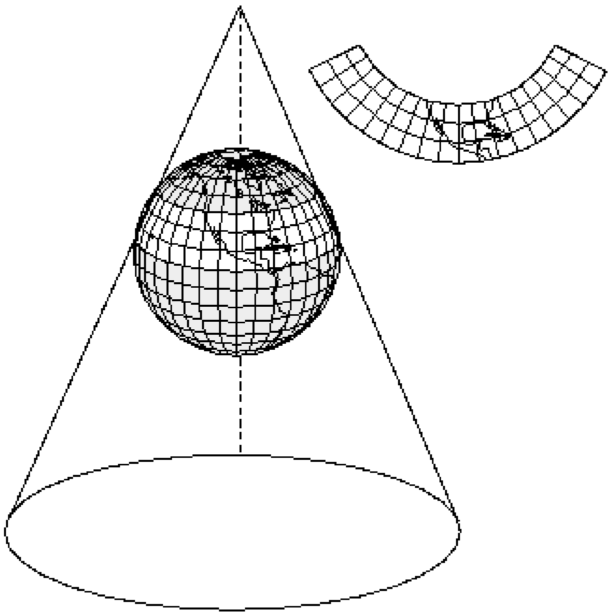
Abbreviation	Projection	Type
LC	Lambert Conformal	Conic
ST	STereographic	Azimuthal
OR	ORthographic	Azimuthal
LE	Lambert Equal-area	Azimuthal
GN	GNomonic	Azimuthal
AE	Azimuthal Equidistant	Azimuthal
SV	Satellite View	Azimuthal
CE	Cylindrical Equidistant	Cylindrical
ME	MErcator	Cylindrical
MO	MOllweide-type	Cylindrical

Mp 1.4 Description of projections: Conic

This module provides background information; it is not required for using or understanding the Ezmap utility.

Ezmap uses three different kinds of projections to project maps of the Earth onto a two-dimensional surface: conic, azimuthal, and cylindrical. Conic projections map the surface of the earth onto a cone which is either tangent to the earth along a single circle or intersects it along two different circles. The cone is then cut from point to mouth, and spread out flat.

Illustration of a conic projection



Discussion

The only conic projection offered by Ezmap is the Lambert Conformal projection. In this projection, the cone passes through the earth at two user-specified latitudes, usually both in the same hemisphere. The cone is cut on the opposite side of the globe from a user-specified meridian (longitude). The cone is then laid flat on the plotter plane with either the north pole or the south pole (depending on your specified latitudes) at the origin.

Mathematically, if LAT1 and LAT2 are the latitudes where the cone passes through the globe, and LAT1<>LAT2, then the "cone constant" is given by:

```
        LOG (COS(LAT1)) - LOG (COS(LAT2))
CONE = -------------------------------------------------
        LOG (TAN (45-S*LAT1/2)) - LOG (TAN (45-S*LAT2/2))
```

where S=1 in the northern hemisphere, and S=-1 in the southern hemisphere.

CONE*360 is the angular separation between the edges of the cut after the cone is opened onto the plane, as measured across the surface of the flattened cone. If (RLAT, RLON) is a point to be projected, then the following formulas give the coordinates of the projected point in the plotter plane.

```
R = (TAN (45-S * RLAT/2)) ** CONE
U = R * SIN (CONE * (RLON-CLON))
V = -S * R * COS (CONE * (RLON-CLON))
```

where CLON is the longitude of the central meridian.

If LAT1=LAT2, then the cone is tangent to the globe along the single standard parallel and

```
CONE = COS (90 - S * LAT1)
```

The vertex of the cone is at a distance from the plane of the equator given by D=R/SIN(LAT1), where R is the radius of the earth and LAT1 is the latitude of the single standard parallel. Note that as LAT1 approaches zero, D approaches infinity, and, as LAT1 approaches 90 degrees, D approaches R, the radius of the earth.

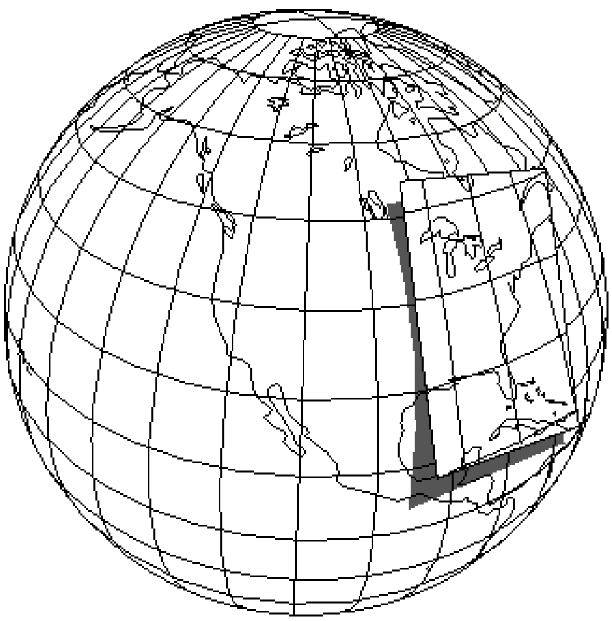
The entire globe projects onto the u/v plane minus a wedge with its apex at the origin. This projection is best used to depict midlatitude regions of limited extent, where it is relatively free of distortion. The Lambert Conformal projection preserves angles. A portion of the u/v plane determined by the MAPSET call is the user coordinate system for drawing.

Mp 1.5 Description of projections: Azimuthal

This module provides background information; it is not required for using or understanding the Ezmap utility.

The second kind of projection that Ezmap uses to project a map of the earth onto the plotter frame is the azimuthal projection. Azimuthal projections map the globe onto a plane whose origin touches the earth at the user-specified point (PLAT, PLON). The image may be rotated by the user-specified angle ROTA.

Illustration of an azimuthal projection



Discussion

Ezmap generates azimuthal projections as follows:

1. Touch a plane (called the u/v plane here) to the earth at latitude 0 and longitude 0 (where the Greenwich parallel meets the equator); the earth is oriented with the North Pole at the top and the South Pole at the bottom.
2. Rotate the earth about its polar axis until the v axis is tangent to the meridian identified by PLON.
3. Rotate the earth, tilting one of the poles towards the plane until the point (PLAT, PLON) is touching the plane at its origin.
4. Rotate the earth clockwise through the angle ROTA about a line perpendicular to the u/v plane passing through the origin.
5. Use lines emanating from a central point within or behind the earth (depending on the projection) to project the globe onto the u/v plane.
6. Set up scales along the u/v axes.
7. Draw a rectangular or elliptical portion of the resulting map. This portion of the u/v plane is the user coordinate system.

If A is the angular separation, in degrees, of the point P to be projected from the point (PLAT, PLON), and if R is the linear distance of the projected point P' from the u/v origin, then the azimuthal projections can be described as relationships between A and R:

Stereographic

```
R = TAN(A/2) = (1-COS(A)) / SIN(A)
```

As A approaches 180 degrees, R approaches infinity. The entire globe is projected onto the entire u/v plane. In practice, distortion becomes great when A is approximately 127 degrees or more. The center of the projection is the point on the earth's surface opposite the point of tangency with the projection plane.

Orthographic

```
R = SIN(A)
```

Points for which A>90 degrees are treated as invisible. Thus, a hemisphere is projected inside a circle of radius 1. The center of the projection is at infinity. All projection lines are parallel to each other and perpendicular to the u/v plane.

Lambert equal-area

```
R = 2 * SIN(A) / SQRT(2*(1+COS(A))
```

As A approaches 180 degrees, R approaches 2. The globe is projected into a circle of radius 2.

Gnomonic

```
R = TAN(A)
```

Points for which A>90 degrees are invisible. Thus, a hemisphere is projected to the entire u/v plane. In practice, distortion becomes great when A is approximately 65 degrees or more. The center of this projection is the center of the earth.

Azimuthal equidistant

```
R = A * ¶ / 180.0
```

As A approaches 180 degrees, R approaches ¶. The globe is projected into a circle of radius ¶.

Basic satellite view

This formula applies only when S1=0, the basic view.

```
R = SQRT(SA*SA-1)*SIN(A)/(SA-COS(A))
```

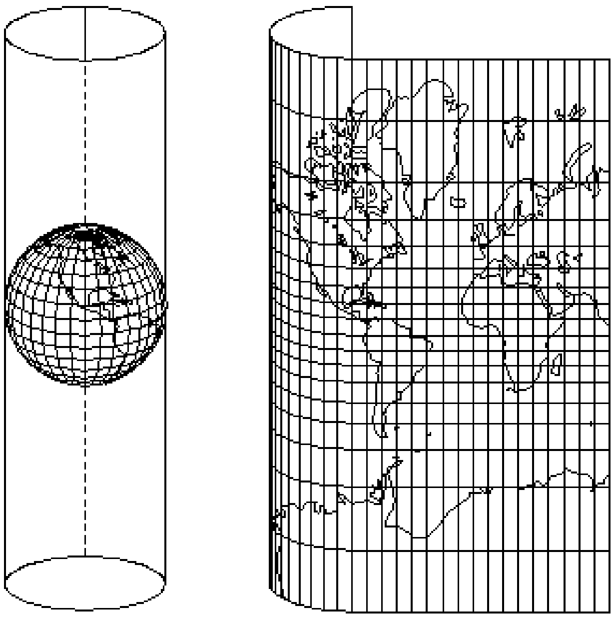
where SA is the distance, in earth radii, from the center of the earth to a satellite above the point (PLAT, PLON). Points for which COS(A) < 1 / SA are invisible. The portion of the earth's surface that would be visible from the satellite is projected inside a circle of radius 1. The center of the projection is at the satellite's position. As the satellite moves further out, the satellite view projection approaches the orthographic projection. Two parameters, S1 and S2, may be used to modify the basic satellite view projection; they can be used to show the earth as it would look to a simple pinhole camera pointed in a specified direction.

Mp 1.6 Description of projections: Cylindrical

This module provides background information; it is not required for using or understanding the Ezmap utility.

The third kind of projection Ezmap uses to project a map of the earth onto the plotter frame is the cylindrical projection. Cylindrical projections map the earth onto a cylinder that is tangent to the earth along a great circle passing through the user-specified point (PLAT, PLON) and tilted at the user-specified angle ROTA.

Illustration of a cylindrical projection



Discussion

The process of creating a cylindrical projection can be described geometrically in ten steps:

1. Imagine that the earth is placed behind the u/v (projection) plane so that the point at latitude 0 and longitude 0 just touches the plane at latitude 0 and longitude 0. The north pole is at the top, and the south pole is at the bottom.
2. Rotate the earth about its polar axis until the v axis is tangent to the meridian identified by PLON.
3. Rotate the earth by tilting one of the poles directly toward you and the other pole directly away from you until the point (PLAT, PLON) is at the origin of the u/v plane.
4. Rotate the earth clockwise through the angle ROTA about a line perpendicular to the u/v plane passing through the point at latitude 0 and longitude 0.
5. Wrap the u/v plane around the globe to form a cylinder with the u axis touching the earth along a great circle.
6. Using the technique specific to the projection type, project geographical outlines, parallels, and meridians outward from the earth's surface onto the cylinder.
7. Cut the cylinder along a line parallel to its axis and opposite the origin.
8. Unwrap the cylinder.
9. Set up linear scales along the u and v axes.
10. Draw a rectangular or elliptical portion of the resulting map. This portion of the u/v plane is the user coordinate system.

The technique used in step 6 above is described in the following three paragraphs for each of Ezmap's three types of cylindrical projections in the simple case where PLAT=PLON=ROTA=0. Let RLAT and RLON be the latitude and longitude, in degrees, of a point to be projected. 180<=RLON<=180. (If PLAT, PLON, and/or ROTA are nonzero, you must substitute for RLAT and RLON a pseudo-latitude and a pseudo-longitude computed from the real latitude and longitude.) The cylindrical projections may then be described as follows:

Cylindrical equidistant

```
U = RLAT
V = RLON
```

where the entire globe is projected into a rectangle in the u/v plane. 180<=U<=180, and 90<=V<=90.

Mercator

```
U = RLON * ¶/180.0
V = ALOG (COT (45 - RLAT/2))
```

The entire globe is projected onto an infinite rectangle in the u/v plane. -¶<=U<=¶. There are no limits on V. In practice, distortion becomes high for latitudes within 5 degrees of either pole.

Mollweide type

(This is not a true Mollweide projection.)

```
U = RLON/90
V = COS (90-RLAT)
```

The entire surface of the globe is projected into an ellipse. -2<=U<=2, -1<=V<=1.

References

Hershey, A.V, "The plotting of Maps on a CRT Printer," NWL Report 1844, 1963.

Lee, Tso-Hwa, "Students' Summary Reports, Work-Study Program in Scientific Computing," NCAR, 1968.

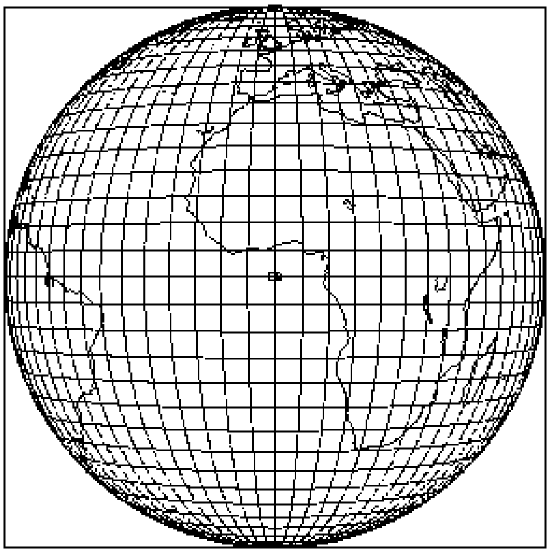
Parker, R.L., "2UCSD SUPERMAP: World Plotting Package."

Steers, J.A., *An Introduction to the Study of Map Projections*, University of London Press, 1962.

Mp 1.7 Producing a quick and dirty plot

SUPMAP creates a map of a desired portion of the globe by using your specifications for the projection, outlines, and latitude/longitude line intervals. An appropriate call to the routine SET is performed, and the routine MAPTRN is initialized so that you may map points of known latitude and longitude to points on the u/v plane and use the u/v coordinates to draw objects on the map.

A quick and dirty map



Code segment from cmpsup.f

```
      REAL PLIM1(2), PLIM2(2), PLIM3(2), PLIM4(2)
      DATA PLIM1 /0.0, 0.0/
      DATA PLIM2 /0.0, 0.0/
      DATA PLIM3 /0.0, 0.0/
      DATA PLIM4 /0.0, 0.0/
```

```
6       CALL SUPMAP (7, 0., 0., 0., PLIM1, PLIM2, PLIM3, PLIM4, 1, 5, 0, 0,IERR)
```

Synopsis

```
      CALL SUPMAP (JPRJ, PLAT, PLON, ROTA, PLIM1, PLIM2, PLIM3, PLIM4,
     +             JLTS, JGRD, IOUT, IDOT, IERR)
```

Arguments

JPRJ

Integer, Input---Defines the projection type with the following values (values less than 1 are treated as 1, and values greater than 10 are treated as 10):

1 Stereographic
2 Orthographic
3 Lambert conformal conic
4 Lambert equal-area
5 Gnomonic
6 Azimuthal equidistant
7 Satellite view
8 Cylindrical equidistant
9 Mercator
10 Mollweide-type

Using the value 2 causes the Ezmap parameter SA to be zeroed. (If SA is greater than 1., a satellite-view projection is used rather than an orthographic projection; it specifies the distance of the satellite from the center of the earth in units of earth radii.) Using the value 7 causes SA to be examined. If it has a nonzero value, the value is left alone. If it has a zero value, its value is reset to 6.631, which is about right for a satellite in a geosynchronous equatorial orbit.
The sign of JPRJ, when IOUT is -1, 0, or +1, indicates whether the continental outlines are to be plotted or not. See the description of IOUT below.

PLAT, PLON, ROTA

Real, Input---Define the origin of the projection and its rotation angle. These arguments are used in the same way as they would be in a call to the routine MAPROJ. See module "Mp 2.2 Choosing your map projection" for a complete description.

JLTS

Integer, Input---Specifies how to choose the rectangular limits of the map to be drawn. JLTS also determines the meaning of PLIM1, PLIM2, PLIM3, and PLIM4, which take real numbers as values. JLTS takes one of the following values:

1 The maximum useful area produced by the projection is plotted. PLIM1, PLIM2, PLIM3, and PLIM4 are not used.
2 The points (PLIM1, PLIM2) and (PLIM3, PLIM4) represent opposite corners of the map. PLIM1 and PLIM3 are latitudes, in degrees. PLIM2 and PLIM4 are longitudes, in degrees. If a cylindrical projection is being used, the first point is assumed to be on the left edge of the map and the second point is on the right edge. The order makes no difference for other map

projections. You cannot use JLTS=2 if any two adjacent corners of the desired map are outside the limb of projection.
3 PLIM1, PLIM2, PLIM3, and PLIM4 specify the minimum value of U, the maximum value of U, the minimum value of V, and the maximum value of V, respectively. Knowledge of the projection equations is necessary for using this option correctly.
Note: PLIM1, PLIM2, PLIM3, and PLIM4 are two-element real arrays. If 1<=JLTS<=4, then only the first element of each array is used.
4 PLIM1, PLIM2, PLIM3, and PLIM4 are positive angles, in degrees, representing angular distances from a point on the map to the left, right, bottom, and top edges of the map, respectively. For most projections, these angles are measured with the center of the earth at the vertex and represent angular distances from the point that projects to the origin of the u/v plane. For a satellite view projection, these angles are measured with the satellite at the vertex and represent angular deviations from the line of sight. Angular limits are particularly useful for polar projections and for the satellite view projection; they are not appropriate for the Lambert conformal conic projection and an error results if you attempt to use JLTS=4 with JPRJ=3.
5 PLIM1, PLIM2, PLIM3, and PLIM4 are two-element arrays giving the latitudes and longitudes, in degrees, of four points that are to be on the edges of the rectangular map. If a cylindrical projection is being used, the first point is assumed to be on the left edge and the second point is on the right edge. The order makes no difference for other map projections. You cannot use JLTS=5 if an entire side of the desired map is outside the limb of projection.
Note: PLIM1, PLIM2, PLIM3, and PLIM4 are two-element real arrays. If 1<=JLTS<=4, then only the first element of each array is used.

MOD (IABS(JGRD), 1000)

Integer, Input---The value, in degrees, of the interval at which lines of latitude and longitude are to be plotted. If the given interval is zero, grid lines and labels are not plotted. If JGRD is less than zero, the perimeter is not plotted. Set JGRD to -1000 to suppress both grid lines and perimeter; set JGRD to +1000 to suppress the grid lines, but leave the perimeter. The value -0 may have a meaning on some computers, but should be avoided; use -1000 instead.

IOUT

Integer, Input---The outline flag determines which geopolitical boundaries are drawn:
0 Suppress US state outlines
-1 Plot US state outlines, but not continental outlines
+1 Plot US state outlines and continental outlines
<=-2 Do not plot any outlines
+2 Plot continental outlines, no US outlines
3 Plot US state outlines, no others
4 Plot continental outlines, international outlines, and US state outlines
>4 Plot continental outlines and international outlines, but no US outlines
In past versions of this software, the sign of IOUT specified whether or not a line of text was to be written on the print output. This usage is no longer valid.

IDOT

Integer, Input---The dotted outline flag takes the following values:

0 Continuous outlines
1 Dotted outlines

IERR

Integer, Output---A nonzero value on return indicates that an error has occurred.

Exercises

1. Given that Boulder, Colorado has a latitude of 40.00 and a longitude of 105.15, draw a satellite projection centered above Boulder.

Mp 1.8 What calls do I need to get my Ezmap plot?

There are essentially two ways to use Ezmap. First, you can use Ezmap to draw a simple black-and-white map, often for use as a background, or you can use Ezmap with the Areas utility to do color-fill or pattern-fill areas, masking, or to act as a background for another color utility like Conpack.

Ezmap functional outline

```
--------------------------------------------------------------
 *  **  1.  Open GKS
        2.  Define position on plotter frame
 *  **  3.  Define map projection
 *  **  4.  Choose map limits
 *  **  5.  Choose desired political boundaries
 *      6.  Initialize Areas
 *  **  7.  Initialize Ezmap
 *      8.  Add geographic boundaries (or outlines) to area map
        9.  Draw geographic labels
        10.  Draw points and lines on your map
        11.  Draw desired latitude and longitude lines
 *      12.  Fill desired areas
    **  13.  Draw geographic outlines
 *  **  14.  Call FRAME
 *  **  15.  Close GKS
--------------------------------------------------------------
```

* Steps needed for maps that have masking or filling.

** Steps needed for a simple black-and-white map.

Mp 1.9 Ezmap parameters: What they do and how to use them

Ezmap relies heavily on parameters to produce the map options you specify (such as which projection, the map limits, the perimeter, and so on). The first argument in a call to MPGET*x* or MPSET*x* is the parameter name. The second argument is either a variable in which the value of the named parameter is to be returned (via MPGET*x*) or an expression specifying the desired new value of the parameter (via MPSET*x*).

Synopsis

```
CALL MPGETC (PNAM, CVAL)
CALL MPGETI (PNAM, IVAL)
CALL MPGETR (PNAM, RVAL)
CALL MPSETC (PNAM, CVAL)
CALL MPSETI (PNAM, IVAL)
CALL MPSETR (PNAM, RVAL)
CALL MAPSAV (INFO)
CALL MAPRST (INFO)
```

Routines

MPGETC

Retrieves character parameter information.

MPGETI

Retrieves integer parameter information.

MPGETR

Retrieves real parameter information.

MPSETC

Sets character parameter information.

MPSETI

Sets integer parameter information.

MPSETR

Sets real parameter information.

MAPSAV

Saves all user-settable Ezmap parameter information to a file.

MAPRST

Retrieves all user-settable Ezmap parameter information from a file in which MAPSAV has saved the information.

Arguments

PNAM

Character expression, Input or Output---The name of the parameter that you want to set or retrieve. Two-character parameter names appear in examples and discussions in place of PNAM. The parameter names and their meanings are enclosed in single quotation marks in the Fortran code to ensure that they are treated as character expressions rather than real or integer variables. Only the first two characters within the quotation marks are examined.

CVAL

Character expression, Input or Output---A character value to be set or retrieved.

IVAL

Integer, Input or Output---An integer value to be set or retrieved.

RVAL

Real, Input or Output---A real value to be set or retrieved.

INFO

Integer, Input or Output---A unit number for a single unformatted record.

Discussion

Module "Mp 1.2 Table of Ezmap parameters" is a quick reference guide to all Ezmap parameters; it also gives references to modules in which parameter-setting examples appear. Section "Mp 7. Ezmap parameter descriptions" provides more thorough descriptions of the Ezmap parameters.

Mp 2. Getting set up

Before you can draw a map, you need to set Ezmap parameters and call several Ezmap setup routines; these steps define how much of which projection you want to view. Because the Ezmap initialization routine MAPINT uses these parameters to do some of the initialization process, it is best to completely define your map before calling MAPINT.

Ezmap functional outline

```
------------------------------------------------------------
      1.  Open GKS
   *  2.  Define position on plotter frame
   *  3.  Define map projection
   *  4.  Choose map limits
   *  5.  Choose desired political boundaries
      6.  Initialize Areas
   *  7.  Initialize Ezmap
      8.  Add geographic boundaries (or outlines) to area map
      9.  Draw geographic labels
      10.  Draw points and lines on your map
      11.  Draw desired latitude and longitude lines
      12.  Fill desired areas
```

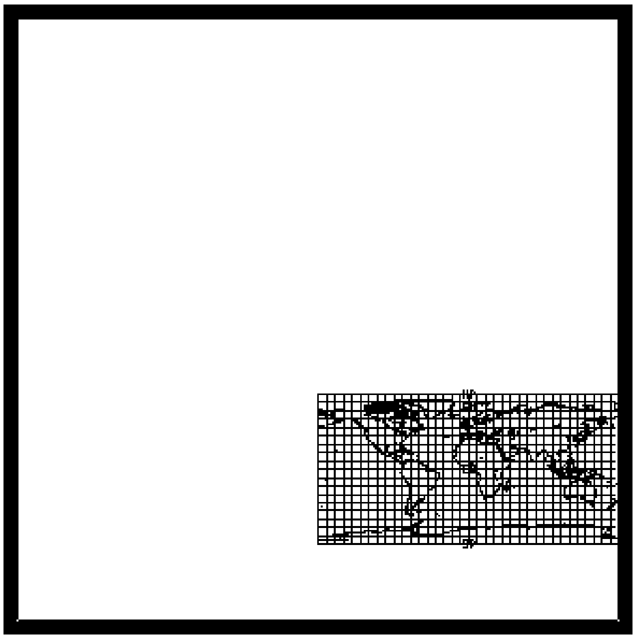
```
      13.  Draw geographic outlines
      14.  Call FRAME
      15.  Close GKS
------------------------------------------------------------
```

* Steps discussed in this section.

Mp 2.1 Positioning the plot in the frame

In most NCAR Graphics utilities, you can use the SPPS routine SET to position your plot in the plotter frame or viewport. In Ezmap, however, the special routine MAPPOS helps you position your plot.

Positioning your plot on the plotter frame



Code segment from cmppos.f

```
1       CALL MAPPOS (0.5, 1.0, 0.0, 0.5)
2       CALL MAPDRW
```

Synopsis

```
      CALL MAPPOS (XVPL, XVPR, YVPB, YVPT)
```

Arguments

XVPL

> *Real, Input*---The position of the left edge of the area in which the viewport is to be positioned, in Normalized Device Coordinates (NDCs). 0.0<=XVPL<=1.0. The default is .05.

XVPR

Real, Input---The position of the right edge of the area in which the viewport is to be positioned, in NDCs. 0.0<=XVPR<=1.0. The default is .95.

YVPB

Real, Input---The position of the bottom edge of the area in which the viewport is to be positioned, in NDCs. 0.0<=YVPB<=1.0. The default is .05.

YVPT

Real, Input---The position of the top edge of the area in which the viewport is to be positioned, in NDCs. 0.0<=YVPT<=1.0. The default is .95.

Discussion

By using the MAPPOS routine, you can squeeze more than one Ezmap plot onto the same frame. MAPPOS centers the viewport and makes it as large as possible within the specified area, and if possible, maintains the proper aspect ratio.

Line 1 of the **cmppos.f** code segment sets the left, right, bottom, and top of the viewport so that it is located in the lower left corner of the plotter frame.

Exercises

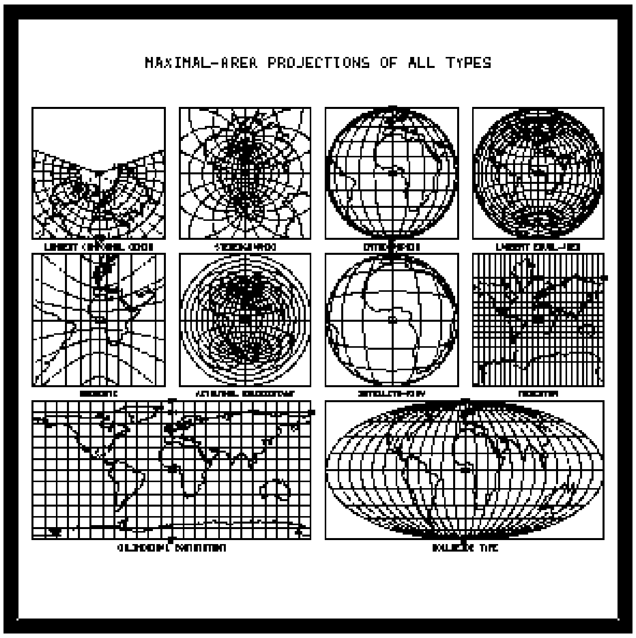
1. Using the **cmppos** example, move the plot to the center of the plotter frame, but keep it the same size.

Mp 2.2 Choosing your map projection

There are ten different map projections available with Ezmap.

To specify the projection you want, call the MAPROJ routine.

Examples of different map projections



Code segment from mpex05.f

```
      CALL MAPPOS (.5125, .73125, .63125, .85)
      CALL MAPROJ ('OR', 0., 0., 0.)
      CALL MAPDRW
```

Synopsis

```
CALL MAPROJ (JPRJ, PLAT, PLON, ROTA)
```

Arguments

JPRJ

Character expression, Input---Specifies the desired projection type, as follows:

LC
Lambert conformal conic with two standard parallels

ST
Stereographic

OR
Orthographic. The Ezmap parameter SA will be zeroed. See the next module.

LE
Lambert equal-area

GN
Gnomonic

AE
Azimuthal equidistant

SV
Satellite view. If Ezmap parameter SA<=1., it will be reset to 6.631 (the value for a satellite in a geosynchronous orbit).

CE
Cylindrical equidistant
ME
Mercator
MO
Mollweide-type

PLAT, PLON, and **ROTA**

Real, Input---Angular quantities, in degrees; their usage depends on the value of JPRJ:

JPRJ<>LC
PLAT and PLON define the latitude and longitude of the point on the globe that projects to the origin of the u/v plane. This is the center of the projection, not the center of the map. 90.<=PLAT<=+90., with positive values in the northern hemisphere and negative values in the southern. 180.<=PLON<=+180., with positive values to the east of Greenwich, England, and negative values to the west. ROTA is the angle between the v axis and north at the origin. It is taken to be positive if the angular movement from north to the v axis is counterclockwise, otherwise ROTA is negative. If the origin is at the north pole, "north" is taken to be in the direction of PLON+180. If the origin is at the south pole, "north" is taken to be in the direction of PLON. For cylindrical projections, the axis of the projection is parallel to the v axis.

JPRJ=LC
PLON defines the central meridian of the projection, and PLAT and ROTA define the two standard parallels. If PLAT=ROTA, the simpler conic projection, with one standard parallel, is used. For more information, refer to the drawing in module "Mp 1.4 Description of projections: Conic."

Discussion

The MAPROJ routine requires you to choose the projection you want to use as well as the center of the projection and the angle of rotation. To minimize distortion, it is usually best to center your projection over the area of interest, and limit the map projection to the minimally necessary area. If you are using Mercator or cylindrical equidistant projections, use PLAT=0 to avoid curved grid lines.

Line 1 of the **mpex05.f** code segment sets the viewport so that the map is drawn in the upper center of the screen. Line 2 chooses an orthographic projection and centers it on latitude 0 and longitude 0, which is just off the west coast of North Africa. Line 3 draws the map projection.

Exercises

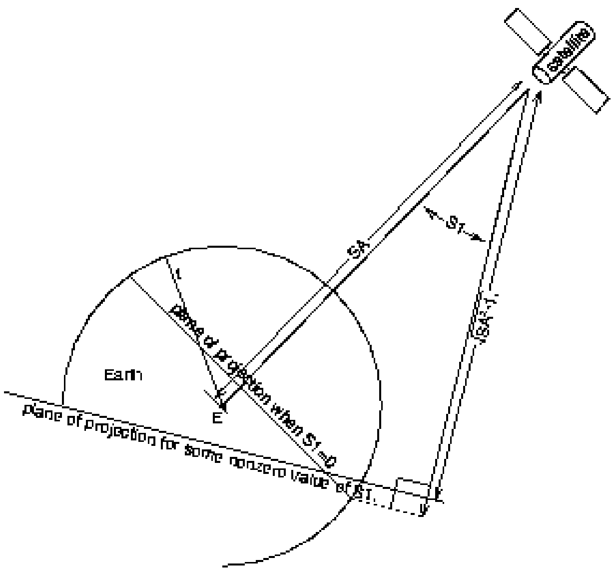
1. Using the **cezmap1** example, draw a cylindrical equidistant map with straight grid lines.
2. Copy **cezmap1.f** to your own file, and name it **cezmap.f**. You will use this file in subsequent exercises to build a customized simple map-drawing routine.

Mp 2.3 Choosing your map projection: Satellite view

Two of the projections, the orthographic and satellite view projections, have the same internal identifier.

To get a satellite projection, it is necessary to set one or more of the three satellite view parameters, SA, S1, and S2.

The satellite projection



Code segment from cmpsat.f

```
      CALL MAPROJ ('SV', 40., 10., 0.)
      CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 2.)
      CALL MPSETR ('S1 - SATELLITE ANGLE 1', 10.)
      CALL MPSETR ('S2 - SATELLITE ANGLE 2', 15.)
      CALL MAPDRW
```

Synopsis

```
CALL MPSETR ('SA', sa)
CALL MPSETR ('S1', s1)
CALL MPSETR ('S2', s2)
```

Arguments

SA

> *Real*---Produces a SAtellite view if SA>1.0, otherwise, you get an orthographic projection. SA is the distance of the satellite from the center of the earth, measured in earth radii. If S1=0 as SA approaches infinity, the satellite view projection approaches the orthographic projection. SA=0.0 by default.

S1

> *Real*---Satellite angle 1 measures the angle, in degrees, between the line from the satellite to the center of the earth and the line of sight of the satellite. Used only when SA is greater than 1.

S1=0.0 by default.

S2

Real---Satellite angle 2. Let the case where S1=S2=0 be called the basic satellite view. If S1<>0.0, then S2 specifies the angle, in degrees, from the positive u axis of the basic satellite view counterclockwise to the line OP, where O is the origin of the basic view, and P is the projection on the basic view, of the desired line of sight from the satellite. When S1 and S2 are not equal to zero, the portion of the earth projected remains the same, but the edge of the part of the u/v plane covered by the projection is an ellipse, a parabola, or a hyperbola with an axis at an angle S2 to the u axis. Used only when SA>1. S2=0.0 by default.

Discussion

The illustration shows a two-dimensional view of a plane passing through the center of the earth (marked E) and a satellite. The surface of the earth appears as a circle. The radius of the earth is defined to be *r*, or in this case, one unit. The distance of the satellite from the center of the earth is SA. Two planes of projection for the satellite view are shown: the basic satellite view (S1=0.) and another satellite view (S1<>0.). Note that something less than a full hemisphere is projected. Also, as S1 increases, the visible portion of the earth that is projected becomes elliptical, then parabolic, then hyperbolic.

If the basic satellite view is desired, it is not necessary for the user to set either S1 or S2. However, to get a satellite view instead of an orthographic projection, it is necessary to set SA>1.0.

Line 1 of the **cmpsat.f** code segment sets up the projection to give us a reasonably undistorted view of the Mediterranean. Line 2 sets the satellite at two earth radii distant (roughly 4000 miles out). Lines 3 and 4 set the viewing angle to be slightly off of straight down, and line 5 draws our map.

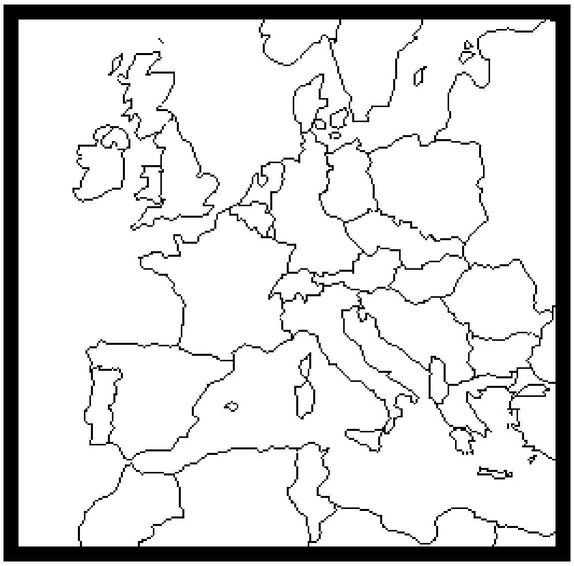
Exercises

1. Beijing, China is at roughly latitude 39 and longitude 116. Draw a satellite view from right over the city, assuming that the satellite is 2.5, 5, and 20 earth radii away. Also for each of these distances, set S1=30.0 and 60.0, and S2=40.0.

Mp 2.4 Choosing map outlines to be drawn

Ezmap offers you several different options for outlining your map. You can use the OU parameter to select continental outlines, international outlines, and state outlines for the US.

Political outlines over Europe



Code segment from cmpou.f

```
      DATA PLIM1 /30., 0./
      DATA PLIM2 /-15., 0./
      DATA PLIM3 /60., 0./
      DATA PLIM4 /30., 0./
      CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
      CALL MAPROJ ('ME', 0., 0., 0.)
      CALL MAPSET ('CO', PLIM1, PLIM2, PLIM3, PLIM4)
      CALL MAPINT
      CALL MAPLOT
```

Synopsis

```
CALL MPSETC ('OU', string)
```

Arguments

OU

Character expression---the OUtline parameter tells which set of outline data to use:

NO
No outlines
CO
Continental outlines (the default)
US
US state outlines
PO
Continental and political (by country) outlines
PS
US state outlines, country outlines, and continental outlines

Discussion

Line 5 of the **cmpou.f** code segment sets the outline dataset for political, or country outlines. Line 6 sets up the map projection. Lines 1 through 4 and 7 define and choose the corners of the map. Lines 8 and 9 initialize Ezmap and draw the map.

Although state outlines for the US are available, major geographic features like rivers and mountains, and political features like the provinces in other countries are not available with NCAR Graphics at this time. We hope to incorporate this information into one of the future releases of the software.

Exercises

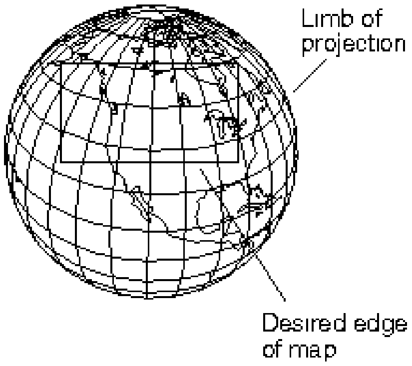
1. Using the **cmpou** example, try drawing no political boundaries.
2. Using your version of **cezmap.f** (see Mp 2.2 exercises), add a parameter to the **cezmap** subroutine calling sequence so that you can choose your outline dataset outside **cezmap**. Compare your results to **cezmap2.f**.

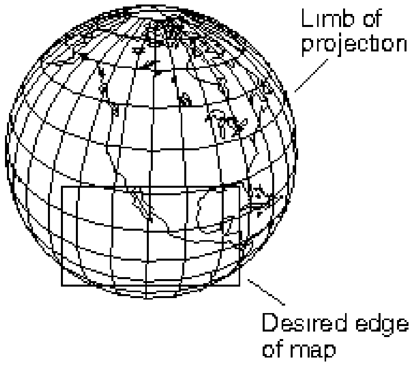
Mp 2.5 Setting limits for the projection

The MAPSET routine provides several methods you can use to set limits for the map to be drawn.

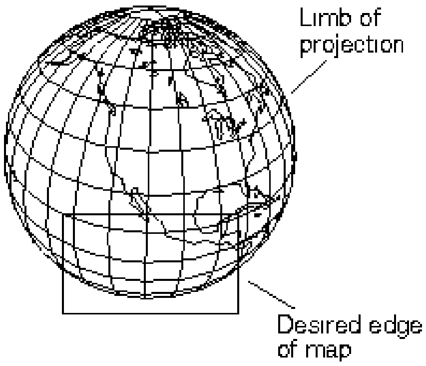
Setting limits on your map projection

`Figure 1  Figure 2`





`Figure 3`



Code segment from cmpou.f

```
      DATA PLIM1 /30., 0./
      DATA PLIM2 /-15., 0./
      DATA PLIM3 /60., 0./
      DATA PLIM4 /30., 0./
      CALL MAPROJ ('ME', 0., 0., 0.)
      CALL MAPSET ('CO', PLIM1, PLIM2, PLIM3, PLIM4)
      CALL MAPINT
      CALL MAPLOT
```

Synopsis

```
      CALL MAPSET (JLIM, PLIM1, PLIM2, PLIM3, PLIM4)
```

Arguments

JLIM

Character expression, Input---Specifies how to choose the rectangular limits of the map to be drawn. JLIM also determines the meaning of PLIM1, PLIM2, PLIM3, and PLIM4, which are two-element real arrays. JLIM takes one of the following values:

MA

The maximum useful area produced by the projection is plotted. PLIM1, PLIM2, PLIM3, and PLIM4 are not used.

CO

The points (PLIM1(1), PLIM2(1)) and (PLIM3(1), PLIM4(1)) represent opposite corners of the map. PLIM1 and PLIM3 are latitudes, in degrees. PLIM2 and PLIM4 are longitudes, in degrees. If a cylindrical projection is being used, the first point is assumed to be on the left edge of the map and the second point is on the right edge. The order makes no difference for other map projections. CO cannot be used if any two adjacent corners of the desired map are outside the limb of projection, hence CO can only be used in cases similar to Figure 1 in the illustration.

LI

PLIM1(1), PLIM2(1), PLIM3(1), and PLIM4(1) specify the minimum value of u, the maximum value of u, the minimum value of v, and the maximum value of v, respectively. Knowledge of the projection equations (see modules Mp 1.4 through Mp 1.6) is necessary for using this option correctly. LI is the only option that can be used in Figure 3 of the illustration "Setting limits on your map projection."

AN

PLIM1(1), PLIM2(1), PLIM3(1), and PLIM4(1) are positive angles, in degrees, representing angular distances from a point on the map to the left, right, bottom, and top edges of the map, respectively. For most projections, these angles are measured with the center of the earth at the vertex and represent angular distances from the point that projects to the origin of the u/v plane. For a satellite view projection, these angles are measured with the satellite at the vertex and represent angular deviations from the line of sight. Angular limits are particularly useful for polar projections and for the satellite view projection; they are not appropriate for the Lambert conformal conic projection and an error results if you attempt to use JLIM=AN with JPRJ=LC.

PO

PLIM1, PLIM2, PLIM3, and PLIM4 are two-element real arrays giving the latitudes and longitudes, in degrees, of four points that are to be on the edges of the rectangular map. If a cylindrical projection is being used, the first point is assumed to be on the left edge and the second point is on the right edge. The order makes no difference for other map projections. PO cannot be used if an entire side of the desired map is outside the limb of projection, as in Figure 3 in the illustration; it can be used in cases like Figure 2.

Discussion

Lines 1 through 4 of the **cmpou.f** code segment set the limits of the projection. Line 6 uses these limits as corners of the projection, so only the first elements of the arrays PLIM1, PLIM2, PLIM3, and PLIM4 are used. Line 5 sets the map projection; lines 7 and 8 initialize and draw the map.

In Figure 1 of the **cmpou** example, JLIM can be set to CO, PO, or LI. However, CO won't work in Figure 2 because one of the necessary corners of specification is outside the projected map area. In Figure 3, only LI works as an option because both a necessary corner and the entire side of the desired map is outside the limb of projection.

Note: Most compilers allow us to treat PLIM1 through PLIM4 as if they were reals, so we can pass in reals in the code when JLIM=MA, CO, LI, or AN.

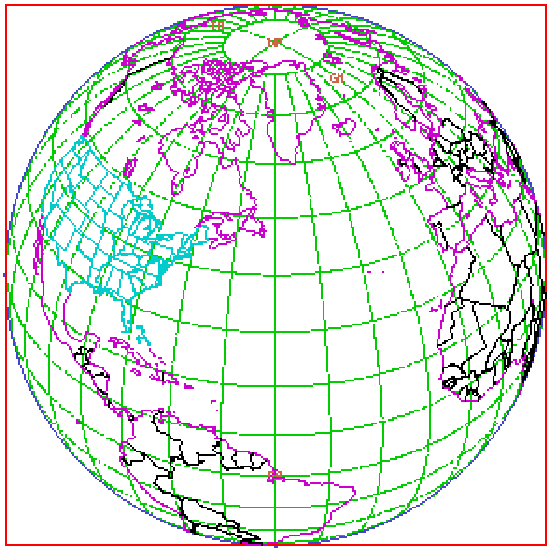
Exercises

1. Using the **cmpou** example, set JLIM to draw the complete globe.
2. The continental US is approximately bounded by latitudes of 22 and 47 and longitudes of -120 and -65. Draw a map of the continental US.
3. Using your version of **cezmap.f** (from the Mp 2.2 exercises), modify it so that you can pass in the limits of your map projection. Test these limits by producing a maximal area satellite view projection. Compare your results with the **cezmap3** example.

Mp 2.6 Setting map line colors

There are seven parameters controlling the line color of various portions of the map. These parameters are most frequently used in simple line maps; they cannot be used to color-fill regions in the map.

Coloring your map



Code segment from cmpclr.f

```
      CALL COLOR
      CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PS')
      CALL MPSETI ('C1 - COLOR INDEX 1 - PERIMETER', 1)
      CALL MPSETI ('C2 - COLOR INDEX 2 - GRID', 2)
      CALL MPSETI ('C3 - COLOR INDEX 3 - LABELS', 3)
      CALL MPSETI ('C4 - COLOR INDEX 4 - LIMB LINE', 4)
      CALL MPSETI ('C5 - COLOR INDEX 5 - CONTINENTAL OUTLINES', 5)
      CALL MPSETI ('C6 - COLOR INDEX 6 - US STATE OUTLINES', 6)
      CALL MPSETI ('C7 - COLOR INDEX 7 - COUNTRY OUTLINES', 7)
      CALL MAPROJ ('SV', 40., -50., 0.)
      CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
      CALL MAPSET ('MA', PLIM1, PLIM2, PLIM3, PLIM4)
      CALL MAPINT
```

Synopsis

```
CALL MPSETI ('C1', ic1)
CALL MPSETI ('C2', ic2)
CALL MPSETI ('C3', ic3)
CALL MPSETI ('C4', ic4)
CALL MPSETI ('C5', ic5)
CALL MPSETI ('C6', ic6)
CALL MPSETI ('C7', ic7)
```

Arguments

Each color index parameter specifies the color index of some part of the map, as indicated below. The default values for all color index parameters are negative; this specifies that no color index change is to be made between parts of the map.

Parameter	**Type**	**Area of effect**

```
C1          Integer  Perimeter
C2          Integer  Grid (lat/
                     lon lines)
C3          Integer  Labels
C4          Integer  Limb line
C5          Integer  Continental
                     outlines
C6          Integer  US state
                     outlines
C7          Integer  Country
                     outlines
--------------------------------
```

Discussion

Line 1 of the **cmpclr.f** code segment sets up the color table by calling the user-supplied routine COLOR, which calls the GKS routine GSCR. Line 2 chooses the "draw everything" option for this map. Lines 3 through 9 set each color index parameter to a color index chosen from the color table. This shows each of the different colors on the map: the perimeter is red, the grid is green, the labels are orange, the limb line is blue, the continents are outlined in magenta, the US states are outlined in aqua, and other countries are outlined in black. Lines 10 and 11 choose a satellite projection from 5 earth radii, and set a maximal viewport to show as much as possible.

If you wanted to make the entire map the same color, you could set the polyline and text colors using the GKS calls GSPLCI and GSTXCI, and not set Ezmap parameters C1 through C7.

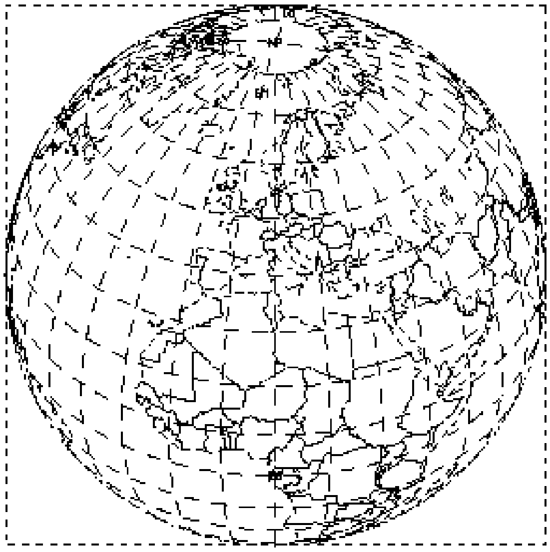
Exercises

1. Using the **cmpclr** example, change the grid lines to aqua and the continent lines to yellow.
2. Using your version of **cezmap.f** (from the Mp 2.2 exercises), change line colors to values you select.

Mp 2.7 Controlling Ezmap lines

Ezmap supplies a default version of MAPUSR that does nothing. By modifying this default version of MAPUSR, you gain complete control over how each line in Ezmap is drawn. You can use it to change colors of lines, dash patterns of lines, or even to eliminate the drawing of certain lines.

Setting each Ezmap line pattern



Code segment from cmpusr.f

```
      CALL MAPROJ ('SV', 40., 10., 0.)
      CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
      CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
      CALL MAPDRW

      SUBROUTINE MAPUSR (IPRT)
C 1110000011100000
      IF (IPRT .EQ. 1) CALL DASHDB (57568)
C 1111111100000000
      IF (IPRT .EQ. 2) CALL DASHDB (65280)
C 0100110001110000
      IF (IPRT .EQ. 4) CALL DASHDB (19568)
C 1111000011110000
      IF (IPRT .EQ. 5) CALL DASHDB (61680)
C 1110010011100100
      IF (IPRT .EQ. 6) CALL DASHDB (58596)
C 0101010101010101
      IF (IPRT .EQ. 7) CALL DASHDB (21845)
      RETURN
      END
```

Synopsis

```
SUBROUTINE MAPUSR (IPRT)
```

Arguments

MAPUSR

Subroutine---This Ezmap subroutine is called just before and just after each portion of the map is drawn. The default version of MAPUSR does nothing, but you can modify it to control how Ezmap draws lines.

IPRT

Integer, Input---If positive, this variable specifies which part of the map is about to be drawn with these values:

1 Perimeter
2 Grid
3 Labels
4 Limb line
5 Continental outlines
6 US state outlines
7 International outlines

A negative value for IPRT indicates that part of the map has been drawn.

Discussion

Lines 1 through 4 of the **cmpusr.f** code segment define and draw a satellite view map projection as usual. In this case, a custom version of MAPUSR must be written and either added to the end of your program or loaded before the NCAR Graphics libraries are loaded. The MAPUSR shown here checks to see which portion of the map is being drawn, and then sets a binary dash pattern using the DASHDB call. Each line of MAPUSR is preceded by a comment line that shows the binary dash pattern converted to base 10 in the call to DASHDB. In each binary dash pattern, 0 represents a space and 1 represents a dash.

Because characters are drawn using the GKS routine GTX, not the Dashline utility, do not attempt to set a dashed line for them in MAPUSR.

This version of MAPUSR uses the dash pattern option to draw the various parts of the map. However, if you set the dotted outline flag DO nonzero to specify dotted outlines, this version of MAPUSR would have no effect.

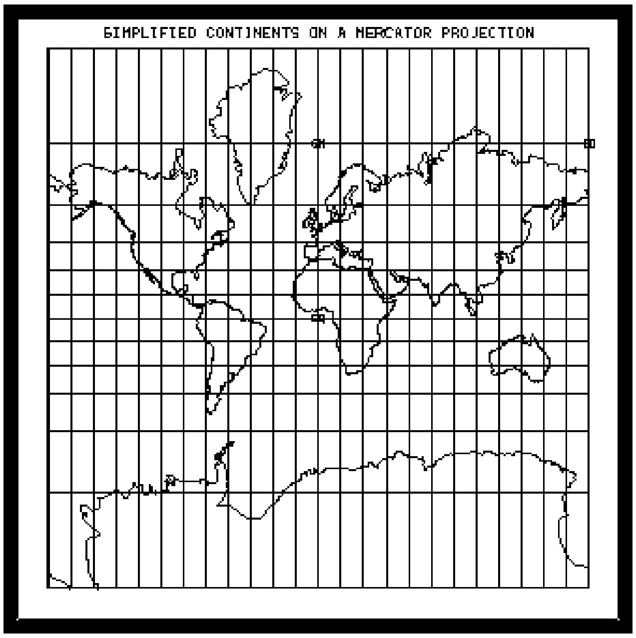
Exercises

1. Modify MAPUSR to change the line colors in the **cmpusr** example.

Mp 2.8 Controlling geographic and political outlines

You can write your own MAPEOD routine to control how Ezmap draws outlines. For example, you may want to use MAPEOD to reduce visual clutter by eliminating lake and island outlines.

Removing visual clutter



Code segment from mpex03.f

```
      SUBROUTINE MAPEOD (NOUT, NSEG, IDLS, IDRS, NPTS, PNTS)
      DIMENSION PNTS (*)
      IF (IDLS .NE. 2 .AND. IDRS .NE. 2) NPTS=0
      IF (IDLS .NE. 1 .AND. IDRS .NE. 1 .AND.
     + IDLS .NE. 3 .AND. IDRS .NE. 3 .AND.
     + IDLS .NE. 11 .AND. IDRS .NE. 11 .AND.
     + IDLS .NE. 79 .AND. IDRS .NE. 79 .AND.
     + IDLS .NE. 99 .AND. IDRS .NE. 99 .AND.
     + IDLS .NE. 104 .AND. IDRS .NE. 104 .AND.
     + IDLS .NE. 107 .AND. IDRS .NE. 107 .AND.
     + IDLS .NE. 163 .AND. IDRS .NE. 163)      NPTS=0
      RETURN
      END
```

Synopsis

```
CALL MAPEOD (NOUT, NSEG, IDLS, IDRS, NPTS, PNTS)
```

Arguments

NOUT

Integer, Input---The number of the outline dataset from which the segment is taken. The following table shows the numbers specifying the NOUT dataset and gives the outline type and outline name:

Number	**Name**	**Contents**
1	CO	Continental outlines only
2	US	US state outlines only

```
3       PS    Continental, US state,
              and international outlines
4       PO    Continental and interna
              tional outlines
----------------------------------------
```

NSEG

Integer, Input---The number of the segment within the outline dataset. The maps in the **mpex09** example show segment numbers for the outline dataset CO; you may modify this program to produce maps showing segment numbers for any outline dataset.

IDLS, IDRS

Integer, Input---Identifiers for the areas to the left and right, respectively, of the segment. (Left and right are defined from the standpoint of a viewer standing at point 1 of the segment and looking toward point 2.) For a complete list of area identifiers, see section "Mp 6. Table of Ezmap area identifiers."

NPTS

Integer, Input---The number of points defining the outline segment, when entering subroutine MAPEOD. NPTS may be zeroed by MAPEOD to suppress plotting of the segment on the map.

PNTS(NPTS)

Real array, Input---PNTS(1) and PNTS(2) are the latitude and longitude of the first point, PNTS(3) and PNTS(4) are the latitude and longitude of the second point, . . . and PNTS(2*NPTS1) and PNTS(2*NPTS) are the latitude and longitude of the last point. All values are in degrees. Longitudes are all between 180 and +180, and no segment crosses the meridian at 180 (+180) degrees.

Discussion

This version of MAPEOD uses area identifiers for the outline dataset CO to suppress all but the major global land masses and the British Isles. Line 3 of the **mpex03.f** code segment culls the segment if there is no ocean on either side of it. Line 4 checks to see if this is a continental outline or an outline of British Isles, and if not, then removes the outline.

MAPEOD is called by Ezmap to examine each segment in an outline dataset just before it is plotted. The default version does nothing. The **mpex03**, **mpex05**, and **mpex09** examples all contain versions of MAPEOD that you might find useful.

Mp 2.9 Rectangular and elliptical perimeters

The Ezmap perimeter is controlled by two parameters. PE determines if a perimeter is drawn, and EL determines if the perimeter is an ellipse or a rectangle.

Map without perimeter

Code segment from cmpel.f

```
      CALL MAPROJ ('SV', 40., 10., 0.)
      CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
      CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
      CALL MPSETI ('PE - PERIMETER FLAG', 0)
      CALL MPSETI ('EL - ELLIPTICAL PERIMETER FLAG', 1)
```

Synopsis

```
CALL MPSETI ('PE', ipe)
CALL MPSETI ('EL', iel)
```

Arguments

PE

Integer---Draws a PErimeter if nonzero. PE<>0 by default.

EL

Integer---If nonzero, draw only that part of the map inside an ELlipse inscribed within the normal rectangular perimeter. This is particularly appropriate for use with azimuthal projections and angular limits specifying a square, in which case the ellipse becomes a circle. This parameter works for any map. EL=0 by default.

Discussion

In Ezmap, a "perimeter" is the boundary of a map produced by a call to Ezmap routines. The perimeter may either be a rectangle in the projection plane or an ellipse inscribed in that rectangle.

An "ellipse" is a particular kind of closed curve as defined by any plane geometry text.

A "limb line" is that line in the Ezmap projection plane separating points into which some point on the globe projects from points into which no point on the globe projects. For example, when you are using an orthographic projection, the visible side of the globe maps into the interior of a circle of radius 1 and centered at the origin; the "limb" of an orthographic projection is therefore that circle. Depending on the projection being used, the "limb lines" may be straight lines, circles, ellipses, parabolas, or hyperbolas. Limb lines can also be complicated curves (sometimes defined by means of a function and sometimes defined by means of a table of X/Y coordinates defining a polygon) for projections not offered by Ezmap.

Lines 1 through 3 of the **cmpel.f** code segment set up the satellite map projection. Lines 4 and 5 turn off perimeter drawing. Examine the plot to see the places where Ezmap has not drawn a dividing line between a geographical outline and the background; this occurs because PE has been turned off.

Exercises

1. Using the **cmpel** example, draw a perimeter around the map.
2. Using the **cmpitm** example, turn on the EL option before the call to **cmpmsk**, and notice the difference it makes in the plot.
3. Using your own version of **cezmap.f** (from the Mp 2.2 exercises), set the perimeter and ellipse options.

Mp 2.10 Saving and retrieving Ezmap parameters

Three Ezmap routines allow you to save and retrieve various aspects of Ezmap's state. MAPRS allows you to redo the SET call that was used in the last call to MAPINT. MAPSAV allows you to save your map transformation and parameter information to a file. MAPRST allows you to retrieve the information saved by MAPSAV so you can use it for another plot.

Synopsis

```
CALL MAPRS
CALL MAPSAV (IUNIT)
CALL MAPRST (IUNIT)
```

Arguments

MAPRS

> Occasionally, you may want to re-execute the call to SET that was done for you by your last call to MAPINT. MAPRS does this. You might use this when you plot user lines over a map generated in a different overlay (such as when using a Gflash buffer), and when the system plot package does not reside in an outer overlay.

MAPSAV(IUNIT)

> Saves current state of Ezmap to a file by writing, on the Fortran unit specified by IUNIT, the

current values of all the user-settable parameters. IUNIT is the number of a Fortran unit to which a single unformatted record is to be written. You must position this unit; MAPSAV does not rewind it, either before or after writing the record.

MAPRST(IUNIT)

Restores a previous state of Ezmap saved by an earlier call to MAPSAV by reading, from the Fortran unit specified by IUNIT, values of all user-settable parameters, and then executing MAPINT. IUNIT is the number of a Fortran unit from which a single unformatted record is to be read. You must position this unit; MAPRST does not rewind it, either before or after reading the record.

Mp 3. Simple maps

For our purposes, we define a "simple map" as a map that is essentially a line drawing. Simple maps do not have color-filled or pattern-filled regions.

You need fewer steps to produce simple black-and-white maps than you do when you use the Areas utility for maps that require masking and filling.

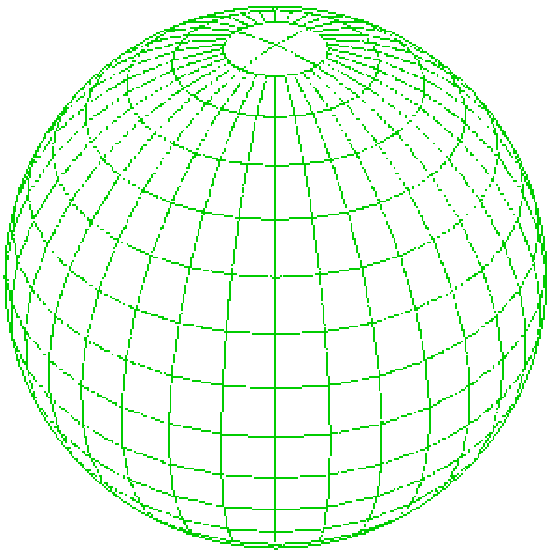
Simple map functional outline

```
-----------------------------------------------
   1.  Open GKS
   2.  Define position on plotter frame
   3.  Define map projection
   4.  Choose map limits
   5.  Choose desired political boundaries
   6.  Initialize Ezmap
   7.  Draw geographic labels
   8.  Draw points and lines on your map
   9.  Draw desired latitude and longitude lines
   10.  Draw geographic outlines
   11.  Call FRAME
   12.  Close GKS
-----------------------------------------------
```

Mp 3.1 Initializing Ezmap

Before using the Ezmap utility, you must initialize it with the MAPINT routine. Also, after calling MAPPOS, MAPROJ, or MAPSET, you must call MAPINT again to re-initialize Ezmap.

Initializing a map



Code segment from cmpgrd.f

```
1       CALL COLOR
2       CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
3       CALL MAPROJ ('SV', 40., -50., 0.)
4       CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
5       CALL MAPSET ('MA', PLIM1, PLIM2, PLIM3, PLIM4)
6       CALL MAPINT
```

Synopsis

```
CALL MAPINT
```

Discussion

Line 1 of the **cmpgrd.f** code segment sets up the color table. Line 2 turns on drawing of the continental and political outlines. Lines 3 through 5 set the projection for a satellite view over the Atlantic Ocean. Line 6 initializes Ezmap. Notice that the call to MAPINT follows calls to MAPROJ and MAPSET, since both these routines set values that MAPINT needs.

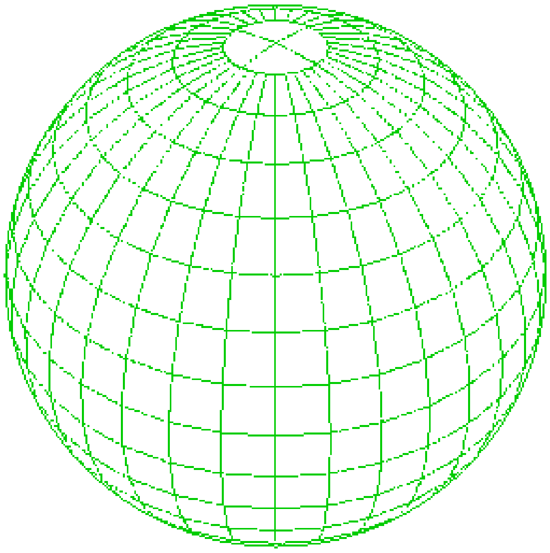
You must call MAPINT to initialize the Ezmap utility after calling MAPPOS, MAPROJ, or MAPSET.

Currently, it is okay to call MAPINT many times. You can check the flag IN (which may be retrieved by a call to MPGETI) to determine whether or not a call to MAPINT is required at a given time. You can change internal parameters such as C1, OU, LA, and so on either before or after a call to MAPINT, but for consistency with Conpack, we set Ezmap parameters before initialization.

Mp 3.2 Grids: Drawing latitude and longitude lines

MAPGRD is the routine that Ezmap uses to draw the limb line and the latitude/longitude (grid) lines. There are also several Ezmap parameters associated with changing grid dashline patterns and accuracy. The grid-spacing parameter GR controls how many degrees apart the latitude and longitude lines are drawn. The grid-drawing parameter GD changes the accuracy used for drawing the latitude and longitude lines. The latitude and longitude line spacings cannot be set individually.

Grid spacing



Code segment from cmpgrd.f

```
      CALL COLOR
      CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
      CALL MPSETI ('C2 - COLOR INDEX 2 - GRID', 2)
      CALL MPSETI ('C4 - COLOR INDEX 4 - LIMB LINE', 2)
      CALL MAPROJ ('SV', 40., -50., 0.)
      CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
      CALL MAPSET ('MA', PLIM1, PLIM2, PLIM3, PLIM4)
      CALL MPSETR ('GR - GRID SPACING', 10.)
      CALL MPSETR ('GD - GRID DRAWING RESOLUTION', 10.)
      CALL MAPINT
      CALL MAPGRD
```

Synopsis

```
      CALL MPSETR ('GR', gr)
      CALL MPSETR ('GD', gd)
      CALL MAPGRD
```

Arguments

GR

Real---The GRid spacing parameter chooses the desired spacing, in degrees, of latitude and longitude lines on the globe. GR=10.0 by default.

GD

Real---The Grid Drawing resolution parameter controls how accurately the latitude and longitude lines are mapped onto the globe. GD specifies the distance in degrees between points defining a grid line. 0.001<=GD<=10. By default, GD=1.0.

Discussion

Line 1 of the **cmpgrd.f** code segment sets up the color table. Line 2 turns on drawing of the continental and political outlines. Lines 3 and 4 set the color parameters so that everything drawn by MAPGRD is drawn in green (color index 2). Lines 5 through 7 set up the projection for a satellite view over the Atlantic Ocean.

Line 8 sets the grid lines to be drawn at intervals of 10 degrees. Line 9 sets the accuracy parameter to 10 so that the map requires a minimum of CPU and plotting time.

GD sets the distance between points on each latitude and longitude curve. By setting GD=10 degrees, we are setting the grid lines so that they may not be extremely accurate between calculated points. However, the first exercise below shows that the resulting difference in the curves is reasonably small (knowing that CPU time and CGM size both increase rapidly as GD gets smaller). Line 10 initializes Ezmap, and line 11 draws the grid lines and the limb line (the line around the globe).

If you plan to use **idt** to zoom in on an area one degree across, then you may want to draw the grid lines using points that are closer together on the globe; this increases the accuracy with which those lines are projected on the map.

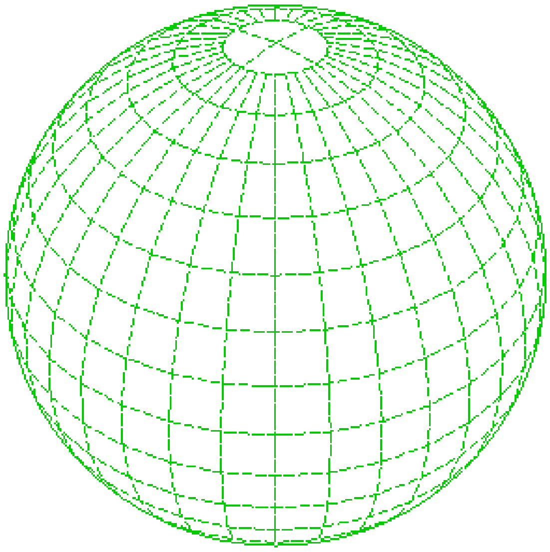
Exercises

1. Using the UNIX command **time**, run the **cmpgd** example with GD=.001, and again with GD=1.0. What is the difference in CPU time on your machine? Plot the two plots on paper and compare their differences.
2. Using **cmpgd.f**, change the grid spacing to 20 degrees.
3. Using your own version of **cezmap.f** (from the Mp 2.2 exercises), add grid spacing to your subroutine call so that you can change it from your main program.

Mp 3.3 Grids: Dash patterns

By default, the Ezmap utility draws grid lines using the Dashline utility. The default gridline pattern is a very short dash followed by a space of the same size. This pattern can be changed using the grid Dashline pattern parameter DA.

Dash patterns in Ezmap



Code segment from cmpdd.f

```
1        CALL MPSETR ('GR - GRID SPACING', 10.)
2        CALL MPSETR ('GD - GRID DRAWING RESOLUTION', 10.)
3        CALL MPSETI ('DA - GRID DASHLINE PATTERN', 64764)
4        CALL MAPINT
5        CALL MAPGRD
```

Synopsis

```
      CALL MPSETI ('DA', ida)
```

Arguments

DA

> *Integer*---The grid DAshline parameter sets a dashed line pattern using the same interface as DASHDB. Construct a 16-bit binary number using 0s to represent blanks and 1s to represent dots or dashes, then convert to base 10. (For example, 0101010101010101 becomes 21845, which produces the dashed line used as the default grid pattern.)

Discussion

Lines 1 and 2 of the **cmpdd.f** code segment set the grid spacing and accuracy just as in the previous module. Line 3 uses the DA parameter to change the grid pattern to one with long lines and small gaps. Line 4 initializes Ezmap, and line 5 draws the grid lines.

Exercises

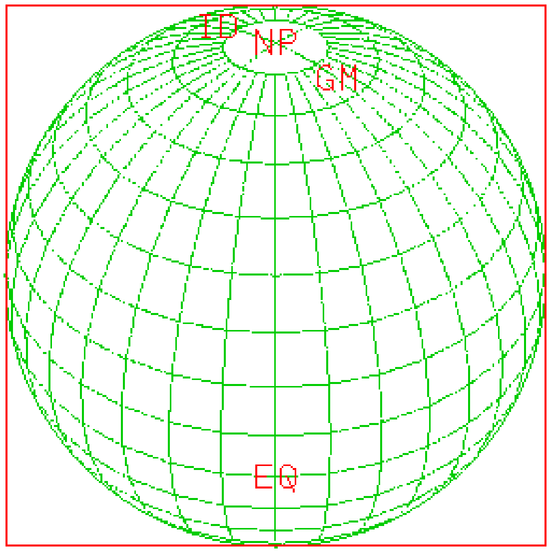
1. Using the **cmpdd** example, make dash patterns use the following two sequences (parentheses enclose the spaces, they are not part of the dash pattern):

```
(- - -   - - -   )
(---   ---   )
```

Mp 3.4 Labeling

There are two parameters and a routine that affect labeling in Ezmap. The routine MAPLBL is called to draw the perimeter as well as labels at the north and south poles, the Greenwich meridian, and the international date line. The LA parameter turns labeling on and off, and LS controls label size.

A labeled map



Code segment from cmplbl.f

```
      CALL COLOR
      CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
      CALL MPSETI ('C2 - COLOR INDEX 2 - GRID', 2)
      CALL MPSETI ('C4 - COLOR INDEX 4 - LIMB LINE', 2)
      CALL MPSETI ('C1 - COLOR INDEX 1 - PERIMETER', 1)
      CALL MPSETI ('C3 - COLOR INDEX 3 - LABELS', 1)
      CALL MAPROJ ('SV', 40., -50., 0.)
      CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
      CALL MAPSET ('MA', PLIM1, PLIM2, PLIM3, PLIM4)
      CALL MPSETI ('LA - MERIDIAN/POLE LABEL FLAG', 1)
      CALL MPSETI ('LS - LABEL SIZE', 40)
      CALL MAPINT
      CALL MAPGRD
      CALL MAPLBL
```

Synopsis

```
CALL MPSETI ('LA', ila)
CALL MPSETI ('LS', ils)
CALL MAPLBL
```

Arguments

LA

> *Integer*---The meridian/pole LAbel flag directs MAPLBL to draw labels at the prime meridian, international dateline, poles, and equator when LA<>0. When LA=0, no labels are drawn. The default is for labels to be drawn.

LS

> *Integer*---The Label Size parameter controls label size using the formula NDC=LS/1024, where NDC is the size in NDCs, and LS is the value given to the LS parameter. The special values 0, 1, 2, and 3 produce labels sized at 0.008, 0.012, 0.016, and 0.024 NDCs, respectively. By default, LS=1 (0.012 NDCs).

Discussion

Line 1 of the **cmplbl.f** code segment sets up the color table. Line 2 turns on drawing of continental and political outlines. Lines 3 through 6 set the color parameters so that everything drawn by MAPGRD is drawn in green (color index 2), and everything drawn by MAPLBL is drawn in orange (color index 1). Lines 7 through 9 set up the projection for a satellite view over the Atlantic Ocean.

Line 10 ensures that labels are drawn, and line 11 sets labels to be about 0.040 NDCs in size. Line 12 initializes Ezmap, line 13 draws the grid and limb lines, and line 14 draws the label and perimeter.

If LA is turned off, MAPLBL won't draw any labels. MAPLBL is also responsible for drawing the perimeter, if desired (when PE<>0). The **cmplbl** example uses white to show things drawn by MAPLBL, and green to show things drawn by MAPGRD.

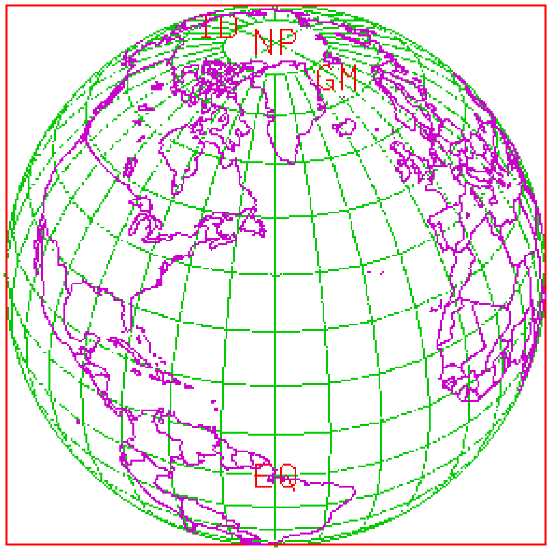
Exercises

1. Using the **cmplbl** example, turn off drawing of the perimeter.
2. Using the **cmplbl** example, set the size of the labels to the default value, but don't set LS=1.
3. Using your own version of **cezmap.f** (from the Mp 2.2 exercises), set up your favorite label options.

Mp 3.5 Drawing political and geographic outlines

The MAPLOT routine draws political and continental outlines. You can control how MAPLOT draws these outlines by using the parameters DO, DD, MV, and RE.

Drawing continental outlines



Code segment from cmplot.f

```
      CALL COLOR
      CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
      CALL MPSETI ('C5 - COLOR INDEX 5 - CONTINENTAL OUTLINES', 5)
      CALL MPSETI ('C7 - COLOR INDEX 7 - COUNTRY OUTLINES', 5)
      CALL MPSETI ('C2 - COLOR INDEX 2 - GRID', 2)
      CALL MPSETI ('C4 - COLOR INDEX 4 - LIMB LINE', 2)
      CALL MPSETI ('C1 - COLOR INDEX 1 - PERIMETER', 1)
      CALL MPSETI ('C3 - COLOR INDEX 3 - LABELS', 1)
      CALL MAPROJ ('SV', 40., -50., 0.)
      CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
      CALL MAPSET ('MA', PLIM1, PLIM2, PLIM3, PLIM4)
      CALL MPSETI ('DO - DOTTED OUTLINE FLAG', 1)
      CALL MPSETI ('DD - DISTANCE BETWEEN DOTS', 7)
      CALL MAPINT
      CALL MAPGRD
      CALL MAPLBL
      CALL MAPLOT
```

Synopsis

```
CALL MPSETI ('DO', ido)
CALL MPSETI ('DD', idd)
CALL MAPLOT
```

Arguments

DO

Integer---The Dotted Outline flag determines whether or not political and continental outlines are solid or dotted. By default DO=0, denoting solid outlines. When DO<>0, dotted outlines are drawn.

DD

Integer---The Distance between Dots parameter determines the space between dots in the dotted line pattern. By default, DD=12 out of 4096, or .003 NDCs. To calculate DD in NDCs, DD=INT(NDC*RE), where NDC is the distance in Normalized Device Coordinates, and RE is the resolution as set by the RE parameter.

MV

Integer---The Minimum Vector length for MAPIT parameter controls the minimum line segment length between two points. Points closer than MV to the previous point are omitted. By default, MV=4, or roughly .001 NDCs.

RE

Integer---The REsolution parameter determines the value DD is divided by to get NDCs. RE=4096 by default.

Discussion

Line 1 of the **cmplot.f** code segment sets up the color table. Line 2 turns on drawing of the continental and political outlines. Lines 3 through 8 set the color parameters so that everything drawn by MAPGRD is drawn in green (color index 2), everything drawn by MAPLBL is drawn in orange (color index 1), and everything drawn by MAPLOT is drawn in magenta (color index 5). Lines 9 through 11 set up the projection for a satellite view over the Atlantic Ocean.

Line 12 turns on dotted line drawing of political and continental outlines, and line 13 spaces these outline dots closer together than the dots that form the grid lines. Line 14 initializes Ezmap, and line 15 draws the grid and limb lines. Line 16 draws the labels and perimeter, and line 17 draws the magenta outlines.

By default, continental outlines are drawn using solid lines from the Dashline utility. If you set DO nonzero, the continental outlines drawn as dots by Ezmap; Dashline is not used. DD controls the distance between dots.

Note: Most users don't need the RE parameter.

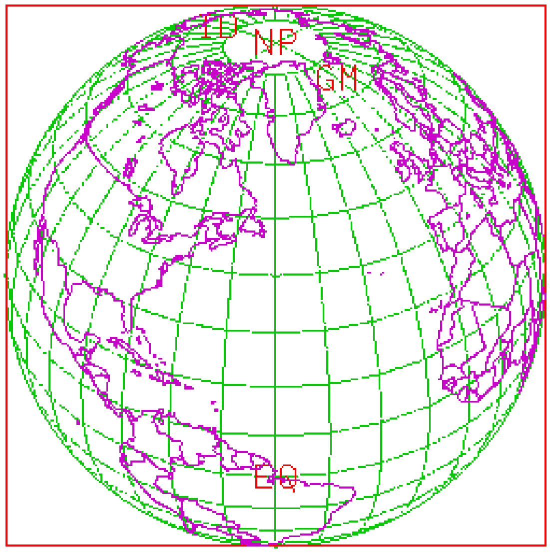
Exercises

1. Using the **cmplot** example, set the dots to be 0.005 NDCs apart.
2. Using the **cmplot** example, draw solid continental outlines.

Mp 3.6 A shortcut

If all you are doing is calling MAPINT, MAPGRD, MAPLBL, and MAPLOT without any calls between, there is a shortcut that you can take to draw your map. MAPDRW calls MAPINT, MAPGRD, MAPLBL, and MAPLOT in that order, making it possible to combine four calls into one.

Plot using MAPDRW



Code segment from cmpdrw.f

```
1        CALL COLOR
2        CALL MPSETC ('OU - OUTLINE DATA FLAG', 'PO')
3        CALL MPSETI ('C5 - COLOR INDEX 5 - CONTINENTAL OUTLINES', 5)
4        CALL MPSETI ('C7 - COLOR INDEX 7 - COUNTRY OUTLINES', 5)
5        CALL MPSETI ('C2 - COLOR INDEX 2 - GRID', 2)
6        CALL MPSETI ('C4 - COLOR INDEX 4 - LIMB LINE', 2)
7        CALL MPSETI ('C1 - COLOR INDEX 1 - PERIMETER', 1)
8        CALL MPSETI ('C3 - COLOR INDEX 3 - LABELS', 1)
9        CALL MAPROJ ('SV', 40., -50., 0.)
10       CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
11       CALL MAPSET ('MA', PLIM1, PLIM2, PLIM3, PLIM4)
12       CALL MPSETR ('GR - GRID SPACING', 10.)
13       CALL MPSETR ('GD - GRID DRAWING RESOLUTION', 10.)
14       CALL MPSETI ('LA - MERIDIAN/POLE LABEL FLAG', 1)
15       CALL MPSETI ('LS - LABEL SIZE', 40)
16       CALL MPSETI ('DO - DOTTED OUTLINE FLAG', 1)
17       CALL MPSETI ('DD - DISTANCE BETWEEN DOTS', 7)
18       CALL MAPDRW
```

Synopsis

```
      CALL MAPDRW
```

Discussion

Lines 1 through 11 of the **cmpdrw.f** code segment match lines 1 through 11 of the **cmplot.f** code segment in the preceding module. Line 1 sets up the color table. Line 2 turns on drawing of the continental and political outlines. Lines 3 through 8 set the color parameters so that everything drawn by MAPGRD is drawn in green (color index 2), everything drawn by MAPLBL is drawn in orange (color index 1), and everything drawn by MAPLOT is drawn in magenta (color index 5). Lines 9 through 11 set up the projection for a satellite view over the Atlantic Ocean.

Lines 12 and 13 of the **cmpdrw.f** code segment set grid parameters discussed in module "Mp 3.2 Grids: Drawing latitude and longitude lines." Lines 14 and 15 set label parameters discussed in the preceding module. Line 16 turns on dotted line drawing of political and continental outlines, and line 17 sets the dots closer together than gridline dots.

The shortcut occurs as line 18 calls MAPDRW. Notice that since MAPDRW calls MAPINT, calls to MAPROJ, MAPSET, and MAPPOS must be made before calling MAPDRW. Also, since MAPDRW calls MAPINT, MAPGRD, MAPLBL, and MAPLOT, parameters that affect those routines must be set before calling MAPDRW.

You can probably use MAPDRW in almost any simple map drawing. However, as shown in Ezmap section "Mp 4. Producing maps with masking or filled areas," it is often desirable to be able to call MAPINT and MAPLOT separately.

Exercises

1. Examine your own version of **cezmap.f** (from the Mp 2.2 exercises) to see if it can use MAPDRW.

Mp 4. Producing maps with masking or filled areas

Drawing maps with masking or filled areas is very much like drawing simple maps. However, it takes a few extra steps and uses a slightly different set of routines.

Ezmap functional outline

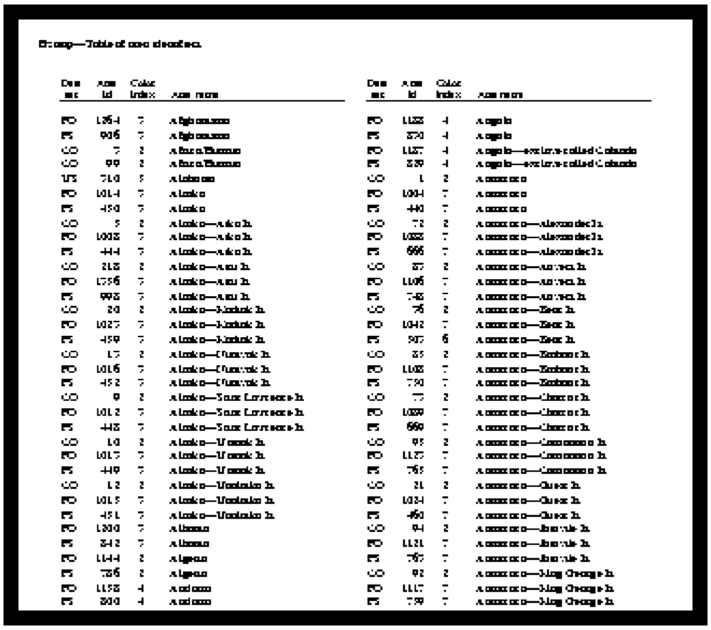
```
----------------------------------------------------------
    1.  Open GKS
    2.  Define position on plotter frame
    3.  Define map projection
    4.  Choose map limits
    5.  Choose desired political boundaries
 *  6.  Initialize Areas
 *  7.  Initialize Ezmap
 *  8.  Add geographic boundaries (or outlines) to area map
 *  9.  Draw geographic labels
    10.  Draw points and lines on your map
 *  11.  Draw desired latitude and longitude lines
 *  12.  Fill desired areas
    13.  Draw geographic outlines
    14.  Call FRAME
    15.  Close GKS
----------------------------------------------------------
```

* Steps discussed in this section.

Mp 4.1 Color and area identifiers in Ezmap

Section "Mp 6. Table of Ezmap area identifiers" and the MAPACI function can be used as tools for choosing a particular geographic area or set of areas for filling, shading, or masking.

Sample Ezmap area identifier table



Code segment from cmpmsk.f

```
      SUBROUTINE MASK (XC, YC, MCS, AREAID, GRPID, IDSIZE)

      INTEGER AREAID(IDSIZE), GRPID(IDSIZE), ID
      REAL XC(MCS), YC(MCS)
      DO 10, I=1, IDSIZE
         IF (GRPID(I) .EQ. 1) ID = AREAID(I)
  10  CONTINUE
      IF ((MAPACI(ID) .EQ. 1) .AND. (MCS .GE. 2)) THEN
         CALL CURVED (XC, YC, MCS)
      ENDIF
      RETURN
      END
```

Synopsis

```
MAPACI (IDAREA)
```

Function

MAPACI

> *Integer, Function*---Obtains, for a given area, a suggested color index appropriate for use in creating a color map having adjacent areas of different colors. Returns an integer between 1 and 7 inclusive.

Arguments

IDAREA

Integer, Input---Area identifier for an area defined by an area map created by MAPBLA.

Discussion

The "Sample Ezmap area identifier table" illustration shows part of the table printed in section "Mp 6. Table of Ezmap area identifiers." The first column of this table contains the abbreviated name of the dataset in which the areas are defined. (PO stands for Political Outlines, US stands for US state outlines, CO is for Continental Outlines, and PS is for all of the above.)

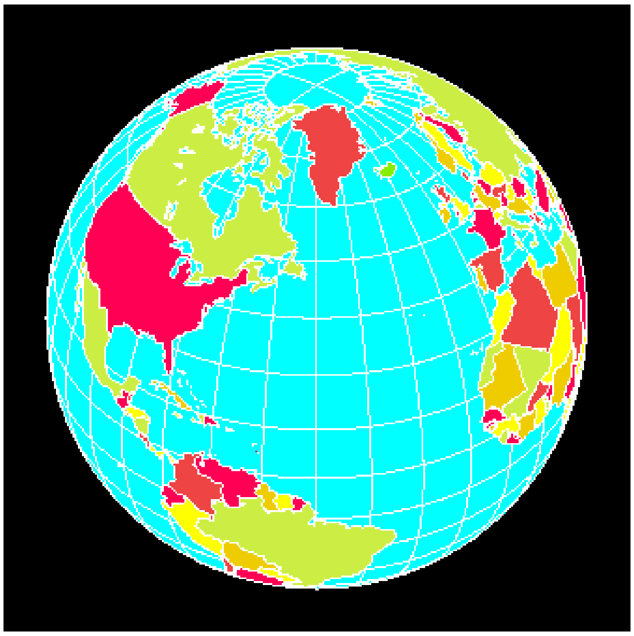
The second column contains the area identifier for the area so you can uniquely address any area in the map database. The third column shows the suggested color index for the area. These color indices are carefully chosen so that no country or state has the same color as any of its neighbors. Also note that all bodies of water like oceans and lakes have a color index of 1. This makes it easy for your program to pick out the oceans for masking purposes or to choose an appropriate color for the oceans.

In line 8 of the **cmpmsk.f** code segment, note that MAPACI is called as a function rather than as a subroutine. MAPACI returns an integer value that can be used to detect ocean or land masses, to set color-fill values, and for many other purposes. A complete discussion of this code segment appears in module "Mp 4.6 Grid lines with masking: Writing a masking routine."

Mp 4.2 Ezmap group identifiers

The Ezmap routine MAPBLA puts two groups of edges into an area map. Group 1 holds all the outline information for the maps; group 2 holds vertical strip outlines. This module discusses how to change which group identifiers are used, and how to set the number of vertical strips.

Plot drawn with vertical strips



Code segment from cmpgrp.f

```
      CALL MPSETI ('VS - VERTICAL STRIPS', 9)
      CALL MPSETC ('OU - OUTLINE DATA FLAG', OUTLN)
      CALL MPSETR ('GR - GRID SPACING', GRD)
      CALL MAPROJ (PROJ, PLAT, PLON, ROTA)
      IF (PROJ .EQ. 'SV') CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 5.)
      CALL MAPSET (JLIM, PLIM1, PLIM2, PLIM3, PLIM4)
      CALL MAPINT
```

Synopsis

```
CALL MPSETI ('VS', ivs)
CALL MPSETI ('G1', ig1)
CALL MPSETI ('G2', ig2)
```

Arguments

VS

Integer---The Vertical Stripping parameter specifies the number of strips into which a plot is to be divided. Vertical strips can be created to minimize the number of points that define an area to be filled so that the plot can be drawn on limited hardware. By default, VS=1.

G1

Integer---The Group 1 parameter is the group identifier for the perimeter and the set of projected boundary lines implied by your map projection and limits.

G2

Integer---The Group 2 parameter is the group identifier for the perimeter and the boundary lines for vertical strips.

Discussion

Currently, there are four groups of edges defined by default in NCAR Graphics:

- 1 Ezmap geographic information
- 2 Ezmap vertical strips
- 3 Conpack contour lines
- 4 Conpack vertical strips

Most people will never need to change any of these group identifiers. However, using Ezmap parameters G1 and G2, you can change the defaults of the first two groups. In most cases, you probably do not want to use the same values for Ezmap edge groups and Conpack edge groups.

The **cmpgrp.f** code segment creates a plot using nine vertical strips; this has the effect of making the areas created both smaller and simpler. Line 1 sets the number of vertical strips by giving VS the value of 9. Lines 2 through 9 then set up map drawing normally.

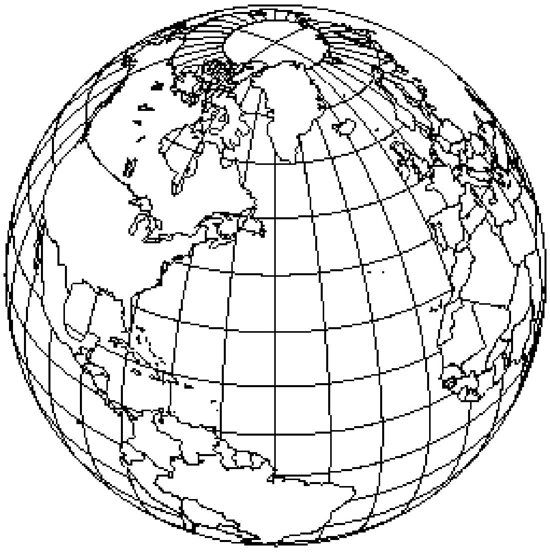
Exercises

1. Change the value of VS in **cmpgrp.f**, and watch how it is plotted to the screen.

Mp 4.3 Initialize Ezmap with Areas

Initializing Ezmap with Areas requires three routines. ARINAM initializes the area map, MAPINT initializes Ezmap, and MAPBLA is used to add the geographic outlines to the area map after both ARINAM and MAPINT have been called.

Initializing Ezmap with Areas



Code segment from cmpmsk.f

```
1       EXTERNAL MASK
2       CALL MAPINT
3       CALL ARINAM (MAP, LMAP)
4       CALL MAPBLA (MAP)
5       CALL MAPGRM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, ISIZ, MASK)
```

Synopsis

```
    CALL MAPINT
    CALL ARINAM (MAP, LMAP)
    CALL MAPBLA (MAP)
```

Arguments

MAP(LMAP)

> *Integer array, Workspace*---An integer array in which an area map of the globe is to be constructed.

LMAP

> *Integer, Input*---Length of the MAP array. LMAP is unchanged by the call.

Discussion

Line 1 of the **cmpmsk.f** code segment declares the masking routine to be external so that the compiler knows that MASK is a subroutine rather than a variable. Line 2 initializes Ezmap and sets up our projection and map limits. Line 3 initializes the area map using the Areas routine ARINAM, and line 4 adds the projections of the geographic outlines to the area map.

Although MAPINT and ARINAM have been discussed previously, they are included in the synopsis because both routines are critical to using Ezmap with Areas. As shown before, MAPINT sets up the map limits and projection, defining a set of lines that divide up the plane. ARINAM initializes the area map array. MAPBLA takes the lines defined by MAPINT and adds them to the area map, using the default of edge group 1 for geographical map outlines and the default of edge group 2 for creating a set of vertical strips. Please see the previous module for further information on setting and using Ezmap group identifiers and vertical strips.

If you set the area map too small, the error message:

```
ERROR 5 IN AREDAM - AREA-MAP ARRAY OVERFLOW
```

occurs when MAPBLA is called. There is no good way to predict exactly how large the area map should be before adding edges to it. Try setting LMAP=50000 and increasing it as necessary. It is not unusual to need LMAP=250000 in maps with lots of tiny regions.

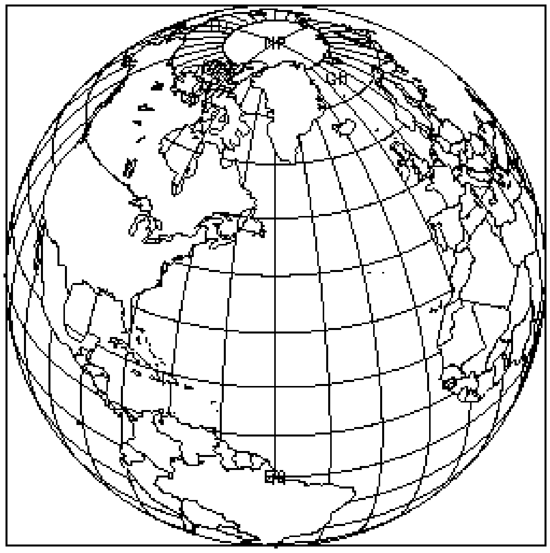
Exercises

1. Copy **cmpmsk.f** into your own directory and name it **cmapa.f**. Set it up so that you can change parameters (such as label size, grid options, and so on) at any time. You will use **cmapa.f** in subsequent exercises.

Mp 4.4 Labeling

Just as we used MAPLBL to add labels to our simple map, we use it to write labels on a map with masked or filled areas. As before, the LA parameter turns labeling on and off, and the LS parameter controls label size for labels drawn by MAPLBL.

A labeled masked map



Code segment from cmplab.f

```
1       EXTERNAL MASK
2       CALL MPSETI ('LS - LABEL SIZE', 20)
3       CALL MAPINT
4       CALL ARINAM (MAP, LMAP)
5       CALL MAPBLA (MAP)
6       CALL MAPGRM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, ISIZ, MASK)
7       CALL MPSETI ('EL - ELLIPTICAL PERIMETER FLAG', 1)
8       CALL MAPLBL
9       CALL MAPLOT
```

Synopsis

```
CALL MAPLBL
CALL MPSETI ('LA', ila)
CALL MPSETI ('LS', ils)
```

Arguments

LA

Integer---The meridian/pole LAbel flag directs MAPLBL to draw labels at the prime meridian, international dateline, poles, and equator when LA<>0. When LA=0, no labels are drawn. The default is for labels to be drawn.

LS

Integer---The Label Size parameter controls label size using the formula NDC=LS/1024 where NDC is the size in NDCs, and LS is the value given to the LS parameter. The special values 0, 1, 2, and 3 produce labels sized at 0.008, 0.012, 0.016, and 0.023 NDCs, respectively. By default, LS=1 (0.012 NDCs).

Discussion

Line 1 of the **cmplab.f** code segment declares the masking routine to be external so that MAPGRM doesn't core dump. Line 2 sets the label size to be a bit larger than normal. Lines 3 and 4 initialize Ezmap and Areas respectively. Line 5 adds geographic and political outlines to the area map. Line 6 draws grid lines over water and uses the area map to mask them over land. Line 7 draws the ellipse that provides the limb line for this plot. Line 8 draws the labels.

If LA is turned off, MAPLBL does not draw any labels. MAPLBL also draws the perimeter if one is specified. The perimeter parameter PE<>0 by default.

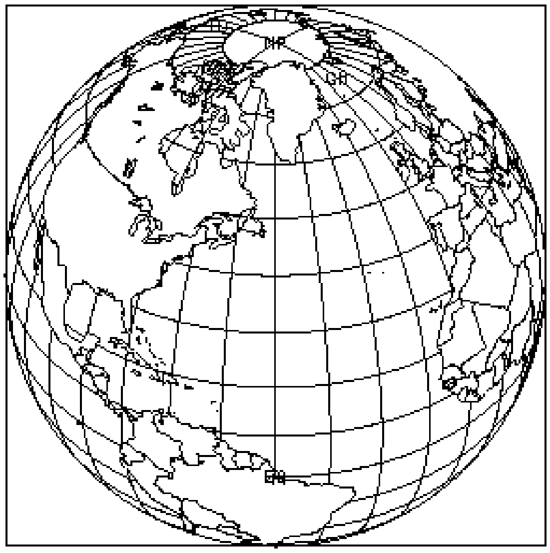
Exercises

1. Change the **cmplbl** example so no perimeter is drawn.
2. Using the **cmplbl** example, set the size of the labels to the default value but don't set LS=1.
3. Using your own version of **cmapa.f**, set up your favorite label options.

Mp 4.5 Grid lines with masking

The Ezmap routine MAPGRM allows you to draw grid lines and mask them over any geographic area such as oceans, land, or specific countries or continents.

A masked map



Code segment from cmplab.f

```
      EXTERNAL MASK
      CALL MAPINT
      CALL ARINAM (MAP, LMAP)
      CALL MAPBLA (MAP)
      CALL MAPGRM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, ISIZ, MASK)
      CALL MAPLBL
      CALL MAPLOT
```

Synopsis

```
    CALL MAPGRM (MAP, XCS, YCS, MCS, IAREA, IGRP, ISIZ, LPR)
```

Arguments

MAP

> *Integer array, Workspace*---Area map initialized in ARINAM and constructed by MAPBLA.

XCS(MCS), YCS(MCS)

> *Real arrays, Workspace*---Workspace used by MAPGRM in calls to the user-supplied routine LPR.

MCS

> *Integer, Input*---Dimension of the arrays XCS and YCS. MCS must be greater than or equal to 2; the value 100 is suggested.

IAREA(ISIZ), IGRP(ISIZ)

> *Integer arrays, Workspace*---To be used by MAPGRM in calls to the user-supplied routine LPR.

ISIZ

> *Integer, Input*---Dimension of the arrays IAREA and IGRP. ISIZ must be greater than or equal to 2. If other groups of boundary lines are added to the area map (by calling MAPITA and MAPIQA or by calling AREDAM), then ISIZ must be increased by the number of such groups.

LPR

> *Subroutine*---A user-supplied line-processing subroutine that is called once for each piece of the masked grid. Each such piece is contained in exactly one area defined by the area map. LPR must be declared external in the routine that calls MAPGRM. This subroutine is discussed in detail in the next module.

Discussion

Line 1 declares MASK as external so that MAPGRM knows that MASK is a subroutine and not an integer variable. Lines 2 through 4 of the **cmplab.f** code segment initialize Ezmap with Areas, as discussed in module "Mp 4.3 Initialize Ezmap with Areas." Line 5 calls MAPGRM with the area map and the user-supplied masking subroutine MASK. MASK is discussed in the next module.

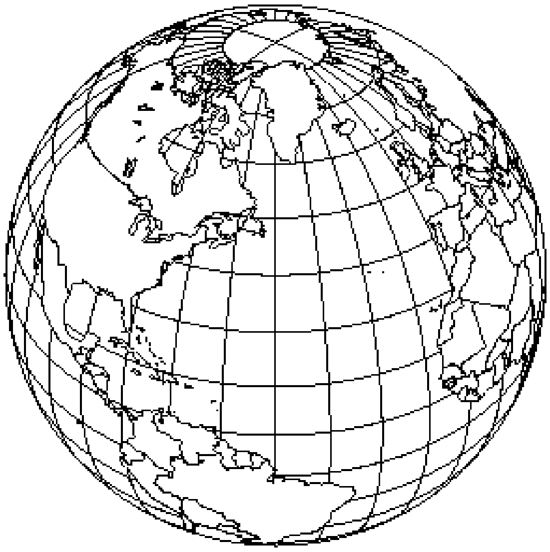
ISIZ, the size of the group and area identifier arrays (IAREA and IGRP), is determined by how many groups of lines you have added to the area map. Remember that the first group is the geographical outline set. Other groups might include vertical stripping if you are using it, contour lines, or lines that you may want to add to the area map using the Ezmap line-drawing routines.

Lines 6 and 7 label the globe and draw continental outlines.

Mp 4.6 Grid lines with masking: Writing a masking routine

Just as Areas doesn't really do any filling or masking, MAPGRD doesn't actually draw masked grid lines. MAPGRD calls a routine that you must write to correctly mask the grid lines. This makes it possible for you to precisely control which portions of the lines are masked.

Masking grid lines



Code segment from cmpmsk.f

```
      SUBROUTINE MASK (XC, YC, MCS, AREAID, GRPID, IDSIZE)
      INTEGER AREAID(IDSIZE), GRPID(IDSIZE), ID
      REAL XC(MCS), YC(MCS)
      DO 10, I=1, IDSIZE
         IF (GRPID(I) .EQ. 1) ID = AREAID(I)
 10   CONTINUE
      IF ((MAPACI(ID) .EQ. 1) .AND. (MCS .GE. 2)) THEN
         CALL CURVED (XC, YC, MCS)
      ENDIF
      RETURN
      END
```

Synopsis

```
      SUBROUTINE LPR (XCS, YCS, NCS, IAREA, IGRP, ISIZ)
      DIMENSION XCS(*), YCS(*), IAREA(*), IGRP(*)
```

Arguments

LPR

> *Subroutine*---A user-supplied line-processing subroutine that is called once for each piece of the masked grid. Each such piece is contained in exactly one area defined by the area map. LPR must be declared external in the routine that calls MAPGRM. You must use the following structure when you write subroutine LPR:

```
 SUBROUTINE LPR (XCS, YCS, NCS,
+ IAREA, IGRP, ISIZ)
 DIMENSION XCS(*), YCS(*), IAREA(*),
+ IGRP(*)
```

> *(code to process polyline defined by XCS, YCS, IAREA, and IGRP)*

```
RETURN
END
```

XCS(NCS), YCS(NCS)

Real arrays, Input---Hold the X and Y coordinates, in the fractional coordinate system, of NCS points defining a piece of the grid. These are input arrays for the subroutine LPR.

NCS

Integer, Input---Number of X and Y coordinates in the arrays XCS and YCS.

IAREA(ISIZ), IGRP(ISIZ)

Integer arrays, Input---Hold ISIZ pairs of identifiers for the area within which the piece of the polyline lies. For each value of I from 1 to ISIZ, IAREA(I) is the area identifier for the area with respect to the group of edges specified by the group identifier IGRP(I).

ISIZ

Integer, Input---Number of values in IAREA and IGRP. ISIZ is equal to the number of groups of edges that you put in the area map.
You can assume that MAPGRM has made this call:

```
CALL SET (VPL, VPR, VPB, VPT, VPL, VPR, VPB, VPT, 1)
```

This call ensures correct results if the normalized device coordinates in XCS and YCS are used in calls to such routines as the GKS line-drawing routine GPL or the Dashline routine CURVED. LPR may make its own SET call to achieve some other effect.

Discussion

In this code segment, the line processing routine (shown as LPR in the synopsis) is named MASK. Lines 4 through 6 set up a do loop to extract the geographic area identifier of the region passed into MASK. Line 7 checks to see if the region is over land or water, and it checks to see if the line has at least two points in it. If both conditions are true, then the line is drawn with the Dashline routine CURVED in line 8. Otherwise, no line is drawn.

By using the color identifier to pick out land values, you could draw grid lines only over land. Similarly, by looking up the area identifiers for a given country in section "Mp 6. Table of Ezmap area identifiers," you could draw grid lines either only over a given country, or over everything except a given country.

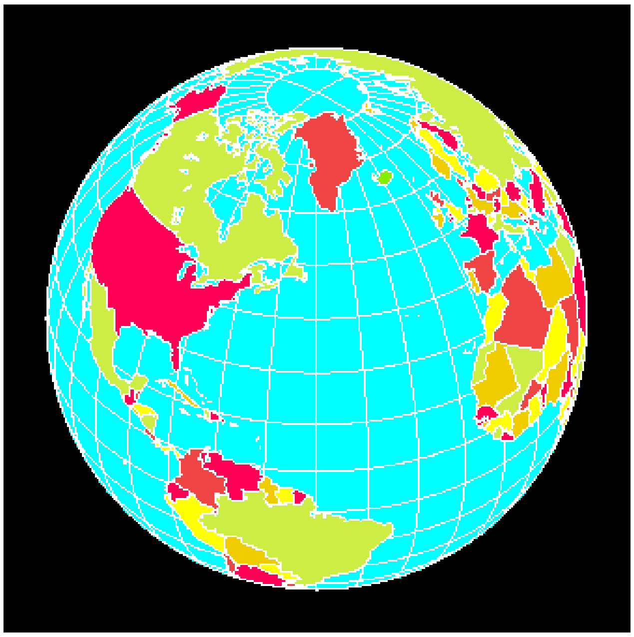
Exercises

1. Using **cmpmsk.f**, modify MASK so that it draws grid lines only over land masses.
2. Modify your own version of **cmapa.f** (from the Mp 4.3 exercises) so it draws only the grid lines that you want.

Mp 4.7 Filling areas

In this module, we use the Areas routine ARSCAM to fill areas, just as we did in the Areas chapter of this tutorial. The biggest difference is in the way we choose the areas to be filled.

A color-filled globe



Code segment from cmpfil.f

```
      CALL COLOR
      CALL MAPINT
      CALL ARINAM (MAP, LMAP)
      CALL MAPBLA (MAP)
      CALL GSFAIS (1)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, ISIZ, FILL)
      CALL MAPGRM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, ISIZ, MASK)
      CALL MAPLOT
```

Synopsis

```
CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, ISIZ, FILL)
```

Arguments

MAP

> *Integer array, Workspace*---The area map that has been initialized by a call to ARINAM and to which edges have been added by calls to AREDAM. If you did not preprocess the area map by calling ARPRAM, ARSCAM calls it before doing anything else.

XWRK(NWRK), YWRK(NWRK)

Real arrays, Workspace---The X and Y coordinates defining the edge of a given area, for use by ARSCAM in calls to the area-processing routine APR.

NWRK

Integer, Input---Dimension of each of the arrays XWRK and YWRK. A nice size for NWRK is 1000.

IAREA(ISIZ), IGRP(ISIZ)

Integer arrays, Workspace---Each dimensioned NGRPS, for use by ARSCAM in calls to the area-processing routine APR.

ISIZ

Integer, Input---Dimension of each of the arrays IAREA and IGRP. ISIZ>=*n*, where *n* is the number of groups in the area map.

FILL

Subroutine---A user-supplied area-processing routine that is called by ARSCAM and that must be declared external in the routine that calls ARSCAM. The FILL subroutine is described in full detail in the next module.

Discussion

Line 1 of the **cmpfil.f** code segment sets up the color table. Lines 2, 3, and 4 initialize Ezmap and Areas, then add the geographic map to the area map.

Line 5 sets the GKS interior fill style to "solid" to produce solid fill. Line 6 calls the Areas scan area map routine with user-supplied fill routine so that each country is filled. Line 7 draws the grid lines over water, and line 8 draws the continental and political boundaries. The order of the overlaying done by these calls is critical to produce proper results. You must call detail-drawing routines after filling, since color-fill draws over anything that might have been there previously.

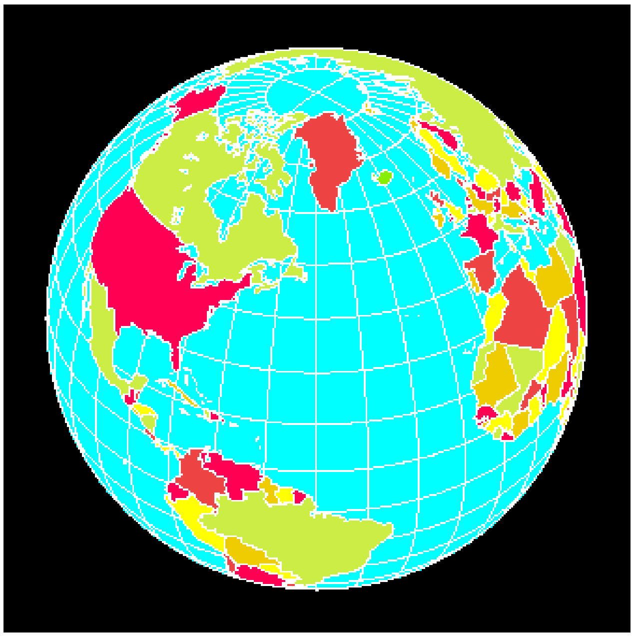
Exercises

1. Using your own version of **cmapa.f** (from the Mp 4.3 exercises), set it up to fill the regions of your choice. You may also want to choose a different color table for your routine.

Mp 4.8 Filling areas: Writing a fill routine

The fill routine you write to work with Ezmap is nearly identical to one you would use to fill any type of area. Because there are so many area identifiers in Ezmap, Ezmap provides a tool that allows you to fill water with one color or fill pattern, then fill adjacent countries or states with different colors or fill patterns. This tool is a function named MAPACI.

A color-filled globe



Code segment from cmpfil.f

```
      SUBROUTINE FILL (XWRK, YWRK, NWRK, IAREA, IGRP, IDSIZ)
      DIMENSION XWRK(*), YWRK(*), IAREA(*), IGRP(*)
      ID=0
      DO 10, I=1, IDSIZ
             IF (IGRP(I) .EQ. 1) ID = IAREA(I)
  10  CONTINUE
      IF (ID .GE. 1) THEN
             CALL GSFACI (MAPACI(ID) + 1)
             CALL GFA (NWRK, XWRK, YWRK)
      ENDIF
      RETURN
      END
```

Synopsis

```
SUBROUTINE FILL (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)
DIMENSION XWRK(*), YWRK(*), IAREA(*), IGRP(*)
```

Arguments

FILL

Subroutine---A user-supplied area-processing routine that is called by ARSCAM and that must be declared external in the routine that calls ARSCAM. The FILL subroutine must have the following form:

```
      SUBROUTINE FILL (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)
      DIMENSION XWRK(*), YWRK(*), IAREA(*), IGRP(*)
```

(code to process the area defined by XWRK, YWRK, IAREA, and IGRP)

```
      RETURN
      END
```

XWRK(NWRK), YWRK(NWRK)

> *Real arrays, Input*---The X and Y coordinates, in the fractional coordinate system, of NWRK points defining a polygonal area. The last of these points is a duplicate of the first. Holes in an area are traced in such a way as to maximize the probability of hardware fill working properly by using vertical lines to get to and from the holes and tracing them in the proper direction.

NWRK

> *Integer, Input*---Number of X and Y coordinates in the arrays XWRK and YWRK. A nice starting value for NWRK is 1000.

IAREA(NGRPS), IGRP(NGRPS)

> *Integer arrays, Input*---Hold NGRPS pairs of identifiers for the area defined by XWRK and YWRK. For each value of I from 1 to NGRPS, IAREA(I) is the area identifier for the area with respect to the group of edges specified by the group identifier IGRP(I).

NGRPS

> *Integer, Input*---Number of values in IAREA and IGRP. NGRPS is equal to the number of groups of edges that you put in the area map.

Discussion

As discussed in module "Mp 4.1 Color and area identifiers in Ezmap," you can either isolate an area by looking up its area identifiers in section "Mp 6. Table of Ezmap area identifiers," or you can isolate the desired area using the color index. In this case, we want to ensure that oceans having an Ezmap color index of 1 are colored aquamarine, which has a GKS color index of 2.

Lines 4 through 6 of the **cmpfil.f** code segment retrieve the area identifier for the geographic region by checking each element of the group array for group identifier 1, and assigning its associated area id to ID. Line 7 checks to see if the area is over the map, and if so, line 8 chooses a color index by retrieving the suggested Ezmap color using MAPACI. Line 9 fills the area.

Exercises

1. Use section "Mp 6. Table of Ezmap area identifiers" or the **ngfile** utility to determine the correct area identifier, then use the **cmpfil** example to change the color of Canada to be different than either the US or Russia.
2. Using your own version of **cmapa.f** (from the Mp 4.3 exercises), modify it to do color fill.

Mp 5. Points, lines, and inverse transformations

Ezmap has two routines that project points onto a map, two sets of routines for adding lines to a map, and one routine that finds the latitude and longitude of a point whose projection on the u/v plane is known. The first set of routines deals strictly with adding lines to geographic or "simple" maps (drawing the lines), and the second set of routines adds lines to an area map (subdividing areas, but no lines are drawn). This section discusses each of these types of routines in detail.

Ezmap functional outline

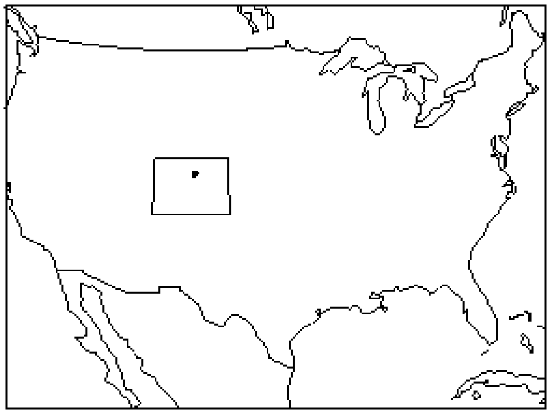
```
-----------------------------------------------------------
    1.  Open GKS
    2.  Define position on plotter frame
    3.  Define map projection
    4.  Choose map limits
    5.  Choose desired political boundaries
    6.  Initialize Areas
    7.  Initialize Ezmap
    8.  Add geographic boundaries (or outlines) to area map
    9.  Draw geographic labels
 *  10.  Draw points and lines on your map
    11.  Draw desired latitude and longitude lines
    12.  Fill desired areas
    13.  Draw geographic outlines
    14.  Call FRAME
    15.  Close GKS
-----------------------------------------------------------
```

* Steps discussed in this section.

Mp 5.1 Projecting a point onto the map

Ezmap offers two routines that project points onto a geographic map: MAPTRA and MAPTRN. Given the latitude and longitude of a point on the globe, both routines return the u/v coordinates of that point’s projection. You can then take this information and do nearly anything with it, like draw a label or a polymarker. Conpack uses MAPTRA to project and draw contour lines.

Marking up a map



Code segment from cmptra.f

```
      CALL MAPDRW
      CALL MAPTRA (40., -105.15, X, Y)
      IF (X .NE. 1.E12) CALL POINTS (X, Y, 1, -3, 0)
```

Synopsis

```
CALL MAPTRA (RLAT, RLON, UVAL, VVAL)
```

Arguments

RLAT

Real, Input---The latitude of a point on the globe. RLAT must be between -90.0 and +90.0 inclusive.

RLON

Real, Input---The longitude of a point on the the globe. RLON must be between -540.0 and +540.0 inclusive. Positive longitudes are measured eastward from the Greenwich meridian, and negative longitudes are measured westward from the Greenwich meridian.

UVAL, VVAL

Real, Output---The point (UVAL, VVAL) is the point on the u/v plane into which the point with latitude RLAT and longitude RLON projects. The units of UVAL and VVAL depend on the projection.

Discussion

MAPTRA is identical to MAPTRN, with one exception: if the point is outside the boundary of the map, MAPTRA returns UVAL equal to 1.E12 and MAPTRN returns the u/v coordinates for the point. If you use MAPTRN, you may need to check to determine if the point is within limits.

If the point is not projectable on the map, both routines return UVAL equal to 1.E12.

Line 1 of the **cmptra.f** code segment draws the map as a reminder that the map projection and limits have been set up and that the map has been drawn. Line 3 calls MAPTRA to get the user coordinates of the point. If the coordinates are on the projection, then an asterisk is drawn over Boulder, Colorado, USA. MAPTRN or MAPTRA can be called any time after MAPINT is called.

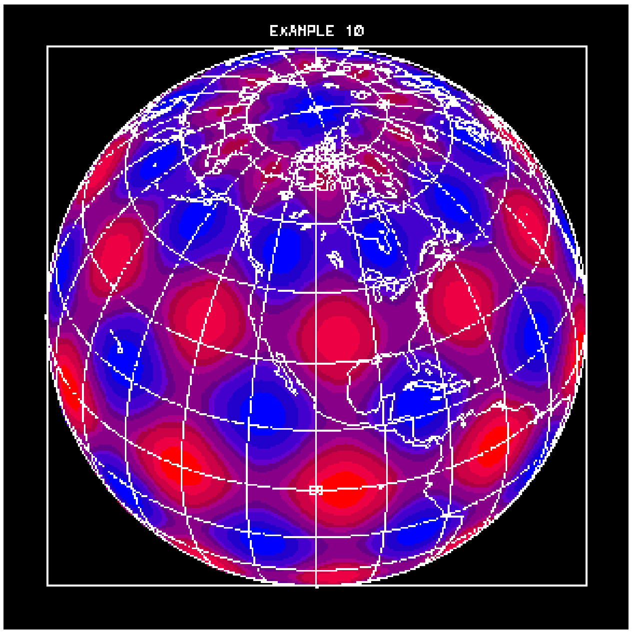
Exercises

1. Using a full-globe mercator projection, draw a red circle over Lhasa, Tibet (29., 91.)
2. Using the preceding exercise, write the city names over Beijing, China (39., 116.) and Machu Picchu, Peru (-13., -72.).

Mp 5.2 Inverse transformations

Sometimes it is necessary to find the latitude and longitude values of a point, knowing only the user coordinates set up by the map transformation. MAPTRI allows you to do this for all Ezmap plots.

A cell array drawn with MAPTRI



Code segment from mpex10.f

```
      DO 103, I=1, NCLS
        X = CFUX (.05 + .90 * (REAL (I-1) + .5) / REAL (NCLS))
        DO 102, J=1, NCLS
          Y = CFUY (.05 + .90 * (REAL (J-1) + .5) / REAL (NCLS))
          CALL MAPTRI (X, Y, RLAT, RLON)
          IF (RLAT .NE. 1.E12) THEN
            DVAL = .25 * (1. + COS (DTOR * 10. * RLAT)) +
     +         .25 * (1. + SIN (DTOR * 10. * RLON)) * COS (DTOR * RLAT)
```

```
8                ICRA (I,J) = MAX (2, MIN (NCLR+1, 2+INT (DVAL * REAL (NCLR))))
9            ELSE
10               ICRA (I,J) = 0
11           ENDIF
12 102     CONTINUE
13 103   CONTINUE
14       CALL GCA (CFUX(.05), CFUY(.05), CFUX(.95), CFUY(.95), 1000, 1000,
        + 1, 1, NCLS, NCLS, ICRA)
15       CALL MAPDRW
```

Synopsis

```
      CALL MAPTRI (X, Y, RLAT, RLON)
```

Arguments

X, Y

> *Real, Input*---The X and Y user coordinates of the point for which you want latitude and longitude values.

RLAT, RLON

> *Real, Output*---The latitude and longitude values for a point specified in user coordinates. If RLAT=1.E12, then the point is outside the map projection.

Discussion

The **mpex10** example uses MAPTRI to create a cell array representing a data field on a projected globe.

Lines 1 through 13 of the **mpex10.f** code segment pick values to fill a cell array that will be used to color the globe. In line 2, CFUX takes an X coordinate in NDCs and returns an X coordinate in user coordinates. Similarly, CFUY in line 4 takes a Y coordinate of a point in NDCs and returns a Y value in user coordinates.

Line 5 uses MAPTRI to retrieve the coordinates of the point in latitude and longitude. Since MAPTRI returns a value of RLAT=1.E12 if the point is not over a plotted portion of the globe, line 6 checks to see if the point maps onto the portion of the globe that will be plotted. If it does, then a value for the cell array is specified there. Otherwise, line 10 sets the cell array element to black. Line 14 fills the cell array, and line 15 draws the map over it.

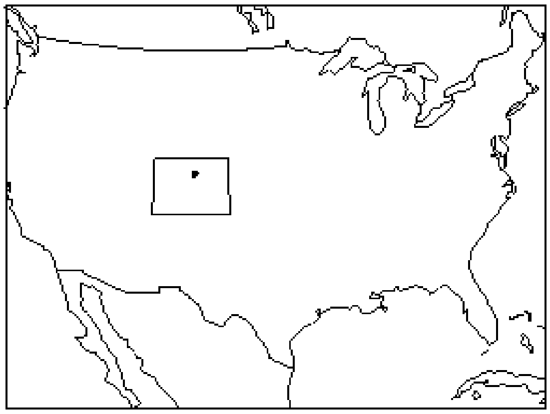
Exercises

1. Modify **mpex10.f** to put a cell array only over the United States.

Mp 5.3 Drawing lines on a simple map

There are two routines to help you add lines to geographic maps. MAPIT draws line segments and omits nonvisible portions. MAPIQ terminates line segments drawn by MAPIT.

Drawing lines on your map



Code segment from cmptra.f

```
      CALL MAPDRW
      CALL MAPIT (37., -109., 0)
      CALL MAPIT (41., -109., 1)
      CALL MAPIT (41., -102., 1)
      CALL MAPIT (37., -102., 1)
      CALL MAPIT (37., -109., 1)
      CALL MAPIQ
```

Synopsis

```
CALL MAPIT (RLAT, RLON, IFST)
CALL MAPIQ
```

Arguments

RLAT

Real, Input---The latitude, in degrees, of a point to which the "pen" is to be moved. RLAT must be between -90. and +90., inclusive.

RLON

Real, Input---The longitude, in degrees, of a point to which the "pen" is to be moved. RLON must be between -540. and +540., inclusive.

IFST

Integer, Input---The first point flag allows you to control whether or not line segments are drawn.
0 Do a "pen-up" move. IFST should be zero for the first point in each curve.
1 Do a "pen-down" move only if the distance from the last point to the new point is greater than MV/RE. IFST should be positive for any point after the first point in a curve.
>1 Do a "pen-down" move regardless of the distance from the last point to the new one.

IFST should be positive for any point after the first point in a curve.

Discussion

MAPIT draws lines on the map. It is called by Ezmap and, if desired, it can be called by the user. MAPIT attempts to omit nonvisible portions of lines and to handle "crossover"---the jump from one side of the map to the other caused by the projection's having slit the globe along some half of a great circle and laid it open with the two sides of the slit at opposite ends of the map. Crossover can occur on cylindrical and conical projections; MAPIT handles it gracefully on the former and not so well on the latter.

The Ezmap parameter DL determines whether MAPIT draws solid lines or dotted lines. Dotted lines are drawn using calls to POINTS. Solid lines are drawn using calls to DASHD, FRSTD, and VECTD. The parameters DD and MV also affect the behavior of MAPIT. For more information on these parameters, see the descriptions in module "Mp 3.5 Drawing political and geographic outlines."

A sequence of calls to MAPIT should be followed by a call to MAPIQ to flush its buffers before a STOP, a CALL FRAME, or a call to change the color index. Points in two contiguous pen-down calls to MAPIT should not be far apart on the globe.

It doesn't matter whether you call MAPIT and MAPIQ before or after calling the map-drawing routines. However, MAPIT and MAPIQ must be called after Ezmap is initialized. In line 1 of the **cmptra.f** code segment, the call to MAPDRW initializes Ezmap.

Line 2 tells MAPIT to start drawing at the lower left corner of Colorado. Lines 3 through 6 draw the four sides of the state, and line 7 quits drawing.

Another application of MAPIT appears in the next module.

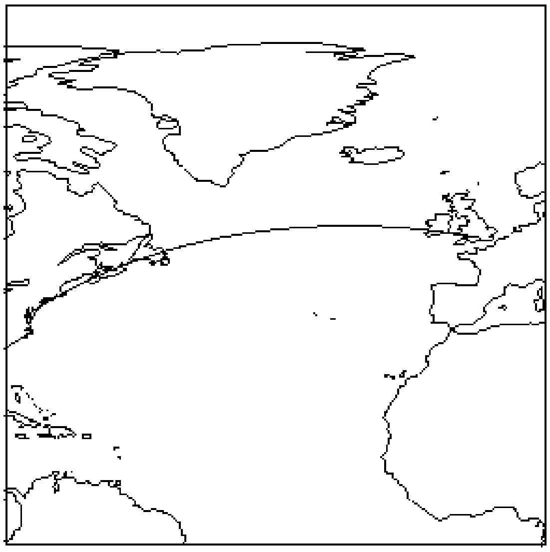
Exercises

1. Using the **cmpsup** example, draw an X across the face of the globe.

Mp 5.4 Drawing a great circle between two points

The routine MAPGCI returns a set of points defining a great circle between two user-specified points. Great circles describe the shortest distance between two points over the surface of the globe. MAPGCI can be handy when doing things like trying to figure out the shortest route between Washington D.C. and London, for example.

Drawing a great circle between Washington D.C. and London



Code segment from cmpgci.f

```
      CALL SUPMAP (8, 0., -50., 0., 0., -80., 90., 10., 2, 0., 0, 0, IERR)
      CALL MAPGCI (38., -77., 51., 0., 100, RLAT, RLON)
      CALL MAPIT (38., -77., 0)
      DO 10, I=1, 100
             CALL MAPIT (RLAT(I), RLON(I), 1)
  10  CONTINUE
      CALL MAPIT (51., 0., 1)
      CALL MAPIQ
```

Synopsis

```
CALL MAPGCI (ALAT, ALON, BLAT, BLON, NOPI, RLTI, RLNI)
```

Arguments

ALAT, ALON

Real, Input---The latitude and longitude of the first point to define a great circle.

BLAT, BLON

Real, Input---The latitude and longitude of the second point to define a great circle.

NOPI

Integer, Input---The number of points to be interpolated.

RLTI(NOPI), RLNI(NOPI)

Real arrays, Output---Arrays containing the latitudes and longitudes of each point interpolated along the great circle, excluding the points (ALAT, ALON) and (BLAT, BLON).

Discussion

Line 1 of the **cmpgci.f** code segment draws a cylindrical equidistant projection of the globe, showing the Atlantic and some portion of the land on either side. Line 2 gives MAPGCI the coordinates for Washington D.C. and London. Line 3 tells MAPIT that the first point is Washington D.C. Lines 4 through 6 draw line segments along the great circle, and line 7 draws the last segment to London. Line 8 quits drawing.

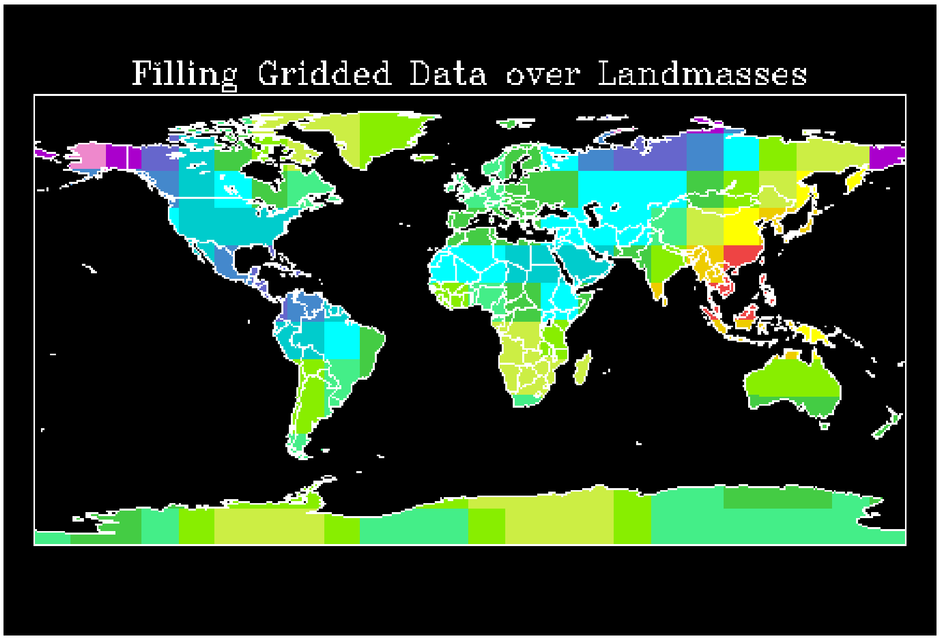
Exercises

1. Given that Lhasa, Tibet is at (29., 91.), and Boulder, Colorado, USA is at (40., -105.15), draw a great circle between the two.

Mp 5.5 Adding lines to an area map

Just as we used MAPIT and MAPIQ to draw lines over a geographic map, we can use MAPITA and MAPIQA to add lines to an area map. MAPITA and MAPIQA do not draw lines, they simply add them to the area map, so if you want to draw the lines as well as see their effects, you must use both MAPIT with MAPIQ and MAPITA with MAPIQA.

Color array over land masses



Code segment 1 from cmpita.f

```
      EXTERNAL FILL
      COMMON /DAT1/ ZDAT(M,N)
      CALL ARINAM (MAP, IMAP)
      CALL MAPINT
```

```
      CALL MAPBLA (MAP)
      DO 1, I = -90, 90, IGRD
         DO 2, J = -180, 180 - IGRD, IGRD
            LEFT = (J + 181) * 1000 + (I + 91)
            CALL MAPITA (REAL(I), REAL(J),      0, MAP, 5, LEFT, 0)
            CALL MAPITA (REAL(I), REAL(J+IGRD), 1, MAP, 5, LEFT, 0)
            CALL MAPIQA (MAP, 5, LEFT, 0)
    2    CONTINUE
    1 CONTINUE
      DO 3, I = -180, 180, IGRD
         DO 4, J = -90, 90 - IGRD, IGRD
            CALL MAPITA (REAL(J),      REAL(I), 0, MAP, 5, 0, 0)
            CALL MAPITA (REAL(J+IGRD), REAL(I), 1, MAP, 5, 0, 0)
            CALL MAPIQA (MAP, 5, 0, 0)
    4    CONTINUE
    3 CONTINUE
      CALL GSFAIS(1)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, 5, FILL)
```

Synopsis

```
      CALL MAPITA (RLAT, RLON, IFST, MAP, IDGRP, IDLEFT, IDRIT)
      CALL MAPIQA (MAP, IDGRP, IDLEFT, IDRIT)
```

Arguments

RLAT

Real, Input---Latitude of a point along the line.

RLON

Real, Input---Longitude of a point along the line.

IFST

Integer, Input---A flag that is set to zero for the first point of a line and nonzero for all other points of the line.

MAP

Integer array, Workspace---An integer array containing an area map.

IDGRP

Integer, Input---Group identifier for the line.

IDLEFT

Integer, Input---Area identifier for the area to the left of the line.

IDRIT

Integer, Input---Area identifier for the area to the right of the line.

Discussion

Line 1 of the **cmpita.f** code segment declares FILL to be external so it can be passed into ARSCAM. Line 2 sets up a common block so that we can pass the data array into the color-fill routine, FILL. Lines 3 through 5 initialize Ezmap and Areas together. The two loops from lines 6 through 13 add the latitude lines to the area map in 15-degree segments. In each case, the area to the left of each segment is defined to have an area identifier based on its latitude and longitude.

Area identifiers used for an entire curve are not necessarily the ones used for each call to MAPITA or MAPIQA. Inside the two routines is a 100-point buffer. Whenever the buffer fills or MAPIQA is called, the buffer is dumped, and the current values of IDGRP, IDLEFT, and IDRIT are used. To set group and area identifiers, make sure to call MAPIQA after every call to MAPITA where you change group or area identifiers.

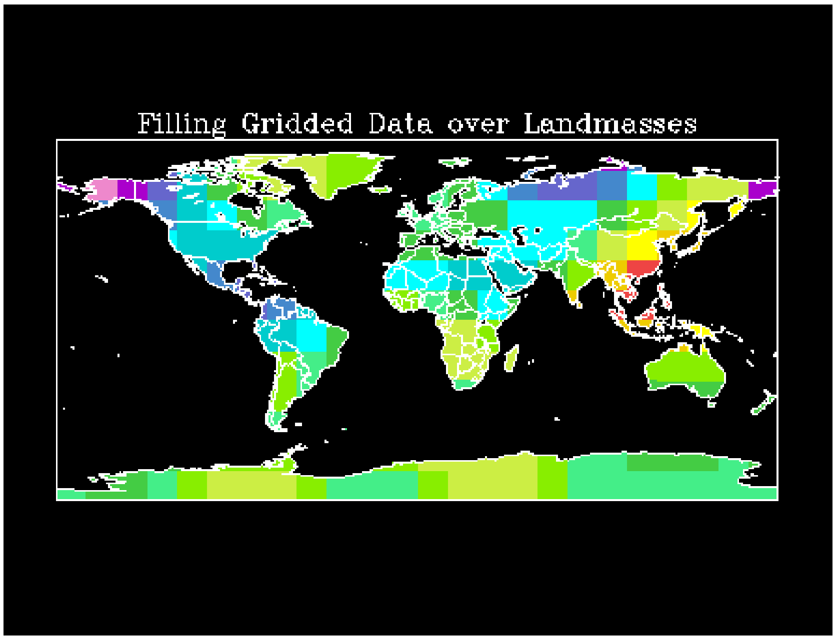
Lines 14 through 20 add the longitude lines to the area map, giving each area identifiers of zero. The effect of these four loops is to give each 15-degree square on the map a single nonzero area identifier. When the area map is processed, each 15-degree square is then assigned its only nonzero area identifier. This is a shortcut that could possibly leave zeros as area identifiers for squares that run into the limb of a globe projection. Line 21 sets color fill to solid, and each square is filled by calling ARSCAM in line 22.

Ideally, you would want to assign the same area identifier to all four lines that form each boxed area, and it may be necessary to do this with some projections. Line 22 calls ARSCAM to fill each square over land with a color based on a Z data value. The FILL routine is discussed on the next page.

Mp 5.5 Adding lines to an area map

This tutorial shows numerous examples of the FILL routine for ARSCAM. The example in this module differs only because it relies on a Z data array to choose a fill option for a specific area. The code segment below is a continuation of the **cmpita.f** code shown in part 1 of this module.

Color array over land masses



Code segment 2 from cmpita.f

```
30        SUBROUTINE FILL (XWRK, YWRK, NWRK, IAREA, IGRP, NSIZE)
31        COMMON /DAT1/ ZDAT(M,N)
32
33        IAREA1 = -1
34        IAREA5 = -1
35        IF (NWRK .LE. 3) RETURN
36        DO 10, I=1, NSIZE
37           IF (IGRP(I) .EQ. 1) IAREA1 = IAREA(I)
38           IF (IGRP(I) .EQ. 5) IAREA5 = IAREA(I)
39    10  CONTINUE
40        IF (IAREA1 .GT. 0) THEN
41           IF (MAPACI (IAREA1) .NE. 1 .AND. IAREA5 .GT. 0) THEN
42              LAT = MOD (IAREA5, 1000)
43              I = LAT / IGRD + 1
44              LON = IAREA5 / 1000
45              J = LON / IGRD + 1
46              ICLR = INT (ZDAT (I, J)) + 1
47              CALL GSFACI (ICLR)
48              CALL GFA (NWRK-1, XWRK, YWRK)
49           ENDIF
50        ENDIF
51        RETURN
52        END
```

Discussion

Line 31 of the **cmpita.f** code segment uses a common block to retrieve a Z data array so specific latitude and longitude areas can be filled with colors based on a given data value. Line 35 checks to see if an area is well-defined; in other words, if an area is defined by less than three points, we return to ARSCAM.

Lines 37 and 38 check our group identifiers and retrieve the area identifiers for areas defined by the geographic map and for areas defined by our two-degree grid over the globe. Line 40 checks to see if the area is over the geographic map. Line 41 checks to see if the area is over land (it has a color index not equal to 1), and if it has a nonzero grid area identifier. If the area is over land, then lines 42 and 44 retrieve the latitude and longitude values for the area, and lines 43 and 45 retrieve the Z array indices.

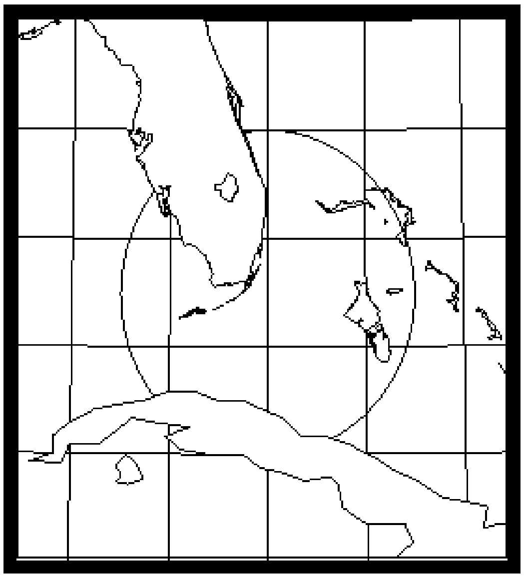
Line 46 sets the color index to be the Z data value plus 1 (since the data ranges between 1 and 15 and we want to use colors 2 through 16 in our color table). You could just as easily use a different function to translate Z data values into color here. Lines 47 through 49 fill the region with the desired color.

Notice how long this program takes to execute. It is much faster to create a cell array and use MAPTRA to overlay it on a globe. For an example of this, see module "Mp 5.2 Inverse transformations."

Mp 5.6 Drawing lines masked by an area map

You can draw lines and mask them over specific parts of the globe by using MAPITM and MAPIQM. Ezmap also uses these same routines to draw masked grid lines.

Masking a curve



Code segment from cmpitm.f

```
      EXTERNAL MASK1
      CALL CMPMSK ('ME', 0., -75., 0., 'PS', 'CO', PLIM1, PLIM2, PLIM3, PLIM4,
     + 2., MAP, LMAP)
      CALL CIRCLE (25., -80., 3.5, CLAT, CLON, NPTS)
      CALL MAPITM (CLAT(1), CLON(1), 0, MAP, XWRK, YWRK, NWRK,
     + IAREA, IGRP, NGRPS, MASK1)
      DO 102, ICIR=2, NPTS
            CALL MAPITM (CLAT(ICIR), CLON(ICIR), 1, MAP, XWRK, YWRK, NWRK,
     +    IAREA, IGRP, NGRPS, MASK1)
 102  CONTINUE
      CALL MAPIQM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS, MASK1)
```

Synopsis

```
 CALL MAPITM (RLAT, RLON, IFST, MAP, XWRK, YWRK, NWRK, IAREA, IGRP,
+             ISIZ, MASK)
 CALL MAPIQM (MAP, XWRK, YWRK, NWRK, IAREA, ISIZ, MASK)
```

Arguments

RLAT

Real, Input---Latitude of a point along the line.

RLON

Real, Input---Longitude of a point along the line.

IFST

Integer, Input---A flag that is set to zero for the first point of a line and nonzero for all other points of the line.

MAP

Integer array, Workspace---An area map that has been constructed by calls to ARINAM and MAPBLA.

XWRK(NWRK), YWRK(NWRK)

Real arrays, Workspace---Real arrays dimensioned MCS, to be used by MAPITM in calls to the user-supplied routine MASK.

NWRK

Integer, Input---Dimension of the arrays XCS and YCS. MCS must be greater than or equal to 2; the value 100 is suggested.

IAREA(ISIZ), IGRP(ISIZ)

Integer arrays, Workspace---Integer arrays to be used by MAPITM in calls to the user-supplied routine MASK.

ISIZ

Integer, Input---Dimension of the arrays IAREA and IGRP. ISIZ>=*n*, where *n* is the number of groups in the area map.

MASK

Subroutine---A user-supplied line-processing routine that is called once for each piece of the masked line. Each such piece is contained in exactly one area defined by the area map. MASK must appear in an EXTERNAL statement in the routine that calls MAPITM. Details on how to write a masking routine are discussed in module "Mp 4.6 Grid lines with masking: Writing a masking routine," and in module "Ar 3.3 Writing a masking or line-drawing routine."

Discussion

You must call MAPITM once for each point along the line. After the last call to MAPITM for a given line, you must call MAPIQM to signal the end of the line.

The projection of the line segment joining two points on the globe is considered to be the straight-line segment joining the projections of the points. No attempt is made to project it as if it were a portion of a great circle, so it is a good idea to place points defining each segment closely together.

- If both endpoints of a line segment are visible, the segment is considered to be entirely visible.
- If both endpoints are invisible, the segment is considered to be entirely invisible.
- If one endpoint is visible and the other is not, a new point is interpolated at the boundary between the visible and invisible portions.

Only the visible portions of the line are drawn in the area map.

There are two types of boundaries between visible and invisible regions:

- The limb is a boundary between a projectable region and an unprojectable one. The limb may be circular, elliptical, or some other shape, depending on the projection being used. For example, an orthographic projection has as its limb a circle, centered at (0., 0.), with a radius of 1.
- The perimeter is a rectangular or elliptical boundary that is defined by the Ezmap parameters you set to specify the region you want to view.

Line 1 declares MASK1 to be external so the compiler recognizes MASK1 as a subroutine and not an integer.

Line 2 of the **cmpitm.f** code segment calls CMPMSK to draw a map with grid lines masked over land. This code has been slightly modified from the code used in **cmpmsk.f**. In **cmpitm.f**, lines 4 through 8 use a single area map both for the grid line masking and for masking the circle that is drawn by using MAPITM and MAPIQM.

Line 3 calls a routine that generates a circle on any part of the globe. Because the globe is not flat, CIRCLE generates a circle of radius RADIUS---3.5 in this case---and rotates that circle to (RLAT, RLON)---25N, 80W in this case. CIRCLE returns CLAT and CLON, two arrays that contain NPTS latitude and longitude coordinates for the desired circle.

In line 4, MAPITM calls MASK1 to determine whether or not to draw the first point in the circle. In lines 5 through 7, MAPITM again calls MASK1 to draw line segments connecting each succeeding point in the circle if those segments are not over land. Line 8 calls MAPIQM to finish the circle.

Exercises

1. Using the **cmpitm** example, draw the grid lines only over land and the circle only over water.

Mp 6. Table of Ezmap area identifiers

Ezmap has four outline datasets that are identified by the index numbers and two-character names shown:

Number	Name	Contents
1	CO	Continental outlines only
2	US	US state outlines only
3	PS	Continental, US state, and international outlines
4	PO	Continental and interna tional outlines

This table shows the association between area identifiers and the names of the areas they identify. The first column gives the outline dataset name, the second column shows an area identifier, the third column shows a suggested color index for the area (to ensure that adjacent areas have different colors), and the name of the area appears in the fourth column.

This table is sorted by area name to help you find the area identifiers you need. A copy of this table is also available online via the **ncargfile** command.

Data set	Area id	Color index	Area name
PO	1264	3	Afghanistan
PS	906	3	Afghanistan
CO	3	2	Africa/Eurasia
CO	99	2	Africa/Eurasia
US	310	5	Alabama
PO	1014	3	Alaska
PS	450	3	Alaska
CO	5	2	Alaska---Atka Is.
PO	1008	3	Alaska---Atka Is.
PS	444	3	Alaska---Atka Is.
CO	218	2	Alaska---Attu Is.
PO	1356	3	Alaska---Attu Is.
PS	998	3	Alaska---Attu Is.
CO	20	2	Alaska---Kodiak Is.
PO	1023	3	Alaska---Kodiak Is.
PS	459	3	Alaska---Kodiak Is.
CO	13	2	Alaska---Nunivak Is.
PO	1016	3	Alaska---Nunivak Is.
PS	452	3	Alaska---Nunivak Is.
CO	9	2	Alaska---Saint Lawrence Is.
PO	1012	3	Alaska---Saint Lawrence Is.
PS	448	3	Alaska---Saint Lawrence Is.
CO	10	2	Alaska---Umnak Is.

PO	1013	3	Alaska---Umnak Is.
PS	449	3	Alaska---Umnak Is.
CO	12	2	Alaska---Unalaska Is.
PO	1015	3	Alaska---Unalaska Is.
PS	451	3	Alaska---Unalaska Is.
PO	1200	3	Albania
PS	842	3	Albania
PO	1144	2	Algeria
PS	786	2	Algeria
PO	1158	4	Andorra
PS	800	4	Andorra
PO	1188	4	Angola
PS	830	4	Angola
PO	1187	4	Angola---exclave called Cabinda
PS	829	4	Angola---exclave called Cabinda
CO	1	2	Antarctica
PO	1004	7	Antarctica
PS	440	7	Antarctica
CO	72	2	Antarctica---Alex ander Is.
PO	1088	7	Antarctica---Alex ander Is.
PS	666	7	Antarctica---Alex ander Is.
CO	83	2	Antarctica---Anvers Is.
PO	1106	7	Antarctica---Anvers Is.
PS	748	7	Antarctica---Anvers Is.
CO	36	2	Antarctica---Bear Is.
PO	1042	7	Antarctica---Bear Is.
PS	503	6	Antarctica---Bear Is.
CO	85	2	Antarctica---Brabant Is.
PO	1108	7	Antarctica---Brabant Is.
PS	750	7	Antarctica---Brabant Is.
CO	73	2	Antarctica---Charcot Is.
PO	1089	7	Antarctica---Charcot Is.
PS	669	7	Antarctica---Charcot Is.
CO	95	2	Antarctica---Corona tion Is.
PO	1123	7	Antarctica---Corona tion Is.
PS	765	7	Antarctica---Corona tion Is.
CO	21	2	Antarctica---Guest Is.
PO	1024	7	Antarctica---Guest Is.
PS	460	7	Antarctica---Guest Is.
CO	94	2	Antarctica---Joinvile Is.
PO	1121	7	Antarctica---Joinvile Is.
PS	763	7	Antarctica---Joinvile Is.
CO	92	2	Antarctica---King George Is.

PO	1117	7	Antarctica---King George Is.
PS	759	7	Antarctica---King George Is.
CO	91	2	Antarctica---Living ston Is.
PO	1116	7	Antarctica---Living ston Is.
PS	758	7	Antarctica---Living ston Is.
CO	14	2	Antarctica---Roosevelt Is.
PO	1017	7	Antarctica---Roosevelt Is.
PS	453	7	Antarctica---Roosevelt Is.
CO	213	2	Antarctica---Ross Is.
PO	1351	7	Antarctica---Ross Is.
PS	993	7	Antarctica---Ross Is.
PO	1098	3	Argentina
PS	713	3	Argentina
PO	1103	3	Argentina---Tierra del Fuego
PS	739	3	Argentina---Tierra del Fuego
US	245	2	Arizona
US	270	5	Arkansas
CO	163	2	Australia
PO	1297	3	Australia
PS	939	3	Australia
CO	193	2	Australia---Kangaroo Is.
PO	1330	3	Australia---Kangaroo Is.
PS	972	3	Australia---Kangaroo Is.
CO	185	2	Australia---Melville and Bathurst Is.
PO	1322	3	Australia---Melville and Bathurst Is.
PS	964	3	Australia---Melville and Bathurst Is.
CO	200	2	Australia---Tasmania
PO	1338	3	Australia---Tasmania
PS	980	3	Australia---Tasmania
PO	1178	3	Austria
PS	820	3	Austria
CO	67	2	Bahamas
CO	68	2	Bahamas
CO	70	2	Bahamas
PO	1083	6	Bahamas
PO	1084	6	Bahamas
PO	1086	6	Bahamas
US	360	4	Bahamas---Andros
US	362	4	Bahamas---Andros
US	363	4	Bahamas---Andros
US	365	4	Bahamas---Andros
US	368	4	Bahamas---Andros
US	394	4	Bahamas---Cat Is.
US	372	4	Bahamas---Eleuthera Is.

US	373	4	Bahamas---Eleuthera Is.
US	374	4	Bahamas---Eleuthera Is.
US	359	4	Bahamas---Grand Bahama
US	366	4	Bahamas---Grand Bahama
US	367	4	Bahamas---Grand Bahama
US	364	4	Bahamas---Great Abaco and Little Abaco
US	369	4	Bahamas---Mores Is.
US	370	4	Bahamas---New Provi dence Is.
PO	1276	4	Bangladesh
PS	918	4	Bangladesh
PO	1160	6	Belgium
PS	802	6	Belgium
PO	1064	2	Belize
PS	582	2	Belize
PO	1157	3	Benin
PS	799	3	Benin
PO	1277	3	Bhutan
PS	919	3	Bhutan
CO	203	2	Bismarck Archipel ago---New Britain
PO	1341	3	Bismarck Archipel ago---New Britain
PS	983	3	Bismarck Archipel ago---New Britain
CO	205	2	Bismarck Archipel ago---New Hanover
PO	1343	3	Bismarck Archipel ago---New Hanover
PS	985	3	Bismarck Archipel ago---New Hanover
CO	206	2	Bismarck Archipel ago---New Ireland
PO	1344	3	Bismarck Archipel ago---New Ireland
PS	986	3	Bismarck Archipel ago---New Ireland
CO	123	1	Black Sea
PO	1217	1	Black Sea
PS	859	1	Black Sea
PO	1101	4	Bolivia
PS	734	4	Bolivia
US	223	7	Border
CO	160	2	Borneo
PO	1298	3	Borneo---Brunei
PS	940	3	Borneo---Brunei
PO	1203	6	Botswana
PS	845	6	Botswana
PO	1094	6	Brazil
PS	705	6	Brazil
PO	1211	5	Bulgaria
PS	853	5	Bulgaria
PO	1279	3	Burma
PS	921	3	Burma

PO	1221	4	Burundi
PS	863	4	Burundi
US	226	4	California
US	240	4	California---San Clem ente Is.
US	234	4	California---San Miguel Is.(island)
US	239	4	California---San Nicholas Is.
US	241	4	California---Santa Cat alina Is.
US	238	4	California---Santa Cruz Is.
US	235	4	California---Santa Rosa Is.
PO	1286	2	Cambodia
PS	928	2	Cambodia
PO	1171	3	Cameroon
PS	813	3	Cameroon
PO	1026	6	Canada
PS	462	6	Canada
CO	62	2	Canada---Akimiski Is.
PO	1075	6	Canada---Akimiski Is.
PS	624	6	Canada---Akimiski Is.
CO	82	2	Canada---Anticosti Is.
PO	1105	6	Canada---Anticosti Is.
PS	747	6	Canada---Anticosti Is.
CO	50	2	Canada---Axel Heiberg Is.
PO	1056	6	Canada---Axel Heiberg Is.
PS	535	6	Canada---Axel Heiberg Is.
CO	54	2	Canada---Baffin Is.
PO	1062	6	Canada---Baffin Is.
PS	568	6	Canada---Baffin Is.
CO	25	2	Canada---Banks Is.
PO	1029	6	Canada---Banks Is.
PS	465	6	Canada---Banks Is.
CO	39	2	Canada---Bathurst Is.
PO	1045	6	Canada---Bathurst Is.
PS	509	6	Canada---Bathurst Is.
CO	65	2	Canada---Belcher Is.
PO	1080	6	Canada---Belcher Is.
PS	643	6	Canada---Belcher Is.
CO	33	2	Canada---Borden Is.
PO	1039	6	Canada---Borden Is.
PS	498	6	Canada---Borden Is.
CO	32	2	Canada---Brock Is.
PO	1038	6	Canada---Brock Is.
PS	493	6	Canada---Brock Is.
CO	40	2	Canada---Byam Martin Is.
PO	1046	6	Canada---Byam Martin Is.
PS	510	6	Canada---Byam Martin Is.
CO	63	2	Canada---Bylot Is.
PO	1078	6	Canada---Bylot Is.
PS	634	6	Canada---Bylot Is.
CO	58	2	Canada---Coats Is.

PO	1070	6	Canada---Coats Is.
PS	610	6	Canada---Coats Is.
CO	47	2	Canada---Cornwall Is.
PO	1053	6	Canada---Cornwall Is.
PS	525	6	Canada---Cornwall Is.
CO	48	2	Canada---Cornwallis Is.
PO	1054	6	Canada---Cornwallis Is.
PS	528	6	Canada---Cornwallis Is.
CO	49	2	Canada---Devon Is.
PO	1055	6	Canada---Devon Is.
PS	529	6	Canada---Devon Is.
CO	28	2	Canada---Eglinton Is.
PO	1033	6	Canada---Eglinton Is.
PS	484	6	Canada---Eglinton Is.
CO	37	2	Canada---Ellef Ringnes and Amund Ringnes Is.
PO	1043	6	Canada---Ellef Ringnes and Amund Ringnes Is.
PS	507	6	Canada---Ellef Ringnes and Amund Ringnes Is.
CO	53	2	Canada---Ellesmere Is.
PO	1060	6	Canada---Ellesmere Is.
PS	562	6	Canada---Ellesmere Is.
CO	26	1	Canada---Great Bear Lake
PO	1030	1	Canada---Great Bear Lake
PS	466	1	Canada---Great Bear Lake
CO	31	1	Canada---Great Slave Lake
PO	1036	1	Canada---Great Slave Lake
PS	490	1	Canada---Great Slave Lake
PS	716	1	Canada---lake
US	415	1	Canada---lake
CO	35	1	Canada---Lake Atha basca
PO	1041	1	Canada---Lake Atha basca
PS	501	1	Canada---Lake Atha basca
CO	44	1	Canada---Lake Mani toba
PO	1050	1	Canada---Lake Mani toba
PS	519	1	Canada---Lake Mani toba
CO	46	1	Canada---Lake Win nipeg
PO	1052	1	Canada---Lake Win nipeg
PS	521	1	Canada---Lake Win nipeg

CO	43	1	Canada---Lake Win nipegosis
PO	1049	1	Canada---Lake Win nipegosis
PS	518	1	Canada---Lake Win nipegosis
CO	38	2	Canada---Lougheed Is.
PO	1044	6	Canada---Lougheed Is.
PS	508	6	Canada---Lougheed Is.
CO	34	2	Canada---Mackenzie King Is.
PO	1040	6	Canada---Mackenzie King Is.
PS	499	6	Canada---Mackenzie King Is.
CO	60	2	Canada---Manitoulin Is.
PO	1072	6	Canada---Manitoulin Is.
CO	64	2	Canada---Mansel Is.
PO	1079	6	Canada---Mansel Is.
PS	642	6	Canada---Mansel Is.
CO	45	2	Canada---Meighen Is.
PO	1051	6	Canada---Meighen Is.
PS	520	6	Canada---Meighen Is.
CO	30	2	Canada---Melville Is.
PO	1035	6	Canada---Melville Is.
PS	488	6	Canada---Melville Is.
PO	1118	6	Canada---Newfound land
PS	760	6	Canada---Newfound land
CO	71	2	Canada---Prince Charles Is.
PO	1087	6	Canada---Prince Charles Is.
PS	661	6	Canada---Prince Charles Is.
CO	84	2	Canada---Prince Edward Is.
PO	1107	6	Canada---Prince Edward Is.
PS	749	6	Canada---Prince Edward Is.
CO	42	2	Canada---Prince of Wales Is.
PO	1048	6	Canada---Prince of Wales Is.
PS	516	6	Canada---Prince of Wales Is.
CO	27	2	Canada---Prince Patrick Is.
PO	1032	6	Canada---Prince Patrick Is.
PS	470	6	Canada---Prince Patrick Is.
CO	23	2	Canada---Queen Char lotte Is.
PO	1027	6	Canada---Queen Char lotte Is.
PS	463	6	Canada---Queen Char

			lotte Is.
CO	41	1	Canada---Reindeer Lake
PO	1047	1	Canada---Reindeer Lake
PS	514	1	Canada---Reindeer Lake
CO	56	2	Canada---Southamp ton Is.
PO	1067	6	Canada---Southamp ton Is.
PS	590	6	Canada---Southamp ton Is.
CO	24	2	Canada---Vancouver Is.
PO	1028	6	Canada---Vancouver Is.
PS	464	6	Canada---Vancouver Is.
CO	29	2	Canada---Victoria Is.
PO	1034	6	Canada---Victoria Is.
PS	485	6	Canada---Victoria Is.
CO	100	2	Canary Is.
CO	101	2	Canary Is.
CO	102	2	Canary Is.
CO	103	2	Canary Is.
PO	1130	2	Canary Is.
PO	1133	2	Canary Is.
PO	1135	2	Canary Is.
PO	1136	2	Canary Is.
PS	772	2	Canary Is.
PS	775	2	Canary Is.
PS	777	2	Canary Is.
PS	778	2	Canary Is.
CO	134	1	Caspian Sea
PO	1249	1	Caspian Sea
PS	891	1	Caspian Sea
PO	1196	5	Central African Republic
PS	838	5	Central African Republic
PO	1194	2	Chad
PS	836	2	Chad
PO	1090	2	Chile
PS	693	2	Chile
PS	702	2	Chile---Is.la de Chiloe
PO	1093	2	Chile---Isla de Chiloe
PO	1091	2	Chile---Tierra del Fuego
PS	699	2	Chile---Tierra del Fuego
PO	1270	5	China
PS	912	5	China
CO	159	2	China---Hainan Dao
PO	1292	5	China---Hainan Dao (island)
PS	934	5	China---Hainan Dao (island)
US	339	1	Clark Hill Reservoir
PO	1082	2	Colombia
PS	647	2	Colombia

US	251	6	Colorado
PO	1185	2	Congo
PS	827	2	Congo
US	409	5	Connecticut
CO	111	2	Corsica
PO	1173	3	Corsica
PS	815	3	Corsica
PO	1068	2	Costa Rica
PS	596	2	Costa Rica
CO	122	2	Crete
PO	1212	2	Crete
PS	854	2	Crete
CO	57	2	Cuba
PO	1069	4	Cuba
PS	602	4	Cuba
CO	61	2	Cuba---Isla de Pinos
PO	1073	4	Cuba---Isla de Pinos
PS	616	4	Cuba---Isla de Pinos
CO	80	2	Curacao
PO	1102	3	Curacao
PS	735	3	Curacao
CO	128	2	Cyprus
PO	1228	3	Cyprus
PS	870	3	Cyprus
PO	1189	2	Czechoslovakia
PS	831	2	Czechoslovakia
US	393	5	Delaware
US	402	1	Delaware---bay
CO	112	2	Denmark---Fyn
PO	1179	2	Denmark---Fyn
PS	821	2	Denmark---Fyn
PO	1169	2	Denmark---Jutland
PS	811	2	Denmark---Jutland
CO	113	2	Denmark---Lolland
PO	1181	2	Denmark---Lolland
PS	823	2	Denmark---Lolland
CO	115	2	Denmark---Sjaelland
PO	1183	2	Denmark---Sjaelland
PS	825	2	Denmark---Sjaelland
PO	1272	4	Disputed area between India and China
PS	914	4	Disputed area between India and China
PO	1242	3	Djibouti
PS	884	3	Djibouti
PO	1100	3	Dominican Republic
PS	724	3	Dominican Republic
PO	1180	3	East Germany
PS	822	3	East Germany
PO	1077	3	Ecuador
PS	632	3	Ecuador
PO	1213	6	Egypt
PS	855	6	Egypt
PO	1061	4	El Salvador
PS	567	4	El Salvador
PO	1150	5	England
PS	792	5	England
PO	1175	6	Equatorial Guinea
PS	817	6	Equatorial Guinea
PO	1230	4	Ethiopia
PS	872	4	Ethiopia

CO	105	2	Faeroe Is.
PO	1147	2	Faeroe Is.
PS	789	2	Faeroe Is.
CO	89	2	Faulkland Is.
PO	1114	5	Faulkland Is.
PS	756	5	Faulkland Is.
CO	222	2	Fiji---Vanua Levu
PO	1360	3	Fiji---Vanua Levu
PS	1002	3	Fiji---Vanua Levu
CO	221	2	Fiji---Viti Levu
PO	1359	3	Fiji---Viti Levu
PS	1001	3	Fiji---Viti Levu
PO	1205	3	Finland
PS	847	3	Finland
US	313	4	Florida
US	322	4	Florida---island
US	348	4	Florida---island
US	349	4	Florida---island
US	350	4	Florida---island
US	351	4	Florida---island
US	353	4	Florida---island
US	343	4	Florida---Keys
US	344	4	Florida---Keys
US	345	4	Florida---Keys
US	346	4	Florida---Keys
US	319	1	Florida---lake
US	347	1	Florida---Lake Okeechobee
US	341	4	Florida---Pine Is.
US	340	4	Florida---Sanibel Is.
PO	1153	3	France
PS	795	3	France
PO	1122	3	French Guiana
PS	764	3	French Guiana
PO	1174	5	Gabon
PS	816	5	Gabon
CO	52	2	Galapagos Is.
PO	1059	3	Galapagos Is.
PS	553	3	Galapagos Is.
PO	1132	2	Gambia
PS	774	2	Gambia
US	320	6	Georgia
PO	1154	5	Ghana
PS	796	5	Ghana
CO	118	2	Gotland
PO	1199	2	Gotland
PS	841	2	Gotland
US	248	1	Great Salt Lake
CO	119	2	Greece
PO	1202	2	Greece
PO	1206	2	Greece
PS	844	2	Greece
PS	848	2	Greece
CO	79	2	Green
PO	1099	2	Greenland
PS	718	2	Greenland
CO	98	2	Greenland---off-shore island
PO	1126	2	Greenland---off-shore island
PS	768	2	Greenland---off-shore

			island
CO	88	2	Guadeloupe---east
PO	1112	3	Guadeloupe---east
PS	754	3	Guadeloupe---east
CO	87	2	Guadeloupe---west
PO	1111	3	Guadeloupe---west
PS	753	3	Guadeloupe---west
PO	1057	3	Guatemala
PS	549	3	Guatemala
PO	1134	5	Guinea
PS	776	5	Guinea
PO	1131	4	Guinea-Bissau
PS	773	4	Guinea-Bissau
PO	1113	4	Guyana
PS	755	4	Guyana
PO	1092	6	Haiti
PS	700	6	Haiti
CO	75	2	Haiti and the Domini can Republic
CO	19	2	Hawaiian Is.---Hawaii
PO	1022	3	Hawaiian Is.---Hawaii
PS	458	3	Hawaiian Is.---Hawaii
CO	16	2	Hawaiian Is.---Kauai
PO	1019	3	Hawaiian Is.---Kauai
PS	455	3	Hawaiian Is.---Kauai
CO	18	2	Hawaiian Is.---Maui
PO	1021	3	Hawaiian Is.---Maui
PS	457	3	Hawaiian Is.---Maui
CO	15	2	Hawaiian Is.---Niihau
PO	1018	3	Hawaiian Is.---Niihau
PS	454	3	Hawaiian Is.---Niihau
CO	17	2	Hawaiian Is.---Oahu
PO	1020	3	Hawaiian Is.---Oahu
PS	456	3	Hawaiian Is.---Oahu
PO	1063	5	Honduras
PS	572	5	Honduras
PO	1197	5	Hungary
PS	839	5	Hungary
CO	97	2	Iceland
PO	1125	7	Iceland
PS	767	7	Iceland
US	242	6	Idaho
CO	146	2	Ile Kerguelen
PO	1268	3	Ile Kerguelen
PS	910	3	Ile Kerguelen
US	284	4	Illinois
PO	1267	2	India
PS	909	2	India
US	311	5	Indiana
CO	177	2	Indonesia---Alor
PO	1312	5	Indonesia---Alor
PS	954	5	Indonesia---Alor
CO	164	2	Indonesia---Bali
PO	1299	5	Indonesia---Bali
PS	941	5	Indonesia---Bali
CO	156	2	Indonesia---Bangka
PO	1289	5	Indonesia---Bangka
PS	931	5	Indonesia---Bangka
CO	162	2	Indonesia---Bawean
PO	1296	5	Indonesia---Bawean
PS	938	5	Indonesia---Bawean

CO	158	2	Indonesia---Belitung
PO	1291	5	Indonesia---Belitung
PS	933	5	Indonesia---Belitung
CO	192	2	Indonesia---Biak and Kepulauan Schouten
PO	1329	5	Indonesia---Biak and Kepulauan Schouten
PS	971	5	Indonesia---Biak and Kepulauan Schouten
PO	1293	5	Indonesia---Borneo---Kalimantan
PS	935	5	Indonesia---Borneo---Kalimantan
CO	180	2	Indonesia---Buru
PO	1316	5	Indonesia---Buru
PS	958	5	Indonesia---Buru
CO	183	2	Indonesia---Ceram
PO	1320	5	Indonesia---Ceram
PS	962	5	Indonesia---Ceram
CO	169	2	Indonesia---Flores
PO	1304	5	Indonesia---Flores
PS	946	5	Indonesia---Flores
CO	182	2	Indonesia---Halmahera
PO	1319	5	Indonesia---Halmahera
PS	961	5	Indonesia---Halmahera
CO	175	2	Indonesia---island
PO	1310	5	Indonesia---island
PS	952	5	Indonesia---island
CO	157	2	Indonesia---Java
PO	1290	5	Indonesia---Java
PS	932	5	Indonesia---Java
CO	191	2	Indonesia---Kepu lauan Aru
PO	1328	5	Indonesia---Kepu lauan Aru
PS	970	5	Indonesia---Kepu lauan Aru
CO	189	2	Indonesia---Kepu lauan Tanimbar
PO	1326	5	Indonesia---Kepu lauan Tanimbar
PS	968	5	Indonesia---Kepu lauan Tanimbar
PO	1324	5	Indonesia---New Guinea---Irian Jaya
PS	966	5	Indonesia---New Guinea---Irian Jaya
CO	152	2	Indonesia---Nias
PO	1281	5	Indonesia---Nias
PS	923	5	Indonesia---Nias
CO	181	2	Indonesia---Obi
PO	1318	5	Indonesia---Obi
PS	960	5	Indonesia---Obi
CO	153	2	Indonesia---Siberut
PO	1283	5	Indonesia---Siberut
PS	925	5	Indonesia---Siberut
CO	167	2	Indonesia---Sulawesi (Celebes)
PO	1302	5	Indonesia---Sulawesi (Celebes)
PS	944	5	Indonesia---Sulawesi

			(Celebes)
CO	151	2	Indonesia---Sumatra
PO	1280	5	Indonesia---Sumatra
PS	922	5	Indonesia---Sumatra
CO	168	2	Indonesia---Sumba
PO	1303	5	Indonesia---Sumba
PS	945	5	Indonesia---Sumba
PS	942	5	Indonesia---Sumbawa & Lombok
CO	165	2	Indonesia---Sumbawa and Lombok
PO	1300	5	Indonesia---Sumbawa and Lombok
CO	176	2	Indonesia---Timor
PO	1311	5	Indonesia---Timor
PS	953	5	Indonesia---Timor
CO	186	2	Indonesia---Waigeo
PO	1323	5	Indonesia---Waigeo
PS	965	5	Indonesia---Waigeo
CO	179	2	Indonesia---Wetar
PO	1315	5	Indonesia---Wetar
PS	957	5	Indonesia---Wetar
US	262	6	Iowa
PO	1246	5	Iran
PS	888	5	Iran
PO	1240	6	Iraq
PS	882	6	Iraq
CO	104	2	Ireland
PO	1141	4	Ireland
PS	783	4	Ireland
CO	76	2	Isla de Chiloe
PO	1233	2	Israel
PS	875	2	Israel
PO	1167	6	Italy
PS	809	6	Italy
PO	1145	4	Ivory Coast
PS	787	4	Ivory Coast
CO	69	2	Jamaica
PO	1085	5	Jamaica
PS	650	5	Jamaica
CO	196	2	Japan---Hokkaido
PO	1333	3	Japan---Hokkaido
PS	975	3	Japan---Hokkaido
CO	188	2	Japan---Honshu
PO	1325	3	Japan---Honshu
PS	967	3	Japan---Honshu
CO	184	2	Japan---Kyushu
PO	1321	3	Japan---Kyushu
PS	963	3	Japan---Kyushu
CO	190	2	Japan---Shikoku
PO	1327	3	Japan---Shikoku
PS	969	3	Japan---Shikoku
PO	1237	5	Jordan
PS	879	5	Jordan
US	257	4	Kansas
US	299	6	Kentucky
PO	1231	5	Kenya
PS	873	5	Kenya
PO	1361	2	Kuwait
PS	1003	2	Kuwait
CO	117	1	Lake Chad

PO	1192	1	Lake Chad
PS	834	1	Lake Chad
US	411	1	Lake Champlain
US	413	2	Lake Champlain---Grand Isle
CO	59	1	Lake Erie
PO	1071	1	Lake Erie
US	333	1	Lake Erie
US	354	2	Lake Erie---Long Point (land)
US	329	1	Lake George
US	337	2	Lake Huron---Barrie Is.
US	325	2	Lake Huron---Bois Blanc Is.
US	355	2	Lake Huron---Chris tian Is.
US	332	2	Lake Huron---Cock burn Is.
US	330	2	Lake Huron---Drum mond Is.
US	342	2	Lake Huron---Fitzwill iam Is.
US	335	2	Lake Huron---Mani toulin Is.
US	246	1	Lake Mead (east)
US	244	1	Lake Mead (west)
US	321	2	Lake Michigan---Bea ver Is.
US	317	2	Lake Michigan---Man itou Is.
US	314	2	Lake Michigan---Washington Is.
CO	129	1	Lake Nyasa
PO	1232	1	Lake Nyasa
PS	874	1	Lake Nyasa
CO	66	1	Lake Ontario
PO	1081	1	Lake Ontario
US	357	1	Lake Ontario
US	371	2	Lake Ontario---island
US	375	2	Lake Ontario---island
US	377	2	Lake Ontario---island
US	323	1	Lake Seminole
US	336	1	Lake St. Clair
CO	51	1	Lake Superior
PO	1058	1	Lake Superior
US	281	1	Lake Superior
US	316	2	Lake Superior---Grand Is.
US	326	2	Lake Superior---island
US	307	2	Lake Superior---Isle Royale
US	286	2	Lake Superior---Made line Is.
US	318	2	Lake Superior---Mich ipicoten Is.
US	288	2	Lake Superior---Oak Is.
US	292	2	Lake Superior---Outer Is.
US	308	2	Lake Superior---Pie Is.

US	289	2	Lake Superior---Stock ton Is.
US	236	1	Lake Tahoe
CO	125	1	Lake Tanganyika
PO	1220	1	Lake Tanganyika
PS	862	1	Lake Tanganyika
US	261	1	Lake Texoma
CO	127	1	Lake Victoria
PO	1227	1	Lake Victoria
PS	869	1	Lake Victoria
CO	55	1	Lakes Michigan and Huron
PO	1065	1	Lakes Michigan and Huron
US	312	1	Lakes Michigan and Huron
PO	1285	6	Laos
PS	927	6	Laos
PO	1238	3	Lebanon
PS	880	3	Lebanon
PO	1216	6	Lesotho
PS	858	6	Lesotho
PO	1140	3	Liberia
PS	782	3	Liberia
PO	1176	4	Libya
PS	818	4	Libya
PO	1177	4	Liechtenstein
PS	819	4	Liechtenstein
CO	77	2	Long Is.
PO	1095	3	Long Is.
US	417	1	Long Is.---Great South Bay
US	272	4	Louisiana
US	275	1	Louisiana---Calcasieu Lake
US	309	4	Louisiana---Cat Is.
US	278	1	Louisiana---Grand Lake
US	285	4	Louisiana---island
US	287	4	Louisiana---island
US	296	4	Louisiana---island
US	297	4	Louisiana---island
US	298	4	Louisiana---island
US	300	4	Louisiana---island
US	301	4	Louisiana---island
US	302	4	Louisiana---island
US	303	4	Louisiana---island
US	304	4	Louisiana---island
US	305	4	Louisiana---island
US	306	4	Louisiana---island
US	290	1	Louisiana---Lake Mau repas
US	294	1	Louisiana---Lake Pont chartrain
US	282	4	Louisiana---Marsh Is.
US	273	1	Louisiana---Sabine Lake
US	291	4	Louisiana---Timbalier Is.
US	280	1	Louisiana---White Lake

PO	1164	5	Luxembourg
PS	806	5	Luxembourg
CO	110	2	"Macias Nguema Biyogo (African island, Fernando Po)"
PO	1172	3	"Macias Nguema Biyogo (African island, Fernando Po)"
PS	814	3	"Macias Nguema Biyogo (African island, Fernando Po)"
CO	132	2	Madagascar
PO	1244	3	Madagascar
PS	886	3	Madagascar
US	427	5	Maine
US	431	5	Maine---island
US	432	5	Maine---island
US	433	5	Maine---island
US	434	5	Maine---island
US	435	5	Maine---island
US	436	5	Maine---island
US	437	5	Maine---island
US	438	5	Maine---island
US	439	5	Maine---island
PO	1229	2	Malawi
PS	871	2	Malawi
PO	1294	6	Malaysia---Borneo---Malaysia Timor
PS	936	6	Malaysia---Borneo---Malaysia Timor
PO	1201	6	Malaysia---Malay Pen insula
PS	843	6	Malaysia---Malay Pen insula
PO	1139	6	Mali
PS	781	6	Mali
CO	108	2	Mallorca
PO	1159	2	Mallorca
PS	801	2	Mallorca
CO	90	2	Martinique
PO	1115	3	Martinique
PS	757	3	Martinique
US	358	4	Maryland
US	403	4	Maryland
US	404	4	Maryland
US	386	1	Maryland---inlet
US	376	4	Maryland---island
US	378	4	Maryland---island
US	380	4	Maryland---island
US	381	4	Maryland---island
US	410	4	Massachusetts
US	426	1	Massachusetts---bay
US	430	1	Massachusetts---inlet
US	428	4	Massachusetts---Mar thas Vineyard
US	429	4	Massachusetts---Nan tucket Is.
PO	1129	4	Mauritania
PS	771	4	Mauritania
CO	142	2	Mauritius
PO	1261	3	Mauritius

PS	903	3	Mauritius
PO	1037	6	Mexico
PS	491	6	Mexico
US	315	2	Michigan (lower)
US	293	2	Michigan (upper)
US	260	4	Minnesota
US	264	1	Minnesota---Lake of the Woods
US	268	4	Minnesota---Lake of the Woods---island
US	269	4	Minnesota---Lake of the Woods---island
US	271	4	Minnesota---Lake of the Woods---island
US	266	4	Minnesota---Lake of the Woods---land west of lake
US	279	4	Minnesota---little piece
US	276	1	Minnesota---Namakan Lake
US	274	1	Minnesota---Rainy Lake
US	265	1	Minnesota---Shoal Lake
US	283	2	Mississippi
US	263	2	Missouri
PO	1275	3	Mongolia
PS	917	3	Mongolia
US	243	5	Montana
PO	1138	5	Morocco
PS	780	5	Morocco
PO	1225	4	Mozambique
PS	867	4	Mozambique
PO	1186	2	Namibia
PS	828	2	Namibia
US	253	5	Nebraska
PO	1274	3	Nepal
PS	916	3	Nepal
PO	1162	3	Netherlands
PS	804	3	Netherlands
US	237	5	Nevada
CO	212	2	New Caledonia (island)
PO	1350	3	New Caledonia (island)
PS	992	3	New Caledonia (island)
CO	187	2	New Guinea
PO	1336	3	New Guinea---Papua
PS	978	3	New Guinea---Papua
US	418	6	New Hampshire
US	414	1	New Hampshire---lake
US	416	1	New Hampshire---lake
CO	215	2	New Hebrides---Espiritu Santo
PO	1353	3	New Hebrides---Espiritu Santo
PS	995	3	New Hebrides---Espiritu Santo
CO	216	2	New Hebrides---Male

			kula
PO	1354	3	New Hebrides---Male kula
PS	996	3	New Hebrides---Male kula
US	400	2	New Jersey
US	405	2	New Jersey---island
US	407	1	New Jersey---Newark Bay
US	250	5	New Mexico
US	356	3	New York
US	420	3	New York---Fishers Is.
US	408	3	New York---Long Is.
US	419	3	New York---Shelter Is.
CO	6	2	New Zealand---Chatham Is.
PO	1009	6	New Zealand---Chatham Is.
PS	445	6	New Zealand---Chatham Is.
CO	219	2	New Zealand---North Is.
CO	220	2	New Zealand---North Is.
PO	1357	6	New Zealand---North Is.
PO	1358	6	New Zealand---North Is.
PS	999	6	New Zealand---North Is.
PS	1000	6	New Zealand---North Is.
CO	214	2	New Zealand---South Is.
PO	1352	6	New Zealand---South Is.
PS	994	6	New Zealand---South Is.
CO	217	2	New Zealand---Stew art Is.
PO	1355	6	New Zealand---Stew art Is.
PS	997	6	New Zealand---Stew art Is.
CO	93	2	Newfoundland
PO	1066	6	Nicaragua
PS	588	6	Nicaragua
PO	1156	5	Niger
PS	798	5	Niger
PO	1161	4	Nigeria
PS	803	4	Nigeria
CO	11	2	North and South America
US	328	4	North Carolina
US	361	4	North Carolina---Cape Fear
US	383	4	North Carolina---island
US	384	4	North Carolina---island
US	388	4	North Carolina---

			island
US	391	4	North Carolina---island
US	392	4	North Carolina---island
US	395	4	North Carolina---island
US	396	4	North Carolina---island
US	397	4	North Carolina---island
US	398	4	North Carolina---island
US	399	4	North Carolina---island
US	255	3	North Dakota
PO	1314	2	North Korea
PS	956	2	North Korea
PO	1146	5	Northern Ireland
PS	788	5	Northern Ireland
PO	1163	4	Norway
PS	805	4	Norway
CO	2	1	Oceans
PO	1005	1	Oceans
PS	441	1	Oceans
US	324	4	Ohio
US	256	3	Oklahoma
PO	1254	4	Oman---northern section
PS	896	4	Oman---northern section
PO	1260	4	Oman---southern section
PS	902	4	Oman---southern section
US	225	3	Oregon
CO	106	2	Outer Hebrides
PO	1148	2	Outer Hebrides
PS	790	2	Outer Hebrides
PO	1265	4	Pakistan
PS	907	4	Pakistan
PO	1074	5	Panama
PS	617	5	Panama
PO	1109	5	Paraguay
PS	751	5	Paraguay
US	352	6	Pennsylvania
PO	1076	5	Peru
PS	630	5	Peru
CO	170	2	Philippines---Luzon
PO	1305	4	Philippines---Luzon
PS	947	4	Philippines---Luzon
CO	174	2	Philippines---Mindanao
PO	1309	4	Philippines---Mindanao
PS	951	4	Philippines---Mindanao
CO	172	2	Philippines---Mindoro
PO	1307	4	Philippines---Mindoro
PS	949	4	Philippines---Mindoro
CO	166	2	Philippines---Palawan

PO	1301	4	Philippines---Palawan
PS	943	4	Philippines---Palawan
CO	173	2	"Philippines---Panay, Negros, and Bohol"
PO	1308	4	"Philippines---Panay, Negros, and Bohol"
PS	950	4	"Philippines---Panay, Negros, and Bohol"
CO	178	2	Philippines---Samar
PO	1313	4	Philippines---Samar
PS	955	4	Philippines---Samar
PO	1195	5	Poland
PS	837	5	Poland
PO	1142	4	Portugal
PS	784	4	Portugal
CO	81	2	Puerto Rico
PO	1104	3	Puerto Rico
PS	746	3	Puerto Rico
PO	1252	4	Qatar
PS	894	4	Qatar
CO	140	2	Reunion Is.
PO	1258	3	Reunion Is.
PS	900	3	Reunion Is.
US	421	6	Rhode Is.
US	422	6	Rhode Is.---Block Is.
US	423	6	Rhode Is.---island
US	424	6	Rhode Is.---island
US	425	6	Rhode Is.---island
CO	124	2	Rhodes
PO	1218	2	Rhodes
PS	860	2	Rhodes
PO	1204	3	Romania
PS	846	3	Romania
PO	1219	5	Rwanda
PS	861	5	Rwanda
CO	7	2	Samoa---Savaii
PO	1010	3	Samoa---Savaii
PS	446	3	Samoa---Savaii
CO	8	2	Samoa---Upolu
PO	1011	3	Samoa---Upolu
PS	447	3	Samoa---Upolu
CO	109	2	Sardinia
PO	1170	6	Sardinia
PS	812	6	Sardinia
PO	1236	3	Saudi Arabia
PS	878	3	Saudi Arabia
US	327	1	Sault Ste. Marie
PO	1149	2	Scotland
PS	791	2	Scotland
CO	107	2	Scotland/England/Wal es
PO	1127	3	Senegal
PS	769	3	Senegal
CO	116	2	Sicily
PO	1191	6	Sicily
PS	833	6	Sicily
PO	1137	6	Sierra Leone
PS	779	6	Sierra Leone
CO	207	2	Solomon Is.---Bouga inville
PO	1345	3	Solomon Is.---Bouga

			inville
PS	987	3	Solomon Is.---Bouga inville
CO	209	2	Solomon Is.---Guadal canal
PO	1347	3	Solomon Is.---Guadal canal
PS	989	3	Solomon Is.---Guadal canal
CO	210	2	Solomon Is.---San Cristobal
PO	1348	3	Solomon Is.---San Cristobal
PS	990	3	Solomon Is.---San Cristobal
PO	1241	2	Somalia
PS	883	2	Somalia
PO	1198	5	South Africa
PS	840	5	South Africa
US	334	2	South Carolina
US	254	2	South Dakota
CO	96	2	South Georgia Is.
PO	1124	5	South Georgia Is.
PS	766	5	South Georgia Is.
PO	1317	4	South Korea
PS	959	4	South Korea
PO	1143	2	Spain
PS	785	2	Spain
CO	149	2	Sri Lanka (Ceylon)
PO	1273	5	Sri Lanka (Ceylon)
PS	915	5	Sri Lanka (Ceylon)
US	406	3	Staten Is.
PO	1208	3	Sudan
PS	850	3	Sudan
PO	1120	5	Surinam
PS	762	5	Surinam
CO	114	2	Svalbard
PO	1182	4	Svalbard
PS	824	4	Svalbard
PO	1226	6	Swaziland
PS	868	6	Swaziland
PO	1184	5	Sweden
PS	826	5	Sweden
PO	1165	5	Switzerland
PS	807	5	Switzerland
PO	1239	4	Syria
PS	881	4	Syria
CO	22	2	Tahiti
PO	1025	3	Tahiti
PS	461	3	Tahiti
CO	171	2	Taiwan
PO	1306	2	Taiwan
PS	948	2	Taiwan
PO	1223	3	Tanzania
PS	865	3	Tanzania
US	295	3	Tennessee
US	252	2	Texas
US	258	2	Texas---island
US	267	2	Texas---island
US	259	2	Texas---Padre Is.
PO	1282	5	Thailand

PS	924	5	Thailand
CO	74	2	Tierra del Fuego
PO	1155	6	Togo
PS	797	6	Togo
CO	86	2	Trinidad and Tobago
PO	1110	6	Trinidad and Tobago
PS	752	6	Trinidad and Tobago
PO	1168	5	Tunisia
PS	810	5	Tunisia
PO	1215	3	Turkey
PS	857	3	Turkey
CO	78	2	Turks and Caicos Is.
PO	1096	4	Turks and Caicos Is.
PS	708	4	Turks and Caicos Is.
PO	1222	2	Uganda
PS	864	2	Uganda
PO	1251	2	United Arab Emirates
PS	893	2	United Arab Emirates
PO	1248	4	Unnamed neutral zone between Iraq and Saudi Arabia
PS	890	4	Unnamed neutral zone between Iraq and Saudi Arabia
PO	1151	2	Upper Volta
PS	793	2	Upper Volta
PO	1119	2	Uruguay
PS	761	2	Uruguay
PO	1031	3	US
PO	1006	6	USSR
PS	442	6	USSR
CO	143	1	USSR---Aral Sea
PO	1262	1	USSR---Aral Sea
PS	904	1	USSR---Aral Sea
CO	130	1	USSR---Azov Sea
PO	1234	1	USSR---Azov Sea
PS	876	1	USSR---Azov Sea
CO	133	2	USSR---Franz Josef Land
CO	137	2	USSR---Franz Josef Land
CO	139	2	USSR---Franz Josef Land
CO	141	2	USSR---Franz Josef Land
CO	144	2	USSR---Franz Josef Land
CO	145	2	USSR---Franz Josef Land
PO	1247	6	USSR---Franz Josef Land
PO	1255	6	USSR---Franz Josef Land
PO	1257	6	USSR---Franz Josef Land
PO	1259	6	USSR---Franz Josef Land
PO	1263	6	USSR---Franz Josef Land
PO	1266	6	USSR---Franz Josef Land

PS	889	6	USSR---Franz Josef Land
PS	897	6	USSR---Franz Josef Land
PS	899	6	USSR---Franz Josef Land
PS	901	6	USSR---Franz Josef Land
PS	905	6	USSR---Franz Josef Land
PS	908	6	USSR---Franz Josef Land
CO	155	1	USSR---Lake Baikal
PO	1288	1	USSR---Lake Baikal
PS	930	1	USSR---Lake Baikal
CO	148	1	USSR---Lake Balkhash (east)
PO	1271	1	USSR---Lake Balkhash (east)
PS	913	1	USSR---Lake Balkhash (east)
CO	147	1	USSR---Lake Balkhash (west)
PO	1269	1	USSR---Lake Balkhash (west)
PS	911	1	USSR---Lake Balkhash (west)
CO	136	2	USSR---Novaya Zem lya
PO	1253	6	USSR---Novaya Zem lya
PS	895	6	USSR---Novaya Zem lya
CO	131	1	USSR---Onazhskoye Ozero
PO	1235	1	USSR---Onazhskoye Ozero
PS	877	1	USSR---Onazhskoye Ozero
CO	154	2	USSR---Ostrov Bol’ shevik
PO	1284	6	USSR---Ostrov Bol’ shevik
PS	926	6	USSR---Ostrov Bol’ shevik
CO	161	2	USSR---Ostrov Bol’shoy Begichev
PO	1295	6	USSR---Ostrov Bol’shoy Begichev
PS	937	6	USSR---Ostrov Bol’shoy Begichev
CO	198	2	USSR---Ostrov B ol’shoy Lyakhovskiy
PO	1335	6	USSR---Ostrov Bol’shoy Lyakhovskiy
PS	977	6	USSR---Ostrov Bol’shoy Lyakhovskiy
CO	121	2	USSR---Ostrov Hiiu maa
PO	1210	6	USSR---Ostrov Hiiu maa

PS	852	6	USSR---Ostrov Hiiu maa
CO	202	2	USSR---Ostrov Iturup
PO	1340	6	USSR---Ostrov Iturup
PS	982	6	USSR---Ostrov Iturup
CO	211	2	USSR---Ostrov Kara ginskiy
PO	1349	6	USSR---Ostrov Kara ginskiy
PS	991	6	USSR---Ostrov Kara ginskiy
CO	135	2	USSR---Ostrov Kolguyev
PO	1250	6	USSR---Ostrov Kolguyev
PS	892	6	USSR---Ostrov Kolguyev
CO	194	2	USSR---Ostrov Kotel’ nyy and Ostrov Fad deyevskiy
PO	1331	6	USSR---Ostrov Kotel’ nyy and Ostrov Fad deyevskiy
PS	973	6	USSR---Ostrov Kotel’ nyy and Ostrov Fad deyevskiy
CO	197	2	USSR---Ostrov Malyy Lyakhovskiy
PO	1334	6	USSR---Ostrov Malyy Lyakhovskiy
PS	976	6	USSR---Ostrov Malyy Lyakhovskiy
CO	201	2	USSR---Ostrov Novaya Sibir’
PO	1339	6	USSR---Ostrov Novaya Sibir’
PS	981	6	USSR---Ostrov Novaya Sibir’
CO	150	2	"USSR---Ostrov Okty abr’skoy Revolyutsii, Komsomolets, Pioner"
PO	1278	6	"USSR---Ostrov Okty abr’skoy Revolyutsii, Komsomolets, Pioner"
PS	920	6	"USSR---Ostrov Okty abr’skoy Revolyutsii, Komsomolets, Pioner"
CO	208	2	USSR---Ostrov Para mushir
PO	1346	6	USSR---Ostrov Para mushir
PS	988	6	USSR---Ostrov Para mushir
CO	120	2	USSR---Ostrov Saare maa
PO	1207	6	USSR---Ostrov Saare maa
PS	849	6	USSR---Ostrov Saare maa
CO	199	2	USSR---Ostrov Sakha lin

PO	1337	6	USSR---Ostrov Sakha lin
PS	979	6	USSR---Ostrov Sakha lin
CO	204	2	USSR---Ostrov Urup
PO	1342	6	USSR---Ostrov Urup
PS	984	6	USSR---Ostrov Urup
CO	126	1	USSR---Ozero Ladozhskoye (Lake Ladoga)
PO	1224	1	USSR---Ozero Ladozhskoye (Lake Ladoga)
PS	866	1	USSR---Ozero Ladozhskoye (Lake Ladoga)
CO	195	2	USSR---Shantarskiye Ostrova
PO	1332	6	USSR---Shantarskiye Ostrova
PS	974	6	USSR---Shantarskiye Ostrova
CO	4	2	USSR---Wrangel Is.
PO	1007	6	USSR---Wrangel Is.
PS	443	6	USSR---Wrangel Is.
PS	585	5	US---Alabama
PS	495	2	US---Arizona
PS	538	5	US---Arkansas
PS	649	4	US---Bahamas---Andros
PS	652	4	US---Bahamas---Andros
PS	653	4	US---Bahamas---Andros
PS	655	4	US---Bahamas---Andros
PS	658	4	US---Bahamas---Andros
PS	687	4	US---Bahamas---Cat Is.
PS	663	4	US---Bahamas---Eleuthera Is.
PS	664	4	US---Bahamas---Eleuthera Is.
PS	665	4	US---Bahamas---Eleuthera Is.
PS	648	4	US---Bahamas---Grand Bahama
PS	656	4	US---Bahamas---Grand Bahama
PS	657	4	US---Bahamas---Grand Bahama
PS	654	4	US---Bahamas---Great Abaco and Little Abaco
PS	659	4	US---Bahamas---Mores Is.
PS	660	4	US---Bahamas---New Providence Is.
PS	469	4	US---California
PS	486	4	US---California---San

			Clemente Is.
PS	478	4	US---California---San Miguel Is.
PS	483	4	US---California---San Nicholas Is.
PS	487	4	US---California---Santa Catalina Is.
PS	482	4	US---California---Santa Cruz Is.
PS	479	4	US---California---Santa Rosa Is.
PS	621	1	US---Clark Hill Reser voir
PS	505	6	US---Colorado
PS	707	5	US---Connecticut
PS	686	5	US---Delaware
PS	696	1	US---Delaware---bay
PS	589	4	US---Florida
PS	600	4	US---Florida---island
PS	633	4	US---Florida---island
PS	635	4	US---Florida---island
PS	636	4	US---Florida---island
PS	637	4	US---Florida---island
PS	639	4	US---Florida---island
PS	626	4	US---Florida---Keys
PS	627	4	US---Florida---Keys
PS	628	4	US---Florida---Keys
PS	629	4	US---Florida---Keys
PS	597	1	US---Florida---lake
PS	631	1	US---Florida---Lake Okeechobee
PS	623	4	US---Florida---Pine Is.
PS	622	4	US---Florida---Sani bel Is.
PS	598	6	US---Georgia
PS	500	1	US---Great Salt Lake
PS	489	6	US---Idaho
PS	554	4	US---Illinois
PS	586	5	US---Indiana
PS	527	6	US---Iowa
PS	517	4	US---Kansas
PS	573	6	US---Kentucky
PS	711	1	US---Lake Champlain
PS	714	2	US---Lake Cham plain---Grand Isle
PS	613	1	US---Lake Erie
PS	640	2	US---Lake Erie---Long Point (land)
PS	608	1	US---Lake George
PS	619	2	US---Lake Huron---Barrie Is.
PS	604	2	US---Lake Huron---Bois Blanc Is.
PS	641	2	US---Lake Huron---Christian Is.
PS	612	2	US---Lake Huron---Cockburn Is.
PS	609	2	US---Lake Huron---Drummond Is.
PS	625	2	US---Lake Huron---Fitzwilliam Is.

PS	615	2	US---Lake Huron---Manitoulin Is.
PS	496	1	US---Lake Mead (east)
PS	494	1	US---Lake Mead (west)
PS	599	2	US---Lake Michigan---Beaver Is.
PS	594	2	US---Lake Michigan---Manitou Is.
PS	591	2	US---Lake Michigan---Washington Is.
PS	645	1	US---Lake Ontario
PS	662	2	US---Lake Ontario---island
PS	667	2	US---Lake Ontario---island
PS	670	2	US---Lake Ontario---island
PS	601	1	US---Lake Seminole
PS	618	1	US---Lake St. Clair
PS	550	1	US---Lake Superior
PS	593	2	US---Lake Superior---Grand Is.
PS	605	2	US---Lake Superior---island
PS	581	2	US---Lake Superior---Isle Royale
PS	556	2	US---Lake Superior---Madeline Is.
PS	595	2	US---Lake Superior---Michipicoten Is.
PS	558	2	US---Lake Superior---Oak Is.
PS	563	2	US---Lake Superior---Outer Is.
PS	583	2	US---Lake Superior---Pie Is.
PS	559	2	US---Lake Superior---Stockton Is.
PS	480	1	US---Lake Tahoe
PS	526	1	US---Lake Texoma
PS	587	1	US---Lakes Michigan and Huron
PS	719	1	US---Long Is.---Great South Bay
PS	540	4	US---Louisiana
PS	543	1	US---Louisiana---Calcasieu Lake
PS	584	4	US---Louisiana---Cat Is.
PS	546	1	US---Louisiana---Grand Lake
PS	555	4	US---Louisiana---island
PS	557	4	US---Louisiana---island
PS	569	4	US---Louisiana---island
PS	570	4	US---Louisiana---island
PS	571	4	US---Louisiana---

			island
PS	574	4	US---Louisiana---island
PS	575	4	US---Louisiana---island
PS	576	4	US---Louisiana---island
PS	577	4	US---Louisiana---island
PS	578	4	US---Louisiana---island
PS	579	4	US---Louisiana---island
PS	580	4	US---Louisiana---island
PS	560	1	US---Louisiana---Lake Maurepas
PS	565	1	US---Louisiana---Lake Pontchartrain
PS	551	4	US---Louisiana---Marsh Is.
PS	541	1	US---Louisiana---Sabine Lake
PS	561	4	US---Louisiana---Timbalier Is.
PS	548	1	US---Louisiana---White Lake
PS	730	5	US---Maine
PS	736	5	US---Maine---island
PS	737	5	US---Maine---island
PS	738	5	US---Maine---island
PS	740	5	US---Maine---island
PS	741	5	US---Maine---island
PS	742	5	US---Maine---island
PS	743	5	US---Maine---island
PS	744	5	US---Maine---island
PS	745	5	US---Maine---island
PS	646	4	US---Maryland
PS	697	4	US---Maryland
PS	698	4	US---Maryland
PS	679	1	US---Maryland---inlet
PS	668	4	US---Maryland---island
PS	671	4	US---Maryland---island
PS	673	4	US---Maryland---island
PS	674	4	US---Maryland---island
PS	709	4	US---Massachusetts
PS	729	1	US---Massachusetts---bay
PS	733	1	US---Massachusetts---inlet
PS	731	4	US---Massachusetts---Marthas Vineyard
PS	732	4	US---Massachusetts---Nantucket Is.
PS	592	2	US---Michigan (lower)
PS	564	2	US---Michigan (upper)
PS	524	4	US---Minnesota

PS	531	1	US---Minnesota--- Lake of the Woods
PS	536	4	US---Minnesota--- Lake of the Woods--- island
PS	537	4	US---Minnesota--- Lake of the Woods--- island
PS	539	4	US---Minnesota--- Lake of the Woods--- island
PS	533	4	US---Minnesota--- Lake of the Woods--- land west of lake
PS	547	4	US---Minnesota---lit tle piece
PS	544	1	US---Minnesota--- Namakan Lake
PS	542	1	US---Minnesota--- Rainy Lake
PS	532	1	US---Minnesota--- Shoal Lake
PS	552	2	US---Mississippi
PS	530	2	US---Missouri
PS	492	5	US---Montana
PS	511	5	US---Nebraska
PS	481	5	US---Nevada
PS	720	3	US---New Hampshire
PS	715	1	US---New Hamp shire---lake
PS	717	1	US---New Hamp shire---lake
PS	694	2	US---New Jersey
PS	701	2	US---New Jersey--- island
PS	704	1	US---New Jersey--- Newark Bay
PS	504	5	US---New Mexico
PS	644	3	US---New York
PS	722	3	US---New York--- Fishers Is.
PS	706	3	US---New York--- Long Is.
PS	721	3	US---New York--- Shelter Is.
PS	607	4	US---North Carolina
PS	651	4	US---North Carolina--- Cape Fear
PS	676	4	US---North Carolina--- island
PS	677	4	US---North Carolina--- island
PS	681	4	US---North Carolina--- island
PS	684	4	US---North Carolina--- island
PS	685	4	US---North Carolina--- island
PS	688	4	US---North Carolina--- island
PS	689	4	US---North Carolina---

			island
PS	690	4	US---North Carolina---island
PS	691	4	US---North Carolina---island
PS	692	4	US---North Carolina---island
PS	513	3	US---North Dakota
PS	603	4	US---Ohio
PS	515	3	US---Oklahoma
PS	468	3	US---Oregon
PS	638	6	US---Pennsylvania
PS	723	6	US---Rhode Is.
PS	725	6	US---Rhode Is.---Block Is.
PS	726	6	US---Rhode Is.---island
PS	727	6	US---Rhode Is.---island
PS	728	6	US---Rhode Is.---island
PS	606	1	US---Sault Ste. Marie
PS	614	2	US---South Carolina
PS	512	2	US---South Dakota
PS	703	3	US---Staten Is.
PS	566	3	US---Tennessee
PS	506	2	US---Texas
PS	522	2	US---Texas---island
PS	534	2	US---Texas---island
PS	523	2	US---Texas---Padre Is.
PS	497	3	US---Utah
PS	712	2	US---Vermont
PS	611	2	US---Virginia
PS	675	2	US---Virginia
PS	682	2	US---Virginia
PS	683	2	US---Virginia
PS	672	2	US---Virginia---island
PS	678	2	US---Virginia---island
PS	680	2	US---Virginia---island
PS	695	2	US---Virginia---island
PS	467	2	US---Washington
PS	471	2	US---Washington---island
PS	472	2	US---Washington---island
PS	473	2	US---Washington---island
PS	474	2	US---Washington---island
PS	475	2	US---Washington---island
PS	476	2	US---Washington---island
PS	477	2	US---Washington---island
PS	620	3	US---West Virginia
PS	545	3	US---Wisconsin
PS	502	4	US---Wyoming
US	247	3	Utah
PO	1097	3	Venezuela
PS	710	3	Venezuela

US	412	2	Vermont
PO	1287	4	Vietnam
PS	929	4	Vietnam
US	331	2	Virginia
US	382	2	Virginia
US	389	2	Virginia
US	390	2	Virginia
US	379	2	Virginia---island
US	385	2	Virginia---island
US	387	2	Virginia---island
US	401	2	Virginia---island
PO	1152	3	Wales
PS	794	3	Wales
US	224	2	Washington
US	227	2	Washington---island
US	228	2	Washington---island
US	229	2	Washington---island
US	230	2	Washington---island
US	231	2	Washington---island
US	232	2	Washington---island
US	233	2	Washington---island
PO	1166	4	West Germany
PS	808	4	West Germany
US	338	3	West Virginia
PO	1128	6	Western Sahara
PS	770	6	Western Sahara
US	277	3	Wisconsin
US	249	4	Wyoming
PO	1245	6	Yemen (Aden)
PS	887	6	Yemen (Aden)
CO	138	2	Yemen (Aden)--- Socotra (island)
PO	1256	6	Yemen (Aden)--- Socotra (island)
PS	898	6	Yemen (Aden)--- Socotra (island)
PO	1243	5	Yemen (Sana)
PS	885	5	Yemen (Sana)
PO	1193	4	Yugoslavia
PS	835	4	Yugoslavia
PO	1190	6	Zaire
PS	832	6	Zaire
PO	1209	5	Zambia
PS	851	5	Zambia
PO	1214	3	Zimbabwe
PS	856	3	Zimbabwe

Mp 7. Ezmap parameter descriptions

Ezmap uses the following parameters; they are listed in alphabetical order.

AR

> *Character*---ARea. For retrieval only. The value of the map limits specifier JLTS from the last call to MAPSET.
> Default: MA

C1

Integer---Color index 1 (for map perimeter).
>=0 Specifies the color index of the map's perimeter. The user must do the calls to GKS routines to define the color indices.
<0 No change in color index from one part of the map to another.
Default: -1

C2

Integer---Color index 2 (for map grid).
>=0 Specifies the color index of the map's grid lines. The user must do the calls to GKS routines to define the color indices.
<0 No change in color index from one part of the map to another.
Default: -1

C3

Integer---Color index 3 (for map labels).
>=0 Specifies the color index of the map's labels. The user must do the calls to GKS routines to define the color indices.
<0 No change in color index from one part of the map to another.
Default: -1

C4

Integer---Color index 4 (for map limb line).
>=0 Specifies the color index of the map's limb line. The user must do the calls to GKS routines to define the color indices.
<0 No change in color index from one part of the map to another.
Default: -1

C5

Integer---Color index 5 (for continental outlines).
>=0 Specifies the color index of continental outlines on the map. The user must do the calls to GKS routines to define the color indices.
<0 No change in color index from one part of the map to another.
Default: -1

C6

Integer---Color index 6 (for US state outlines).
>=0 Specifies the color index of US state outlines on the map. The user must do the calls to GKS routines to define the color indices.
<0 No change in color index from one part of the map to another.
Default: -1

C7

Integer---Color index 7 (for countries).

>=0 Specifies the color index of country outlines (political boundaries) on the map. The user must do the calls to GKS routines to define the color indices.

<0 No change in color index from one part of the map to another.

Default: -1

DA

Integer---Grid DAshline pattern. A 16-bit quantity that defines a dashed-line pattern. The value of DA is converted to a 16-bit binary number where 0 represents a blank and 1 represents a dot or dash.
Default: 21845 (binary 0101010101010101)

DD

Real---Distance between Dots. The distance between dots along a dotted line drawn by MAPIT.
Default: 12. (out of 4096. See the description for the RE parameter.)

DL

Integer---Dotted Line flag.

<>0 User calls to MAPIT draw dotted lines.

0 Lines drawn by MAPIT are solid or dashed, depending on the current state of the Dashline package.

Default: 0

DL may be reset by a user version of MAPUSR to change the way in which the perimeter, the grid, the limb lines, and the outlines are drawn.

DO

Integer---Dotted Outline flag.

<>0 Outlines are dotted.

0 Outlines are solid.

Default: 0

EL

Integer---ELliptical perimeter flag.

<>0 Only that part of the map which falls inside an ellipse inscribed within the normal rectangular perimeter is drawn. This is particularly appropriate for use with azimuthal projections and angular limits specifying a square (in which case the ellipse becomes a circle), but it will work for any map.

0 That part of the map that falls within the rectangular perimeter is drawn.

Default: 0

ER

Integer---ERror. The current value of the error flag. For retrieval only.
Default: 0 (no error)

G1

Integer---Group 1 (group identifier for geographic entities and perimeter). The group identifier to be used by MAPBLA when putting into the area map the group of edges that define the division of the plotter frame into the projected images of geographic entities.
Default: 1

G2

Integer---Group 2 (group identifier for vertical strips). The group identifier to be used by MAPBLA when putting into the area map the group of edges that define the division of the plotter frame into vertical strips.
Default: 2

GD

Real---Grid Drawing resolution. The distance between points used to draw the grid, in degrees. User values must fall between .001 and 10.
Default: 1.

GR

Real---GRid spacing. The desired grid spacing, in degrees.
0 Grid drawing is suppressed.
Default: 10.

IN

Integer---INitialization flag. For retrieval only.
<>0 Ezmap needs to be initialized (by a CALL MAPINT).
0 Ezmap does not need to be initialized.
Default: 0

LA

Integer---meridian/pole LAbel flag.
<>0 Label the meridians and the poles.
0 Do not label the meridians and poles.
Default: 1

LS

Integer---Label Size. Controls label size by specifying a character width, to be used in a call to PWRIT.

Default: 1 (gives a character width of 12 plotter units)

MV

Real---Minimum Vector length. To be used by MAPIT. Any point closer to the previous point than this is omitted.
Default: 4. (out of 4096; see the description for RE)

OU

Character---OUtline data flag. Says which set of outline data to use:
NO No outlines.
CO Continental outlines.
PO Continental outlines plus international political outlines.
US US state outlines.
PS Continental outlines plus international political outlines plus U.S. state outlines.
Default: CO

PE

Integer---PErimeter flag. Draws the perimeter around the map.
<>0 Draw the perimeter.
0 No perimeter is drawn.
Default: 1

PN

Real---**P**LO**N** value. The value of PLON from the last call to MAPROJ. For retrieval only.
Default: 0.

PR

Character---PRojection specifier value. The value of the projection specifier JPRJ from the last call to MAPROJ. For retrieval only.
Default: CE

PT

Real---**P**LA**T** value. The value of PLAT from the last call to MAPROJ. For retrieval only.
Default: 0.

P1

Real---**P**LM**1**(1) value. Retrieves the PLM1(1) value from the call to MAPSET. For retrieval only.
Default: 0.

P2

Real---**P**LM**2**(1) value. Retrieves the PLM2(1) value from the call to MAPSET. For retrieval only. Default: 0.

P3

Real---**P**LM**3**(1) value. Retrieves the PLM3(1) value from the call to MAPSET. For retrieval only. Default: 0.

P4

Real---**P**LM**4**(1) value. Retrieves the PLM4(1) value from the call to MAPSET. For retrieval only. Default: 0.

P5

Real---PLM1(2) value. Retrieves the PLM1(2) value from the call to MAPSET. For retrieval only. Default: 0.

P6

Real---PLM2(2) value. Retrieves the PLM2(2) value from the call to MAPSET. For retrieval only. Default: 0.

P7

Real---PLM3(2) value. Retrieves the PLM3(2) value from the call to MAPSET. For retrieval only. Default: 0.

P8

Real---PLM4(2) value. Retrieves the PLM4(2) value from the call to MAPSET. For retrieval only. Default: 0.

RE

Integer---REsolution. The width of the target plotter, in plotter units.
Default: 4096

RO

Integer---ROtation. The value of ROTA from the last call to MAPROJ. For retrieval only.
Default: 0

SA

Real---SAtellite view distance. The value is the distance of the satellite from the center of the earth, in multiples of the earth's radius. See also S1 and S2.

>1. A satellite-view projection replaces the orthographic projection.

0. The orthographic projection is used.

Default: 0.

S1

Real---Satellite angle 1. Measures the angle, in degrees, between the line from the satellite to the center of the earth and the line of sight of the satellite. Used only when SA>1.
Default: 0.

S2

Real---Satellite angle 2. Used only when SA>1.
Let the case where S1=S2=0 be called the basic satellite view. If S1<>0.0, then S2 specifies the angle, in degrees, from the positive u axis of the basic satellite view counterclockwise to the line OP, where O is the origin of the basic view, and P is the projection on the basic view, of the desired line of sight from the satellite. When S1 and S2 are not equal to zero, the portion of the earth projected remains the same, but the edge of the part of the u/v plane covered by the projection is an ellipse, a parabola, or a hyperbola with an axis at an angle S2 to the u axis.
Default: 0.

SR

Real---Latitude/longitude Search Radius. A search radius, in degrees, used by MAPINT in finding the latitude/longitude range of a map. User values must be between .001 and 10. This value should only be changed on the advice of an NCAR consultant.
Default: 1.

VS

Integer---Vertical-Stripping parameter. Determines whether MAPBLA puts edge group G2 into the area map to define a set of more than one vertical strip.

<=0 MAPBLA is prevented from putting edge group G2 into the area map.

>0 Requests that MAPBLA put edge group G2 into the area map and specifies the number of vertical strips.

Default: 1 (A single vertical strip is defined to encompass the entire plot.)

XL

Real---**XL**OW value. The value of XLOW from the last call to MAPPOS. For retrieval only.
Default: 0.05

XR

Real---**XR**OW value. The value of XROW from the last call to MAPPOS. For retrieval only.
Default: 0.95

YB

Real---**YB**OW value. The value of YBOW from the last call to MAPPOS. For retrieval only. Default: 0.05

YT

Real---**YT**OW value. The value of YTOW from the last call to MAPPOS. For retrieval only. Default: 0.95

Using Conpack

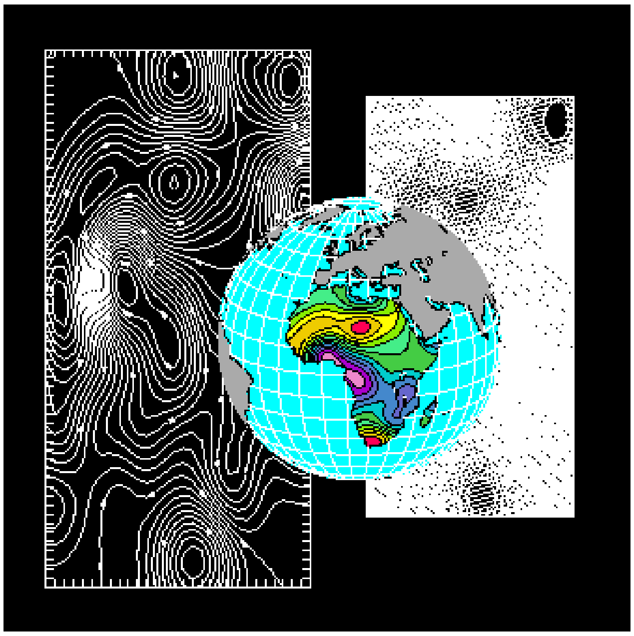
Table of contents

Using Conpack

Cp 1. What is Conpack?

Conpack is a color contouring utility. Conpack can be used to generate contour plots of almost any rectangular array of data; it has an extensive variety of options.

Several uses for Conpack



Cp 1.1 Table of Conpack user entry points

Because of the flexibility of Conpack, the process of generating a plot usually requires calling several routines. This module organizes the routines according to their functions.

Quick contouring routines

These routines draw contour plots using a single call.

CPCNRC

> Draws black-and-white contours with a single call. Simulates old routine CONREC. See module Cp 1.4.

CPEZCT

> Draws black-and-white contours with a single call. Simulates old routine EZCNTR (in the old CONREC utility). See module Cp 1.3.

Conpack initialization and data support routines

These routines aid in converting data to a form Conpack can use, initialize Conpack to handle specific kinds of data, or add features to an area map for filled or masked contours.

CPRECT

> Initializes contouring of data on a dense rectangular array of data. See module Cp 3.2.

CPSPS1

Initializes contouring of data on a sparse rectangular grid. See module Cp 3.3.

CPSPS2

Initializes contouring of data on an irregularly spaced rectangular grid. See module Cp 3.4.

IDSFFT

A Bivar routine that interpolates randomly spaced data onto a dense regular grid. See module Cp 3.5.

CPCLAM

Adds contour lines to area map. See module Cp 5.3.

CPLBAM

Adds label boxes to an area map, so that lines aren't drawn through labels. See module Cp 5.4.

CPMPXY

Maps from a rectangular coordinate system to some other coordinate system. See module Cp 3.10; also see module Cp 3.7.

Background routine

Most people want a perimeter drawn around their contour plot.

CPBACK

Draws a perimeter. See module Cp 2.2.

Labeling routines

Labeling is controlled by several routines, depending on which label you want to modify.

CPLBDR

Draws labels. See module Cp 6.1.

CPPKLB

Picks a set of labels for labeled contour levels. See module Cp 6.2.

CPCHLL

Changes drawing of line labels.

CPCHIL

Changes drawing of information labels.

CPCHHL

Changes drawing of high and low labels.

CPCHCF

Changes drawing of constant field message.

Contour line drawing routines

Contour lines are drawn as a separate step.

CPCLDR

Draws contour lines. See module Cp 4.2.

CPCLDM

Draws contour lines masked by existing area map. See module Cp 5.5.

CPDRPL

Provides polyline drawing for CPCLDM. See module Cp 5.6.

CPCICA

Incorporates, into a user's cell array, color indices determined by examining where the user's contours lie relative to the cell array. See module Cp 9.3.

CPCHCL

Changes drawing of contour lines.

CPCLTR

Traces the contour lines at a given level and retrieves them for some sort of user-defined processing.

CPPKCL

Picks a set of contour levels. See module Cp 4.7.

Parameter access routines

There are seven calls that aid in changing and retrieving parameter values.

CPSETC

Sets character values. See module Cp 1.6.

CPSETI

Sets integer values. See module Cp 1.6.

CPSETR

Sets real values. See module Cp 1.6.

CPRSET

Resets default values. See module Cp 1.6.

CPGETC

Retrieves current character values. See module Cp 1.6.

CPGETI

Retrieves current integer values. See module Cp 1.6.

CPGETR

Retrieves current real values. See module Cp 1.6.

Cp 1.2 Table of Conpack parameters

This table is a quick reference guide to the examples in which Conpack parameters are used and the modules in which Conpack parameters are described. More complete descriptions appear in section "Cp 10. Conpack parameter descriptions."

The behavior of a typical routine in an NCAR Graphics utility is sometimes determined entirely by the routine's arguments, but frequently it is also affected by the value of one or more of the utility's parameters. A "parameter" is a variable that controls the behavior of a utility; parameters are accessed via parameter-access routines that can set or retrieve the parameter value.

Instructions for setting and retrieving Conpack parameters are provided in module "Cp 1.6 Conpack parameters:What they do and how to use them."

```
--------------------------------------------------------------------------------
Parameter  Brief description                Fortran type        Examples   Module
```

AIA	Area Identifier Above	Integer array	---	Cp 5.1
			ccpscam	---
AIB	Area Identifier Below	Integer array	---	Cp 5.1
			ccpscam	---
CAF	Cell Array Flag	Integer	---	Cp 10.
CFA	Constant Field label Angle	Real	ccpila	Cp 7.6
CFB	Constant Field label Box flag	Integer	ccpllb	Cp 7.11
CFC	Constant Field label Color index	Integer	ccplbdr	Cp 7.10
CFF	Constant Field Found flag	Integer	ccpcff	Cp 7.16
CFL	Constant Field label Line width	Real	ccplll	Cp 7.12
CFP	Constant Field label Positioning flag	Integer	ccpcfx	Cp 7.14
CFS	Constant Field label Size	Real	ccpils	Cp 7.7
CFT	Constant Field label Text string	Character	ccpilt	Cp 7.8
CFW	Constant Field label White space width	Real	ccpllw	Cp 7.13
CFX	Constant Field label X coordinate	Real	ccpcfx	Cp 7.14
CFY	Constant Field label Y coordinate	Real	ccpcfx	Cp 7.14
CIS	Contour Interval Specifier	Real	ccpcis,	Cp 4.6,
			ccpcis	Cp 6.4
CIT	Contour Interval Table	Real array	ccpcit,	Cp 4.5,
			ccpcit	Cp 6.3
CIU	Contour Interval Used	Real	---	Cp 10.
CLC	Contour Line Color index	Integer array	ccpclc	Cp 4.13
CLD	Contour Line Dash pattern	Character array or Integer array	ccpcld	Cp 4.11
CLL	Contour Line Line width	Real array	ccpcll	Cp 4.12
CLS	Contour Level Selection flag	Integer	---	Cp 4.4,
			ccpncls,	Cp 4.9,
			ccphand	Cp 4.10
CLU	Contour Level Use flags	Integer array	ccpspv,	Cp 3.6,
			ccpclu,	Cp 4.14,
			ccpclu	Cp 6.5
CLV	Contour Level Values	Real array	ccppkcl,	Cp 4.8,
			ccphand	Cp 4.10
CMN	Contour MiNimum	Real	ccpcis	Cp 4.6
CMX	Contour MaXimum	Real	ccpcis	Cp 4.6
CTM	Character TeMporary	Character	---	Cp 10.
CWM	Character Width Multiplier	Real	ccpllt,	Cp 6.12,
			ccpils	Cp 7.7,
			ccpllw	Cp 7.13
DPS	Dash Pattern Size	Real	---	Cp 10.
DPU	Dash Pattern Use flag	Integer	---	Cp 10.
DPV	Dash Pattern Vector length	Real	---	Cp 10.
GIC	Group Identifier for Contour lines	Integer	ccpvs	Cp 5.2
GIL	Group Identifier for Label boxes	Integer	ccpvs	Cp 5.2
GIS	Group Identifier for Strips	Integer	ccpvs	Cp 5.2
HIC	HIgh label Color index	Integer	ccplbdr	Cp 7.10
HIT	HIgh label Text string	Character	---	Cp 10.
HCF	HaChure Flag	Integer	ccphcf	Cp 9.2
HCL	HaChure Length	Real	ccphcf	Cp 9.2
HCS	HaChure Spacing	Real	ccphcf	Cp 9.2
HLA	High/Low label Angle	Real	ccpila	Cp 7.6

HLB	High/Low label Box flag	Integer	ccpllb	Cp 7.11
HLC	High/Low label Color index	Integer	ccplbdr	Cp 7.10
HLL	High/Low label Line width	Real	ccplll	Cp 7.12
HLO	High/Low label Overlap flag	Integer	ccphl	Cp 7.15
HLS	High/Low label Size	Real	ccpils	Cp 7.7
HLT	High/Low label Text strings	Character	ccphlt	Cp 7.9
HLW	High/Low label White space width	Real	ccpllw	Cp 7.13
HLX	High/Low search radius in X	Integer	ccphl	Cp 7.15
HLY	High/Low search radius in Y	Integer	ccphl	Cp 7.15
ILA	Information Label Angle	Real	ccpila	Cp 7.6
ILB	Information Label Box flag	Integer	ccpllb	Cp 7.11
ILC	Information Label Color index	Integer	ccplbdr	Cp 7.10
ILL	Information Label Line width	Real	ccplll	Cp 7.12
ILP	Information Label Positioning flag	Integer	ccpcfx	Cp 7.14
ILS	Information Label Size	Real	ccpils	Cp 7.7
ILT	Information Label Text string	Character	ccpilt	Cp 7.8
ILW	Information Label White space width	Real	ccpllw	Cp 7.13
ILX	Information Label X coordinate	Real	ccpcfx	Cp 7.14
ILY	Information Label Y coordinate	Real	ccpcfx	Cp 7.14
IWM	Integer Workspace for Masking	Integer	---	Cp 10.
IWU	Integer Workspace Usage	Integer	ccprwu	Cp 3.12
LBC	Label Box Color index	Integer	ccpllb	Cp 7.11
LBX	Label Box X coordinate	Real	---	Cp 10.
LBY	Label Box Y coordinate	Real	---	Cp 10.
LIS	Label Interval Specifier	Integer	ccpcis, ccpcis	Cp 4.6, Cp 6.4
LIT	Label Interval Table	Integer array	ccpcit, ccpcit	Cp 4.5, Cp 6.3
LIU	Label Interval Used	Integer	ccpcit	Cp 6.3
LLA	Line Label Angle	Real	ccpllo	Cp 6.10
LLB	Line Label Box flag	Integer	ccpllb	Cp 7.11
LLC	Line Label Color index	Integer array	ccpllc	Cp 6.11
LLL	Line Label Line width	Real	ccplll	Cp 7.12
LLO	Line Label Orientation	Integer	ccpllo	Cp 6.10
LLP	Line Label Positioning	Integer	---	Cp 6.6
LLS	Line Label Size	Real	ccpllt	Cp 6.12
LLT	Line Label Text string	Character array	ccpllt	Cp 6.12
LLW	Line Label White space	Real	ccpllw	Cp 7.13
LOC	LOw label Color index	Integer	ccplbdr	Cp 7.10
LOT	LOw label Text string	Character	---	Cp 10.
MAP	MAPping flag	Integer	ccpmap, ccpcir	Cp 3.8, Cp 3.9
NCL	Number of Contour Levels	Integer	ccppkcl, ccphand	Cp 4.8, Cp 4.10
NEL	Numeric Exponent Length	Integer	ccpnet	Cp 7.5
NET	Numeric Exponent Type	Integer	ccpnet	Cp 7.5
NEU	Numeric Exponent Use flag	Integer	ccpnet	Cp 7.5
NLS	Numeric Leftmost Significant digit flag	Integer	ccpnsd	Cp 7.3
NLZ	Numeric Leading Zero flag	Integer	ccpnof	Cp 7.4
NOF	Numeric Omission Flags	Integer	ccpnof	Cp 7.4
NSD	Number of Significant Digits	Integer	ccpnsd	Cp 7.3
NVS	Number of Vertical Strips	Integer	ccpvs	Cp 5.2
ORV	Out-of-Range Value	Real	ccpmap	Cp 3.8

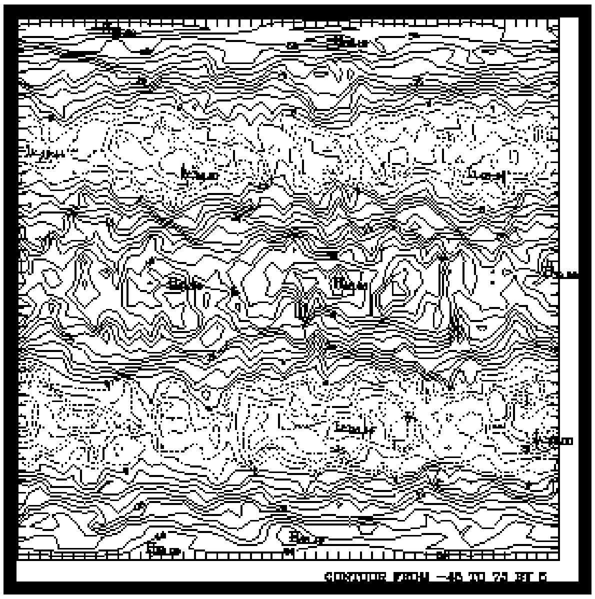
PAI	Parameter Array Index	Integer	---	Cp 1.6,
			ccpspv,	Cp 3.6,
			ccpcit,	Cp 4.5,
			ccppkcl,	Cp 4.8,
			ccphand,	Cp 4.10,
			ccpcit	Cp 6.3
PC1	Penalty scheme Constant 1	Real	ccppc1	Cp 6.9.1
PC2	Penalty scheme Constant 2	Real	ccppc2	Cp 6.9.2
PC3	Penalty scheme Constant 3	Real	ccppc3	Cp 6.9.3
PC4	Penalty scheme Constant 4	Real	ccppc4	Cp 6.9.4
PC5	Penalty scheme Constant 5	Real	ccppc4	Cp 6.9.4
PC6	Penalty scheme Constant 6	Real	ccppc4	Cp 6.9.4
PIC	Point Interpolation flag for Contours	Integer	ccpt2d	Cp 9.1
PIE	Point Interpolation flag for Edges	Integer	---	Cp 10.
PW1	Penalty scheme Weight 1	Real	ccppc1	Cp 6.9.1
PW2	Penalty scheme Weight 2	Real	ccppc2	Cp 6.9.2
PW3	Penalty scheme Weight 3	Real	ccppc3	Cp 6.9.3
PW4	Penalty scheme Weight 4	Real	ccppc4	Cp 6.9.4
RC1	Regular scheme Constant 1	Real	ccprc	Cp 6.8
RC2	Regular scheme Constant 2	Real	ccprc	Cp 6.8
RC3	Regular scheme Constant 3	Real	ccprc	Cp 6.8
RWC	Real Workspace for Contours	Integer	ccprwc	Cp 6.7
RWG	Real Workspace for Gradients	Integer	---	Cp 10.
RWM	Real Workspace for Masking	Integer	---	Cp 10.
RWU	Real Workspace Usage	Integer	ccprwu	Cp 3.12
SET	do-SET-call flag	Integer	ccpset	Cp 2.7
SFS	Scale Factor Selector	Real	ccpklb	Cp 10.
SFU	Scale Factor Used	Real	ccpklb	Cp 10.
SPV	SPecial Value	Real	ccpspv	Cp 3.6
SSL	Smoothed Segment Length	Real	ccpt2d	Cp 9.1
T2D	Tension on 2-Dimensional splines	Real	ccpt2d	Cp 9.1
T3D	Tension on 3-Dimensional splines	Real	---	Cp 10.
VPB	ViewPort Bottom	Real	ccpvp	Cp 2.6
VPL	ViewPort Left	Real	ccpvp	Cp 2.6
VPR	ViewPort Right	Real	ccpvp	Cp 2.6
VPS	ViewPort Shape	Real	ccpvp	Cp 2.6
VPT	ViewPort Top	Real	ccpvp	Cp 2.6
WDB	WinDow Bottom	Real	---	Cp 10.
WDL	WinDow Left	Real	---	Cp 10.
WDR	WinDow Right	Real	---	Cp 10.
WDT	WinDow Top	Real	---	Cp 10.
WSO	WorkSpace Overflow flag	Integer	---	Cp 10.
XC1	X Coordinate at index 1	Real	ccpga,	---
			ccpset,	---
			---	Cp 2.3,
			ccpmap,	Cp 3.8,
			ccpcir	Cp 3.9
XCM	X Coordinate at index M	Real	ccpga,	---
			ccpset,	---
			---	Cp 2.3,
			ccpmap,	Cp 3.8,
			ccpcir	Cp 3.9
YC1	Y Coordinate at index 1	Real	ccpga,	---
			ccpset,	---
			---	Cp 2.3,
			ccpmap,	Cp 3.8,
			ccpcir	Cp 3.9

YCN	Y Coordinate at index N	Real	ccpga,	---
			ccpset,	---
			---	Cp 2.3,
			ccpmap,	Cp 3.8,
			ccpcir	Cp 3.9
ZD1	Z data array Dimension 1	Integer	---	Cp 10.
ZDM	Z data array Dimension M	Integer	---	Cp 10.
ZDN	Z data array Dimension N	Integer	---	Cp 10.
ZDS	Z data array Dimension Selector	Integer	---	Cp 10.
ZDU	Z Data value, Unscaled	Real	---	Cp 10.
ZDV	Z Data Value	Real	---	Cp 10.
ZMN	Z MiNimum value	Real	---	Cp 10.
ZMX	Z MaXimum value	Real	---	Cp 10.

Cp 1.3 Producing a "quick and dirty" plot

If all you want to do is get a quick look at some data that are arrayed on a regular rectangular grid, you may want to try using the CPEZCT routine. CPEZCT often provides good way to preview your data.

Contours generated with CPEZCT



Code segment from ccpezct.f

```
      CALL CPEZCT (ZREG, M, N)
```

Synopsis

```
      CALL CPEZCT (ZREG, M, N)
```

Arguments

ZREG(M,N)

Real, Input---A filled M by N data array holding values to be contoured.

M

Integer, Input---The first dimension of ZREG.

N

Integer, Input---The second dimension of ZREG.

CPEZCT is one of the two Conpack routines that draw contours with a single call. CPEZCT also calls FRAME, the SPPS routine for separating pictures in your CGM. To generate a plot, you only have to call CPEZCT.

CPEZCT makes several assumptions:

- All of the array will be contoured.
- The data are regularly spaced.
- You don't want specific contour levels.
- You don't care too much about the scale factors.
- You want highs and lows marked.
- Contour lines with negative values will be drawn with a dashed-line pattern.

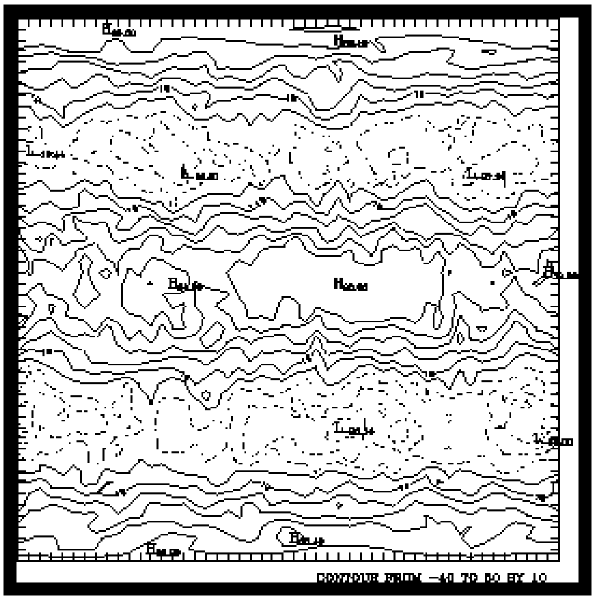
Exercises

1. Call CPEZCT with some of your own data.

Cp 1.4 More complex black and white plots

You may need more control over drawing black-and-white contour plots than CPEZCT offers, but you probably still want to obtain initial results quickly. CPCNRC offers basic control over labels, contour line patterns, choosing contour levels, and other functions.

Contours between bounds



Code segment from ccpcnrc.f

```
1       CALL CPCNRC (ZREG, M, M, N, -40., 50., 10., 0, 0, -366)
```

Synopsis

```
      CALL CPCNRC (ZREG, K, M, N, FLOW, FHGH, FINC, NSET, NHGH, NDSH)
```

Arguments

ZREG(K,***n***)

Real Array, Input---The array of data to be contoured. Dimensioned K by *n*, where *n*>=N.

K

Integer, Input---The first dimension of the declared Fortran array ZREG.

M

Integer, Input---The first dimension of the data in the array.

N

Integer, Input---The second dimension of the data array. N<=*n*, where *n* is the declared second dimension of ZREG.

FLOW

Real, Input---The desired lowest contour level. If FLOW>=FHGH, Conpack chooses the set of contour levels.

FHGH

Real, Input---The desired highest contour level. If FHGH<=FLOW, Conpack chooses the set of contour levels.

FINC

Real, Input---Determines how contour levels are chosen, as follows:

>0 FINC is the desired contour interval to be used. If FINC>0 and FLOW<FHGH, the intervals used will be FLOW, FLOW+FINC, FLOW+2*FINC, ... FLOW+*n**FINC, where *n* is the largest integer such that FLOW+*n**FINC<=FHGH. If FLOW>=FHGH, then the contour levels will be those integer multiples of FINC that fall between the minimum value in ZREG and the maximum value in ZREG.

<=0 Conpack chooses the contour interval to give at least 16 contour levels (if FINC=0), or MAX(1,INT(-FINC)) contour levels (if FINC<0) between the minimum and maximum values in ZREG. All the contour levels will be integer multiples of the chosen interval. If FLOW<FHGH, then no contour lines will be drawn outside the range (FLOW, FHGH).

NSET

Integer, Input---Determines how the plot is mapped into the plotter frame as follows:

0 The default configuration is used. Conpack calls SET with:

```
CALL SET (.05, .95, .05, .95, WDL, WDR, WDB, WDT, 1)
```

where WDL, WDR, WDB, and WDT are the left, right, bottom, and top window boundaries. CPBACK is called to draw a perimeter.

<0 The contour plot fills the current viewport. First, Conpack calls GETSET, then SET:

```
CALL GETSET (VPL, VPR, VPB, VPT, ...)
CALL SET (VPL, VPR, VPB, VPT, WDL, WDR, WDB, WDT, 1)
```

where VPL, VPR, VPB, and VPT are the left, right, bottom, and top viewport boundaries and where WDL, WDR, WDB, and WDT are the left, right, bottom, and top window boundaries. CPBACK is **not** called to draw a perimeter.

>0 Conpack does not call SET. It is assumed that you have either called SET directly, or that you have called a utility that calls SET (such as Ezmap). CPBACK is not called to draw a perimeter. This option should be used to overlay contours on an existing background.

NHGH

Integer, Input---Determines whether highs, lows, and data points are to be labeled as follows:

0 Each high is marked with an "H", and each low is marked with an "L". The value of the point is written as a subscript to the "H" or the "L".

>0 Each data point is marked with the value at that point. No attempt is made to deal with overlapping labels, so this option results in a mess on very dense data. Parameters HLA and HLS affect label angle and size.

<0 No high, low, or data points are labeled.

NDSH

> *Integer, Input*---Specifies the dash pattern(s) to be used for drawing contour lines. ABS(NDSH) is the decimal value for the 10-bit dash pattern to be used. Conpack makes this dash pattern into a 16bit pattern by appending a copy of the high-order six bits.
>
> > **0** or **1** or **1023**
> > Solid lines are used for all contour lines.
> > **>0** The specified dash pattern is used for all contour lines (except for special values 1 and 1023).
> > **<0** The dash pattern is used only for contour lines with negative values.

Discussion

The **ccpcnrc.f** code segment and its plot show that a single call to CPCNRC gives you a considerable amount of control over black-and-white plots.

The first four arguments for CPCNRC are the data array and its dimensions. The fifth and sixth arguments limit the contour lines to be drawn between -40. and 50, and the seventh argument sets the contour interval to 10.

The eighth argument gives NSET the value of zero to produce the default background. The ninth argument sets NHGH to zero, causing highs and lows to be labeled in the style shown.

The final argument sets a negative value for NDSH; this plots contour lines with negative values using dashed lines, and it plots contour lines with a zero or a positive value using solid lines. To set NDSH, first create the dash pattern of your choice using a ten-bit binary string where 0 represents a space and 1 represents a dash. The **ccpcnrc** example started with the number 0101101110*2*; this was then converted to 366*10* and used as a negative value for NDSH. This NDSH value specifies the string 0101101110010110*2* because the first six bits of the 10-bit binary string are copied and added to the end to create a 16-bit string.

Exercises

1. Use CPCNRC to contour your data. Use the default line type and label highs and lows.
2. Using **ccpcnrc.f**, try changing the dash pattern a few times.

Cp 1.5 What calls do I need to get my Conpack plot?

Conpack programs usually have a very similar structure. Parameters are usually set first, Conpack must be initialized, labels drawn, contours drawn, and a background added.

Important note: The sequence of calls presented in this Conpack functional outline and shown throughout this Conpack tutorial is commonly used. This approach speeds new users' learning of this sometimes complex material; it does not specify the *only* way you can use Conpack. As you gain experience using Conpack, you will probably have creative insights that will allow you to achieve a wide variety of results.

This is one of the strengths of Conpack: the order of the calls can be varied to achieve different results in your output plots. When you fill contours with solid colors (called solid fill), you need to draw the background after the fill so the background grid overwrites the fill colors and not vice versa. In other cases, it may be useful to call the background-drawing routine CPBACK before making other drawing calls.

Conpack functional outline

```
----------------------------------------------------------
 *  1.  Open GKS
    2.  Set window and viewport
    3.  Put in titles and other labels
    4.  Set up general parameters
    5.  Set up area maps
 *  6.  Initialize Conpack
    7.  Force Conpack to choose contour levels if necessary
    8.  Modify Conpack-chosen parameters
    9.  Draw labels
 *  10.  Draw background
 *  11.  Draw contour lines
 *  12.  Call FRAME
 *  13.  Close GKS
----------------------------------------------------------
```

* Steps needed to produce a contour plot.

Discussion

One of the most common user problems is making Conpack calls in an incorrect order. Since Conpack requires information in a nonintuitive order, this tutorial discusses Conpack routines in an order designed to help you learn them.

For example, Conpack usually uses the dimensions of your data to determine what shape to draw the rectangle around your contours (the contour rectangle). So you can't ask Conpack to draw a perimeter around your contours until it has your data. However, most of us think about drawing the boundary of a plot before drawing contours in it. Therefore, this tutorial first discusses drawing contour rectangles and other backgrounds for Conpack before it discusses Conpack initialization, which is typically the first Conpack call in a program.

The following lines are highlighted in the Conpack functional outline:

- Open GKS
- Initialize Conpack
- Draw background
- Draw contour lines
- Call FRAME
- Close GKS

These lines show the absolute minimum calls that must be made to draw a plot with Conpack. Both CPEZCT and CPCNRC make the Conpack calls for you. The lines without asterisks in the functional outline show proper placement of frequently desired calls that are not required.

Because opening GKS, closing GKS, and clipping are discussed in the *NCAR Graphics Fundamentals* guide, they are discussed only very briefly here. GKS must be opened and closed for every CGM file---these calls are included in the Conpack functional outline as a reminder that graphics code does not run without them.

Conpack can be initialized with one of the three routines CPRECT, CPSPS1, or CPSPS2, depending on the type of data to be contoured. Section "Cp 3. Initializing Conpack" describes each type of data and the appropriate method for initializing Conpack in each case.

Contour lines are drawn with either a call to CPCLDR or CPCLDM. If you just want to draw contour lines, use CPCLDR. If you want to protect your labels from being drawn over by contour lines, or if you need to use Areas for some other reason, use CPCLDM.

As discussed in the *NCAR Graphics Fundamentals* guide, FRAME is an SPPS routine that flushes all the buffers and closes a frame so that you can either quit NCAR Graphics or draw another picture. FRAME must be called to finish a Conpack plot, and it is included in the Conpack functional outline as a reminder that you must always call it.

Cp 1.6 Conpack parameters: What they do and how to use them

Conpack relies heavily on parameters and parameter-setting routines. Parameter-setting options allow you to specify all aspects of your contour plot. Seven routines help you set and retrieve parameter values, and the Parameter Array Index (PAI) helps you access individual elements in parameter arrays. The illustration shows how PAI accesses the sixth element in the Conpack parameter arrays.

The first argument in a call to CPGETI, CPSETI, CPGETR, CPSETR, CPGETC, or CPSETC is the parameter name. The second argument is either a variable in which the value of the named parameter is to be returned (via CPGET*x*) or an expression specifying the desired new value of the parameter (via CPSET*x*).

Use the PAI parameter to access individual array elements

	CLV	CLU	AIA	AIB	CLC	CLD	...
	1	1	1	1	1	1	1
	2	2	2	2	2	2	2
	3	3	3	3	3	3	3
	4	4	4	4	4	4	4
	5	5	5	5	5	5	5
PAI	6	6	6	6	6	6	6
	7	7	7	7	7	7	7
	8	8	8	8	8	8	8
	n	n	n	n	n	n	n

Synopsis

```
CALL CPGETC (PNAM, CVAL)
CALL CPGETI (PNAM, IVAL)
CALL CPGETR (PNAM, RVAL)
CALL CPSETC (PNAM, CVAL)
CALL CPSETI (PNAM, IVAL)
CALL CPSETR (PNAM, RVAL)
CALL CPRSET
```

Routines

CPGETC

Retrieves character parameter information.

CPGETI

Retrieves integer parameter information.

CPGETR

Retrieves real parameter information.

CPSETC

Sets character parameters.

CPSETI

Sets integer parameters.

CPSETR

Sets real parameters.

CPRSET

Resets all parameters to their initial default values.

Arguments

PNAM

Character expression, Input---The name of the parameter that you want to set or retrieve. Three-character parameter names appear in examples and discussions in place of PNAM. The parameter names and their meanings are enclosed in single quotation marks in the Fortran code to ensure that they are treated as character expressions rather than real or integer variables. Only the first three characters within the quotation marks are examined.

CVAL

Character expression, Input or Output---A character string or character variable.

IVAL

Integer, Input or Output---An integer value or integer variable.

RVAL

Real, Input or Output---A real value or real variable.

Discussion

In the Conpack functional outline, note that Conpack parameters are set before Conpack is initialized. Unless you are modifying Conpack-chosen parameters, it is always best to set your parameters before the initialization subroutine is called.

The CPRSET subroutine allows you to quickly reset all of the Conpack parameters to their default values between contour plots.

Some of the parameters are not simple integers or reals, but arrays. To access an individual element of a specific parameter array, you can specify individual elements using PAI (Parameter Array Index). For example, to set the *i*th element of the CLV array, first set PAI=I, then set CLV to the desired value, let's say 0.25:

```
CALL CPSETI ('PAI', I)
CALL CPSETR ('CLV', .25)
```

These calls set the *i*th contour level value to 1/4.

As individual parameters are described, examples of the CPSET*x* and CPGET*x* routines are shown, and the parameters that require CPSET*x* or CPGET*x* routine calls are clearly indicated.

Some parameters are described as Fortran type integer, or as integer arrays. These values can be set using:

```
CALL CPSETI ('PAR', ipar)
```

or

```
CALL CPSETR ('PAR', REAL(ipar))
```

where PAR is some integer parameter and *ipar* represents its integer value.

You can also set a real parameter using:

```
CALL CPSETI ('XXX', ival)
```

if the desired value is REAL(*ival*).

Caution: To avoid a common user problem, be sure to double-check your GET and SET calls to ensure that you don't mix integer and real parameters.

Cp 2. Backgrounds for your plots

Conceptually, the first thing that you want to do with your plot is to set up a background, so drawing backgrounds is discussed first. Because backgrounds are rarely drawn first when using Conpack, the functional outline below provides you with a suggested location in your code for background drawing. In later sections, the background drawing occurs in different places in the code. Later sections also demonstrate some techniques for producing the results you want.

Note: The Gridall or Autograph utilities are often used to draw backgrounds for Conpack; these calls can be made before you initialize Conpack.

Conpack functional outline

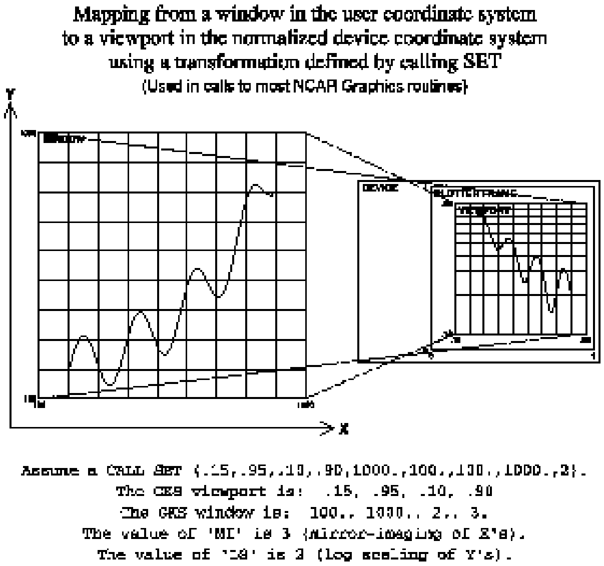
```
----------------------------------------------------------
    1.  Open GKS
    2.  Set window and viewport
    3.  Put in titles and other labels
 *  4.  Set up general parameters
    5.  Set up area maps
    6.  Initialize Conpack
    7.  Force Conpack to choose contour levels if necessary
    8.  Modify Conpack-chosen parameters
    9.  Draw labels
 *  10.  Draw background
    11.  Draw contour lines
    12.  Call FRAME
    13.  Close GKS
----------------------------------------------------------
```

* Steps discussed in this section.

Cp 2.1 Generating a background

Conpack offers an option for drawing a simple perimeter. You can also use Ezmap to draw maps with geographical and/or political outlines, Gridall to draw perimeters, axes, and grids, or Autograph for alternative backgrounds.

Conpack background options



Discussion

Conpack is designed to allow you to use most other utilities to draw backgrounds for your contour plot. This flexibility requires you to understand the SPPS routine SET, the Conpack parameter SET, and how other Conpack parameters affect the SET call.

The illustration shows how the coordinate system of your data (the user coordinate system) is mapped first into a window, then into the viewport (a portion of your display screen). Before any graphics can be drawn, the window, viewport, and the transformation between them must be set up so that everything can be drawn correctly.

The best way to set up the window, the viewport, and the transformation in NCAR Graphics is to call the SPPS routine SET. SET sets up the normalization transformation between the user coordinate system and the normalized device coordinate (NDC) system of the viewport. In other words, you call the SPPS routine SET to tell the graphics package the limits of your data space and where on the screen or paper you want your plot to be drawn.

Conpack has 15 different internal parameters that directly affect how Conpack makes a SET call: VPL, VPR, VPB, VPT, VPS, WDL, WDR, WDB, WDT, XC1, XCM, YC1, YCN, SET, and MAP.

The simplest background you can make is to draw a perimeter around your plot using the Conpack routine CPBACK. This does not require you to use any of the above parameters. By default, CPBACK calls the SPPS routine SET for you. This option is covered in module "Cp 2.2 Drawing a contour perimeter."

The next easiest background you can make is to use XC1, XCM, YC1, and YCN to define the X and Y coordinates of your data in the user coordinate system. In this case, you probably want to let Conpack call the SPPS routine SET for you. This option is covered in module "Cp 2.3 Setting X/Y axis values for a contour background."

If you are drawing a map background with Ezmap, you need to follow the Ezmap chapter to set up your map. After the map background is initialized, you can tell Conpack to overlay your data on a map by setting the MAP parameter, then by setting your minimum and maximum latitude and longitude data locations using XC1, XCM, YC1, and YCN. In this case, Ezmap calls the SPPS routine SET for you, so you must tell Conpack that SET has already been called; you do this by setting the Conpack parameter SET. This option is covered in detail in module "Cp 3.8 Non-Cartesian data: Latitude/longitude data."

One of the most complex things that you can do with these options is to use a transformation to map your data to some other place on the plane by setting the Conpack parameter MAP. Then you might set XC1, YC1, XCM, YCN, WDB, WDT, WDL, and WDR to direct the drawing of contours in the window that you define. Setting the window parameters is very similar to setting the viewport parameters; see module "Cp 2.6 Contour perimeter: Size, shape, and location using Conpack."

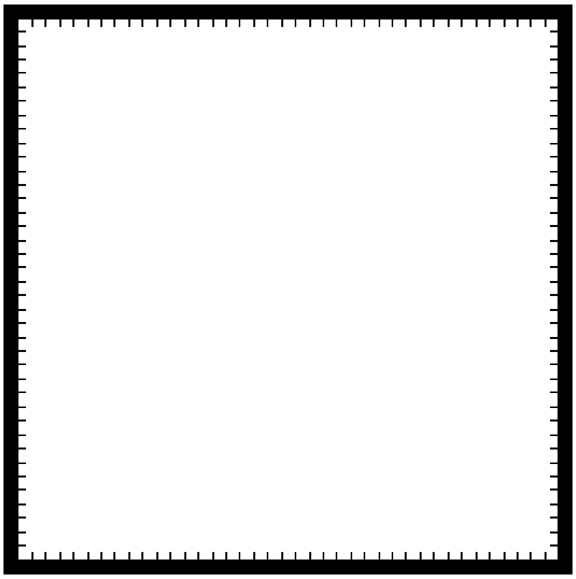
Occasionally, you may want to use another utility with Conpack, or you may want to call SET yourself. In this case, you simply set the Conpack parameter SET to zero, then make your call to the SPPS routine SET. This option is covered in module "Cp 2.7 Contour perimeter: Size, shape, and location using SET."

In all of the above situations where Conpack calls SET, you can set VPL, VPR, VPB, and VPT to move the plot around on the plotter frame, and you can use VPS to define the desired proportions of your contour plot. Moving your plot around on the plotter frame and changing its proportions is covered in module "Cp 2.6 Contour perimeter: Size, shape, and location using Conpack."

Cp 2.2 Drawing a contour perimeter

The Conpack routine CPBACK draws a perimeter around your plot only if you want the simple default rectangle. By default, CPBACK calls the SPPS routine SET for you. If you are using any mapping functions by setting the MAP parameter, or if you want labels for your tick marks, you need to draw your own perimeter using another NCAR Graphics utility such as Gridall.

Perimeter background for Conpack



Code segment from ccpback.f

```
1       CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
2       CALL CPBACK (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPBACK (Z, RWRK, IWRK)
```

Arguments

All three arguments to the CPBACK call are the arrays used in the last call to CPRECT, CPSPS1, or CPSPS2, the contents of which must not have been changed since that call.

Z

> *Real array, Input*---The data array.

RWRK

> *Real array, Workspace*---The real workspace array. The dimensions of RWRK are set by CPRECT, CPSPS1, or CPSPS2.

IWRK

> *Integer array, Workspace*---The integer workspace array. The dimensions of IWRK are set by CPRECT, CPSPS1, or CPSPS2.

Discussion

By default, CPBACK calls the SPPS routine SET. Then it calls the Gridall routine PERIM to draw the perimeter. Modules "Cp 2.4 Changing perimeter options" and "Cp 2.5 Labeling X/Y axis values for a contour background" demonstrate how you can call Gridall directly to draw a perimeter and label tick marks.

Line 1 of the **ccpback.f** code segment initializes Conpack with a call to CPRECT. This call is thoroughly discussed in module "Cp 3.2 Dense gridded data" and is mentioned here as a reminder that CPBACK must not be called before initializing Conpack. Line 2 calls CPBACK to draw a perimeter around the contours (contours are not drawn in this example).

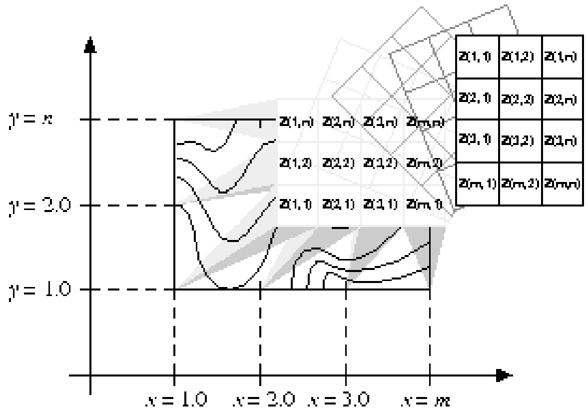
Perimeter attributes such as line width, line pattern, line color, and so forth can be changed with the same parameters you use with contour lines. Please see the line attributes modules Cp 4.11 through Cp 4.14.

If you have non-Cartesian data, you need to set the MAP parameter to some nonzero value by using the CPSETI routine. Setting the MAP parameter has the side effect of causing CPBACK to do nothing. Hence, if you have non-Cartesian data, and you want a perimeter around your plot, you need to draw it with the GKS routine GPL, the Dashline routine CURVED, the Gridall utility, or the Autograph utility.

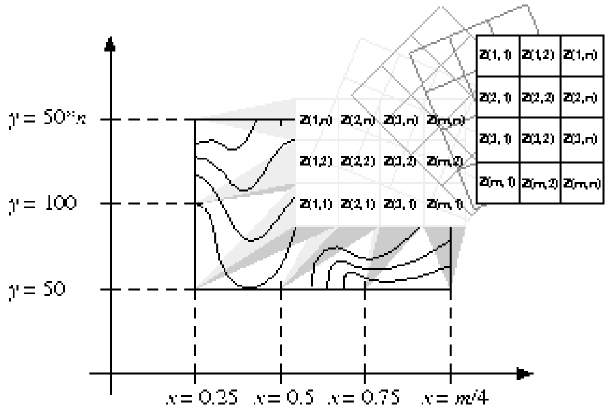
Cp 2.3 Setting X/Y axis values for a contour background

When plotting real data, it is often necessary to know what portion of the X/Y plane your data cover. Conpack has four parameters that set minimum and maximum values for X and Y. The following illustrations show the relationship between the Cartesian plane and data array indices.

Default mapping



User mapping



Synopsis

```
CALL CPSETR ('XC1', XMIN)
CALL CPSETR ('XCM', XMAX)
CALL CPSETR ('YC1', YMIN)
CALL CPSETR ('YCN', YMAX)
```

Arguments

XC1

Real---The X Coordinate at index 1 parameter is the X coordinate value that corresponds to a value of 1 for the first subscript of the data array. Alternatively, you can think of XC1 as being the leftmost X data location. If XC1=XCM, then XC1 gets the value 1.0.

XCM

Real---The X Coordinate at index M parameter is the X coordinate value that corresponds to a value of 'ZDM' for the first subscript of the data array. Alternatively, you can think of XCM as the rightmost X data location. If XC1=XCM, then XCM gets the value REAL(ZDM), where ZDM is the first dimension of the array of dense gridded or interpolated data to be contoured.

YC1

Real---The Y Coordinate at index 1 parameter is the Y coordinate value that corresponds to a value of 1 for the second subscript of the data array. Alternatively, you can think of YC1 as being the bottom Y data location. If YC1=YCN, then YC1 gets the value 1.0.

YCN

Real---The Y Coordinate at index N parameter is the Y coordinate value that corresponds to a value of 'ZDN' for the second subscript of the data array. Alternatively, you can think of YCN as the top Y data location. If YC1=YCN, then YCN gets the value REAL(ZDN), where ZDN is the second dimension of the array of dense gridded or interpolated data to be contoured.

Discussion

The preceding parameters associate X and Y data locations with the subscripts of your Z data array. By default, the lower left corner of the plot is at (1.0, 1.0), and the upper right corner of the plot is at (REAL(ZDM), REAL(ZDN)), where ZDM and ZDN represent the X and Y dimensions of the data array respectively.

In the default mapping illustration, note that XC1=1.0, XCM=REAL(*m*), YC1=1.0, and YCN=REAL(*n*). Also notice how the usual drawing of the Z array is rotated into the Cartesian plane.

In the user mapping illustration, the user sets XC1=0.25, XCM=*m*/4.0, YC1=50, and YCN=50**n*. In this case, the data are both translated on the X/Y plane and stretched to fit. Any real values for XC1, XCM, YC1, and YCN are allowable, making any translation from a data array to the Cartesian plane possible.

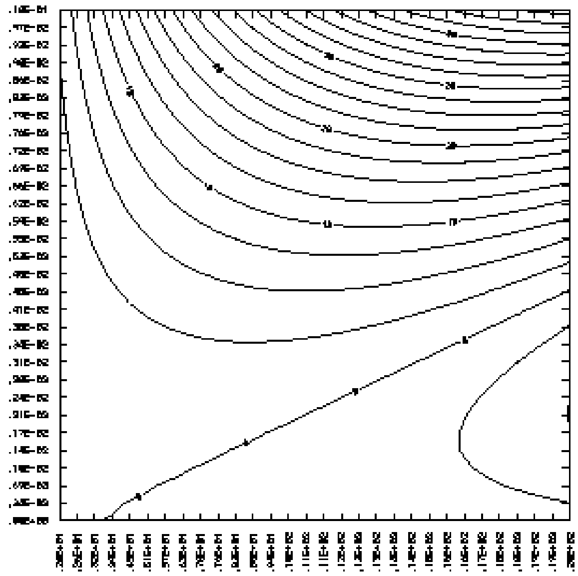
In cases where XC1>XCM, the mapping will be mirror imaged in the X direction.

In cases where YC1>YCN, the mapping will be mirror imaged in the Y direction.

Cp 2.4 Changing perimeter options

You can label tick marks or change the sizes and spacing of the tick marks on your perimeter, but these functions are not available from Conpack. You must use the Gridall utility to change the appearance of perimeter tick marks.

Gridall perimeter



Code segment from ccpga.f

```
      CALL CPSETR ('XC1 - X COORDINATE AT INDEX 1', 2.0)
      CALL CPSETR ('XCM - X COORDINATE AT INDEX M', 20.0)
      CALL CPSETR ('YC1 - Y COORDINATE AT INDEX 1', 0.0)
      CALL CPSETR ('YCN - Y COORDINATE AT INDEX N', .01)
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL LABMOD ('(E7.2)', '(E7.2)', 0, 0, 10, 10, 0, 0, 1)
      CALL GRIDAL (K-1, 0, N-1, 0, 1, 1, 5, 0., 0.)
```

Synopsis

```
      CALL GRIDAL (MJRX, MNRX, MJRY, MNRY,IXLB, IYLB, IGPH, XINT, YINT)
```

Arguments

MJRX

Integer, Input---The number of major divisions (spaces between tick marks) in the X axes.

MNRX

Integer, Input---Determines the number of minor divisions in the X axes.

MJRY

Integer, Input---The number of major divisions in the Y axes.

MNRY

Integer, Input---Determines the number of minor divisions in the Y axes.

The meanings of these arguments depend on the current setting of the linear/log flag in SPPS. This flag is set by the final argument in the SET call.

If the axis is linear, MJRX specifies the number of major divisions of the X/Y axis, and MNRX specifies the number of minor divisions within each major division. These values specify the number of *spaces* between GRIDAL lines or ticks, not the number of lines or ticks. Counting those at the ends, there is always one *more* major tick than the number of major divisions specified by MJRX. There is always one *less* minor tick than the number of minor divisions specified by MNRX.

If the axis is logarithmic, major division points occur at a value 10*MJRX* times the previous point. So if the minimum and maximum X-axis values are 3. and 3000., and MJRX is 1, then the major division points are 3., 30., 300., and 3000. If MNRX<=10, nine minor divisions occur between major divisions. For example, between 3. and 30., there would be minor division points at 6., 9., 12., 15., 18., 21., 24., and 27. If MNRX>10., minor divisions are omitted.

IXLB

Integer, Input---Specifies X-axis appearance.

-1 No X axis is drawn. No X-axis labels.
0 X axis is drawn. No X-axis labels.
1 X axis and X-axis labels are drawn.

IYLB

Integer, Input---Specifies Y-axis appearance.

-1 No Y axis is drawn. No Y-axis labels.
0 Y axis is drawn. No Y-axis labels.
1 Y axis and Y-axis labels are drawn.

IGPH

Integer, Input---Specifies the background type as indicated by one of the following integers:

IGPH	X axis	Y axis
0	grid	grid
1	grid	perimeter
2	grid	axis
4	perimeter	grid
5	perimeter	perimeter
6	perimeter	axis
8	axis	grid
9	axis	perimeter
10	axis	axis

XINT,YINT

Real, Input---The user "world" coordinates of the point of intersection of the two axes when IGPH equals 10. For other values of IGPH for which one of the axes is the axis type, XINT and/or YINT specifies the position of that axis.

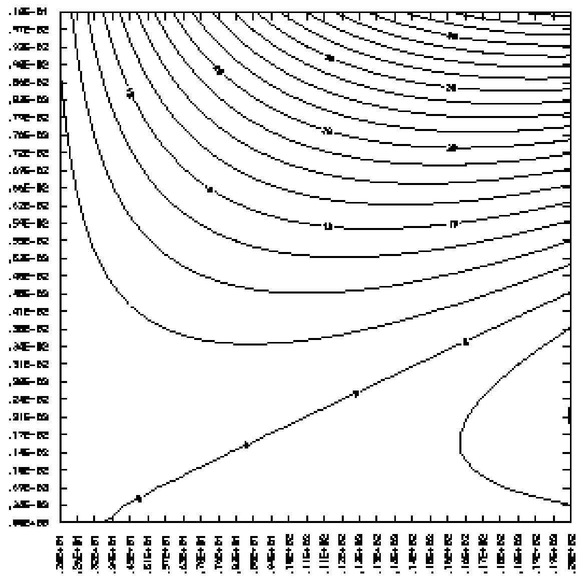
Discussion

Lines 1 through 4 of the **ccpga.f** code segment set the minimum and maximum X and Y locations for the data using Conpack routine CPSETR. In line 5, CPRECT takes these data locations and uses them to make a SET call that can be used by GRIDAL. The GRIDAL call in line 7 sets the number of major tick marks to one less than the number of rows and columns in the data array to simulate the results of CPBACK and to denote locations of data values. MNRX and MNRY are set to zero to turn off the minor tick mark option and mimic CPBACK. IXLB and IYLB are set to 1 to draw and label X and Y axes, and IGPH is set to 5 to draw a perimeter around the contours. Since we are drawing a perimeter, XINT and YINT are dummy parameters. The next module discusses the effect of line 6.

Cp 2.5 Labeling X/Y axis values for a contour background

If you are drawing a contour plot in Cartesian space, there are several ways to label your X/Y axes, and to change options on your tick marks, none of which involve Conpack. The easiest method is to use the two Gridall routines GRIDAL and LABMOD to draw and label your perimeter.

Labeled X/Y values



Code segment from ccpga.f

```
1        CALL CPSETR ('XC1 - X COORDINATE AT INDEX 1', 2.0)
2        CALL CPSETR ('XCM - X COORDINATE AT INDEX M', 20.0)
3        CALL CPSETR ('YC1 - Y COORDINATE AT INDEX 1', 0.0)
4        CALL CPSETR ('YCN - Y COORDINATE AT INDEX N', .01)
5        CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
6        CALL LABMOD ('(E7.2)', '(E7.2)', 0, 0, 10, 10, 0, 0, 1)
```

```
7       CALL GRIDAL (K-1, 0, N-1, 0, 1, 1, 5, 0., 0.)
```

Synopsis

```
CALL LABMOD (FMTX, FMTY, NUMX, NUMY, ISZX, ISZY, IXDC, IYDC, IXOR)
```

Arguments

FMTX

Character expression, Input---Contains format specifications for the X-axis numerical labels produced by the Gridall routine GRIDAL. The specification must begin with a left parenthesis and end with a right parenthesis, and it must not be more than ten characters long. Conversions of types E, F, G, and I are allowed; for example, you might use (F8.2). The default is (E10.3).

FMTY

Character expression, Input---Contains format specifications for the Y-axis numerical labels produced by GRIDAL. The specification must begin with a left parenthesis and end with a right parenthesis, and it must not be more than ten characters long. Conversions of types E, F, G, and I are allowed; for example, you might use (E10.0). The default is (E10.3).

NUMX

Integer, Input---The number of characters in each X-axis numeric label.

0 The label is the substring LBLX(*m*:*n*), where LBLX(*m*:*m*) is the first nonblank character in LBLX, and LBLX(*n*:*n*) is the last nonblank character following LBLX(*m*:*m*). This is the default.

<>0 The number of characters in each X-axis numeric label; if LBLX is a string produced by the format FMTX, then the label is the substring LBLX(1:NUMX). This causes the labels to be centered differently than if a zero value is used.

NUMY

Integer, Input---Same as NUMX, except it is used for Y-axis labels.

ISZX, ISZY

Integer, Input---Character sizes for the labels, specified in 1024ths of a screen width, just as for the SPPS routine WTSTR. The special values of 0, 1, 2, and 3 produce very small, small, medium, and large character sizes. The default value for both is 10, which means 10/1024ths of a screen width. If you have called SETI, ISZX and ISZY are not 1024ths of a screen width, they are some other fraction of a screen width as determined by the arguments of SETI.

IXDC

Integer, Input---The distance, in 1024ths of a screen width, from the left edge of the current viewport to the label specified by FMTY, NUMY, and ISZY. There are two special values for IXDC:

0 The Y-axis labels end 20/1024ths of a screen width from the left of the viewport. This is equivalent to setting IXDC=20.

1 Y-axis labels begin 20/1024ths of a screen width from the right of the viewport. This is equivalent to setting IXDC=20*w*, where *w*/1024 is the width of the viewport in NDCs.

The default value is 20.

Negative values of IXDC specify the distance from the right edge of the viewport.

When GRIDAL is called with IGPH=2, 6, or 10, IXDC is the distance from the Y axis, rather than from the minimum viewport coordinate, and the special values 0 and 1 are equivalent to 20 and 20.

IYDC

Integer, Input---The distance, in 1024ths of a screen width, from the bottom edge of the current viewport to the label specified by FMTX, NUMX, and ISZX. There are two special values for IYDC:

0 The X-axis labels end 20/1024ths of a screen width (0.02 NDCs) below the viewport. This is equivalent to setting IYDC=20.

1 The X-axis labels begin 0.02 NDCs above the viewport. This is equivalent to setting IYDC=20*h*, where *h*/1024 is the height of the viewport in NDCs.

The default value is 20.

Negative values of IYDC specify the distance from the top edge of the viewport.

When GRIDAL is called with IGPH=8, 9, or 10, IYDC is the distance from the X axis, rather than from the minimum viewport coordinate, and the special values 0 and 1 are equivalent to 20 and 20.

IXOR

Integer, Input---Specifies the orientation of the X-axis labels.

0 Horizontal. This is the default.

1 Vertical.

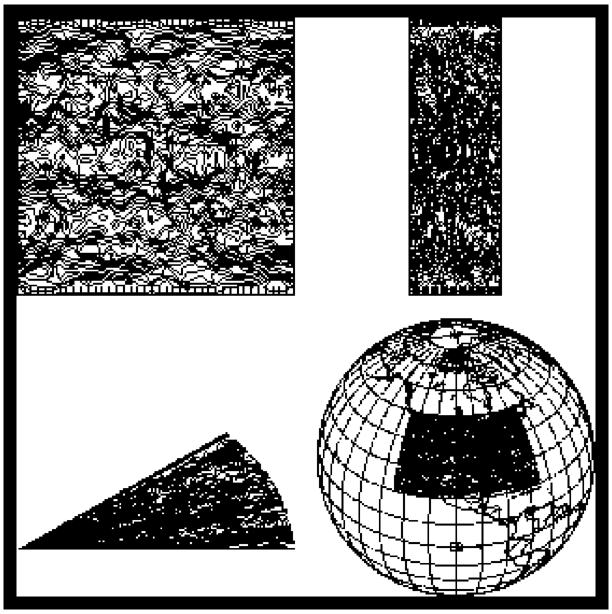
Discussion

Line 6 of the **ccpga.f** code segment sets FMTX and FMTY using a good numerical format for the X and Y axis values. NUMX and NUMY are set to 0 to produce a simple numeric label. ISZX and ISZY are set to 10 to show off the labels. IXDC and IYDC are set to the default 0 because label location is not critical here, and IXOR is set to 1 so that X-axis labels don’t overlap. Further discussion of the LABMOD routine appears in Chapter 15 "Drawing axes, perimeters, and grids" of the *NCAR Graphics Fundamentals* guide.

Cp 2.6 Contour perimeter: Size, shape, and location using Conpack

It is often desirable to put more than one contour plot on a page, or to change the size of a contour plot so that it can fit into a publication. Conpack offers four parameters, VPL, VPR, VPB, and VPT, to allow you to place the contour plot anywhere on the plotter frame. A fifth parameter, VPS, allows you to specify the shape of your contour rectangle.

Contour viewport



Code segment from ccpvp.f

```
      CALL CPSETR ('VPL - VIEWPORT LEFT', 0.02)
      CALL CPSETR ('VPR - VIEWPORT RIGHT', 0.48)
      CALL CPSETR ('VPB - VIEWPORT BOTTOM', 0.52)
      CALL CPSETR ('VPT - VIEWPORT TOP', 0.98)
      CALL CPSETR ('VPL - VIEWPORT LEFT', 0.52)
      CALL CPSETR ('VPR - VIEWPORT RIGHT', 0.98)
      CALL CPSETR ('VPB - VIEWPORT BOTTOM', 0.52)
      CALL CPSETR ('VPT - VIEWPORT TOP', 0.98)
      CALL CPSETR ('VPS - VIEWPORT SHAPE', -0.33)
      CALL CPSETR ('VPL - VIEWPORT LEFT', 0.02)
      CALL CPSETR ('VPR - VIEWPORT RIGHT', 0.48)
      CALL CPSETR ('VPB - VIEWPORT BOTTOM', 0.02)
      CALL CPSETR ('VPT - VIEWPORT TOP', 0.48)
      CALL CPSETR ('VPS - VIEWPORT SHAPE', -1.5)
```

Synopsis

```
CALL CPSETR ('VPL', vpl)
CALL CPSETR ('VPR', vpr)
CALL CPSETR ('VPB', vpb)
CALL CPSETR ('VPT', vpt)
CALL CPSETR ('VPS', vps)
```

Arguments

VPL, VPR, VPB, VPT

Real---The ViewPort Left, ViewPort Right, ViewPort Bottom, and ViewPort Top parameters, respectively. These parameters specify a rectangle within which the viewport is to be placed. They are always in Normalized Device Coordinate (NDC) units. In other words: 0.0<=VPL, VPR, VPB, VPT<=1.0. By default, VPL=VPB=.05, and VPR=VPT=.95.

VPS

Real---The ViewPort Shape parameter is used only when Conpack calls the SPPS routine SET (that is, when the Conpack parameter SET<>0). VPS specifies viewport shape as follows:

<0.0
Specifies the exact shape of the contour rectangle. ABS(VPS) = *width* / *height*, where *width* is the desired width of the rectangle and *height* is the desired height of the rectangle.

0.0
Specifies a contour rectangle completely filling VPL, VPR, VPB, and VPT.

0<VPS<1
Specifies a plot of the shape determined by XC1, XCM, YC1, and YCN, reverting to the shape specified by VPL, VPR, VPB, and VPT if the ratio of the shorter side to the longer side would be less than the value specified. By default, VPS=0.25.

>=1.0
Specifies a plot of the shape determined by XC1, XCM, YC1, and YCN, reverting to a square if the ratio of the longer side to the shorter side would be greater than VPS.

Whatever its shape, the contour rectangle is made as large as possible and centered in the viewport specified by VPL, VPR, VPB, and VPT.

Discussion

The viewport parameters are only used when the Conpack parameter SET<>0, so that Conpack calls the SPPS routine SET.

In the **ccpvp.f** program, there are calls to Conpack routines that draw contours between lines 4 and 5, lines 9 and 10, and lines 14 and 15 of the code segment. Lines 1 through 4 of the **ccpvp.f** code segment set up the viewport so that the first plot is drawn in the upper left corner. Lines 5 through 8 put the second plot in the upper right corner. Line 9 specifies a contour rectangle three times as high as it is wide.

Lines 10 through 13 tell Conpack to draw contours in the lower left corner, and line 14 specifies a contour rectangle wider than it is tall. This contour drawing uses rho/theta data; this is why it draws contours in a wedge rather than a square.

Contours in the lower right corner are drawn without setting VPR, VPL, VPB, or VPT because the viewport is set up by Ezmap.

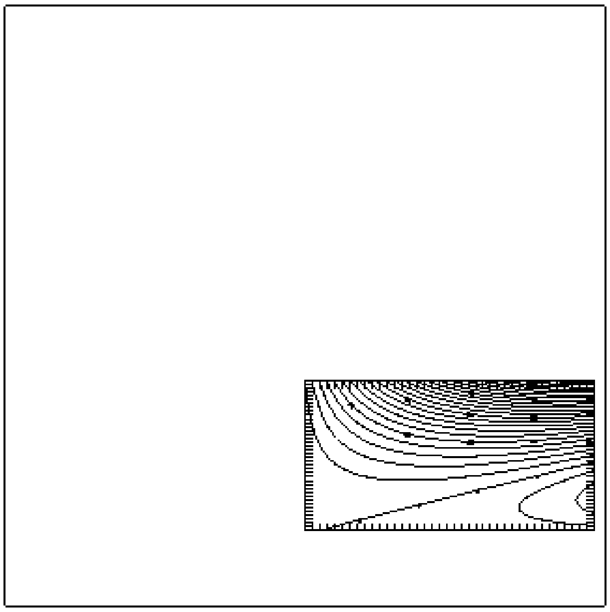
Exercises

1. Give each contour rectangle in the **ccpvp** example a different shape using each of the different range values of VPS.
2. Change the viewports so that the contour rectangles overlap by a small amount in the center of the plot.
3. Make each plot display full size on a different frame.

Cp 2.7 Contour perimeter: Size, shape, and location using SET

The second way to adjust contour perimeter size, shape, and location is to use the SPPS routine SET. By default, Conpack uses a rectangle of the shape implied by the dimensions of your data array, and makes that rectangle as large as possible within the viewport. Occasionally, when your data array defines a long, narrow rectangle, Conpack will use the entire viewport as the contour rectangle. You can call the SPPS routine SET directly to define the location, shape, and size of the contour rectangle.

Defining contour rectangle shape and location



Code segment from ccpset.f

```
      CALL SET (0.5, 0.98, 0.125, 0.375, 0., 2., 0., 1., 1)
      CALL CPSETI ('SET - DO-SET-CALL FLAG', 0)
      CALL CPSETR ('XC1 - X COORDINATE AT INDEX 1', 0.)
      CALL CPSETR ('XCM - X COORDINATE AT INDEX M', 2.)
      CALL CPSETR ('YC1 - Y COORDINATE AT INDEX 1', 0.)
      CALL CPSETR ('YCN - Y COORDINATE AT INDEX N', 1.)
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('SET', iset)
CALL SET (VL, VR, VB, VT, WL, WR, WB, WT, LS)
```

Arguments

SET

Integer---The doSETcall flag is a Conpack parameter that determines whether you call the SPPS routine SET or Conpack calls SET for you.

0 Conpack does not call SET. When you call SET, arguments 5 through 8 (the window coordinates) must be consistent with the range of coordinates specified by MAP, XC1, XCM, YC1, and YCN.
1 A Conpack initialization routine calls SET. This is the default.

VL, VR, VB, VT

Real, Input---These arguments for the SPPS routine SET are values that define the left, right, bottom, and top edges of the viewport or a rectangle in NDC units. These coordinates determine where the plot is placed on your output device. These are values between 0.0 and 1.0, inclusive.

WL, WR, WB, WT

Real, Input---These arguments for the SPPS routine SET are values that define the left, right, bottom, and top edges of a window or rectangle in the user coordinate system. These coordinates determine what part of your data is visible in the plot.

LS

Integer, Input---This argument for the SPPS routine SET specifies linear/logarithmic mapping options for the X and Y axes:
1 X linear, Y linear
2 X linear, Y logarithmic
3 X logarithmic, Y linear
4 X logarithmic, Y logarithmic

Discussion

The SPPS routine SET is described more thoroughly in the *NCAR Graphics Fundamentals* guide. SET allows for logarithmic scaling with the LS argument, and mirror imaging by swapping the values for the bottom and top or the left and right window arguments. If you want to use log scaling in your contour plot, or if you want to use some other utility before drawing your contours, then you may need to call SET directly. Before calling SET yourself, you should study the section "Cp 2. Backgrounds for your plots" to gain a clear understanding of how Conpack interacts with the SET routine.

The call to SET in line 1 of the **ccpset.f** code segment defines the portion of the plotter frame used for the plot. Line 2 uses the Conpack parameter SET to tell Conpack that subroutine SET has already been called. Lines 3 through 6 place the contour rectangle in the X/Y plane. XC1, XCM, YC1, and YCN are the same as WL, WR, WB, and WT in the SET call because this is where user coordinates are defined for Conpack and SET. Contours are then initialized and drawn normally. Because CPRECT, CPSPS1, and CPSPS2 all use information specified in the CPSET*x* calls, it is best to set general parameters before calling the Conpack initialization routines.

Exercises

1. Run the **ccpset** example to produce a copy of **ccpset.f**, then modify it to move the contour rectangle into the upper left corner of the frame.

2. Modify **ccpset.f** so that the plot fills the whole frame, and have Conpack draw in a square contour rectangle.
3. In **ccpset.f**, comment out the line:

   ```
   CALL CPSETI ('SET', 0)
   ```

   and explain the results.

Cp 3. Initializing Conpack

The way you initialize Conpack is highly dependent on the type of data you have. Fortunately, after determining your data type, you can initialize Conpack with a single subroutine call as one of the first steps to produce your contour plot.

Conpack functional outline

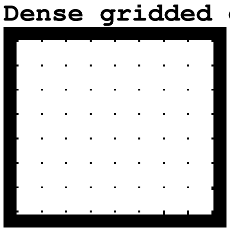
```
-----------------------------------------------------------
    1.  Open GKS
    2.  Set window and viewport
    3.  Put in titles and other labels
    4.  Set up general parameters
    5.  Set up area maps
 *  6.  Initialize Conpack
    7.  Force Conpack to choose contour levels if necessary
    8.  Modify Conpack-chosen parameters
    9.  Draw labels
    10.  Draw background
    11.  Draw contour lines
    12.  Call FRAME
    13.  Close GKS
-----------------------------------------------------------
```

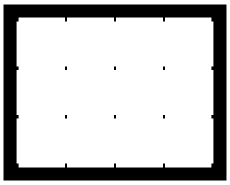
* Steps discussed in this section.

Cp 3.1 Data types

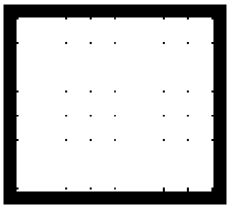
Before Conpack can draw anything, it must know what kind of data you have. In some cases you must do some preprocessing to get your data into an acceptable form. The next several modules discuss initializing Conpack with the following data types. Decide what kind or kinds of data you have, then turn to the relevant modules.

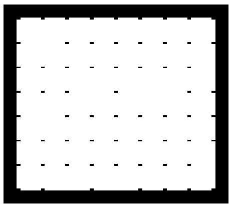
Dense gridded data **Sparse gridded data**



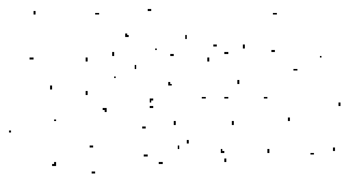


Irregularly spaced gridded data **Missing data**

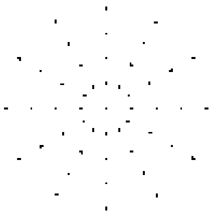




Nongridded data



Non-Cartesian data **Non-Cartesian data**



Dense gridded data

Dense gridded data implies that you have data points at regular intervals on a rectangular grid. If you have dense gridded data, you should use CPRECT to initialize Conpack.

Sparse gridded data

Sparse gridded data implies that you have data points on a rectangular grid, but that the grid has large spacing between gridlines. This is the effect of having a tick-tack-toe grid versus 1/4-inch graph paper. If you were to contour your data using CPRECT, you would find that the contours were long straight line segments that join with sharp corners at data points. CPSPS1 is a smoother that interpolates your data onto a dense grid, and initializes Conpack.

Generally, if your data set has less than 35 data points in either direction, it is better to use CPSPS1. However, this distinction is very subjective. Most modelers define "sparse" for global datasets as a spacing of 2 degrees or more between data points.

Irregularly spaced gridded data

Irregularly spaced gridded data is frequently gathered by satellites, or generated by large models. In essence, the data lie on a grid, but the grid may have very wide spacing in some places and very narrow spacing in others. Because CPSPS2 interpolates from the irregular grid onto a regular grid, it doesn't matter if your data are sparse or dense.

Nongridded data

Nongridded data, or randomly spaced data, has no relationship to a grid at all. The effect is as if you took a handful of rice and tossed it down on a piece of paper, then used the locations of the rice grains as your data points. Before Conpack can use this kind of data, you must interpolate the data onto a grid.

Bivar, an interpolation package that works with Conpack, is distributed with NCAR Graphics. Bivar can be used on data that do not have abrupt changes (like ridges or cliffs on a topographic map). In other words, Bivar assumes that the function describing the data has a continuous first derivative. If your data do not fit this description, or if you have an interpolation package that fits your data better, don't use Bivar.

Missing data

Missing data points can occur in any gridded data; this has no bearing on which of the initialization routines are used. All that is necessary is to use the SPV parameter to tell Conpack what data value signals a missing data point.

Non-Cartesian data

Non-Cartesian data are data that do not lie on an X/Y coordinate plane. Conpack has transformations for latitude/longitude data, and for polar coordinate data. Conpack also has a mechanism for dealing with other kinds of data.

Latitude/longitude data is fairly common, and Conpack is designed so that it is easy to overlay lat/lon data over a map.

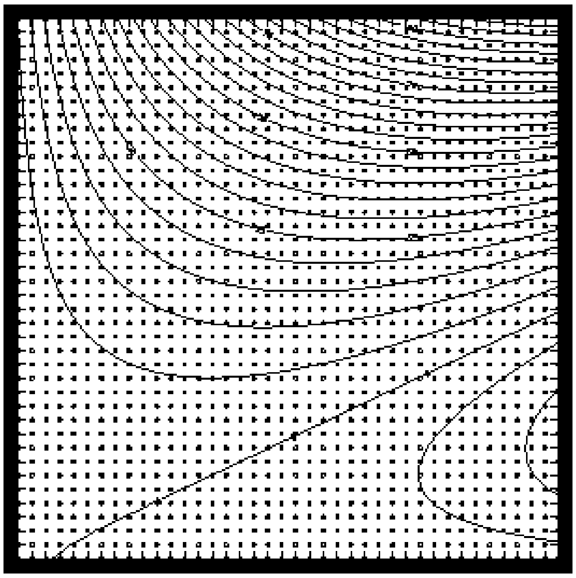
Conpack can also easily be used to display polar coordinate data, using a method similar to the lat/lon display method.

Almost any data that can be transformed into data on a rectangular grid can be contoured by Conpack by writing a transformation, and adding it to the CPMPXY subroutine provided by Conpack.

Cp 3.2 Dense gridded data

Dense gridded data implies that you have data points at regular intervals on a rectangular grid. If you have dense gridded data, you should use CPRECT to initialize Conpack. CPRECT initializes the internal pointers that are used to manage workspace; it decides appropriate ranges for X and Y coordinates used to draw contour lines.

Dense gridded data



Code segment from ccprect.f

```
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
      CALL MARK (M, N)
```

Synopsis

```
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
```

Arguments

Z(K,***n***)

> *Real array, Input*---The array of data values to be contoured. Z is dimensioned K by *n*, where *n*>=N.

K

> *Integer, Input*---The first dimension of the array Z.

M

> *Integer, Input*---The first dimension of the array of data in Z. M<=K.

N

> *Integer, Input*---The second dimension of the array of data in Z. N<=*n*, where *n* is the declared second dimension of the array Z.

RWRK(LRWK)

Real array, Workspace---A real work array of length LRWK.

LRWK

Integer, Input---The length of RWRK. Generally, a good value to start with is 2500.

IWRK(LIWK)

Integer array, Workspace---An integer work array of length LIWK.

LIWK

Integer, Input---The length of IWRK. Generally, a good value to start with is 1000.

Discussion

Line 1 of the **ccprect.f** code segment initializes Conpack with a call to CPRECT. Notice that the call to CPRECT comes before the call to draw the background with CPBACK in line 2. As discussed in module "Cp 2.2 Drawing a contour perimeter," this is necessary because Conpack uses the data information to decide how to shape the contour rectangle. Line 3 draws contours with CPCLDR. Line 4 marks the data locations with example subroutine MARK.

Occasionally you may see error messages similar to this:

```
CPGRWS      200 WORDS REQUESTED, 100
WORDS AVAILABLE
CPGIWS      200 WORDS REQUESTED, 100
WORDS AVAILABLE
```

CPGRWS tells you that your real workspace is too small, and CPGIWS tells you that your integer workspace is too small. Increase LRWK or LIWK to the number of words requested in the error message, and repeat until the error message goes away.

The size of LRWK and LIWK depends on the complexity of your contour plot, so plots with lots of small, tight contour levels require more space than large, loose contour plots like **ccprect**. If you are running a job with lots of similar plots, and need to minimize workspace, see module "Cp 3.12 Setting minimal workspace" for the most reliable method of optimizing these array sizes.

Exercises

1. Try running the **ccprect** example using the function:

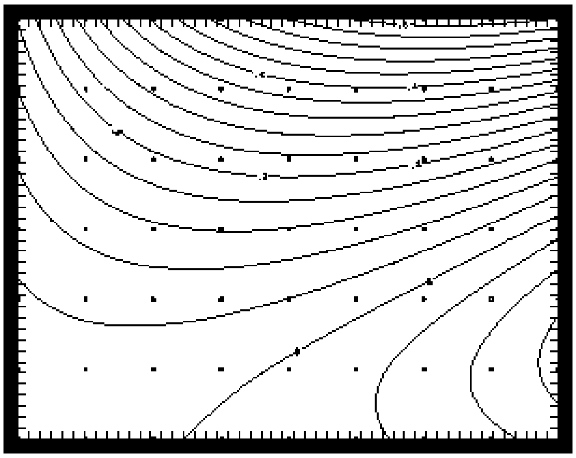
   ```
   Z(I,J) = sin(1./real(i*j)) * sin(real(i*j)) *

   cos(real(i)/real(j)) + cos(real(i*j))
   ```

   in subroutine GETDAT instead.

Cp 3.3 Sparse gridded data

Sparse gridded data implies that you have data points on a rectangular grid, but that the grid has large spacing between gridlines. This is like having a tick-tack-toe grid versus 1/4-inch graph paper. If you contour sparse data with CPRECT, the contours would be long and straight with sharp corners at data points. CPSPS1 is a smoother that interpolates your data onto a dense grid, then initializes Conpack.

Sparse gridded data



Code segment from ccpsps1.f

```
      CALL CPSPS1 (Z, K, M, N, RWRK, LRWK, IWRK, LIWK, ZDAT, LZDT)
      CALL CPBACK (ZDAT, RWRK, IWRK)
      CALL CPCLDR (ZDAT, RWRK, IWRK)
      CALL MARK (X, Y, M, N)
```

Synopsis

```
      CALL CPSPS1 (ZSPS, K, M, N, RWRK, LRWK, IWRK, LIWK, ZREG, LZRG)
```

Arguments

ZSPS(K,***n***)

> *Real array, Input*---The sparse array of data from which the dense array is to be generated. Dimensioned K by *n*, where *n*>=N.

K

> *Integer, Input*---The first dimension of the array ZSPS.

M

> *Integer, Input*---The first dimension of the sparse array of data in ZSPS. M<=K.

N

Integer, Input---The second dimension of the sparse array of data in ZSPS. N<=*n*, where *n* is the declared second dimension of the array ZSPS.

RWRK(LRWK)

Real array, Workspace---The real work array.

LRWK

Integer, Input---The length of RWRK. Should be at least

```
100 * ((NSPV+99)/100) + 3 * M *  N + MAX (M + N + N, 4 * IZDM)
```

where NSPV is the number of special or missing values in the sparse array, and IZDM is the first dimension of the dense array.

IWRK(LIWK)

Integer array, Workspace---Integer work array.

LIWK

Integer, Input---The length of IWRK. Should be at least

```
100 * ((NSPV+99)/100)
```

ZREG(LZRG)

Real array, Output---The densely gridded array to which the interpolated data are returned.

LZRG

Integer, Input---The length of ZREG.

Discussion

Line 1 of the **ccpsps1.f** code segment initializes Conpack with a call to CPSPS1. Note that this call comes before the call to draw the background with CPBACK in line 2. As discussed in module "Cp 2.2 Drawing a contour perimeter," this is necessary because Conpack uses the data information to decide how to shape the contour rectangle. Line 3 calls CPCLDR with ZDAT, the interpolated data, rather than with the uninterpolated data. Line 4 marks the data locations with example subroutine MARK.

If your sparse array defines a long, narrow rectangle, Conpack chooses a roughly square rectangle of contours for you. It is possible to change this by specifying the dimensions of the interpolated array ZREG, using the parameters ZDS, ZDM, and ZDN. For more information on changing the shape of your contour rectangle, see the contour perimeter modules Cp 2.6 and Cp 2.7.

By default, both CPSPS1 and CPSPS2 use bicubic splines under tension to fit the sparse array data values. The routines used are from Fitpack, by Alan Cline. Spline tension can be changed with the T2D parameter. More information appears in module "Cp 9.1 Smoothing contours."

Occasionally you may see error messages like this:

```
CPGRWS     200 WORDS REQUESTED, 100
WORDS AVAILABLE
CPGIWS     200 WORDS REQUESTED, 100
WORDS AVAILABLE
```

CPGRWS tells you that your real workspace is too small, and CPGIWS tells you that your integer workspace is too small. Increase LRWK or LIWK to the number of words requested in the error message, and repeat until the error message goes away.

CPSPS1 cannot extrapolate data beyond the boundaries of the data field.

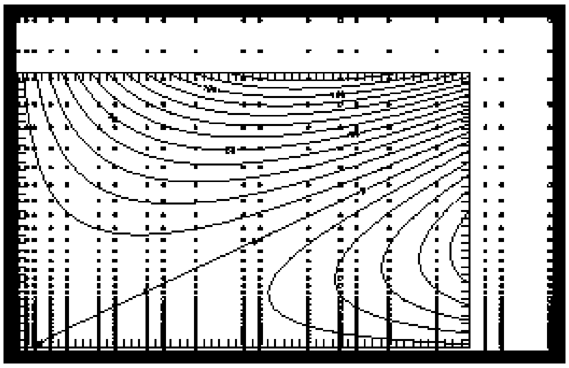
Exercises

1. Modify the **ccpsps1** example so that it calls CPRECT instead of CPSPS1. Notice the differences in the two plots.

Cp 3.4 Irregularly spaced gridded data

Irregularly spaced gridded data is the kind of data most frequently gathered by satellites, or generated by large models. In essence, the data lie on a grid, but the grid may have very wide spacing in some places and very narrow spacing in others. CPSPS2 interpolates from a sparse rectangular grid which is spaced irregularly in X and Y to a dense rectangular grid; it then initializes contouring of the data in the dense grid. CPSPS2 may be viewed as a data-smoothing routine.

Irregularly spaced gridded data



Code segment from ccpsps2.f

```
      CALL CPSPS2 (X, Y, Z, M, M, N, RWRK, LRWK, IWRK, LIWK, ZREG, LZRG)
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL CPCLDR (ZREG, RWRK, IWRK)
      CALL MARK (X, Y, M, N)
```

Synopsis

```
CALL CPSPS2 (X, Y, Z, K, M, N, RWRK, LRWK, IWRK, LIWK, ZREG, LZRG)
```

Arguments

X(M)

Real array, Input---The array of X coordinates of the irregular rectangular grid. These must be in strictly increasing numerical order.

Y(N)

Real array, Input---The array of Y coordinates of the irregular rectangular grid. These must be in strictly increasing numerical order.

Z(K,***n***)

Real array, Input---The sparse array of data, from which the dense array is to be generated. Dimensioned K by *n*, where *n*>=N.

K

Integer, Input---The first dimension of the array Z.

M

Integer, Input---The first dimension of the sparse array of data in Z. M<=K.

N

Integer, Input---The second dimension of the sparse array of data in Z. N<=*n*, the declared second dimension of the array Z.

RWRK(LRWK)

Real array, Workspace---The real work array.

LRWK

Integer, Input---The length of RWRK.

```
LRWK \xb3  100 * ((NSPV+99)/100) + 3 * M * N + M + N + N)
```

where NSPV is the number of special values or missing data points in the sparse array, as determined by the SPV parameter.

IWRK(LIWK)

Integer array, Workspace---The integer work array.

LIWK

Integer, Input---The length of IWRK.

```
LIWK >= 100 * ((NSPV+99) / 100)
```

ZREG(LZRG)

Real array, Output---The densely gridded array to which the interpolated data are returned.

LZRG

Integer, Input---The length of ZREG. The larger you dimension ZREG, the more interpolated data points will be generated, giving smoother curves.

```
LZRG = MAX (1600, M * N)
```

is a good size to start with.

Discussion

Line 1 of the **ccpsps2.f** code segment initializes Conpack with a call to CPSPS2. Notice that the call to CPSPS2 comes before the call to draw the background with CPBACK in line 2. This is necessary because Conpack uses the data information to decide how to shape the contour rectangle. Note that line 3 calls CPCLDR with ZREG, the interpolated data, rather than with the uninterpolated data. Line 4 marks the data locations with example subroutine MARK.

If your sparse array defines a long, narrow rectangle, Conpack chooses a roughly square rectangle of contours for you. It is possible to change this by specifying the dimensions of the interpolated array ZREG, using the parameters ZDS, ZDM, and ZDN. For more information on changing the shape of your contour rectangle, see the contour perimeter modules Cp 2.6 and Cp 2.7.

By default, both CPSPS1 and CPSPS2 use bicubic splines under tension to fit the sparse array data values. The routines used are from Fitpack, by Alan Cline. Spline tension can be changed with the T2D parameter. More information appears in module "Cp 9.1 Smoothing contours."

Occasionally you may see error messages similar to this:

```
CPGRWS     200 WORDS REQUESTED, 100
WORDS AVAILABLE
CPGIWS     200 WORDS REQUESTED, 100
WORDS AVAILABLE
```

CPGRWS tells you that your real workspace is too small, and CPGIWS tells you that your integer workspace is too small. Increase LRWK or LIWK to the number of words requested in the error message, and repeat until the error message goes away.

CPSPS2 cannot extrapolate data beyond the boundaries of the data field.

Another common error is the failure to specify the arrays X and Y in strictly increasing order. When this happens, you see this error message:

```
ERROR    6 IN CPSPS2 - ERROR IN CALL TO MSSRF1
```

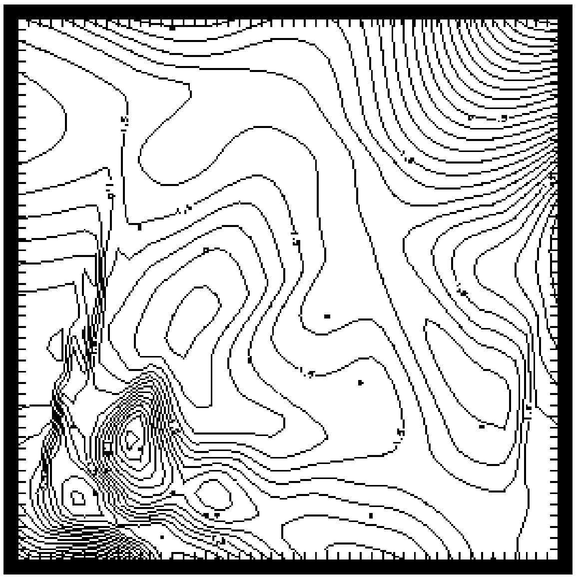
Exercises

1. Choose different values for the arrays X and Y in the **ccpsps2** example, then compile and run **ccpsps2.f** again. Did the shape of the contour rectangle change?
2. Try setting XC1, XCM, YC1, and YCN in **ccpsps2.f** to agree with X(1), X(M), Y(1), and Y(N). Note how the contour rectangle shape changes.

Cp 3.5 Nongridded data

Nongridded data, or randomly spaced data, are data that have no relationship to a grid at all. The effect is as if you took a handful of rice and tossed it down on a piece of paper, then used the locations of the rice grains as your data points. Before Conpack can use this kind of data, you must preprocess it with an interpolation package such as Bivar. IDSFFT is a Bivar subroutine that returns an array of dense gridded data for use with CPRECT, but it has some limitations.

Contouring with nongridded data



Code segment from cidsfft.f

```
      DO 101, I=2, MREG
         XREG(I) = XMIN + (XMAX-XMIN) * REAL(I-1)/(MREG-1)
101   CONTINUE
      DO 102, I=2, NREG
         YREG(I) = YMIN + (YMAX-YMIN) * REAL(I-1)/(NREG-1)
102   CONTINUE
```

```
      CALL IDSFFT (1, NRAN, XRAN, YRAN, ZRAN, MREG, NREG, MREG, XREG, YREG,
     + ZREG, IWRK, RWRK)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPCLDR (ZREG, RWRK, IWRK)
      CALL MARK (XRAN, YRAN, NRAN)
```

Synopsis

```
      CALL IDSFFT (MD, NRAN, XRAN, YRAN, ZRAN, MREG, NREG, KREG, XREG,
     +              YREG, ZREG, IWK, WK)
```

Arguments

MD

Integer, Input---Mode of computation (must be 1, 2, or 3).

1 Used if this is the first call to this subroutine, or if the value of NRAN has been changed from the previous call, or if the contents of the XRAN or YRAN arrays have been changed from the previous call.

2 Used if the values of NRAN, the XRAN array, and the YRAN array are unchanged from the previous call, but new values for XREG, YREG are being used. If MD=2 and NRAN has been changed since the previous call to IDSFFT, an error return occurs.

3 Used if the values of NRAN, MREG, NREG, XRAN, YRAN, XREG, and YREG are unchanged from the previous call, that is, if the only change on input to IDSFFT is in the ZRAN array. If MD=3 and NRAN, MREG, or NREG has been changed since the previous call to IDSFFT, an error return occurs.

Between the call with MD=2 or MD=3 and the preceding call, the IWK and WK work arrays should not be disturbed.

NRAN

Integer, Input---Number of random data points (must be 4 or greater).

XRAN(NRAN)

Real array, Input---Array of dimension NRAN containing the X coordinates of the data points.

YRAN(NRAN)

Real array, Input---Array of dimension NRAN containing the Y coordinates of the data points.

ZRAN(NRAN)

Real array, Input---Array of dimension NRAN containing the Z coordinates of the data points.

MREG

Integer, Input---Number of output grid points in the X direction (must be 1 or greater).

NREG

Integer, Input---Number of output grid points in the Y direction (must be 1 or greater).

KREG

Integer, Input---First dimension of ZREG as declared in the calling program. KREG>=MREG, else an error return occurs.

XREG(MREG)

Real array, Input---Array of dimension MREG containing the X coordinates of the output grid points.

YREG(NREG)

Real array, Input---Array of dimension NREG containing the Y coordinates of the output grid points.

ZREG(KREG,NREG)

Real array, Output---Real, densely gridded two-dimensional array of dimension (KREG, NREG), storing the interpolated Z values at the output grid points.

IWK(*)

Integer array, Workspace---Integer work array that must be dimensioned at least 31 * NRAN + MREG * NREG.

WK(*)

Real array, Workspace---Real work array of dimension at least 6 * NRAN.

Discussion

Lines 1 through 6 of the **cidsfft.f** code segment set up the grid locations XREG and YREG for the interpolated data values, ZREG. Without these points, the call to IDSFFT in line 7 is unable to produce a set of interpolated data values. Line 8 uses CPRECT with the interpolated data to initialize Conpack, and line 9 calls CPCLDR to draw the contours. Once again, subroutine MARK is used to show approximate locations for the nongridded data values.

Bivar requires that nongridded data points must be distinct and their projections in the X/Y plane must not be collinear. In other words, you can't have two Z values for the same X,Y pair, and your X,Y values can't all lie on the same line.

Inadequate work space in IWK and WK may cause incorrect results. If you decide to share the work arrays with Conpack work arrays, make sure that your integer work space is large enough. In general, IDSFFT needs more integer space than Conpack, and Conpack needs more real work space than IDSFFT. Also, Conpack relies on the fact that workspace arrays are not modified between calls to

Conpack routines, so IDSFFT must be called before calls to Conpack.

In versions of NCAR Graphics prior to Version 3.2, you must load your own copy of Bivar for use with nongridded data. First, execute:

    ncargex cbex01

This places several files in your directory. Edit **cbex01.f**, and remove all lines that precede the line:

```
C PACKAGE BIVAR
```

The remainder of the file is the Bivar package. In NCAR Graphics Version 3.2, Bivar resides in **libncarg.a**, so you only need to load the NCAR Graphics library to use Bivar.

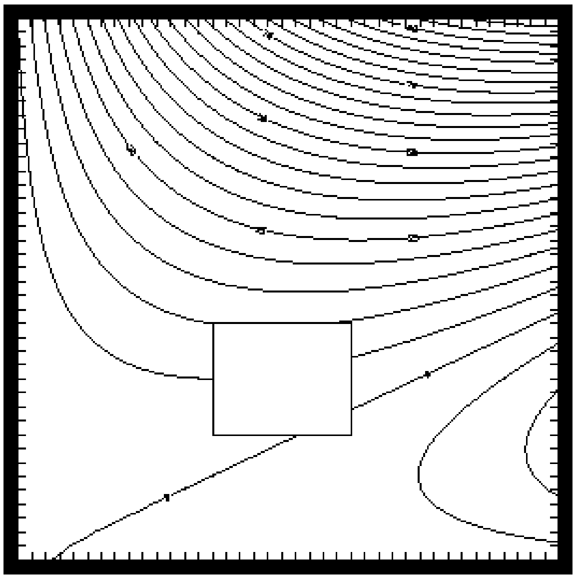
Exercises

1. Modify **cidsfft.f** so that XREG and YREG are not initialized, and see what happens when you run it.
2. Think about what kind of an interpolation package would be best for your data. If you decide that a bivariate interpolation package is not optimal, try running the **cidsfft** example with another interpolation package.

Cp 3.6 Missing or special data

Because any set of data can have missing or special data, Conpack has a parameter option that allows you to tag special data points. Once the special value parameter is set, initialize Conpack with CPRECT, CPSPS1, or CPSPS2, depending on what kind of data you have.

Marking missing or special data



Code segment from ccpspv.f

```
      CALL CPSETR ('SPV - SPECIAL VALUE', -1.0)
      CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', -2)
      CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 1)
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETR ('SPV', spv)
CALL CPSETI ('PAI', ipai)
CALL CPSETI ('CLU', iclu)
```

Arguments

SPV

Real---The SPecial Value parameter specifies a special value for certain types of data.
- **0** No data are treated as a special value. This is the default.
- **<>0** This value is a special value that can be used in data fields to mark missing, suspect, or undesired data locations.

PAI

Integer---The Parameter Array Index is used when the parameter to be accessed is a parameter array (rather than that of a single-value parameter). When the parameter is an array, the current value of PAI is used as the index of the element in the array whose value is to be set or retrieved. In this module, PAI is used to specify the contour level use flags parameter (CLU). PAI can specify three special elements in CLU that refer to three lines which are not technically contours:
- **-1** The edge of the contour plot.
- **-2** The edge of any area filled with special values.
- **-3** The edge of any area in which the mapping routine CPMPXY returns an out-of-range value.

CLU(NCL)

Integer array---The Contour Level Use flags parameter determines how the associated contour level in the internal parameter array CLV is used. NCL is the Number of Contour Levels. The value of CLU determines how a line is drawn:
- **0** No contour line is drawn.
- **1** Contour line is drawn without labels.
- **2** Contour labels are drawn without line.
- **3** Contour labels and line are both drawn.

When drawing lines that are not contour lines, any nonzero value for CLU draws the line with no labels.

Discussion

In the **ccpspv** example, the special value flag is set so that any data point with a value of -1.0 is tagged as missing data. This is all that is necessary to specify missing data. Lines 2 and 3 of the **ccpspv.f** code segment set the parameter array index to point to the element of CLU that determines whether or not the special-value edge is drawn; lines 2 and 3 then set that element nonzero so the edge will be drawn. Conpack is then initialized and contours are drawn normally.

Exercises

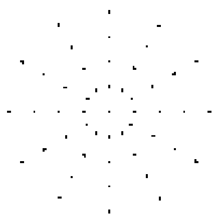
1. Modify **ccpspv.f** so that the missing data region is not a rectangle, and make a second special value region.
2. Modify **ccpspv.f** so that an outline is not drawn around the special value region.
3. What would you have to do to draw contours through a region where data do not exist?

Cp 3.7 Non-Cartesian data

Conpack recognizes two kinds of non-Cartesian data: latitude/longitude data and polar coordinate data. Conpack maps these kinds of data from an array or a rectangle in the Cartesian plane to the desired Ezmap projection or polar coordinate space. Translation is done between these spaces using the subroutine CPMPXY. You can modify CPMPXY to produce other mappings.

Examples of non-Cartesian data

Non-Cartesian data



Non-Cartesian data

Discussion

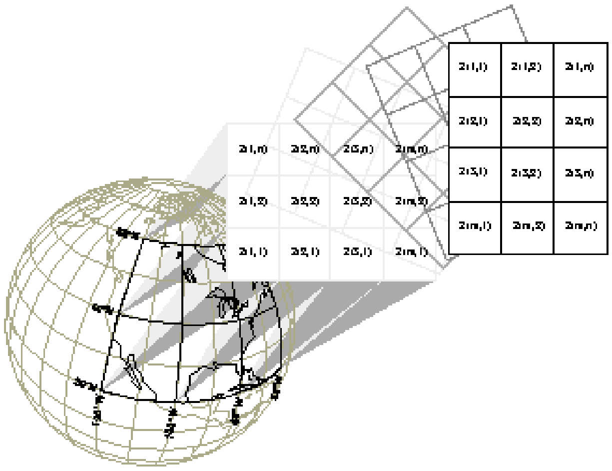
Conpack treats non-Cartesian data in the same way that it treats Cartesian data, except that it requires a mapping function between the non-Cartesian space and the Cartesian plane. What this means is that you need to choose or write a transformation from the Cartesian plane to your non-Cartesian space. This step is done by setting the MAP parameter nonzero, and in some cases, modifying the Conpack routine CPMPXY.

Then decide what kind of data you have in the Cartesian space (for example dense gridded, sparse gridded, irregularly spaced gridded, or nongridded), and initialize for that type of data. After these steps are done, contours are drawn normally. The next few modules discuss examples of how to use your data in various non-Cartesian spaces.

Cp 3.8 Non-Cartesian data: Latitude/longitude data

When plotting latitude/longitude data, you need to know what portion of the globe your data cover, and what kind of data you will have after projection onto the Cartesian plane. Conpack has four parameters to set the corners of your plot, a mapping parameter to specify lat/lon data, and an out-of-range parameter to specify points that don't map into your space or viewport.

Relationship between global coordinates and data array indices



Synopsis

```
CPSETI ('MAP', 1)
CPSETR ('XC1', xmin)
CPSETR ('XCM', xmax)
CPSETR ('YC1', ymin)
CPSETR ('YCN', ymax)
CPSETR ('ORV', val)
```

Arguments

MAP

Integer---The MAPping flag for subroutine CPMPXY.
- **0** Cartesian data. This is the default.
- **1** Latitude/longitude data.
- **2** Rho/theta data.

XC1

Real---The X Coordinate at index 1 parameter is the longitude, in degrees, that corresponds to a value of 1 for the first subscript of the data array. Alternatively, you can think of XC1 as being the minimum value of your longitude data locations.

XCM

Real---The X Coordinate at index M parameter is the longitude, in degrees, that corresponds to a value of M for the first subscript of the data array, where M is the first dimension of your data.

Alternatively, you can think of XCM as being the maximum value of your longitude data locations.

YC1

Real---The Y Coordinate at index 1 parameter is the latitude, in degrees, that corresponds to a value of 1 for the second subscript of the data array. Alternatively, you can think of YC1 as being the minimum value of your latitude data locations.

YCN

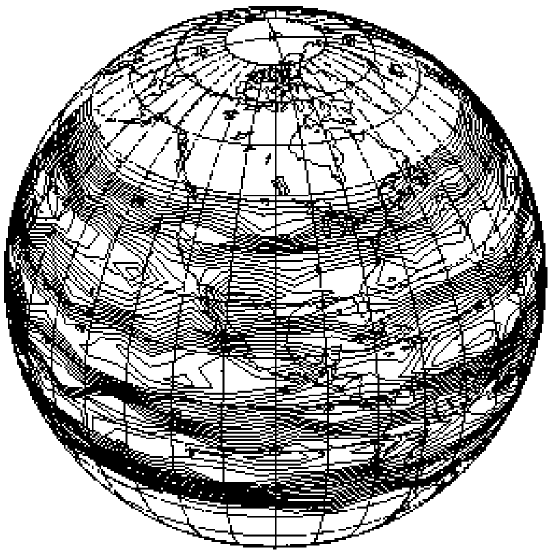
Real---The Y Coordinate at index N parameter is the latitude, in degrees, that corresponds to a value of N for the second subscript of the data array, where N is the second dimension of your data. Alternatively, you can think of YCN as being the maximum value of your latitude data locations.

ORV

Real---When the Out-of-Range Value parameter is set nonzero, ORV is used as the value of X and Y coordinates expected to be returned by the mapping subroutine CPMPXY to say that a point is out of range under the current mapping. If you are plotting points over an Ezmap projection, you should set ORV=1.E12 so that contours that don't project onto the map are not drawn.

Cp 3.8 Non-Cartesian data: Latitude/longitude data

Contours mapped onto the globe



Code segment from ccpmap.f

```
      PARAMETER (RMNLON=-190., RMXLON=20., RMNLAT=-20., RMXLAT=50.)
      DATA RLAT1 /RMNLAT, 0.0/
```

```
3       DATA RLAT2 /RMXLAT, 0.0/
4       DATA RLON1 /RMNLON, 0.0/
5       DATA RLON2 /RMXLON, 0.0/
6       CALL MAPSTI ('GR - GRID', 10)
7       CALL MAPSTC ('OU - OUTLINE DATASET', 'PO')
8       CALL MAPROJ ('SV - SATELLITE-VIEW', 35., -100., 0.)
9       CALL MAPSET ('MA', RLAT1, RLON1, RLAT2, RLON2)
10      CALL MAPDRW
11      CALL CPSETI ('SET - DO-SET-CALL FLAG', 0)
12      CALL CPSETR ('XC1 - X COORDINATE AT INDEX 1', RMNLON)
13      CALL CPSETR ('XCM - X COORDINATE AT INDEX M', RMXLON)
14      CALL CPSETR ('YC1 - Y COORDINATE AT INDEX 1', RMNLAT)
15      CALL CPSETR ('YCN - Y COORDINATE AT INDEX N', RMXLAT)
16      CALL CPSETI ('MAP - MAPPING FLAG', 1)
17      CALL CPSETR ('ORV - OUT-OF-RANGE VALUE', 1.E12)
18      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
19      CALL CPCLDR (ZREG, RWRK, IWRK)
```

Discussion

The **ccpmap** example shows how XC1, XCM, YC1, and YCN associate longitude and latitude data locations with the subscripts of your Z data array. The MAP parameter is used to specify the map transformation you want, and the ORV parameter is set to tell Conpack what value CPMPXY returns for a point on the globe that has no valid projection on your map. If ORV is not set, your program may core dump in cases like the example where contours extend beyond the edges of the projection. You must use the above parameters to set ranges for longitude and latitude before you can overlay your contour plot onto a map drawn by Ezmap.

Lines 1 through 5 of the **ccpmap.f** code segment set the minimum and maximum latitude and longitude values for both the geographic and contour mappings. Lines 6 through 10 draw a black-and-white map of the globe using routines discussed in the Ezmap chapter. Line 6 specifies that latitude and longitude lines should be drawn at 10-degree intervals. Line 7 forces drawing of political and continental boundary lines. Lines 8 and 9 specify a satellite view of the globe. Line 10 draws the map. Up to this point, everything is just as it would be if no contours were to be drawn.

Lines 11 through 19 overlay contours on the Ezmap projection. As discussed in module "Cp 2.7 Contour perimeter: Size, shape, and location using SET," the Conpack parameter SET in line 11 tells Conpack that subroutine SET has been called by Ezmap. XC1, XCM, YC1, and YCN fix the corners of the array on the globe. Line 16 sets the Conpack parameter MAP so that Conpack knows that lat/lon data is being used. Line 12 sets ORV to "1.E12" so that contours will be terminated at the edge of the visible portion of the globe. Notice that all of the parameter-setting routines are called before the CPRECT call to initialize Conpack for drawing.

Lines 18 and 19 initialize Conpack and draw contours in the same way as for a simple Cartesian plot.

In some cases, you may want to draw contours across the international date line. To do this, you must make sure that XC1<XCN, even if it means making XCN>180. Notice that contours are drawn across the date line in the example.

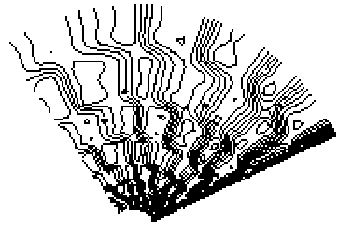
Exercises

1. Using what you learned in this module, convert **ccprect.f** to overlay contours on a black-and-white satellite view of China (with the corner points of the contoured region at 15N, 73E and 55N, 145E).
2. Convert the **ccpsps1** example to overlay contours on a Lambert Conformal map of South America (with the corner points of the contoured region and of the map at 55S, 95W and 15N, 35W).

Cp 3.9 Non-Cartesian data: Polar coordinates

When plotting polar coordinate data (rho/theta data), you need to know what portion of the Cartesian plane your data cover, and what kind of data you will have after projection onto the Cartesian plane. Conpack has four parameters to set the minimum and maximum values of rho and theta, and a mapping parameter to specify polar coordinates.

Data plotted in polar coordinates



Code segment from ccpcir.f

```
      CALL SET (0.05, 0.95, 0.05, 0.95, -RHOMX, RHOMX, -RHOMX, RHOMX, 1)
      CALL CPSETI ('SET - DO-SET-CALL FLAG', 0)
      CALL CPSETR ('XC1 - X COORDINATE AT INDEX 1', RHOMN)
      CALL CPSETR ('XCM - X COORDINATE AT INDEX M', RHOMX)
      CALL CPSETR ('YC1 - Y COORDINATE AT INDEX 1', THETMN)
      CALL CPSETR ('YCN - Y COORDINATE AT INDEX N', THETMX)
      CALL CPSETI ('MAP - MAPPING FLAG', 2)
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPCLDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CPSETR ('XC1', rho-min)
CPSETR ('XCM', rho-max)
CPSETR ('YC1', theta-min)
CPSETR ('YCN', theta-max)
CPSETI ('MAP', imap)
```

Arguments

XC1

Real---The X Coordinate at index 1 parameter is the value of rho that corresponds to a value of 1 for the first subscript of the data array. Alternatively, you can think of XC1 as being the minimum rho value of your data.

XCM

Real---The X Coordinate at index M parameter is the value of rho that corresponds to a value of M for the first subscript of the data array, where M is the first dimension of your data. Alternatively, you can think of XCM as being the maximum rho value of your data.

YC1

Real---The Y Coordinate at index 1 parameter is the value of theta, in degrees, that corresponds to a value of 1 for the second subscript of the data array. Alternatively, you can think of YC1 as being the minimum theta value of your data.

YCN

Real---The Y Coordinate at index N parameter is the value of theta, in degrees, that corresponds to a value of N for the second subscript of the data array, where N is the second dimension of your data. Alternatively, you can think of YCN as being the maximum theta value of your data.

MAP

Integer---The MAPping flag for subroutine CPMPXY.
0 Cartesian data. This is the default.
1 Latitude/longitude data.
2 Rho/theta data.

Discussion

Line 1 of the **ccpcir.f** code segment calls the SPPS routine SET so that the polar plot appears in the center of the viewport. Line 2 uses the Conpack parameter SET to tell Conpack that subroutine SET has been called. XC1, XCM, YC1, and YCN fix the minimum and maximum values of rho and theta.

Line 7 sets the MAP parameter so that Conpack knows we are using polar coordinates. Notice that all of the parameters are set before the CPSPS2 call to initialize Conpack for drawing.

Lines 8 and 9 initialize Conpack and draw contours just as they would be for a simple Cartesian plot.

Exercises

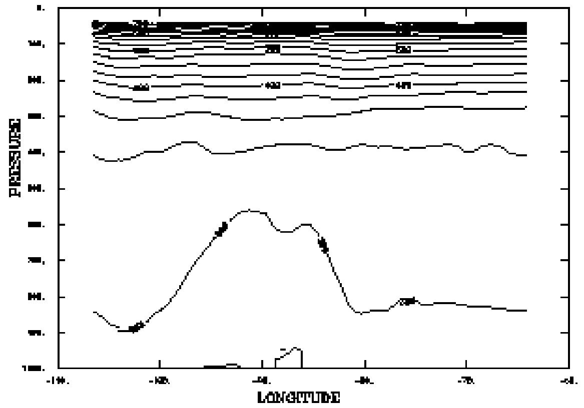
1. Modify **ccprect.f** so that it fills the top half of a circle in polar coordinates.

Cp 3.10 Non-Cartesian data: Other data types

CPMPXY maps Conpack output from a rectangular coordinate system superimposed on the data grid to some other coordinate system. The default version handles the mapping of longitudes and latitudes into rectangular coordinates on a given Ezmap projection. It also handles mapping polar coordinates into rectangular coordinates. You can supply your own version of CPMPXY to produce any other desired mapping. CPMPXY is not called by you; it is called by Conpack when the parameter MAP is nonzero.

This module explains how to write simple height and pressure transformations.

Contouring a pressure function



Code segment from ccpmpxy.f

```
      SUBROUTINE CPMPXY (IMAP, XINP, YINP, XOTP, YOTP)
      PARAMETER (JX=60, KX=26)
      COMMON /HEIGHT/ Z (JX, KX)
      ELSEIF (IMAP .EQ. 3 .OR. IMAP .EQ. 4) THEN
        XOTP = XINP
        X = XINP
        IIX = INT(X)
        DIFX = X - FLOAT(IIX)
        Y = YINP
        IY = INT(Y)
        DIFY = Y - FLOAT(IY)
        IXP1 = MIN0 (JX, IIX + 1)
        IYP1 = MIN0 (KX, IY + 1)
        Z1 = Z(IIX, IY) + DIFY * (Z(IIX, IYP1) - Z(IIX, IY))
        Z2 = Z(IXP1, IY) + DIFY * (Z(IXP1, IYP1) - Z(IXP1, IY))
        ZR = Z1 + DIFX * (Z2 - Z1)
        YOTP = ZR
        IF (IMAP .EQ. 4) YOTP = 1000. * EXP (-ZR / 7.)
```

Synopsis

```
SUBROUTINE CPMPXY (IMAP, XINP, YINP, XOTP, YOTP)
```

Arguments

IMAP

Integer, Input---The current (nonzero) value of the parameter MAP.

XINP

Real, Input or Output---The X coordinate of a point on the contour plot. If XC1=XCM (the default situation), then XINP lies in the range from 1 to M, where M is the first dimension of the array being contoured (equal to the value of the parameter ZDM). In this case, the X coordinate has the same range as the first index of the data array. If you set XC1<>XCM, then XINP lies in the range from XC1 (corresponding to an index value of 1) to XCM (corresponding to an index value of M).

YINP

Real, Input or Output---The Y coordinate of a point on the contour plot. If YC1=YCN (the default situation), then YINP lies in the range from 1 to N, where N is the second dimension of the array being contoured (equal to the value of the parameter ZDN). In this case, the Y coordinate has the same range as the second index of the data array. If you set YC1<>YCM, then YINP lies in the range from YC1 (corresponding to an index value of 1) to YCN (corresponding to an index value of N).

XOTP

Real, Output---The mapped X coordinate of a point on the contour plot, in a coordinate system consistent with the current window, as specified by arguments 5 through 8 of the last call to the SPPS routine SET or by the equivalent GKS call.

YOTP

Real, Output---The mapped Y coordinate of a point on the contour plot, in a coordinate system consistent with the current window, as specified by arguments 5 through 8 of the last call to the SPPS routine SET or by the equivalent GKS call.

Discussion

IMAP contains the value of the mapping parameter MAP and is used by CPMPXY to determine which transformation is used. In the **ccpmpxy** example, IMAP=4 on entry to CPMPXY. XINP and YINP are the untransformed coordinates of a point on a contour line. These points may not lie on the grid of initial data points. XOTP and YOTP are the output data points for CPMPXY, and represent the final locations of the points defining contour lines.

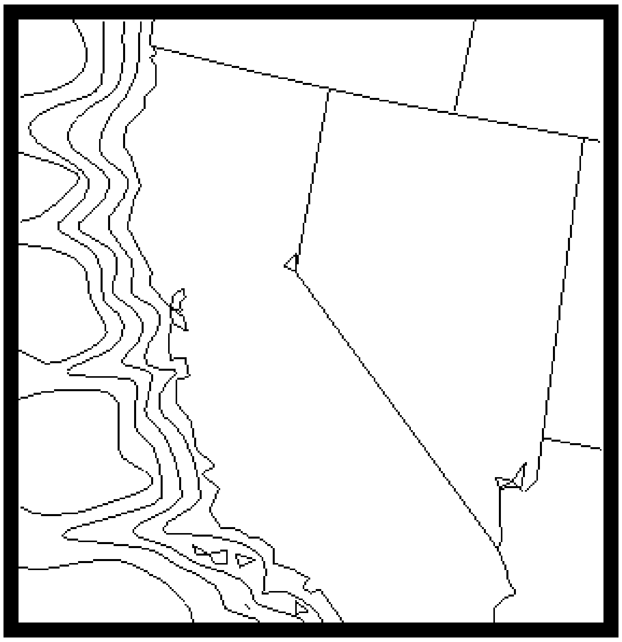
Line 3 of the **ccpmpxy.f** code segment passes the data array containing height values into CPMPXY for use in calculating the pressure transformation. Line 4 checks to see whether a height or pressure transformation is specified. Since there is no transformation in the X direction, line 5 gives the X output coordinate the same value as the X input coordinate.

Because the point on the contour line probably is not on one of the grid points in the original data, it is necessary to interpolate the height and pressure data value in this case. Therefore, lines 7 through 15 find the nearest data points in the Y direction and interpolate for the height at the point on the contour line. Line 16 calculates the interpolated height, and line 17 returns it as the Y output coordinate. If the MAP parameter IMAP specifies a pressure transformation, line 18 calculates that transformation and returns the pressure value as the Y output coordinate.

Cp 3.11 Contouring non-rectangular domains

This module describes a common problem that does not have any quick solution at this time. Many scientists want to contour data that is not neatly laid out over a rectangular domain. There are two common problems with this kind of data. In the first case, contours can flow over the irregular boundary, and in the second case, they cannot.

Contouring with an irregular domain



Discussion

If contours can be allowed to flow over the boundary (such as air temperature over the ocean), there are two ways to end the contour lines and draw the boundary. You can put special values into your regular grid over areas where you don't have data values, or you can use Areas to mask the contour lines over the regions where you don't have data values. See module "Cp 3.6 Missing or special data" for an example of how to use special values, or see section "Cp 8. Overlaying filled contours on a map" for details on how to mask contour lines.

Some contours cannot be allowed to flow over the boundary. For example, ocean temperature doesn't exist over land, so contours showing ocean temperature must follow the shoreline. Unfortunately, the data are usually not dense enough along the boundary to force contours to conform to the boundary line, and worse, Bivar assumes that the data have a continuous first derivative. What this means in practice is that by using the methods discussed in section "Cp 3. Initializing Conpack," contour lines are almost guaranteed to cross boundary lines. When interpolation is being used in cases like this, it may be possible to find an interpolation package that better fits your needs for interpolation libraries (FITPACK may be a good choice).

Cp 3.12 Setting minimal workspace

Because FORTRAN 77 lacks pointer variables and dynamic storage allocation, Conpack requires that you allocate real and integer workspace for your calculations.

The amount of workspace needed to produce a contour plot often depends on how many tightly curved contour lines you have and on how several parameters are set. A plot having only a few contours with large, gradual curves needs much less workspace than a plot with small, winding contours.

Although it is possible to calculate an upper bound for both integer and real work arrays, the process is tedious, and it must be done for each Conpack routine called. Please see the Conpack programmer document if you need to calculate an upper bound for your work arrays. It is much easier to let Conpack tell you how much space it needs, and then adjust your array sizes downward to match.

Results from running ccprwu.f

```
Integer Workspace Used:   100 Real Workspace Used:   200
```

Code segment from ccprwu.f

```
1       CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
2       CALL CPBACK (Z, RWRK, IWRK)
3       CALL CPCLDR (Z, RWRK, IWRK)
4       CALL CPGETI ('IWU - INTEGER WORKSPACE USAGE', IIWU)
5       CALL CPGETI ('RWU - REAL WORKSPACE USAGE', IRWU)
6       WRITE (6,*) 'Integer Workspace Used: ',IIWU,
       + ' Real Workspace Used: ',IRWU
```

Synopsis

```
      CALL CPGETI ('IWU', iwu)
      CALL CPGETI ('RWU', irwu)
```

Arguments

IWU

> *Integer*---The Integer Workspace Usage parameter is intended for retrieval only. It is set to zero by any of the initialization routines. It is then updated by each subsequent Conpack routine called to reflect the largest number of words of integer workspace needed at any one time.

RWU

> *Integer*---The Real Workspace Usage parameter is intended for retrieval only. It is set to zero by any of the initialization routines. It is then updated by each subsequent Conpack routine called to reflect the largest number of words of real workspace needed at any one time.

Discussion

Lines 4 and 5 of the **ccprwu.f** code segment retrieve IWU and RWU, then line 6 prints them. They are retrieved after all the calls to Conpack have been completed so we can see how much workspace was used for the entire plot. In this case, an absolute minimum of workspace is used. As mentioned in previous modules, it is a good idea to start with LRWK=2500, and LIWK=1000 (where LRWK is the length of RWRK, and LIWK is the length of IWRK), then move to smaller values if memory is at a premium on your machine.

Since you use IWU and RWU most frequently when you are planning to run the same job over and over with different data, we recommend that you always make LRWK and LIWK somewhat larger than the values returned by RWU and IWU.

Exercises

1. Using the **ccpezct** example, determine the amount of real and integer workspace used.

Cp 4. Conpack Contour line basics

The contour line basics section covers drawing lines and setting contour line attributes (like line pattern, color, thickness, and level values) for simple line contour drawings and for contour drawings with filled contour levels.

In the Conpack functional outline, note that setting contour line attributes and drawing the lines may require you to set parameters in two places, and then call a contour drawing routine.

Conpack functional outline

```
-----------------------------------------------------------
    1.  Open GKS
    2.  Set window and viewport
    3.  Put in titles and other labels
 *  4.  Set up general parameters
    5.  Set up area maps
    6.  Initialize Conpack
 *  7.  Force Conpack to choose contour levels if necessary
 *  8.  Modify Conpack-chosen parameters
    9.  Draw labels
    10.  Draw background
 *  11.  Draw contour lines
    12.  Call FRAME
    13.  Close GKS
-----------------------------------------------------------
```

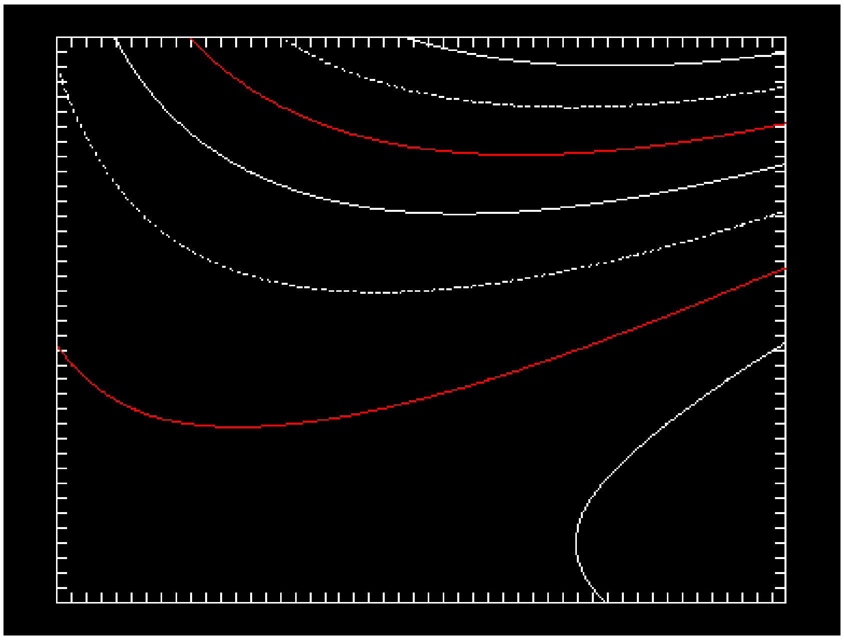
* Steps discussed in this section.

Cp 4.1 Contour line basics

This section covers two issues: how to draw contour lines and what attributes you want to assign or change for one or more of those contour lines. In the next module, you can learn how to draw contour lines using CPCLDR. If you want to use the Areas utility to color-fill or pattern-fill your contour levels, you may not need to draw the contour lines with this routine. More information appears in section "Cp

5. Filling contour levels."

Contour line attributes



Discussion

If you want to change one or more contour level values, you must first decide if you want to change a value that Conpack chooses for you, or if you want to assign values to every contour level. After this choice is made, you change contour level values much the same as other contour line attributes.

To change a single contour line attribute, it is necessary to specify the contour line you want to modify, then set the appropriate Conpack parameters using the CPSET*x* routines.

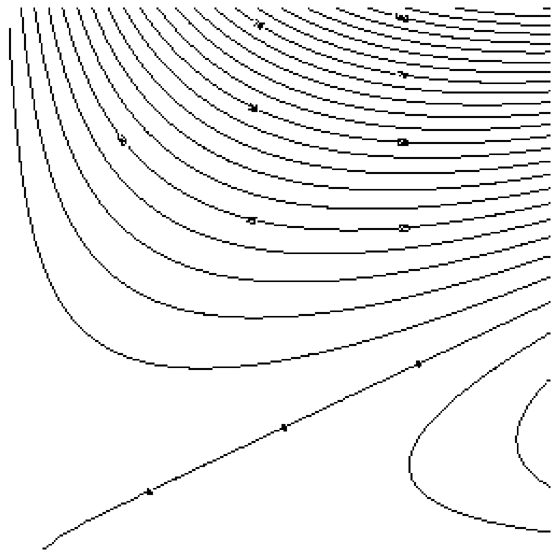
Cp 4.2 Drawing contour lines

There are two contour-drawing routines in Conpack: CPCLDR and CPCLDM.

This module discusses the routine CPCLDR; it draws contour lines across the entire contour rectangle.

CPCLDM (the other routine) is discussed in the masking areas modules Cp 5.5 and Cp 5.6. CPCLDM is used with the Areas utility to draw contours over a specified region of the contour rectangle; it should be used with non-default contour line labels.

Basic contours



Code segment from ccpcldr.f

```
1       CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
2       CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPCLDR (Z, RWRK, IWRK)
```

Arguments

All three arguments to the CPCLDR call are arrays used in the last call to CPRECT, CPSPS1, or CPSPS2. Don't change the contents of these arrays between calls to Conpack.

Z

Real array, Input---The data array.

RWRK

Real array, Workspace---The real work array.

IWRK

Integer array, Workspace---The integer work array.

Discussion

Line 1 of the **ccpcldr.f** code segment initializes Conpack with a call to CPRECT. This can also be done with CPSPS1 or CPSPS2. Line 2 draws the contour lines with CPCLDR. If you call CPCLDR after calling CPSPS1 or CPSPS2, make sure that you call it with the interpolated data array rather than with your original data array.

If you are using irregularly spaced data or sparse data, it is extremely important to pass your regularly spaced data array to CPCLDR. Otherwise, you will get incorrect results.

If your data all have the same values, in other words, if you have a constant data field, CPCLDR writes out a constant field message on your CGM and does not draw any contours.

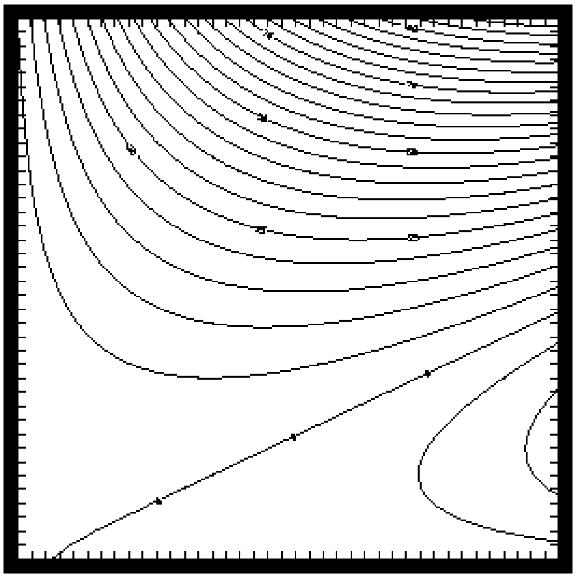
Exercises

1. Modify **ccpcldr** to use sparse gridded data.

Cp 4.3 Four methods of contour level selection

Conpack chooses contour levels only when they are needed, and not in the initialization subroutines. This feature allows you to use several methods to pick your contour levels. By default, Conpack attempts to choose about 16 "nice" levels. If these don't meet your needs, you can modify one or more Conpack-chosen levels, you can fix the number of levels that you get, or you can specify each contour level.

A "nice" contour plot



Discussion

There are four methods used to decide how many and which contour levels are drawn. By default, Conpack employs a rather complicated scheme using several parameters to choose "nice" contour level values. This scheme is discussed in the default contour level selection modules Cp 4.4 through Cp 4.6.

After Conpack chooses the contour level values, then you can change their values or even add new contour levels. You must first force Conpack to choose contour levels using CPPKCL, and then retrieve the number of contour levels before modifying the levels using the same parameters Conpack uses by default. Information about modifying the contour level values chosen by Conpack appears in the modifying Conpack-chosen levels modules Cp 4.7 and Cp 4.8.

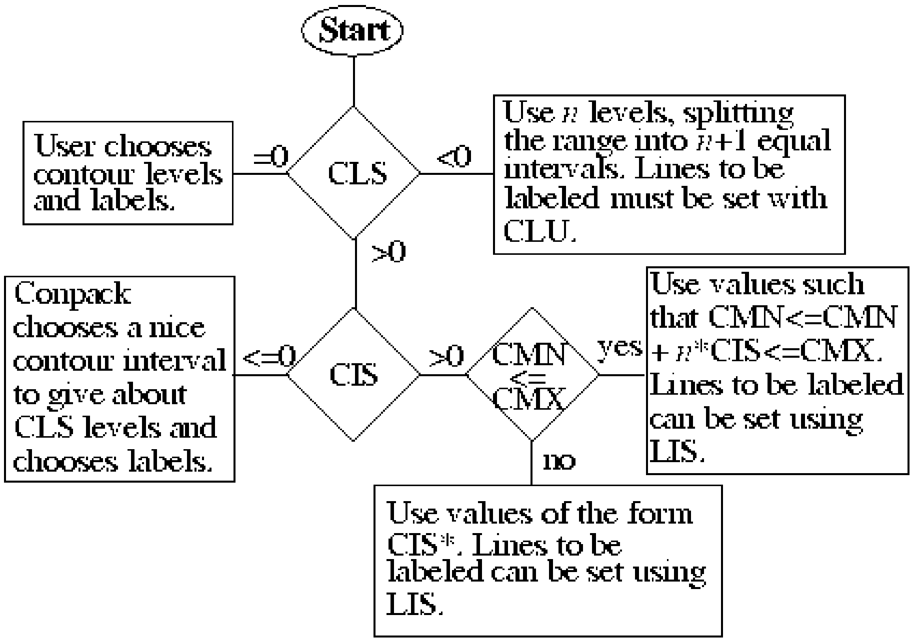
The third method you can use to decide how many contour levels are drawn is to set CLS to the negative of the number of contour levels that you want. When this method is used, the minimum and maximum data values are located, and contours are drawn at *n* equal intervals over the range of the data. This method may produce contours at inconvenient or "ugly" values; this is discussed in module "Cp 4.9 Setting *n* equally spaced contour levels."

The most reliable way to get contours where you want them is to set all of them yourself, using the CLS, NCL, PAI, CLV, and CLU parameters. This method is discussed in module "Cp 4.10 Choosing your own contour levels."

Cp 4.4 Default contour level selection

By default, contour levels in Conpack are chosen using the five parameters CLS, CIS, CMN, CMX, and CIT. These parameters are used to place contours at "nice" values for most sets of data. As with everything in Conpack, these parameters are completely configurable.

Default contour levels



Synopsis

```
CALL CPSETI ('CLS', icls)
```

Arguments

CLS

Integer---The Contour Level Selection flag is critical to how contour levels are chosen. It not only determines how many contour levels are chosen, it also determines which contour level selection method is used. By default *n*=16.

0

Conpack does not pick contour levels at all; the current values of the parameters NCL, CLV, and associated arrays are not changed. They thus retain the values chosen by you, or by Conpack during a previous call. This option is used in module "Cp 4.10 Choosing your own contour levels."

-n

Where *n* is a positive integer, Conpack generates exactly *n* contour lines, splitting the range from the minimum field value to the maximum field value into *n*+1 equal intervals (contour levels). This option is discussed further in the modifying Conpack-chosen levels modules Cp 4.7 and Cp 4.8.

+n

By default, Conpack chooses contour intervals at *base**k*, where the base value is some nice value either chosen by Conpack or set with the CIS parameter, and *k* is an integer. A nice value for *base* is defined by the contents of CIT, and chosen in such a way as to give at least *n* contour levels. Using the default CIT, you may get as many as 2*n* levels. In other words, *n*<=*kmaxkmin*+1<=2*n*. This option is discussed further in the default contour level selection modules Cp 4.4 through Cp 4.6.
If CIS>0 and CMN>CMX (your contour field minimum is set greater than your contour field maximum), then Conpack uses values of the form CIS**k*, where *k* is used as above. If CIS>0 and CMN<=CMX (your contour minimum is set less than or equal to your contour maximum), then Conpack uses values of the form CMN+CIS**k*, where *k* is an integer greater than or equal to CMN and less than or equal to CMX. See module "Cp 4.6 Default contour level selection: Fixed contour intervals and labels."

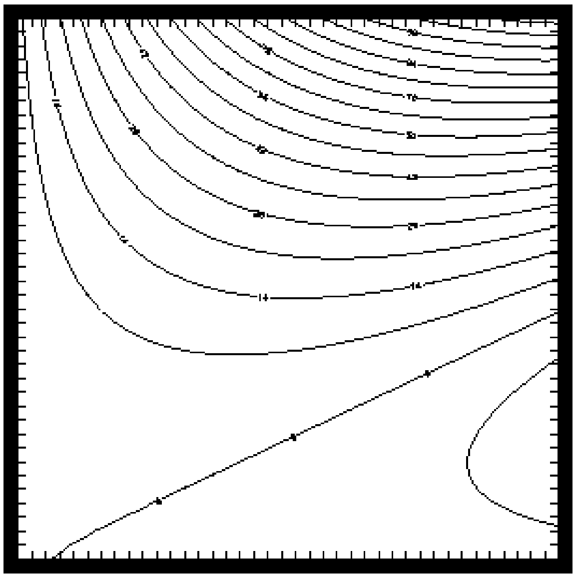
Discussion

The flowchart shows how the contour level selection process hinges on the CLS parameter. If CLS=0, then Conpack assumes that you will choose contour levels. If CLS<0, then Conpack chooses ABS(*n*) equally spaced contour levels, splitting the range from the minimum field value to the maximum field value into *n*+1 equal intervals. If CLS>0, then CIS is used to differentiate between choosing "nice" contour levels and choosing levels with a specific interval.

Cp 4.5 Default contour level selection: Default intervals and labels

There are two ways to get a "nice" contour interval. You can either set it directly using the CIS parameter, or you can give Conpack some options using the CIT parameter.

Changing nice contour intervals



Code segment from ccpcit.f

```
      DATA CIT /1., 2., 3., 4., 5., 6., 7., 8., 9., 0./
      DATA LIT /2,  2,  2,  2,  2,  2,  2,  2,  2,  0/
      CALL OPNGKS
      DO 101, I=1, 10
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETR ('CIT - CONTOUR INTERVAL TABLE', CIT(I))
         CALL CPSETI ('LIT - LABEL INTERVAL TABLE', LIT(I))
 101  CONTINUE
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CPSETI ('PAI', n)
CPSETR ('CIT', cit)
CPSETI ('LIT', lit)
```

Arguments

PAI

Integer---The Parameter Array Index must be set to specify the array element that is to be changed. For example, if you want to change the third array element of CIT to 3.33, you would set PAI=3, then set CIT to 3.33.

CIT(10)

Real array---The Contour Interval Table parameter specifies what constitutes the "nice" value that Conpack is allowed to use as a contour interval; it can use values of the form "*c* times 10 to the power *n*", where *c* is one of the nonzero elements of CIT. (For example, if 2. occurs in CIT, it means that Conpack can use a contour interval of 2 or 20 or 200 or .2 or .02, and so forth.)

LIT(10)

Integer array---For each element of CIT, the associated element of the Label Interval Table parameter specifies how far apart the labeled levels are. The default contents of CIT and LIT are:

PAI	CIT(PAI)	LIT(PAI)
1	1.0	5
2	2.0	5
3	2.5	4
4	4.0	5
5	5.0	5
6	0.0	0
7	0.0	0
8	0.0	0
9	0.0	0
10	0.0	0

Given the default values for CIT, we might get, in a given situation, a contour interval of 25, with every 4th level labeled, or a contour interval of 40, with every 5th level labeled. We could not get a contour level of 30, even though that might be desirable for more nearly the approximate number of levels desired.

Discussion

Lines 1, 2, and 4 through 7 of the **ccpcit.f** code segment set CIT so that "nice" values are the same as they were in the old CONREC routine. LIT is set up to force every other contour level to be labeled no matter what. Lines 9 through 11 draw contours normally. Notice that "nice" values in this plot are multiples of 7.

Line labels are discussed further in sections "Cp 7. General labels" and "Cp 6. Contour line labels."

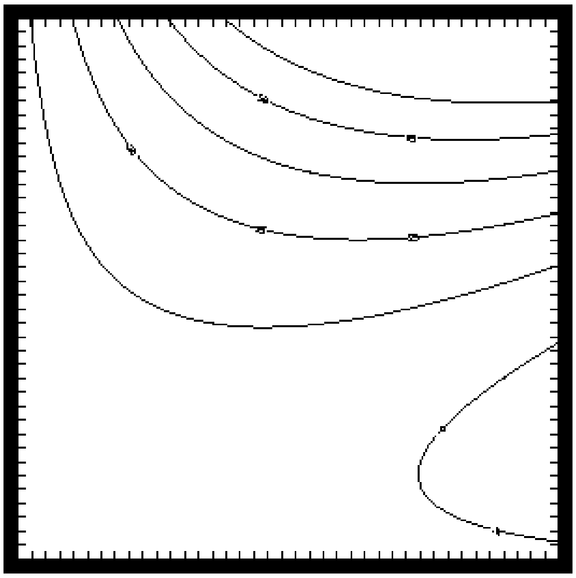
Exercises

1. Convince Conpack that increments of 33.3 are "nice" values.

Cp 4.6 Default contour level selection: Fixed contour intervals and labels

If the CLS parameter is set so that Conpack chooses contour levels (CLS>0), then it is possible to mandate a contour interval using the CIS parameter. Using the LIS parameter, it is also possible to specify which levels will get labels.

Fixed contour intervals



Code segment from ccpcis.f

```
      CALL CPSETR ('CIS - CONTOUR INTERVAL SPECIFIER', 15.)
      CALL CPSETI ('LIS - LABEL INTERVAL SPECIFIER', 2)
      CALL CPSETR ('CMN - CONTOUR MINIMUM', -5.)
      CALL CPSETR ('CMX - CONTOUR MAXIMUM', 75.)
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETR ('CIS', cis)
      CALL CPSETI ('LIS', lis)
      CALL CPSETR ('CMN', cmn)
      CALL CPSETR ('CMX', cmx)
```

Arguments

CIS

Real---The use of the Contour Interval Specifier parameter depends on the Contour Level Selection flag. When CLS>0, as discussed in module "Cp 4.4 Default contour level selection," a CIS value of:

<=0 Conpack chooses the contour interval using the CIT table to choose a "nice" base value. This is the default.

>0 Conpack uses CIS as the precise contour interval to be used. In this case, you may want to specify LIS to be some nonzero value to label every *n*th contour level. Labeling contour levels is discussed in sections "Cp 7. General labels" and "Cp 6. Contour line labels."

If CLS<=0, then CIS is ignored.

LIS

Integer---The use of the Label Interval Specifier parameter depends on CLS and CIS. When CLS>0 and CIS>0, then you must set LIS to specify the interval between labeled contour levels. By default, LIS=5, meaning every fifth contour is labeled.
Note: LIS is only used with the default contour line labels.

CMN

Real---The use of the Contour MiNimum and the Contour MaXimum value also depend on CLS and CIS. When CLS>0 and CIS>0 and CMN<=CMX, then contour levels are set at CMN, CMN+CIS, CMN+2*CIS, ... CMN+*n**CIS<=CMX. By default, CMN=1.0 and CMX=0.0, so CMN>CMX.

CMX

Real---The Contour MaXimum value. By default, CMX=0.0.

Discussion

If you specify CLS=1, CIS=1/3, and LIS=3, you get contours at values like 1/3, 2/3, 3/3, 4/3, etc., and labels on contour lines with values like 1.0, 2.0, 3.0, and so on.

Line 1 sets CIS to 15., then line 2 specifies labels on every other line. Lines 3 and 4 set the low and high contour levels to -5. and 75. Contours are set to *n**CIS+CMN (in this case -5., 10., 25., ...).

If you set CIS, you must also specify LIS to get contour line labels.

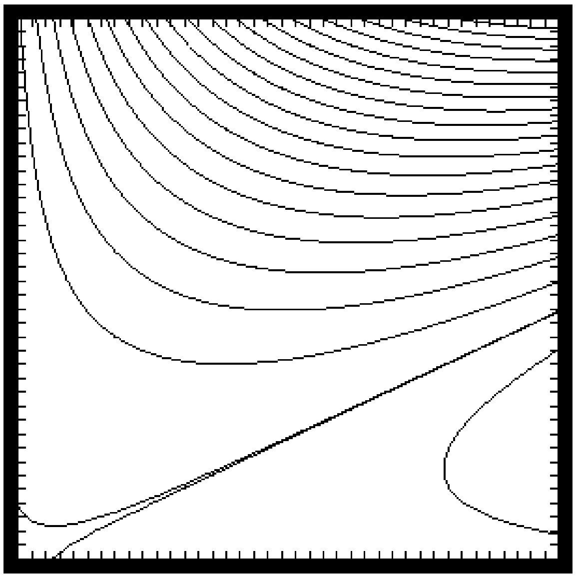
Exercises

1. Using the **ccpcis** example, force Conpack to use a contour interval of 17., and to label every third contour level.
2. Set CMN and CMX equal to 0.0, and describe what the results will be.

Cp 4.7 Modifying Conpack-chosen levels: Picking levels

Before you can modify Conpack-chosen contour levels, you must force Conpack to choose contour levels for you by using subroutine CPPKCL. CPPKCL is called by the first Conpack routine that needs a contour level; it is not called by CPRECT, CPSPS1, or CPSPS2.

Modified contour levels



Code segment from ccppkcl.f

```
      CALL CPRECT (ZREG, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NOCL)
      CALL CPSETI ('NCL - NUMBER OF CONTOUR LEVELS', NOCL+1)
      CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', NOCL+1)
      CALL CPSETR ('CLV - CONTOUR LEVEL VALUES', 0.0)
```

Synopsis

```
CALL CPPKCL (ZREG, RWRK, IWRK)
```

Arguments

All three arguments to the CPPKCL call are arrays used in the last call to CPRECT, CPSPS1, or CPSPS2, the contents of which must not have been changed since that call. Since the dimensions of the following arrays are passed in common blocks, they are not shown here.

ZREG

> *Real array, Input*---The densely gridded data array.

RWRK

> *Real array, Workspace*---The real work array.

IWRK

> *Integer array, Workspace*---The integer work array.

Discussion

Notice that you must call CPPKCL after the initialization routine (CPRECT, CPSPS1, or CPSPS2) because it depends on values set by the initialization routine. Also, since modifying contour level options chosen by Conpack must come after the initialization routines, this is one of the few times when you want to set parameters after the Conpack initialization routines. After parameters are chosen, contour lines and a background are drawn normally.

If the contour level selection parameter CLS is zero, CPPKCL does nothing. If the data field is found to be essentially constant by the initialization routines, (the constant field flag CFF<>0), then CPPKCL does nothing.

Calling CPPKCL sets the number of contour levels parameter NCL to the number of contour levels picked. Elements 1 through NCL of the parameter array CLV (contour level values) are set to the levels themselves. The Ith elements of the associated parameter arrays are set as follows:

```
-------------------------------------------------
CLU=1 or 3                Contour level use flag
LLT=' ' (single blank)    Contour line label text
CLD='$$$$$$$$$$$$$$$$'    Contour line dash pattern
CLC=-1                    Contour line color
LLC=-1                    Contour line label color
CLL = 0.                  Contour line line width
-------------------------------------------------
```

Thus, after CPPKCL is called, the situation for each contour level is as follows:

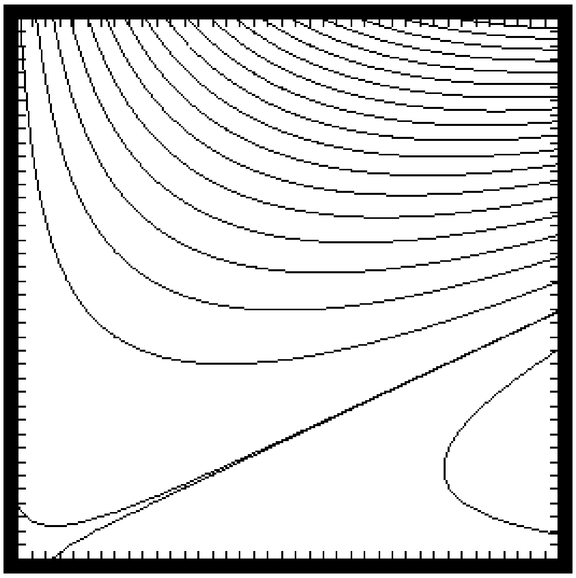
- Contour lines are to be drawn by a call to CPCLDR or CPCLDM (because CLU<>0).
- The lines are to be labeled if CLU=3, but not labeled if CLU=1.
- The dash pattern for the level is solid, and neither the color of the line, the color of the labels on it, nor its width are to be set. These options are discussed in the line attributes modules Cp 4.11 through Cp 4.14.
- The text of the label associated with the level is, as yet, unspecified.

By resetting elements in these parameter arrays, you have complete control over each contour line.

Cp 4.8 Modifying Conpack-chosen levels: Changing the levels

After contour levels are chosen, there are two Conpack parameters that can be changed to modify the already chosen contour levels, number of contour levels (NCL), and contour level values (CLV).

Modified contour levels



Code segment from ccppkcl.f

```
      CALL CPRECT (ZREG, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NOCL)
      CALL CPSETI ('NCL - NUMBER OF CONTOUR LEVELS', NOCL+1)
      CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', NOCL+1)
      CALL CPSETR ('CLV - CONTOUR LEVEL VALUES', 0.0)
```

Synopsis

```
CALL CPSETI ('NCL', ncl)
CALL CPSETI ('PAI', ipai)
CALL CPSETR ('CLV', clv)
```

Arguments

NCL

Integer---The Number of Contour Levels value is used by many Conpack routines. The Conpack initialization routines set NCL to zero if CLS<>0. NCL is then set to some number of contour levels when contour levels are picked by CPPKCL or the contour-drawing routines.

PAI

Integer---The Parameter Array Index must be set to specify the array element that is to be changed. For example, if you want to change the third contour level value to 3.33, you would set PAI=3, then set CLV to 3.33.

CLV(NCL)

Real array---Each of the first NCL elements of the parameter array CLV (Contour Level Values) specifies a numerical value given to a contour level.

Discussion

Line 2 of the **ccppkcl.f** code segment forces Conpack to choose contour lines. Line 3 retrieves the number of contour lines. Line 4 increments the number of contour lines by 1. Lines 5 and 6 then add a zero-value contour line. The order in which the contour lines are stored in the CLV array doesn’t matter, so it’s easiest to add contour lines to the end of the array.

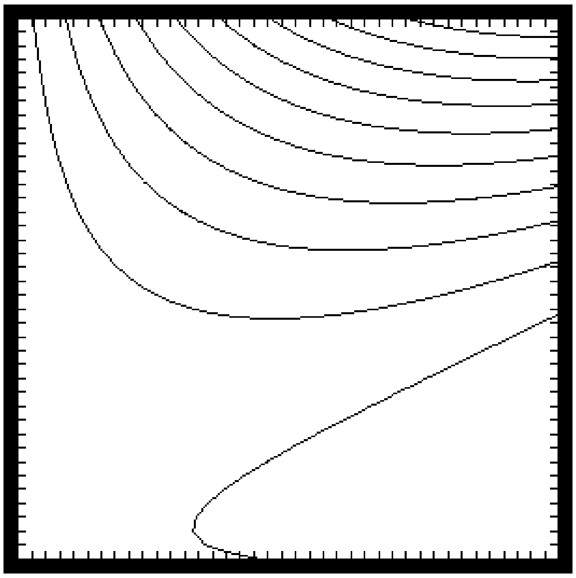
Exercises

1. Using the **ccppkcl** example, add contour lines at 5.0 and +5.0, eliminating the contour level at 0.0.
2. Remove every other contour line from the **ccppkcl** example.

Cp 4.9 Setting *n* equally spaced contour levels

After you learn what goes on in the default contour level selection process, it is relatively simple to make Conpack choose some number of equally spaced contour levels. All you need to do is to set the CLS parameter, and optionally, the contour minimum and maximum parameters (CMN and CMX respectively).

Ten equally spaced contour lines



Code segment from ccpncls.f

```
      CALL CPSETI ('CLS - CONTOUR LEVEL SELECTION FLAG', -10)
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('CLS', icls)
```

Arguments

CLS

Integer---The Contour Level Selection flag is critical to how contour levels are chosen. It not only determines how many contour levels are chosen, it also determines which contour level selection method is used. By default *n*=16.

0

Conpack does not pick contour levels at all; the current values of the parameters NCL, CLV, and associated arrays are not changed. They thus retain the values chosen by you, or by Conpack during a previous call. This option is used in module "Cp 4.10 Choosing your own contour levels."

-n

Where *n* is a positive integer, Conpack generates exactly *n* contour levels, splitting the range from the minimum field value to the maximum field value into *n*+1 equal intervals. This option is discussed further in the modifying Conpack-chosen levels modules Cp 4.7 and Cp 4.8.

+n

By default, Conpack chooses contour intervals at *base***k*, where the base value is some nice value either chosen by Conpack or set with the CIS parameter, and *k* is an integer. A nice value for *base* is defined by the contents of CIT, and chosen in such a way as to give at least *n* contour levels. Using the default CIT, you may get as many as 2*n* levels. In other words, *n*<=*kmax*-*kmin*+1<=2*n*. This option is discussed further in the default contour level selection modules Cp 4.4 through Cp 4.6.

If CIS>0 and CMN>CMX (your contour field minimum is set greater than your contour field maximum), then Conpack uses values of the form CIS**k*, where *k* is used as above. If CIS>0 and CMN<=CMX (your contour minimum is set less than or equal to your contour maximum), then Conpack uses values of the form CMN+CIS**k*, where *k* is an integer greater than or equal to CMN and less than or equal to CMX.

Discussion

Giving CLS a negative value is most commonly used with solid-fill color contour levels since the color table must have a pre-set number of colors in it.

Line 1 of the **ccpncls.f** code segment sets the CLS parameter; the value 10 tells Conpack to use exactly ten contour lines at equally spaced intervals. Contours are then initialized and drawn normally.

Notice that in this case, contour line labels are not drawn. The only way to get contour labels drawn using this technique is to set the CLU parameter for the specific lines that you want labeled. This option is discussed in section "Cp 6. Contour line labels."

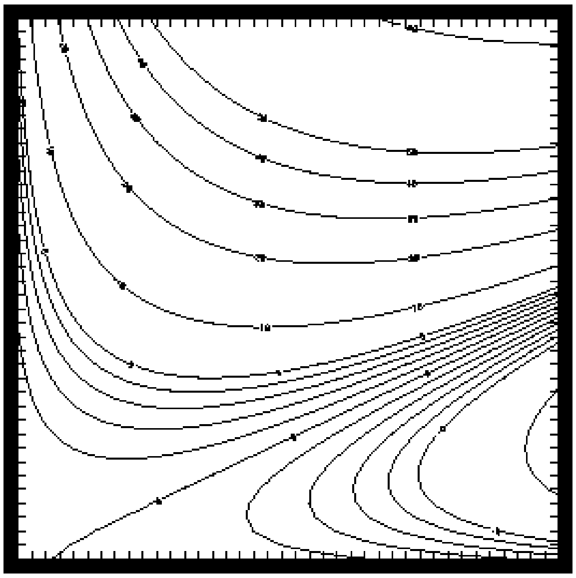
Exercises

1. Using the **ccpncls** example, set CMN=10., and CMX=10.0.
2. Don't set CLS, but set NCL (the number of contour levels) to 10, and explain what happens.

Cp 4.10 Choosing your own contour levels

In some cases, you may want to take the controls into your own hands and specify each contour level for one or more Conpack plots. Setting all of the contour levels involves first telling Conpack that you are choosing the levels and how many levels you want. Then you assign a value for each level.

Nineteen hand-picked contour levels



Code segment from ccphand.f

```
       DATA RLEVEL /-20., -10., -5., -4., -3., -2., -1., 0., 1., 2., 3., 4.,
      + 5., 10., 20., 30., 40., 50., 100. /
       CALL CPSETI ('CLS - CONTOUR LEVEL SELECTION FLAG', 0)
       CALL CPSETI ('NCL - NUMBER OF CONTOUR LEVELS', INCL)
       DO 10, I=1, INCL
          CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
          CALL CPSETR ('CLV - CONTOUR LEVEL VALUES', RLEVEL(I))
          IF (AMOD (RLEVEL(I), 5.) .EQ. 0)
      +      CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
   10  CONTINUE
       CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
       CALL CPBACK (Z, RWRK, IWRK)
       CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETI ('CLS', cls)
      CALL CPSETI ('NCL', ncl)
      CALL CPSETI ('PAI', i)
      CALL CPSETR ('CLV', clv)
```

Arguments

CLS

Integer---The Contour Level Selection flag is critical to how contour levels are chosen. It not only determines how many contour levels are chosen, it also determines which contour level selection method is used. By default *n*=16.

0

Conpack does not pick contour levels at all; the current values of the parameters NCL, CLV, and associated arrays are not changed. They thus retain the values chosen by you, or by Conpack during a previous call.

-n

Where *n* is a positive integer, Conpack generates exactly *n* contour levels, splitting the range from the minimum field value to the maximum field value into *n*+1 equal intervals. This option is discussed further in the modifying Conpack-chosen levels modules Cp 4.7 and Cp 4.8.

+n

By default, Conpack chooses contour intervals at *base***k*, where the base value is some nice value either chosen by Conpack or set with the CIS parameter, and *k* is an integer. A nice value for *base* is defined by the contents of CIT, and chosen in such a way as to give at least *n* contour levels. Using the default CIT, you may get as many as 2*n* levels. In other words, *n*<=*kmax*-*kmin*+1<=2*n*. This option is discussed further in the default contour level selection modules Cp 4.4 through Cp 4.6.

If CIS>0 and CMN>CMX (your contour field minimum is set greater than your contour field maximum), then Conpack uses values of the form CIS**k*, where *k* is used as above. If CIS>0 and CMN<=CMX (your contour minimum is set less than or equal to your contour maximum), then Conpack uses values of the form CMN+CIS**k*, where *k* is an integer greater than or equal to CMN and less than or equal to CMX.

NCL

Integer---The Number of Contour Levels value is used by many Conpack routines. The Conpack initialization routines set NCL to zero. If CLS<>0, NCL is then set to some number of contour levels when contour levels are picked by CPPKCL or the contour-drawing routines.

PAI

Integer---The Parameter Array Index must be set to specify the array element that is to be changed. For example, if you want to change the third contour level value to 3.33, you would set PAI=3, then set CLV to 3.33.

CLV(NCL)

Real array---Each of the first NCL elements of the parameter array CLV (Contour Level Values) specifies a numerical value given to a contour level.

Lines 1 and 2 of the **ccphand.f** code segment create an array with data values that will become contour level values. Line 3 tells Conpack that we are choosing our own contour levels by setting CLS=0. Line 4 sets the number of contour levels to be the same as elements in our data array LEVEL. Lines 5 through 9 set each contour level value, first by specifying the contour level number with PAI, then by specifying a value with CLV. We also set CLU so that labels are drawn on levels divisible by 5. Conpack is then initialized, and contours are drawn normally. Notice that contour level values do not need to be assigned in any particular order.

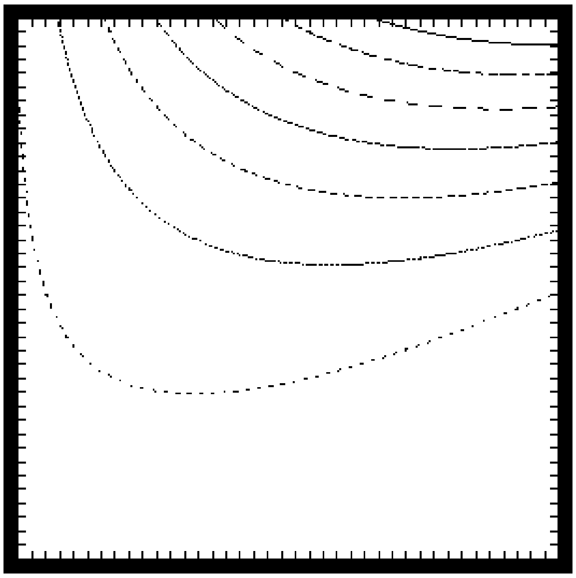
Exercises

1. Change **ccphand.f** so that it draws contour levels at 5, 25, 35, 65, and 60.
2. Change **ccphand.f** so that it draws contour levels at (10, 0, 10, 20, . . . 100). You have learned two ways to do this.

Cp 4.11 Line attributes: Line dash patterns

Contour line patterns are controlled by the CLD parameter. CLD is a parameter array that allows you to choose different line patterns for each contour level.

Contour line patterns



Code segment from ccpcld.f

```
      DATA IBTS / 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
     1            1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
     2            1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
     3            1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
     4            1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
     5            1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1,
     6            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 /

      CALL CPSETI ('CLS - CONTOUR LEVEL SELECTION FLAG', -7)
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (Z, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NOCL)
      DO 11, I=1, NOCL
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
```

```
              CALL CPSETI ('CLD - CONTOUR LINE DASH PATTERN', IPAT (IBTS (1, I)))
        11  CONTINUE
            CALL CPBACK (Z, RWRK, IWRK)
            CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('PAI', i)
CALL CPSETI ('CLD', icld)
CALL CPSETC ('CLD', 'string')
```

Arguments

CLD(NCL)

> *Integer array or Character expression array*---Each element of the Contour Line Dash pattern parameter array specifies a dash pattern to be drawn at a specific contour level if DPU<>0. DPU is the Dash Pattern Use flag. By default, DPU=3.

Discussion

Setting values of CLD may not affect contour line patterns when contour lines are drawn with CPCLDM. CLS is primarily used when drawing contour lines with CPCLDR. There are two methods for setting CLD; you can set it either as an integer array or as a character expression array.

When CLD is an integer array, the dash pattern relies on a 16-bit binary number where each 1 produces a dash and each 0 produces a space. You need to choose a dash pattern in binary, like "1111000011110000", then convert it to decimal:

```
CALL CPSETI ('CLD', 61680)
```

In this scenario, a solid line has the value 65535. Note that CLD is set with the CPSETI routine in this case.

When CLD is a character expression array, the dash pattern allows you to make patterns of nearly any Fortran character. Dollar signs ($) are used to specify dashes, and single quotation marks (') are used to specify blanks. So, to get the same dash pattern as in the DASHDB example:

```
CALL CPSETC ('CLD', '$$$$''''''''$$$$''''''''')
```

Note that when you set CLD with a character string, you need to call CPSETC. Also, since CLD is an array, you must use PAI with CLD and set a dash pattern for each contour line to be drawn. Three elements of CLD, with negative array indices, allow you to modify some lines that are not contour lines:

```
------------------------------------------------------
PAI  CLD
------------------------------------------------------
-1   Specifies a dash pattern for the edge of the grid
-2   Specifies a dash pattern for the edge of any area
      filled with special values (SPV)
-3   Specifies a dash pattern for the edge of any area
      in which CPMPXY returns an out-of-range
```

```
      value (ORV)
------------------------------------------------------
```

Dash patterns are assigned to contour levels in the order of the CLD array, not contour level value. For example, if you assign:

```
----------------------------------
CLD[1] = short dashes  CLV[1] = 6.
CLD[2] = long dashes   CLV[2] = 4.
CLD[3] = solid         CLV[3] = 5.
----------------------------------
```

your lowest contour line (4.0), has a long dash pattern, your middle contour line (5.0) is solid, and your highest contour line (6.0) has a short dash pattern.

Lines 1 through 7 of the **ccpcld.f** code segment declare the binary bit patterns that we want to convert into decimal for use as dash patterns. Line 9 forces Conpack to use only seven contour lines since we'll only define seven dash patterns. Line 10 initializes Conpack, and line 11 picks the contour levels so that we can change their dash patterns. Line 12 retrieves the number of contour levels so that the code is more flexible, and lines 13 through 16 set a different dash pattern for each contour line.

Line 15, the heart of this process, first retrieves one of the dash patterns, then converts it to a decimal number using a function discussed later in this module. Then the contour line dash pattern is set using Conpack parameter CLD as an integer array. Line 17 draws a background, and line 18 draws contours.

In many cases, it can be cumbersome to convert from a 16-bit binary number to a decimal number. You can make this conversion with the UNIX command **dc** if you want to hard-wire your dash patterns. Or you can use this Fortran function to construct dash patterns inside your program:

```
      INTEGER FUNCTION IPAT (IBTS)
      INTEGER IBTS (16)
      IPAT = 0
      DO 101, I=1, 16
          IPAT=IOR(ISHIFT(IPAT,1),IBTS(I))
101   CONTINUE
      RETURN
      END
```

IBTS is an integer array in which you store each element of your 16-bit binary pattern. ISHIFT(M,N) is a function supplied with NCAR Graphics that shifts M left by N bits. IOR(K,L) is an NCAR Graphics-supplied function that does a logical "or" of K and L. The function IPAT is included in the **ccpcld** example.

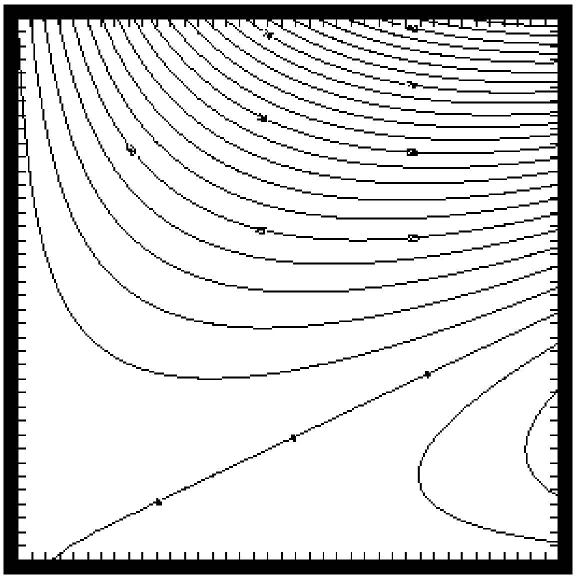
Exercises

1. Starting with **ccpcld.f**, use the default contour levels and make lines below zero have the pattern "$$$''''''''''", the zero line have the pattern "zero", and the lines above zero have the pattern "$$$$$$".
2. Using either the **dc** command or IOR and ISHIFT, create a dash-dot-dash-dot pattern, and draw the perimeter with it. **Hint:** Use a single dash to represent the dot in the pattern.

Cp 4.12 Line attributes: Line thickness

Contour line thickness, or width, is controlled by the CLL parameter. CLL is a parameter array, allowing you to choose different line widths for each contour level.

Contour line patterns



Code segment from ccpcll.f

```
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (Z, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCON)
      DO 11, I=1, NCON
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETR ('CLL - CONTOUR LINE LINE WIDTH', REAL (I) / 3.)
 11   CONTINUE
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETI ('PAI', i)
      CALL CPSETR ('CLL', cll)
```

Arguments

CLL(NCL)

> *Real array*---Each element of the Contour Line Line width parameter specifies a line width to be drawn with the associated element of the contour level value array CLV. Each line width is expressed as a multiple of the normal line width; values less than or equal to zero imply that the line width should not be set. Alternatively, you can change all line widths simultaneously by

calling GKS routine GSLWSC prior to the Conpack drawing routines.
There are three special elements in CLL, corresponding to negative index values:

```
--------------------------------------------
CLL(-1)  Specifies a line width for the edge
          of the grid.
CLL(-2)  Specifies a line width for the edge
          of any area filled with special
          values.
CLL(-3)  Specifies a line width for the edge
          of any area in which CPMPXY
          returns an out-of-range value.
--------------------------------------------
```

Line widths are assigned to contour levels in the order of the CLL array, not contour level value. For example, if you assign:

```
----------------------------
CLL[1] = thin    CLV[1] = 6.
CLL[2] = thick   CLV[2] = 4.
CLL[3] = medium  CLV[3] = 5.
----------------------------
```

your lowest contour level (4.0), has a thick line, your middle contour line (5.0) has a medium line, and your highest contour line (6.0) has a thin line.

Discussion

Line 1 of the **ccpcll.f** code segment initializes Conpack. Line 2 forces contour lines to be chosen so that we can modify them. Line 3 gets the number of contour levels, then lines 4 through 7 assign increasingly thick widths to each contour line by cycling through each contour level in a DO loop. This generates the thick lines in the upper right corner of the graphic. Lines 8 and 9 draw contours normally.

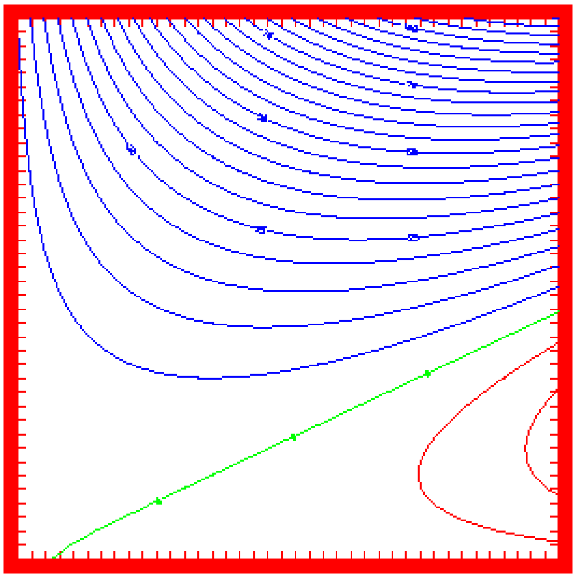
Exercises

1. Using the **ccpcll** example, set all lines to the default width, but make the zero line twice as thick as the rest.
2. Using the **ccpcll** example, make the perimeter three times thicker than normal.
3. Using the GKS routine GSLWSC, set all contour line widths to three times their normal thickness.

Cp 4.13 Line attributes: Line color

Contour line color is controlled by the CLC parameter and the GKS color-setting routine GSCR. CLC is a parameter array, allowing you to choose different line colors for each contour level.

Contour line color



Code segment from ccpclc.f

```
      CALL COLOR
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (Z, RWRK, IWRK)
      CALL CPGETI (’NCL - NUMBER OF CONTOUR LEVELS’, NCON)
      DO 11, I=1, NCON
         CALL CPSETI (’PAI - PARAMETER ARRAY INDEX’, I)
         CALL CPGETR (’CLV - CONTOUR LEVEL VALUES’, CVAL)
         IF (CVAL .LT. 0.0) THEN
            CALL CPSETI (’CLC - CONTOUR LINE COLOR INDEX’, 1)
         ELSE IF (CVAL .EQ. 0.0) THEN
            CALL CPSETI (’CLC - CONTOUR LINE COLOR INDEX’, 2)
         ELSE
            CALL CPSETI (’CLC - CONTOUR LINE COLOR INDEX’, 3)
         ENDIF
  11  CONTINUE

      SUBROUTINE COLOR
      CALL GSCR (1, 0, 0., 0., 0.)
      CALL GSCR (1, 1, 1., 0., 0.)
      CALL GSCR (1, 2, 0., 1., 0.)
      CALL GSCR (1, 3, 0., 0., 1.)
```

Synopsis

```
CALL CPSETI (’PAI’, i)
CALL CPSETI (’CLC’, iclc)
```

Arguments

CLC(NCL)

Integer array---Each element of the Contour Line Color parameter specifies a color index to be used when drawing a contour line at the level specified by the associated element of the contour level value array CLV. Each color index represents an RGB triplet set with the GSCR subroutine.

There are three special elements in CLC that correspond to negative index values:

```
---------------------------------------
CLC(-1)  Specifies a line color for the
          edge of the grid.
CLC(-2)  Specifies a line color for the
          edge of any area filled with
          special values.
CLC(-3)  Specifies a line color for the
          edge of any area in which
          CPMPXY returns an out-of
         -range value.
---------------------------------------
```

Note: Negative values of CLC elements imply no change in color.

Discussion

Lines 1 and 18 through 21 of the **ccpclc.f** code segment set up a color table so that lines can be drawn with different colors. An explanation of color tables appears in Chapter 7 "Color tables and color mapping systems" of *NCAR Graphics Fundamentals*.

Line 2 initializes Conpack, and line 3 forces contour lines to be chosen so they can be modified. Line 4 gets the number of contour levels, so that we can choose a color for each contour line using the DO loop beginning in line 5. Since in this case we want to be sure that line color is correlated to contour level value, we get the contour level value in line 7, and use it in the if block to assign red to lines with negative values, green to the zero line, and blue to lines above zero. After setting contour line colors, we draw contours normally.

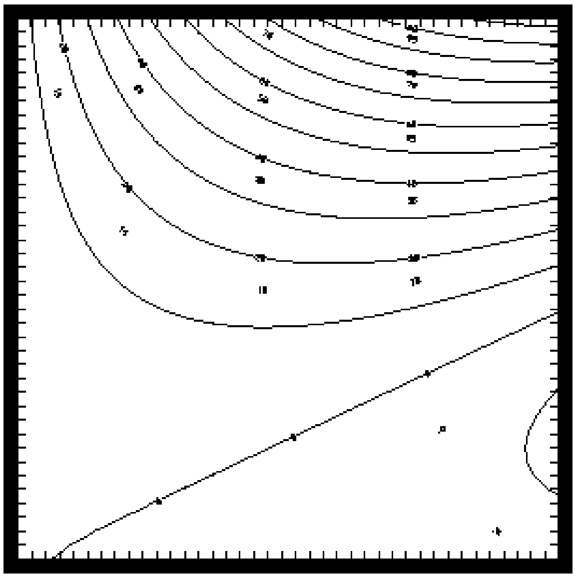
Exercises

1. Using the **ccpclc** example, make the perimeter line white. (There are two ways to do this.)
2. Using the **ccpclc** example, make each contour level a different color.

Cp 4.14 Line attributes: Turning contour line drawing off and on

Just as you can control specific values of individual contour lines, you can control exactly which lines are drawn. The parameter controlling this function, CLU, does two duties. First it determines whether or not a line will be drawn, then it determines if the line will be labeled. Control over the label-drawing aspect of the CLU parameter is discussed in module "Cp 6.5 Selecting lines for labels: Labeling specific lines."

Drawing individual contour lines



Code segment from ccpclu.f

```
      CALL CPSETI ('CLS - CONTOUR LEVEL SELECTION FLAG', 0)
      CALL CPSETI ('NCL - NUMBER OF CONTOUR LEVELS', 40)
      DO 11, I=1, 40
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETR ('CLV - CONTOUR LEVEL VALUES', 5.0 * REAL(I) - 15.)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', MOD (I, 4))
  11  CONTINUE
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETI ('CLU', iclu)
```

Arguments

CLU(NCL)

Integer array---Each element of the Contour Level Use flags parameter array indicates how the contour level indicated by the associated element of the CLV array is drawn.

0
: No contour line or labels are drawn at this level.

1
: A contour line is drawn without labels.

2
: Contour labels are drawn, but no line is drawn.

3
: Both a contour line and labels are drawn.

There are three special elements in CLU corresponding to negative index values:

```
---------------------------------------------
CLU(-1)  Specifies a flag for the edge of the
          grid.
CLU(-2)  Specifies a flag for the edge of any
          area filled with special values.
CLU(-3)  Specifies a flag for the edge of any
          area in which CPMPXY returns an
          out-of-range value.
---------------------------------------------
```

In each case, if the flag is zero, the associated edge is not drawn, otherwise an edge is drawn. No labels are drawn for these special elements of CLU.

Discussion

Study the **ccpclu** example to see that the following values are assigned to these parameters:

```
-----------------------
PAI  CLV(PAI)  CLU(PAI)
-----------------------
1    -10.0     1
2    -5.0      2
3    0.0       3
4    5.0       0
5    10.0      1
6    15.0      2
-----------------------
```

Also notice that contour level 10.0 is a solid line, contour level -5.0 is labeled but not drawn, contour level 0.0 is drawn and labeled, and contour level 5.0 is neither drawn nor labeled, in accordance with the CLU parameter.

Hence, line 1 of the **ccpclu.f** code segment tells Conpack that we will choose the contour levels. Line 2 sets the number of contour levels to 40, and lines 3 through 7 assign contour level values and set the CLU parameter to cycle through each of its four options. Lines 8 through 10 draw contours normally.

Exercises

1. Modify **ccprect.f** so that every contour level is drawn but not labeled.
2. Study the **ccpspv** example, and modify the code so it does not draw a boundary for the special value area.

Cp 5. Filling contour levels

What you have learned about filling areas with the Ezmap utility helps you understand how to fill contour levels with the Conpack utility.

Before you fill areas using Ezmap, you need to initialize Areas with the Areas initialization routine. You then initialize Ezmap with some special Ezmap initialization routines. Finally, you can fill regions you defined with Ezmap by calling the Areas routine ARSCAM. You fill contour levels with Conpack and Areas in exactly the same way.

Conpack functional outline

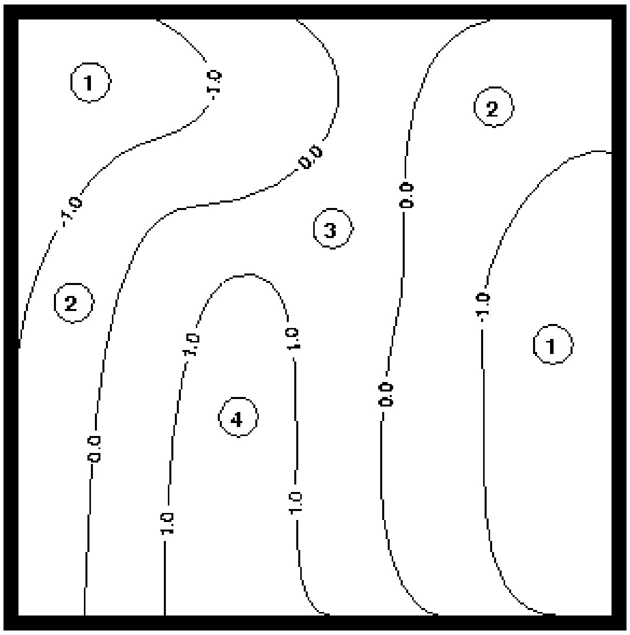
```
----------------------------------------------------------
    1.  Open GKS
    2.  Set window and viewport
    3.  Put in titles and other labels
 *  4.  Set up general parameters
 *  5.  Set up area maps
 *  6.  Initialize Conpack
 *  7.  Force Conpack to choose contour levels if necessary
 *  8.  Modify Conpack-chosen parameters
 *  9.  Draw labels
    10.  Draw background
 *  11.  Draw contour lines
    12.  Call FRAME
    13.  Close GKS
----------------------------------------------------------
```

* Steps discussed in this section.

Cp 5.1 Area identifiers in Conpack

Recall from the Areas chapter that area identifiers are defined as numbers specifying an area to the right or to the left of an edge. Conpack uses the concept of above a contour line or below a contour line to assign area identifiers. Contours are always drawn so values less than the contour level ("below" it) are to the left, and values greater than or equal to the contour level ("above" it) are to the right. So areas "below" the contour in Conpack are equivalent to areas to the "left" of an edge in the Areas utility; areas "above" the contour are equivalent to areas to the "right" of an edge.

Linking an area identifier to a contour level



Synopsis

```
CALL CPSETI ('AIA', iaia)
CALL CPSETI ('AIB', iaib)
```

Arguments

AIA(NCL)

Integer array---Each element in the Area Identifier Above parameter array is an area identifier for the level above the contour line defined by the corresponding element in the CLV array.
AIA has three special elements that correspond to negative values of PAI:

```
----------------------------------------------
PAI  AIA
----------------------------------------------
-1   Specifies the area identifier for the area
      outside the edge of the grid
-2   Specifies an area identifier for any
      region filled with special values
-3   Specifies an area identifier for any area
      in which CPMPXY returns the out-of
     -range-value ORV
----------------------------------------------
```

AIB(NCL)

Integer array---Each element in the Area Identifier Below parameter array is an area identifier for the level below the contour line defined by the corresponding element in the CLV array.

Discussion

Notice in the illustration that all the areas above contour level -1.0 have area identifiers of 2, even though they are not connected. This is desirable so that you can do things like fill all contour levels of a given value with the same color or pattern. Also, because area identifiers are assigned using the concept of above and below (this is unique to contour plots), you don't need to know the direction in which contour lines are drawn.

If AIA=AIB=0 for a particular contour level, then that level is ignored by the CPCLAM subroutine when it adds contour levels to the area map. Otherwise, that contour level is added to the area map, and AIA and AIB are used as the area identifiers for each side of the contour line.

When you let Conpack choose contour levels, then AIA(I)=I+1, and AIB(I)=I, where I is the Ith contour level. We use this automatic numbering of contour levels in the next module.

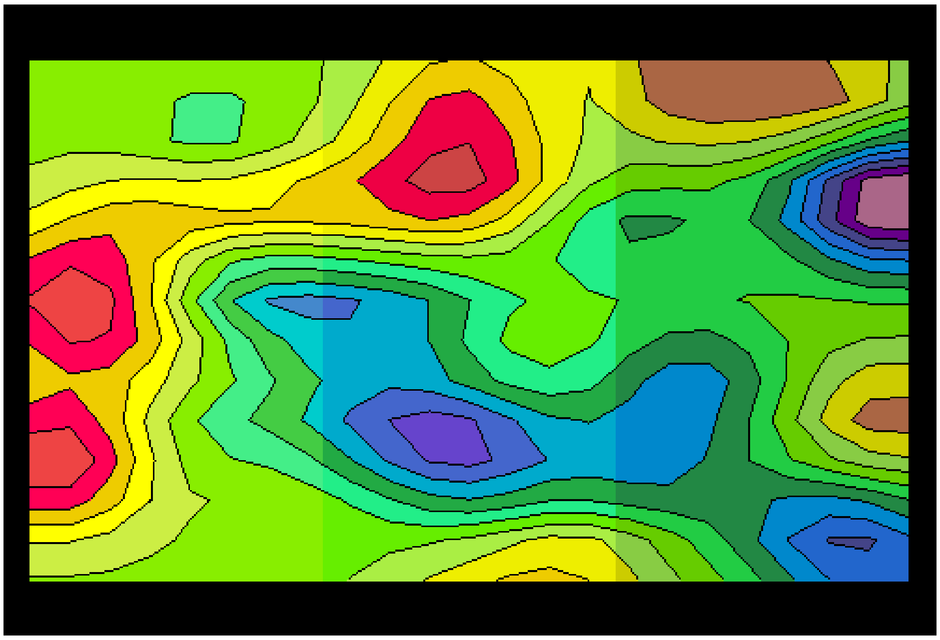
Note: If you set the elements of AIA and AIB manually, you should make sure that they are consistent. In other words, if the contour levels with nonzero values of AIA and AIB are $c1$, $c2$, ... cn, where $c1 < c2 < \ldots < cn$, and ak is the area identifier above ck, and bk is the area identifier below ck, then for all $1 \le k \le n-1$, $ak = bk+1$. Otherwise you will confuse the Areas utility, and strange results will occur.

Cp 5.2 Group identifiers in Conpack

An earlier part of this Conpack chapter stated that contour lines are added to group 3 and Conpack vertical strips are added to group 4. However, if this is inconvenient for your purposes, you can change it. Three parameters control the group in which Conpack puts things: GIC for contour lines, GIL for label boxes, and GIS for vertical strips.

Some hardware devices will fail when attempting to fill an area that is defined by too many points. To reduce the number of points in any given area, you can define vertical strips, and then fill the areas within these strips. As you specify more strips, you reduce the number of points that define any given area in a contour level.

Vertical strips in Conpack



Code segment from ccpvs.f

```
      CALL CPSETI ('NVS - NUMBER OF VERTICAL STRIPS', 3)
      CALL CPRECT (ZDAT, 23, 23, 14, RWRK, LWRK, IWRK, LWRK)

      SUBROUTINE FILL (XCRA, YCRA, NCRA, IAREA, IGRP, NGRPS)
         IFILL = 0
         DO 101, I=1, NGRPS
                IF (IGRP(I) .EQ. 3) IFILL = IAREA(I)
 101     CONTINUE
```

Synopsis

```
      CALL CPSETI ('GIC', igic)
      CALL CPSETI ('GIL', igil)
      CALL CPSETI ('GIS', igis)
      CALL CPSETI ('NVS', nvs)
```

Arguments

GIC

> *Integer*---The Group Identifier for Contour lines parameter allows you to specify the group in an area map to which you want contour lines added. GIC=3 by default.

GIL

> *Integer*---The Group Identifier for Label boxes parameter allows you to specify the group in an area map to which you want label boxes added. GIL=3 by default.

GIS

> *Integer*---The Group Identifier for vertical Strips parameter specifies the group identifier for a group of edges added to an area map by the subroutine CPCLAM in order to create a set of vertical strips. GIS is only used if NVS<>0. By default, GIS=4.

NVS

> *Integer*---The Number of Vertical Strips parameter specifies how many vertical strips are to be used. When NVS is nonzero, an extra group of edges, with group identifier GIS, is added to the area map by CPCLAM. These edges include the boundary of the viewport and vertical lines that break the viewport area into NVS vertical strips. The area identifier for the outside of the viewport is -1, and all other area identifiers are 0. By default, NVS=1.

Discussion

Vertical strips are not needed by most modern hardware. However, it is important to understand the concept, since by default a single vertical strip is created by every call to CPCLAM. This is why our filling and shading routines check each group and area identifier, and sift out only the area identifiers specific to the group that we want to shade or color. Lines 5 through 8 of the **ccpvs.f** code segment select only area identifiers belonging to the contour level group.

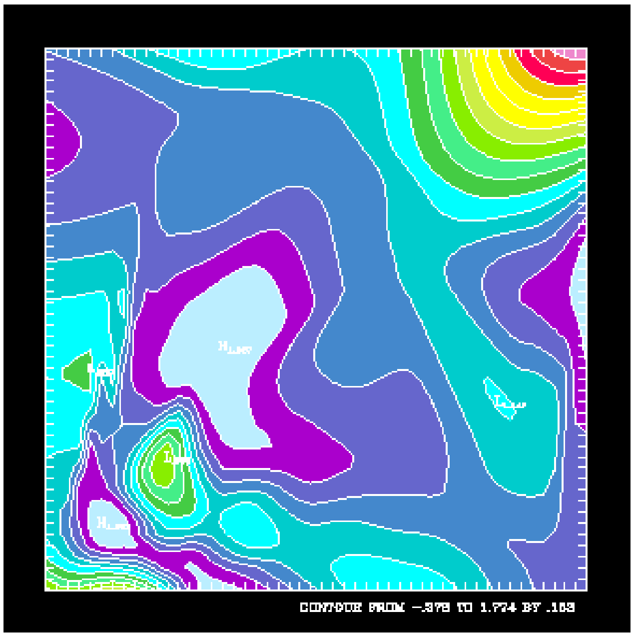
Notice that line 1 of the **ccpvs.f** code segment sets the number of vertical strips to 3. Then line 2 initializes Conpack normally. Vertical strips are usually invisible when finished, but in this example, they are given different color shades to emphasize them for you to study.

GIC, GIL, and GIS must be set before any call to CPCLAM or CPLBAM so they will be effective; this is discussed in the next module. Occasionally you may want to set GIC and GIL because you have another edge group that is already assigned to group 3. However, unless you need to change GIC, GIL, or GIS, it is recommended that you do not change their values.

Cp 5.3 Initialize Conpack with Areas

As in Ezmap, you need three calls to initialize Conpack with Areas. You need to initialize Areas by calling ARINAM, initialize Conpack by calling CPRECT, CPSPS1, or CPSPS2, and add contours to the area map using CPCLAM.

Conpack with Areas



Code segment from ccpfil.f

```
      EXTERNAL FILL
      CALL COLOR
      CALL ARINAM (MAP, LMAP)
      CALL CPSETI ('CLS - CONTOUR LEVEL SELECTION FLAG', NCL)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL GSFAIS (1)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, FILL)
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL CPLBDR (ZREG, RWRK, IWRK)
      CALL CPCLDR (ZREG, RWRK, IWRK)
```

Synopsis

```
      CALL ARINAM (MAP, LMAP)
      CALL CPRECT (ZREG, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPSPS1 (ZSPS, KSPS, MSPS, NSPS,RWRK, LRWK, IWRK, LIWK,ZREG,
     +             LZRG)
      CALL CPSPS2 (XSPS, YSPS, ZSPS, KSPS,MSPS, NSPS, RWRK, LRWK, IWRK,
     +             LIWK, ZREG, LZRG)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
```

Arguments

The first three arguments to CPCLAM are arrays used in the last call to CPRECT, CPSPS1, or CPSPS2, the contents of which must not have been changed since that call.

ZREG(K,M) or **ZREG(LZRG)**

Real array, Input---The array containing regularly spaced data to be contoured.

RWRK(LRWK)

Real array, Workspace---The real workspace array.

IWRK(LIWK)

Integer array, Workspace---The integer workspace array.

MAP(LMAP)

Integer array, Workspace---The array containing the area map to which contour lines are to be added. Dimensioned as specified in the last call to the Areas routine ARINAM.

Discussion

The routine CPCLAM, which adds contour lines generated from the data in the ZREG array to the area map in the MAP array, may be called at any time after the initialization call to CPRECT, CPSPS1, or CPSPS2. The area map must previously have been initialized by a call to the routine ARINAM, in the Areas utility.

The contour lines added to the area map are as specified by the first NCL (Number of Contour Lines) elements of the parameter arrays CLV, AIA, and AIB. If NCL is zero, CPPKCL is called to generate these values.

If, for a given value of I between 1 and NCL inclusive, the Ith element of either AIA or AIB is nonzero, then the contour lines specified by the Ith element of CLV are added to the area map, with the Ith element of AIA as the area identifier for the area "above" each line (where field values are greater than they are along the line), and with the Ith element of AIB as the area identifier for the area "below" each line (where field values are less than they are along the line).

If a given contour level occurs twice in the array CLV, then the desired value of AIA may be associated with one contour level, and the desired value of AIB may be associated with the other contour level that has the same value.

If the parameter T2D (Tension on 2-Dimensional splines) has a nonzero value, the contour lines are smoothed, using cubic splines under tension.

Four other types of lines are added to the area map by CPCLAM:

- The edge of the current viewport is placed in groups GIC and GIS, and a set of vertical strips is placed in group GIS.
- The edge of the grid.
- The edges of special-value areas, if any.
- The edges of out-of-range areas, if any.

The area identifier for the outside of the viewport is always -1. You can specify area identifiers for these areas using negative values for PAI and setting values in AIA. Please see the previous module for details.

Lines are added to the area map in the following order:

1. The edge of the viewport and the vertical lines within it
2. The edges of the out-of-range areas, if any
3. The edge of the grid
4. The edges of the special-value areas, if any
5. The contour lines, in order of increasing contour level

The edge of the viewport may be added to the area map twice:

- As part of the edge group with group identifier GIC
- As part of the edge group with group identifier GIS

The object of number 1 in the preceding sequence is to prevent problems that arise when mapping is turned on and the mapping function has a line of discontinuity (for example, when using Ezmap with a cylindrical equidistant projection). Vertical lines are added by the subroutine CPCLAM to break up the areas represented by the area map into smaller pieces. Whether this is done or not is under your control, by means of the NVS and GIS parameters. For more information, see module "Cp 5.2 Group identifiers in Conpack."

If, during the last call to CPRECT, CPSPS1, or CPSPS2, the data being contoured were found to be essentially constant, then no contour lines are added to the area map; however, the other lines are added.

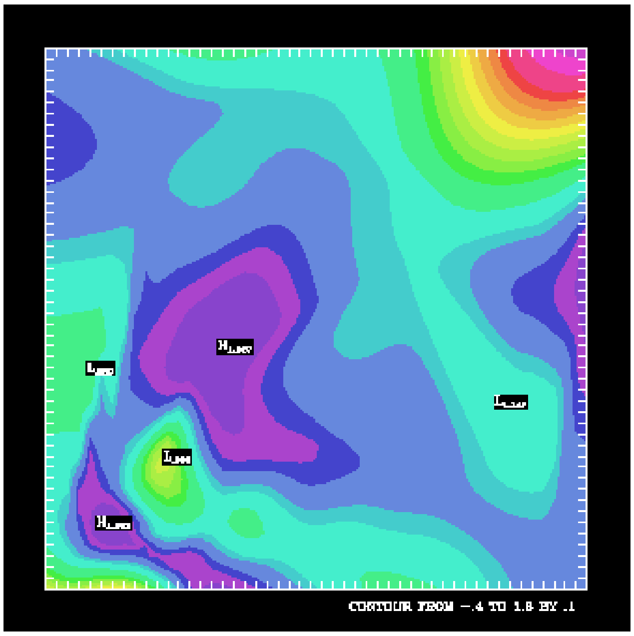
Line 1 of the **ccpfil.f** code segment declares the fill routine to be external so we can pass it into ARSCAM. Line 3 initializes Areas. Line 4 sets the number of contour levels to a fixed value, in this case 15, or the number of colors defined in the color table.

Line 5 initializes Conpack, and line 6 adds the contour lines to the area map. Line 7 sets the interior fill style to be solid so that we don’t get hollow fill, and line 8 fills the contour levels using the Areas routine ARSCAM. Lines 9 through 11 draw a perimeter, labels, and contour lines.

Cp 5.4 Adding label boxes to the area map

Examine the graphic in the previous module to see that the contour line labels have disappeared, and that the high and low labels are not readily distinguishable from the levels that contain them. In cases like this, you may want to place these labels in boxes to make them more visible. Since you use a masking procedure to draw these boxes and protect their interiors from contour lines and the fill options being used, the boxes must also be added to the area map.

Adding label boxes to an area map



Code segment from ccplbam.f

```
      EXTERNAL FILL
      CALL COLOR
      CALL ARINAM (MAP, LMAP)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL CPLBAM (ZREG, RWRK, IWRK, MAP)
      CALL GSFAIS (1)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, FILL)
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CALL CPLBAM (ZREG, RWRK, IWRK, MAP)
```

Arguments

The first three arguments to CPLBAM are arrays used in the last call to CPRECT, CPSPS1, or CPSPS2, the contents of which must not have been changed since that call.

ZREG

Real array, Input---The densely gridded data array.

RWRK

Real array, Workspace---The real workspace array.

IWRK

Integer array, Workspace---The integer workspace array.

MAP

Integer array, Workspace---The array holding the area map to which the label boxes will be added. MAP must be initialized with a call to ARINAM, which is part of the Areas utility, before calling CPLBAM.

Discussion

The CPLBAM routine, which adds label boxes to an area map, may be called at any time after the initialization call to CPRECT, CPSPS1, or CPSPS2, and after the initialization call to ARINAM. If, during the last call to CPRECT, CPSPS1, or CPSPS2, the data being contoured were found to be essentially constant, then the box for the constant field label is added to the area map. Otherwise, boxes for the information label, the high and low labels, and/or the contour line labels are added, as follows:

- A box for the information label is added only if the Conpack parameter ILT, which specifies the text of that label, is nonblank.
- Boxes for the high labels are added only if the parameter HIT, which specifies the text of those labels, is nonblank.
- Boxes for the low labels are added only if the parameter LOT, which specifies the text of those labels, is nonblank.
- Boxes for the contour line labels are added only if the parameter LLP, which specifies how those labels are to be positioned, has an absolute value of 2 or 3, and if, for some I between 1 and NCL, inclusive, the Ith element of the parameter array CLU has a value implying that contour lines at the contour level specified by the Ith element of CLV are to be labeled.

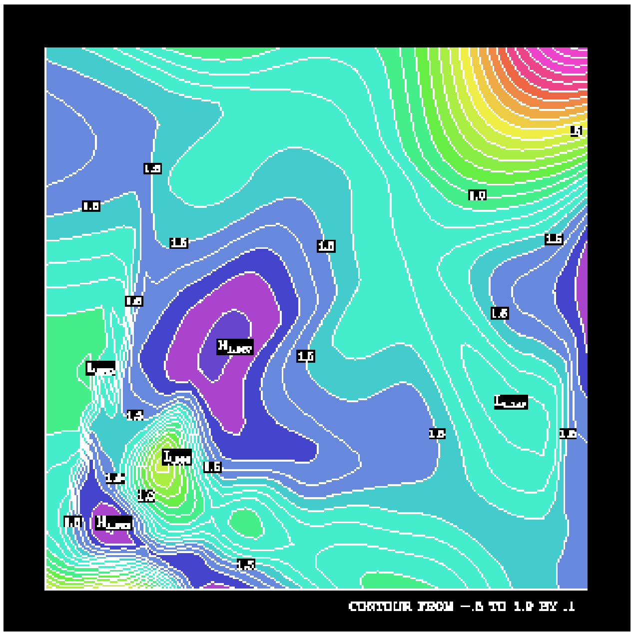
Remember that the box is added to the area map; it is not drawn. Moreover, if you draw contour lines without masking them against the area map, they may cross through label boxes.

Line 1 of the **ccplbam.f** code segment declares the fill routine to be external so that it can be passed to ARSCAM. Line 2 sets up our color table. Lines 3 through 6 initialize Areas, initialize Conpack, add contour lines to the area map, then add label boxes to the area map. Lines 7 and 8 color-fill the contour levels. Lines 9 and 10 draw a perimeter and labels.

Cp 5.5 Masking areas: Label boxes

CPCLDM allows you to draw contour lines masked by your area map. One of its arguments is a routine that decides whether to draw portions of the contour line based on area identifier information provided for them. This routine is discussed in detail in the next module.

Protecting label boxes from contour lines



Code segment from ccpcldm.f

```
      EXTERNAL FILL
      EXTERNAL CPDRPL

      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 2)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL ARINAM (MAP, LMAP)
      CALL CPLBAM (ZREG, RWRK, IWRK, MAP)
      CALL CPLBDR (ZREG, RWRK, IWRK)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, FILL)
      CALL CPCLDM (ZREG, RWRK, IWRK, MAP, CPDRPL)
```

Synopsis

```
      CALL CPCLDM (ZREG, RWRK, IWRK, MAP, MASK)
```

Arguments

The first three arguments to CPCLDM are arrays used in the last call to CPRECT, CPSPS1, or CPSPS2, the contents of which must not have been changed since that call. The area map, MAP, must also be initialized with a call to ARINAM before calling CPCLDM. The dimensions of the following arrays are passed into CPCLDM in COMMON blocks.

ZREG

Real array, Input---The densely gridded data array.

RWRK

Real array, Workspace---The real workspace array.

IWRK

Integer array, Workspace---The integer workspace array.

MAP

Integer array, Workspace---The array containing the area map to be used for masking the contour lines as they are drawn.

MASK

Subroutine---An external subroutine to process the polylines that result from masking the generated contour lines and other edges against the area map. MASK must be declared external in the routine that calls CPCLDM. You can either write your own MASK routine, or you can use the Conpack routine CPDRPL to protect label boxes. Module "Cp 5.6 Masking areas: Box masking routines" discusses the CPDRPL routine.

Discussion

The routine CPCLDM may be called at any time after the initialization call to CPRECT, CPSPS1, or CPSPS2, and after the initialization call to ARINAM. CPCLDM does not draw the lines; it generates them, masks them against a user-specified area map, and generates calls, one for each polyline resulting from the masking process, to the user-specified routine, MASK.

Each such polyline lies in precisely one of the areas defined by the area map. The MASK routine may use the information provided by its arguments (that describe the area the polyline is in) to decide whether or not to draw the polyline.

The contour lines generated are those specified by the first NCL elements of the parameter arrays CLV and CLU. If NCL is zero, CPPKCL is called to generate these values. Each element of CLV specifies a contour level and the corresponding element of CLU specifies whether or not contour lines are to be generated at that level. If the parameter T2D has a nonzero value, the contour lines are smoothed, using cubic splines under tension.

If the element of the parameter array CLU corresponding to PAI=-1 is nonzero, the edge of the grid is also generated. If the element of CLU corresponding to PAI=-2 is nonzero, the edges of special-value areas (if any) are generated. If the element of CLU corresponding to PAI=-3 is nonzero, the edges of out-of-range areas (if any) are generated. By default, none of these edges are generated.

Groups of lines are generated in the following order:

1. Contour lines for each of the specified levels, in ascending numeric order
2. The edges of special-value areas, if any
3. The edges of out-of-range areas, if any
4. The edge of the grid

The color, dash pattern, and line width to be used for the lines drawn may be specified by elements of the parameter arrays CLC, CLD, and CLL, respectively. Each of these contains elements corresponding to values of PAI from 1 to NCL and three special elements, corresponding to PAI=1, 2, and 3.

Before and after each group of lines is generated, the routine CPCHCL is called; a user-supplied version of this routine may override the settings for color, dash pattern, and line width. Also, of course, the routine MASK, which actually does the drawing, may override the settings of these quantities.

Conpack's Dash Pattern Use parameter DPU affects the pattern used to draw the lines. When DPU<=0, lines are drawn by calling the SPPS routine CURVE. The lines are all solid and unlabeled and specified dash patterns are not used. When DPU>0, lines are drawn by calling the Dashline routine CURVED. Lines are solid or dashed, depending on the dash pattern specified by the appropriate element of CLD.

If LLP=1 or LLP=1, then the dash pattern for the lines to be labeled is constructed by replicating, DPU times, the dash pattern specified by the appropriate element of CLD, and then appending to it the characters specified by the appropriate element of LLT. If LLP=1 or LLP=1, then label boxes are not created for contour line labels.

If, during the last call to CPRECT, CPSPS1, or CPSPS2, the data being contoured were found to be essentially constant, then no contour lines are generated. Instead, the constant-field label is written. Other lines are still generated.

Lines 1 and 2 of the **ccpcldm.f** code segment declare the fill and masking routines to be external so that they can be passed into ARSCAM and CPCLDM respectively. Line 4 chooses the regular labeling scheme so that label boxes can be drawn. Lines 5 and 6 initialize Conpack and Areas respectively.

Lines 7 and 8 add labels to the area map, then draw them. Similarly, line 9 adds contour lines to the area map, and line 11 draws them. By filling the contour levels before drawing the contour lines, we ensure that the contour lines are visible in the final plot.

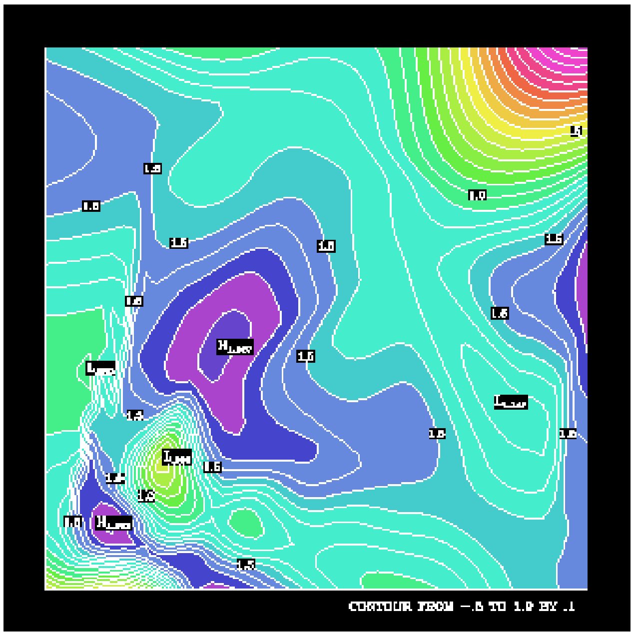
Exercises

1. Using the **ccpcldm** example, call CPCLDR instead of CPCLDM, and see what happens.

Cp 5.6 Masking areas: Box masking routines

CPCLDM requires a masking routine to decide when contour lines will be masked, and which lines should be drawn. Conpack offers the default masking routine CPDRPL to mask label boxes. If you need a smarter routine, for example one that masks contour levels only over the ocean, please see module "Cp 8.2 Masking areas: Label boxes and land masses." If you use CPDRPL, be sure to declare it external in the routine that calls CPCLDM.

Protecting label boxes from contour lines



Code segment from ccpdrpl.f

```
      IDR=1
      DO 101, I=1, NAI
         IF (IAI(I) .LT. 0) IDR=0
 10   CONTINUE
      IF (IDR .NE. 0) THEN
         IF (IDUF .EQ. 0) THEN
            CALL CURVE (XCS, YCS, NCS)
         ELSE
            CALL CURVED (XCS, YCS, NCS)
         ENDIF
      ENDIF
      RETURN
      END
```

Synopsis

```
CALL CPDRPL (XCS, YCS, NCS, IAREA, IGRP, NGRP)
```

Arguments

XCS(NCS)

Real array, Input---An array containing the X coordinates of NCS points defining a polyline.

YCS(NCS)

Real array, Input---An array containing the Y coordinates of NCS points defining a polyline.

NCS

Integer, Input---The number of points defining the polyline.

IAREA(NGRP)

Integer array, Input---An array of area identifiers for the area in which the polyline lies. For each I from 1 to NGRP, IAREA(I) is the area identifier of the area with respect to the edge group IGRP(I).

IGRP(NGRP)

Integer array, Input---An array of group identifiers. See the description of IAREA, above.

NGRP

Integer---The number of group identifiers in the array IGRP.

Discussion

CPDRPL is an internal Conpack routine, it must be declared external when used with CPCLDM. Documentation is provided here in case you need to modify it. CPDRPL uses a call to CURVE or a call to CURVED, depending on the current value of the DPU parameter.

If the only reason you are using CPCLDM instead of CPCLDR is to avoid drawing contour lines through labels, follow these steps:

1. In the subroutine that calls CPCLDM, insert the declaration EXTERNAL CPDRPL.
2. In the call to CPCLDM, use CPDRPL for the final argument.

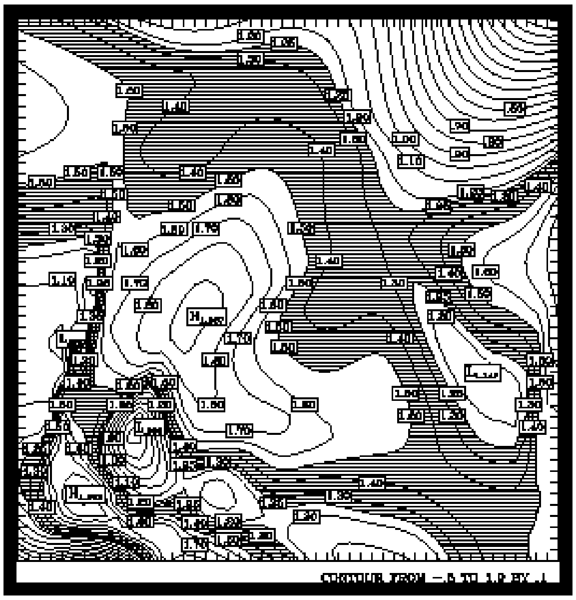
Each time CPDRPL is called, it draws the polyline defined by its first three arguments if, and only if, none of the area identifiers defined by the other three arguments is negative.

The code in the **ccpdrpl** example is taken from the default version of CPDRPL. Note that lines 1 through 4 of the **ccpdrpl.f** code segment first check for a negative area identifier, then turn off line drawing if it is negative. If the area identifier is positive (it's not in a label box or outside the perimeter), then line 6 checks to see if Conpack is drawing using Dashline, then draws the contour line in a way consistent with how the parameters are set.

Cp 5.7 Filling contour levels

Filling contour levels is just like filling geographic areas in Ezmap, only easier, because in this case you know exactly what your area identifiers are. In this case, it is only necessary to call ARSCAM with an appropriate fill routine.

Filling selected contour levels



Code segment from ccpscam.f

```
      EXTERNAL SFILL
      EXTERNAL CPDRPL

      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCL)
      DO 111, I=1, NCL
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
         CALL CPSETI ('AIA - AREA IDENTIFIER ABOVE', 0)
         CALL CPSETI ('AIB - AREA IDENTIFIER BELOW', 0)
111   CONTINUE
      CALL CPSETI ('NCL - NUMBER OF CONTOUR LEVELS', NCL + 2)
      CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', NCL + 1)
      CALL CPSETR ('CLV - CONTOUR LEVEL VALUES', 1.25)
      CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
      CALL CPSETI ('AIA - AREA IDENTIFIER ABOVE', 1)
      CALL CPSETI ('AIB - AREA IDENTIFIER BELOW', 2)
      CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', NCL + 2)
      CALL CPSETR ('CLV - CONTOUR LEVEL VALUES', 1.5)
      CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
      CALL CPSETI ('AIA - AREA IDENTIFIER ABOVE', 3)
      CALL CPSETI ('AIB - AREA IDENTIFIER BELOW', 1)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, SFILL)
      CALL CPCLDM (ZREG, RWRK, IWRK, MAP, CPDRPL)
```

Synopsis

```
CALL ARSCAM (MAP, XCS, YCS, MCS, IAREA, IGRP, ISIZ, FILL)
```

Arguments

MAP

Integer array, Workspace---The area map that has been initialized by a call to ARINAM and to which edges have been added by calls to AREDAM.

XCS(MCS), YCS(MCS)

Real arrays, Workspace---The X and Y coordinates defining the edge of a given area, for use by ARSCAM in calls to the area-processing routine FILL.

MCS

Integer, Input---The dimension of each of the arrays XCS and YCS.

IAREA(ISIZ), IGRP(ISIZ)

Integer arrays, Input---Arrays that are dimensioned ISIZ for use by ARSCAM in calls to the area-processing routine FILL.

ISIZ

Integer, Input---The dimension of each of the arrays IAREA and IGRP. ISIZ<=*n*, where *n* is the number of groups in the area map.

FILL

Subroutine---The user-supplied area-processing routine that is called by ARSCAM. FILL must be declared external in the routine that calls ARSCAM. FILL is described in full detail in the next module.

Discussion

ARSCAM is an Areas routine and is documented more completely in the Areas chapter: "Ar 3.4 Filling areas" and "Ar 3.5 Writing an area-fill routine." Because you should be comfortable with ARSCAM by now, the primary purpose of the **ccpscam** example is to illustrate how you might set up some area identifiers for contour levels in the case where you want to fill only a specific set of levels.

Lines 1 and 2 of the **ccpscam.f** code segment are required so that Fortran knows that the fill and contour line drawing routines are subroutines. Line 4 initializes Conpack, and line 5 forces contour lines to be chosen so that we can modify contour line information. Line 6 retrieves the number of contour levels, and lines 7 through 12 turn on contour line drawing and contour line labels for every line. They also turn off area identifier information for all contour lines. It is necessary to turn off area identifier information in this case because when we add the contour lines to shade between, we need to override Conpack's default area identifiers.

Line 13 tells Conpack that we're adding two new contour lines. Line 15 sets the value for the first new contour line at 1.25, and line 20 sets the value for the second at 1.5. Notice that Conpack has already chosen a contour line at 1.5. Conpack doesn't mind that it now has two contour lines with the same value. Because we set the area identifiers to zero for the contour at 1.5 that Conpack chose, Areas overrides those values with the new area identifiers that we set in lines 22 and 23. Notice that the area identifier for the area above the contour at 1.25 is set to 1 in line 17, and that the area identifier for the area below the contour at 1.5 is also set to 1 in line 23. It is critical that these two values match for correct area fill.

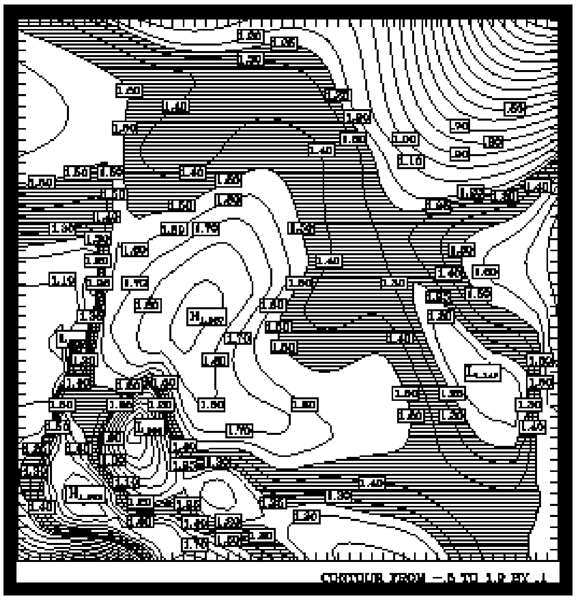
Exercises

1. Modify subroutine CCPSCM in the **ccpscam** example so that it accepts a minimum and maximum contour level value for filling, and so that it fills from that minimum to that maximum level.
2. Using the **ccpscam** example, modify subroutine CCPSCM so that every contour level has a different area identifier.

Cp 5.8 Filling contour levels: Writing a fill routine

It is easy to write a fill routine to handle contour levels. Since you assigned the area identifiers, and you know that contour levels are placed in group 3 unless you change the GIC parameter, all you do is set up a relationship between the area identifiers that you want to fill and choose a filling option.

Filling selected contour levels



Code segment from ccpscam.f

```
      SUBROUTINE SFILL (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)

      REAL XWRK(*), YWRK(*), ISCR(5000)
      INTEGER IAREA(*), IGRP(*), RSCR(5000)

      DO 10, I=1, NGRPS
         IF (IGRP(I) .EQ. 3) IAREA3 = IAREA(I)
  10  CONTINUE
      IF (IAREA3 .EQ. 1) THEN
```

```
10          CALL SFSETR ('SP - SPACING BETWEEN FILL LINES', .006)
11          CALL SFNORM (XWRK, YWRK, NWRK, RSCR, 5000, ISCR, 5000)
12      ENDIF
13      RETURN
14      END
```

Arguments

APR

Subroutine---The Area-Processing Routine that is supplied by the user and called by ARSCAM. APR must be declared external in the routine that calls ARSCAM. APR must have the following form:

```
SUBROUTINE APR (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)
DIMENSION XWRK(*), YWRK(*)
DIMENSION IAREA(*), IGRP(*)
```

(code to process the area defined by XWRK, YWRK, IAREA, and IGRP)

```
RETURN
END
```

XWRK(NWRK), YWRK(NWRK)

Real arrays, Input---The X and Y coordinates, in the fractional coordinate system, of NWRK points defining a polygonal area. The last of these points is a duplicate of the first. Holes in an area are traced in such a way as to maximize the probability of hardware fill working properly by using vertical lines to get to and from the holes and tracing them in the proper direction.

NWRK

Integer, Input---The number of X and Y coordinates in the arrays XWRK and YWRK.

IAREA(NGRPS), IGRP(NGRPS)

Integer arrays, Input---Hold NGRPS pairs of identifiers for the area defined by XWRK and YWRK. For each value of I from 1 to NGRPS, IAREA(I) is the area identifier for the area with respect to the group of edges specified by the group identifier IGRP(I).

NGRPS

Integer, Input---Number of values in IAREA and IGRP. NGRPS equals the number of groups of edges that you put in the area map.

Discussion

In the **ccpscam** example, the area-processing routine is named SFILL. Lines 1 through 5 of the **ccpscam.f** code segment set up the calling sequence and arrays as defined by ARSCAM. They also set up two new scratch arrays, ISCR and RSCR, so that Softfill has some integer and real workspace to work with. Lines 6 through 8 retrieve the area identifier associated with group 3, our contour levels in this plot. Lines 9 through 12 draw the horizontal fill lines to fill the areas that we gave area identifier 1 in the previous module.

Exercises

1. Modify SFILL in the **ccpscam** example so that it fills the region with a solid blue color rather than lines.
2. Using exercise 2 from module "Cp 5.7 Filling contour levels," fill each contour level with a different pattern.

Cp 6. Contour line labels

Because of the complexity of the Conpack labeling scheme, labeling has been divided into two sections, by function:

- Contour line labels (selecting lines to be labeled, label placement, label attributes)---Section "Cp 6. Contour line labels."
- General labels (plot annotations, numeric label control, titling)---Section "Cp 7. General labels."

Note: Many contour line label options are covered in section "Cp 7. General labels" because they share common parameters or functions with general labels.

Calls to set up contour line labels appear in four places in the Conpack functional outline:

- General label parameters can be set before the initializing subroutine.
- Parameters that work on specific contour levels (such as CLU) must be set after contour lines are chosen.
- The label-drawing routine CPLBDR draws contour line labels in some cases.
- The line-drawing routine CPCLDR draws contour line labels by default.

Conpack functional outline

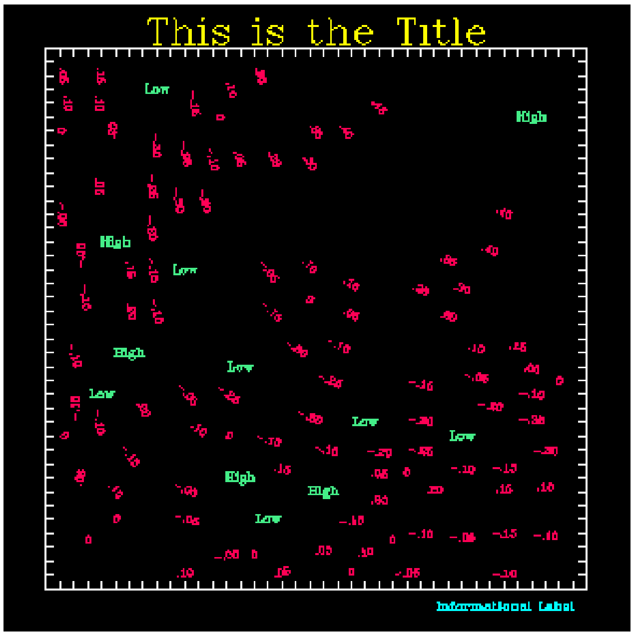
```
----------------------------------------------------------
    1.  Open GKS
    2.  Set window and viewport
    3.  Put in titles and other labels
 *  4.  Set up general parameters
    5.  Set up area maps
    6.  Initialize Conpack
    7.  Force Conpack to choose contour levels if necessary
 *  8.  Modify Conpack-chosen parameters
 *  9.  Draw labels
    10.  Draw background
 *  11.  Draw contour lines
    12.  Call FRAME
    13.  Close GKS
----------------------------------------------------------
```

* Steps discussed in this section.

Cp 6.1 Drawing labels

Conpack produces several types of labels for contour plots: contour line labels, labels for the high and low points in your plot, and an information label. You must call the CPLBDR routine to draw any of these labels, except for the contour line labels that are drawn by default. Other utilities produce other types of labels used with contour plots; these are discussed in section "Cp 7. General labels."

Plot annotations



Code segment from ccplbdr.f

```
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPLBDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPLBDR (ZREG, RWRK, IWRK)
```

Arguments

All three arguments to the CPLBDR call are arrays used in the last call to CPRECT, CPSPS1, or CPSPS2, the contents of which must not have been changed since that call. Since the dimensions of the following array are passed in common blocks, they are not shown here.

ZREG

> *Real array, Input*---The densely gridded data array.

RWRK

Real array, Workspace---The real work array.

IWRK

Integer array, Workspace---The integer work array.

Discussion

CPLBDR may be called at any time after the initialization call to CPRECT, CPSPS1, or CPSPS2. If, during the last call to CPRECT, CPSPS1, or CPSPS2, the data being contoured were found to be essentially constant, then the constant field label is drawn. Otherwise, the information label, the high and low labels, and/or the contour line labels are drawn, as follows:

- The information label is drawn only if the parameter ILT, which specifies the text of that label, is not blank.
- High labels are drawn only if the parameter HIT, which specifies the text of those labels, is not blank.
- Low labels are drawn only if the parameter LOT, which specifies the text of those labels, is not blank.
- Contour line labels are drawn by CPLBDR only if the parameter LLP, which specifies how those labels are to be positioned, has an absolute value of 2 or 3, and if, for some I between 1 and NCL inclusive, the Ith element of the parameter array CLU has a value implying that contour lines at the contour level specified by the Ith element of CLV are to be labeled.

The parameters referring to high, low, and information labels in the preceding discussion are described in detail in section "Cp 7. General labels." Because of the complexity of contour line labeling schemes, section "Cp 6. Contour line labels" provides a detailed description of most of the contour line label options, and section "Cp 7. General labels" describes the rest of the contour line label options.

Line 1 of the **ccplbdr.f** code segment initializes Conpack. Line 2 draws a perimeter, and line 3 draws labels using CPLBDR. Notice that we don't call CPCLDR in this program. By not calling CPCLDR, contour lines are never drawn. Also, by not drawing the contours before calling CPLBDR to draw labels, the information label in the bottom right corner of the plot fails to get the information it needs to label minimum and maximum values as well as contour interval.

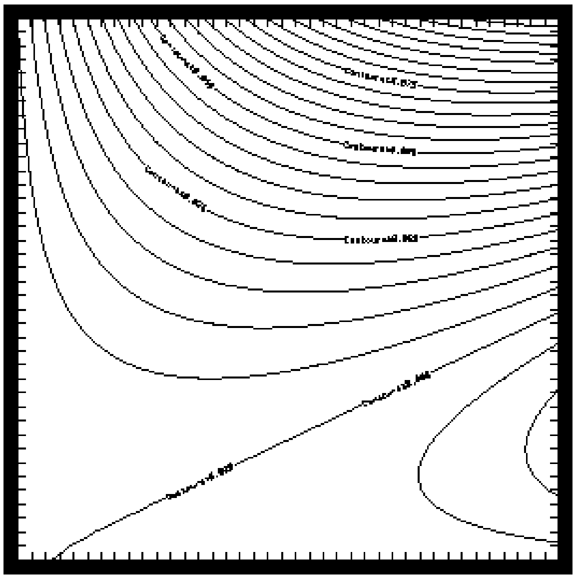
Exercises

1. Add a CPLBDR call to the **ccptitle** example after the call to CPCLDR. Notice which of the labels are new. Why are contour line labels drawn before the call to CPLBDR is added?

Cp 6.2 Forcing labels to be chosen

Occasionally you need to force labels to be chosen so that you can modify specific line labels, or so that you can check a specific line label. The routine CPPKLB forces labels to be chosen.

Changing a specific line label



Code segment from ccpklb.f

```
      CHARACTER*16 STRING
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (Z, RWRK, IWRK)
      CALL CPPKLB (Z, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NOCL)
      DO 10, I=1, NOCL
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPGETR ('CLV - CONTOUR LEVEL VALUES', CLVL)
         WRITE (STRING, 5) CLVL
         CALL CPSETC ('LLT - LINE LABEL TEXT STRING', STRING)
   10 CONTINUE
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
    5 FORMAT ('Contour at ', F5.3)
```

Synopsis

```
CALL CPPKLB (ZREG, RWRK, IWRK)
```

Arguments

All three arguments to the CPPKLB call are arrays used in the last call to CPRECT, CPSPS1, or CPSPS2, the contents of which must not have been changed since that call. Since the dimensions of the following arrays are passed in common blocks, they are not shown here.

ZREG

Real array, Input---The densely gridded data array.

RWRK

Real array, Workspace---The real workspace array.

IWRK

Integer array, Workspace---The integer workspace array.

Discussion

Normally, CPPKLB is called by Conpack when labels for the contour levels are needed. You can call CPPKLB directly (after the initialization call to CPRECT, CPSPS1, or CPSPS2) when you want to modify the resulting parameter arrays that specify the labels.

If the constant field found flag CFF is nonzero, indicating that the data were found to be essentially constant on the last call to CPRECT, CPSPS1 or CPSPS2, CPPKLB does nothing. Otherwise, CPPKLB examines the first NCL elements of the parameter array CLV (which defines the contour levels) and the associated parameter arrays, looking for levels that are to be labeled (CLU = 2 or 3) for which no label is specified (LLT = ' '). (The value of LLT is a single blank.) If any such levels are found, CPPKLB generates labels for them.

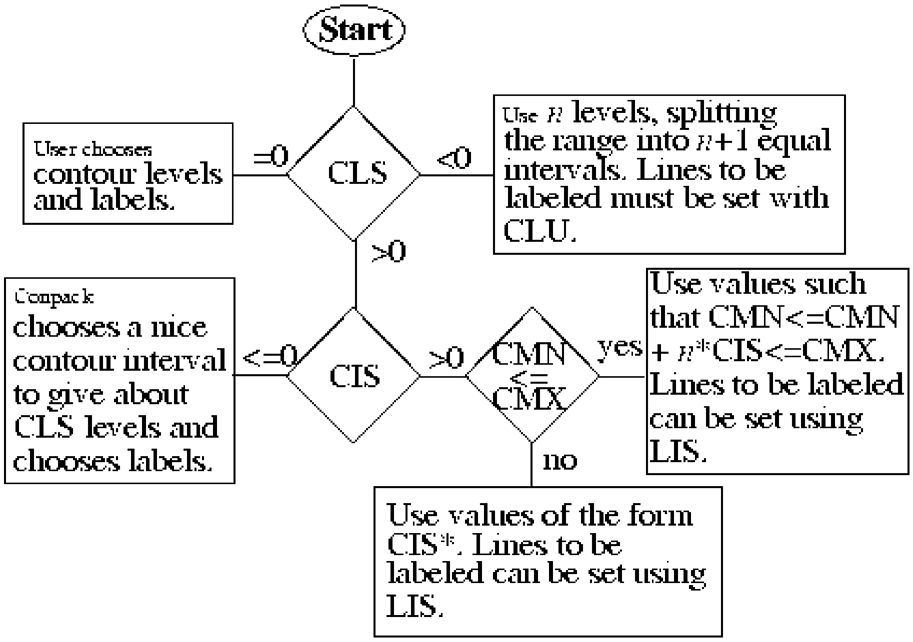
SFU, the scale factor used parameter, may be set as a by-product of choosing the labels. SFU and SFS, the scale factor selector parameter, are described in section "Cp 10. Conpack parameter descriptions." After calling CPPKLB, a user program may examine the generated labels and change them in various ways.

Line 2 of the **ccpklb.f** code segment initializes Conpack, and line 3 forces contour levels to be chosen. Line 4 forces labels to be picked so that we can modify those labels. Lines 6 through 8 retrieve each value of the contour lines, then the label is changed in line 10 to the string created in line 9. Lines 12 and 13 draw a perimeter and contour lines.

Cp 6.3 Selecting lines for labels: Default method

Default contour labels are drawn by the contour line drawing routine CPCLDR. Because of the interactivity of the default contour line drawing options and the default labeling options, these are discussed here again briefly. Default line labels are only chosen when CLS=CIS=0.0.

Default contour labels



Code segment from ccpcit.f

```
      DATA CIT /1., 2., 3., 4., 5., 6., 7., 8., 9., 0./
      DATA LIT /2,  2,  2,  2,  2,  2,  2,  2,  2,  0/
      CALL OPNGKS
      DO 101, I=1, 10
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETR ('CIT - CONTOUR INTERVAL TABLE', CIT(I))
         CALL CPSETI ('LIT - LABEL INTERVAL TABLE', LIT(I))
 101  CONTINUE
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('PAI', n)
CALL CPSETR ('CIT', cit)
CALL CPSETI ('LIT', lit)
CALL CPGETI ('LIU', liu)
```

Arguments

PAI

Integer---The Parameter Array Index must be set to specify the array element to be changed. For example, to change the third array element of LIT to 5, set PAI=3, then set LIT to 5.

CIT(10)

Real array---The Contour Interval Table parameter specifies what constitutes the "nice" value that

Conpack is allowed to use as a contour interval. CIT can use values of the form "*c* times 10 to the power *n*", where *c* is one of the nonzero elements of CIT. (For example, if 2. occurs in CIT, it means that CONPACK can use a contour interval of 2 or 20 or 200 or .2 or .02 and so forth.) NCL is the Number of Contour Lines.

LIT(10)

Integer---The Label Interval Table parameter. For each element of CIT, the associated element of LIT specifies how far apart the labeled levels are. The default contents of CIT and LIT are:

PAI	CIT	LIT
1	1.0	5
2	2.0	5
3	2.5	4
4	4.0	5
5	5.0	5
6	0.0	0
7	0.0	0
8	0.0	0
9	0.0	0
10	0.0	0

LIU

Integer---The Label Interval Used parameter allows you to retrieve *n*, where every *n*th contour level is labeled.

Discussion

Given the default values for CIT, we might obtain, in a given situation, a contour interval of 25 with every 4th level labeled, or a contour interval of 40 with every 5th level labeled. We could not get a contour level of 30, even though that might be desirable for more nearly the approximate number of levels desired.

Lines 1, 2, and 4 through 7 of the **ccpcit.f** code segment set CIT so that "nice" values are the same as they were in the old CONREC routine. LIT is set up to force every other contour level to be labeled no matter what. Lines 9 through 11 draw contours normally.

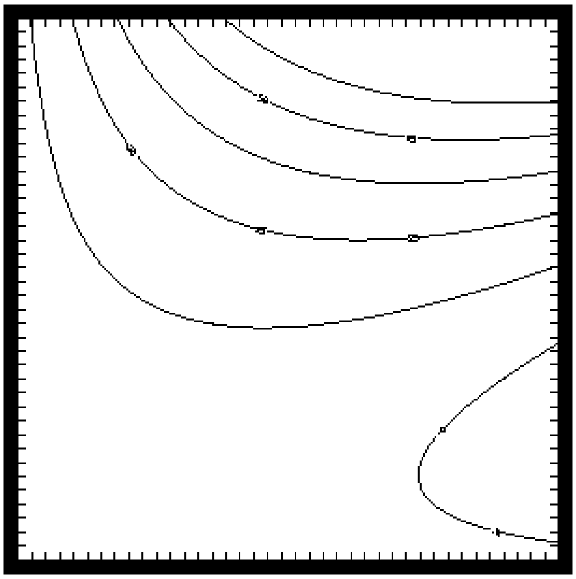
Exercises

1. Using the **ccpcit** example, set Conpack so that no matter what contour interval is chosen, every third contour is labeled.
2. Using the **ccpcit** example, set Conpack so that the default "nice" values are used, but so that each nice value has a different label interval.
3. Using your program from exercise 2, use LIU to determine the label interval.

Cp 6.4 Selecting lines for labels: Labeling every *n*th line

Default contour labels are drawn by the contour line drawing routine CPCLDR. Because of the interactivity of the default contour line drawing options and the default labeling options, these are discussed here again briefly. If the CLS parameter is set so that Conpack chooses contour levels (CLS>0), then it is possible to mandate a contour interval using the CIS parameter. Using the LIS parameter, it is also possible to set the levels that are labeled.

Labeling fixed contour intervals



Code segment from ccpcis.f

```
      CALL CPSETR ('CIS - CONTOUR INTERVAL SPECIFIER', 15.)
      CALL CPSETI ('LIS - LABEL INTERVAL SPECIFIER', 2)
      CALL CPSETR ('CMN - CONTOUR MINIMUM', -5.)
      CALL CPSETR ('CMX - CONTOUR MAXIMUM', 75.)
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETR ('CIS', cis)
      CALL CPSETI ('LIS', lis)
```

Arguments

CIS

Real---The Contour Interval Specifier parameter. When CLS>0, as discussed in module "Cp 4.4 Default contour level selection," a CIS value of:

<=0 Conpack chooses the contour interval using the CIT table to choose a "nice" base value. This is the default.

>0 Conpack uses the value of CIS as the contour interval to be used. In this case, you may

want to specify LIS to be some nonzero value to label every *n*th contour level.
If CLS<=0, then CIS is ignored.

LIS

Integer---The Label Interval Specifier parameter depends on the CLS and CIS parameters. When CLS>0 and CIS>0, then you may want to set LIS to specify the interval between labeled contour levels. LIS=5 by default.

Discussion

For example, if you specify CLS=1, CIS=1/3, and LIS=3, you would get contours at values like 1/3, 2/3, 3/3, 4/3, and so on, and labels on contour lines with values like 1.0, 2.0, 3.0, and so on.

Line 1 of the **ccpcis.f** code segment sets CIS to 15., then line 2 specifies labels on every other line. Lines 3 and 4 set the low and high contour levels to -5. and 75. Contours are set to *n**CIS+CMN (in this case -5., 10., 25., ...).

The CMN and CMX parameters are documented in module "Cp 4.6 Default contour level selection: Fixed contour intervals and labels."

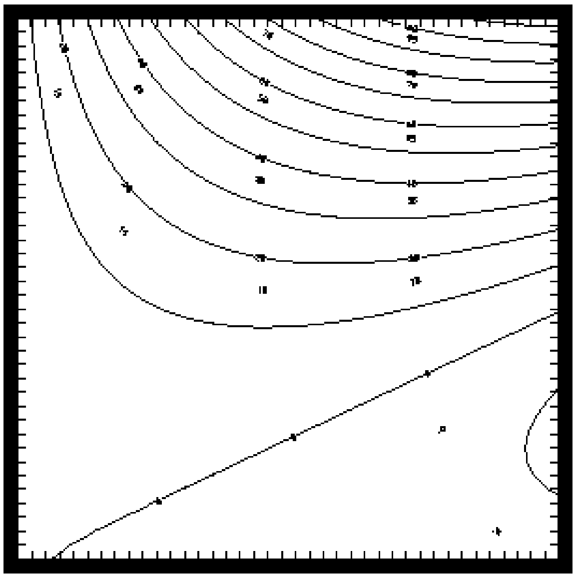
Exercises

1. Using the **ccpcis** example, force Conpack to label every fourth contour level.

Cp 6.5 Selecting lines for labels: Labeling specific lines

In the same way that you can control specific values of individual contour lines, you can control exactly which lines are labeled. The CLU parameter controls this function and does two duties. It first determines whether or not a line will be drawn, then it determines if the line will be labeled. Control over the line-drawing aspect of the CLU parameter is discussed in module "Cp 4.14 Line attributes: Turning contour line drawing off and on."

Labeling individual contour lines



Code segment from ccpclu.f

```
      CALL CPSETI ('CLS - CONTOUR LEVEL SELECTION FLAG', 0)
      CALL CPSETI ('NCL - NUMBER OF CONTOUR LEVELS', 40)
      DO 11, I=1, 40
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETR ('CLV - CONTOUR LEVEL VALUES', 5.0 * REAL (I) - 15.)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', MOD (I, 4))
  11  CONTINUE
      CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('CLU', iclu)
```

Arguments

CLU(NCL)

Integer array---Each element of the Contour Level Use flags parameter indicates how the contour level indicated by the associated element of the CLV array is to be used. NCL is the Number of Contour Levels.

0
No contour line or labels are drawn at this level.

1
A contour line is drawn without labels.

2
Contour labels are drawn but no line is drawn.

3
Both a contour line and labels are drawn.

Note: No labels are drawn on lines specifying special-value, edge-of-grid, or out-of-range lines

(-3<=PAI<=-1).

Discussion

As shown in the **ccpclu** example, the following values are assigned to these parameters:

PAI	CLV(PAI)	CLU(PAI)
1	-10.0	1
2	-5.0	2
3	0.0	3
4	5.0	0
5	10.0	1
6	15.0	2

Also notice that contour level -10.0 is a solid line, contour level -5.0 is labeled but not drawn, contour level 0.0 is drawn and labeled, and contour line 5.0 is neither drawn nor labeled, in accordance with the CLU parameter.

Line 1 of the **ccpclu.f** code segment tells Conpack that we will choose the contour levels. Line 2 sets the number of contour levels to 40, and lines 3 through 7 assign contour level values and set the CLU parameter to cycle through each of its four options. Lines 8 through 10 draw contours normally.

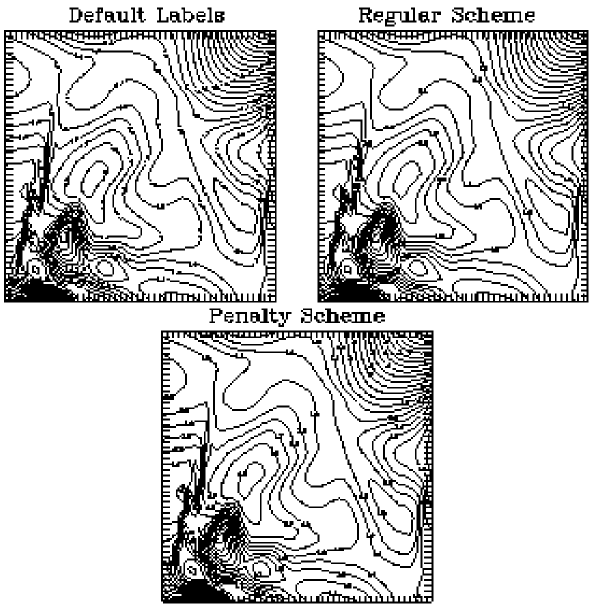
Exercises

1. Modify the **ccpclu** example so that every contour line is drawn and labeled.

Cp 6.6 Three methods of label placement

Conpack provides three different ways to position labels on contour lines. You set the LLP parameter to choose one of these labeling schemes. The simplest method provides little control, and the most sophisticated method offers excellent control.

Three different labeling options



Code showing how to use the LLP parameter appears in label placement modules Cp 6.7 through Cp 6.9.4.

Synopsis

```
CALL CPSETI ('LLP', llp)
```

Arguments

LLP

Integer---The Line Label Positioning flag has several options that specify the labeling method to be used.

0
: No labels are drawn

1 or **-1**
: Labels are positioned along contour lines by setting up a character dash pattern including the label and then drawing the contour with the Dashline utility. Although this is the least costly method, it has the disadvantages that you cannot control label orientation, and some labels may have other contour lines drawn through them. LLP = 1 by default.
Note: This scheme does not work if DPU is set to a value less than or equal to zero.

2
: Labels are positioned along the contour using the moderate cost scheme discussed in module "Cp 6.8 Label placement: Regular scheme."

-2
: Same as above, except that smoothing is suspended during label placement. This saves some CPU time and provides a set of labels that are nearly as good.

3
: Labels are drawn using the more expensive penalty scheme discussed in the label placement

penalty scheme modules Cp 6.9 through Cp 6.9.4.

-3

Same as above, except that smoothing is suspended during label placement. This saves some CPU time and provides a set of labels that are nearly as good.

Discussion

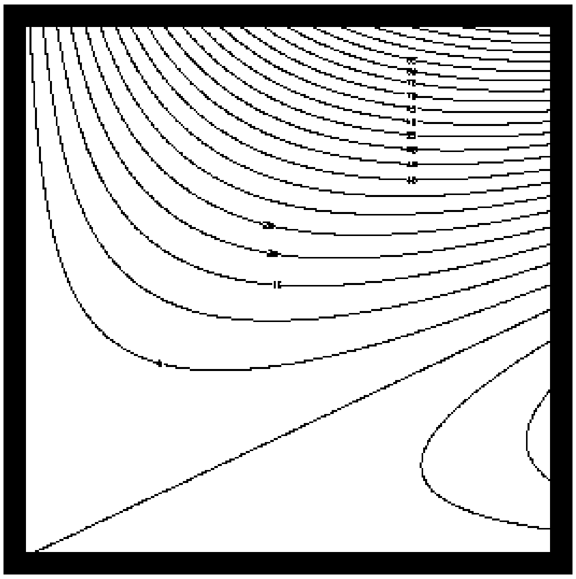
Since LLP=1 by default, every label drawn in sections "Cp 3. Initializing Conpack" and "Cp 4. Contour line basics" uses the inexpensive Dashline option. Review the preceding examples and note that labels are drawn by the CPCLDR subroutine when contours are drawn. When you set LLP to 2, -2, 3, or -3, then it is necessary to call CPLBDR to draw contour labels. The CPLBDR subroutine is discussed in section "Cp 7. General labels."

Also note that when you set LLP=1 or LLP=-1, the labels are actually included in the contour line. The other schemes perform the line labeling separate from the line drawing, and so should be used with the Areas utility to prevent writing over contour lines.

Cp 6.7 Label placement: Default method

By default, contour lines with labels are drawn by Dashline using a dash pattern followed by a label. This is the least expensive contour line labeling option, and can occasionally produce labels in awkward locations, or no labels at all on a given "labeled" line. When labels are in less than ideal locations, you may want to use either the regular scheme or or the penalty scheme method for labeling lines. However, when you get lines where labeling is "on," but no labels appear, you can often force labels to appear by modifying the RWC parameter.

Modifying label placement when using dense gridded data



Code segment from ccprwc.f

```
1        CALL CPRECT (Z, K, M, N, RWRK, LRWK, IWRK, LIWK)
2        CALL CPBACK (Z, RWRK, IWRK)
3        CALL CPPKCL (Z, RWRK, IWRK)
4        CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCL)
5        DO 10, I=1, NCL
6           CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
7           CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
8    10  CONTINUE
9        CALL CPSETI ('RWC - REAL WORKSPACE FOR CONTOURS', 125)
10       CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('RWC', irwc)
```

Arguments

RWC

Integer---The Real Workspace for Contours parameter specifies the amount of real workspace to be allotted to hold coordinates of points defining contour lines and associated quantities. The value of RWC is normally of no particular interest to you, but when labels are placed on a line, the length of the portion of the contour line to be drawn in the default method is limited by the value of RWC. In this case, if labels don't appear on the line, increase RWC. Similarly, when using the penalty scheme, the length of the line over which the penalty function is evaluated is also limited by the value of RWC. In this case, if labels are placed too closely together, increase RWC. RWC=100 by default.

Discussion

In the **ccprwc** example, the data points are extremely dense; this causes the labels to be drawn infrequently. You can increase the number of labels in this case by increasing the value of RWC.

Lines 1 and 2 of the **ccprwc.f** code segment initialize data and draw a background. Lines 3 through 8 force contour line labels to be turned on for each line. Line 9 sets RWC so that some contour labels are drawn and some are not, and line 10 draws the lines and labels.

The amount of real workspace that Conpack is allowed to use for coordinates of points on contour lines is determined by the value of the RWC parameter. By default, each contour line is drawn in pieces of no more than RWC points each. If you have a very dense grid, this means that the contour lines are drawn in rather short chunks (short as measured in NDCs).

Dashline restarts the dash pattern at the beginning of each chunk. What happens is that each chunk is of just the right length so that the dash pattern "$$$$$$$$0" never puts out a single "0," but the dash pattern "$$$$$$$$5" which is on a much longer line, does eventually write out a single "5."

By setting RWC=250, the size of the chunks doubles, and you start getting more labels.

Notice that if you increase RWC too much, you eventually have to make the real workspace larger.

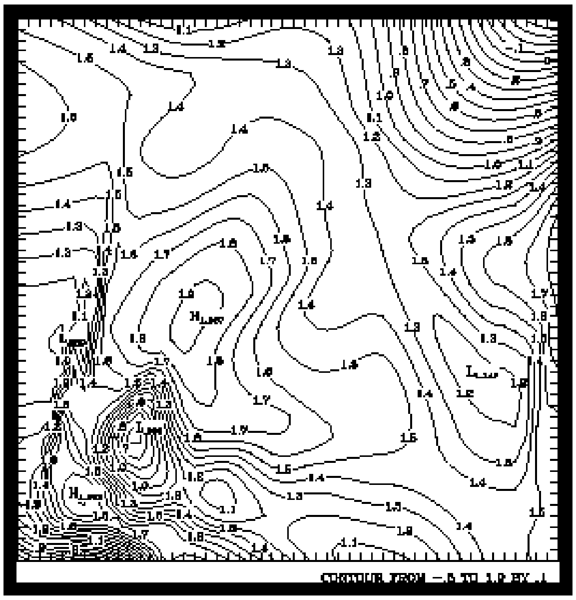
Exercises

1. Using **ccprwc.f**, set RWC=100, then set RWC=250, and compare the plots for differences.

Cp 6.8 Label placement: Regular scheme

The regular scheme for labeling relies on three parameters and a random number generator to position labels. This method is somewhat more cumbersome and time consuming than the default method, but it generally gives better label placement.

Labels at moderate cost



Code segment from ccprc.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 2)
      CALL CPSETR ('RC1 - REGULAR SCHEME CONSTANT 1', .1)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      DO 12, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
  12  CONTINUE
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CALL CPSETR ('RC1', rc1)
CALL CPSETR ('RC2', rc2)
CALL CPSETR ('RC3', rc3)
```

Arguments

RC1

Real---The Regular scheme Constant 1 parameter specifies the desired distance from the beginning of a contour line to the first label on that line when LLP=2 or -2. By default, RC1=.25 in viewport coordinates.

RC2

Real---The Regular scheme Constant 2 parameter specifies the desired nominal distance between labels when LLP=2 or -2. By default, RC2=.25 in viewport coordinates.

RC3

Real---The Regular scheme Constant 3 parameter specifies the desired maximum variation in the distance between labels when LLP=2 or -2. By default, RC3=.05 in viewport coordinates.

Discussion

The *n*th label on each labeled contour line will be at the distance RC1 + RC2 * (*n*-1) + RC3 * Rn, where RC1, RC2, and RC3 are described above, and Rn is a random number such that -1<Rn<1. Labels that would overlap the edge of the viewport or each other are omitted.

You can call CPLBAM to add label boxes for contour line labels to the area map; this prevents contour lines from passing through the labels. This is discussed in module "Cp 5.4 Adding label boxes to the area map."

Line 1 of the **ccprc.f** code segment tells Conpack that we want the moderately expensive labeling scheme. Line 2 sets the distance from the beginning of the contour line to the first label to be small, so that even short contour lines will be labeled. Line 3 initializes Conpack, and lines 4 through 9 force labeling of every line so that you can see the full effect of the labeling option. Line 10 draws the labels with a call to CPLBDR. You must use CPLBDR to draw labels when using the regular scheme for labeling.

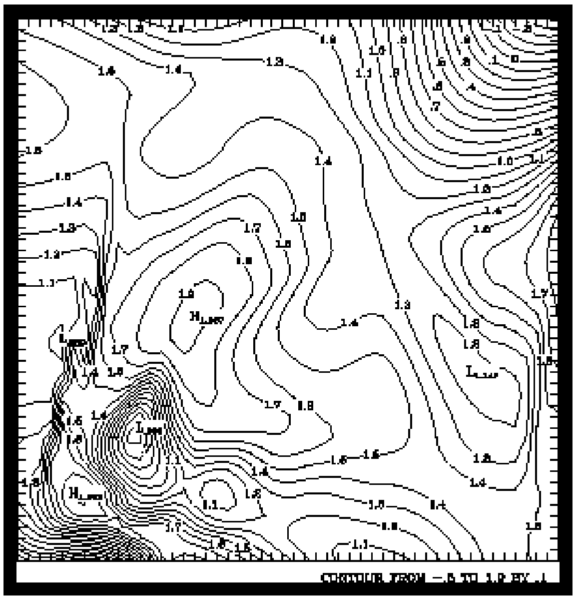
Exercises

1. Run **ccprc.f** using default labels and compare the contour plot here with the results. Note that there are more lines with labels in this example.
2. Using the **ccprc** example, set RC2 first to 0.1 NDCs, then to 0.4, and compare your plots.
3. Using the **ccprc** example, set RC3 first to 0.01 NDCs, then to 1.0, and compare your plots.

Cp 6.9 Label placement: Penalty scheme

The "penalty scheme" label placement option is both the most complex of the three options and the most CPU intensive. However, it generally produces the best results. When the penalty scheme is used, contour lines are traced in order of increasing contour level. At each point on a contour line, several tests are performed; these tests are discussed below. After that, the penalty scheme function is evaluated at the remaining points, and labels are placed where the function returns the smallest value. The penalty scheme function is discussed in this and the next four modules.

Penalty scheme labels



Discussion

When ABS(LLP)=3, a penalty scheme (based on a scheme developed by Starley Thompson and Phil Rasch) is used. The contour lines are traced in order of increasing contour level. At each point along each line, several tests are performed; if the point fails any of the tests, no label is placed there. At the remaining points on the line, a penalty function is evaluated. A label is placed at the point where the penalty function has the smallest value. This process is repeated for each line until it results in the placement of no further labels. The contour line labels are drawn by a call to the label-drawing routine CPLBDR.

You can call CPLBAM to add label boxes for contour line labels to the area map; this prevents contour lines from passing through the labels. This is discussed in module "Cp 5.4 Adding label boxes to the area map."

The penalty scheme is controlled by the ten parameters PC1, PC2, PC3, PC4, PC5, PC6, PW1, PW2, PW3, and PW4. These constants are briefly described here as needed, and complete descriptions appear in the following modules.

The penalty scheme uses seven tests to determine if a point on a contour line is unacceptable as a location for a label. The point P on a contour line is rejected as the center point of a label under these conditions:

1. There are fewer than three points on the contour. In this case, the number of points on a line refer to the number of points used to define a contour.
2. The point P is too close to the center point of any other label on the same or an adjacent contour line, where the meaning of too close is defined by PC6. This test makes sure that labels are not

placed too closely together.

3. The label would extend outside the current viewport or contour rectangle.
4. The label at point P would overlap any previous label. This applies both to labels on other contour levels and to labels on the same contour level.
5. The estimated gradient of the field at point P is too large, where "too large" is defined by the value of PC1 and where PW1<>0. Labels are not drawn if the slope of the field is very steep and contour lines are close together.
6. The estimated number of contour lines crossing a label at point P is too large, where "too large" is defined by the value of PC2 and where PW2<>0.
7. The contour line has too sharp a bend at P, where "too sharp" is defined by PC3 and where PW3<>0. Labels are not placed on tight bends or sharp curves on a contour line.

Exercises

1. Study the **ccppc** example and describe where some of the preceding tests might have failed.

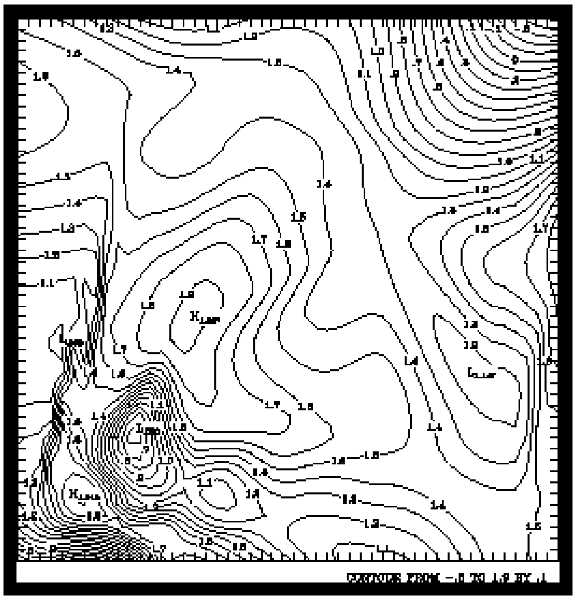
Cp 6.9.1 Label placement: Penalty scheme---Gradient term

After a point passes the seven label-placement tests, Conpack can apply the penalty function to determine where labels may be placed. This penalty function uses the four terms shown in the chart below.

Conpack places labels at points on a contour line where PFUN is minimized. Because of the number of parameters involved, each term of the penalty function is discussed in separate modules. This module describes the gradient term.

```
--------------------------------------------------------------
PFUN=                                            Term
--------------------------------------------------------------
PW1 * GRAD / (GRAV + PC1 * GRSD)                 Gradient
+ PW2 * ENCB / PC2                               Crossing contours
+ PW3 * CDIR / PC3                               Tight curves
+ PW4 * MIN (1 - EXP {-[(D(I)-PC4)/PC5]2})       Optimum distance
--------------------------------------------------------------
```

Effect of slope on label placement



Code segment from ccppc1.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 3)
      CALL CPSETR ('PC1 - PENALTY SCHEME CONSTANT 1', 3.0)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      DO 12, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
  12  CONTINUE
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CPSETR ('PC1', pc1)
CPSETR ('PW1', pw1)
```

Arguments

PC1

Real---The Penalty scheme Constant 1 parameter controls the largest gradient or slope allowed at the position of a label. PC1=1.0 by default, and it can take any positive real value. You should rarely need to set PC1>3.0.

PW1

Real---The Penalty scheme Weight 1 parameter specifies the weight or importance of the gradient term in the penalty function. PW1=2.0 by default, and any positive real value is valid.

Discussion

The gradient term of the penalty scheme is:

```
PW1 * GRAD / (GRAV + PC1 * GRSD)
```

where GRAD is the estimated gradient at the point P, GRAV is the average gradient over the whole field being contoured, and GRSD is the standard deviation of the estimated gradients over the whole field. PC1 specifies, as a multiple of the standard deviation, how far the gradient at point P is allowed to be from the average gradient before it prevents P from being considered as a label position. And PW1 specifies the weight of the gradient term in the penalty function.

This means that labels are not placed on steep slopes going down, nor on steep slopes going up. If you want labels on the steep slopes, you have two choices: you can either make PC1 larger or you can set PW1=0.0. By increasing PC1, you increase the steepness of a slope that can be labeled. By decreasing PW1, you reduce the importance of the slope in the labeling scheme.

To simplify the weighting of terms, the base part of each term is always between 0.0 and 1.0 inclusive. 0.0<=GRAD/(GRAV+PC1*GRSD)<=1.0. If you want to make the gradient term twice as important as the other terms in the penalty function, set PW1=2.0, and PW2, PW3, and PW4=1.0. Or, if you only care about making sure that labels are not placed on the steep parts of fields, set PW1=1.0, and PW2, PW3, and PW4=0.0.

Line 1 of the **ccppc1.f** code segment chooses the penalty scheme labeling option by setting LLP=3. Line 2 sets the gradient constant so that any lines on a steep slope may be labeled. Line 4 initializes Conpack. Lines 5 through 9 turn on line labeling for every line. Notice that line 10 draws labels by calling CPLBDR; penalty scheme labels must be drawn with CPLBDR.

Exercises

1. Using the **ccppc1** example, set PC1=.5, 1.5, and 2.0 in three different runs or frames, and compare the differences in the plots.
2. Using the **ccppc1** example, set PW1=0.0, 1.0, and 5.0 in three different runs or frames, and compare the differences in the plots.

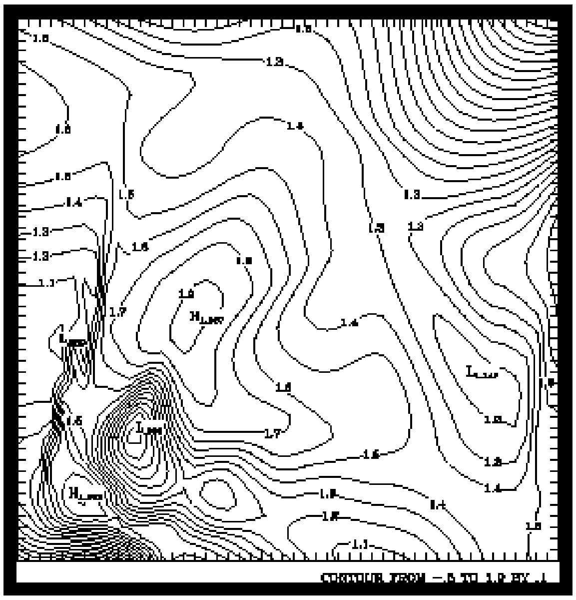
Cp 6.9.2 Label placement: Penalty scheme---Crossing contours

After a point passes the seven label-placement tests, Conpack can apply the penalty function to determine where labels may be placed. This penalty function uses the four terms shown in the chart below.

Conpack places labels at points on a contour line where PFUN is minimized. Because of the number of parameters involved, each term of the penalty function is discussed in separate modules. This module describes the crossing contours term.

```
-------------------------------------------------------------
PFUN=                                       Term
-------------------------------------------------------------
PW1 * GRAD / (GRAV + PC1 * GRSD)            Gradient
+ PW2 * ENCB / PC2                          Crossing contours
+ PW3 * CDIR / PC3                          Tight curves
+ PW4 * MIN (1 - EXP {-[(D(I)-PC4)/PC5]2})  Optimum distance
```

Effect of line spacing on label placement



Code segment from ccppc2.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 3)
      CALL CPSETR ('PC2 - PENALTY SCHEME CONSTANT 2', 1.0)
      CALL CPSETR ('PW2 - PENALTY SCHEME WEIGHT 2', 1.0)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      DO 12, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
 12   CONTINUE
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CPSETR ('PC2', pc2)
CPSETR ('PW2', pw2)
```

Arguments

PC2

Real---The Penalty scheme Constant 2 parameter specifies the maximum estimated number of contour bands allowed to cross a label. PC2=5.0 by default.

PW2

Real---The Penalty scheme Weight 2 parameter specifies the weight (or importance) of the "crossing contours" term in the penalty function. PW2=0.0 by default.

Discussion

The "crossing contours" term of the penalty function becomes larger as ENCB increases. ENCB is the estimated number of contour lines crossing a label at point P. PC2 specifies the largest number of lines that a contour label is allowed to cross. By increasing the value of the Conpack parameter RWG, you can increase the accuracy of ENCB.

In many senses, the gradient and "crossing contours" terms of the penalty function represent the same thing. Any locations where contour lines are close enough together for labels to cross them are also regions having a relatively steep slope.

To simplify the weighting of terms, the base part of each term is always between 0.0 and 1.0 inclusive. In other words, 0.0<=ENCB / PC2<=1.0. If you want to make the crossing contours term twice as important as the other terms in the penalty function, set PW2=2.0, and PW1, PW3, and PW4=1.0. Or, if you only care about making sure labels do not cross contour lines, set PW2=1.0, and PW1, PW3, and PW4=0.0.

By default, PW2=0.0; this allows labels to cross any number of contour lines. If you want to change it so that labels are unlikely to cross contour lines, set PW2>=1.0, and PC2=1.0, as shown in the **ccppc2.f** code segment.

Line 1 of the **ccppc2.f** code segment chooses the penalty scheme labeling option by setting LLP=3. Line 2 sets the crossing contours constant so that labels do not cross contour lines. Line 3 changes the weighting function so that this option is observed. Line 4 initializes Conpack. Lines 5 through 10 turn on line labeling for every line. Notice that line 11 draws labels by calling CPLBDR; penalty scheme labels must be drawn with CPLBDR.

Exercises

1. Using the **ccppc2** example, set PC2=2.0, 3.0, and 4.0 in three different runs or frames, and compare the differences in the plots.
2. Using the **ccppc2** example, set PW2=0.5, 1.5, and 5.0 in three different runs or frames, and compare the differences in the plots.

Cp 6.9.3 Label placement: Penalty scheme---Tight curves

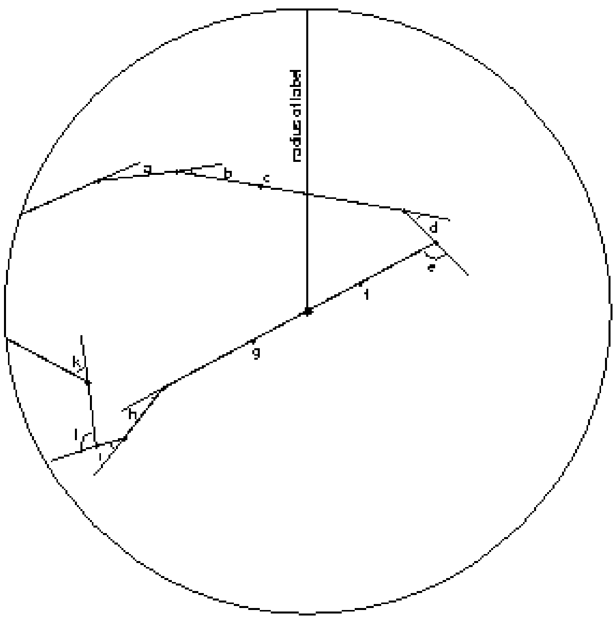
After a point passes the seven label-placement tests, Conpack can apply the penalty function to determine where labels may be placed. This penalty function uses the four terms shown in the chart below.

Conpack places labels at points on a contour line where PFUN is minimized. Because of the number of parameters involved, each term of the penalty function is discussed in separate modules. This module describes the "tight curves" term.

```
-----------------------------------------------------------
PFUN=                                         Term
```

```
------------------------------------------------------------
PW1 * GRAD / (GRAV + PC1 * GRSD)            Gradient
+ PW2 * ENCB / PC2                          Crossing contours
+ PW3 * CDIR / PC3                          Tight curves
+ PW4 * MIN (1 - EXP {-[(D(I)-PC4)/PC5]2})  Optimum distance
------------------------------------------------------------
```

What constitutes a tight curve



Code segment from ccppc3.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 3)
      CALL CPSETR ('PC3 - PENALTY SCHEME CONSTANT 3', 10.0)
      CALL CPSETR ('PW3 - PENALTY SCHEME WEIGHT 3', 2.0)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      DO 12, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
  12  CONTINUE
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CALL CPSETR ('PC3', pc3)
CALL CPSETR ('PW3', pw3)
```

Arguments

PC3

Real---The Penalty scheme Constant 3 parameter specifies, in degrees, the maximum cumulative change in direction to be allowed along that portion of the contour line covered by a circle centered on a label and having a radius equal to half the width of the label. PC3=60.0 by default.

PW3

Real---The Penalty scheme Weight 3 parameter specifies the weight (importance) of the "tight curves" term in the penalty function. PW3=1.0 by default.

Discussion

The "tight curves" term of the penalty scheme is:

```
PW3 * CDIR / PC3
```

where CDIR is the cumulative change in the direction of the contour line in a circular region centered at the point P with a radius equal to half of the largest dimension of the label. PC3 specifies the largest such cumulative change allowed in degrees. The illustration shows that CDIR is the sum of the angles specifying the change in direction of the contour inside the label circle (CDIR=$a+b+c+d+e+f+g+h+i$, where c, f, and g are zero in the illustration).

By default, PC3=60.0, so a contour line cannot bend more than 60 degrees within the diameter of the label if a label is to be placed at that point. The idea is that labels should not cover up interesting details of contour lines.

To simplify the weighting of terms, the base part of each term is always between 0.0 and 1.0 inclusive. In other words, 0.0 <= CDIR / PC3 <= 1.0. If you want to make the tight curves term twice as important as the other terms in the penalty function, set PW3=2.0, and PW1, PW2, and PW4=1.0. Or, if you only care about making sure that labels are not placed on tight bends, set PW3=1.0, and PW1, PW2, and PW4=0.0.

The **ccppc3** example follows much the same form as **ccppc1** and **ccppc2**, and it illustrates the use of PC3 and PW3. Study these examples to determine the differences.

Exercises

1. Using the **ccppc3** example, set PC3=30.0, 90.0, and 180.0 in three different runs or frames, and compare the differences in the plots.
2. Using the **ccppc3** example, set PW3=0.0, 1.5, and 5.0 in three different runs or frames, and compare the differences in the plots.

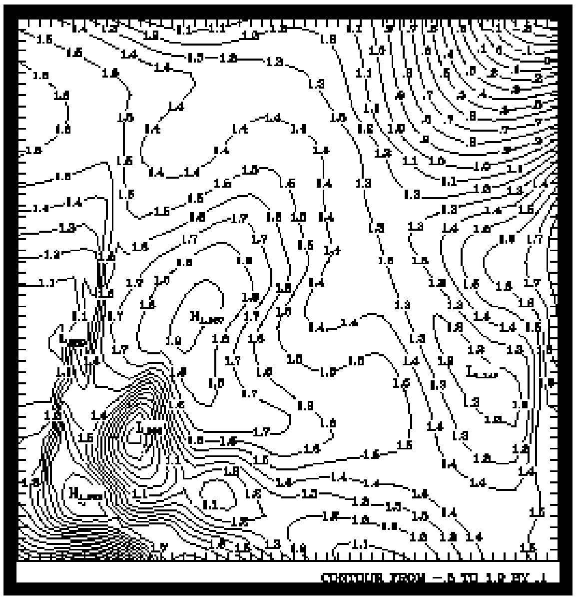
Cp 6.9.4 Label placement: Penalty scheme---Label spacing

After a point passes the seven label-placement tests, Conpack can apply the penalty function shown below to determine where labels may be placed.

Conpack places labels at points on a contour line where PFUN is minimized. This module describes the optimum distance term.

PFUN=	Term
PW1 * GRAD / (GRAV + PC1 * GRSD)	Gradient
+ PW2 * ENCB / PC2	Crossing contours
+ PW3 * CDIR / PC3	Tight curves
+ PW4 * MIN (1 - EXP {-[(D(I)-PC4)/PC5]2})	Optimum distance

Effect of optimum distance on label placement



Code segment from ccppc4.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 3)
      CALL CPSETR ('PC4 - PENALTY SCHEME CONSTANT 4', 0.1)
      CALL CPSETR ('PC5 - PENALTY SCHEME CONSTANT 5', 0.2)
      CALL CPSETR ('PC6 - PENALTY SCHEME CONSTANT 6', 0.05)
      CALL CPSETR ('PW4 - PENALTY SCHEME WEIGHT 4', 5.0)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      DO 12, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
 12   CONTINUE
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETR ('PC4', pc4)
      CALL CPSETR ('PC5', pc5)
      CALL CPSETR ('PC6', pc6)
      CALL CPSETR ('PW4', pw4)
```

Arguments

PC4

Real---The Penalty scheme Constant 4 parameter specifies the optimal label spacing as a fraction of the viewport width. PC4=0.05 by default.

PC5

Real---The Penalty scheme Constant 5 parameter is a "folding distance;" it helps place labels at an optimal distance from each other. PC5=0.15 by default.

PC6

Real---The Penalty scheme Constant 6 parameter specifies the minimum distance allowed between any two labels on the same contour line, as a fraction of the viewport width. PC6=0.3 by default.

PW4

Real---The Penalty scheme Weight 4 parameter specifies the weight (importance) of the optimum distance terms (PC4, PC5, and PC6) in the penalty function. PW4=1.0 by default.
To simplify weighting, the base part of each term is always between 0.0 and 1.0 inclusive. 0.0 <= MIN(1-EXP{-[(D(I)-PC4)/PC5]*2*} <= 1.0.

Discussion

The optimum distance term of the penalty function becomes small when the distance of the point P from the center of some labels previously placed on another contour line matches the optimum value specified by PC4. D(I) represents the distance to the Ith such label center. The minimum is taken over all values of I. If point P is exactly PC4 units away from some previous label, then MIN (1EXP(...)) has the value 0; if not, it may get as large as 1.0. PC5, also expressed as a fraction of the viewport width, limits how far labels can be placed from the optimal distance, and PC6 limits how closely together labels can be placed.

Let the penalty term be written as:

$$\min_{(\text{all } i)} \left\{ 1 - e^{-\left(\frac{d_i - PC4}{PC5}\right)^2} \right\}$$

For a given label L*i*, the penalty would be:

$$p_i = 1 - e^{-\left(\frac{d_i - PC4}{PC5}\right)^2}$$

where d*i* is the distance from the point P (under consideration) to the center of L*i*. "d*i*-PC4" is the amount by which that distance differs from the desired optimum value. The following table may help explain what PC5 does:

d*i* - PC4	penalty	d*i* - PC4	penalty
0.	0.		
.25*PC5	.06059	1.25*PC5	.79039

```
.50*PC5       .22120   1.50*PC5      .89460
.75*PC5       .43022   1.75*PC5      .95323
1.00*PC5      .63212   2.00*PC5      .98168
--------------------------------------------
```

One problem is that, if you make PC5 too small, it may happen that no point P you consider gives you a penalty less than .99, so you will be using the best of a bad lot.

The **ccppc4** example follows much the same form as **ccppc1**, **ccppc2**, and **ccppc3**; it illustrates the use of PC4, PC5, PC6, and PW4. Study these examples to determine the differences.

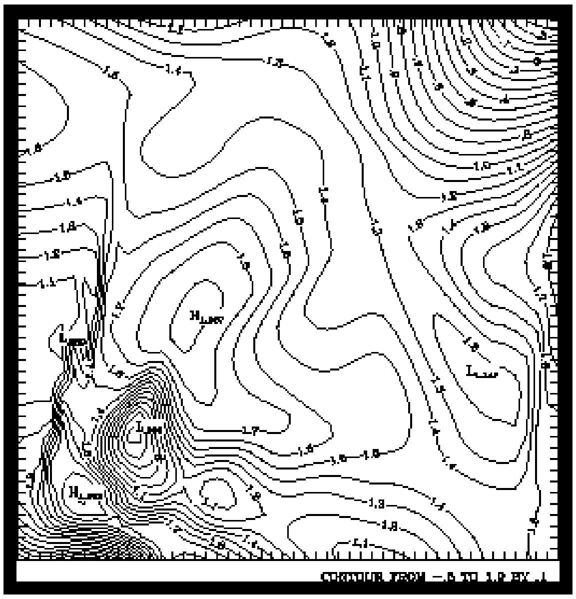
Exercises

1. Using the **ccppc4** example, set the optimum distance parameters so that most lines have only one label.

Cp 6.10 Label attributes: Angle and orientation

If you are using the regular scheme or the penalty scheme for labeling your lines, you can change the angle or orientation of the labels on your contours. with the LLA and LLO parameters.

Labels parallel to contours



Code segment from ccpllo.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 3)
      CALL CPSETI ('LLO - LINE LABLE ORIENTATION', 1)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      DO 12, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
```

```
8          CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
9   12  CONTINUE
10      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETR ('LLA', rlla)
      CALL CPSETI ('LLO', llo)
```

Arguments

LLA

Real---The Line Label Angle parameter is only used when you are using the regular or penalty scheme of line labeling (LLP=2, -2, 3, or -3). LLP is the angle in degrees at which contour line labels are to be written. LLA=0.0 by default.

LLO

Integer---The Line Label Orientation parameter is only used when you are using the regular or penalty scheme of line labeling (LLP=2, -2, 3, or -3).

0 All labels are written at the angle specified by LLA. This is the default.

<>0 Each label is written parallel to the tangent to the contour line at the center of the label. In other words, labels are written parallel to the contour line.

By default, all labels are written horizontally, no matter what direction contour lines are running at the point of the label.

Discussion

Line 1 of the **ccpllo.f** code segment sets the labeling scheme to the penalty scheme, thereby enabling LLA and LLO. Line 2 sets LLO=1 so that labels are drawn parallel to the contour line. Conpack is then initialized, contour lines are chosen, and lines 5 through 9 turn on line labeling for every line. Line 10 draws labels using the call to CPLBDR since we are not using the default labels.

Compare the results here with the plot in module "Cp 6.9 Label placement: Penalty scheme."

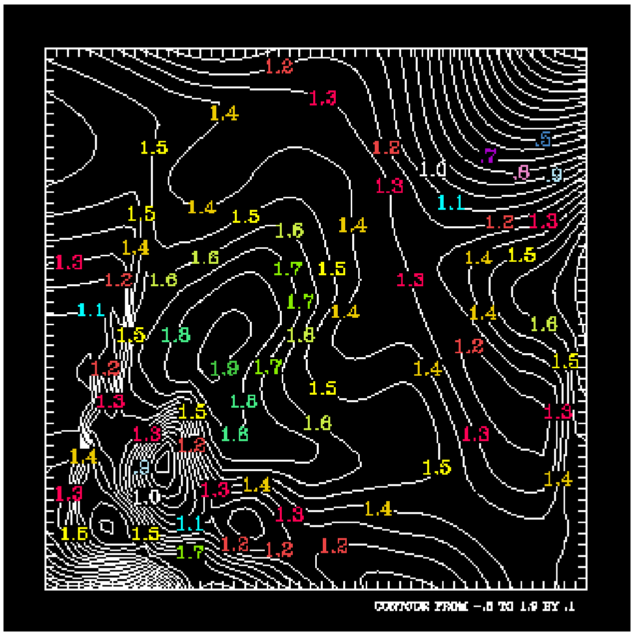
Exercises

1. Using the **ccpllo** example, set all line labels so that they are at a 30-degree angle from horizontal.
2. Using the **ccprc** example, set all line labels so that they run parallel to the contours.

Cp 6.11 Label attributes: Color

Contour line label color is controlled by the LLC parameter and the GKS color setting routine GSCR. LLC is a parameter array, allowing you to choose different label colors for each contour level.

Coloring contour line labels



Code segment from ccpllc.f

```
      CALL COLOR
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 2)
      CALL CPSETR ('LLS - LINE LABEL SIZE', .02)
      CALL CPSETC ('HLT - HIGH/LOW LABEL TEXT STRINGS', ' '' ')
      DO 11, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
         CALL CPSETI ('LLC - LINE LABEL COLOR INDEX', MOD (I, 16) + 1)
  11  CONTINUE
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL CPCLDM (ZREG, RWRK, IWRK, MAP, CPDRPL)
      CALL CPLBDR (ZREG, RWRK, IWRK)

      SUBROUTINE COLOR
      CALL GSCR (1, 0, 0., 0., 0.)
      CALL GSCR (1, 1, 1.0, 1.0, 1.0)
      CALL GSCR (1, 2, 0.0, 0.9, 1.0)
      CALL GSCR (1, 3, 0.9, 0.25, 0.0)
      CALL GSCR (1, 4, 1.0, 0.0, 0.2)
      CALL GSCR (1, 5, 1.0, 0.65, 0.0)
      CALL GSCR (1, 6, 1.0, 1.0, 0.0)
```

Synopsis

```
    CALL CPSETI ('LLC', llc)
```

Arguments

LLC(NCL)

Integer array---The Line Label Color parameter array is an array of color indices, each element of which specifies the color for labels on contour lines at the level specified by the associated element of the parameter array CLV. NCL is the Number of Contour Levels. If the value of an element of LLC is:

> **>=0** The color index for labels on the contour line specified by the associated element of CLV.
> **<0** The current text color is to be used.

Discussion

Accessing each element of the LLC array is done by setting the PAI parameter, as shown in line 9 of the **ccpllc.f** code segment. For a complete description of the PAI parameter and how it works, see module "Cp 1.6 Conpack parameters: What they do and how to use them."

Line 1 of the **ccpllc.f** code segment sets up a color table using the subroutine named "color," partially defined in lines 17 through 24. A complete discussion of setting up a color table appears in Chapter 7 "Color tables and color mapping systems" of the *NCAR Graphics Fundamentals* guide. Line 2 of the code segment initializes Conpack, and line 3 picks contour levels so that we can force labeling of every line in lines 8 through 12.

Line 5 selects the regular scheme for labeling so that the color attribute is used, and line 11 chooses colors by cycling through the color table. Line 6 enlarges the label size to make the labels more visible over the contour lines, and line 7 turns off high and low labels to emphasize the desired labels. High and low label options are discussed in module "Cp 6.1 Drawing labels." Note that contour lines are masked in line 14.

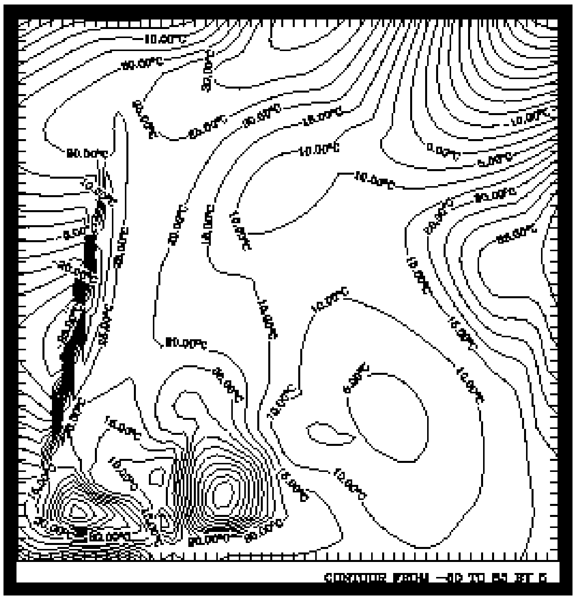
Exercises

1. Using the **ccpllc** example, make all the lines turquoise.
2. Using the **ccpllc** example, make all the lines above zero green, the zero line purple, and the negative lines blue.

Cp 6.12 Label attributes: Label text options

There are two label text options that can be used with the regular or penalty labeling scheme (LLP=2, -2, 3, or -3). You can control the size of the label, and you can control the text of the label.

Changing label text and size



Code segment from ccpllt.f

```
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPGETI ('NCL - NUMBER OF CONTOUR LEVELS', NCONS)
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 2)
      CALL CPSETC ('HLT - HIGH/LOW LABEL TEXT STRINGS', ' '' ')
      CALL CPSETI ('LLO - LINE LABEL ORIENTATION', 1)
      DO 11, I=1, NCONS
         CALL CPSETI ('PAI - PARAMETER ARRAY INDEX', I)
         CALL CPSETI ('CLU - CONTOUR LEVEL USE FLAGS', 3)
         CALL CPGETR ('CLV - CONTOUR LEVEL VALUES', VALUE)
         WRITE (STRING, 5) VALUE
         CALL CPSETC ('LLT - LINE LABEL TEXT STRING', STRING)
 11   CONTINUE
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL CPCLDM (ZREG, RWRK, IWRK, MAP, CPDRPL)
      CALL CPLBDR (ZREG, RWRK, IWRK)
  5   FORMAT (F6.2, ':SRIL:O:N:C', F6.2)
```

Synopsis

```
CALL CPSETC ('LLT', cllt)
```

Arguments

LLT(NCL)

> *Character*20 array*---The Line Label Text string parameter allows you to specify a different label of up to 20 characters for each contour line. The Ith element of LLT specifies the label for the contour specified by the Ith value of CLV, regardless of the order of the values in the CLV array. Since the label is drawn using the Plotchar utility, you may use colon-enclosed function codes to create subscripts, use Greek characters, or use other Plotchar options. By default, Conpack chooses the value of the contour level as its label.

LLS

Real---The Line Label Size parameter specifies the size (width) of a character in a contour line label as a fraction of the width of the viewport. This nominal size is multiplied by CWM.

CWM

Real---All label sizes are multiplied by the Character Width Multiplier parameter. This makes it easy to scale the sizes of each type of label up and down with respect to the others, or to increase and decrease all label sizes at once. CWM=1.0 by default.

Discussion

LLS is defined as a fraction of the viewport width: 0.0<LLS<1.0. If you set LLS=1.0, and CWM=1.0, you would find that each character is as wide as the viewport.

The LLT parameter allows you to give each contour line a label of your choice. In the **ccpllt** example, each contour level value is followed by "*o*C". Long labels are not terribly useful in publication-quality plots because they don't follow the curves of the contours. By default, Conpack prints the value of the level as the label, with numbers represented in the form discussed in the numeric control modules Cp 7.3 through Cp 7.5.

Line 1 of the **ccpllt.f** code segment initializes Conpack for regularly spaced data, and line 2 forces the choosing of contour levels so that we can turn on labeling for each line in line 11. Line 4 chooses the regular labeling scheme so that the text options will be used. Line 5 turns off high and low labels to show the contour labels more clearly. This option is described in module "Cp 6.1 Drawing labels."

Line 6 forces labels to be placed parallel to the contour line being labeled. Line 10 retrieves the value of the contour level so that we can make a meaningful label for each line. Line 11 does an internal write to the variable string so that each line is labeled with its index in the CLV array. Line 12 sets up the label array so that each line is given the appropriate label. In lines 14 through 16, a background is drawn using CPBACK, contours are drawn by CPCLDM, and labels are drawn with a call to CPLBDR, as this tutorial has now demonstrated many times. Line 17 provides a format for the contour line label text and places the "degrees Celcius" symbol after the contour line value.

Exercises

1. Change the notation in the **ccpllt** example so that the value of each contour level is followed by "°F".

Cp 7. General labels

Because of the complexity of the Conpack labeling scheme, labeling has been divided into two sections, by function:

- Contour line labels (selecting lines to be labeled, label placement, label attributes)---Section "Cp 6. Contour line labels."
- General labels (plot annotations, numeric label control, titling)---Section "Cp 7. General labels."

Note: Many contour line label options are covered in section "Cp 7. General labels" because they share common parameters or functions with general labels.

Conpack functional outline

```
-----------------------------------------------------------
    1.  Open GKS
    2.  Set window and viewport
 *  3.  Put in titles and other labels
 *  4.  Set up general parameters
    5.  Set up area maps
    6.  Initialize Conpack
    7.  Force Conpack to choose contour levels if necessary
    8.  Modify Conpack-chosen parameters
 *  9.  Draw labels
    10.  Draw background
    11.  Draw contour lines
    12.  Call FRAME
    13.  Close GKS

-----------------------------------------------------------
```

* Steps discussed in this section.

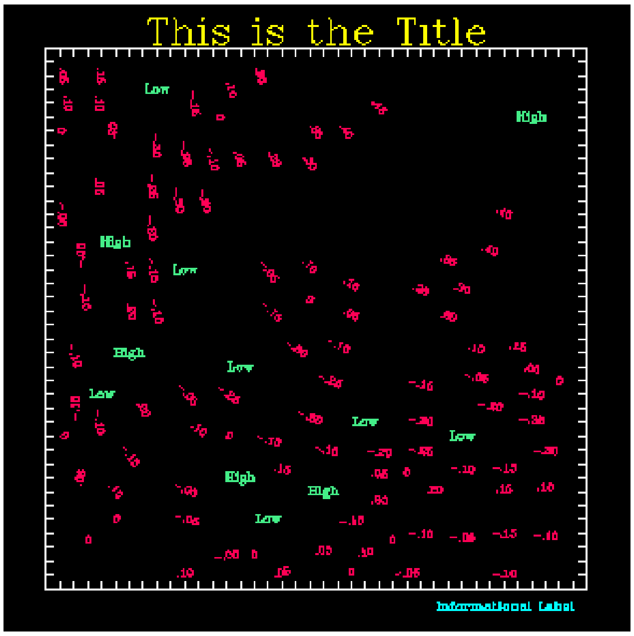
Cp 7.1 Overview of labeling in Conpack

There are many types of labels used in contour plots:

- Plot titles
- Constant field labels
- Information labels for the entire plot
- High/low labels
- Contour line labels

Each kind of label has specific options that are discussed in the appropriate module. Section "Cp 7. General labels" covers ways to produce titles, constant field labels, information labels, high/low labels, and how to control the numeric format of all other labels.

Types of labels in Conpack



Discussion

The plot titles and annotations shown in yellow in the example are technically not produced as Conpack functions. You must use the Plotchar utility to draw all user-defined titles and annotations after the contour plot. However, since most people want to title their plots, this procedure is covered in the following modules.

Controlling contour line labels (shown in red in the example) is a multifaceted operation that depends on which lines you want to be labeled, how much you care about where the labels go on any given contour line, and whether or not you want boxes around your labels (so that contour lines don't run through them or so that you can make them show up in color-filled or pattern-filled regions).

Controlling which contour lines you want to label and where the labels go on any given contour line, as well as other aspects like label size, text, and color are covered in section "Cp 6. Contour line labels." If you are interested in boxing your labels, please see modules "Cp 5.4 Adding label boxes to the area map," "Cp 5.5 Masking areas: Label boxes," and "Cp 5.6 Masking areas: Box masking routines."

High and low labels (shown in green in the example) have control mechanisms very similar to information labels, and so are discussed in the following modules.

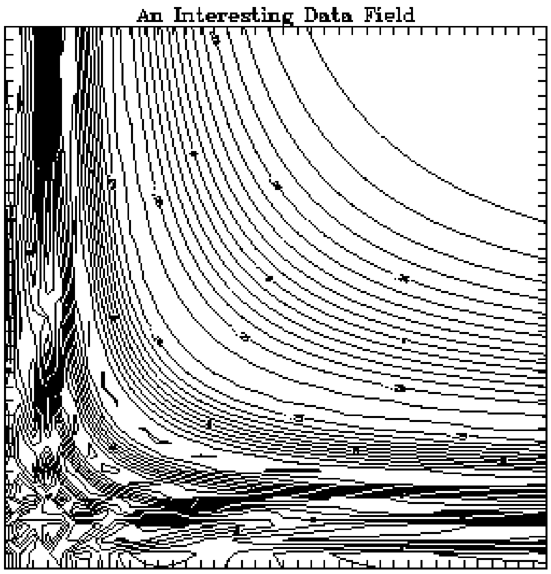
The information label (shown in blue in the example) has control mechanisms very similar to the high and low labels and so is discussed in the following modules.

The constant field label is the least used of the Conpack labels. This label is only plotted when your data field is essentially constant. Constant field labels can be modified with options similar to the high, low, and information labels.

Cp 7.2 Titles

Titles are not produced with any Conpack subroutines. They are placed using the SPPS utilities GETSET and SET, and they are drawn with the Plotchar utility.

Conpack titles



Code segment from ccptitle.f

```
      CALL GSCLIP (0)
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
      CALL GETSET (VPL, VPR, VPB, VPT, WL, WR, WB, WT, LL)
      SIZE = AMIN1 (.5 * (1.0 - VPT), .017)
      Y = AMIN1 ((1.0 + VPT) / 2.0 + .017, .017 + VPT)
      CALL SET (0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1)
      CALL PLCHHQ (.5, Y, 'An Interesting Data Field', SIZE, 0., 0.)
```

Synopsis

```
CALL PLCHHQ (XPOS, YPOS, CHRS, SIZE, ANGD, CNTR)
```

Arguments

XPOS, YPOS

> *Real, Input*---The X and Y coordinate positions for the characters to be plotted. These are world coordinates. See the description of CNTR below.

CHRS

> *Character expression, Input*---A string specifying the characters to be plotted.
> **Caution:** Do not use the characters ^ (caret), _ (underscore), or ‘ (single left quotation mark).

Although these are ASCII characters, they do not exist in the PLCHHQ complex or duplex character sets. If you use one of these, the output character will be the five-pointed star that indicates an unrecognized character.

SIZE

Real, Input---Defines a multiplier for character size. Set the value of SIZE to a real number in one of the following ranges:

<=0. The absolute value specifies the size as a multiple of the digitized size.

0.<SIZE<1.

Specifies the desired width of a character as a fraction of the distance across the plotter frame (in NDCs).

ANGD

Real, Input---The angle at which the character string is to be plotted, in degrees counterclockwise from the positive X axis.

CNTR

Real, Input---The centering option. Set the value of CNTR to a real number as follows:

-1. The point (XPOS, YPOS) is the center of the top edge or the left edge of the first character (depending on whether that character is plotted "down" or "across").

1. The point (XPOS, YPOS) is the center of the bottom edge or the right edge of the last character (depending on whether that character is plotted "down" or "across").

0. The point (XPOS, YPOS) is the midpoint of the line joining the center of the top edge or the left edge of the first character (depending on whether that character is plotted "down" or "across") to the center of the bottom edge or the right edge of the last character (depending on whether that character is plotted "down" or "across").

s Any real number can be used as *s*. The point (XPOS, YPOS) is a point obtained by linear interpolation along the line joining the point associated with the value -1. and the point associated with the value +1.

Discussion

More detailed information is available in Chapter 11 "Drawing text and symbols" of *NCAR Graphics Fundamentals*.

Line 1 of the **ccptitle.f** code segment turns off clipping so that the information label appears. Lines 2, 3, and 4. initialize Conpack, draw a background, and draw contours as usual. Line 5 uses the SPPS routine GETSET to find out where the contour rectangle has been placed. The first four arguments to GETSET return the left, right, bottom, and top sides of the contour rectangle respectively. Since we want to place the title a little above the plot in this case, we find out the distance between the top of the plot and the top of the viewport, then make our characters roughly half that distance tall (a good size).

Line 7 computes the Y coordinate for the center of the text so that line 9 can call PLCHHQ to draw the title. Notice that line 8 calls SET. This resets the viewport to the entire plotter frame. The SET call also resets user coordinates to Normalized Device Coordinates so that the characters are drawn in NDC units rather than in Conpack's user coordinates.

Exercises

1. Move the title of the **ccptitle** example so it appears below the plot, and change it to say "This is a title."
2. Draw the title of the **ccptitle** example sideways on the left edge of the plot.

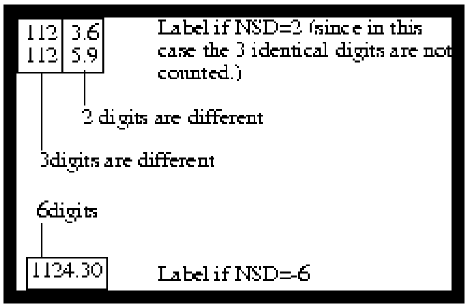
Cp 7.3 Numeric control: Significant digits

Two parameters control the number of significant digits in numeric labels: NSD and NLS. These parameters control high, low, information, and constant field label output simultaneously.

Controlling significant digits

NSD, Number of Significant Digits

Assume that the contour field varies between 1123.6 and 1125.9



NLS, Numeric Leftmost Significant digit

Assume the use of three significant digits
If NLS=0: .500 1.00 1.50 . . . 9.50 10.5
If NLS<>0: .5 1.0 1.5 . . . 9.5 10.5

Synopsis

```
CALL CPSETI ('NSD', nsd)
CALL CPSETI ('NLS', nls)
```

Arguments

NSD

Integer---The Number of Significant Digits parameter specifies the maximum number of significant digits used in numeric labels. By default, NSD=4.

***n*<0**
Specifies that ABS(*n*) significant digits should be used.
***n*>=0**

Specifies that *m*+*n* digits should be used, where *m* is the number of digits that are the same for all values in the contour field. For example, if the minimum value is 1123.6 and the maximum value is 1125.9, then the value of *m* is 3.

NLS

Integer---The Numeric Leftmost Significant digit flag controls what is considered to be the the first significant digit.

0 The leftmost nonzero digit of a number is considered its first significant digit.

<>**0** The digit in the same digit position as the leftmost nonzero digit of the largest number (in absolute value) in the data field being contoured is considered to be the leftmost significant digit. This option tends to make labels more consistent. In a Fortran FORMAT statement, the form would be F*n.m*, where *n* is the number of digits in either the largest or smallest contour field value. This is the default.

Discussion

As shown in the "NSD, Number of Significant Digits" example in the illustration, the parameter NSD can mean different things depending on how it is set. If you set it to a negative number, it sets the exact number of significant digits. However, if you set it to a positive number, Conpack finds the range of the contoured field, then finds the number of digits that are the same in the minimum and maximum values, and does not count those digits as part of the number of significant digits. Where we have three similar digits and two dissimilar digits in the illustration, we set NSD=2 to get five digits in our label. The number of significant digits in each type of label varies, and this affects exactly how the label is written.

The "NLS, Numeric Leftmost Significant digit" example in the illustration shows how the NLS flag works. By default, Conpack takes the absolute value of the largest and smallest values in the data field. Using the largest value, Conpack determines the number of significant digits for that value from NLS. Then Conpack makes all labels have the same number of digits following the decimal point as that value. The illustration shows that when NLS<>0, each label has one digit following the decimal point. However, when NLS=0, the leftmost nonzero digit of each label is the first significant digit. Therefore, each label has a different number of digits after the decimal point.

Exercises

1. Using the **cidsfft** example, add a call to CPLBDR to draw information and high/low labels. Use both positive and negative values of NSD to get five significant digits.
2. Using your solution to exercise 1, turn off NLS and see if there is any difference. Which option overrides the other?

   Hint: if you would like to see an example using these two parameters, run the **ccpnsd** example.

Cp 7.4 Numeric control: Formatting

Two parameters control leading and trailing zeros, trailing decimals, and so forth in numeric labels: NLZ and NOF. These parameters control high, low, information, constant field, and contour line label output simultaneously.

Controlling numeric format

Table 1: NLZ, Numeric Leading Zero flag

.395	Label with NLZ=0
0.395	Label with NLZ<>0

Table 2: NOF, Numeric Omission Flag

This	**Not this**		
46.2000		NOF=0	(nothing omitted)
46.2	46.2000	NOF=1	(trailing zeros omitted)
13	13.	NOF=2	(trailing decimal omitted)
13	13.00	NOF=3	(trailing zeros and decimal omitted)
10*13*	1 x 10*13*	NOF=4	("1 x" omitted)

Synopsis

```
      CALL CPSETI ('NLZ', nlz)
      CALL CPSETI ('NOF', nof)
```

Arguments

NLZ

Integer---The Numeric Leading Zero flag controls whether or not a leading zero is placed before a decimal point.
0 No zero is added to numeric labels. This is the default.
<>0 A zero is placed before decimal points.

NOF

Integer---The Numeric Omission Flag specifies which parts of a numeric label may be omitted.
0 No part may be omitted.
1 Trailing zeros (as in .2500) may be omitted.
2 Trailing decimals (as in 1.) may be omitted.
4 Leading "1 x" notation may be omitted (as in 1 x 10*13*)
To get combined effects, add two or more of the above values together, and set NOF equal to the sum. By default, NOF=6.

Discussion

In the "NLZ, Numeric Leading Zero flag" example of the illustration, note that setting NLZ<>0 puts a zero before a decimal number. The Numeric Omission Flag NOF controls three other numerical aspects: trailing zeros, trailing decimals, and whether or not you get a leading "1 x" in exponential notation.

You can turn NOF off by setting NOF=0. If you want trailing zeros omitted, set NOF=1. If you want trailing decimals omitted, set NOF=2. Or, if you have exponential numbers, you can change 1 x 10*13* to just 10*13* by setting NOF=4.

Notice that in the "NOF, Numeric Omission Flag" example, NOF=3 is a combination of NOF=1 and NOF=2. Similarly, NOF=5 is a combination of NOF=1 and NOF=4. NOF=6 is the default value, and indicates that exponentials are written without a leading "1 x", and trailing decimals are omitted. To get all the options, set NOF=7. The number of significant digits in each type of label varies, and this affects exactly how the label is written.

Note: Contour line labels and values in the information label that are known to have been rounded to "nice" values have trailing zeros trimmed, no matter what value is assigned to NOF.

Exercises

1. Using the **ccpnsd** example, comment out all statements referring to LIT and CIT, then remove trailing zeros and decimals.
2. Using the above exercise, put a zero before any decimal numbers.

   Hint: Run the **ccpnof** example if you would like to see the NOF and NLZ parameters demonstrated.

Cp 7.5 Numeric control: Exponents

Three parameters control exponents in numeric labels: NEL, NET, and NEU. These parameters control high, low, information, constant field, and contour line label output simultaneously.

Controlling exponent format

```
Table 3: NEL, Numeric Exponent Length
-------------------------------
3.5 x 1013     Label with NEL<=0
3.5 x 10+0013  Label with NEL=4

-------------------------------

Table 4: NET, Numeric Exponent Type
------------------------------
3.5 E+13      Label with NET=0
3.5 x 1013    Label with NET=1
3.5 x 10**13  Label with NET=2

------------------------------

Table 5: NEU, Numeric Exponent Use flag
---------------------------
6.9 x 104  Label with NEU<=0
69000      Label with NEU=3
---------------------------
```

Synopsis

```
      CALL CPSETI ('NEL', nel)
```

```
CALL CPSETI ('NET', net)
CALL CPSETI ('NEU', neu)
```

Arguments

NEL

Integer---The Numeric Exponent Length parameter controls whether or not leading zeros are used in an exponent. NEL=0 by default.

<=0 Exponents are written in the shortest possible form. Plus signs are omitted.

***n*>0**

Exponents are written with a sign, and the exponent is padded to a length of *n* with leading zeros.

NET

Integer---The Numeric Exponent Type parameter determines what characters are used between the mantissa of a numeric label and the exponent. By default, NET=1.

0 Specifies the use of an E, just as you would get if you used a Fortran FORMAT(E*n*.*m*)

1 Specifies that the exponent is superscripted.

2 Specifies a format like "1.11 x 10**12"

NEU

Integer---The Numeric Exponent Use flag decides when to force the use of exponential notation. By default, NEU=5.

<=0 Forces exponential notation in all labels.

***n*>0**

Specifies exponential notation when it uses no more than *n* characters. Otherwise, the form requiring the fewest characters is used.

Discussion

The "NEL, Numeric Exponent Length" illustration shows that NEL controls how many zeros are used to pad an exponent. By setting NEL>0, you can make your labels more consistent. NEL is normally used when NET=0 or NET=1.

NET is used to determine exactly how exponents are displayed: either in a Fortran format or in mathematical notation format. By default, exponents are displayed in a mathematical format.

The "NEU, Numeric Exponent Use flag" illustration shows that you can either force exponential form on every label, or you can use it only for labels that would be excessively long without exponents. The number of significant digits in each type of label varies, and this affects exactly how the label is written.

Exercises

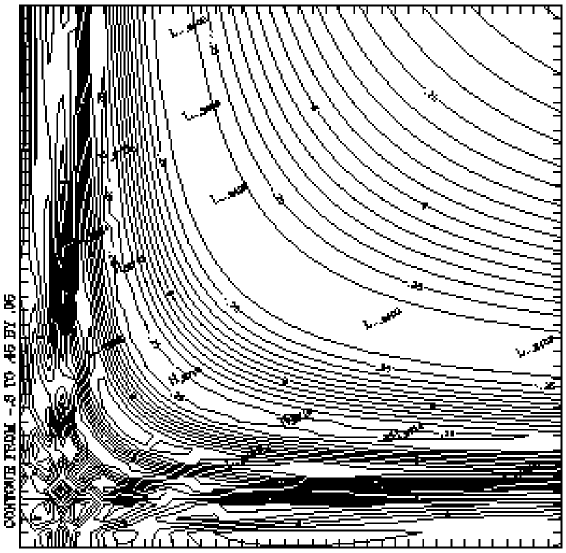
1. Using the **ccpnet** example, change the exponents to use the Fortran E format.
2. Using the **ccpnet** example, pad the exponents so that they are a minimum of two digits.

3. Using the **ccpnet** example, remove leading zeros from labels.

Cp 7.6 Label attributes: Angles

Three parameters control label angles or label orientations for high, low, information, and constant field labels: HLA, ILA, and CFA. These parameters allow you to control each kind of label individually.

Label angles



Code segment from ccpila.f

```
      CALL GSCLIP (0)
      CALL CPSETR ('HLA - HIGH/LOW LABEL ANGLE', 30.)
      CALL CPSETR ('ILA - INFORMATION LABEL ANGLE', -100.)
      CALL CPSETR ('ILX - INFORMATION LABEL X COORDINATE', -0.02)
      CALL CPSETR ('ILY - INFORMATION LABEL Y COORDINATE', 0.25)
      CALL CPSETI ('ILP - INFORMATION LABEL POSITION', 0)
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPLBDR (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETR ('CFA', cfa)
      CALL CPSETR ('HLA', hla)
      CALL CPSETR ('ILA', cila)
```

Arguments

CFA

Real---The Constant Field label Angle parameter specifies the angle in degrees at which a constant field label is to be drawn. By default, CFA=0.

HLA

Real---The High/Low label Angle parameter specifies the angle at which high and low labels are to be drawn. By default, HLA=0. All high and low labels must be drawn at the same angle.

ILA

Real---The Information Label Angle parameter specifies the angle in degrees at which the information label is written. By default, ILA=0.

Discussion

Line 1 of the **ccpila.f** code segment turns off clipping so that the information label shows up. Lines 2 and 3 set up the angles for the high, low, and information labels. Because there is no option to control label angle separately for the high and low labels, both high and low labels must be drawn at the same angle.

Lines 4 through 6 move the information label to the lower left edge of the plot. These parameters are described in detail in module "Cp 7.14 Label placement: Constant field and information labels."

In this example, we know that the data are nonconstant. There is therefore no reason to set up a constant field label angle. However, if you are plotting many data fields with the same subroutine, you may want to set CFA here as well. Lines 7 through 10 call Conpack normally.

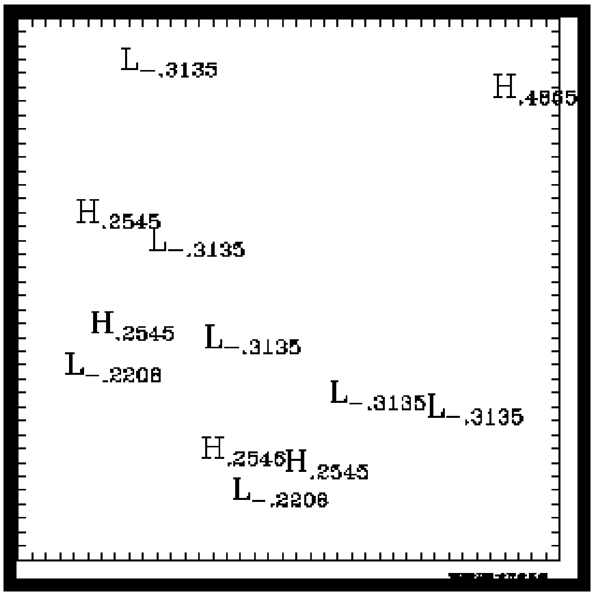
Exercises

1. Modify the **ccpila** example so that it prints out a constant field message, and turn the constant field message upside down.

Cp 7.7 Label attributes: Size

There are three options controlling contour label size, one for each type of label: CFS controls the constant field label, HLS controls the high and low labels, and ILS controls the information label.

Changing label sizes



Code segment from ccpils.f

```
      CALL CPSETR ('HLS - HIGH/LOW LABEL SIZE', .030)
      CALL CPSETR ('ILS - INFORMATION LABEL SIZE', .005)
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPLBDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETR ('CFS', cfs)
CALL CPSETR ('HLS', hls)
CALL CPSETR ('ILS', rils)
CALL CPSETR ('CWM', cwm)
```

Arguments

CFS

Real---The Constant Field label Size parameter controls the character size for constant field labels. 0.0<=CFS<=1.0. By default, CFS=.012.

HLS

Real---The High/Low label Size parameter controls the character size for high and low labels. 0.0<=HLS<=1.0. By default, HLS=.012.

ILS

Real---The Information Label Size parameter controls the character size for information labels. 0.0<=ILS<=1.0. By default, ILS=.012.

CWM

> *Real*---The Character Width Multiplier parameter allows you to scale all label sizes up or down simultaneously (all character size multipliers are multiplied by CWM). CWM affects the parameters CFS, DPS, HLS, ILS, LLS, CFW, DPV, HLW, ILW, and LLW. By default, CWM=1.0.

Discussion

All labels are sized in fractions of the width of the viewport. Rarely will you want labels larger than .02, and as shown in the **ccpils** example, it takes a printer of extremely high resolution to do justice to text at .005.

If you like the proportions of the labels, but you want to make them all smaller or larger at the same time, use the CWM parameter. For example, if you need the high/low labels and the information label to be 50% larger for better readability in publication, you would set CWM to 1.5.

Lines 1 and 2 of the **ccpils.f** code segment set high and low labels to be large, and set the information label to be very small. As shown in the graphic, the information label is small enough to be unreadable on many devices. Conpack is then called normally; however in this case, we eliminate the call to draw the contours since contours would obscure the high and low labels that we want to show.

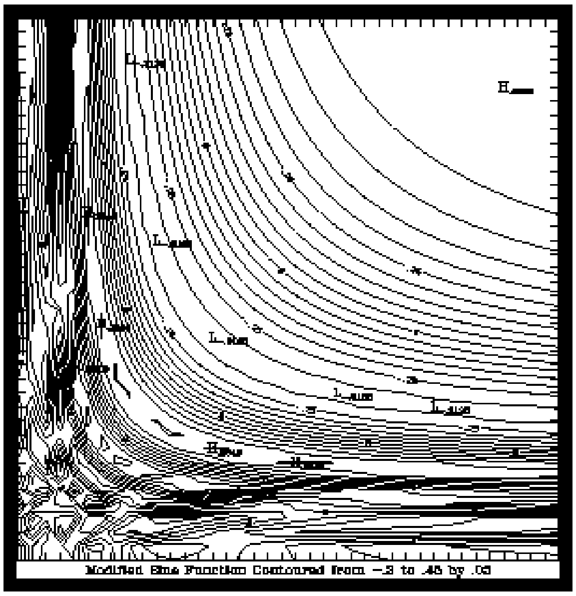
Exercises

1. Can you set high labels to be a different size than low labels? Why or why not?
2. Change the **ccpils** example so that all labels are drawn 25% larger than the default size.

Cp 7.8 Label attributes: Constant field and information text

The CFT parameter controls the constant field text, and the ILT parameter controls the information label text. Both labels can use special variables that retrieve and insert numerical information about your contour field.

Changing information label text



Code segment from ccpilt.f

```
       CALL CPSETC ('ILT - INFORMATION LABEL TEXT STRING', 'Modified Sine
     + Function Contoured from $CMN$ to $CMX$ by $CIU$')
       CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
       CALL CPBACK (Z, RWRK, IWRK)
       CALL CPLBDR (Z, RWRK, IWRK)
       CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETC ('CFT', cft)
CALL CPSETC ('ILT', cilt)
```

Arguments

CFT (CHARACTER*40)

> *Character expression*---The Constant Field label Text string parameter specifies the text of the constant field label. It can be up to 40 characters long and can display the numeric value of the field if the string ZDV is embedded in it. By default, CFT='CONSTANT FIELD VALUE IS ZDV'. If CFT=' ' (a single blank), the constant field label is not drawn.

ILT (CHARACTER*100)

> *Character expression*---The Information Label Text string parameter specifies the text of the information label. The following strings are replaced with numeric values in the label:

```
--------------------------------------
$CIU$  Contour interval used
$CMN$  Minimum contour level
$CMX$  Maximum contour level
$SFU$  Current scale factor
```

```
$ZMN$  Minimum value in the data array
$ZMX$  Maximum value in the data array
--------------------------------------
```

In each case except SFU, the given value is first divided by the current scale factor. You can insert U just before the final $ (for example $CIUU$ or $CMXU$) to request the unscaled value. CIU is only correct if Conpack chooses contour levels; otherwise, to get a correct value, you may need to set the parameter CIU before the label is drawn.
By default, ILT='CONTOUR FROM CMN TO $CMX BY CIU'. If ILT=' ' (a single blank), then the information label is not drawn.

Discussion

Lines 1 and 2 of the **ccpilt.f** code segment set up the information label text string to give us "Modified Sine Function Contoured from -.3 to .45 by .05." Notice that we have used the variables $CMN, $CMX, and $CIU to get the minimum and maximum contour levels, and the contour level interval. The rest of the code draws contours and labels in the familiar way.

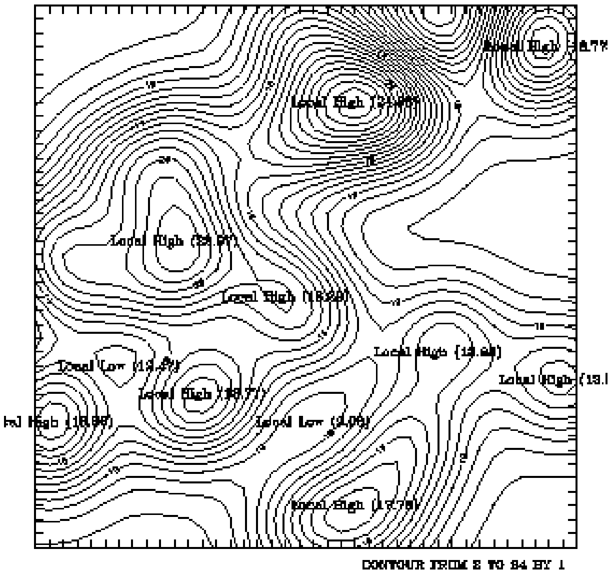
Exercises

1. Change the **ccpilt** example to use the default information label, but make it print out the scale factor. In addition, move the information label to the left side of the plot and make the text run vertically.

Cp 7.9 Label attributes: High/low text

The HLT parameter controls high and low label text strings. It allows you to specify the character strings for labeling highs and lows either individually or together.

Changing the text in high and low labels



Code segment from ccphlt.f

```
1       CALL CPSETC ('HLT - HIGH/LOW LABEL TEXT STRINGS',
       + 'Local High ($ZDV$)''Local Low ($ZDV$)')
2       CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
3       CALL CPBACK (Z, RWRK, IWRK)
4       CALL CPCLDR (Z, RWRK, IWRK)
5       CALL CPLBDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETC ('HLT', hlt)
```

Arguments

HLT(CHARACTER*40)

> *Character expression*---The High/Low label Text strings parameter allows you to specify the text strings you want to print for high and low labels. By setting HLT to a single text string (one with no embedded single quotation marks), highs and lows are both labeled the same way.
> By setting HLT to a text string that has two single quotation marks together inside the string, the text before the single quotation marks will be used to label highs, and the text after the single quotation marks will be used to label lows. Neither the high nor the low text string may exceed 20 characters.
> The substring ZDV, when included in a label, is to be replaced by the value of the high or low divided by the scale factor. $ZDVU$ is to be replaced by the unscaled value of the high or low.

Discussion

To give you a feel for how HLT works, here is a table showing the Fortran string and the corresponding result:

Fortran string	High label	Low label
1. 'ZDV'	1.363	-5.6
2. 'Inflection Point'	Inflection Point	Inflection Point
3. 'High''Low'	High	Low
4. 'H (ZDV)''L (ZDV)'	H (1.363)	L (-5.6)
5. 'H:B:ZDV:E:''L:B:ZDV:E:'	H*1.363*	L*-5.6*

In string 1, high and low labels are set to the numerical value of the point by using ZDV. Since there is only a single quoted string in string 2, all highs and lows are labeled with the words "Inflection Point". String 3 labels highs with the word "High", and lows with the word "Low". String 4 uses ZDV to print an "H" for highs, and "L" for lows, and the value of the high or the low in parentheses.

String 5 gives the default labels for highs and lows. In the first part, "H" is the text, ":B:" is a Plotchar function code to induce subscripting, "ZDV" prints the numerical value of the high as a subscript, and

":E:" centers the "H" on the high. The low label is nearly identical, using "L" to specify lows, rather than "H". For more information on the Plotchar function codes, please see Chapter 11 "Drawing text and symbols" in the *NCAR Graphics Fundamentals* guide.

Line 1 of the **ccphlt.f** code segment sets the high and low labels to be rather long strings of "Local High (*val*)" and "Local Low (*val*)" where *val* is the value of the high or low. The clipping of the labels is caused by the length of the label. Generally, it is best to make high and low labels as short as possible. Lines 2 through 5 initialize and draw contours with the usual calls.

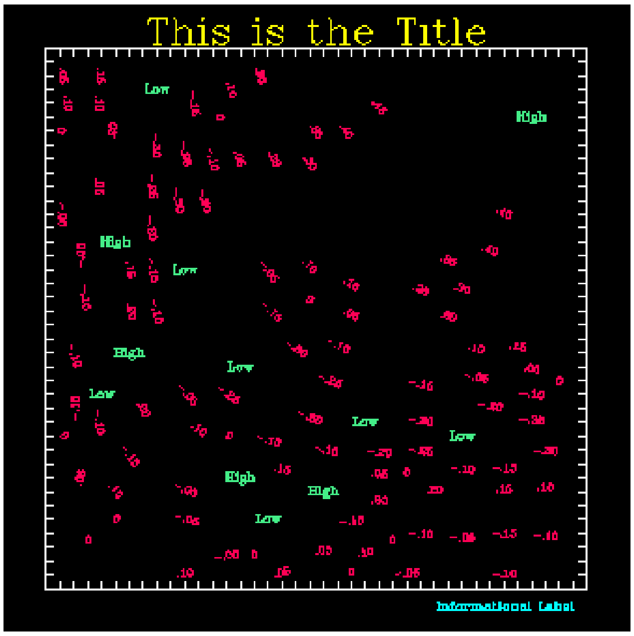
Exercises

1. Using the **ccphlt** example, set high labels to the default, and set low labels so that they just print out the value of the low.
2. Change the **ccphlt** example to make the words "High" and "Low" come out as superscripts to the values of the highs and lows.

Cp 7.10 Label attributes: Color

Five parameters control the color of high, low, information, and constant field labels: HIC, LOC, HLC, ILC, and CFC. You can control each kind of label individually, or the high and low labels as an entity.

Label color



Code segment from ccplbdr.f

```
      CALL GSCLIP (0)
      CALL COLOR
      CALL CPSETI ('HLC - HIGH/LOW LABEL COLOR INDEX', 9)
      CALL CPSETC ('HLT - HIGH/LOW LABEL TEXT STRINGS', 'High' 'Low')
```

```
      CALL CPSETI ('ILC - INFORMATION LABEL COLOR INDEX', 2)
      CALL CPSETC ('ILT - INFORMATION LABEL TEXT', 'Informational Label')
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (Z, RWRK, IWRK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPLBDR (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('CFC', icfc)
CALL CPSETI ('HLC', ihlc)
CALL CPSETI ('ILC', icilc)
```

Arguments

CFC

Integer---The Constant Field label Color index specifies which color is used to draw the constant field label.

<0 The label (and the box around it, if any) are drawn in the color specified by the current text color index. This is the default.

>=0 The label (and the box around it, if any) is drawn in the color specified by the color table.

HIC

Integer---The HIgh label Color index is used to determine the color index for high labels. HIC is partially dependent on HLC, described below. HIC=-1 by default.

LOC

Integer---The LOw label Color index is used to determine the color index for low labels. LOC is partially dependent on HLC, described below. LOC=-1 by default.

HLC

Integer---The High/Low label Color index is used to determine the color index for high and low labels. HLC=-1 by default.

The color index for high labels is determined in this manner:

If HIC>=0, HIC is used as the color index.
If HIC<0, but HLC>=0, HLC is used as the color index.
If both HIC and HLC are less than zero, the current text color index is used.

The color index for low labels is determined similarly:

If LOC>=0, LOC is used as the color index.
If LOC<0, but HLC>=0, HLC is used as the color index.
If both LOC and HLC are less than zero, the current text color index is used.

ILC

Integer---The Information Label Color index specifies the color of the information label.

Discussion

The HIC, LOC, and HLC parameters show that there are several ways to set high and low label color. If you want the label color to match the text color, do nothing (use the default setting). If you want to set high and low labels together, you can do this with a single parameter: set HLC>0. If you want to set high labels to one color, and low labels to another color, you can do this by setting HIC and LOC.

If you have defined a color table, and you set any of the preceding parameters equal to zero, then the label(s) come out in the background color---the one color that does not show up on your plot! In other words, if you set ILC=0 and your background color is black, then both your background and your information label are black, making the information label invisible.

Line 1 of the **ccplbdr.f** code segment turns off clipping so that the information label shows up. Line 2 sets up the color table via a subroutine call. Lines 3 and 5 specify the color indices for the high, low, and information labels. Lines 4 and 6 specify the text for these labels.

In this example, we know that the data are nonconstant. There is therefore no reason to set up a constant field label color index. However, if you are plotting many data fields with the same subroutine, you may want to set CFC here as well.

Lines 7 through 11 call Conpack normally. Notice that the text of the high, low, and information labels is not the default. Changing label text is discussed in the label text modules Cp 7.8 and Cp 7.9.

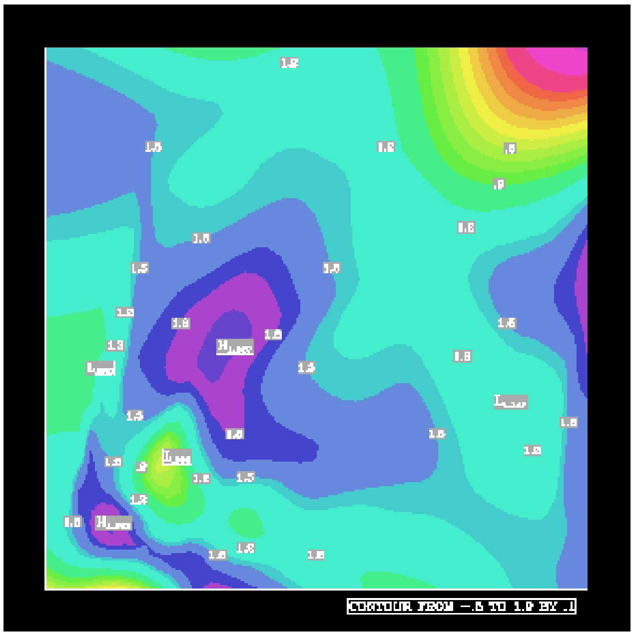
Exercises

1. Modify the **ccplbdr** example so the title is white, highs are green, lows are red, contour line labels are blue, and the information label is magenta.

Cp 7.11 Label box attributes: Outline and fill

There are a few label attributes that have not been previously discussed because you may need to put your contours and labels into an area map to use them. Information, constant field, contour line, and high/low labels can all be boxed to protect them from contour lines or various fill options. The label box flags, CFB, HLB, ILB, and LLB control whether or not a label box is outlined or filled.

Setting contour label box attributes



Code segment from ccpllb.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 2)
      CALL CPSETI ('LLB - LINE LABEL BOX FLAG', 2)
      CALL CPSETI ('HLB - HIGH/LOW LABEL BOX FLAG', 3)
      CALL CPSETI ('ILB - INFORMATION LABEL BOX FLAG', 1)
      CALL CPSETI ('LBC - LABEL BOX COLOR INDEX', 18)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('CFB', icfb)
CALL CPSETI ('HLB', ihlb)
CALL CPSETI ('ILB', ilb)
CALL CPSETI ('LLB', llb)
CALL CPSETI ('LBC', lbc)
```

Arguments

CFB

Integer---The Constant Field label Box flag controls how the constant field label is boxed.

0 Labels drawn by CPLBDR are not boxed. CFB=0 by default.
1 The perimeter of the box is drawn in the same color as the label, after the label is drawn.
2 The box is filled in the color specified by LBC before the label is drawn.
3 Both 1 and 2 above.

HLB

Integer---The High/Low label Box flag controls how high and low labels are boxed.

0 Labels drawn by CPLBDR are not boxed. HLB=0 by default.
1 The perimeter of the box is drawn in the same color as the label, after the label is drawn.
2 The box is filled in the color specified by LBC before the label is drawn.
3 Both 1 and 2 above.

ILB

Integer---The Information Label Box flag controls how information labels are boxed.
0 Labels drawn by CPLBDR are not boxed. ILB=0 by default.
1 The perimeter of the box is drawn in the same color as the label after the label is drawn.
2 The box is filled in the color specified by LBC before the label is drawn.
3 Both 1 and 2 above.

LLB

Integer---The Line Label Box flag controls how contour line labels are boxed:
0 Labels drawn by CPLBDR are not boxed. LLB=0 by default.
1 The perimeter of the box is drawn in the same color as the label after the label is drawn.
2 The box is filled in the color specified by LBC before the label is drawn.
3 Both 1 and 2 above.

LBC

Integer---The Label Box Color index parameter sets the color for the interiors of label boxes. By default, LBC=0.
<0 Current fill color index is used.
>=0 Specifies color index to be used.

Discussion

If you are using a device that can overlay images, such as an X-window display or a Video Display Terminal (VDT), you don't need to use the label box options in conjunction with Areas. Simply draw your labels (by calling CPLBDR) after drawing your contours (by calling (CPCLDR).

Line 1 of the **ccpllb.f** code segment chooses the regular scheme for line labeling. Line 2 sets the line label box to be filled. Line 3 sets the high and low label boxes to be filled and outlined. Line 4 sets the information label to be outlined but not filled. Line 5 sets the internal box color to gray. Line 6 initializes Conpack, and line 7 draws the labels.

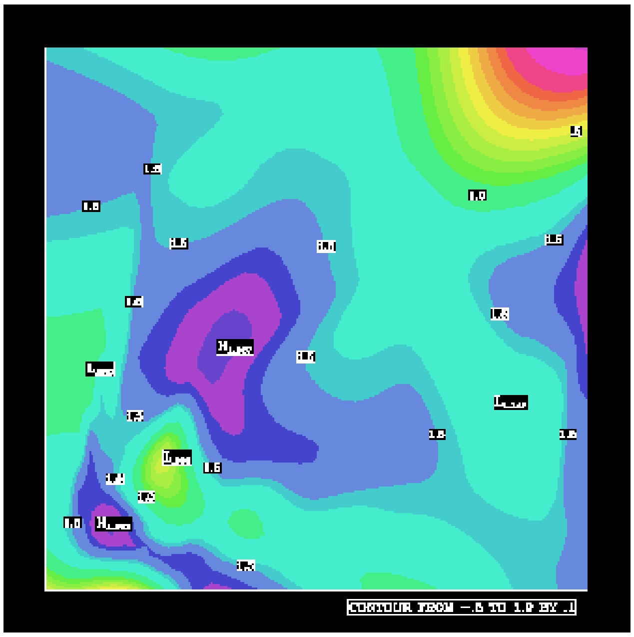
Exercises

1. Change the box color to aqua in the **ccpllb** example.
2. Modify the **ccpllb** example so that all the boxes are black with white outlines.

Cp 7.12 Label box attributes: Line width

Label boxes for constant field, high/low, information, and contour line labels have line width (thickness) control parameters: CFL, HLL, ILL, and LLL respectively.

Label box width



Code segment from ccplll.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 2)
      CALL CPSETI ('LLB - LINE LABEL BOX FLAG', 1)
      CALL CPSETI ('HLB - HIGH/LOW LABEL BOX FLAG', 1)
      CALL CPSETI ('ILB - INFORMATION LABEL BOX FLAG', 1)
      CALL CPSETR ('LLL - LINE LABEL LINE WIDTH', 1.)
      CALL CPSETR ('HLL - HIGH/LOW LABEL LINE WIDTH', 3.)
      CALL CPSETR ('ILL - INFORMATION LABEL LINE WIDTH', 5.)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPPKCL (ZREG, RWRK, IWRK)
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL ARINAM (MAP, LMAP)
      CALL CPLBAM (ZREG, RWRK, IWRK, MAP)
      CALL CPLBDR (ZREG, RWRK, IWRK)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, FILL)
```

Synopsis

```
CALL CPSETR ('CFL', cfl)
CALL CPSETR ('HLL', hll)
CALL CPSETR ('ILL', rill)
CALL CPSETR ('LLL', rlll)
```

Arguments

CFL

Real---The Constant Field label box Line width parameter determines the line width of boxes drawn around labels. CFL=0.0 by default.

<=**0.** Line width is not set before drawing boxes around constant field labels.

>**0.** The value indicates the desired width as a multiple of the normal line width.

HLL

Real---The High/Low label box Line width parameter determines the line width of boxes drawn around labels. HLL=0.0 by default.

<=**0.** Line width is not set before drawing boxes around high/low labels.

>**0.** The value indicates the desired width as a multiple of the normal line width.

ILL

Real---The Information Label box Line width parameter determines the line width of boxes drawn around labels. ILL=0.0 by default.

<=**0.** Line width is not set before drawing boxes around information labels.

>**0.** The value indicates the desired width as a multiple of the normal line width.

LLL

Real---The Line Label Line width parameter determines the line width of boxes drawn around labels. LLL=0.0 by default.

<=**0.** Line width is not set before drawing boxes around contour line labels.

>**0.** The value indicates the desired width as a multiple of the normal line width.

Discussion

Lines 5 through 7 set the width of box lines in the **ccplll.f** code segment. Line 5 sets the contour label box line thickness to 1.0, giving us the default line thickness. Line 6 sets the high/low box line thickness to 3.0, drawing a reasonably thick line, and line 7 sets the information box line thickness to 5.0, drawing a very heavy line. Not all output devices are created equal, and on many printers you may have to use a much broader spectrum of widths to get the kind of differentiation shown here.

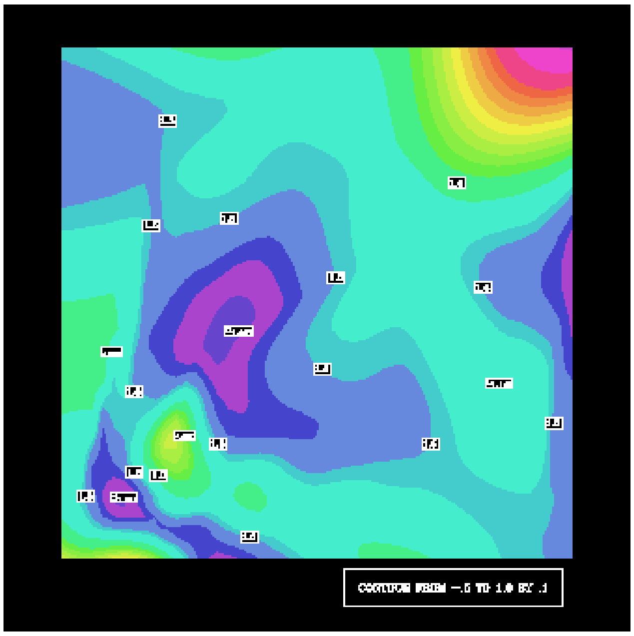
Exercises

1. Using the **ccplll** example, change the high/low label box line thicknesses to .1, and the information label box line thickness to 10.

Cp 7.13 Label box attributes: Size

Conpack also provides parameters that control the amount of white space in each label box. These parameters therefore control the size of the label box.

Controlling label box size



Code segment from ccpllw.f

```
      CALL CPSETI ('LLP - LINE LABEL POSITIONING', 2)
      CALL CPSETI ('LLB - LINE LABEL BOX FLAG', 1)
      CALL CPSETI ('HLB - HIGH/LOW LABEL BOX FLAG', 1)
      CALL CPSETI ('ILB - INFORMATION LABEL BOX FLAG', 1)
      CALL CPSETR ('LLW - LINE LABEL WHITE SPACE', .02)
      CALL CPSETR ('HLW - HIGH/LOW LABEL WHITE SPACE WIDTH', 0.)
      CALL CPSETR ('ILW - INFORMATION LABEL WHITE SPACE WIDTH', .03)
      CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (ZREG, RWRK, IWRK)
      CALL ARINAM (MAP, LMAP)
      CALL CPLBAM (ZREG, RWRK, IWRK, MAP)
      CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, FILL)
      CALL CPLBDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CALL CPSETR ('LLW', rllw)
```

Arguments

CFW

Real---The Constant Field label White space parameter controls the width of the white space around a constant field label. W=LLW*CWM, where W is white space width in fractions of the viewport width; CWM is described below. CFW=0.005 by default.

HLW

Real---The High/Low label White space parameter controls the width of the white space around a high/low label. W=LLW*CWM, where W is white space width in fractions of the viewport width; CWM is described below. HLW=0.005 by default.

ILW

Real---The Information Label White space parameter controls the width of the white space around an information label. W=LLW*CWM, where W is white space width in fractions of the viewport width; CWM is described below. ILW=0.005 by default.

LLW

Real---The contour Line Label White space parameter controls the width of the white space around a contour line label. W=LLW*CWM where W is white space width in fractions of the viewport width; CWM is described below. LLW=0.005 by default.

CWM

Real---The Character Width Multiplier parameter affects all label sizes. This makes it easy to scale the sizes of all labels up or down simultaneously. CWM affects CFS, DPS, HLS, ILS, LLS, CFW, DPV, HLW, ILW, and LLW. By default, CWM=1.0.

Discussion

Lines 5 through 7 of the **ccpllw.f** code segment control the sizes of label boxes. Line 5 tells Conpack that we don't want any white space around the contour label; this produces small contour line labels. Line 6 asks for a large amount of white space around the high and low labels; this produces very large boxes. Line 7 asks for a moderate amount of white space around the information label and produces a nice margin.

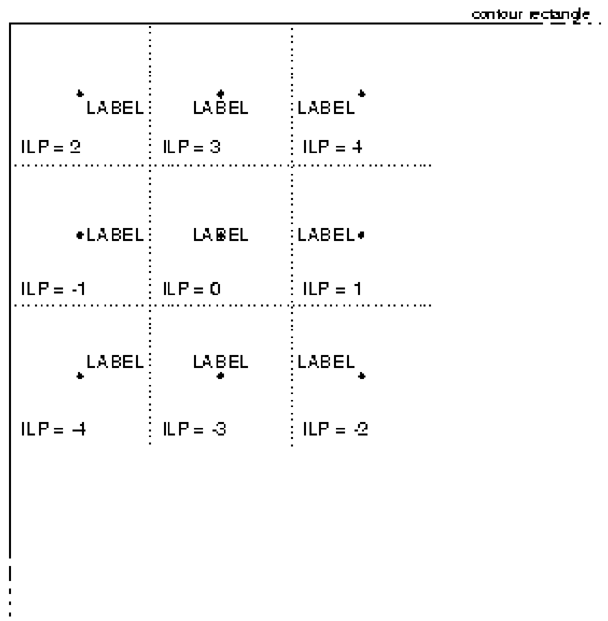
Exercises

1. Using the **ccpllw** example, make the label boxes for the contour lines reasonably large.

Cp 7.14 Label placement: Constant field and information labels

Six parameters allow you to control the position of the constant field label or the information label in your plot. CFX and CFY specify the X and Y coordinates of the base point of the constant field label. ILX and ILY specify the X and Y coordinates of the base point of the information labels. CFP and ILP allow you to control the labels' positions relative to the base point.

Constant field and information label positioning



Code segment from ccpcfx.f

```
      CALL CPSETR ('CFX - CONSTANT FIELD LABEL X COORDINATE', 0.)
      CALL CPSETR ('CFY - CONSTANT FIELD LABEL Y COORDINATE', 1.)
      CALL CPSETI ('CFP - CONSTANT FIELD POSITION FLAG', 2)
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPLBDR (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
```

Synopsis

```
CALL CPSETR ('CFX', cfx)
CALL CPSETR ('CFY', cfy)
CALL CPSETI ('CFP', icfp)
CALL CPSETR ('ILX', rilx)
CALL CPSETR ('ILY', rily)
CALL CPSETI ('ILP', ilp)
```

Arguments

CFX and **CFY**

> *Real*---The Constant Field label X coordinate and Constant Field label Y coordinate parameters specify the X and Y viewport coordinates, in NDCs, of a point relative to which the constant field label is to be positioned as specified by the value of CFP. By default, CFX=CFY=0.5.

ILX and **ILY**

> *Real*---The Information Label X coordinate and Information Label Y coordinate parameters specify the X and Y viewport coordinates, in NDCs, of a point relative to which the information label is to be positioned as specified by the value of ILP. By default, ILX=0.98, and ILY=-.02.

CFP and **ILP**

> *Integer*---The Constant Field label Positioning flag and the Information Label Positioning flag allow you to change the X and Y centering options of the text so that you can place any corner or the center of the label at the point specified by (CFX, CFY) or (ILX, ILY). The illustration shows the positions and values of CFP and ILP for a label.

Discussion

If the dot in the illustration represents the point (ILX, ILY), the "ILP=" value shows how the label is drawn relative to the point. If the dot in the illustration represents the point (CFX, CFY), the parameter CFP produces the same results as the "ILP=" values shown.

Lines 1 and 2 of the **ccpcfx.f** code segment place the constant field label in the upper left corner of the contour rectangle. We know that the upper left corner of the contour rectangle has the coordinates (0.0, 1.0), so we set CFX=0.0, and CFY=1.0.

To avoid positioning the label outside the contour rectangle, line 3 sets CFP=2 so that the label has its upper left corner at the point (CFX, CFY).

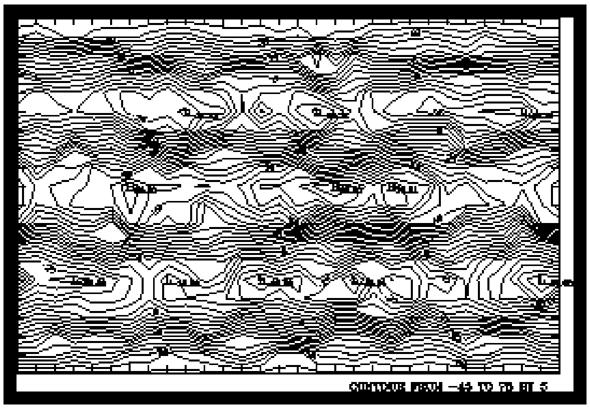
Exercises

1. Using the **ccpcfx** example, move the constant field label to below the lower right corner of the plot so that it is positioned in the same place as an information label.
2. Using the **ccpila** example, move the information label to the top right corner of the plot.

Cp 7.15 Label placement: High/low

High and low labels have three placement control parameters: HLO (the overlap flag), HLX, and HLY (the X and Y search radius parameters). These parameters allow you to control how closely together labels are positioned, and whether or not they are allowed to overlap other labels or the contour rectangle.

High and low label placement



Code segment from ccphl.f

```
      CALL CPSETI ('HLX - HIGH/LOW SEARCH RADIUS IN X', 2)
      CALL CPSETI ('HLY - HIGH/LOW SEARCH RADIUS IN Y', 2)
      CALL CPSETI ('HLO - HIGH/LOW LABEL OVERLAP FLAG', 0)
      CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
      CALL CPBACK (Z, RWRK, IWRK)
      CALL CPCLDR (Z, RWRK, IWRK)
      CALL CPLBDR (Z, RWRK, IWRK)
```

Synopsis

```
      CALL CPSETI ('HLX', ihlx)
      CALL CPSETI ('HLY', ihly)
      CALL CPSETI ('HLO', ihlo)
```

Arguments

HLX and **HLY**

Integer---The High/Low search radius in X and High/Low search radius in Y parameters specify the search radius for high and low values in a contour field. Since data points are searched, the smallest search radius in either the X or Y direction is one data point. For example, if Z(I,J) were a potential low, and the search radius in each direction were 1, then Z(I-1,J-1), Z(I-1,J), Z(I1,J+1), Z(I,J-1), Z(I,J+1), Z(I+1,J-1), Z(I+1,J), and Z(I+1,J+1) would be compared against Z(I,J) to see if any were smaller.

>0 Specifies the search radius for highs and lows.
<=0 Conpack picks a reasonable value. HLX is set to approximately 1/8*ZDM, where ZDM is the number of data points or interpolated data points in the X direction. Similarly, HLY is set to approximately 1/8*ZDN, where ZDN is the number of data points or interpolated data points in the Y direction. 2<=HLX, HLY<=15.

HLO

Integer---The High/Low label Overlap flag determines what objects high and low labels can overlap. You can produce combined effects by adding any of the following values together. For example, to eliminate high and low labels that overlap the information label and to eliminate high and low labels that overlap each other, set HLO = 1 + 2 = 3.

0 Labels are not checked for any kind of overlap.
1 High and low labels that overlap the information label are omitted.
2 High and low labels that overlap other high and low labels are omitted.
4 High and low labels that overlap the contour rectangle are omitted.
8 High and low labels that overlap the contour rectangle are moved inwards enough to eliminate overlap. (HLO=4 has precedence over HLO=8.)

Discussion

The best way to think about HLX and HLY is that they describe a rectangle 2*HLX units wide and 2*HLY units high, where a unit is the distance between your data points. All data points within that rectangle are compared with the center point; and if all the points are lower or higher than the center point, then it is marked as a local high or low.

Lines 1 and 2 of the **ccphl.f** code segment set the search area for highs and lows to be a square 2 units wide by 2 units high. All points next to the center point are checked to see if they are higher or lower than the center. By doing this, we make sure that every possible high or low is marked. Line 3 turns off checking to see if high and low labels overlap with anything else. Lines 4 through 7 draw contours with the familiar calls.

Exercises

1. Correct the **ccphl** example so that it gives a single high label and a single low label for the plot.

Cp 7.16 Constant field found flag

The constant field found flag, CFF, is set to a nonzero value if CPRECT, CPSPS1, or CPSPS2 detects a constant field. Users may not set this parameter.

Code segment from ccpcff.f

```
1        CALL CPRECT (Z, M, M, N, RWRK, LRWK, IWRK, LIWK)
2        CALL CPGETI ('CFF - CONSTANT FIELD FOUND FLAG', ICFF)
3        IF (ICFF .NE. 0) GOTO 101
4           CALL CPBACK (Z, RWRK, IWRK)
5           CALL CPLBDR (Z, RWRK, IWRK)
6           CALL CPCLDR (Z, RWRK, IWRK)
7  101   WRITE (6,*) 'The field is constant.'
8        STOP
9        END
```

Synopsis

```
      CALL CPGETI ('CFF', icff)
```

Arguments

CFF

> *Integer*---The Constant Field Found flag is nonzero if CPRECT, CPSPS1, or CPSPS2 finds a constant field. You cannot set this parameter, and its default value is zero.

Discussion

Line 2 of the **ccpcff.f** code segment retrieves CFF so that we can check it in line 3 to see if a constant field was found. In this case, when a constant field is found, we exit the program. You could use this option to detect errors in your program or data, then conclude with a graceful exit.

Cp 8. Overlaying filled contours on a map

This is one of the most complex subjects in Conpack. Not only does it require knowledge of most of the preceding sections on Conpack, it also relies heavily on understanding both Areas and Ezmap. In this section, we develop a subroutine that can be easily modified to draw a "default" overlay of contours on a map with color fill.

Conpack functional outline

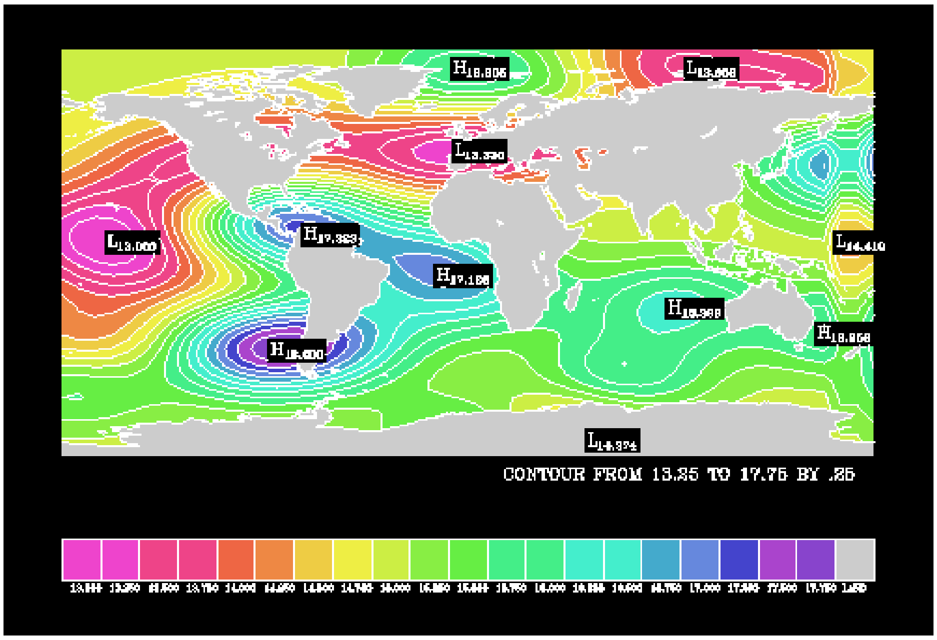
```
----------------------------------------------------------
    1.  Open GKS
 *  2.  Set window and viewport
    3.  Put in titles and other labels
 *  4.  Set up general parameters
 *  5.  Set up area maps
 *  6.  Initialize Conpack
 *  7.  Force Conpack to choose contour levels if necessary
 *  8.  Modify Conpack-chosen parameters
 *  9.  Draw labels
 *  10.  Draw background
 *  11.  Draw contour lines
    12.  Call FRAME
    13.  Close GKS

----------------------------------------------------------
```

* Steps discussed in this section.

Cp 8.1 Initialization

In Areas section "Ar 2. Initializing Areas," Ezmap module "Mp 4.3 Initialize Ezmap with Areas," and Conpack module "Cp 5.3 Initialize Conpack with Areas," we discuss each of the routines that must be called to initialize Conpack, Ezmap, and Areas together. This module does not introduce any new routines, it simply describes how to use all the necessary subroutines together.

Filled contours in a map



Code segment from colcon.f

```
1         EXTERNAL FILL
2         EXTERNAL MASK
3
4         CALL GSFAIS (1)
5         CALL ARINAM (MAP, LMAP)
6         CALL MPSETR ('GR - GRID SPACING', 0.)
7         CALL MPSETC ('OU - OUTLINE DATA FLAG', 'CO')
8         CALL MAPROJ (PROJ, PLAT, PLON, 0.0)
9         IF (PROJ .EQ. 'SV') CALL MPSETR ('SA - SATELLITE VIEW DISTANCE', 10.)
10        CALL MAPSET ('CO', RLATMN, RLONMN, RLATMX, RLONMX)
11        CALL MAPINT
12        CALL MAPBLA (MAP)
13        CALL CPSETI ('SET - DO-SET-CALL FLAG', 0)
14        CALL CPSETI ('MAP - MAPPING FLAG', 1)
15        CALL CPSETR ('ORV - OUT-OF-RANGE VALUE', 1.E12)
16        CALL CPSETR ('XC1 - X COORDINATE AT INDEX 1', RLONMN)
17        CALL CPSETR ('XCM - X COORDINATE AT INDEX M', RLONMX)
18        CALL CPSETR ('YC1 - Y COORDINATE AT INDEX 1', RLATMN)
19        CALL CPSETR ('YCN - Y COORDINATE AT INDEX N', RLATMX)
20        CALL CPSETI ('LLP - LINE LABEL POSITIONING', 0)
21        CALL CPSETI ('HLB - HIGH/LOW LABEL BOX FLAG', 2)
22        CALL CPSETI ('HLC - HIGH/LOW LABEL COLOR INDEX', 1)
23        CALL CPRECT (ZREG, MREG, MREG, NREG, RWRK, LRWK, IWRK, LIWK)
24        CALL CPCLAM (ZREG, RWRK, IWRK, MAP)
25        CALL CPLBAM (ZREG, RWRK, IWRK, MAP)
```

Discussion

The **colcon** example performs both filling and masking, so you need to declare the filling and masking routines external; lines 1 and 2 of the **colcon.f** code segment do this. Line 4 sets the fill area option to solid. GKS is opened in the main program (not shown here) to finish calling the routines to initialize GKS.

Line 5 initializes Areas with a call to ARINAM. This is the only routine required for initializing Areas.

Lines 6 through 11 are the Ezmap routines needed to initialize our plot. In this case, line 6 turns off grid lines. Line 7 turns on continental outlines. Lines 8 and 9 choose a projection and give a default satellite height in case the satellite projection is chosen. Line 10 sets the corners of the map to be the same as the corners of our contour data, as it appears in lines 16 through 19. Line 11 initializes our geographic map, and line 12 adds those lines to the area map.

You may want to draw or fill contours over a geographic map with no masking over geographic regions. In cases like this, it is not a good idea to call MAPBLA and add the geographic map to the area, map because adding the geographic map to the area map is enormously expensive, both in memory (since the area map has to be at least 150,000 words) and in CPU time.

Lines 13 through 23 set up various Conpack options then initialize Conpack. Because MAPINT makes a call to SET, you need to tell Conpack to honor that call; line 13 does this. Line 14 tells Conpack that we have lat/lon data, and lines 16 through 19 set the minimum and maximum latitude and longitude data locations. Remember that longitude is the X direction and latitude is the Y direction.

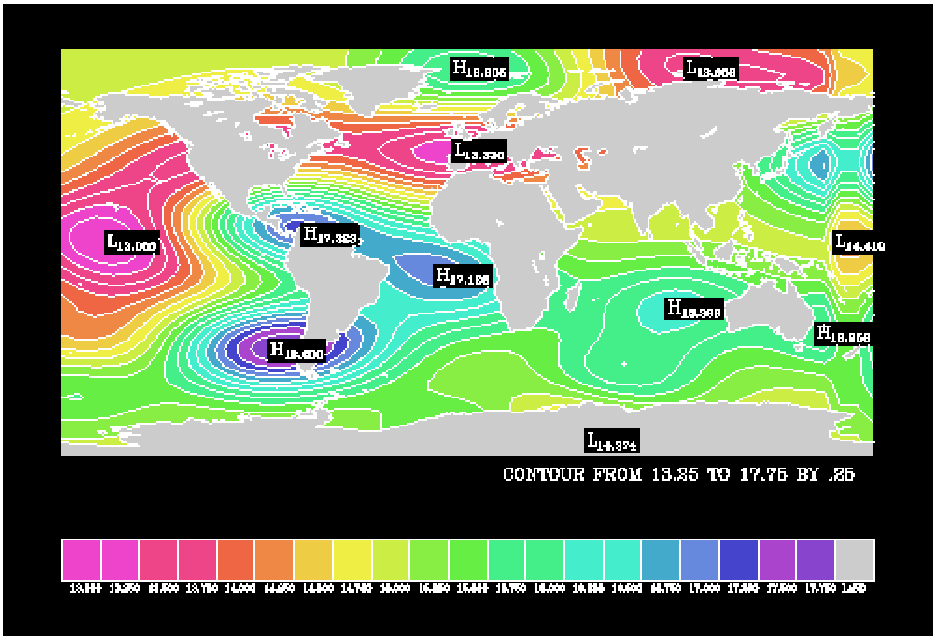
Since colors denote the contour level values in this example, line 20 turns off line labeling. Lines 21 and 22 control the high/low label box fill and color options. Line 23 finally initializes Conpack, then lines 24 and 25 add the contours and the labels to the area map.

Note: This example does not use real data.

Cp 8.2 Masking areas: Label boxes and land masses

In Areas module "Ar 3.3 Writing a masking or line-drawing routine," Ezmap module "Mp 4.6 Grid lines with masking: Writing a masking routine," and Conpack module "Cp 5.6 Masking areas: Box masking routines," we discuss writing and using various masking routines. This module expands on the Conpack masking routine CPDRPL to create a routine that masks contour lines over both labels and land masses. We also discuss the ordering of the calls that fill and draw in this program.

Filled contours in a map



Code segment from colcon.f

```
      EXTERNAL MASK
      CALL ARSCAM (MAP, XWRK, YWRK, NWRK, IAREA, IGRP, NOGRPS, FILL)
      CALL MAPLOT
      CALL CPLBDR (ZREG, RWRK, IWRK)
      CALL CPCLDM (ZREG, RWRK, IWRK, MAP, MASK)

      SUBROUTINE MASK (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)
      INTEGER IAREA(NGRPS), IGRP(NGRPS)
      REAL XWRK(NWRK), YWRK(NWRK)

      IDMAP = -1
      IDCONT = -1
      DO 10, I=1, NGRPS
          IF (IGRP(I) .EQ. 1) IDMAP = IAREA(I)
          IF (IGRP(I) .EQ. 3) IDCONT = IAREA(I)
 10   CONTINUE
      IF (NWRK .LT. 2) RETURN
      IF ((IDMAP .GT. 0).AND.(IDCONT .GT. 0).AND.(MAPACI(IDMAP) .EQ. 1)) THEN
          CALL CURVED (XWRK, YWRK, NWRK)
      ENDIF
      RETURN
      END
```

Arguments

MASK

Subroutine---A user-supplied external subroutine to process polylines resulting from masking the generated contour lines and other edges against the area map. The routine that calls CPCLDM must declare MASK as external.
MASK is called repeatedly and must have the following form:

```
SUBROUTINE MASK (XCS, YCS, NCS, IAREA, IGRP, ISIZ)
DIMENSION XCS(*), YCS(*)
DIMENSION IAREA(*), IGRP(*)
```

(code to process polyline defined by arguments)

```
RETURN
END
```

The arrays XCS and YCS hold the X and Y coordinates of NCS points defining a polyline that is to be considered for drawing. For each I such that 1<=I<=ISIZ, IAREA(I) is the area identifier for the area in which the polyline lies, relative to the area identifier group IGRP(I). The X and Y coordinates are all NDCs, and you can assume that the appropriate SET call has been made. If it is decided to draw the line, it may be done with a call to the SPPS routine CURVE, to the Dashline routine CURVED, or to the GKS routine GPL.
If the only object of using CPCLDM is to avoid drawing contour lines through label boxes, then the Conpack routine CPDRPL may be used for MASK. In the subroutine that calls CPCLDM, insert the declaration EXTERNAL CPDRPL, and then use CPDRPL for the last argument.

Discussion

Examine the CPDRPL routine and note that label boxes always have negative area identifiers. By refusing to draw contour lines over areas with nonpositive areas, we can be sure that we don't draw contour lines over label boxes. Also, since we want to draw contour lines over the ocean, we can pick out the oceans using the MAPACI function.

Lines 7 through 9 of the **colcon.f** code segment set our subroutine call and variables as specified by the CPCLDM documentation. Lines 11 and 12 initialize storage variables for the geographic map and contour area identifiers respectively. Lines 13 through 16 check each group identifier, and if it's one of the two groups we're interested in, then we retrieve the area identifier for that group. Line 20 protects against trying to draw points with a line-drawing routine. Line 18 checks to see if we are over the geographic map, over the contour area, and over the ocean. If so, then line 19 draws the contour line segment.

Note that our contouring routine first fills contours (line 2), second draws the map outlines (line 3), then draws labels and masked contour lines (lines 4 and 5). You must do the filling first because if you first draw lines and then fill over them, they become invisible. By calling MAPLOT to draw the continental outlines before filling the label boxes, we are using layering to effectively mask out geographic lines that run through the label boxes. This is the only way to do this because there is currently no way to mask the geographical outlines without modifying Ezmap source code.

Note: This example does not use real data.

Exercises

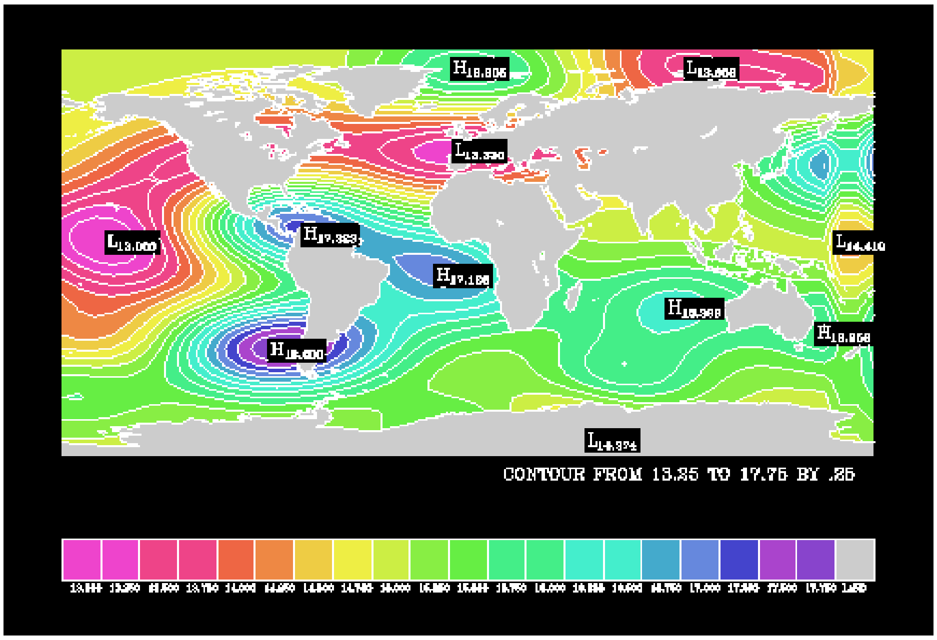
1. Using the **colcon** example, draw contour lines over land and fill them over water.

Cp 8.3 Filling specific contour and geographic areas

Filling specific contour levels and geographic areas is yet another application of the APR subroutine in ARSCAM.

Examples of fill routines are presented in Areas module "Ar 3.5 Writing an area-fill routine," Ezmap module "Mp 4.6 Grid lines with masking: Writing a masking routine," and Conpack module "Cp 5.6 Masking areas: Box masking routines." The difference between these previously discussed routines and this one is that we want to fill regions based on both geographic and contour information.

Filled contours in a map



Code segment from colcon.f

```
      SUBROUTINE FILL (XWRK, YWRK, NWRK, IAREA, IGRP, NGRPS)
      DIMENSION XWRK(*), YWRK(*), IAREA(*), IGRP(*)

      IDMAP = -1
      IDCONT = -1
      DO 10, I=1, NGRPS
         IF (IGRP(I) .EQ. 1) IDMAP = IAREA(I)
         IF (IGRP(I) .EQ. 3) IDCONT = IAREA(I)
  10  CONTINUE
      IF (NWRK .LE. 3) RETURN
      IF ((IDMAP .GT. 0) .AND. (IDCONT .GT. 0)) THEN
          IF (MAPACI(IDMAP) .EQ. 1) THEN
             CALL GSFACI (IDCONT + 2)
             CALL GFA (NWRK-1, XWRK, YWRK)
          ELSE
             CALL GSFACI (2)
             CALL GFA (NWRK-1, XWRK, YWRK)
          ENDIF
      ENDIF
      RETURN
      END
```

Discussion

Lines 1 and 2 of the **colcon.f** code segment set up the fill routine as specified by ARSCAM. Lines 4 and 5 initialize the geographic and contour area identifier variables. Lines 6 through 9 check each group identifier: if it is a geographic group, the area identifier is stored in IDMAP; if it is a contour group, the area identifier is stored in IDCONT.

Line 10 checks to see if the fill area is defined by two or fewer points, in which case we return to ARSCAM. Line 11 checks to see if we are over the geographic map, the area to be contoured, and not over a label. Line 12 checks to see if we are over the ocean, and if we are, the area is filled according to contour level value in lines 13 and 14. If we aren't over the ocean, lines 16 and 17 fill the area with gray.

Note: This example does not use real data.

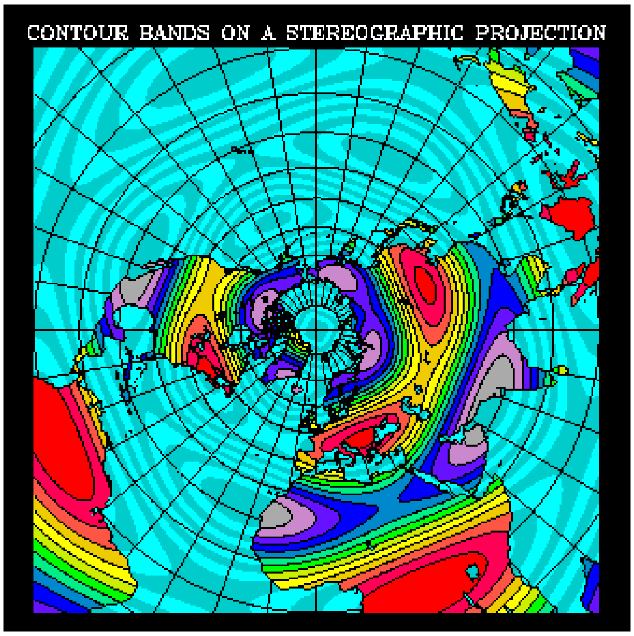
Exercises

1. Run the **colcon** example using the UNIX command **time**. Then comment out the call to MAPBLA and modify FILL so that it fills the entire plot, and time it again.
2. Modify the **colcon** example so that contour levels are filled only over land regions.

Cp 8.4 Filling areas: Polar projection

This module shows an example of how to fill contours over a polar projection of the globe. Filling contours over a polar projection requires all the same steps discussed in the previous three modules.

Contours over the north pole



Code segment from ccppole.f

```
1       EXTERNAL COLRAM
2       EXTERNAL COLRCL
3       EXTERNAL COLRLL
4       CALL ARINAM (IAMA, LAMA)
5       CALL MAPPOS (.03, .97, .01, .95)
6       CALL MAPROJ ('ST', 90., 0., 0.)
7       CALL MAPSET ('AN', 90., 90., 90., 90.)
8       CALL MAPINT
9       CALL MAPBLA (IAMA)
10      CALL CPSETI ('SET - DO-SET-CALL FLAG', 0)
11      CALL CPSETI ('MAP - MAPPING FLAG', 1)
12      CALL CPSETR ('XC1 - X COORDINATE AT INDEX 1', -180.)
13      CALL CPSETR ('XCM - X COORDINATE AT INDEX M', +180.)
14      CALL CPSETR ('YC1 - Y COORDINATE AT INDEX 1',  -90.)
15      CALL CPSETR ('YCN - Y COORDINATE AT INDEX N',  +90.)
16      CALL CPSETI ('CLS - CONTOUR LEVEL SELECTION FLAG', 1)
17      CALL CPSETR ('CMN - CONTOUR LEVEL MINIMUM', 0.)
18      CALL CPSETR ('CMX - CONTOUR LEVEL MAXIMUM', 110.)
19      CALL CPSETR ('CIS - CONTOUR INTERVAL SPECIFIER', 10.)
20      CALL CPSETR ('ORV - OUT-OF-RANGE VALUE', 1.E12)
21      CALL CPRECT (ZDAT, NLON, NLON, NLAT, RWRK, LRWK, IWRK, LIWK)
22      CALL CPCLAM (ZDAT, RWRK, IWRK, IAMA)
23      CALL ARSCAM (IAMA, XCRA, YCRA, NCRA, IAIA, IGIA, 10, COLRAM)
24      CALL MAPGRM (IAMA, XCRA, YCRA, NCRA, IAIA, IGIA, 10, COLRLL)
25      CALL MAPSTC ('OU - OUTLINE DATA FLAG', 'CO')
26      CALL MAPLOT
27      CALL CPCLDM (ZDAT, RWRK, IWRK, IAMA, COLRCL)
```

Discussion

In lines 1 through 3 of the **ccppole.f** code segment, the area filling routine, COLRAM, the contour line masking routine, COLRCL, and the grid line masking routine, COLRLL are declared to be external so that they won't be confused with variables when their names are passed to ARSCAM, CPCLDM, and MAPGRM respectively.

Line 4 initializes the area map, and lines 5 through 8 set up and initialize the geographic map. Line 9 adds the geographic outlines to the area map. Line 10 tells Conpack that the SPPS routine SET has been called, and line 11 tells Conpack that we're overlaying contours over a geographic map. Line 20 tells Conpack that values off the map projection are not to be contoured.

Lines 12 though 15 set up the limits of the contour data over the globe, and lines 16 through 19 set the contour level values. Line 21 initializes Conpack. Line 22 adds contour lines to the area map. Line 23 color-fills the map. Line 24 draws grid lines over the ocean but not land. Line 26 draws the geographic outlines over the plot, and line 27 outlines the contour lines.

Notice that the areas are filled first, and then outlines are drawn on top. This prevents your outlines from being overwritten by the area filling.

The most frequent mistake that users make with this kind of plot is to forget that the data along the right edge of their global grid must equal the data along the left edge. When this mistake is made, they will either have a gap between the meridians into which those edges project or, if they improperly map those edges into the same meridian, the contours there won't fit together. Also, if the values along the top edge all map into the North Pole, they must be equal to one another. Similarly, if the values along the bottom edge all map into the South Pole, they must all be equal.

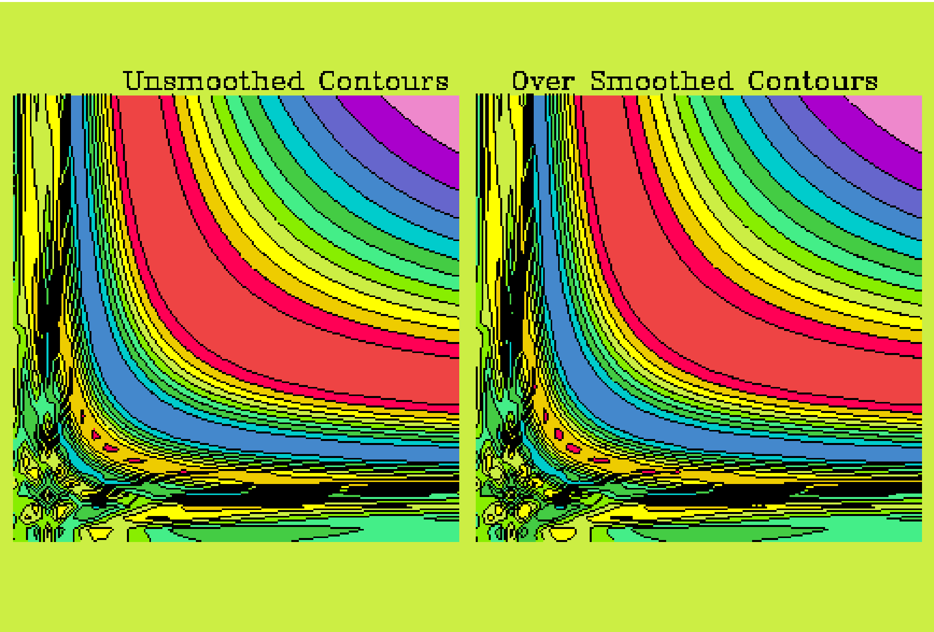
Cp 9. Advanced topics

This section contains information about frequently requested functionality that doesn't fit neatly into the preceding discussion of contouring. This section provides examples of smoothing contours, drawing hachures (tick marks on contours to indicate slope), using a cell array technique for color-filling plots with dense data, and a demonstration of how to make a movie.

Cp 9.1 Smoothing contours

Conpack supports two different methods for smoothing contour lines. By setting the Conpack smoothing parameter T2D, you can force Conpack to either use cubic splines under tension to smooth curves or interpolate between existing data points to smooth curves. You can generate closely spaced data points with an interpolation package, or you can use CPSPS1 or CPSPS2 on gridded data.

Examples of smoothing



Code segment from ccpt2d.f

```
      CALL CPSETR ('T2D - TENSION ON 2-DIMENSIONAL SPLINES', .0001)
      CALL CPRECT (ZDAT, M, M, N, RWRK, LWRK, IWRK, LWRK)
      CALL ARINAM (IAMA, LMAP)
      CALL CPCLAM (ZDAT, RWRK, IWRK, IAMA)
      CALL ARSCAM (IAMA, XCRA, YCRA, LWRK, IAREA, IGRP, 2, FILL)
```

Synopsis

```
    CALL CPSETR ('T2D', t2d)
```

```
CALL CPSETI ('PIC', ipic)
CALL CPSETR ('SSL', ssl)
```

Arguments

T2D

Real---The Tension on 2-Dimensional splines parameter controls how much smoothing is done.
0. No 2-D smoothing occurs. This is the default.
>0. T2D is used as the tension on cubic splines used for smoothing. Smoothing is done *after* mapping. This option generates more accurate contours than the default case.
<0. ABS(T2D) is used as the tension on cubic splines. In this case, smoothing is done *before* mapping requested by the MAP flag is done. This option can reduce CPU time when both smoothing and mapping are required.

PIC

Integer---the Point Interpolation flag for Contours parameter specifies the number of points to interpolate between each pair of points defining a segment of a contour line, prior to any mapping requested by the MAP parameter.
0 No points interpolated. This is the default.
>0 Normally used only if MAP<>0 and T2D=0. This maps straight line segments of contour lines more nearly correctly into curved line segments on a background. If T2D<>0, the interpolated points are used and the smoothed line will be constrained to pass through them.
<0 Normally used only if MAP<>0 and T2D=0. ABS(PIC) points are interpolated, but the interpolated points are not, in general, used to draw the line segment. This does a finer search for changes in visibility to locate more carefully which points are outside the Ezmap transformation range.

SSL

Real---The Smoothed Segment Length parameter specifies the distance between points used to draw the curves generated by 2-D smoothing; it is expressed as a fraction of the width of the window in the coordinate system in which smoothing is being done. By default, SSL=.01.

Discussion

Line 1 of the **ccpt2d.f** code segment smooths contour lines by setting the T2D parameter. Set T2D before contour lines are added to the area map (if one is used) or before contour lines are drawn. Small T2D values (like .0001) give very smooth curves, and large values like 50.0 give barely smoothed or polygonal curves.

There are two symptoms of too much smoothing using the T2D parameter. First, when low tension is set (small T2D values), it is possible to get loops in the contour line. Second, since each contour line is smoothed individually, it is possible for contour lines to overlap when using T2D. It is possible to minimize these problems by setting PIC so that points are interpolated between each data point before smoothing. This will constrain the smoothed curves to more closely approximate the original polylines.

When contour lines cross, the area identifiers get confused. This can cause colors to mix unpredictably in color-filled plots (this caused the background to be green rather than black in the **ccpt2d** example). To avoid crossed contour lines, you can use CPSPS1 or CPSPS2 to do 3-D smoothing. CPSPS1 and CPSPS2 use bicubic splines under tension and then sample the generated surface to get a dense array of data, which is returned to you for contouring. This is CPU- and memory-intensive. However, this method will not generate crossing contours or loops in contour lines.

The data in the dense array returned by CPSPS1 and CPSPS2 may have a larger range than the original data. For example, if you are measuring water temperature that is never below 0 degrees Celsius, you may discover your interpolated array has values less than 0 after calling CPSPS1 or CPSPS2. In these cases, you should either perform your own inter-polation or modify the contents of the interpolated array to fit your restrictions before contouring.

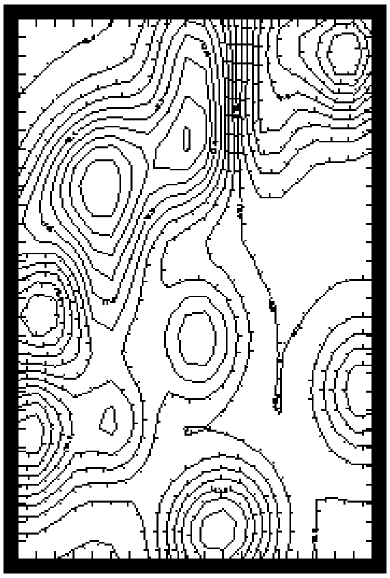
Exercises

1. Using **ccpt2d.f**, set T2D=.0005, and observe what happens.
2. Modify **ccpt2d.f** so that it smooths the data with CPSPS1. Notice how the contour lines change.

Cp 9.2 Hachuring: Indicating slope on a contour plot

Hachuring is the process of drawing "hachures" or short line segments that indicate the direction of the slope. Usually, hachures are drawn on the downslope side of a contour line and are perpendicular to it, but Conpack also allows you to draw hachures on the upslope side of the contour line.

Contours with hachures



Code segment from ccphcf.f

```
      CALL CPSETI ('HCF - HACHURE FLAG', 2)
      CALL CPSETR ('HCS - HACHURE SPACING', .05)
```

```
3       CALL CPSETR ('HCL - HACHURE LENGTH', .01)
4       CALL CPRECT (ZREG, M, M, N, RWRK, LRWK, IWRK, LIWK)
5       CALL CPBACK (ZREG, RWRK, IWRK)
6       CALL CPCLDR (ZREG, RWRK, IWRK)
```

Synopsis

```
CALL CPSETI ('HCF', ihcf)
CALL CPSETR ('HCS', hcs)
CALL CPSETR ('HCL', hcl)
```

Arguments

HCF

Integer---The HaChure Flag determines which lines are hachured as follows:

0 Hachuring is off. This is the default.
1 All contours are hachured.
2 Closed contours are hachured only if the interior of the contour is downslope. Open contours are all hachured.
3 Closed contours are hachured only if the interior of the contour is downslope. Open contours are not hachured.
4 Closed contours are hachured only if the interior of the contour is downslope. Open contours are hachured only if the interior of the contour is downslope, where "interior" is defined by computing the total change in direction along the contour. If that total is positive, the interior is to the left; if that total is negative, the interior is to the right.
<0 Each value has the same effect as ABS(HCF), except that the word "downslope" above is replaced by the word "upslope."

HCS

Real---The HaChure Spacing parameter sets hachure spacing as a fraction of the width of the viewport. This is the distance between hachures along a contour line. The default value is .01.

HCL

Real---The HaChure Length parameter sets the length of hachures as a fraction of the width of the viewport. If HCL>0, then hachures are drawn on the downslope side of the contour. If HCL<0, then hachures are drawn on the upslope side of the contour. The default value is .004.

Discussion

Line 1 of the **ccphcf.f** code segment shows that hachuring is set by setting the HCF parameter. Line 2 shows how to change hachure spacing, and line 3 demonstrates how to change the hachure length. Since these parameters cannot be applied to individual lines, it is best to call them before initializing Conpack.

Occasionally when using hachuring, it may be necessary to increase the value of RWC so that the hachuring routine works with all of each contour line. Otherwise, contours may be marked incorrectly.

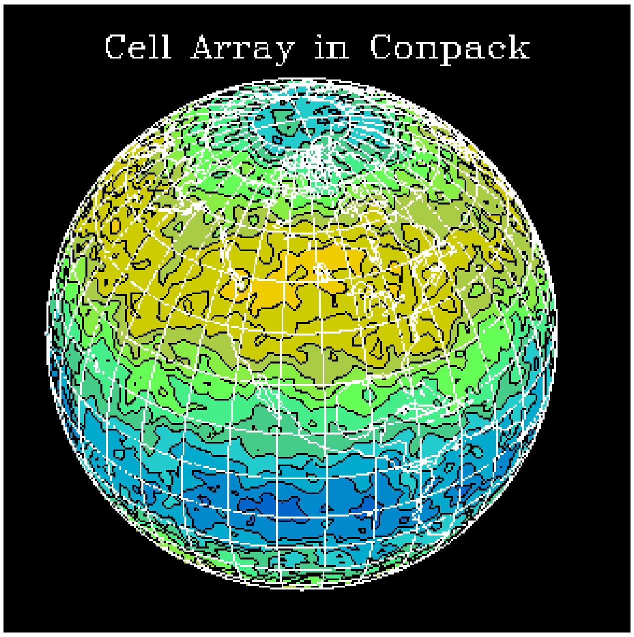
Exercises

1. Try out the other hachuring options using example **ccphcf.f**.

Cp 9.3 Cell arrays

If you have very dense data that you want to color fill, it is usually much faster to make a cell array for the plot rather than contouring it. This method has better accuracy for the "contours," and it can take significantly less CPU time to generate.

A cell array contour plot



Code segment from ccpcica.f

```
      CALL COLOR
      CALL SETMAP ('SV', 40., -105.)
      CALL SETCTR (0, 1, -180., 180., -90., 90., 1.E36)
      CALL CPRECT (ZDAT, NLON, NLON, NLAT, RWRK, LRWK, IWRK, LIWK)
      CALL CPCICA (ZDAT, RWRK, IWRK, ICRA, ICAM, ICAM, ICAN, 0., 0., 1., 1.)
      CALL GCA (CFUX(0.), CFUY(0.), CFUX(1.), CFUY(1.), ICAM, ICAN, 1, 1,
     + ICAM, ICAN, ICRA)
      CALL CPCLDR (ZDAT, RWRK, IWRK)
      CALL MAPLOT
      CALL MAPGRD
```

Synopsis

```
      CALL CPCICA (ZDAT, RWRK, IWRK, ICRA, ICA1, ICAM, ICAN, XCPF, YCPF,
     +             XCQF, YCQF)
```

Arguments

ZDAT

Real array, Workspace---The array of data points which must not have been modified since the last call to CPRECT, CPSPS1, or CPSPS2.

RWRK

Real array, Workspace---The real workspace array which must not have been modified since the last call to CPRECT, CPSPS1, or CPSPS2.

IWRK

Integer array, Workspace---The integer workspace array which must not have been modified since the last call to CPRECT, CPSPS1, or CPSPS2.

ICRA(ICA1,***n***)

Integer array, Input or Output---The cell array. On input, this array should be zeroed. On output, it contains color indices of the the cell array.

ICA1

Integer, Input---The first dimension of the Fortran array ICRA. Make ICA1 fairly large to avoid having your cells show. A good minimum value for ICA1 is 200.

ICAM

Integer, Input---The first dimension of the cell array. This is the first dimension of the portion of the array that will be used.

ICAN

Integer, Input---The second dimension of the cell array. ICAN<=*n*, where *n* is the dimension of the array declared in the DIMENSION or INTEGER statement.

XCPF, YCPF

Real, Input---The coordinates, in NDCs, of the corner of the area into which cell (1, 1) of the cell array maps.

XCQF, YCQF

Real, Input---The coordinates, in NDCs, of the corner of the area into which cell (ICAM, ICAN) of the cell array maps.

Discussion

Line 1 of the **ccpcica.f** code segment calls a user-defined subroutine to generate a color table. Line 2 calls a second user-defined subroutine where all the Ezmap options are set to draw a map. Line 3 calls another user-defined subroutine to set the necessary general contouring options.

Line 4 initializes Conpack so that line 5 can generate a cell array. Line 6 plots the cell array onto the map by using the functions CFUX and CFUY to determine the user coordinates of the corners of the plot. Line 7 outlines the contour lines using a call to CPCLDR, and line 8 outlines the map. Line 9 draws the grid lines on the map.

Exercises

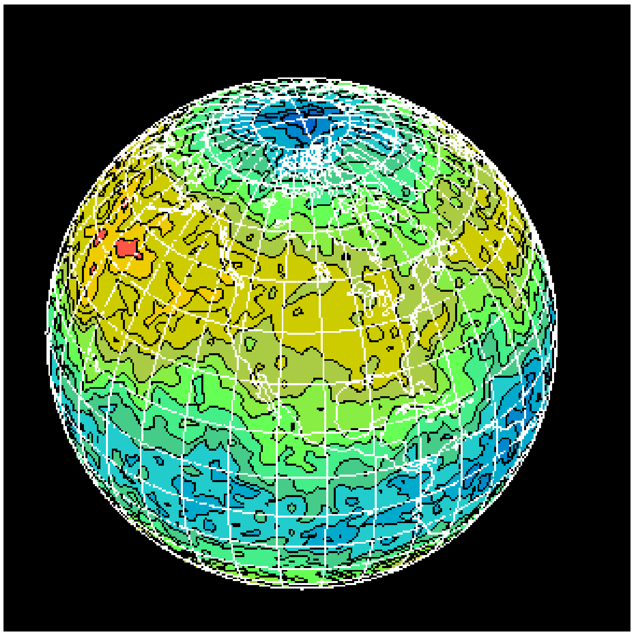
1. Draw a contour cell array without a map behind it.

Cp 9.4 Making a movie

When making movies of contours over a globe, people sometimes have trouble organizing their code to minimize CPU time. This module demonstrates how to use Conpack, Ezmap, and Areas in conjunction with the Gflash utility to make a movie. The Gflash utility allows you to create CGM segments or partial pictures, and then overlay them. The code segment from the **ccpmovi** example creates a single map image, then plots several different sets of contour data over the map.

There are four Gflash routines. GFLAS1 initiates storage of plotting instructions into a disk file. GFLAS2 terminates the storage of plotting instructions to disk and resumes putting plotting instructions to the metafile output stream. GFLAS3 inserts the plotting instructions saved on disk into the output stream. GFLAS4 allows the user to access a disk file of plotting instructions generated with a GFLAS1 and GFLAS2 sequence in a previous job for use in a GFLAS3 call.

One frame from a movie



Code segment from ccpmovi.f

```
1        CALL GOPKS (6, IDNUM)
2        CALL GOPWK (1, 2, 1)
3        CALL GACWK (1)
4        CALL GSCLIP (0)
5        CALL GSASF (IASF)
6        CALL COLOR
7        CALL GOPWK (9, 1, 3)
8        CALL GFLAS1 (0)
9        CALL DRWMAP ('SV', 40., -105.)
10       CALL GFLAS2
11       CALL SETCTR (0, 1, -180., 180., -90., 90., 1.E36)
12       DO 30, I=1, N
13           CALL MKDAT (I, ZDAT, NLON, NLAT)
14           CALL DRWCTR (ZDAT, NLON, NLAT)
15           CALL GFLAS3 (0)
16           CALL FRAME
17   30  CONTINUE
18       CALL GCLWK (9)
19       CALL GDAWK (1)
20       CALL GCLWK (1)
21       CALL GCLKS
```

Synopsis

```
CALL GFLAS1 (IDNUM)
CALL GFLAS2 (IDNUM)
CALL GFLAS3 (IDNUM)
CALL GFLAS4 (IDNUM, FNAME)
```

Arguments

IDNUM

> *Integer, Input*---A buffer identifier. This identifier must be between 0 and 99 inclusive, and is used by GFLAS3 to identify the buffer to be added to the CGM file. GFLAS1 automatically assigns the name GNFB*xx* (GKS New Flash Buffer) to the file that will receive subsequent plotting instructions, where *xx* is the integer value of IDNUM.

FNAME

> *Character expression, Input*---The name of the file in which plotting instructions are stored.

Discussion

Before using Gflash, you must open GKS. Once GKS is open, you can set GKS options (such as clipping and color tables shown in lines 4 and 6 of the **ccpmovi.f** code segment). When a segment is copied to a GKS workstation (including a CGM file), all attribute settings (except color table settings) in effect at the time of a segment's creation will be stored in the segment.

Lines 2 and 3 of the **ccpmovi.f** code segment open and activate a CGM workstation to which the movie file will be written. Before creating your first segment, you must open a workstation with workstation type 3. Line 7 opens workstation 9 to write to unit 1, using workstation type 3. You can use any workstation number and any unit number that are not already being used.

Line 8 begins writing our segment to buffer 0, and the file GNFB00 is created in the current working directory to hold the segment. Line 9 calls a user-supplied routine to draw a map of the globe, and line 10 quits writing to buffer 0.

Line 11 sets up the necessary contour options that will be the same for every plot, and line 13 creates a different dataset for each plot. The user routine DRWCTR plots these datasets in line 14. The contours here are being drawn directly to the default CGM file, **gmeta**.

Line 15 overlays the global map on the contours, and line 16 closes the frame. Line 18 is required to close the Gflash workstation, and the number 9 must correspond to the workstation identifier number set in line 7. Lines 19 and 20 deactivate and close the CGM workstation, and line 21 closes GKS.

For more complete information about Gflash, GKS workstations, and CGM buffer files, please see Chapter 6 "What you need to know about GKS workstations" in the *NCAR Graphics Fundamentals* guide.

Cp 10. Conpack parameter descriptions

The parameter descriptions follow, in alphabetical order.

AIA

> *Integer array*---Area Identifier Above. Each element of the parameter array AIA is an area identifier for the area above the contour level specified by the corresponding element of the parameter array CLV. The corresponding element of the parameter array AIB is an area identifier for the area below that level. If, for a particular contour level, both AIA and AIB are zero, that level is ignored by the CPCLAM routine; otherwise, contour lines at that level are added to the area map and the given values of AIA and AIB are used as left and right area identifiers.
> There are three special elements in the parameter array AIA, corresponding to PAI values of 1, 2, and 3:

```
----------------------------------------------
PAI  AIA
----------------------------------------------
-1   Specifies an area identifier for the area
     outside the edge of the grid.
-2   Specifies an area identifier for any area
     filled with special values.
-3   Specifies an area identifier for any area
     in which the mapping routine CPMPXY
     returns the out-of-range value ORV.
----------------------------------------------
```

If contour levels are chosen automatically, rather than being supplied by you, the value supplied for the Ith element of AIA is I+1 and the value supplied for the Ith element of AIB is I.
Default: See description above.

AIB

Integer array---Area Identifier Below. See the description of AIA.
Default: See description of AIA.

CAF

Integer---Cell Array Flag. CAF determines the way in which the routine CPCICA (which is called to store color indices in a user cell array) gets from an area identifier associated with a particular cell to a color index for that cell. Let IAID be the area identifier that is associated with a given cell; then for CAF values of:

<0 The routine CPSCAE is called. The default version of CPSCAE behaves as follows: if IAID>=0, the value of IAID is stored in the cell array as as a color index; if IAID<0, the cell array is unchanged. A user version of CPSCAE may be supplied to do something more complicated; since the value of CAF is one of the arguments of CPSCAE, its value may be used to select one of a number of color-setting schemes built into the routine.

>=0 The value CAF+IAID is computed. If that value is greater than or equal to zero, it is stored in the cell array as a color index; otherwise the cell array is unchanged. In this case, CAF acts as an offset, mapping the area identifiers to a unique portion of the color table.

Default: 0

CFA

Real---Constant Field label Angle. CFA specifies the angle (in degrees counterclockwise from a vector pointing to the right) at which a constant field label is to be written.
Default: 0.

CFB

Integer---Constant Field label Box flag.

0 Constant field label is not boxed at all. This is the default.
1 Perimeter of the box is drawn (in the same color as the label) after the label is drawn.
2 Box is filled (in the color specified by LBC) before the label is drawn.
3 Both 1 and 2 above.

CFC

Integer---Constant Field label Color index.

<0 Constant field label and the box around it (if any) are drawn in the color specified by the current text color index.
>=0 Specifies the desired color index for the label and the box. If a box is drawn around the label, it is made the same color as the label itself.

Default: -1

CFF

Integer---Constant Field Found flag. CFF may not be set by you; its retrieved value will be nonzero if and only if CPRECT, CPSPS1, or CPSPS2 detected a constant field.
Default: 0 (prior to any call to CPRECT, CPSPS1, or CPSPS2)

CFL

Real---Constant Field label Line width.
<=0. Line width is not set before drawing a box around the constant field label.
>0. Specifies the desired width, as a multiple of the normal line width.
Default: 0.

CFP

Integer---Constant Field label Positioning flag. CFP determines how the constant field label is to be positioned. There are nine possible values, each of which specifies a point of the label box that is to lie on the point defined by CFX and CFY:
-4 Lower left corner of label box.
-3 Center of bottom of label box.
-2 Lower right corner of label box.
-1 Center of left edge of label box.
0 Center of label box. This is the default.
+1 Center of right edge of label box.
+2 Upper left corner of label box.
+3 Center of top edge of label box.
+4 Upper right corner of label box.
Left, right, bottom, and top are defined from the viewpoint of someone reading the label right side up.

CFS

Real---Constant Field label Size. CFS specifies the nominal size (width) of a character in the constant field label, as a fraction of the width of the viewport. This nominal size is multiplied by CWM.
Default: .012

CFT

Character expression---Constant Field label Text string. CFT specifies the text of the constant field label, which is written when a constant data field is detected; it is a character string of 40 characters or less. The embedded string 'ZDV' will be replaced by the numeric value of the field.
If CFT is given the value ' ', (a single blank), the constant field label will not be written.
Default: 'CONSTANT FIELD - VALUE IS ZDV'

CFW

Real---Constant Field label White space width. CFW specifies the nominal width of white space to be left around the constant field label, as a fraction of the width of the viewport. This nominal width is multiplied by CWM.
Default: .005

CFX

Real---Constant Field label X coordinate. CFX specifies the X coordinate of the basepoint of the constant field label. The given value is mapped linearly onto the viewport:

<0. Beyond the left edge of the viewport.
0. Left edge of the viewport.
0.<CFX<1.
Between the left edge and the right edge of the viewport.
1. Right edge of the viewport.
>1. Beyond the right edge of the viewport.

Default: .5 (centered horizontally in the viewport)

CFY

Real---Constant Field label Y coordinate. CFY specifies the Y coordinate of the basepoint of the constant field label. The given value is mapped linearly onto the viewport:

<0. Below the bottom edge of the viewport.
0. Bottom edge of the viewport.
0.<=CFY<=1.
Between the bottom edge and the top edge of the viewport.
1. Top edge of the viewport.
>1. Above the top edge of the viewport.

Default: .5 (centered vertically in the viewport)

CIS

Real---Contour Interval Specifier. See the description of CLS. When CLS>0, CIS is used.

<=0. Conpack is to choose the contour interval. (See also the description of the parameter arrays CIT and LIT).
>0. The actual contour interval to be used; in this case, LIS may be given a nonzero value *n* to specify that every *n*th contour level should be labeled. See also the descriptions of CMN and CMX.
Default: 0.

CIT

Real array---Contour Interval Table. Each nonzero element of the ten-element parameter array CIT is a real number greater than or equal to 1. and less than 10. When Conpack picks the contour interval, it chooses a number of the form "*c* x 10*k* ", where *c* is one of the elements of CIT and *k* is an integer. The nonzero elements of CIT must be sorted in ascending order, and they must appear at the beginning of the array.
CIT and LIT are used together. Each element of CIT specifies a set of contour levels that may be used, and the corresponding element of LIT specifies which of those levels are to be labeled. A

contour at "*nc* x 10*k* " is labeled if *n* is a multiple of *l*, where *l* is the element of LIT corresponding to the element *c* of CIT. For example, if the first element of CIT is 1. and the first element of LIT is 5, then Conpack is allowed to use contour levels 1., 2., 3., 4., etc., with labels at 5., 10., 15., 20., etc. (1. x 5, 2. x 5, etc.).
The default contents of CIT and LIT are:

PAI	CIT	LIT
1	1.0	5
2	2.0	5
3	2.5	4
4	4.0	5
5	5.0	5
6	0.0	0
7	0.0	0
8	0.0	0
9	0.0	0
10	0.0	0

CIU

Real---Contour Interval Used. CIU is normally intended for output only. When the selection of the contour interval is left up to Conpack, CIU gets the value it chooses. When contour levels are completely set by you, the value of CIU should be set as well, for two reasons:

- To make the desired value appear in an information label (in place of the embedded string 'CIU')
- So that it may be used by the penalty scheme for positioning labels

CIU must be set after the contour levels are set. (As a side effect of setting element 1 of CLV, CIU is set to zero.) If you supply contour levels, but do not set the value of CIU, and the penalty scheme is used to position labels, the required contour interval is estimated. In certain situations, this can cause problems (if, for example, the same contour level appears twice in CLV, once to force lines at that level to be drawn and once to force that level to be used as the boundary for a shaded area).
Default: 0.

CLC

Integer array---Contour Line Color index. Each element of the parameter array CLC, if greater than or equal to zero, is a color index for contour lines at the level specified by the associated element of CLV. CLC values less than zero imply that the lines will be drawn in the color specified by the current polyline color index. There are three special elements in the array CLC, corresponding to PAI values of 1, 2, and 3:

PAI	CLC
-1	Specifies a color index for the edge of the grid.
-2	Specifies a color index for the edge of any area filled with special values.
-3	Specifies a color index for the edge of

```
     any area in which the mapping routine
     CPMPXY returns the out-of-range
     value ORV.
---------------------------------------------
```

Default value of each element of CLC: -1

CLD

Character array---Contour Line Dash pattern. Each element of the parameter array CLD is a dash pattern (as expected by the Dashline utility) to be used, when DPU is nonzero, to draw contour lines at the level specified by the associated element of the contour level array CLV. Elements of CLD may be set using a call to CPSETI, with a 16-bit integer as the second argument, or using a call to CPSETC, with a character string of 32 or fewer characters as the second argument. In either case, the result will be a character string internally.
There are three special elements in the parameter array CLD, corresponding to PAI values of 1, 2, and 3:

```
----------------------------------------------
PAI  CLD
----------------------------------------------
-1   Specifies a dash pattern for the edge of
     the grid.
-2   Specifies a dash pattern for the edge of
     any area filled with special values.
-3   Specifies a dash pattern for the edge of
     any area in which the mapping routine
     CPMPXY returns the out-of-range
     value ORV.
----------------------------------------------
```

When Conpack picks the contour levels, the default value supplied for each associated dash pattern is the character constant '$$$$$$$$$$$$$$$$'. This is also the default value for each of the three special elements.
Default: '$$$$$$$$$$$$$$$$'

CLL

Real array---Contour Line Line width. Each element of the parameter array CLL specifies the line width used to draw contour lines at the level specified by the associated element of the contour level array CLV. Each is expressed as a multiple of the normal line width; values less than or equal to zero imply that line width should not be set.
There are three special elements in the parameter array CLL, corresponding to PAI values of 1, 2, and 3:

```
--------------------------------------------
PAI  CLL
--------------------------------------------
-1   Specifies a line width for the edge of
     the grid.
-2   Specifies a line width for the edge of
     any area filled with special values.
-3   Specifies a line width for the edge of
     any area in which the mapping routine
```

```
     CPMPXY returns the out-of-range
     value.
--------------------------------------------
```

When Conpack picks the contour levels, the default value supplied for each associated line width is 0. This is also the default value for each of the three special elements.
Default: See description above.

CLS

Integer---Contour Level Selection flag. CLS specifies how contour levels are to be selected. Set the value of CLS before calling CPRECT, CPSPS1, or CPSPS2, as follows:

0 Conpack will not pick contour levels at all; the current values of the parameters NCL, CLV, and associated arrays will not be changed. They will thus retain the values chosen by Conpack during a previous call or the values supplied by you.
-n Conpack will generate *n* contour levels, splitting the range from the minimum field value to the maximum field value into *n*+1 equal intervals.
+n If CIS is less than or equal to zero, Conpack will use values of the form *b***k*, where *b* is a base value chosen by Conpack and *k* is an integer. The base value *b* will be a *nice value* (as defined by the contents of the parameter array CIT), chosen in such a way as to give at least *n* contour levels (with the default contents of the array CIT, you may get as many as 2*n* levels).

If CIS>0 and CMN>CMX, Conpack will use values of the form CIS**k*, where *k* is an integer.
If CIS>0 and CMN<=CMX, Conpack will use values of the form CMN+CIS**k* which are greater than or equal to CMN and less than or equal to CMX, where *k* is an integer.
Default: 16

CLU

Integer array---Contour Level Use flags. Each element of the parameter array CLU indicates how the associated contour level, in the parameter array CLV, is to be used.

0 No contour line is to be drawn at the associated level.
1 The line is to be drawn without labels.
2 The labels are to be drawn, but not the line.
3 Both the line and the labels are to be drawn.

There are three special elements in the parameter array CLU, corresponding to PAI values of 1, 2, and 3:

```
------------------------------------------
PAI  CLU
------------------------------------------
-1   Specifies a flag for the edge of the
     grid.
-2   Specifies a flag for the edge of any
     area filled with special values.
-3   Specifies a flag for the edge of any
     area in which the mapping routine
     CPMPXY returns the out-of-range
     value.
------------------------------------------
```

In each case, if the flag is zero, the associated edge is not drawn; otherwise, the associated edge is

drawn.
When Conpack chooses the contour levels, the associated elements of CLU are given one of the values 1 or 3, depending on whether the line is to be labeled or not. The default values of the special elements are all zeros.
Default: See description above.

CLV

Real array---Contour Level Values. Each of the first NCL elements of the parameter array CLV is a contour level for which something is to be done, such as the drawing of contour lines, the drawing of contour labels, and/or the addition of contour lines to an area map.
Only elements 1 through NCL may be accessed via the parameter-setting routines. Thus, code to set the contour levels and associated quantities must begin with a call to set NCL.
A side effect of setting the element numbered PAI of CLV is that the associated element number PAI in each of the parameter arrays AIA, AIB, CLC, CLD, CLL, CLU, LLC, and LLT is also given a default value. These values are as follows:

Parameter	**Default value**
AIA	PAI+1
AIB	PAI
CLC	-1.
CLD	$$$$$$$$$$$$$$$$
CLL	0.
CLU	1
LLC	-1
LLT	' ' (a single blank)

Thus, in code to set contour levels and associated quantities, each contour level must be set before the quantities associated with it.
A side effect of setting element number 1 of CLV is that the CIU parameter, which indicates what contour interval was used, is set to zero. It is assumed that this will only happen when you are providing all the contour levels, in which case the concept of the *contour interval* may or may not be well defined. See the description of CIU for more information.
Default: See description above.

CMN

Real---Contour MiNimum. When CLS>0 and CIS>0, if CMN<=CMX, then the contour levels used will be CMN, CMN+CIS, CMN+2*CIS, ..., CMN+*n**CIS, where *n* is the largest integer such that CMN+*n**CIS<=CMX. The labeled levels will be those for which *n* is a multiple of LIS.
Default: 1.

CMX

Real---Contour MaXimum. See CMN, above.
Default: 0.

CTM

Character expression---Character TeMporary. The parameter name CTM refers to a temporary character buffer in Conpack; CTM may be used in the CPCHCL routine to get the dash pattern for the current line and in the CPCHHL, CPCHIL, and CPCHLL routines to get the text of the label being written or to change it.
Default: None

CWM

Real---Character Width Multiplier. All character size parameters are multiplied by CWM; this makes it easy to scale the sizes up or down simultaneously. This affects the parameters:

```
-----------------------
 CFS  DPS  HLS  ILS  LLS
 CFW  DPV  HLW  ILW  LLW
-----------------------
```

Default: 1.

DPS

Real---Dash Pattern Size. DPS specifies the nominal size (width) of a character in a dash pattern, as a fraction of the width of the viewport. This nominal size is multiplied by CWM.
Default: .010

DPU

Integer---Dash Pattern Use flag.

<=0 No dash patterns are used. Contour lines are drawn using calls to CURVE.

***n*>0**

Dash patterns are used. Contour lines are drawn using calls to CURVED. When ABS(LLP)=1, contour lines are caused to be labeled by using a dash pattern formed by concatenating *n* repetitions of the appropriate element of CLD (the nominal dash pattern for the line) and the appropriate element of LLT (the numeric label for the line); in this case, DPU specifies the value of *n*.

Default: 3

DPV

Real---Dash Pattern Vector length. DPV specifies the nominal length of the solid vector generated by a dollar sign, or the gap vector generated by an apostrophe, in a dash pattern, as a fraction of the width of the viewport. This nominal length is multiplied by CWM.
Default: .005

GIC

Integer---Group Identifier for Contour lines. GIC specifies the group identifier for contour lines added to an area map by the CPCLAM routine.
Default: 3

GIL

Integer---Group Identifier for Label boxes. GIL specifies the group identifier for label boxes added to an area map by the CPLBAM routine.
Default: 3

GIS

Integer---Group Identifier for vertical Strips. GIS specifies the group identifier for a group of edges added to an area map by the CPCLAM routine in order to create a set of vertical strips. This is done only if the NVS parameter is nonzero. See NVS, below.
Default: 4

HCF

Integer---Hachuring Flag. HCF specifies the type of hachuring to be used, as follows:

0 Hachuring is turned off. This is the default.
1 All contours will be hachured.
2 Closed contours will be hachured only if the interior of the contour is "downhill." Open contours will be unconditionally hachured.
3 Closed contours will be hachured only if the interior of the contour is "downhill." No open contours will be hachured.
4 Closed contours will be hachured only if the interior of the contour is "downhill." Open contours will be hachured only if the "interior" of the contour is "downhill," where "interior" is defined by computing the total change in direction along the contour. If that total is positive, the "interior" is to the left; if it is negative, the "interior" is to the right.
-1 All contours will be hachured.
-2 Closed contours will be hachured only if the interior of the contour is "uphill." Open contours will be unconditionally hachured.
-3 Closed contours will be hachured only if the interior of the contour is "uphill." No open contours will be hachured.
-4 Closed contours will be hachured only if the interior of the contour is "uphill." Open contours will be hachured only if the "interior" of the contour is "uphill," where "interior" is defined by computing the total change in direction along the contour. If that total is positive, the "interior" is to the left; if it is negative, the "interior" is to the right.

The user is expected to increase the value of the RWC parameter from its default 100 to a value large enough to accommodate any contour that can arise from the data field being used; this may require increasing the size of the real workspace array. (This is so that the hachuring routine is assured of seeing all of each contour line. Otherwise, it can’t tell which way the "inside" and the "outside" of the closed contours are.)
Closed contours that are divided into pieces that appear to be open contours will be hachured. Closed contours may appear to be open contours because portions of the closed contour disappear as a result of the mapping implied by nonzero values of the Conpack parameter MAP and the out-of-range value ORV, or because they run through special-value areas, as defined by the parameter SPV, or because the user has forgotten to increase the value of RWC, as described in the preceding paragraph. The rationale for this is that no real confusion can arise from hachuring a contour line that shouldn’t be, but confusion may result from not hachuring a contour line that should be.

HCL

Real---HaChure Length. HCL specifies the hachure length, stated as a fraction of the width of the viewport. A positive value implies the use of hachures on the downslope side of the contour. A negative value implies the use of hachures on the upslope side of the contour.
Default: .004

HCS

Real---HaChure Spacing. This is the distance between one hachure and the next along a contour line. HCS specifies the hachure spacing as a fraction of the width of the viewport. The value of HCS is forced to lie in the range .0001 to 10. Values outside this range will be interpreted as the nearest value within the range.
Default: .01

HIC

Integer---HIgh label Color index. HIC is used in determining the color index for high labels. See the description of HLC, below.
Default: -1

HIT

Character expression---HIgh label Text string. HIT specifies the text string to be used in labeling a high. See the description of HLT, below.
Default: 'H:B:ZDV:E'

HLA

Real---High/Low label Angle. HLA specifies the angle (in degrees counterclockwise from a vector pointing to the right) at which high and low labels are to be written.
Default: 0.

HLB

Integer---High/Low label Box flag.
 0 High and low labels are not boxed at all.
 1 Perimeter of the box is drawn (in the same color as the label) after the label is drawn.
 2 Box is filled (in the color specified by LBC) before the label is drawn.
 3 Both 1 and 2 above.
Default: 0

HLC

Integer---High/Low label Color index. HLC is used in determining the color index for high and low labels.
The color index for high labels is determined in this manner:

If HIC>=0, HIC is used as the color index.
If HIC<0, but HLC>=0, HLC is used as the color index.
If both HIC and HLC are less than zero, the current text color index is used.

The color index for low labels is determined similarly:

If LOC>=0, LOC is used as the color index.
If LOC<0, but HLC>=0, HLC is used as the color index.
If both LOC and HLC are less than zero, the current text color index is used.

If a box is drawn around a label, it is made the same color as the label.

To set the color index of all high and low labels, simply supply the desired value for HLC. To have high and low labels that are colored differently, set HIC and LOC. The default values of HLC, HIC, and LOC are all 1.

Default: -1

HLL

Real---High/Low label Line width.

<=0. Line width is not set before drawing boxes around high and low labels.
>0. The desired width, as a multiple of the normal line width.

Default: 0.

HLO

Integer---High/Low label Overlap flag. The value of HLO determines what is to be done about the problem of high and low labels overlapping other objects. Add one or more of the following values and use the resulting sum to obtain the combination of effects you want:

0 The problem will be ignored. High and low labels will not be checked for overlap with anything else.
1 Omit high and low labels that overlap the information label.
2 Omit high and low labels that overlap other high and low labels found before them.
4 Omit high and low labels that overlap the edges of the viewport.
8 High and low labels that overlap the edges of the viewport should be moved inward by just enough to take care of the problem.

The action implied by the value 4 takes precedence over that implied by the value 8; if you add both 4 and 8, the effect will be as if you had added 4 alone.

Default: 3 (the sum of 1 and 2)

HLS

Real---High/Low label Size. HLS specifies the nominal size (width) of a character in a high or low label, as a fraction of the width of the viewport. This nominal size is multiplied by CWM.

Default: .012

HLT

Character expression---High/Low label Text strings. You can specify the character strings used to label highs and lows individually, by setting HIT and LOT, or together, by setting HLT. If HLT is set, and there are no apostrophes in the given character string, then both HIT and LOT will be set equal to HLT and it will therefore be used as the label for both highs and lows. If there are

apostrophes in the string, what precedes the first apostrophe will be used as the value of HIT (the label for a high) and what follows it will be used as the value of LOT (the label for a low). Remember that, in Fortran, an apostrophe in a string that is delimited by apostrophes is represented by two apostrophes.
The substring 'ZDV' may be used to represent the numeric value of the high or the low, divided by the current scale factor. The substring '$ZDVU$' may be used to represent the unscaled value.
Examples:

Fortran string	**High label**	**Low label**
`'H''L'`	H	L
`'HI''LO'`	HI	LO
`'$ZDV$'`	1.362	.764
`'H ($ZDV$)''L ($ZDV$)'`	H (1.362)	L (.764)
`'H:B:$ZDV$:E:''L:B:$ZDV$:E:'`	H*1.362*	L.*764*

Note that, in the final example above, the subscripting capability of the Plotchar utility is used. The terminating function code E ensures that the H or the L will be centered on the high or low. To center the whole thing, either remove the Es or change them to Ns.
Neither of the character strings HIT and LOT may contain more than 20 characters. If HIT is blank, highs will not be labeled. If LOT is blank, lows will not be labeled.
There is no default value for HLT. The default value for HIT is 'H:B:ZDV:E:' and the default value of LOT is 'L:B:ZDV:E:', as shown in the final example above.
Default: None

HLW

Real---High/Low label White space width. HLW specifies the nominal width of white space to be left around a high or low label, as a fraction of the width of the viewport. This nominal width is multiplied by CWM.
Default: .005

HLX

Integer---High/Low search radius in X.

- **>0** Specifies the half-width of the index-value neighborhood used in searching the contour field for highs and lows. The "neighborhood" is the area around the index value. See the example below.
- **<=0** Conpack picks a reasonable value to use (approximately 1/8 of ZDM, but not less than 2 nor greater than 15).

For example: If HLX=3 and HLY=4, then the values in ZDAT examined to determine if (I,J) is a high or a low are those having indices (K,L), where either K<>I, or L<>J, K is between MAX(1,I3) and MIN(ZDM,I+3), inclusive, and L is between MAX(1,J4) and MIN(ZDN,J+4), inclusive.
Default: 0

HLY

Integer---High/Low search radius in Y.

- **>0** Specifies the half-height of the index-value neighborhood used in searching the contour field for highs and lows. The "neighborhood" is the area around the index value.
- **<=0** Conpack picks a reasonable value to use (approximately 1/8 of ZDN, but not less than 2 nor greater than 15).

For an example, see the description of HLX, above.
Default: 0

ILA

Real---Information Label Angle. ILA specifies the angle (in degrees counterclockwise from a vector pointing to the right) at which the information label is written.
Default: 0.

ILB

Integer---Information Label Box flag.

- **0** Information label is not boxed at all.
- **1** Perimeter of the box is drawn (in the same color as the label) after the label is drawn.
- **2** Box is filled (in the color specified by LBC) before the label is drawn.
- **3** Both 1 and 2 above.

Default: 0

ILC

Integer---Information Label Color index.

- **<0** The information label and the box around it (if any), are drawn in the color specified by the current text color index.
- **>=0** Specifies the desired color index for the label and the box. If a box is drawn around the label, it is made the same color as the label itself.

Default: -1

ILL

Real---Information Label Line width.

- **<=0.** Line width is not set before drawing a box around the information label.
- **>0.** Specifies the desired width, as a multiple of the "normal" line width.

Default: 0.

ILP

Integer---Information Label Positioning flag. ILP determines how the information label is to be positioned. There are nine possible values, each of which specifies a point of the label box that is to lie on the point defined by ILX and ILY, as shown below:

- **-4** Lower left corner of label box.
- **-3** Center of bottom of label box.
- **-2** Lower right corner of label box.
- **-1** Center of left edge of label box.

0 Center of label box.
+1 Center of right edge of label box.
+2 Upper left corner of label box.
+3 Center of top edge of label box.
+4 Upper right corner of label box.

Left, right, bottom, and top are defined from the viewpoint of someone reading the label right side up.
Default: 4 (the upper right corner of the label box will be placed on the point (ILX, ILY).

ILS

Real---Information Label Size. ILS specifies the nominal size (width) of a character in the information label, as a fraction of the width of the viewport. This nominal size is multiplied by CWM.
Default: .012

ILT

Character expression---Information Label Text string. ILT is a string of <=100 characters that specifies the text of the information label. The following substrings will be replaced by a numeric value:

```
--------------------------------------
$CIU$  Contour interval used
$CMN$  Minimum contour level
$CMX$  Maximum contour level
$SFU$  Current scale factor
$ZMN$  Minimum value in the data array
$ZMX$  Maximum value in the data array
--------------------------------------
```

In each case except SFU, the given value will have been divided by the current scale factor. To request the use of an unscaled value, you can insert "U" just before the final "$" (examples: "$CIUU$" or "$CMNU$").
The value that replaces CIU is only correct if Conpack itself has chosen the contour levels; otherwise, it may be necessary for you to set the value of CIU, which is described above.
If ILT is given the value ' ' (a single blank), no information label will be printed.
Default: 'CONTOUR FROM CMNTO CMX BY CIU'

ILW

Real---Information Label White space width. ILW specifies the nominal width of white space to be left around the information label, as a fraction of the width of the viewport. This nominal width is multiplied by CWM.
Default: .005

ILX

Real---Information Label X coordinate. ILX specifies the X coordinate of the basepoint of the information label. The given value is mapped linearly onto the viewport.

<0. Outside the viewport on the left side.
0. Left edge of the viewport.
0.<ILX<1.
Between the left and right edges of the viewport.
1. Right edge of the viewport.
>1. Beyond the right edge of the viewport.
Default: .98

ILY

Real---Information Label Y coordinate. ILY specifies the Y coordinate of the basepoint of the information label. The given value is mapped linearly onto the viewport.
<0. Outside the viewport on the bottom edge.
0. Bottom edge of the viewport.
0.<ILY<1.
Between the bottom and top edges of the viewport.
1. Top edge of the viewport.
>1. Beyond the top edge of the viewport.
Default: -.02

IWM

Integer---Integer Workspace for Masking. IWM specifies the amount of integer workspace to be allotted for use by CPCLDM, which draws contour lines masked by an area map, in calls to the Areas routine ARDRLN. Assume a parameter value *n*; the space used will be 2*n* (*n* for the array IAI and *n* for the array IAG, in calls to ARDRLN). The value *n* must be greater than or equal to the number of group identifiers used in generating the area map.
Default: 10

IWU

Integer---Integer Workspace Usage. For retrieval only, it is set to zero by the call to CPRECT, CPSPS1, or CPSPS2. As Conpack routines are called, the value of IWU is updated to reflect the largest number of words of integer workspace needed at any one time. Therefore, by retrieving its value after an entire plot has been constructed, you can find out how large an integer workspace was actually required.
Default: None

LBC

Integer---Label Box Color index. If label boxes (of whatever type) are filled, the filling is done using the color index specified by LBC.
<0 The current fill area color index is used.
>=**0** Specifies the color index to be used.
Default: 0 (the background color)

LBX

Real---Label Box X coordinate. Not to be set by the user. The value may be retrieved in one of the routines CPCHCF, CPCHHL, CPCHIL, or CPCHLL. LBX specifies the X coordinate (in the current world coordinate system) of the center of the box surrounding the label that has caused the routine to be called.
Default: 0.

LBY

Real---Label Box Y coordinate. Not to be set by the user. The value may be retrieved in one of the routines CPCHCF, CPCHHL, CPCHIL, or CPCHLL. LBY specifies the Y coordinate (in the current world coordinate system) of the center of the box surrounding the label that has caused the routine to be called.
Default: 0.

LIS

Integer---Label Interval Specifier. When you give CLS a positive value, indicating that Conpack is to choose contour levels at intervals of the form *b*k* , where *b* is a base value and *k* is an integer, and you give CIS a positive value (indicating that CIS is the desired value of *b*), then you must set LIS to specify the interval between labeled contour levels. See the descriptions of the CLS and CIS parameters.
For example: If you specify CLS=1, CIS=1/3, and LIS=3, you will get contours at values like 1/3, 2/3, 3/3, 4/3..., with labels at values like 1, 2, 3, 4....
Default: 5

LIT

Integer array---Label Interval Table. See the description of the CIT parameter.
Default: See CIT

LIU

Integer---Label Interval Used. LIU is for retrieval only. When Conpack chooses the contour interval and decides that every *n*th one should be labeled, it sets LIU to *n*.
Default: None

LLA

Real---Line Label Angle. LLA specifies the angle (in degrees counterclockwise from a vector pointing to the right) at which contour line labels are to be written when ABS(LLP)>=2 and LLO=0.
Default: 0.

LLB

Integer---Line Label Box flag.
- **0** Contour line labels drawn by CPLBDR are not boxed at all.
- **1** Perimeter of the box is drawn (in the same color as the label) after the label is drawn.

2 Box is filled (in the color specified by LBC) before the label is drawn.
3 Both 1 and 2 above.
Default: 0

LLC

Integer array---Line Label Color index. An array of color indices for labels associated with the contour levels specified by CLV.

>=0 The color index for labels on contour lines at the level specified by the associated element of CLV.

<0 The current text color index is to be used.

This parameter only affects line labels when ABS(LLP)=2 or 3 and the labels are therefore drawn by CPLBDR. It does not affect line labels when ABS(LLP)=1 and the line labels are therefore drawn by the Dashline utility, as called by CPCLDM or CPCLDR.
Default: -1 for all elements of LLC

LLL

Real---Line Label Line width.

<=0. Line width is not set before drawing boxes around contour line labels.

0. The desired width, as a multiple of the normal line width.

Default: 0.

LLO

Integer---Line Label Orientation. LLO only has effect when ABS(LLP)>=2, specifying the use of either the regular scheme or the penalty scheme for positioning labels on contour lines.

0 All labels are written at the angle specified by LLA.

<>0 Each label is written in the *local direction* of the contour line. The *local direction* is the direction of the tangent to the contour line at the label position.

Default: 0

LLP

Integer---Line Label Positioning. LLP specifies whether or not labels should be produced, and if so, how:

0 No line labels should be produced.

1 or **-1**
Labels should be positioned along contour lines by using the old CONREC scheme of setting up a character dash pattern including the label and using the software dash utility to draw the labels (which requires having DPU<>0). This scheme has the disadvantages that you cannot control the orientation or shield the labels from having contour lines drawn through them.

2 or **-2**
Labels should be positioned at regular intervals along the line. See the descriptions of the RC1, RC2, and RC3 parameters.

3 or **-3**
Labels should be positioned by using a penalty scheme, which can give much better results than either of the other values. See the descriptions of the parameters with names in the form

of PC*n* and PW*n*.
When LLP=2 or 3, the 2D smoothing, if any, implied by the value of T2D is suspended during label positioning, so that fewer label positions will be considered. This is quite a bit faster and the results are nearly as good as if the smoothing were done. To force smoothing, use LLP=2 or 3.
Default: 1

LLS

Real---Line Label Size. LLS specifies the nominal size (width) of a character in a contour line label, as a fraction of the width of the viewport. This nominal size is multiplied by CWM.
Default: .010

LLT

Character array---Line Label Text string. For each I from 1 to NCL, element I of the parameter array LLT is a string of 20 or fewer characters, to be used as a label for the contour level specified by the Ith element of CLV. Since the character string will be plotted using the Plotchar routine PLCHHQ, it can contain colon-enclosed function codes to do things like create superscripts.
If you do not supply the elements of this array, the elements will be filled in by Conpack.
Default: None

LLW

Real---Line Label White space. LLW specifies the nominal width of white space to be left around a contour line label, as a fraction of the width of the viewport. This nominal width is multiplied by CWM.
Default: .005

LOC

Integer---LOw label Color index. LOC is used in determining the color index for low labels. See the description of the HLC parameter.
Default: -1

LOT

Character expression---LOw label Text string. LOT specifies the text string to be used in labeling a low. See the HLT parameter.
Default: 'L:B:ZDV:E:'

MAP

Integer---MAPping flag.
0 Specifies that the X and Y coordinates used to create the contour map are not to be transformed by the user-replaceable CPMPXY routine.
<>**0** Specifies that X and Y coordinates are to be transformed by the user-replaceable CPMPXY routine. Two nonzero values for MAP provide useful mappings with the default version of CPMPXY:

1 If the first subscript of the data array is a linear function of the longitude and the second is a linear function of the latitude, then you can transform all graphics output onto a map background created by calls to Ezmap routines just by setting MAP=1, XC1 = minimum longitude, XCM = maximum longitude, YC1 = minimum latitude, and YCN = maximum latitude. Also, the Conpack parameter SET must be given the value 0 to prevent Conpack from calling SET and thereby overriding the call done by Ezmap.
2 If the first subscript of the data array is a linear function of rho and the second is a linear function of theta, where rho and theta are polar coordinates, then to map all graphics output properly, you can set MAP=2, XC1 = minimum rho, XCM = maximum rho, YC1 = minimum theta, and YCN = maximum theta. In this case, you must either use SET=0 and do an appropriate SET call or use SET=1 and give the parameters WDB, WDL, WDR, and WDT values consistent with the mapped values of X and Y, which will all be of the form "rho*cos(theta)" and "rho*sin(theta)", respectively.

Using any other nonzero value of MAP will result in the identity mapping (X→X, Y→Y, a→a, 1→1, and so on.)
You can supply your own version of the CPMPXY routine and build as many different mappings into it as desired. See the description of CPMPXY.
Default value of MAP: 0

NCL

Integer---Number of Contour Levels. If Conpack is to pick contour levels (see the description of the CLS parameter), then the initial call to CPRECT, CPSPS1, or CPSPS2 causes NCL to be set to zero. Subsequently, during the first call to a Conpack routine requiring contour levels to have been chosen, NCL will be set as part of the process of choosing them. If you choose the contour levels, the first parameter that you must set is NCL.
Default: None

NEL

Integer---Numeric Exponent Length.
<=0 Exponents in numeric labels should be written in the shortest possible form; plus signs are omitted and the exponent magnitude is written with no leading zeros.
***n*>0**
Exponents should be written with a sign (+ or -) and the exponent magnitude should be padded with leading zeros to a length of *n* characters.
Default: 0

NET

Integer---Numeric Exponent Type. NET determines what characters are to be used between the mantissa of a numeric label and the exponent.
0 Specifies the use of an E, as in Fortran "E format."
1 Specifies the use of function codes, as expected by the Plotchar utility, to generate "x 10*n*", where *n* is a superscript exponent.
2 Specifies the use of "x10**".
Default: 1

NEU

Integer---Numeric Exponent Use flag.

<=0 Forces the use of the exponential form in all numeric labels.

***n*>0**

Specifies use of the form without an exponent as long as it requires no more than *n* characters; otherwise the form requiring the fewest characters should be used.

NLS

Integer---Numeric Leftmost Significant digit flag.

0 Leftmost nonzero digit of a number represented by a numeric label is to be considered its first significant digit.

<>0 Digit in the same digit position as the leftmost nonzero digit of the largest number (in absolute value) in the data field being contoured is to be considered the leftmost significant digit.

Using NLS<>0 tends to make the numeric labels more nearly consistent with one another.
For example: Assume the use of three significant digits.
Using NLS=0:

```
--------------------------------------
 .500  1.00  1.50  ...  9.50  10.5  ...
--------------------------------------
```

Using NLS=1:

```
---------------------------------
 .5  1.0  1.5  ...  9.5  10.5  ...
---------------------------------
```

Default: 1

NLZ

Integer---Numeric Leading Zero flag.

<>0 A zero is placed before any numeric label that would otherwise begin with a decimal point (0.345 rather than .345).

0 No zero is added to numeric labels.

Default: 0

NOF

Integer---Numeric Omission Flags. Specifies what parts of a numeric label may be omitted. Add one or more of the following values and use the resulting sum to obtain the combination of effects you want:

0 Specifies that no part may be omitted.

1 Omit trailing zeros after a decimal point (as in 46.200).

2 Omit a trailing decimal point (as in "13.").

4 Omit an unnecessary leading "1" or "1." (as in "1 x 10^{13}").

Contour line labels generated by Conpack and values in the information label that are known to

have been rounded to nice values (like 'CIU', 'CMN', and 'CMX') will have trailing zeros trimmed in any case, regardless of the value of NOF.
Default: 6 (the sum of 4 and 2)

NSD

Integer---Number of Significant Digits. NSD specifies the maximum number of significant digits used in numeric labels representing contour field values.

***n*<0**
Specifies that ABS(*n*) significant digits should be used.

***n*>=0**
Specifies that *m*+*n* digits should be used, where *m* is the number of digits that are the same for all values in the contour field. For example, if the minimum value is 1123.6 and the maximum value is 1125.9, then the value of *m* is 3.

Default: 4

NVS

Integer---Number of Vertical Strips. When NVS is nonzero, an extra group of edges, with group identifier GIS, is added to the area map by the CPCLAM routine. These edges include the boundary of the viewport and enough vertical lines to break the area occupied by the viewport up into NVS vertical strips. The object of this is to break up the contour bands that are to be filled into smaller and simpler pieces; this may be necessary if the graphics device in use limits the number of points that may be used to define a polygon to be filled. The area identifier for the outside of the viewport is 1; all other area identifiers used are 0s.
Default: 1

ORV

Real---Out-of-Range Value.

0. No out-of-range value specified.

<>0. Specifies an out-of-range value, to be used as the value of X and Y coordinates returned by the mapping routine CPMPXY to say that a point is out-of-range (invisible) under the current mapping.

Default: 0.

PAI

Integer---Parameter Array Index. The value of PAI must be set, before accessing any Conpack parameter that is an array, to indicate which element of the array is meant. (The Conpack parameter access routines have the form CPGET*x* and CPSET*x*.) For example, to set the tenth contour level to zero, use code like this:

```
CALL CPSETI ('PAI', 10)
CALL CPSETR ('CLV', 0.)
```

PC1

Real---Penalty scheme Constant 1. PC1 is the "largest gradient" constant used in the penalty

scheme for positioning labels. The largest gradient allowed at the position of a label will be:
GRAV + PC1 * GRSD
where GRAV is the average gradient and GRSD is the standard deviation of the gradients.
Default: 1.

PC2

Real---Penalty scheme Constant 2. PC2 is the "crossing contours" constant used in the penalty scheme for positioning labels. It specifies the maximum (estimated) number of contour bands allowed to cross a label.
Default: 5.

PC3

Real---Penalty scheme Constant 3. PC3 is the "tight curves" constant used in the penalty scheme for positioning labels. It specifies, in degrees, the maximum cumulative change in direction to be allowed along that portion of the contour line covered by a circle centered on a label and having a radius equal to half the width of the label.
Default: 60.

PC4

Real---Penalty scheme Constant 4. PC4 is the "optimum distance" constant used in the penalty scheme for positioning labels. It specifies the optimal distance in the term in the penalty function that attempts to force labels to be at an optimal distance from each other.
Default: .05

PC5

Real---Penalty scheme Constant 5. PC5 is the "folding distance" constant used in the penalty scheme for positioning labels, in the term that attempts to force labels to be at an optimal distance from each other.
Default: .15

PC6

Real---Penalty scheme Constant 6. PC6 is the "minimum distance" constant used in the penalty scheme for positioning labels. It specifies the minimum distance to be allowed between any two labels on the same contour line, as a fraction of the width of the viewport.
Default: .30

PIC

Integer---Point Interpolation flag for Contours. PIC specifies the number of points to interpolate between each pair of points defining a segment of a contour line, prior to any mapping implied by the Conpack parameter MAP.

- **0** No points are interpolated.
- **<>0** Should normally be used only if MAP<>0, which turns mapping on, and T2D=0, which

turns the 2D smoother off. The intent is to map straight-line segments of contour lines more nearly correctly into curved-line segments on a background (one drawn by Ezmap, for example). If the 2D smoother is turned on, the additional points will be used and the smoothed curve will be constrained to pass through them; this may be useful.
<0 Causes ABS(PIC) points to be interpolated, but the interpolated points are not, in general, used to draw the line segment. The object, in this case, is simply to do a finer search for changes in visibility (out-of-range state, as defined by values of ORV returned by the CPMPXY routine) along the segment.
Default: 0

PIE

Integer---Point Interpolation flag for Edges. PIE specifies the number of points to interpolate between each pair of points defining a segment of an edge (the edge of the grid, the edge of an out-of-range area, or the edge of a special-value area).
0 No points are interpolated.
<>0 Should normally be used only if MAP<>0, which turns mapping on. The intent is to map straight-line segments of edge lines more nearly correctly into curved-line segments on a background (one drawn by the Ezmap utility, for example).
<0 Causes ABS(PIE) points to be interpolated, but the interpolated points are not, in general, used to draw the line segment. The object, in this case, is simply to do a finer search for changes in out-of-range state (visibility) along the segment. (The edges of out-of-range areas, however, are drawn using all such interpolated points.)
Using too large a value of PIE, either positive or negative, will cause the tracing of the edges of out-of-range areas to be very time-consuming, because the number of points to be examined is ZDM*ZDN*PIE*PIE.
Default: 0

PW1

Real---Penalty scheme Weight 1. PW1 specifies the weight for the "gradient" term in the penalty function.
Default: 2.

PW2

Real---Penalty scheme Weight 2. PW2 specifies the weight for the "crossing contours" term in the penalty function.
Default: 0.

PW3

Real---Penalty scheme Weight 3. PW3 specifies the weight for the "tight curves" term in the penalty function.
Default: 1.

PW4

Real---Penalty scheme Weight 4. PW4 specifies the weight for the "optimum distance" term in the penalty function.
Default: 1.

RC1

Real---Regular scheme Constant 1. RC1 specifies the desired distance from the beginning of a contour line to the first label on that line when they are positioned using the regular scheme. The *n*th label on each labeled contour line will be at a distance RC1+RC2*(*n*1)+RC3*Rn units (in the fractional coordinate system) from the beginning of the line, where "Rn" is a random number between 1 and 1.
Default: .25

RC2

Real---Regular scheme Constant 2. RC2 specifies the desired nominal distance between labels when they are positioned using the regular scheme. See the description of RC1, above.
Default: .25

RC3

Real---Regular scheme Constant 3. RC3 specifies the desired maximum variation in the distance between labels when they are positioned using the regular scheme. See the description of RC1, above.
Default: .05

RWC

Integer---Real Workspace for Contours. RWC specifies the amount of real workspace to be allotted to hold coordinates of points defining contour lines and associated quantities. Assume a parameter value *n*. If no 2D smoothing is requested, the total space used will be 2*n* (*n* for X coordinates and another *n* for Y coordinates). If 2D smoothing is requested, the total space used will be 7*n* (*n* for X coordinates, *n* for Y coordinates, and 5*n* for scratch arrays).
Normally, the value of RWC is of no particular interest to you, since the same contour lines are produced with a small value as would be produced with a larger value. There are two situations in which it becomes of more interest: 1) When the penalty scheme is used to position labels, the length of the portion of the contour line over which the penalty function is evaluated is limited by the value of RWC. If RWC is set too small, too many labels may be put on a given contour line and some of them may be too close to each other. 2) When hachuring has been activated (HCF<>0), it is important that the routine that does the hachuring see entire contours at once, so that it may properly decide whether a contour is open or closed, and, in the latter case, where the interior of the closed contour is.
In both of these situations, the solution is to increase the value of RWC.
Default: 100

RWG

Integer---Real Workspace for Gradients. RWG specifies the amount of real workspace to be

allotted to hold gradients that are to be computed and used in positioning labels using the penalty scheme. Using a larger value provides for a more accurate representation of the gradient field, up to the point at which it exceeds ZDM*ZDN.
Default: 1000

RWM

Integer---Real Workspace for Masking. RWM specifies the amount of real workspace to be allotted for use by CPCLDM, which draws contour lines masked by an area map, in calls to the Areas routine ARDRLN. Assume a parameter value *n*: the space used will be 2*n* (*n* for the Xcoordinate array XCS and *n* for the Ycoordinate array YCS, in calls to ARDRLN). Any value of *n* greater than or equal to 2 will work; smaller values will cause the generation of more calls to the user subroutine RTPL (one of the arguments of CPCLDM).
Default: 100

RWU

Integer---Real Workspace Usage. For retrieval only, RWU is set to zero by the call to CPRECT, CPSPS1, or CPSPS2. Thereafter, as Conpack routines are called, the value of RWU is updated to reflect the largest number of words of real workspace needed at any one time. Therefore, by retrieving its value after an entire plot has been constructed, you can find out how large a real workspace was actually required.
Default: None

SET

Integer---do-SET-call flag.

0 No SET call is to be done by Conpack.

1 SET call is to be done by CPRECT, CPSPS1, or CPSPS2.

Arguments 5 through 8 of a SET call done by the user must be consistent with the ranges of the X and Y coordinates being used by Conpack, as specified by the values of the parameters XC1, XCM, YC1, YCN, and MAP. See the descriptions of those parameters
Default: 1

SFS

Real---Scale Factor Selector. The scale factor is that value (usually, but not necessarily, a power of 10) by which the actual values of contour field values are to be divided to get the value of a numeric label.

>0. The value selected is the scale factor used.

If SFS is given a value less than or equal to zero, the value is truncated to form an integer directing Conpack to select a scale factor, as follows:

0. Scale factor should be selected in such a way as to reduce the ZDAT element having the largest absolute value to the range from .1 to .999.

-1. Scale factor should be selected in such a way as to reduce the ZDAT element having the largest absolute value to the range from 1. to 9.999.

-2. Scale factor should be selected in such a way as to place the decimal point in the ZDAT element having the largest absolute value after the rightmost significant digit of that value

(as defined by the values of NSD and NLS).
-3. Scale factor should be selected in such a way as to remove extra zeros from the ends of the ZDAT element having the largest absolute value. For example, if that element were .000136, the scale factor would be 10 to the power 3; if that element were 136000 (assuming three significant digits are desired), the scale factor would be 10 to the power 3. If there are no extra zeros on either end of the ZDAT element having the largest absolute value, the scale factor will be 1.
<=-4.
Scale factor should be selected in such a way as to reduce all contour labels to integers.
Default: 1.

SFU

Real---Scale Factor Used. For retrieval only, SFU gives the value of the scale factor selected for use by Conpack.
Default: None

SPV

Real---SPecial Value.
0. No special value is used.
<>0. Specifies a special value, which may be used in data fields to signal missing data. No contour lines will be drawn within any grid cell with a special value at one or more of its four corners.
Default: 0.

SSL

Real---Smoothed Segment Length. SSL specifies the distance between points used to draw the curves generated by 2D smoothing; it is expressed as a fraction of the width of the window in the coordinate system in which the smoothing is being done.
Default: .01

T2D

Real---Tension on 2-Dimensional splines.
0. No 2D smoothing is done.
<0. 2D smoothing (using cubic splines under tension) is done. The absolute value of T2D is the desired tension. Smoothing is done before the mapping, if any, requested by the Conpack parameter MAP.
>0. 2D smoothing (using cubic splines under tension) is done. The absolute value of T2D is the desired tension. Smoothing is done after mapping.
Default: 0.

T3D

Real---Tension on 3-Dimensional splines. T3D specifies the tension on the 3D (bicubic) splines used by CPSPS1 or CPSPS2 to smooth the data being contoured.

Default: 1.

VPB

Real---ViewPort Bottom. VPB is only used when the Conpack parameter SET<>0, specifying that Conpack should do the call to SET. It specifies the position of the bottom edge of the area in which the viewport is to be placed, expressed as a fraction between 0. (the bottom edge of the plotter frame) and 1. (the top edge of the plotter frame). See the description of VPS.
Default: .05

VPL

Real---ViewPort Left. VPL is only used when the Conpack parameter SET<>0, specifying that Conpack should do the call to SET. It specifies the position of the left edge of the area in which the viewport is to be placed, expressed as a fraction between 0. (the left edge of the plotter frame) and 1. (the right edge of the plotter frame). See the description of VPS.
Default: .05

VPR

Real---ViewPort Right. VPR is only used when the Conpack parameter SET<>0, specifying that Conpack should do the call to SET. It specifies the position of the right edge of the area in which the viewport is to be placed, expressed as a fraction between 0. (the left edge of the plotter frame) and 1. (the right edge of the plotter frame). See the description of VPS.
Default: .95

VPS

Real---ViewPort Shape. VPS is only used when the Conpack parameter SET<>0, specifying that Conpack should do the call to SET. It specifies the desired viewport shape, as follows:

<0. Specifies the exact shape of the viewport; the absolute value is the ratio of the width of the viewport to its height.

0. Specifies a viewport completely filling the area specified by VPL, VPR, VPB, and VPT.

0.<VPS<1.
Specifies a plot of the shape determined by the values of XC1, XCM, YC1, and YCN, reverting to the shape specified by VPL, VPR, VPB, and VPT if the ratio of the shorter side to the longer side would be less than the value of VPS.

>=1. Specifies a plot of the shape determined by the values of XC1, XCM, YC1, and YCN, reverting to a square if the ratio of the longer side to the shorter side would be greater than the value of VPS.

The viewport, whatever its final shape, is centered in, and made as large as possible in, the area specified by the VPB, VPL, VPR, and VPT parameters.
Default: .25

VPT

Real---ViewPort Top. VPT is only used when the Conpack parameter SET<>0, specifying that Conpack should do the call to SET. It specifies the position of the top edge of the area in which the

viewport is to be placed, expressed as a fraction between 0. (the bottom edge of the plotter frame) and 1. (the top edge of the plotter frame). See the description of VPS.
Default: .95

WDB

Real---WinDow Bottom. When Conpack does the call to SET, the WDB parameter is used to determine argument number 7, the user Y coordinate at the bottom of the window. If WDB<>WDT, WDB is used. If WDB=WDT, but YC1<>YCN, then YC1 is used. Otherwise, the value 1. is used.
Default: 0.

WDL

Real---WinDow Left. When Conpack does the call to SET, the WDL parameter is used to determine argument number 5, the user X coordinate at the left edge of the window. If WDL<>WDR, WDL is used. If WDL=WDR, but XC1<>XCM, then XC1 is used. Otherwise, the value 1. is used.
Default: 0.

WDR

Real---WinDow Right. When Conpack does the call to SET, the WDR parameter is used to determine argument number 6, the user X coordinate at the right edge of the window. If WDR<>WDL, WDR is used. If WDR=WDL, but XCM<>XC1, then XCM is used. Otherwise, the value REAL(ZDM) is used.
Default: 0.

WDT

Real---WinDow Top. When Conpack does the call to SET, the WDB parameter is used to determine argument number 8, the user Y coordinate at the top of the window. If WDT<>WDB, WDT is used. If WDT=WDB, but YCN<>YC1, then YCN is used. Otherwise, the value REAL(ZDN) is used.
Default: 0.

WSO

Integer---WorkSpace Overflow flag. WSO determines what to do when a real or integer workspace overflow occurs, as follows:

- **0** Execution terminates with a fatal-error call to SETER.
- **1** An error message is written to the error file, then execution continues.
- **2** No error message is written, and execution continues.

When execution continues, the resulting plot will be incomplete. The values of IWU and RWU may be retrieved to find out how much workspace would have been used if the call did not cause a workspace overflow. Note that if you provide these amounts on a subsequent run, you are not assured that workspace overflow will be averted.
Default: 1

XC1

Real---X Coordinate at index 1. XC1 specifies the X coordinate value that corresponds to a value of 1 for the first subscript of the data array, prior to any mapping implied by a nonzero value of MAP. If XC1=XCM, 1. is used.
Default: 0.

XCM

Real---X Coordinate at index M. XCM specifies the X coordinate value that corresponds to a value of ZDM for the first subscript of the data array, prior to any mapping implied by a nonzero value of MAP. If XC1=XCM, REAL(ZDM) is used.
Default: 0.

YC1

Real---Y Coordinate at index 1. YC1 specifies the Y coordinate value that corresponds to a value of 1 for the second subscript of the data array, prior to any mapping implied by a nonzero value of MAP. If YC1=YCM, 1. is used.
Default: 0.

YCN

Real---Y Coordinate at index N. YCN specifies the Y coordinate value that corresponds to a value of ZDN for the second subscript of the data array, prior to any mapping implied by a nonzero value of MAP. If YC1=YCN, REAL(ZDN) is used.
Default: 0.

ZD1

Integer---Z data array Dimension 1. ZD1 specifies the first dimension of the array ZDAT, which contains the data to be contoured. If CPRECT is called, it sets ZD1 (the argument KZDT is the desired value). If CPSPS1 or CPSPS2 is called, it either picks a value of ZD1 (if ZDS<>0) or expects the user to have done so (if ZDS=0).
Default: 1

ZDM

Integer---Z data array Dimension M. ZDM specifies the first dimension of the array of data to be contoured. Its value will be less than or equal to the value of ZD1. If CPRECT is called, it sets ZDM (the argument MZDT is the desired value). If CPSPS1 or CPSPS2 is called, it either picks a value of ZDM (if ZDS<>0) or expects the user to have done so (if ZDS=0).
Default: 1

ZDN

Integer---Z data array Dimension N. ZDN specifies the second dimension of the array of data to be contoured. If CPRECT is called, it sets ZDN (the argument NZDT is the desired value). If

CPSPS1 or CPSPS2 is called, it either picks a value of ZDN (if ZDS<>0) or expects the user to have done so (if ZDS=0).
Default: 1

ZDS

Integer---Z data array Dimension Selector.
<>0 CPSPS1 or CPSPS2 will select values for ZD1, ZDM, and ZDN
0 ZD1, ZDM, and ZDN will be expected to have been set by the user.
Note that, if the size of the dense array is not a product of the size of the sparse array and some perfect square, the aspect ratio of the dense grid may be slightly different from that of the sparse grid.
Default: 1

ZDU

Real---Z Data value, Unscaled. See ZDV, the next entry.
Default: None

ZDV

Real---Z Data Value. ZDV is mostly for output. Its value may be retrieved in a user version of CPCHHL to retrieve the value of the high or low that is being labeled.
If a character string representing the value is desired, CPGETC may be used to obtain it (as modified by the current scale factor); thus, to obtain the character representation of an arbitrary value in a form consistent with the other values on a contour plot, set ZDV with a call to CPSETR and retrieve the value of ZDV with a call to CPGETC; if an unscaled value is desired, use the parameter name ZDU in the call to CPGETC.
Default: None

ZMN

Real---Z MiNimum value. The minimum value in the field, as found by CPRECT, CPSPS1, or CPSPS2. For output only.
Default: None

ZMX

Real---Z MaXimum value. The maximum value in the field, as found by CPRECT, CPSPS1, or CPSPS2. For output only.
Default: None

Index

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Primary references appear in bold type. Some entries with multiple references do not have bold type; this indicates that they convey information at approximately the same level of detail.

A

B

C

CFL
CFP
CFP
CFS
CFS
CFT
CFT
CFW
CFW
CFX
CFX
CFY
CFY
CIS
CIS
CIS
CIT
CIT
CIT
CIU
CLC
CLC
CLD
CLD
CLL
CLL
CLS
CLS
CLS
CLS
CLU
CLU
CLU
CLU
CLV
CLV
CLV
CMN
CMN
CMX
CMX
CTM
CWM
CWM
CWM
CWM
DPS
DPU

DPV
GIC
GIC
GIL
GIL
GIS
GIS
HCF
HCF
HCL
HCL
HCS
HIC
HIC
HIT
HLA
HLA
HLB
HLB
HLC
HLC
HLL
HLL
HLO
HLO
HLS
HLS
HLT
HLT
HLW
HLW
HLX
HLX
HLY
HLY
ILA
ILA
ILB
ILB
ILC
ILC
ILL
ILL
ILL
ILL
ILP
ILP

ILS
ILS
ILT
ILT
ILW
ILW
ILX
ILX
ILY
ILY
IWM
IWU
IWU
LBC
LBC
LBX
LBY
LIS
LIS
LIS
LIT
LIT
LIT
LIU
LIU
LLA
LLA
LLB
LLB
LLC
LLC
LLL
LLL
LLO
LLO
LLP
LLP
LLS
LLS
LLT
LLT
LLW
LLW
LOC
LOC
LOT
MAP
MAP

NCL
NCL
NCL
NEL
NEL
NET
NET
NEU
NEU
NLS
NLS
NLZ
NLZ
NOF
NOF
NSD
NSD
NVS
NVS
ORV
ORV
PAI
PAI
PAI
PAI
PAI
PC1
PC1
PC2
PC2
PC3
PC3
PC4 through PC6
PC4 through PC6
PIC
PIC
PIE
PW1
PW1
PW2
PW2
PW3
PW3
PW4
PW4
RC1 through RC3
RC1 through RC3
RWC

RWC
RWG
RWM
RWU
RWU
SET
SFS
SFU
SPV
SPV
SSL
SSL
T2D
T2D
T3D
VPB
VPB
VPL
VPL
VPR
VPR
VPS
VPS
VPT
VPT
WDB
WDL
WDR
WDT
WSO
XC1
XC1
XC1
XCM
XCM
XCM
YC1
YC1
YC1
YCN
YCN
YCN
ZD1
ZDM
ZDN
ZDS
ZDU
ZDV

F

G

H

I

J

K

L

M

Q

R

S

T

U

V

W

X