MTV Plot Data Format Version 1.4.1

Rev. 0

July, 1995

Kenny Toh

Technology CAD
Intel Corporation
2200 Mission College Boulevard
Santa Clara, California 95052

Table of Contents

CHAPTER 1

MTV Plot Data Format

1 INTRODUCTION

The MTV plot data format, or MTVDAT format, is used to specify data to be plotted in the **PLOTMTV** visualization program. The most important feature of this format is that multiple types of data may be specified in a single file or data-stream, thus avoiding the proliferation of data-files of various formats. As of this date, the MTVDAT format includes the following types of data:

- **2D Curves**
  Connected sets of (x, y) points in 2D space
- **3D Curves**
  Connected sets of (x, y, z) points in 3D space
- **3D Contour Curves**
  3D Triangles/rectangles to be contoured
- **3D rectangular mesh data**
  Z-values on a rectangular x-y grid, used primarily for contours
- **4D volumetric data**
  T-values on a rectangular x-y-z grid
- **2D/3D Vector data**
  Vectors in 2D/3D space
- **Probability Graph**
  2D (x, y) data plotted on a probability grid
- **Histogram**
  Frequency of occurrence of data plotted as 2D bars
- **Bar Charts**
  Data-points represented by 2D bars

The MTVDAT format also allows the inclusion of plot-options with the data, so that the properties of the plot may be customized.

The MTVDAT data-format is presently limited to ASCII and machine-dependent binary data.

2 THE MTVDAT FORMAT

2.1 Overview

Data in the MTVDAT format is organized into sets of data, or **datasets**. There are different types of datasets, corresponding to the various types of data that can be represented in the MTVDAT format, e.g., CURVE2D datasets, CONTOUR datasets.

Datasets are identified using **command** lines. An **command** line is of the format "`$ DATA=<TYPE>`"; the "`$`" character is the first non-blank character and is followed by a keyword specifying the type of data to be read. The **command** line is followed by the plot data, written in the appropriate format. The dataset ends with the next **command** line, or when a "`$ END`" line is encountered in the data-stream. An end-of-file (EOF) will also signify the end of the dataset, if reading from a data-file. Thus, using **command** lines, multiple datasets and multiple types of datasets can be included in a single file or data-stream.

Each dataset is assigned a number of properties, which instruct the plotting program on how to treat the data mathematically, and on how to construct the plot of the dataset. For example, in order to plot contours of a rectangular mesh, the plot program needs to know the contour intervals as well as labeling information for the plot. These plot resources are by default automatically assigned by the MTVDAT reader. However, the MTVDAT format also allows these resources to be set using optional **instruction** lines. **Instruction** lines are identified by a percent "`%`" sign as the first non-blank character in the line.

Comments can also be inserted in the data-stream; comments are preceded by the pound "`#`" character. Plot annotations can also be included as part of a dataset - see **Appendix B** for more details.

MTVDAT Special Line Identifiers				
First Character	Required Arguments	Optional Arguments	Type	Description
$	DATA = CURVE2D	NAME = <name>	Command	Identify dataset type
$	DATA = CURVE3D	NAME = <name>		
$	DATA = COLUMN	NAME = <name>		
$	DATA = CONTCURVE	NAME = <name>		
$	DATA = CONTOUR	NAME = <name>		
$	DATA = GRID4D	NAME = <name>		
$	DATA = VECTOR	NAME = <name>		
$	DATA = PROBABILITY	NAME = <name>		
$	DATA = HISTOGRAM	NAME = <name>		
$	DATA = BARCHART	NAME = <name>		
$	DATA = END		Command	Identify end of data-file or data-stream
%		<argument>=<value>	Instruction	Set data/plot properties
#			Comment	Include comments
@			Annotation	Plot Annotations

Table 1: The MTVDAT format uses special command, instruction, command and annotation lines to specify datasets and their properties.

Both **command** and **instruction** lines consist of `<argument>=<value>` pairs, that is, an argument or keyword separated from a value by an equal sign "=". In general, the argument of the `<argument>=<value>` pair is case-insensitive while the value string is case-sensitive. Multiple `<argument>=<value>` pairs can be specified on a single line, provided that they are separated by spaces or tabs. Both types of lines are terminated either by a "#" character or by the end-of-line (EOLN) character. Furthermore, **command** and **instruction** lines can be continued on a new line using the backslash "\" character. Examples of **command** and **instruction** lines are shown below.

```
$ Data=Curve2d name="Curve A"   # command

% linetype=1 linelabel="Hello"  # instruction; multiple
                                # <arg>=<value> pairs may be
                                # placed in the same line

% LINELABEL="Hello"             # arguments are case-insensitive...
% LINELABEL="HELLO"             # but values are case-sensitive

% grid = ON                     # some values are booleans
% grid = True                   # valid boolean strings are
% grid = T                      #    TRUE: "True", "T", "On"
% grid = False                  #    FALSE: "False", "F", "Off"
% grid = off                    # Booleans are case-insensitive
% grid                          # If the value is omitted, a TRUE
                                # value is assigned

$ data = curve2d \
  name = "Curve A"              # "\" denotes a new line

% xlabel = "Hello World"        # space-separated strings MUST be
                                # enclosed in single/double quotes
% ylabel = hello                # quotes can be omitted for single
                                # words
```

2.2 COMMANDS: Identifying Datasets

Several types of datasets are currently supported, identified in the following manner:

```
$ DATA = CURVE2D
$ DATA = CURVE3D
$ DATA = COLUMN
$ DATA = CONTCURVE
$ DATA = CONTOUR
$ DATA = GRID4D
$ DATA = VECTOR
$ DATA = PROBABILITY
$ DATA = HISTOGRAM
$ DATA = BARCHART
```

Detailed descriptions and formatting information for each type of dataset are provided in the following sections.

As described earlier, a single file or data-stream can contain one or more datasets, each preceded by a **command** line. Thus, a data-file in the MTVDAT format has the following structure:

```
$ DATA = CURVE2D
   [optional properties]
   [2D curve data]
$ DATA = CURVE2D
   [optional properties]
   [2D curve data]
$ DATA = CURVE3D
   [optional properties]
   [3D curve data]
$ DATA = <TYPE>
   [optional properties]
   [data]
$ DATA = <TYPE>
   [optional properties]
   [data]
 ...
 ...
 ...
$ END
```

The **command** line forces the MTVDAT reader to invoke a special reader for each type of data. Thus, a "`$DATA=CURVE2D`" **command** tells the MTVDAT reader to expect data in a CURVE2D format, which is essentially 2 columns of data. Similarly, a "`$DATA=CONTOUR`" **command** calls up the MTVDAT CONTOUR reader, which reads in 3D data and calculates contour lines of equal z-values. The data read in is saved in a dataset structure for later plotting.

Datasets can also be identified using unique character strings, by including the "`NAME=<label>`" argument in the **command** line. The following example defines two CURVE2D datasets named "`Curve A`" and "`Curve B`" respectively.

```
$ DATA=CURVE2D NAME="Curve A"
  0 0
  1 1
  2 2
$ DATA=CURVE2D NAME="Curve B"
  1 0
  2 0
  3 0
$ END
```

Note that with the exception of the string used for the dataset name, the **command** line is case-insensitive. Furthermore, as mentioned previously, a **command** can be continued on a new line using the backslash "\" character. Thus the following **commands** are identical:

```
$ DATA=CURVE2D NAME="Curve A"
$ Data = Curve2d name = "Curve A"
$ data = curve2d \
  name = "Curve A"
```

2.3 INSTRUCTIONS: Customizing the Plot

The properties of the data or the intended plot can also be customized by including **instruction** lines along with each dataset specification. A single **instruction** line contains one or more `<argument>=<value>` pairs, separated by spaces or tabs. The list of acceptable `<argument>=<value>` keywords depends on the type of data being read; refer to **Appendix A** for a complete list of **Plot Resources**.

The example below demonstrates the use of **instruction** lines to set line-types and labels for curves. The following defines a CURVE2D dataset, where the curve in the dataset has been assigned specific properties, such as linewidth, labels, and markers.

```
$ DATA=CURVE2D NAME="Curve A"
% linetype=1 linewidth=2 linelabel="Cv 1"
% markertype=3  # Draw the markers too
  0 0
  1 1
  2 2
$ END
```

Note in the above that the **command** line is similar in form to the **instruction** line, the primary difference being that the **command** line begins with a "`$`" character and the **instruction** line begins with a "`%`" character.

2.4 ANNOTATIONS: Annotating the Plot

Plot annotations, such as text labels, arrows and lines, may also be attached to sets of data using **annotation** lines. Annotations are specified in the data-file as strings beginning with the "`@`" character. Multiple annotations can be specified in each dataset. Furthermore, each annotation can be specified in data (world) coordinates or in plot coordinates.

As with **instruction** lines, an **annotation** line contains one or more `<argument>=<value>` pairs, separated by spaces or tabs. The list of acceptable `<argument>=<value>` keywords is specified in **Appendix B**.

The example below demonstrates the use of **annotations** to place a label and an arrow pointing to one of the data-points within the plot.

```
$ DATA=CURVE2D NAME="Curve A"

% linetype=1 markertype=4 markersize=2
  0 0
  1 1
  2 2

# Arrow annotation syntax:
#   @ arrow P1 P2 [properties]
#
#              P1--------------->P2
#            label

@ arrow x1=1.30 y1=0.70 x2=1.03 y2=0.97 linelabel="Mid Point"

$ END
```

3 CURVE2D FORMAT

3.1 Overview

The CURVE2D format is used to specify a list of joined points in 2D (x, y) space. A CURVE2D dataset may contain one or more curves, each of which contain one or more points.

3.2 Resources

The CURVE2D format accepts the standard list of **Plot Resources**, which are listed in **Appendix A**. By default, each curve in the dataset is plotted in a continuous solid line and individual points are not marked; use **Curve Resources** for changing these and other curve properties. In addition, the CURVE2D format accepts the following resources:

CURVE2D Property Argument List

Name	Type	Default	Range	Description
boundary	Boolean	False	True/False	Enlarge plot boundary
binary	Boolean	False	True/False	Machine-dependent binary
npts	integer	-	-	Binary array size
curveno	integer	-	$-\infty$ — ∞	Curve integer identifier

boundary Enlarges the default plot-boundary. By default, **boundary=False**, and the data is plotted with a plot-boundary equal to the data limits, i.e., the rectangular boundary on the plot falls exactly on the minimum and maximum x-y data-values. This can obscure data-points that lie on the boundary, however. **Boundary=True** enlarges the plot-boundary slightly so that all the data-points are visible.

binary Specifies whether the curve data is written in ASCII or machine-dependent binary format. If **True**, the program tries to read 2 double-precision binary arrays of size **npts** containing the x and y coordinates of the points in the curve. The arrays are assumed to be on the line following the **binary**=**True instruction**. Default is **False**.

npts Specifies the number of points in a curve. Used only for reading data in binary format.

curveno Specifies the integer number attached as an attribute to the curve. The integer identifier is used only for debugging purposes (use “%curveid=on” option to display curve numbers during plotting).

3.3 ASCII Format Specification

Each curve consists of a number of (x, y) points. The coordinates and optional integer identifier (ID) of the point must be specified on the same line as follows:

```
x-coordinate y-coordinate [point-ID]
```

Note in the above that the various fields are space or tab-separated. Each field is format-free, meaning that the numbers may be specified using decimals, exponential or even integer formats.

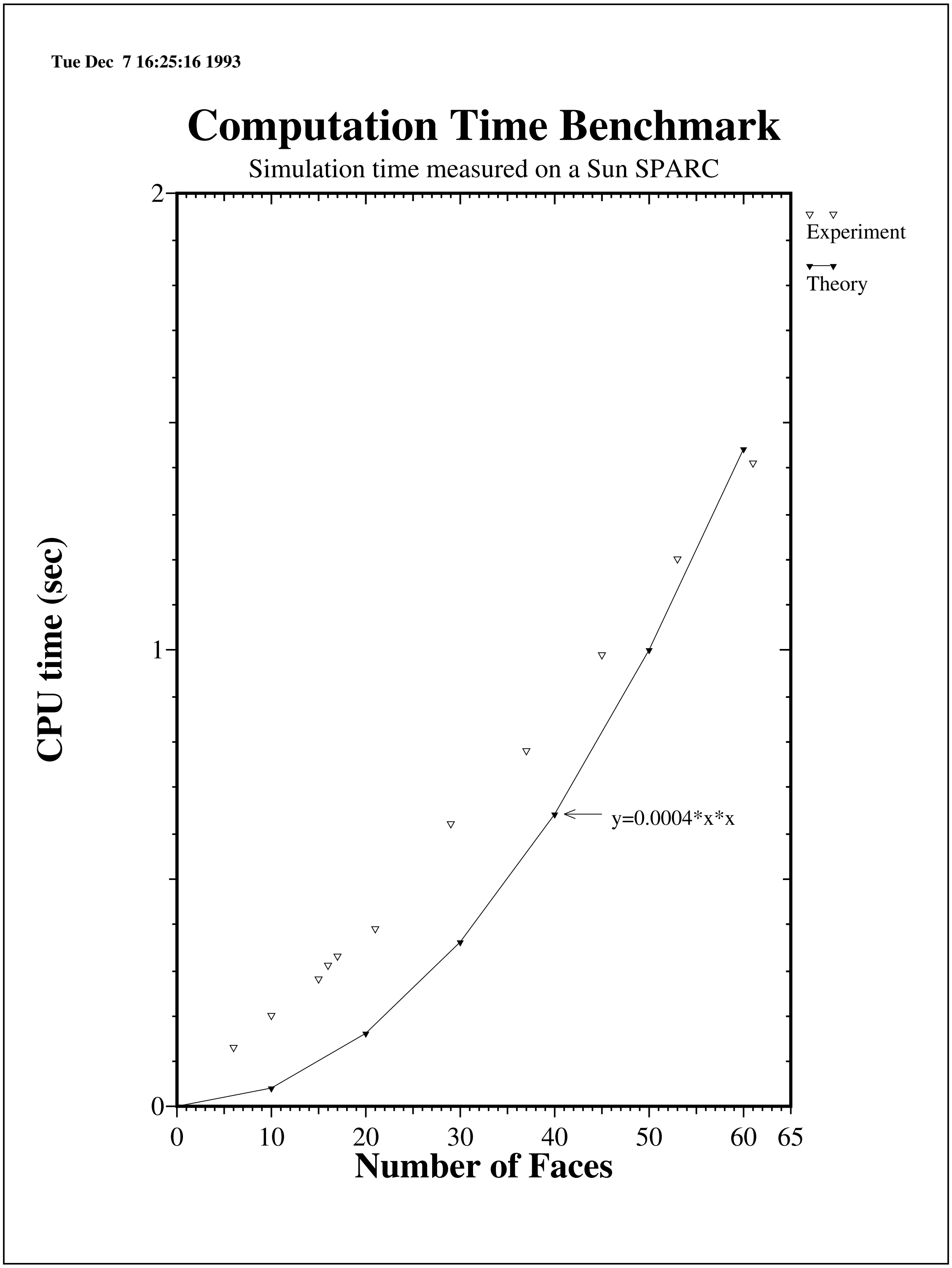
The points on each curve are specified on adjacent lines. Multiple curves are separated by one or more blank lines. The CURVE2D format thus has the following form:

```
$ DATA=CURVE2D

% [optional instructions]

# ASCII Curve
% [optional instructions]
x1 y1   #[point 1, curve 1]
x2 y2   #[point 2, curve 1]
 ...
xn yn   #[point n, curve 1]

# ASCII Curve
 ...

$ END
```

3.4 Example

The following shows an example file containing a CURVE2D dataset with 2 curves. The resultant MTV plot is shown on the following page.

```
#
# The first dataset in the file is assumed to be a "CURVE2D" dataset,
# unless specified otherwise using the "DATA=TYPE" command.
#

$ DATA=CURVE2D

# Plot properties
% toplabel  = "Computation Time Benchmark"
% subtitle  = "Simulation time measured on a Sun SPARC"
% xlabel    = "Number of Faces"
% ylabel    = "CPU time (sec)"
% xmin      =  0.0            # Set the plot boundaries explicitly
% xmax      = 65.0
% ymin      =  0.0
% ymax      =  2.0

# First curve
% linelabel="Experiment"
% linetype=0  markertype=10   # Just draw markers
   6     0.13
  10     0.20
  15     0.28
  16     0.31
  17     0.33
  21     0.39
  29     0.62
  37     0.78
  45     0.99
  53     1.20
  61     1.41

# Second curve
% linelabel="Theory"
% markertype=11
   0    0
  10    0.04
  20    0.16
  30    0.36
  40    0.64
  50    1.00
  60    1.44

# Annotate the second curve
@ arrow x1=45 y1=0.64 x2=41 y2=0.64 linelabel="y=0.0004*x*x"

$ END
```


Tue Dec 7 16:25:16 1993
Computation Time Benchmark
Simulation time measured on a Sun SPARC
Experiment
Theory
y=0.0004*x*x
CPU time (sec)
0
1
2
0
10
20
30
40
50
60
65
Number of Faces

3.5 Binary Format Specification

CURVE2D datasets may also be specified in a machine-dependent binary format; this format will work only for files produced and read on the same compute platform. The machine-dependence is a disadvantage, but it is balanced by the fact that reading binary data can be up to 10 times faster than reading ASCII data. Note also that when programs are exchanging data using pipes, machine-dependence becomes a moot issue, since the pipe-linked programs must be on the same machine in order for pipes to work.

CURVE2D binary data is identified by the following **instruction** line:

```
% binary=True npts=<value>
```

In the above, `npts` is the number of points in the curve. The binary data-stream must begin on the line immediately following the **instruction** line above. The binary stream consists of 2 arrays of type double-precision which contain x and y coordinate data respectively. Each array is assumed to be of size `npts`. Note that the binary format does not provide a means of specifying the point IDs.

Binary and ASCII curves may be mixed in the same file or data-stream. However, a single curve must be either totally ASCII or totally binary; ASCII and binary formats may not be mixed in a single curve. The CURVE2D format combining both ASCII and binary curves thus has the following form:

```
$ DATA=CURVE2D

% [optional instructions]

# ASCII Curve
% [optional instructions]
x1 y1   #[point 1, curve 1]
x2 y2   #[point 2, curve 1]
 ...
xn yn   #[point n, curve 1]

# Binary Curve
% [optional instructions]
% binary=True npts=<number>
[x-coordinate binary double-precision array][1]
[y-coordinate binary double-precision array]

# ASCII Curve
 ...

# Binary Curve
 ...

$ END
```

1. The square brackets denote an array; the brackets are not to be written out to the data-stream.

3.6 Programming Example

The following short program illustrates writing binary data in the CURVE2D format.

```
#include <stdio.h>
#include <math.h>
main()
{
#define BINARYFILE "data.mtvdat"
   FILE   *fp;
   double xarr[100], yarr[100];
   int    i, npts=100;

   /* Open up a file */
   if ((fp=fopen(BINARYFILE,"w")) == NULL) {
      (void) fprintf(stderr,"cat: Couldn't open file %s\n",BINARYFILE);
      exit(-1);
   }

   /* Fill arrays */
   for (i=0; i<npts; i++) {
      xarr[i] = 0.1*i;
      yarr[i] = cos(xarr[i]);
   }

   /* Write out CURVE2D header */
   (void) fprintf(fp,"$ DATA=CURVE2D\n");

   /* Write out binary data */
   (void) fprintf(fp,"%% BINARY npts=%d\n",npts);
   if (fwrite((char *)xarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1);
   }
   if (fwrite((char *)yarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1);
   }
   (void) fprintf(fp,"$ END\n");

   /* Close the file */
   (void) fclose(fp);
}
```

4 CURVE3D FORMAT

4.1 Overview

The CURVE3D format is used to specify a list of joined points in 3D (x, y, z) space. A CURVE3D dataset may contain one or more curves, each of which contain one or more points. Note that the CURVE3D format is similar to the CURVE2D format except for the addition of an extra coordinate.

4.2 Resources

The CURVE3D format accepts the standard list of **Plot Resources**, which are listed in **Appendix A**. By default, each curve in the dataset is plotted in a continuous solid line and individual points are not marked; use **Curve Resources** for changing these and other curve properties. In addition, the CURVE3D format accepts the following resources:

CURVE3D Property Argument List

Name	Type	Default	Range	Description
boundary	Boolean	False	True/False	Enlarge plot boundary
binary	Boolean	False	True/False	Machine-dependent binary
npts	int	-	-	Binary array size
curveno	integer	-	$-\infty$ — ∞	Curve integer identifier

boundary Enlarges the default plot-boundary. By default, **boundary=False**, and the data is plotted with a plot-boundary equal to the data limits, i.e., the cubic boundary on the 3D plot falls exactly on the minimum and maximum x-y-z data-values. This can obscure data-points that lie on the boundary, however. **Boundary=True** enlarges the plot-boundary slightly so that all the data-points are visible.

binary Specifies whether the curve data is written in ASCII or machine-dependent binary format. If **True**, the program tries to read 3 double-precision binary arrays of size **npts** containing the x, y and z coordinates of the points in the curve. The arrays are assumed to be on the line following the **binary=True instruction**. Default is **False**.

npts Specifies the number of points in a curve. Used only for reading data in binary format.

curveno Specifies the integer number attached as an attribute to the curve. The integer identifier is used only for debugging purposes (use “%curveid=on” option to display curve numbers during plotting).

4.3 ASCII Format Specification

Each curve consists of a number of (x, y, z) points. The coordinates and optional integer identifier (ID) of the point must be specified on the same line as follows:

```
x-coordinate y-coordinate z-coordinate [point-ID]
```

Note in the above that the various fields are space or tab-separated. Each field is format-free, meaning that the numbers may be specified using decimals, exponential or even integer formats.

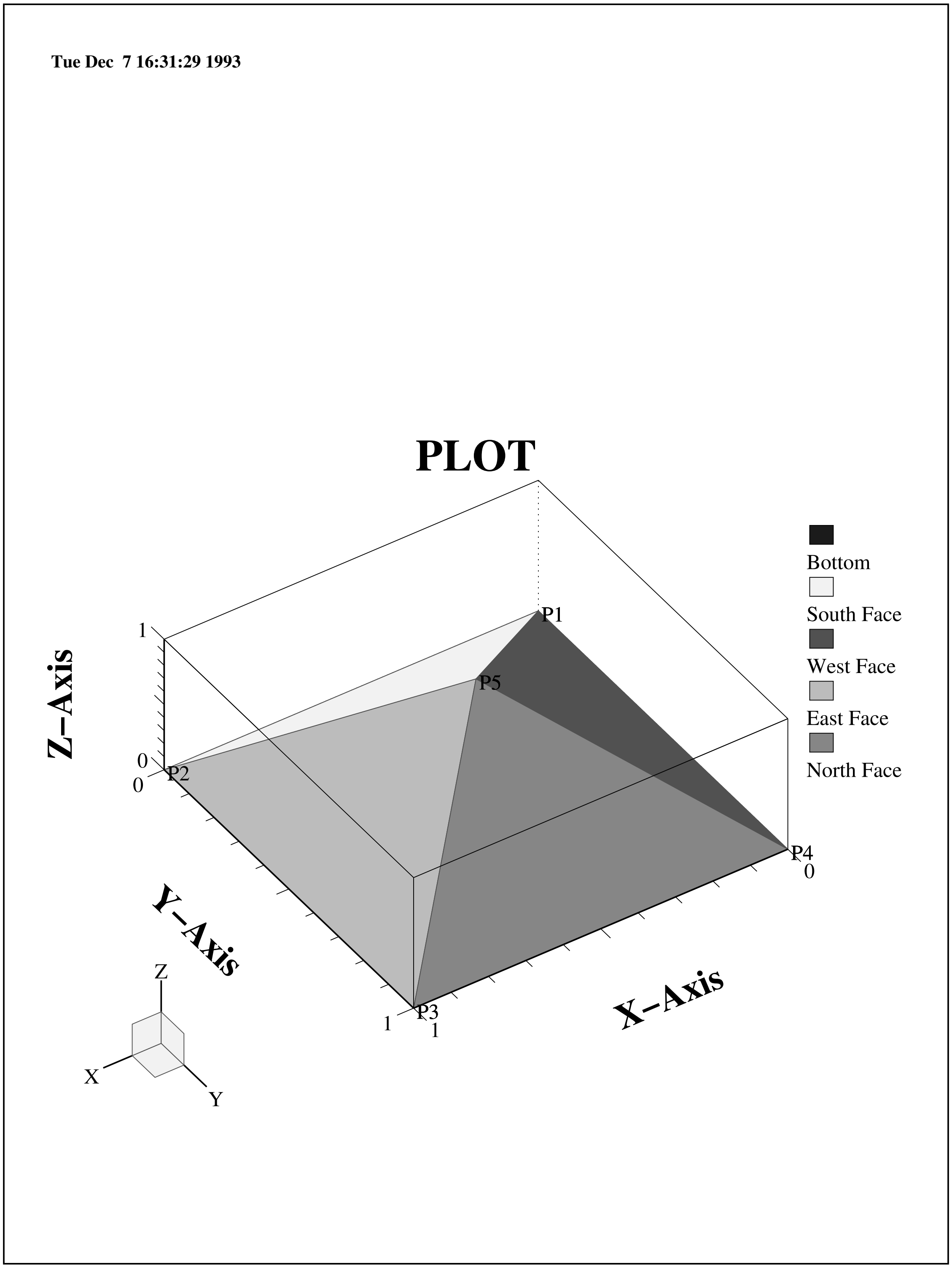
The points on each curve are specified on adjacent lines. Multiple curves are separated by one or more blank lines. The ASCII CURVE3D format thus has the following form:

```
$ DATA=CURVE3D

# ASCII Curve
% [optional instructions]
x1 y1 z1  #[point 1, curve 1]
x2 y2 z2  #[point 2, curve 1]
 ...
xn yn zn  #[point n, curve 1]

# ASCII Curve
 ...

$ END
```

4.4 Example

The following shows an example file containing a CURVE3D dataset. This dataset has five filled curves which make up the surfaces of a pyramid.

```
#
# Specify the 5 faces of a pyramid
# Paint each face with a different color
#
$ DATA=CURVE3D Name=Pyramid

% dfilltype=1                   # Solid fill for each face
% pointid=True                  # Print point ID's
% hiddenline=True               # Apply hiddenline/hiddensurface sorting

% fillcolor=1 linelabel="South Face"
0.0 0.0 0.0   1
1.0 0.0 0.0   2
0.5 0.5 1.0   5

% fillcolor=2 linelabel="East Face"
1.0 0.0 0.0   2
1.0 1.0 0.0   3
0.5 0.5 1.0   5

% fillcolor=3 linelabel="North Face"
1.0 1.0 0.0   3
0.0 1.0 0.0   4
0.5 0.5 1.0   5

% fillcolor=4 linelabel="West Face"
0.0 1.0 0.0   4
0.0 0.0 0.0   1
0.5 0.5 1.0   5

% fillcolor=5 linelabel="Bottom"
0.0 0.0 0.0   1
1.0 0.0 0.0   2
1.0 1.0 0.0   3
0.0 1.0 0.0   4

$ END
```


Tue Dec 7 16:31:29 1993
PLOT
Z–Axis
Y–Axis
X–Axis
P1
P2
P3
P4
P5
Bottom
South Face
West Face
East Face
North Face
Z
X
Y
1
0
0
1
1
0

4.5 Binary Format Specification

CURVE3D datasets may also be specified in a machine-dependent binary format; this format will work only for files produced and read on the same compute platform. The machine-dependence is a disadvantage, but it is balanced by the fact that reading binary data can be up to 10 times faster than reading ASCII data. Note also that when programs are exchanging data using pipes, machine-dependence becomes a moot issue, since the pipe-linked programs must be on the same machine in order for pipes to work.

CURVE3D binary data is identified by the following **instruction** line:

```
% binary=True npts=<number-of-points>
```

In the above, `npts` is the number of points in the curve. The binary data-stream must begin on the line immediately following the **instruction** line above. The binary stream consists of 3 arrays of type double-precision which contain x, y, and z coordinate data respectively. Each array is assumed to be of size `npts`. Note that the binary format does not provide a means of specifying the point IDs.

Binary and ASCII curves may be mixed in the same file or data-stream. However, a single curve must be either totally ASCII or totally binary; ASCII and binary formats may not be mixed in a single curve. The CURVE3D format thus has the following form:

```
$ DATA=CURVE3D

% [optional instructions]

# ASCII Curve
% [optional instructions]
x1 y1 z1  #[point 1, curve 1]
x2 y2 z2  #[point 2, curve 1]
 ...
xn yn zn  #[point n, curve 1]

# Binary Curve
% [optional instructions]
% binary=True npts=<number>
[x-coordinate binary double-precision array]1
[y-coordinate binary double-precision array]
[z-coordinate binary double-precision array]

# ASCII Curve
 ...

# Binary Curve
 ...
$ END
```

1. The square brackets denote an array; the brackets are not to be written out to the data-stream.

4.6 Programming Example

The following short program illustrates writing binary data in the CURVE3D format.

```
#include <stdio.h>
#include <math.h>
main()
{
#define BINARYFILE "data.mtvdat"
   FILE   *fp;
   double xarr[100], yarr[100], zarr[100];
   int    i, npts=100;

   /* Open up a file */
   if ((fp=fopen(BINARYFILE,"w")) == NULL) {
      (void) fprintf(stderr,"cat: Couldn't open file %s\n",BINARYFILE);
      exit(-1);
   }

   /* Fill arrays */
   for (i=0; i<npts; i++) {
      xarr[i] = 0.1*i;
      yarr[i] = cos(xarr[i]);
      zarr[i] = sin(xarr[i]);
   }

   /* Write out CURVE3D header */
   (void) fprintf(fp,"$ DATA=CURVE3D\n");

   /* Write out binary data */
   (void) fprintf(fp,"%% BINARY npts=%d\n",npts);
   if (fwrite((char *)xarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1);
   }
   if (fwrite((char *)yarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1);
   }
   if (fwrite((char *)zarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1);
   }
   (void) fprintf(fp,"$ END\n");

   /* Close the file */
   (void) fclose(fp);
}
```

5 COLUMN FORMAT

5.1 Overview

The COLUMN format specifies 2D curves in multiple columns. The format can be considered a superset of the CURVE2D format; each column represents a single CURVE2D dataset, so multiple CURVE2D datasets can be constructed from the COLUMN data.

Unfortunately, the COLUMN format is not as flexible as the CURVE2D format, in that the COLUMN format does not allow modification of the line/marker properties for the different columns of data. If such is desired, use the CURVE2D format instead.

5.2 Resources

The COLUMN format accepts the standard list of **Plot Resources**, which are listed in **Appendix A**. By default, each curve in the dataset is plotted in a continuous solid line and individual points are not marked; use **Curve Resources** for changing these and other curve properties. In addition, the COLUMN format accepts the following resources:

COLUMN Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
xcolumn	String	NULL	N/A	x-coordinate column

xcolumn Specifies the column containing the x-coordinate data. If not specified, the x-column is assumed to be the first column. See the Format Specification for more details.

5.3 ASCII Format Specification

The COLUMN format has the following form:

```
$ DATA=COLUMN

% [optional instructions]

% xcolumn = "column i"
"column 1" "column 2" ... "column i" ... "column m"  # column labels

# 1st set of curves (ASCII)
% [optional instructions]
    y1.1      y2.1          yi.1          yn.1
    y1.2      y2.2          yi.2          yn.2
    ...
    y1.n      y2.n          yi.n          yn.n

# 2nd set of curves (ASCII)
% [optional instructions]
    y1.1      y2.1          yi.1          yn.1
    y1.2      y2.2          yi.2          yn.2
    ...
    y1.n      y2.n          yi.n          yn.n

# More curves (ASCII)
    ...

$ END
```

As shown, the column format consists of multiple columns of data. Each column is identified with a string label. One of the columns is the x-ordinate; the other columns are the various y-ordinates. Each data-point in all the columns is specified on the same line as follows (assuming the 1st column is the x-column):

```
x-coordinate y1-coordinate y2-coordinate ... yn-coordinate
```

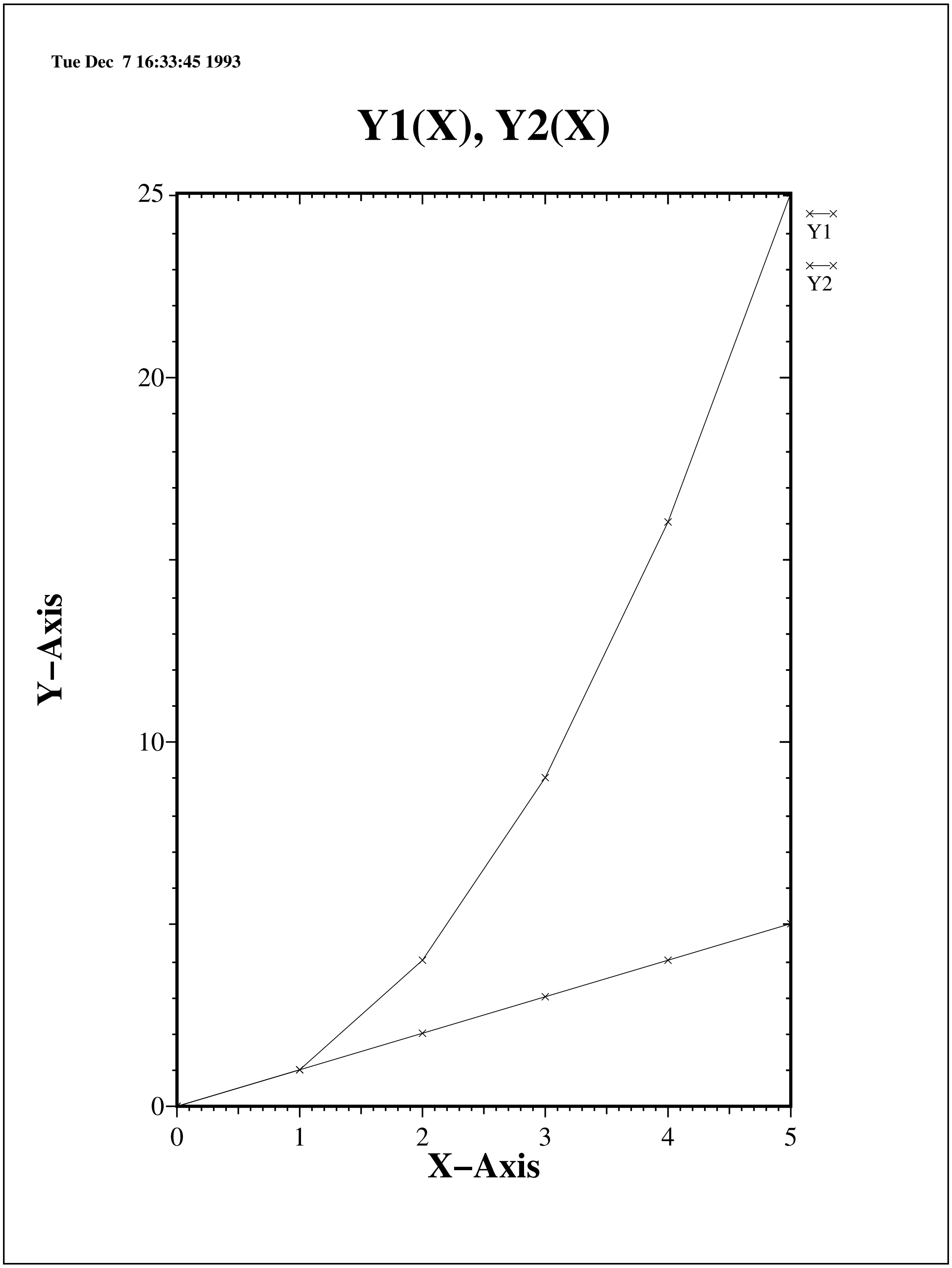
Note in the above that the various fields are space or tab-separated. Each field is format-free, meaning that the numbers may be specified using decimals, exponential or even integer formats.

Each column, when paired with the x-column, forms a set of curves which is stored in a CURVE2D dataset. Thus, given *N* columns, *N-1* CURVE2D datasets are generated. As in the CURVE2D format, the points on each curve are specified on adjacent lines. Multiple curves are separated by one or more blank lines.

5.4 Example

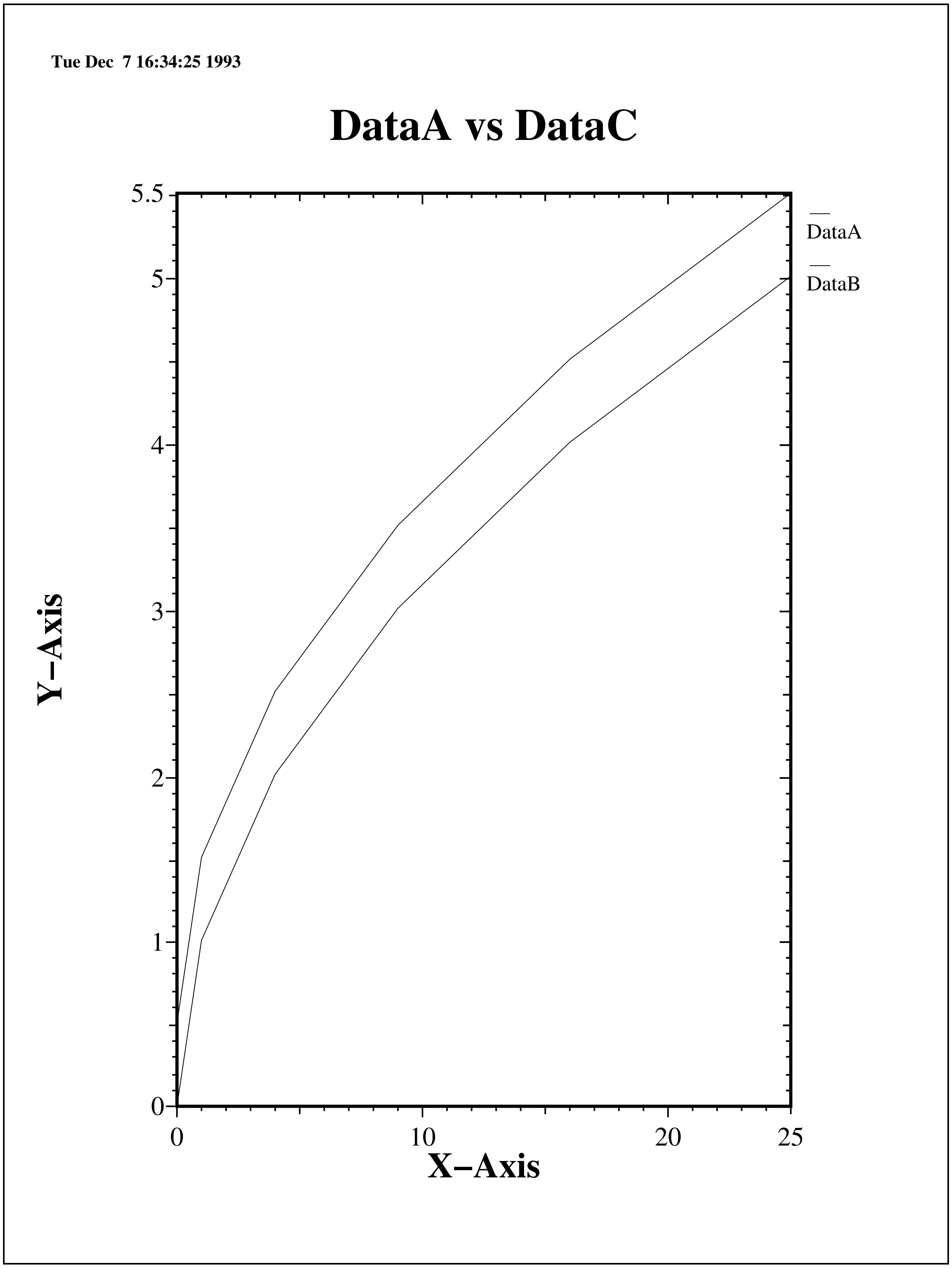
The following shows an example file containing a COLUMN dataset. The dataset contains 3 columns, so 2 CURVE2D datasets are generated when the data is read in. The resultant plot, containing both datasets, is shown on the opposite page.

```
#
# This column dataset contains 3 columns.
# Because xcolumn is not specified, the first column is taken to be the
# x-coordinate data.
#
$ DATA=COLUMN

% toplabel="Y1(X), Y2(X)"

X   Y1   Y2
% markertype=3
0.0 0.0  0.0
1.0 1.0  1.0
2.0 2.0  4.0
3.0 3.0  9.0
4.0 4.0 16.0
5.0 5.0 25.0

$ END
```


Tue Dec 7 16:33:45 1993
Y1(X), Y2(X)
Y1
Y2
Y−Axis
X−Axis
25
20
10
0
0
1
2
3
4
5

The next example shows a similar column dataset. In this case, the x-column is the third column, labeled "DataC".

```
#
# This column dataset contains 3 columns.
# The xcolumn is specified to be that labeled "DataC"
#
$ DATA=COLUMN

% xcolumn = DataC
DataA DataB DataC
0.0 0.5  0.0
1.0 1.5  1.0
2.0 2.5  4.0
3.0 3.5  9.0
4.0 4.5 16.0
5.0 5.5 25.0

$ END
```


Tue Dec 7 16:34:25 1993
DataA vs DataC
5.5
5
4
3
2
1
0
0
10
20
25
X−Axis
Y−Axis
DataA
DataB

6 CONTCURVE FORMAT

6.1 Overview

The CONTCURVE format specifies a 3-dimensional surface as a collection of 3D triangles. The format is identical to that of the CURVE3D format; refer to the section on the CURVE3D format for full details on the format specification.

Curves specified in the CONTCURVE format are converted into triangles which are then sliced to obtain contours of the surface. Curves with 3 points become triangles, while curves with greater then 3 points are triangulated.[1]

Once the data has been read in, the 3D mesh is sliced to obtain contours; each contour is a curve of iso-z-values. By default, approximately 10 contour lines are plotted at constant z-intervals; the number of contours, their intervals and plot style (i.e., lines, gradated colors, mesh) may be set using **Dataset Properties** (see **Appendix A**).

6.2 Resources

The CONTCURVE format accepts the standard list of **Plot Resources**, as well as the following:

CONTCURVE Property Argument List

Name	Type	Default	Range	Description
binary	Boolean	False	True/False	Machine-dependent binary
npts	int	-	-	Binary array size

binary Specifies whether the curve data is written in ASCII or machine-dependent binary format. If **True**, the program tries to read 3 double-precision binary arrays of size **npts** containing the x, y and z coordinates of the points in the curve. The arrays are assumed to be on the line following the **binary=True instruction**. Default is **False**.

npts Specifies the number of points in a curve. Used only for reading data in binary format.

1. Note that the triangulation routine has not been perfected yet; the routine fails when passed 2 or more data-points sharing the same x-y coordinates. Furthermore, the triangles generated are sometimes obtuse, resulting in strange plots.

6.3 Format Specification

Each curve consists of a number of (x, y, z) points. The points on a curve may be specified either in ASCII or in binary. The ASCII format requires the coordinates of the points in the curve to be listed one-per-line as follows:

```
x-coordinate y-coordinate z-coordinate [point-ID]  # point 0
x-coordinate y-coordinate z-coordinate [point-ID]  # point 1
 ...
x-coordinate y-coordinate z-coordinate [point-ID]  # point n
```

In contrast, the binary format has the following form:

```
% binary=True npts=<number-of-points>
[x-coordinate binary double-precision array]1
[y-coordinate binary double-precision array]
[z-coordinate binary double-precision array]
```

where the **instruction** line containing the `binary` flag is immediately followed by arrays containing x, y, and z coordinate data.

Multiple curves are separated by one or more blank lines. The CONTCURVE format thus has the following form:

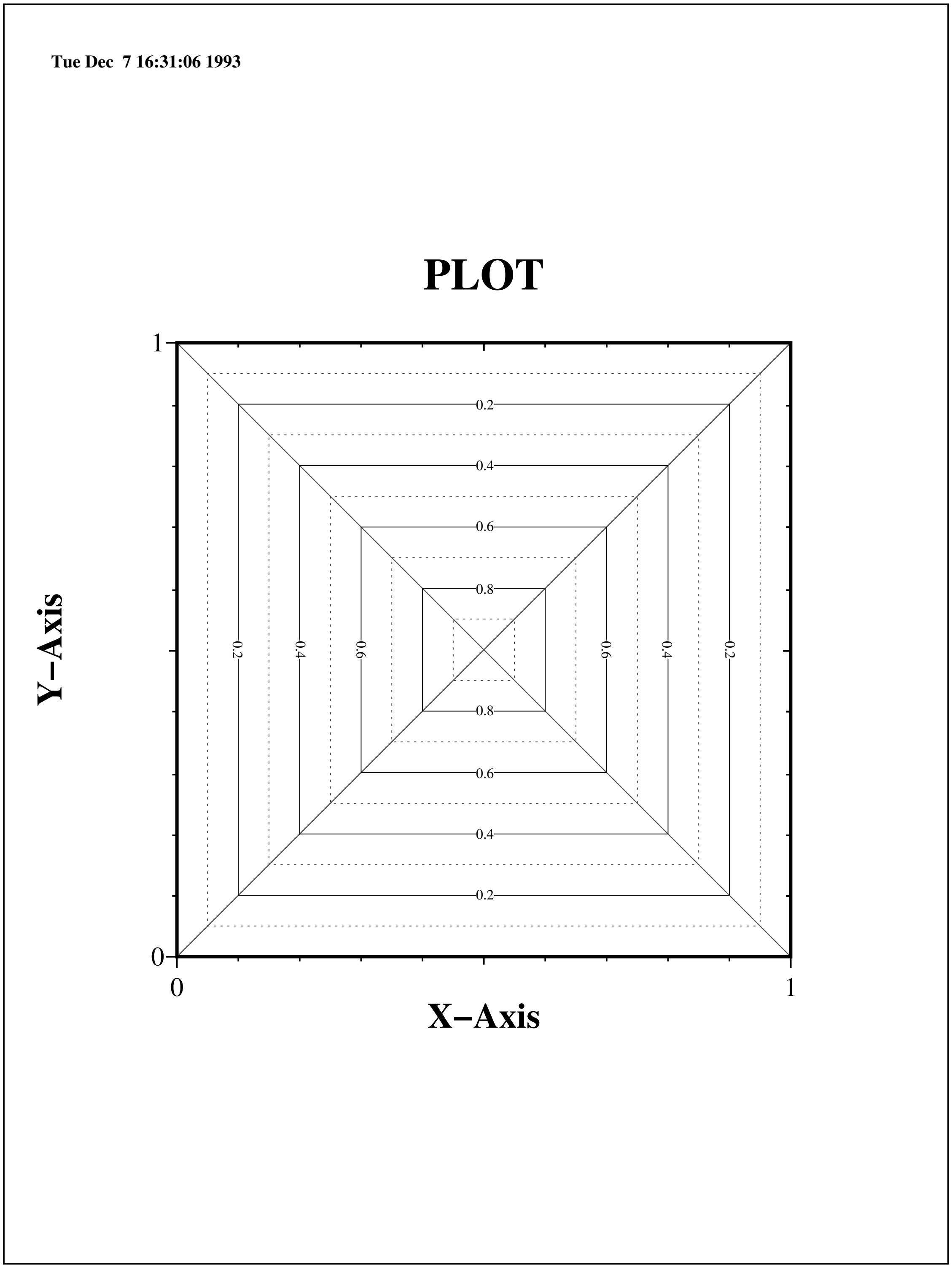
```
$ DATA=CONTCURVE
# ASCII Curve
% [optional instructions]
x1 y1 z1  #[point 1, curve 1]
x2 y2 z2  #[point 2, curve 1]
 ...
xn yn zn  #[point n, curve 1]

# Binary Curve
% [optional instructions]
% binary=True npts=<number-of-points>
[x-coordinate binary double-precision array]
[y-coordinate binary double-precision array]
[z-coordinate binary double-precision array]

# ASCII Curve
 ...

# Binary Curve
 ...
$ END
```

1. The square brackets denote an array; the brackets are not to be written out to the data-stream.

6.4 Example

The following shows an example file containing a CONTCURVE dataset. This dataset has five filled curves which make up the surfaces of a pyramid. The curves are stored as 6 triangles, which are then sliced for contours of equal-z values.

```
#
# Specify the 5 faces of a pyramid
# Paint each face with a different color
#
$ DATA=CONTCURVE Name=Pyramid

% meshplot=true           # plot the underlying mesh too

% fillcolor=1 linelabel="South Face"
0.0 0.0 0.0   1
1.0 0.0 0.0   2
0.5 0.5 1.0   5

% fillcolor=2 linelabel="East Face"
1.0 0.0 0.0   2
1.0 1.0 0.0   3
0.5 0.5 1.0   5

% fillcolor=3 linelabel="North Face"
1.0 1.0 0.0   3
0.0 1.0 0.0   4
0.5 0.5 1.0   5

% fillcolor=4 linelabel="West Face"
0.0 1.0 0.0   4
0.0 0.0 0.0   1
0.5 0.5 1.0   5

% fillcolor=5 linelabel="Bottom"
0.0 0.0 0.0   1
1.0 0.0 0.0   2
1.0 1.0 0.0   3
0.0 1.0 0.0   4

$ END
```


Tue Dec 7 16:31:06 1993
PLOT
1
0
0
1
X−Axis
Y−Axis
0.2
0.4
0.6
0.8
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.6
0.4
0.2

7 CONTOUR FORMAT

7.1 Overview

The CONTOUR format specifies a 3-dimensional surface on a rectangular grid. The data consists of a grid-specification, denoting the points on the x-y grid, followed by the main body, consisting of z-values at the nodes of the x-y grid. The surface z-values are assumed to be single-valued functions of x and y. The x-y rectangular grid may be a uniform or non-uniform grid, i.e., the spacing of the grid in x and y does not necessarily have to be uniform. A single dataset may contain only one 3D mesh.

Once the data has been read in, the 3D mesh is sliced to obtain contours; each contour is a curve of iso-z-values. By default, approximately 10 contour lines are plotted at constant z-intervals; the number of contours, their intervals and plot style (i.e., lines, gradated colors, mesh) may be set using **Dataset Properties** (see **Appendix A**).

7.2 Resources

The CONTOUR format accepts all of the **Plot Resources** listed in **Appendix A**, except for the **Curve Resources** and the **Global Curve Resources**.[1] In addition, the CONTOUR format accepts the following resources (specified in **instruction** lines):

CONTOUR Property Argument List

Name	Type	Default	Range	Description
nx	int	-	-	No. of grid points in x (required)
xmin	double	0.0	$-\infty$ — ∞	Uniform grid boundary
xmax	double	nx - 1	$-\infty$ — ∞	Uniform grid boundary
ny	int	-	-	No. of grid points in y (required)
ymin	double	0.0	$-\infty$ — ∞	Uniform grid boundary
ymax	double	ny - 1	$-\infty$ — ∞	Uniform grid boundary
binary	Boolean	False	True/False	Machine-dependent binary
joincurve	integer	0	0 — 2	Join contours at boundary 0 = Don't join 1 = Join with boundary at zmin 2 = Join with boundary at zmax

1. It is also possible to set curve properties (e.g., linetype, linewidth) of the contour curves derived from a CONTOUR dataset using the **contours** Dataset Resource, e.g., "`% contours = (0.1 lw=2 lt=1)`". See **Appendix A** for details.

nx Specifies the number of grid-points on the x-axis. Minimum value is 2. Required for both uniform and non-uniform grid specification.

xmin

xmax Specifies the limits of the grid in x. For a uniform grid, the size of a grid-division is **(xmax - xmin)/(nx - 1)**. **xmin** defaults to 0, while **xmax** defaults to **nx-1**.

xgrid Specifies whether the grid is uniform or non-uniform on the x-axis. If **True**, the x-coordinates of the grid must be specified following this **instruction**.

ny Specifies the number of grid-points on the y-axis. Minimum value is 2. Required for both uniform and non-uniform grid specification.

ymin

ymax Specifies the limits of the grid in y. For a uniform grid, the size of a grid-division is **(ymax - ymin)/(ny - 1)**. **ymin** defaults to 0, while **ymax** defaults to **ny-1**.

ygrid Specifies whether the grid is uniform or non-uniform on the y-axis. If **True**, the y-coordinates of the grid must be specified following this **instruction**.

binary Specifies whether the following data is written in ASCII or machine-dependent binary. Must be specified individually for x-grid coordinates, y-grid coordinates and surface z-values. If **True**, a double-precision binary array of appropriate size is expected to follow this **instruction**.

joincurve Causes contour curves to be joined at the data-boundaries. This is done by defining a boundary layer around the rectangular border, and setting the z-value of that boundary layer at either the maximum or minimum z-value. **joincurve**="**HIGH**" or **joincurve=2** sets the border z-value to its maximum value: this is useful for plots which have high average z-values. **joincurve**="**LOW**" or **joincurve=1** sets the border z-value to its minimum value: this is useful for plots which have low average z-values. The default value is 0, in which case the contours are not joined at the boundaries.

7.3 Uniform Grid Format Specification

The uniform grid assumes that the x-y points are arranged on a regular rectangular grid bounded by `xmin`, `xmax`, `ymin` and `ymax`. Within this grid, there are `nx` and `ny` grid points in x and y. The CONTOUR format for specifying a uniform grid is as follows:

```
$ DATA=CONTOUR
% nx=<value> [xmin=<value> xmax=<value>]
% ny=<value> [ymin=<value> ymax=<value>]
% [binary=<T/F>]
 z[0,0] z[1,0] z[2,0] ... z[i,0]
 z[0,1] z[1,1] z[2,1] ... z[i,1]
 ...
 z[0,j] z[1,j] z[2,j] ... z[i,j]
$ END
```

The boundary limits and number of grid-points are specified using **instructions**. Note that the boundary limits are optional; if `xmin`, `xmax`, etc. are omitted, `xmin=0` and `xmax=nx-1` values are substituted. The uniform grid boundary specification is followed by `nx` × `ny` z-values on the grid, arranged by column and row. In C programming terms, this format corresponds to printing a 2D array in the following manner:

```
/* j is the outer-most loop */
for (j=0; j<ny; j++)
for (i=0; i<nx; i++)
   (void) printf("%g\n", array[i][j]);
```

The z-values do not all have to be on the same line; they can be specified one value per line as in the above example, or several values per line. Note however that the CONTOUR reader has is a 1000-character limit per line - if too many z-values are printed on the same line, only the first 1000 characters (approximately 100 floating-point numbers) will be read.

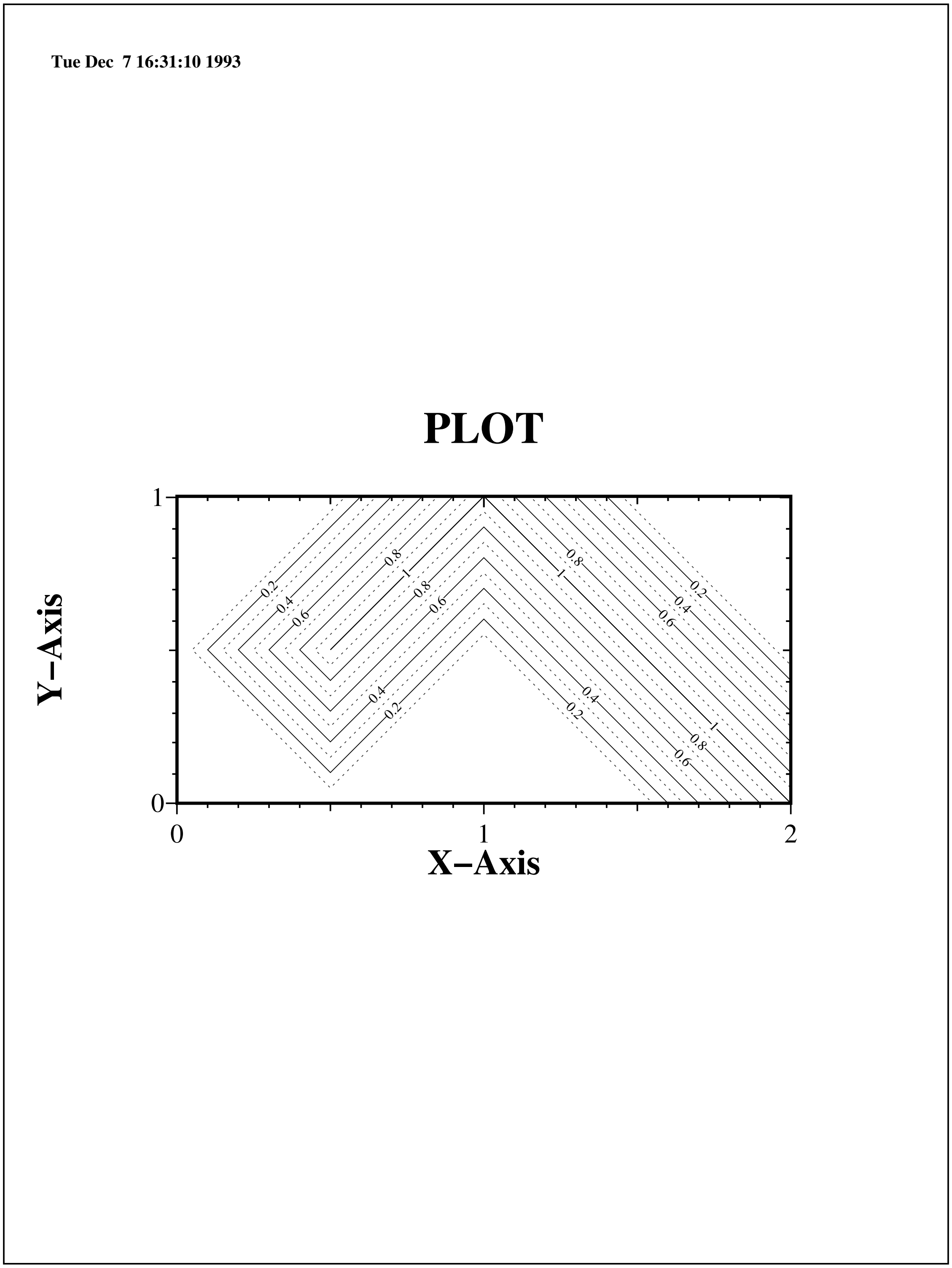
7.4 Example

As an example, the dataset shown below defines a rectangular grid bounded by `xmin=0.0`, `xmax=2.0`, `ymin=0.0` and `ymax=1.0`. There are 5 grid-points in x (`nx=5`) and 3 in y (`ny=3`), corresponding to a uniform square grid 0.5 units in width.

```
$ DATA=CONTOUR
% XMIN=0.0   XMAX=2.0    NX=5
% YMIN=0.0   YMAX=1.0    NY=3
0.0 0.0 0.0 0.0 1.0
0.0 1.0 0.0 1.0 0.0
0.0 0.0 1.0 0.0 0.0
$ END
```

The dataset above produces the mapping shown below. The resultant contours are shown on the following page.

z(x,y)	x=0.0	x=0.5	x=1.0	x=1.5	x=2.0
y=0.0	0.0	0.0	0.0	0.0	1.0
y=0.5	0.0	1.0	0.0	1.0	0.0
y=1.0	0.0	0.0	1.0	0.0	0.0


Tue Dec 7 16:31:10 1993
PLOT
Y−Axis
X−Axis
0
1
2
0.2
0.4
0.6
0.8
1

The z-values can also be specified in a machine-dependent binary format. The machine-dependent binary format is defined by the presence of the "`Binary=True`" string in an **instruction** line. The binary data-stream must begin on the line immediately following the above **instruction** line. The stream consists of a 1-dimensional array of type double-precision. The array-size is `nx` × `ny`. The following code-fragment illustrates writing out binary data in the CONTOUR format:

```
double xarr[20], yarr[20], zarr[400];
int nx=20, ny=20, i, j;

/* Fill x, y arrays */
for (i=0; i<nx; i++) xarr[i] = 0.1*i;
for (j=0; j<ny; j++) yarr[j] = 0.1*j;

/* Fill 1D z-array: note that z[i,j] = zarr[i+nx*j] */
for (i=0; i<nx; i++)
for (j=0; j<ny; j++)
   zarr[i+nx*j] = sqrt(xarr[i]*xarr[i] + yarr[j]*yarr[j]);

/* Write out contour header */
(void) fprintf(fp,"$ DATA=CONTOUR\n");
(void) fprintf(fp,"%% xmin=0.0 xmax=2.0 ymin=0.0 ymax=2.0\n");
(void) fprintf(fp,"%% nx=%d ny=%d\n",nx,ny);

/* Write out contour data */
(void) fprintf(fp,"%% BINARY\n");
if (fwrite((char *)zarr,sizeof(double),nx*ny,fp) != nx*ny) {
   (void) fprintf(stderr,"   ***Binary write error of data array!\n");
   return;
}
(void) fprintf(fp,"$ END\n");
```

This code fragment yields the following:

```
$ DATA=CONTOUR
% xmin=0.0 xmax=2.0 ymin=0.0 ymax=2.0
% nx=20 ny=20
% BINARY
[binary double-precision array][1]
$ END
```

1. The square brackets denote an array; the brackets are not to be written out to the data-stream.

7.5 Non-Uniform Grid Specification

The x-y grid may also be composed of non-uniform rectangles. Non-uniformity is specified using "XGRID=True" and "YGRID=True", together with nx and ny for the number of grid-points. The data then consists of nx points for the x-grid, ny points for the y-grid, followed by nx × ny points for the z-values on the grid. The non-uniform CONTOUR format has the following form:

```
$ DATA=CONTOUR

% nx=<value> xgrid=True [binary=<T/F>]
 x[0] x[1] ... x[i] ... x[nx-1]

% ny=<value> ygrid=True [binary=<T/F>]
 y[0] y[1] ... y[j] ... y[ny-1]

% [binary=<T/F>]
 z[0,0] z[1,0] z[2,0] ... z[i,0]
 z[0,1] z[1,1] z[2,1] ... z[i,1]
 ...
 z[0,j] z[1,j] z[2,j] ... z[i,j]

$ END
```

In the above, the x-grid values **must** be specified before the y-grid values which in turn **must** come before the z-values. Furthermore, the x, y and z-grid arrays can be specified in binary, provided that a "BINARY=True" **instruction** is included before each occurrence of a binary array.

The example below defines a non-uniform rectangular grid bounded by xmin=0.0, xmax=2.0, ymin=0.0 and ymax=1.0. There are 5 grid-points in x (nx=5) and 3 in y (ny=3).

```
$ DATA=CONTOUR

% NX=5 XGRID
0.0 0.2 1.0 1.7 2.0

% NY=3 YGRID
1.0 0.6 0.0

# The z-data
0.0 0.0 0.0 0.0 1.0
0.0 1.0 0.0 1.0 0.0
0.0 0.0 1.0 0.0 0.0
$ END
```

The dataset defined previously results in this mapping:

z(x,y)	x=0.0	x=0.2	x=1.0	x=1.7	x=2.0
y=1.0	0.0	0.0	0.0	0.0	1.0
y=0.6	0.0	1.0	0.0	1.0	0.0
y=0.0	0.0	0.0	1.0	0.0	0.0

The non-uniform grid specification may also be mixed with the uniform format, that is, either of the x or y grid-values can be non-uniform. The example below shows a grid non-uniform in x and uniform in y.

```
$ DATA=CONTOUR

% NX=5 XGRID
0.0 0.2 1.0 1.7 2.0

% NY=3 YMIN=0 YMAX=2

# The z-data
0.0 0.0 0.0 0.0 1.0
0.0 1.0 0.0 1.0 0.0
0.0 0.0 1.0 0.0 0.0
$ END
```

As might be expected, the following mapping is obtained.

z(x,y)	x=0.0	x=0.2	x=1.0	x=1.7	x=2.0
y=0.0	0.0	0.0	0.0	0.0	1.0
y=1.0	0.0	1.0	0.0	1.0	0.0
y=2.0	0.0	0.0	1.0	0.0	0.0

8 4D GRID FORMAT

8.1 Overview

The GRID4D format specifies 4-dimensional data as values on a rectangular volumetric x-y-z grid. The format is an extension of the CONTOUR format from 3D to 4D. As with the CONTOUR format, the data consists of a grid-specification, denoting the points on the x-y-z grid, followed by the main body of t-values at the nodes of the x-y-z grid. The t(x,y,z) values are assumed to be single-valued functions of x, y and z. The rectangular volumetric grid may be uniform or non-uniform. Furthermore, the grid-points as well as the t-values on the grid may be specified in ASCII or machine-dependent binary formats.

Once the data has been read in, the 4D volume is sliced along each of its six orthogonal surfaces; contours of the t-values are plotted on each surface. By default, the contours are plotted in gradated colors with approximately 10 contour intervals between tmin and tmax. The number of contours and their intervals may be set using **Dataset Properties** (see **Appendix A**).

8.2 Resources

The GRID4D format accepts all of the **Plot Resources** listed in **Appendix A**, except for the **Curve Resources** and the **Global Curve Resources**.[1] In addition, the GRID4D dataset accepts the following resources (specified in **instruction** lines):

GRID4D Property Argument List

Name	Type	Default	Range	Description
nx	int	-	-	No. of grid points in x (required)
xmin	double	0.0	$-\infty$ — ∞	Uniform grid boundary
xmax	double	nx - 1	$-\infty$ — ∞	Uniform grid boundary
ny	int	-	-	No. of grid points in y (required)
ymin	double	0.0	$-\infty$ — ∞	Uniform grid boundary
ymax	double	ny - 1	$-\infty$ — ∞	Uniform grid boundary
nz	int	-	-	No. of grid points in z (required)
zmin	double	0.0	$-\infty$ — ∞	Uniform grid boundary
zmax	double	nz - 1	$-\infty$ — ∞	Uniform grid boundary
binary	Boolean	False	True/False	Machine-dependent binary

1. At present, it is not possible to manually set curve properties (e.g., linetype, linewidth) of the contour curves derived from a GRID4D dataset.

nx Specifies the number of grid-points on the x-axis. Minimum value is 2. Required for both uniform and non-uniform grid specification.

xmin

xmax Specifies the limits of the grid in x. For a uniform grid, the size of a grid-division is **(xmax - xmin)/(nx - 1)**. **xmin** defaults to 0, while **xmax** defaults to **nx-1**.

xgrid Specifies whether the grid is uniform or non-uniform on the x-axis. If **True**, the x-coordinates of the grid must be specified following this **instruction**.

ny Specifies the number of grid-points on the y-axis. Minimum value is 2. Required for both uniform and non-uniform grid specification.

ymin

ymax Specifies the limits of the grid in y. For a uniform grid, the size of a grid-division is **(ymax - ymin)/(ny - 1)**. ymin defaults to 0, while **ymax** defaults to **ny-1**.

ygrid Specifies whether the grid is uniform or non-uniform on the y-axis. If **True**, the y-coordinates of the grid must be specified following this **instruction**.

nz Specifies the number of grid-points on the z-axis. Minimum value is 2. Required for both uniform and non-uniform grid specification.

zmin

zmax Specifies the limits of the grid in z. For a uniform grid, the size of a grid-division is **(zmax - zmin)/(nz - 1)**. **zmin** defaults to 0, while **zmax** defaults to **nz-1**.

zgrid Specifies whether the grid is uniform or non-uniform on the z-axis. If **True**, the z-coordinates of the grid must be specified following this **instruction**.

binary Specifies whether the following data is written in ASCII or machine-dependent binary. Must be specified individually for x-grid coordinates, y-grid coordinates, z-grid coordinates and volume t-values. If **True**, a double-precision binary array of appropriate size is expected to follow this **instruction**.

8.3 Format Specification

The grid divisions in x, y, and z may be specified uniformly or non-uniformly. The following **instruction** line is used to specify a uniform distribution in the x-axis.

```
% nx=<value> [xmin=<value> xmax=<value>]
```

To specify a non-uniform grid in x, specify "XGRID=True" in an instruction line, followed by the x-grid values. The x-grid values can be specified in binary, provided a "BINARY=True" **instruction** is included before the binary data-stream.

```
% nx=<value> xgrid=True [binary=<T/F>]
 x[0] x[1] ... x[i] ... x[nx-1]
```

The y and z grid values are specified in a similar fashion.

The t-values on the volumetric grid are specified next, either in ASCII or in binary. The GRID4D format for a uniform grid has the following form:

```
$ DATA=GRID4D
% nx=<value> [xmin=<value> xmax=<value>]
% ny=<value> [ymin=<value> ymax=<value>]
% nz=<value> [zmin=<value> zmax=<value>]

 t[0,0,0] t[1,0,0] t[2,0,0] ... t[i,0,0]
 t[0,1,0] t[1,1,0] t[2,1,0] ... t[i,1,0]
 ...
 t[0,j,0] t[1,j,0] t[2,j,0] ... t[i,j,0]

 t[0,0,1] t[1,0,1] t[2,0,1] ... t[i,0,1]
 t[0,1,1] t[1,1,1] t[2,1,1] ... t[i,1,1]
 ...
 t[0,j,1] t[1,j,1] t[2,j,1] ... t[i,j,1]

 ...
 t[0,0,k] t[1,0,k] t[2,0,k] ... t[i,0,k]
 t[0,1,k] t[1,1,k] t[2,1,k] ... t[i,1,k]
 ...
 t[0,j,k] t[1,j,k] t[2,j,k] ... t[i,j,k]
$ END
```

In C programming terms, this format corresponds to printing a 3D array in the following manner:

```
/* k is the outer-most loop */
for (k=0; k<nz; k++)
for (j=0; j<ny; j++)
for (i=0; i<nx; i++)
   (void) printf("%g\n", array[i][j][k]);
```

8.4 Example

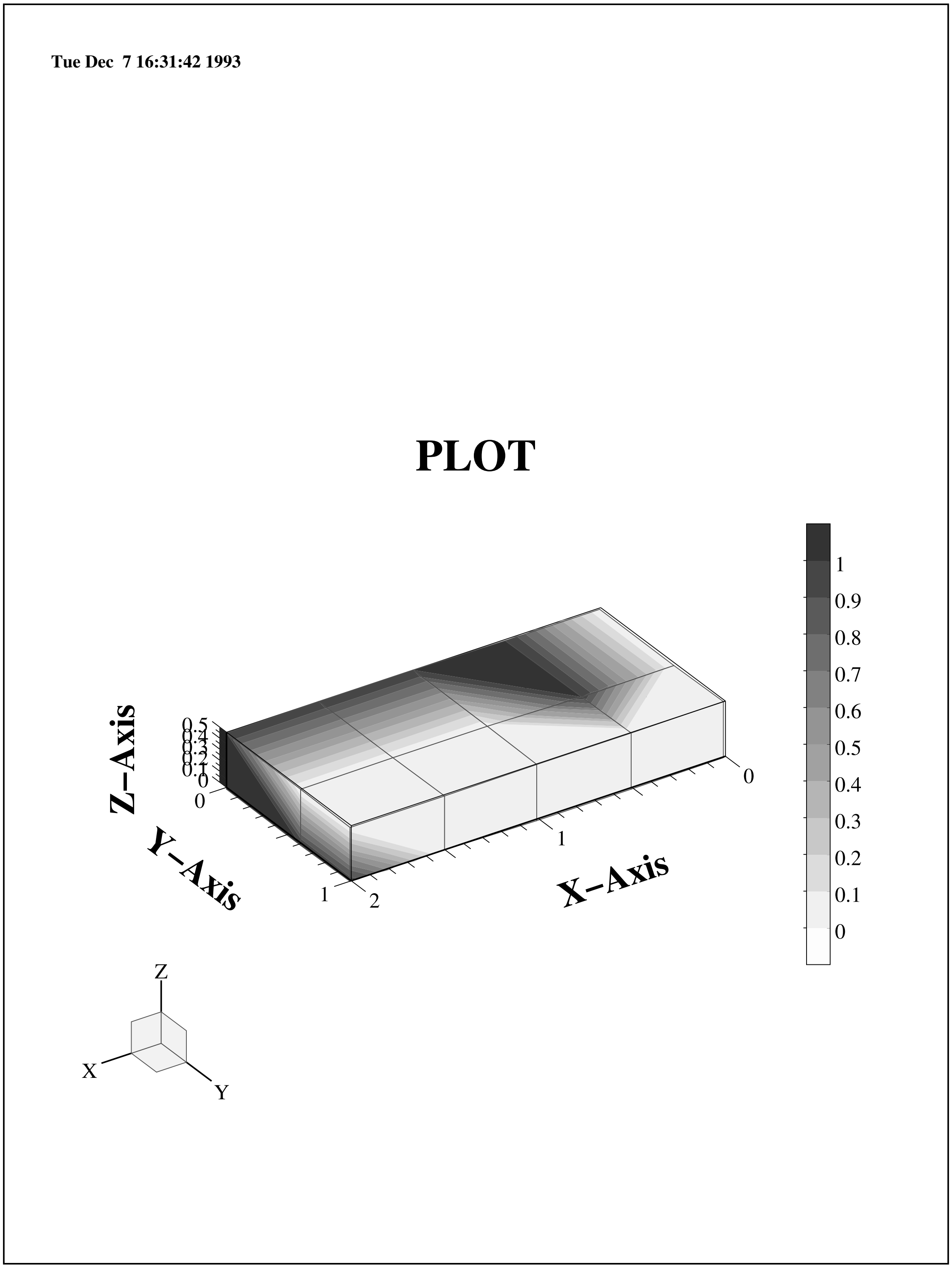
The following example defines a GRID4D dataset bounded by `xmin=0.0`, `xmax=2.0`, `ymin=0.0`, `ymax=1.0`, `zmin=0.0` and `zmax=0.5`.

```
$ DATA = GRID4D

% axisscale=f
% meshplot

# Regular grid in x
% NX=5 XMIN=0.0 XMAX=2.0

# Irregular grid in y
% NY=3 YGRID=True
1.0 0.6 0.0

# Regular grid in z
% NZ=2 ZMIN=0.0 ZMAX=0.5

# The t-data for z=0.0
0.0 0.0 0.0 0.0 1.0
0.0 0.0 0.0 0.0 1.0
1.0 0.0 0.0 0.0 1.0

# The t-data for z=0.5
0.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0
0.0 1.0 1.0 1.0 1.0
$ END
```

The above results in the following mappings:

t(x,y,z),z=0.0	x=0.0	x=0.5	x=1.0	x=1.5	x=2.0
y=1.0	0.0	0.0	0.0	0.0	1.0
y=0.6	0.0	0.0	0.0	0.0	1.0
y=0.0	1.0	0.0	0.0	0.0	1.0

t(x,y,z),z=0.5	x=0.0	x=0.5	x=1.0	x=1.5	x=2.0
y=1.0	0.0	0.0	0.0	0.0	0.0
y=0.6	0.0	1.0	0.0	0.0	0.0
y=0.0	0.0	1.0	1.0	1.0	1.0


Tue Dec 7 16:31:42 1993
PLOT
Z−Axis
Y−Axis
X−Axis
0
0.1
0.2
0.3
0.4
0.5
1
2
0
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Z
X
Y

9 VECTOR FORMAT

9.1 Overview

The VECTOR format specifies a set of 3D vectors. Each vector is specified as a physical point, and a direction vector attached to that point. Both the point and the direction vector are specified in 3D.

Once the vectors have been read in, a scale-factor is calculated based on the magnitude of the largest vector in the vector dataset, and the boundary limits of the dataset; the vectors in the plot are drawn using this scale-factor. By default all vectors of sufficient length (as determined by dimensions on the plot) are plotted with arrow-heads. The vector scale-factor and the shape of the vector can be controlled using **Dataset Properties** (see **Appendix A**).

9.2 Resources

The VECTOR format accepts all of the **Plot Resources** listed in **Appendix A**, except for the **Global Curve Resources**. Use **Curve Resources** to set the line-type, color, and width of the vector arrows. In addition, the VECTOR dataset accepts the following resources (specified in **instruction** lines):

VECTOR Property Argument List

Name	Type	Default	Range	Description
binary	Boolean	False	True/False	Machine-dependent binary
npts	int	-	-	Binary array size

binary Specifies whether the curve data is written in ASCII or machine-dependent binary format. If **True**, the program tries to read 6 double-precision binary arrays of size **npts** containing the x, y and z coordinates and vx, vy and vz components of the vectors. The arrays are assumed to be on the line following the **binary=True instruction**. Default is **False**.

npts Specifies the number of vectors to be read. Used only for reading data in binary format.

9.3 Format Specification

Each vector consists of a physical point (x, y, z) and a direction vector (vx, vy, vz). The vectors can be specified either in ASCII or in binary. The ASCII format requires the coordinates and direction-vector of each vector to be specified one per line as follows:

```
x-coord y-coord z-coord vx-component vy-component vz-component # V 0
x-coord y-coord z-coord vx-component vy-component vz-component # V 1
 ...
x-coord y-coord z-coord vx-component vy-component vz-component # V n
```

In contrast, the binary format requires the coordinates and vector components of all the vectors to be written out separately in 6 different arrays:

```
% binary=True npts=<number-of-vectors>
[x-coordinate binary double-precision array]1
[y-coordinate binary double-precision array]
[z-coordinate binary double-precision array]
[vx-component binary double-precision array]
[vy-component binary double-precision array]
[vz-component binary double-precision array]
```

The arrays must be preceded by an **instruction** line containing the `binary` flag and the number of vectors contained in the arrays.

The VECTOR format thus has the following format:

```
$ DATA=VECTOR
# ASCII format
% [optional instructions]
x1 y1 z1 vx1 vy1 vz1  # vector 1
x2 y2 z1 vx2 vy2 vz2  # vector 2
 ...
xn yn zn vxn vyn vzn  # vector n

# BINARY format
% [optional instructions]
% binary=True npts=<number-of-vectors>
[x-coordinate binary double-precision array]†
[y-coordinate binary double-precision array]
[z-coordinate binary double-precision array]
[vx-component binary double-precision array]
[vy-component binary double-precision array]
[vz-component binary double-precision array]
$ END
```

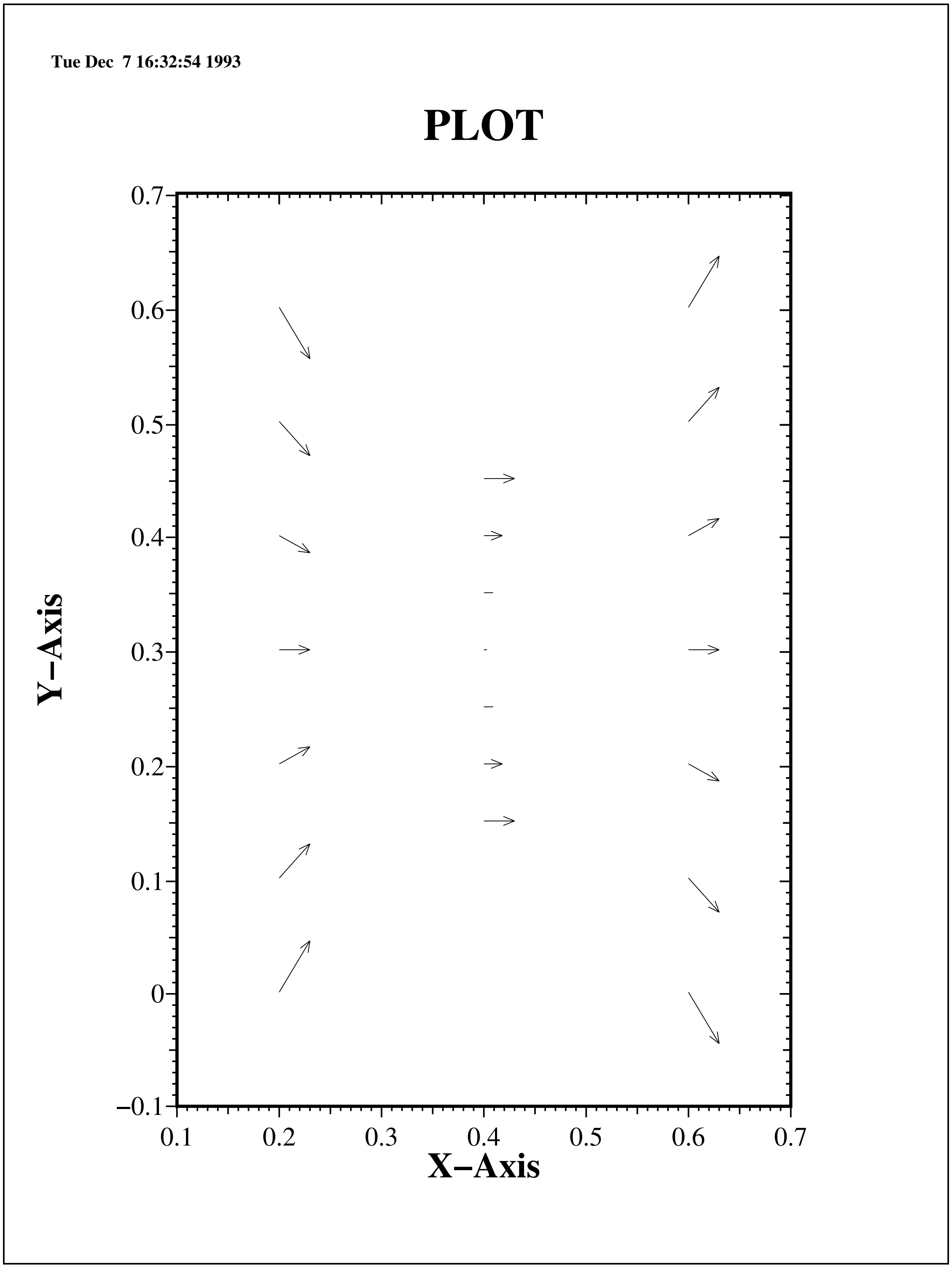
1. The square brackets denote an array; the brackets are not to be written out to the data-stream.

Note that it is possible to mix ASCII and binary formats; some vectors can be specified in ASCII while others can be specified in binary.

9.4 Example

The following shows an example file containing a VECTOR dataset.

```
#
# Data file for vector
#
$ DATA=VECTOR

# Set the plot boundaries explicitly
% xmin =  0.1   xmax =  0.7
% ymin = -0.1   ymax =  0.7
% zmin = -0.5   zmax =  0.5

# Make the vectors a little longer
% vscale = 0.015

# Default vector color is 0
# Default vector width is 1

# 1st set of vectors at x=0.2
0.2 0.0 0.0   2.0  3.0 0.0
0.2 0.1 0.0   2.0  2.0 0.0
0.2 0.2 0.0   2.0  1.0 0.0
0.2 0.3 0.0   2.0  0.0 0.0
0.2 0.4 0.0   2.0 -1.0 0.0
0.2 0.5 0.0   2.0 -2.0 0.0
0.2 0.6 0.0   2.0 -3.0 0.0

# 2nd set of vectors at x=0.4 (parallel to x-axis)
0.4 0.15 0.0   2.0  0.0 0.0
0.4 0.20 0.0   1.2  0.0 0.0
0.4 0.25 0.0   0.6  0.0 0.0
0.4 0.30 0.0   0.2  0.0 0.0
0.4 0.35 0.0   0.6  0.0 0.0
0.4 0.40 0.0   1.2  0.0 0.0
0.4 0.45 0.0   2.0  0.0 0.0

# 3rd set of vectors at x=0.6
0.6 0.0 0.0   2.0 -3.0 0.0
0.6 0.1 0.0   2.0 -2.0 0.0
0.6 0.2 0.0   2.0 -1.0 0.0
0.6 0.3 0.0   2.0  0.0 0.0
0.6 0.4 0.0   2.0  1.0 0.0
0.6 0.5 0.0   2.0  2.0 0.0
0.6 0.6 0.0   2.0  3.0 0.0

$ END
```


Tue Dec 7 16:32:54 1993
PLOT
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−0.1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
X−Axis
Y−Axis

9.5 Programming Example

The following short program illustrates writing binary data in the VECTOR format.

```
#include <stdio.h>
#include <math.h>
main()
{
#define BINARYFILE "data.mtvdat"
   FILE   *fp;
   double xarr[100], yarr[100], zarr[100];
   double vxarr[100], vyarr[100], vzarr[100];
   int    i, npts=100;

   /* Open up a file */
   if ((fp=fopen(BINARYFILE,"w")) == NULL) {
      (void) fprintf(stderr,"cat: Couldn’t open file %s\n",BINARYFILE);
      exit(-1);
   }

   /* Fill arrays */
   for (i=0; i<npts; i++) {
      xarr[i] = 0.1*i;          vxarr[i] = xarr[i];
      yarr[i] = cos(xarr[i]);   vyarr[i] = yarr[i];
      zarr[i] = sin(xarr[i]);   vzarr[i] = zarr[i];
   }

   /* Write out vector header */
   (void) fprintf(fp,"$ DATA=VECTOR\n");

   /* Write out binary data */
   (void) fprintf(fp,"%% BINARY npts=%d\n",npts);
   if (fwrite((char *)xarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1); }
   if (fwrite((char *)yarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1); }
   if (fwrite((char *)zarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1); }
   if (fwrite((char *)vxarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1); }
   if (fwrite((char *)vyarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1); }
   if (fwrite((char *)vzarr,sizeof(double),npts,fp) != npts) {
      (void) fprintf(stderr,"   ***Binary write error of data array!\n");
      exit(-1); }
   (void) fprintf(fp,"$ END\n");
   (void) fclose(fp);   /* Close the file */
}
```

10 PROBABILITY PLOT FORMAT

10.1 Overview

The PROBABILITY-PLOT format is used to specify a list of 2D points to be plotted on a probability plot. Each point is specified as a value and an optional probability of occurrence; the x-axis of the resultant plot is on an arithmetic scale, while the y-axis shows the distance between percentages obtained from a normal distribution. A PROBABILITY-PLOT dataset contains only one curve.

10.2 Resources

The PROBABILITY-PLOT accepts all of the **Plot Resources** listed in **Appendix A**, except for the **Global Curve Resources**. By default, the points in the dataset are plotted with square markers (`markertype=4`), and the points are not joined by a line (`linetype=0`). Use **Curve Resources** for changing these default curve properties. In addition, the PROBABILITY-PLOT format accepts the following resources:

PROBABILITY-PLOT Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
readprob	Boolean	False	True/False	Read probability of occurrence

readprob Probability data can be specified as a single list of unordered x-values, in which case the data is sorted and a cumulative probability (y-value) is assigned to each data-point. This corresponds to **readprob=False**. Alternately, the data may be specified as a list of (x, y) pairs, where x is the data value, and y is the user-specified cumulative probability. In this last scheme, **readprob=True**, and 2 columns of data must be specified.

10.3 Format Specification

PROBABILITY_PLOT data may be specified in either a single column or double column format.

10.3.1 Single Column Format: Compute Probabilities

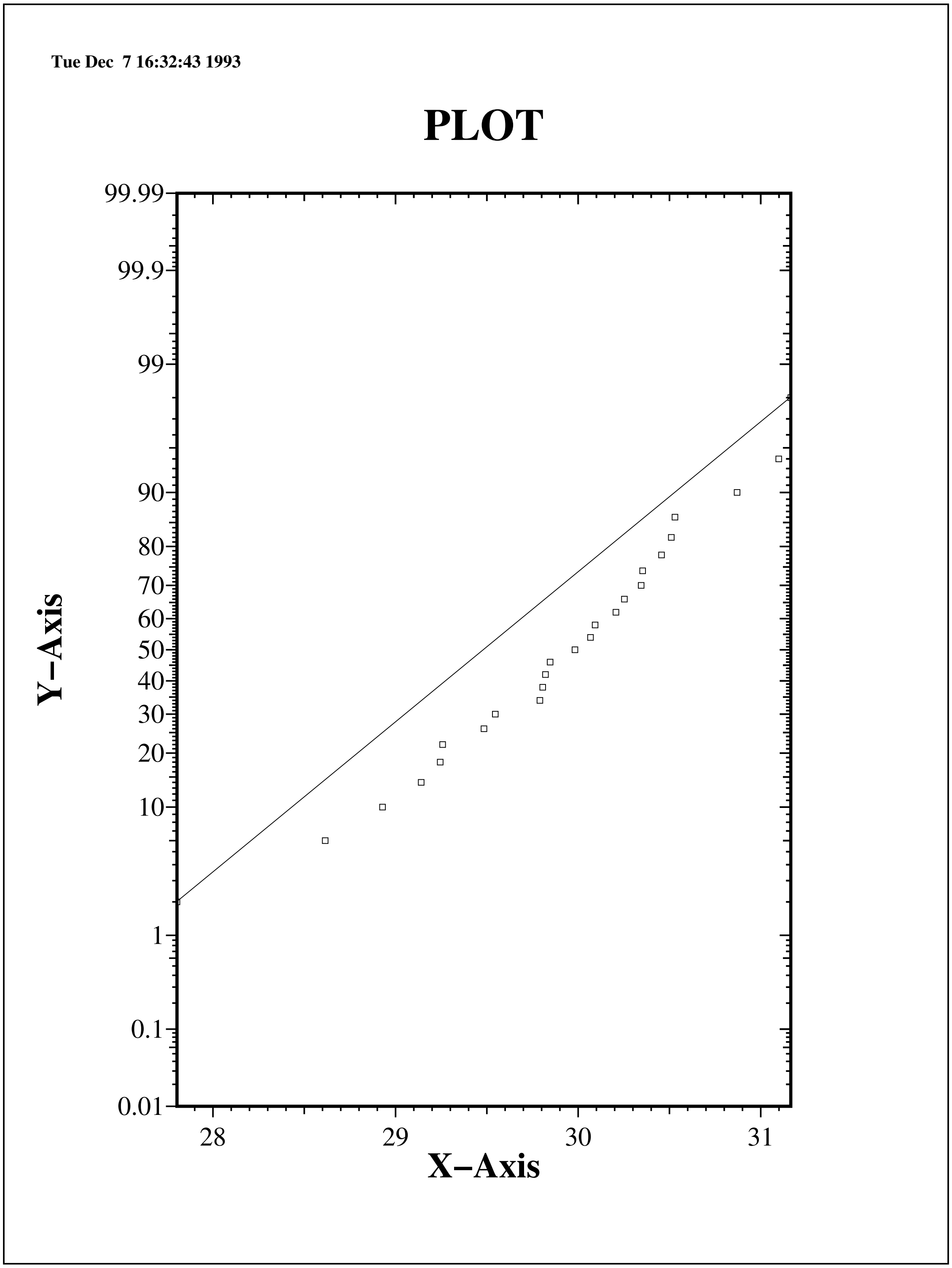
In the single column format, which corresponds to **readprob=False**, the data consists of a single column of x-values. These x-values are read in, then sorted by magnitude. The cumulative probability of occurrence of these data-points is then calculated and assigned as the y-value of each data-point, using the simple function `y(i)=(2i - 1)/(2n)`, where `i` is the rank order of the data-point, and `n` is the total number of points.

The single-column format has the following form:

```
$ DATA=PROBABILITY

% [optional instructions]

% readprob=False
x1     #[point 1]
x2     #[point 2]
 ...
xn     #[point n]

$ END
```

An example data-file in the single-column format is shown below. The resultant MTV plot is shown on the following page.

```
$ DATA=PROBABILITY

#
# This format provides a sequence of numbers, from which an ordered list
# and the corresponding probabilities are calculated by the program
#
% readprob = False
2954.63e-2
2980.58e-2
3053.02e-2
3009.29e-2
3116.46e-2

3025.33e-2
2861.48e-2
2979.04e-2
3109.85e-2
3045.65e-2

2984.65e-2
2892.86e-2
3051.01e-2
3006.77e-2
2925.79e-2

3034.51e-2
3020.70e-2
2948.42e-2
2998.25e-2
3035.31e-2

2780.27e-2
2914.04e-2
2924.46e-2
3087.05e-2
2982.11e-2

# Draw a straight line through the data
@line x1=27.8 y1=0.02 x2=31.16 y2=0.98

$ END
```


Tue Dec 7 16:32:43 1993
PLOT
99.99
99.9
99
90
80
70
60
50
40
30
20
10
1
0.1
0.01
28
29
30
31
X−Axis
Y−Axis

10.3.2 Double Column Format: Read Probabilities

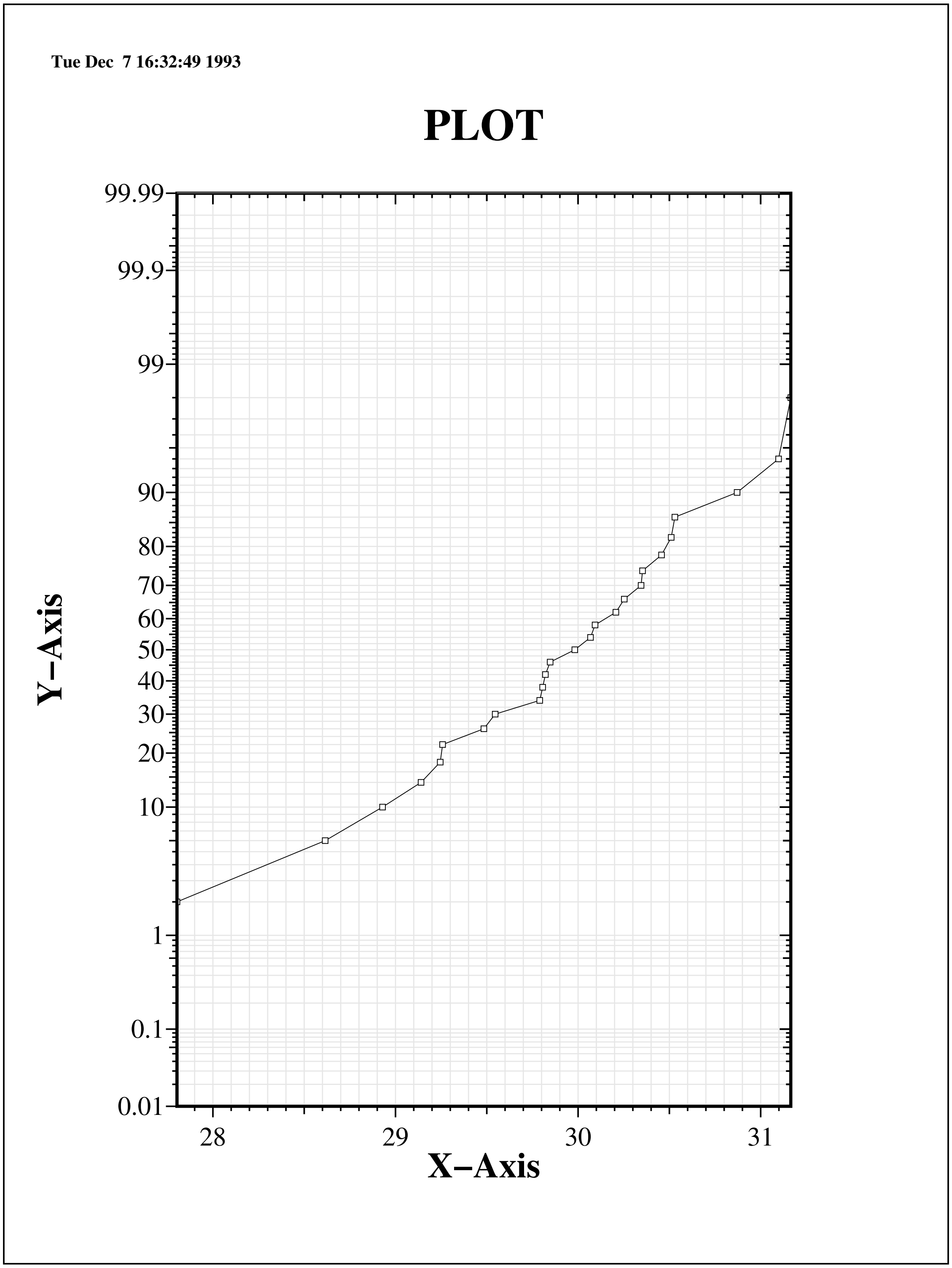
In the double column format, which corresponds to **readprob=True**, the data consists of 2 columns of (x, y) pairs. The x-value corresponds to the data-value, while the y-value is the cumulative probability corresponding to the data-value. The probability is specified in a range of 0.0 - 1.0. The data is read in and stored in order of occurrence; unlike the single-column format described earlier, in the double column format, the data is **not** sorted in x.

The double-column format has the following form:

```
$ DATA=PROBABILITY

% [optional instructions]

% readprob=True
x1 y1    #[point 1]
x2 y2    #[point 2]
 ...
xn yn    #[point n]

$ END
```

An example data-file in the double-column format is shown below. The resultant MTV plot is shown on the following page.

```
$ DATA=PROBABILITY

#
# This format provides a sequence of numbers, with values and cumulative
# probabilities
#

% grid=True      # Draw the grid too
% readprob=True
% linetype=1     # Draw a line through the data-points
2780.3e-2  0.02
2861.5e-2  0.06
2892.9e-2   0.1
  2914e-2  0.14
2924.5e-2  0.18
2925.8e-2  0.22
2948.4e-2  0.26
2954.6e-2   0.3
  2979e-2  0.34
2980.6e-2  0.38
2982.1e-2  0.42
2984.7e-2  0.46
2998.2e-2   0.5
3006.8e-2  0.54
3009.3e-2  0.58
3020.7e-2  0.62
3025.3e-2  0.66
3034.5e-2   0.7
3035.3e-2  0.74
3045.7e-2  0.78
  3051e-2  0.82
  3053e-2  0.86
3087.1e-2   0.9
3109.8e-2  0.94
3116.5e-2  0.98

$ END
```


Tue Dec 7 16:32:49 1993
PLOT
99.99
99.9
99
90
80
70
60
50
40
30
20
10
1
0.1
0.01
28
29
30
31
X−Axis
Y−Axis

11 HISTOGRAM FORMAT

11.1 Overview

The HISTOGRAM format is used to create a histogram plot where the frequency of occurrence of data in specified ranges is plotted as 2D bars. The horizontal axis of the plot represents ranges of data-point values (e.g., 0-1, 1-2, 2-3) while the vertical axis is scaled according to the number of data-points falling into each data-range or bin (e.g., 5 points in [0,1] range). Histogram data is specified as a single column of data-values.

11.2 Resources

The HISTOGRAM format accepts all of the **Plot Resources** listed in **Appendix A**, except for the **Global Curve Resources**. Each histogram bar is plotted as a solid rectangle with a default fillcolor (`fillcolor=5`). The linetype, line thickness and other such curve properties of the bars can be changed using **Curve Resources**. The HISTOGRAM format also accepts the following **Dataset Resources**:

HISTOGRAM Property Argument List

Name	Type	Default	Range	Description
binstart	double	xmin	$-\infty$ — ∞	Histogram starting point
binwidth	double	0.1(xmax-xmin)	$-\infty$ — ∞	Histogram bin size

binstart Specifies the beginning point from which the data-values will be collected into bars. The data is collected in ranges of size **binwidth** beginning from `x`=**binstart**. Thus the histogram bars are constructed over the ranges [**binstart**, **binstart** + **binwidth**], [**binstart** + **binwidth**, **binstart** + **2**×**binwidth**] and so on. **binstart** defaults to `xmin`, the minimum data value.

binwidth Specifies the collection range or bin size for the histogram. The data is collected in ranges of size **binwidth** beginning from `x`=**binstart**. Thus the histogram bars are constructed over the ranges [**binstart**, **binstart** + **binwidth**], [**binstart** + **binwidth**, **binstart** + **2**×**binwidth**] and so on. **binwidth** defaults to `0.1×(xmax−xmin)`, such that 10 histogram bars are plotted.

11.3 ASCII Format Specification

HISTOGRAM data consists of a single column of x-values. These x-values are read in and sorted into bins, i.e., the number of data-points falling into the various data-ranges are counted. Bars are constructed with heights proportional to the data count, and widths proportional to the data range.

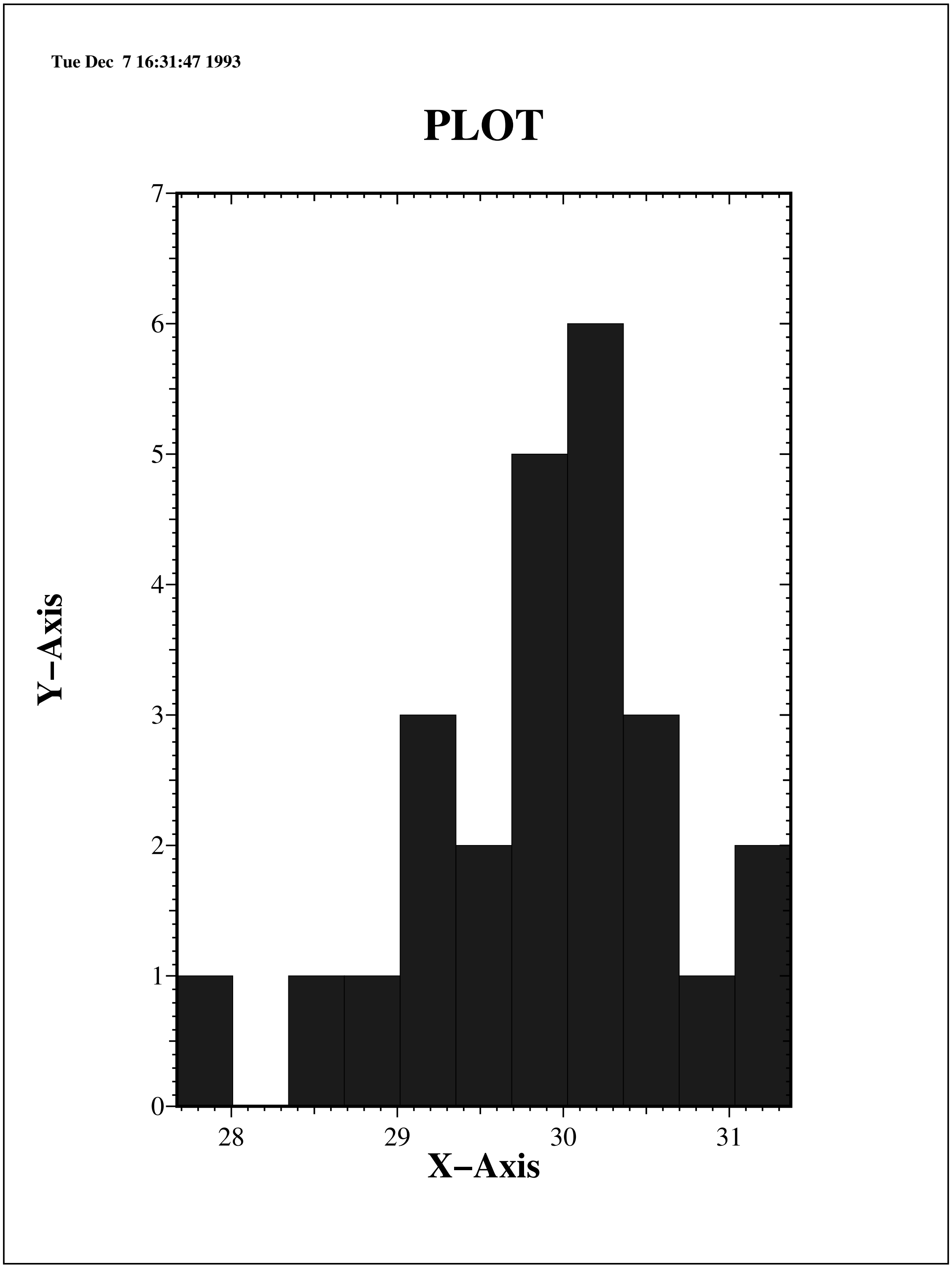
The HISTOGRAM format has the following form:

```
$ DATA=HISTOGRAM

% [optional instructions]

x1     #[point 1]
x2     #[point 2]
x3     #[point 3]
 ...
xn     #[point n]

$ END
```

11.4 Example

An example of a HISTOGRAM dataset is shown below.

```
$ DATA=HISTOGRAM

#
# This format provides a sequence of numbers to be plotted as a histogram
#

#% binwidth=1.0
#% binstart=28.00

2954.63e-2
2980.58e-2
3053.02e-2
3009.29e-2
3116.46e-2

3025.33e-2
2861.48e-2
2979.04e-2
3109.85e-2
3045.65e-2

2984.65e-2
2892.86e-2
3051.01e-2
3006.77e-2
2925.79e-2

3034.51e-2
3020.70e-2
2948.42e-2
2998.25e-2
3035.31e-2

2780.27e-2
2914.04e-2
2924.46e-2
3087.05e-2
2982.11e-2

$ END
```


Tue Dec 7 16:31:47 1993
PLOT
7
6
5
4
3
2
1
0
28
29
30
31
X−Axis
Y−Axis

12 BARCHART FORMAT

12.1 Overview

The BARCHART format specifies multiple sets of data points on a bar graph. Each data point is represented by a two-dimensional bar. The bottom horizontal axis represents dataset values (e.g., months). The bars are plotted against the vertical axis, which is scaled according to the range of data-point values (e.g., sales by month). More than one set of bars or data-point values may be specified and plotted side-by-side (e.g., sales by month, gross profits by month).

12.2 Resources

The BARCHART format accepts all of the **Plot Resources** listed in **Appendix A**, except for the **Global Curve Resources**. Each bar is plotted as a solid rectangle with a fillcolor dependent on the order of bar sets. The linetype, line thickness and other such curve properties of the bars can be changed using **Curve Resources**. In addition, the fillcolor and filltype of different bar sets may be specified using **instruction** lines followed by a list of values corresponding to each bar (see examples on the following pages). The BARCHART format also accepts the following **Dataset Resources**:

BARCHART Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
barmin	double	0.0	$-\infty$ — ∞	Minimum bar extent

barmin Specifies the minimum y-extent of each bar in the bar-chart. The 2D bar is drawn from `y=`**barmin** to `y=yval`, where `yval` is the data-point value of the bar. By default, **barmin**=0.0.

12.3 ASCII Format Specification

The BARCHART format has the following form:

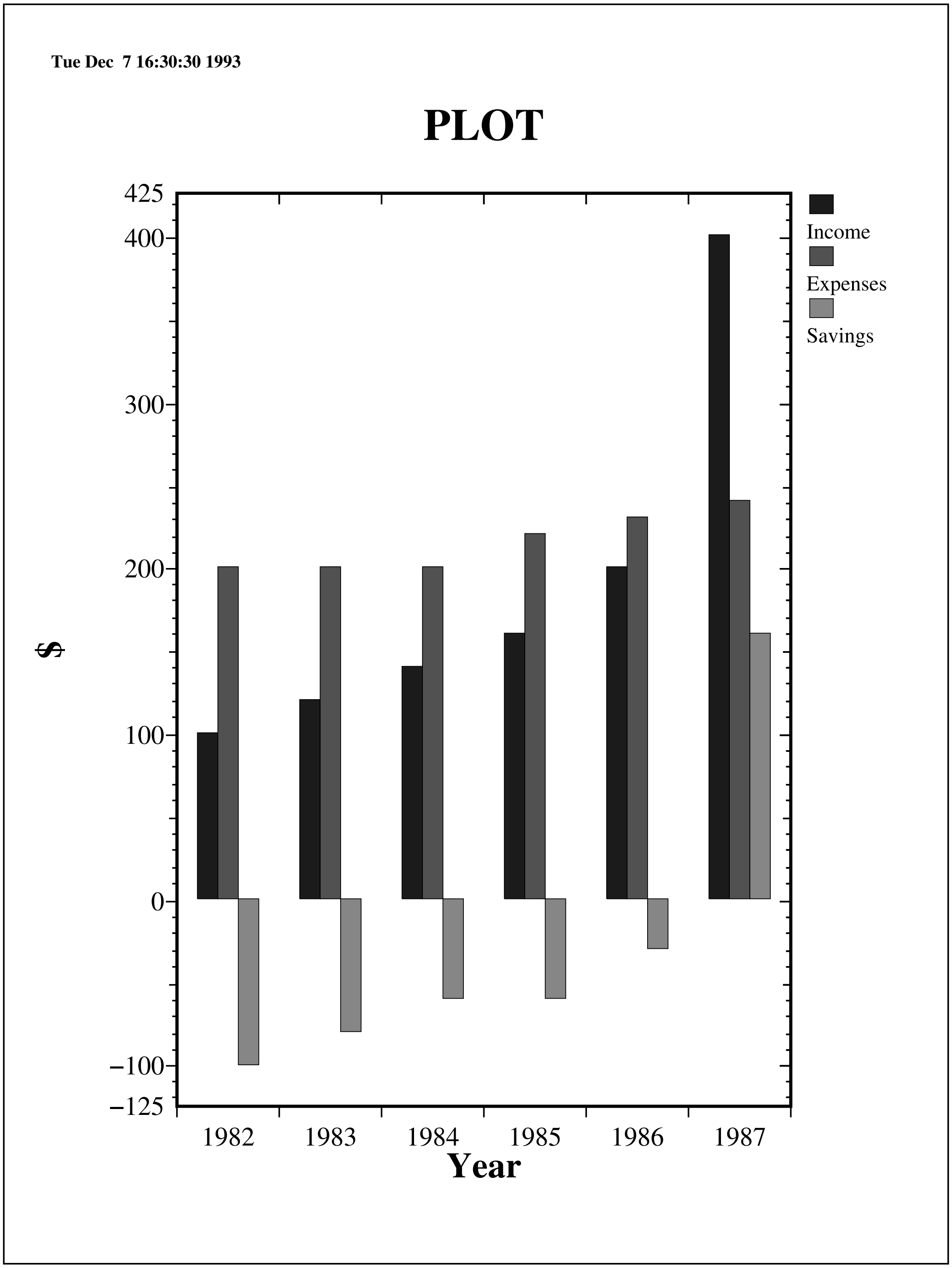
```
$ DATA=BARCHART

% [optional instructions]

          "Bar-Set-1"  "Bar-Set-2"  "Bar-Set-3" ... "Bar-Set-m"
"value-1"   xa1          xb1          xc1               xm1
"value-2"   xa2          xb2          xc2               xm2
 ...
"value-n"   xan          xbn          xcn               xmn
% fillcolor color-1     color-2    color-3       color-m   # optional
% filltype   type-1      type-2     type-3        type-m    # optional

$ END
```

As shown above, BARCHART data is specified as multiple columns of data. The first line in the dataset is used to determine the number of sets of bars as well as the names of each set (e.g., "`Sales`", "`Profit`"). Each subsequent line consists of the name of the data-value (e.g., "`Month`"), and the value for each set of data. The filltype and fillcolor for each set of data may also be specified in columnar form as shown.

12.4 Example

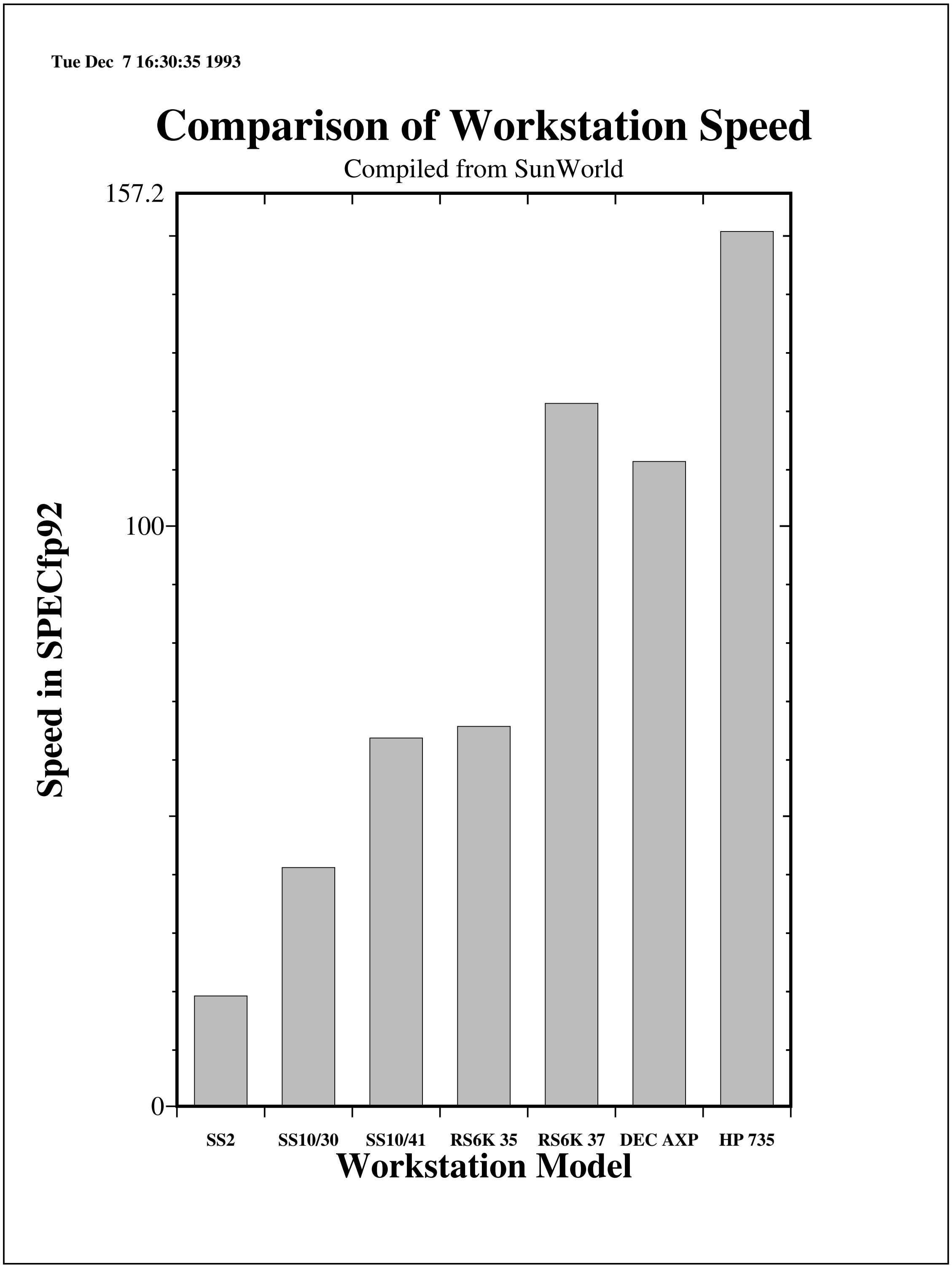
An example of a BARCHART dataset containing 3 sets of bars is shown below.

```
$ DATA=BARCHART

# This format is used for the bar-chart
#

#% barmin = -100     # Could use this to set the bar minimum value
% xlabel = "Year"
% ylabel = "$"

         "Income"   "Expenses"   "Savings"
1982       100        200          -100
1983       120        200           -80
1984       140        200           -60
1985       160        220           -60
1986       200        230           -30
1987       400        240           160
% fillcolor  5          4             3

$ END
```


Tue Dec 7 16:30:30 1993
PLOT
425
400
300
200
100
0
−100
−125
$
1982
1983
1984
1985
1986
1987
Year
Income
Expenses
Savings

The example below demonstrates the use of a BARCHART dataset with only one set of bars.

```
$ DATA=BARCHART

# A single set of bars

% xlabel = "Workstation Model"
% ylabel = "Speed in SPECfp92"
% toplabel = "Comparison of Workstation Speed"
% subtitle = "Compiled from SunWorld"
% sidelabel = false   # Don't plot the side labels

               "Speed"
"SS2"            19.0
"SS10/30"        41.1
"SS10/41"        63.4
"RS6K 350"       65.4
"RS6K 370"      121.0
"DEC AXP 400"   111.0
"HP 735"        150.6
$ END
```


Tue Dec 7 16:30:35 1993
Comparison of Workstation Speed
Compiled from SunWorld
157.2
100
0
Speed in SPECfp92
SS2
SS10/30
SS10/41
RS6K 35
RS6K 37
DEC AXP
HP 735
Workstation Model

APPENDIX A

MTV Plot Resources

1 PLOT RESOURCES

MTV Plot Resources are `<argument>=<value>` strings used to set properties of a plot. The Plot Resources modify the properties of the dataset and of any plots containing the dataset. The various types of Plot Resources are:

- Plotset Resources
- 3D View Resources
- Dataset Resources
- Curve Resources
- Global Curve Resources

Plot Resources are included as **instruction** lines in the data-stream along with the actual plot data in the MTVDAT data format. Each **instruction** line begins with a percent "%" character and contains one or more argument-value pairs, separated by spaces or tabs. In general, the argument of the `<argument>=<value>` pair is case-insensitive, while the value string is case-sensitive. The example below defines a CURVE2D dataset with properties set using **instruction** lines.

```
$ DATA=CURVE2D NAME="Curve A"

# Plotset Resources
% toplabel = "Hello World"
% grid = True

# Dataset Resources
% splinetype=3   # Curves connected with Quadratic B-splines

# Curve 1 Resources
% linetype=1 linewidth=2 linelabel="Cv 1"
% markertype=3  # Draw the markers too
  0 0
  1 1
  2 2

# Curve 2 Resources
% linetype=2 linewidth=1 linelabel="Cv 2"
% markertype=4  # Draw the markers too
  2 0
  1 1
  0 2
$ END
```

2 PLOTSET RESOURCES

2.1 Overview

Plotset Resources are used to control the shape, size and appearance of a plot.

Plotset Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Plot Titles				
xlabel	string	"X-Axis"	N/A	X-Axis Label
ylabel	string	"Y-Axis"	N/A	Y-Axis Label
zlabel	string	"Z-Axis"	N/A	Z-Axis Label
toplabel	string	"PLOT"	N/A	Plot Title
subtitle	string	NULL	N/A	Plot Sub-Title
comment	string	NULL	N/A	User-defined Comment
sidelabel	Boolean	True	True/False	Plot side-labels
sidelabellength labeloffset	integer	0	-100 — 300	Increase/decrease area for side-labels
Plot Aspect-ratio/Appearance				
grid	Boolean	False	True/False	Plot grid
equalscale	Boolean	False	True/False	Equal scaling in X and Y
fitpage	Boolean	True	True/False	Scale plot to fit page
xyratio	double	0.75	0.01 — 4	Plot size ratio (Y/X)
Axis-Scales				
xautorange	Boolean	True	True/False	Automatic tick scaling in X
xticks	integer	4	1 — 50	Number of tickmarks in X
yautorange	Boolean	True	True/False	Automatic tick scaling in Y
yticks	integer	4	1 — 50	Number of tickmarks in Y
zautorange	Boolean	True	True/False	Automatic tick scaling in Z
zticks	integer	4	1 — 50	Number of tickmarks in Z

Plotset Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Axis-Scales (continued)				
autorange	Boolean	True	True/False	Automatic tick scaling in X,Y,Z
xlog	Boolean	False	True/False	Logarithmic Scale in X
ylog	Boolean	False	True/False	Logarithmic Scale in Y
zlog	Boolean	False	True/False	Logarithmic Scale in Z
xscale	double	1.0	10^{-99} — ∞	Scale factor for X-Axis Label
yscale	double	1.0	10^{-99} — ∞	Scale factor for Y-Axis Label
zscale	double	1.0	10^{-99} — ∞	Scale factor for Z-Axis Label
xticklabel	string	NULL	N/A	User-specified ticklabels, specified as: xticklabel=(x-coordinate, label)
yticklabel	string	NULL	N/A	User-specified ticklabels, specified as: yticklabel=(y-coordinate, label)
zticklabel	string	NULL	N/A	User-specified ticklabels, specified as: zticklabel=(z-coordinate, label)
innerticks	Boolean	False	True/False	Draw ticks inside plot area
Plot Boundaries				
xmin vxmin	double	Dataset xmin	-∞ — ∞	X-Axis plot boundary
xmax vxmax	double	Dataset xmax	-∞ — ∞	X-Axis plot boundary
ymin vymin	double	Dataset ymin	-∞ — ∞	Y-Axis plot boundary
ymax vymax	double	Dataset ymax	-∞ — ∞	Y-Axis plot boundary
zmin vzmin	double	Dataset zmin	-∞ — ∞	Z-Axis plot boundary

Plotset Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Plot Boundaries (continued)				
zmax vzmax	double	Dataset zmax	-∞ — ∞	Z-Axis plot boundary
Miscellaneous Options				
xflip	Boolean	False	True/False	Flip X-Scale
yflip	Boolean	False	True/False	Flip Y-Scale
xabs	Boolean	False	True/False	Plot against abs(x)
yabs	Boolean	False	True/False	Plot against abs(y)
zabs	Boolean	False	True/False	Plot against abs(z)
overlay	Boolean	True	True/False	Modify overlaid datasets

2.2 Resource Specification

Plot Titles

xlabel Specifies the x-axis label. Defaults to "`X-Axis`".

ylabel Specifies the y-axis label. Defaults to "`Y-Axis`".

zlabel Specifies the z-axis label. Used only in 3D plots. Defaults to "`Z-Axis`".

toplabel Specifies the plot title, which appears above the plot itself. Defaults to "`PLOT`".

subtitle Specifies an additional title, which appears in a smaller font below the plot title. By default this title is left blank.

comment Specifies an additional title. This title appears in the upper-right corner of the plot. By default this title is left blank.

sidelabel Specifies whether the curve labels and contour color labels are to be plotted on the side of the plot. Defaults to **True**.

sidelabelength

labeloffset Specifies the additional size in pixels that the area for the side-labels is to be increased or decreased. Negative values decrease the size of the side-label area. Defaults to **0**.

Plot Aspect-ratio/Appearance

grid Specifies whether a dotted grid is drawn within the 2D plot. Grid intervals depends on the axis-scale values. Defaults to **False**.

equalscale Specifies whether the 2D plot is drawn with equally-scaled x and y axes. If **True**, the plot-distance per unit in x and y are set equal; thus if the x-limits are equal to the y-limits the plot will be square-shaped. Note that if x and y differ significantly in magnitude, setting **equalscale=True** will result in a long narrow rectangular plot. If **equalscale=False**, and **fitpage=True**, the x and y scale factors will be chosen such that the plot fills the page. If **equalscale=False**, and **fitpage=False**, the plot is drawn with aspect ratio **xyratio**. Default **equalscale** value is **True**.
Note: Use the View Resource **axisscale** to control the aspect ratio of a 3D plot.

fitpage Used with **equalscale** to set the 2D plot aspect ratio. Defaults to **True**.

xyratio Specifies the 2D plot y:x aspect ratio, i.e., the ratio of the plot-height (y) to the plot-width (x). Used only if both **equalscale=False** and **fitpage=False**. Default value is 0.75.

Axis-Scales

xautorange Specifies whether automatic ranging is used to plot axis-scales on the x-axis. Automatic ranging causes the program to select the best axis-scales possible for the x-axis. Both major and minor ticks are drawn; tick intervals depend on the range of axis-values. When **False**, the axis is drawn with **xticks** equally-spaced major tickmarks. Defaults to **True**.

xticks Sets the number of major ticks on the x-axis. Specifying this option also turns off automatic ranging of the x-axis. Minor ticks are not drawn. Default value is 4.

yautorange Specifies whether automatic ranging is used to plot axis-scales on the y-axis. See the description of **xautorange**. Defaults to **True**.

yticks Sets the number of major ticks on the y-axis. See the description of **xticks**. Default value is 4.

zautorange Specifies whether automatic ranging is used to plot axis-scales on the z-axis. See the description of **xautorange**. Defaults to **True**.

zticks Sets the number of major ticks on the z-axis. See the description of **xticks**. Default value is 4.

autorange Equivalent to setting **xautorange**, **yautorange**, and **zautorange** simultaneously.

xlog Specifies if the x-axis is to be drawn on a logarithmic or linear scale. When **True**, major tick marks are drawn at every logarithmic decade (multiple of 10.0), while minor tickmarks are drawn at every integer value within the decade. Defaults to **False**.

ylog Specifies if the y-axis is to be drawn on a logarithmic or linear scale. Defaults to **False**.

zlog Specifies if the z-axis is to be drawn on a logarithmic or linear scale. Defaults to **False**.

xscale Specifies the scale factor for the x-axis scale labels. This is useful when the x-ordinate values are very large or very small; the scale values are then divided by **xscale** before being placed on the plot. For example, if the plot has major ticks at 1.2e10 and 1.3e10, and **xscale=1e10**, then the tick labels will show "1.2" and "1.3", and a "(× 1e10)" label will be appended to the x-axis label. Defaults to 1.0.

yscale Specifies the scale factor for the y-axis scale labels. See the description of **xscale**. Defaults to 1.0.

zscale Specifies the scale factor for the z-axis scale labels. See the description of **xscale**. Defaults to 1.0.

xticklabel

yticklabel

zticklabel Specifies the label to be used for a given coordinate on the plot boundary. Ticklabels are psecied as a coordinate-label pair enclosed in brackets:

xticklabel `= ( x-coordinate, label )`
yticklabel `= ( y-coordinate, label )`
zticklabel `= ( z-coordinate, label )`

The following example illustrates the use of successive ticklabel statements to specify labels on the plot boundaries.

```
$ DATA=CURVE2D

# Xticklabel specification
% xticklabel = (0.1 "Tick1")
% xticklabel = (0.2 "Tick2")
% xticklabel = (0.3 "Tick 3")

# Now define curve datapoints
0 0
1 1
2 2

$ END
```

innerticks Forces tickmarks to be drawn only on the inside of the plot. Defaults to **False**, in which case tickmarks on the upper x-axis and on the right y-axis are drawn inside the plot, while the tickmarks on the lower x-axis and on the left y-axis are drawn outside the plot.

Plot Boundaries

xmin
vxmin Sets the minimum x-axis value of the plot boundary.

xmax
vxmin Sets the maximum x-axis value of the plot boundary.

ymin
vymin Sets the minimum y-axis value of the plot boundary.

ymax
vymin Sets the maximum y-axis value of the plot boundary.

zmin
vzmin Sets the minimum z-axis value of the plot boundary.

zmax
vzmax Sets the maximum z-axis value of the plot boundary.

Miscellaneous Options

xflip Specifies whether the direction of increasing x is to be reversed. If **False**, the minimum x-value is placed to the left of the maximum x-value on the plot; x increases from left to right. If **True**, the minimum x-value is placed to the right of the maximum x, and x increases from right to left. This gives the appearance of flipping the plot about the x-axis. Defaults to **False**.

yflip Specifies whether the direction of increasing y is to be reversed. If **False**, the minimum y-value is placed underneath the maximum y-value on the plot; y increases from bottom to top. If **True**, the minimum y-value is placed above the maximum y, and y increases from top to bottom. This gives the appearance of flipping the plot about the y-axis. Defaults to **False**.

xabs Specifies if the data is to be plotted against the absolute value of the x-ordinate. Defaults to **False**.

yabs Specifies if the data is to be plotted against the absolute value of the y-ordinate. Defaults to **False**.

zabs Specifies if the data is to be plotted against the absolute value of the z-ordinate. Defaults to **False**.

overlay Specifies if datasets overlaid in the same plot are to be modified. If **True**, the line-types and line-colors of contours and curves are jogged so as to provide contrast between the different datasets. Defaults to **True**.

3 3D VIEW RESOURCES

3.1 Overview

View Resources affect the shape and viewing angle of the 3D plot. These resources are used to select the appropriate view of the 3D plot.

3D View Property Argument List

Name	Type	Default	Range	Description
View Point				
eyepos.x	double	1.0	$-\infty$ — ∞	Eye-position relative to view center
eyepos.y	double	1.5	$-\infty$ — ∞	Eye-position relative to view center
eyepos.z	double	0.5	$-\infty$ — ∞	Eye-position relative to view center
viewcenter.x	double	0.5(xmin+xmax)	$-\infty$ — ∞	View center
viewcenter.y	double	0.5(ymin+ymax)	$-\infty$ — ∞	View center
viewcenter.z	double	0.5(zmin+zmax)	$-\infty$ — ∞	View center
Axis Options				
window.xmin	double	-0.7	$-\infty$ — ∞	Window boundary
window.xmax	double	0.7	$-\infty$ — ∞	Window boundary
window.ymin	double	-0.7	$-\infty$ — ∞	Window boundary
window.ymax	double	0.7	$-\infty$ — ∞	Window boundary
axislabel	Boolean	True	True/False	Draw axis labels
axismove	Boolean	False	True/False	Rotate axis with view
axisscale	Boolean	True	True/False	Apply unequal scaling (plot as cube)
xaxisscale	double	1.0	10^{-99} — ∞	Scale of x-axis relative to y and z
yaxisscale	double	1.0	10^{-99} — ∞	Scale of y-axis relative to x and z
zaxisscale	double	1.0	10^{-99} — ∞	Scale of z-axis relative to x and y

3D View Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Miscellaneous Options				
leftworld	Boolean	False	True/False	Left-handed coordinate system
hiddenline	Boolean	False	True/False	Apply hidden-surfaces
paintcube	Boolean	False	True/False	Paint plot surfaces
axisguides	Boolean	True	True/False	Show miniature axis guide

3.2 Resource Specification

View Point

eyepos.x

eyepos.y

eyepos.z Specify the eye-position relative to the view center. The 3D eye-position (or viewing vector) defaults to (1.0, 1.5, 0.5).

viewcenter.x

viewcenter.y

viewcenter.z Specify the view-center about which the 3D data is to be viewed. By default, the view-center is placed at the center of the plot-boundaries, e.g., **viewcenter.x** = **0.5(xmin+xmax)**, where **xmin** and **xmax** are the boundaries of the plot in x (specified in Plotset Resources).

window.xmin

window.xmax

window.ymin

window.ymax Specify the relative size of the 2D window in the 3D plot. The 2D window dimensions are used to zoom or pan on selected areas of the 3D plot. By default, **window.xmin=-0.7**, **window.xmax=0.7**, **window.ymin=-0.7** and **window.ymax=0.7**.

Axis Options

axislabel Specifies whether to plot the axis labels in the 3D plot. Default value is **True**.

axismove Specifies whether the plot axis labels and tickmarks rotate as the view is rotated. In a 3D plot, only 1 of 4 possible axes in x, y, or z has labels and tickmarks attached to it. If **axismove** is **False**, the axes are plotted such that the labeled z-axis is always on the left-most side of the plot, while the x and y labeled axes are always on the bottom sides of the view cube closest to the eye. If **True**, the labeled plot axes are fixed to specific coordinates and rotate as the view rotates. The default is **False**.

axisscale Specifies whether the axes are scaled unequally. If **True**, the unequal scaling results in a cube-like plot, where the x, y and z axes have the same length regardless of the actual boundary range. If **False**, the axes are scaled proportionally to the actual boundary range. The default is **True**.

xaxisscale

yaxisscale

zaxisscale Specifies the ratio of the x, y and z axes. If any of these options are specified, **axisscale** is automatically set to **True**, and the 3D plot is drawn as a cubic block where the lengths of the axes have the proportions **xaxisscale**:**yaxisscale**:**zaxisscale**. For example, if **xaxisscale=1.0**, **yaxisscale=1.0**, and **zaxisscale=2.0**, then the 3D plot is drawn as a cube with height twice that of the length and width. **xaxisscale**, **yaxisscale** and **zaxisscale** all default to 1.0.

Miscellaneous Options

leftworld Specifies whether to use a left-handed coordinate system or a right-handed coordinate system. If **True**, a left-handed coordinate system is used. The default is **False**, and the 3D plot is displayed using a right-handed coordinate system.

hiddenline Specifies whether polygons are drawn with hidden lines/surfaces. The hiddenline routine uses a simple painter's algorithm; polygons further from the eye are drawn first and obscured or painted over by polygons closer to the eye. Default is **False**.

paintcube Specifies whether the inner surfaces making up the view cube are painted. Default is **False**.

axisguides Specifies whether the miniature axis guides are drawn on the lower left corner of the plot. Defaults to **True**.

4 DATASET RESOURCES

4.1 Overview

Dataset Resources are used to modify the properties of datasets. In general, dataset resources affect the way the data is manipulated and displayed in the plot, e.g., the number of contour slices in a contour plot.

Dataset Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Contour Options				
contstyle	integer	1	1 — 4	Contour style 1=Contour Lines 2=Gradated Colored Contours 3=Mesh 4=Contour Lines superimposed on gradated colored contours
contfill	Boolean	False	True/False	Colored contours (contstyle=2)
cstep	double	0.1(cmax-cmin)	0.0 — ∞	Contour step size (linear)
nsteps	integer	10	0 — 50	Number of contour steps (linear)
logsteps	integer	1	1 — 10	Steps per logarithmic decade
contours	string	NULL	N/A	Individual contour specification
cmax	double	Data maximum	-∞ — ∞	Maximum contour level
cmin	double	Data minimum	-∞ — ∞	Minimum contour level
linetypes	integer	2	1 — 3	No of contour linetypes
contlabel	Boolean	True	True/False	Plot contour labels
contclip	Boolean	False	True/False	Clip filled contours
interpolate	integer	0	0 — 3	Contour interpolation style (only for rectangular grids) 0=Rectangle → Rectangle 1=Rectangle → Triangles 2=Rectangle → 4 Triangles 3=Rectangle → Flat Rectangle
bitmap	Boolean	False	True/False	Used with interpolate=3 to draw contours as bitmaps.
logx	Boolean	False	True/False	Contour log-interpolation in x

Dataset Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
logy	Boolean	False	True/False	Contour log-interpolation in y
logz	Boolean	False	True/False	Contour log-interpolation in z
Mesh Options				
meshplot	Boolean	False	True/False	Plot mesh
boundary	Boolean	True	True/False	Plot boundary
regbound	Boolean	False	True/False	Fill region boundary
fillbound	Boolean	False	True/False	Fill material boundary
Vector Options				
vlog	Boolean	False	True/False	Plot vectors in log-scale
vlogscale	double	-	$0 — \infty$	Vector scale-factor (log)
vscale	double	-	$0 — \infty$	Vector scale-factor (linear)
vhead	Boolean	True	True/False	Draw vector arrow-head
vtail	Boolean	False	True/False	Draw vector arrow-tail
Barchart Options				
barmin	double	0.0	$-\infty — \infty$	Minimum bar extent
Histogram Options				
binwidth	double	0.1(xmax-xmin)	$-\infty — \infty$	Histogram bin size
binstart	double	xmin	$-\infty — \infty$	Histogram starting point
Debug Options				
printid	Boolean	False	True/False	Print all element IDs
pointid	Boolean	False	True/False	Print point IDs
nodeid	Boolean	False	True/False	Print node IDs
triaid	Boolean	False	True/False	Print triangle IDs
rectid	Boolean	False	True/False	Print rectangle IDs
curveid	Boolean	False	True/False	Print curve IDs

Dataset Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
regionid	Boolean	False	True/False	Print region IDs
Curve Options				
splinetype	integer	0	0 — 6	Spline Type 0=None (Linear) 1=Cubic B-Spline 2=Doubled Cubic B-Spline 3=Quadratic B-Spline 4=Catmull-Rom Spline 5=Cubic Bezier Spline 6=Quadratic Bezier Spline
applyfill	Boolean	True	True/False	Fill curves as requested
Miscellaneous Options				
annotate	Boolean	True	True/False	Plot annotations

4.2 Resource Specification

Contour Options

contstyle Specifies the contour plot style. **contstyle=1** results in contour lines, **contstyle=2** causes the regions between adjacent contours to be filled with gradated colors, while **contstyle=3** causes the 3D surface mesh to be drawn in place of contours. **contstyle=4** draws contour lines on top of gradated colored contours, and is used to highlight the different contour levels. **contstyle=1**, **contstyle=2**, and **contstyle=4**, when used in combination with **meshplot=True** causes contours and the underlying mesh to be drawn simultaneously. By default, **contstyle=1**.

contfill Specifies whether to draw contours in gradated colors. **contfill=True** is equivalent to setting **contstyle=2**, while **contfill=False** corresponds to **contstyle=1**.

cstep Causes contours to be selected with a constant step-size of **cstep**. This results in equally-spaced contour increments inside the given [**cmin**, **cmax**] limits. However, the minimum and maximum contour values do NOT necessarily correspond to the minimum and maximum limits of the quantity; the values of the contours are rounded so as to obtain numerically-pleasing values. For example, with **cmin=0.36** and **cmax=1.03**, contour values of z=0.4, 0.5,... 1.0 are obtained. By default, contours are plotted using a **cstep** value corresponding to roughly 10 equally spaced contours.

nsteps Causes contours to be selected with a constant step-size of **(cmax-cmin)/nsteps**. The contours are selected inclusive of **cmin** and **cmax**, resulting in levels at z=**cmin**, z=**cmin**+i(**cmax-cmin**)/**nsteps**, ... z=**cmax**. For example, with **cmin=0.36**, **cmax=1.03** and **nsteps=5**, contour values of z=0.36, 0.494, 0.628, 0.762, 0.896, and 1.03 are obtained. If **nsteps=0**, a single contour is drawn at **0.5(cmin+cmax)**. The default value is 10.
Note: If **nsteps** is NOT explicitly specified, the contours are calculated using the **cstep** option, where **cstep** = `rounded_value_of`**((cmax-cmin)/10**).

logsteps Specifies the number of contours to be plotted in a logarithmic decade. Setting **logsteps** causes contours to be selected with logarithmic spacing, which is preferred for plotting data which ranges widely in magnitude. If **logsteps=1**, the ratio between adjacent contours is 10.0 and contours are rounded to the nearest power of 10. For example, if the quantity ranges in z-value from 0.5e10 to 3.2e15, contours will be plotted at 1e10, 1e11, 1e12, 1e13, 1e14 and 1e15. **logsteps** values greater than 1 cause more than one contour to be plotted per decade, e.g., **logsteps=2** plots contours at 1e10, 5e10, 1e11, etc. The default **logsteps** value is 1.

contours Specifies a string of contour values. This option is used to manually set the number and type of contours in the contour plot. The **contours** string has the syntax:

contours`=([new] value1 value2 ... [linewidth=<>] ...)`

where `value1` etc., are user-defined contour levels; the optional "`new`" character string causes all previously specified contour values to be over-written. Curve properties such as linewidth, linetype, and marker-types for the contours may also be defined in the same line, using curve keywords such as "`linewidth`" and "`linetype`"; see Curve Properties for the detailed specification. Multiple **contours** statements can be used together, provided that each statement is on a separate line; **contours** statements following the first **contours** statement append their contour values to the current list. By default, **contours** is a blank string, and the contour levels are selected automatically using any of **cstep**, **nsteps**, and **logsteps**.

The following example illustrates the use of successive **contours** statements to set contour levels and the curve properties of the contours.

```
$ DATA=CONTOUR

# Contour step specification
% contours = (0.1 0.2 0.3 lw=2 lt=1 lc=2 mt=2)
% contours = (0.5 0.6 0.7 lw=1 lt=2 lc=3)

# Now define the contour grid
% xmin=0 xmax=2 nx=5
% ymin=0 ymax=1 ny=3
0 0 0 0 0
0 1 0 1 0
0 0 1 0 0

$ END
```

cmin

cmax Specify the range in which contours are to be plotted. These default respectively to the minimum and maximum z-values of the data in the selected dataset.

linetypes Specifies the number of line-types for the contour plot. Valid values range from 1 to 3. The default is to use 2 line-types, in which case major contours are drawn in solid lines while minor contours are drawn dashed.

contlabel Specifies whether contour labels are to be placed on contour lines. **contlabel** is ignored when drawing colored contours (**contstyle=2**).

contclip Specifies if the filled contours are to be plotted for values greater than cmin and cmax. **contclip** defaults to **False**; if **contstyle=2**, then the contours are filled above and below the specified color ranges. For example, if contours are specified at values of 0.1, 0,2 and 0.3, data-ranges less than 0.1 and greater than 0.3 will be filled as well. But if **contclip=True**, then only the contours between 0.1 and 0.3 will be filled with colors.

interpolate Specifies the contour interpolation style for rectangular-grid-based data. By default, **interpolate=0**, and each rectangle in the grid is plotted as is. If **interpolate=1** or **interpolate=2**, each rectangle is split up into 2 or 4 triangles respectively. The distinction between rectangles and triangles is made because the 4 points of a single rectangle in 3D space are often not coplanar. Finally, **interpolate=3** is a special scheme where the data is broken up into planar rectangles with constant z-values; each rectangle is centered about the data-points on the original rectangular grid.

logx

logy

logz Specifies whether contours are to be calculated using logarithmic or linear interpolation. Typically, contour data with a wide range of values is plotted using logarithmically varying steps and logarithmic interpolation in z (**logz=True**). By default, **logx=False**, **logy=False** and **logz=False**.

Mesh Options

meshplot Specifies whether the underlying mesh is to be plotted. Particularly useful for contour plots and mesh-based datasets. Default is **False**.

boundary Specifies whether the outlines of the device materials are to be plotted. Useful only for mesh-based datasets. Default is **True**.

regbound Specifies whether the boundaries of the various device regions are to be drawn. Regions with identical materials are plotted in identical colors. Useful only for mesh-based datasets. Default is **False**.

fillbound Specifies whether the various device regions are to be filled with colors corresponding to different materials. Regions with identical materials are plotted in identical colors. Useful only for mesh-based datasets. Default is **False**.

Vector Options

vlog Specifies whether vectors are to be plotted on a logarithmic or linear scale. If **True**, the length of the vectors is proportional to the logarithm of the vector magnitude. Default is **False**.

vlogscale Specifies the scale factor for vectors plotted on a logarithmic scale. The default scaling is based on the boundary limits of the data and the logarithm of the maximum magnitude of the vectors.

vscale Specifies the scale factor for vectors plotted on a linear scale. The default scaling is based on the boundary limits of the data and the maximum magnitude of the vectors.

vhead Specifies whether the vectors are to be plotted with arrow-heads. Only vectors of sufficient length (as determined by dimensions on the plot) are plotted with arrow-heads. Default is **True**.

vtail Specifies whether the vectors are to be plotted with cross-tails. Default is **False**.

Barchart Options

barmin Specifies the minimum y-extent of each bar in the bar-chart. The 2D bar is drawn from `y=`**barmin** to `y=yval`, where `yval` is the data-point value of the bar. By default, **barmin=0.0**.

Histogram Options

binstart Specifies the beginning point from which the data-values will be collected into bars. The data is collected in ranges of size **binwidth** beginning from `x=`**binstart**. Thus the histogram bars are constructed over the ranges [**binstart**, **binstart** + **binwidth**], [**binstart** + **binwidth**, **binstart** + 2×**binwidth**] and so on. **binstart** defaults to `xmin`, the minimum data value.

binwidth Specifies the collection range or bin size for the histogram. The data is collected in ranges of size **binwidth** beginning from `x=`**binstart**. Thus the histogram bars are constructed over the ranges [**binstart**, **binstart** + **binwidth**], [**binstart** + **binwidth**, **binstart** + 2×**binwidth**] and so on. **binwidth** defaults to `0.1×(xmax-xmin)`, such that 10 histogram bars are plotted.

Debugging Options

printID Specifies whether the various element integer identifiers are to be printed on the plot. Equivalent to setting **pointID**, **nodeID** etc individually.

pointID Specifies whether point integer identifiers are to be printed on the plot. Points are contained in CURVE2D, CURVE3D, CONTCURVE, CONTOUR, GRID4D, PROBABILITY-PLOT and MESH-based datasets. Default is **False**.

nodeID Specifies whether node integer identifiers are to be printed on the plot. Nodes are connected to points, and contained in CONTOUR, GRID4D and MESH-based datasets. Default is **False**.

triaID Specifies whether triangle integer identifiers are to be printed on the plot. Triangles are connected to nodes which in turn are connected to points; triangles are contained in CONTOUR, GRID4D and MESH-based datasets. Default is **False**.

rectID Specifies whether rectangle integer identifiers are to be printed on the plot. Rectangles are connected to nodes which in turn are connected to points; rectangles are contained in CONTOUR and GRID4D datasets. Default is **False**.

curveID Specifies whether curve integer identifiers are to be printed on the plot. Curves are connected to points; curves are contained in CURVE2D, CURVE3D and CONTOUR datasets. Individual curve IDs can be set using the **curveno** option specific to CURVE2D and CURVE3D datasets (see documentation for CURVE2D and CURVE3D data formats). Default is **False**.

regionID Specifies whether region integer identifiers are to be printed on the plot. Regions are connected to nodes which in turn are connected to points; regions are contained only in MESH-based datasets. Default is **False**.

Curve Options

splinetype Specifies if spline interpolation is to be used to connect the points on a curve. Applies to CONTOUR and CURVE datasets only. 6 splinetypes are provided. The *Cubic B-Spline* (**splinetype=1**), *Doubled Cubic B-Spline* (**splinetype=2**), *Quadratic B-Spline* **(splinetype=3)** produce curves that approximate the control points; the curves pass close to but do not go through the actual data points. *Catmull-Rom Splines* (**splinetype=4**) pass through the control points but can produce wild variations in between points. *Quadratic Bezier Spline* (**splinetype=5**) and *Cubic Bezier Spline* (**splinetype=6**) curves hit the midpoints of the control points. The default, **splinetype=0** is to connect the data-points using straight lines.

applyfill Specifies whether curves with specified fills are actually filled. If **False**, only the curve outlines are plotted. Actually a useful debugging aid for inspecting overlapping curves. Applies only to CURVE datasets. Default is **True**.

Miscellaneous Options

annotate Specifies whether annotations attached to the specified dataset are to be plotted. If **False**, the annotations are not plotted. Allows multiple datasets to be overlaid without too much confusion due to overlapping annotations. Default is **True**.

5 CURVE RESOURCES

5.1 Overview

Curve Resources are used to modify the properties of curves in contour and curve datasets. Note that curve resources may be specified using abbreviated keywords, i.e., "`lw=5`" is identical to "`linewidth=5`".

Curve Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Line Options				
linelabel	string	NULL	N/A	Curve label
linewidth lw	integer	1	0 — 32	Curve width
linetype lt	integer	1	0 — 10	Curve type 0=None 1=Solid 2=Dashed 3=Dotted 4=Dot-Dash 5=Long-Dots 6=Double-Dot 7=Long-Dash 8=Sparse Dot-Dash 9=Triple-Dot 10=Dot-Dot-Dash
linecolor lc	integer	1	-1 — 10	Line color -1=Background color[a] 0=Foreground color[b] 1=Yellow 2=Blue 3=Green 4=Red 5=Dark Blue 6=Orange 7=Pink 8=Light Pink 9=Cyan 10=Brown

Curve Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Marker Options				
markertype mt	integer	0	0 — 13	Marker type 0=None 1=Dot 2=Cross 3=X 4=White Square 5=Black Square 6=White Diamond 7=Black Diamond 8=White Triangle 9=Black Triangle 10=White Inverted Triangle 11=Black Inverted Triangle 12=White Circle 13=Black Circle
markercolor mc	integer	1	-1 — 10	Marker color Refer to linecolors
markersize ms	integer	1	1 — 10	Marker size
Fill Options				
filltype	integer	0	0 — 8	Fill type 0=None 1=Solid 2=Square Crosshatch 3=Left-to-right Diagonal 4=Right-to-left Diagonal 5=Crossed Diagonals 6=Vertical Lines 7=Horizontal Lines 8=Dotted Horizontal Lines
fillcolor	integer	1	-1 — 10	Fill color Refer to linecolors

a. The background color is typically blue on an X11 plot. On a Postscript plot, the background color is always white.

b. The foreground color is typically white on an X11 plot. On a Postscript plot, the foreground color is always black.

5.2 Resource Specification

Lines

linelabel Specifies a label (legend) to be placed to the left of the plot. For contour-curves, the label is placed at various locations on the curve itself. Default is a blank string.

linewidth

lw Specifies the width of the line to be used in drawing the curve. Default is 1.

linetype

lt Specifies the type of line-pattern to be used in drawing the curve. Line-patterns include solid lines, dashed lines and dotted lines. Default is 1 (solid line).

linecolor

lc Specifies the color to be used in drawing the curve. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

Markers

markertype

mt Specifies the markertype to be used in drawing the points on the curve. The default value is 0, in which case markers are not plotted.

markercolor

mc Specifies the color to be used in drawing the markers. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

markersize

ms Specifies the size of markers on the curve. The default value is 1, which is the smallest size.

Fills

filltype Specifies the filltype to be used in drawing the curve. The default is 0, in which case only the outline of the curve is drawn.
Note: The **filltype** is ignored if the Dataset Resource **applyfill** is False.

fillcolor Specifies the color to be used in the fill-pattern. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

6 GLOBAL CURVE RESOURCES

6.1 Overview

Global Curve Resources are used to modify the properties of *all* the curves in contour and curve datasets. These resources are used primarily to set the default properties for curves; Curve Resources are then used to set properties of individual curves. Note that the argument names of Global Curve Resources differ from Curve Resource names only by a "d" in at the beginning of the name.

Global Curve Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Line Options				
dlinewidth	integer	1	0 — 32	Curve width
dlinetype	integer	1	0 — 10	Curve type 0=None 1=Solid 2=Dashed 3=Dotted 4=Dot-Dash 5=Long-Dots 6=Double-Dot 7=Long-Dash 8=Sparse Dot-Dash 9=Triple-Dot 10=Dot-Dot-Dash
dlinecolor	integer	1	-1 — 10	Line color -1=Background color[a] 0=Foreground color[b] 1=Yellow 2=Blue 3=Green 4=Red 5=Dark Blue 6=Orange 7=Pink 8=Light Pink 9=Cyan 10=Brown

Global Curve Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Marker Options				
dmarkertype	integer	0	0 — 13	Marker type 0=None 1=Dot 2=Cross 3=X 4=White Square 5=Black Square 6=White Diamond 7=Black Diamond 8=White Triangle 9=Black Triangle 10=White Inverted Triangle 11=Black Inverted Triangle 12=White Circle 13=Black Circle
dmarkercolor	integer	1	-1 — 10	Marker color Refer to linecolors
dmarkersize	integer	1	1 — 10	Marker size
Fill Options				
dfilltype	integer	0	0 — 8	Fill type 0=None 1=Solid 2=Square Crosshatch 3=Left-to-right Diagonal 4=Right-to-left Diagonal 5=Crossed Diagonals 6=Vertical Lines 7=Horizontal Lines 8=Dotted Horizontal Lines
dfillcolor	integer	1	-1 — 10	Fill color Refer to linecolors

a. The background color is typically blue on an X11 plot. On a Postscript plot, the background color is always white.
b. The foreground color is typically white on an X11 plot. On a Postscript plot, the foreground color is always black.

6.2 Resource Specification

Lines

dlinewidth Specifies the width of the line to be used in drawing the curve. Default is 1.

dlinetype Specifies the type of line-pattern to be used in drawing the curve. Line-patterns include solid lines, dashed lines and dotted lines. Default is 1 (solid line).

dlinecolor Specifies the color to be used in drawing the curve. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

Markers

dmarkertype Specifies the markertype to be used in drawing the points on the curve. The default value is 0, in which case markers are not plotted.

dmarkercolor Specifies the color to be used in drawing the markers. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

dmarkersize Specifies the size of markers on the curve. The default value is 1, which is the smallest size.

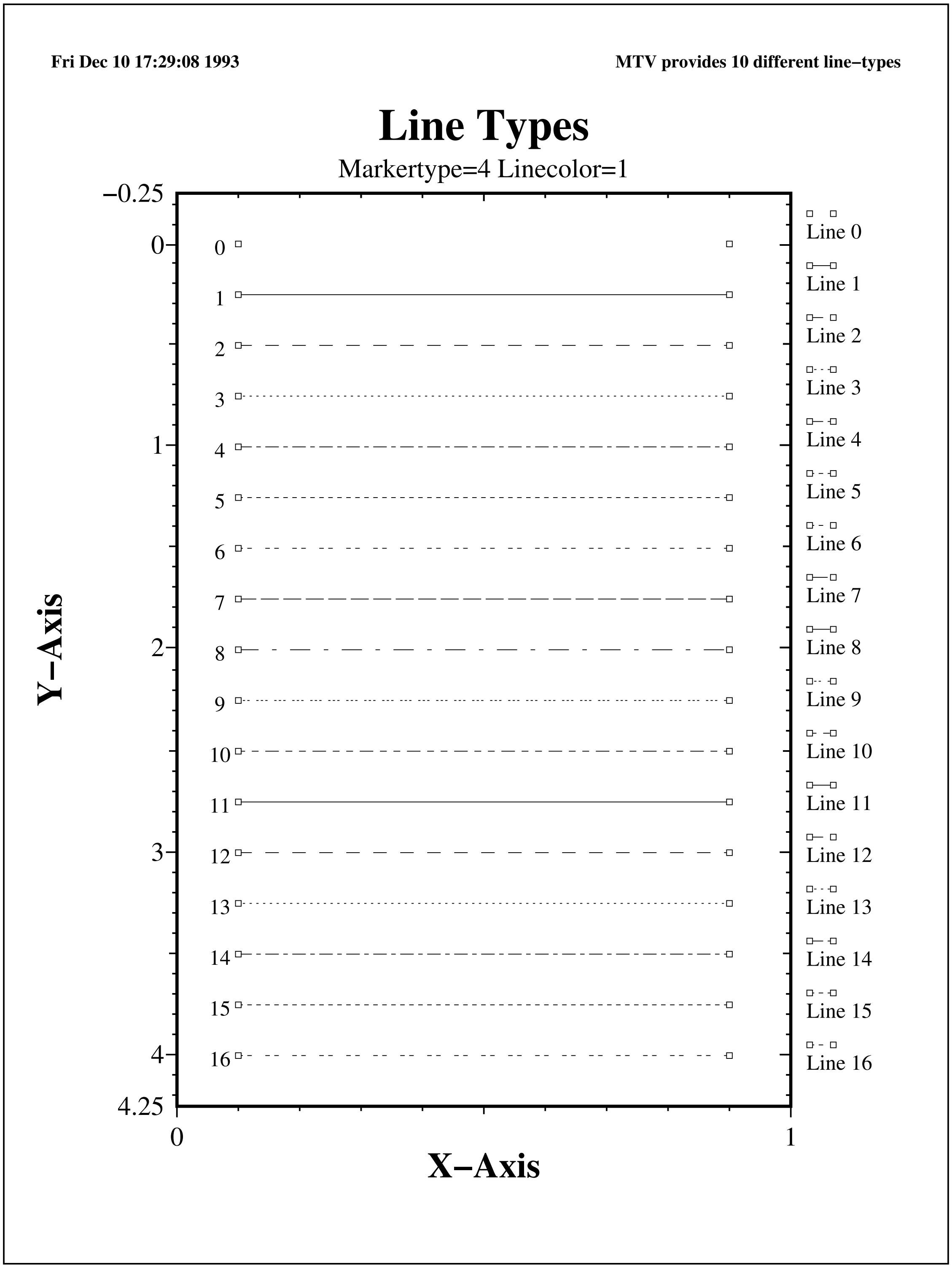
Fills

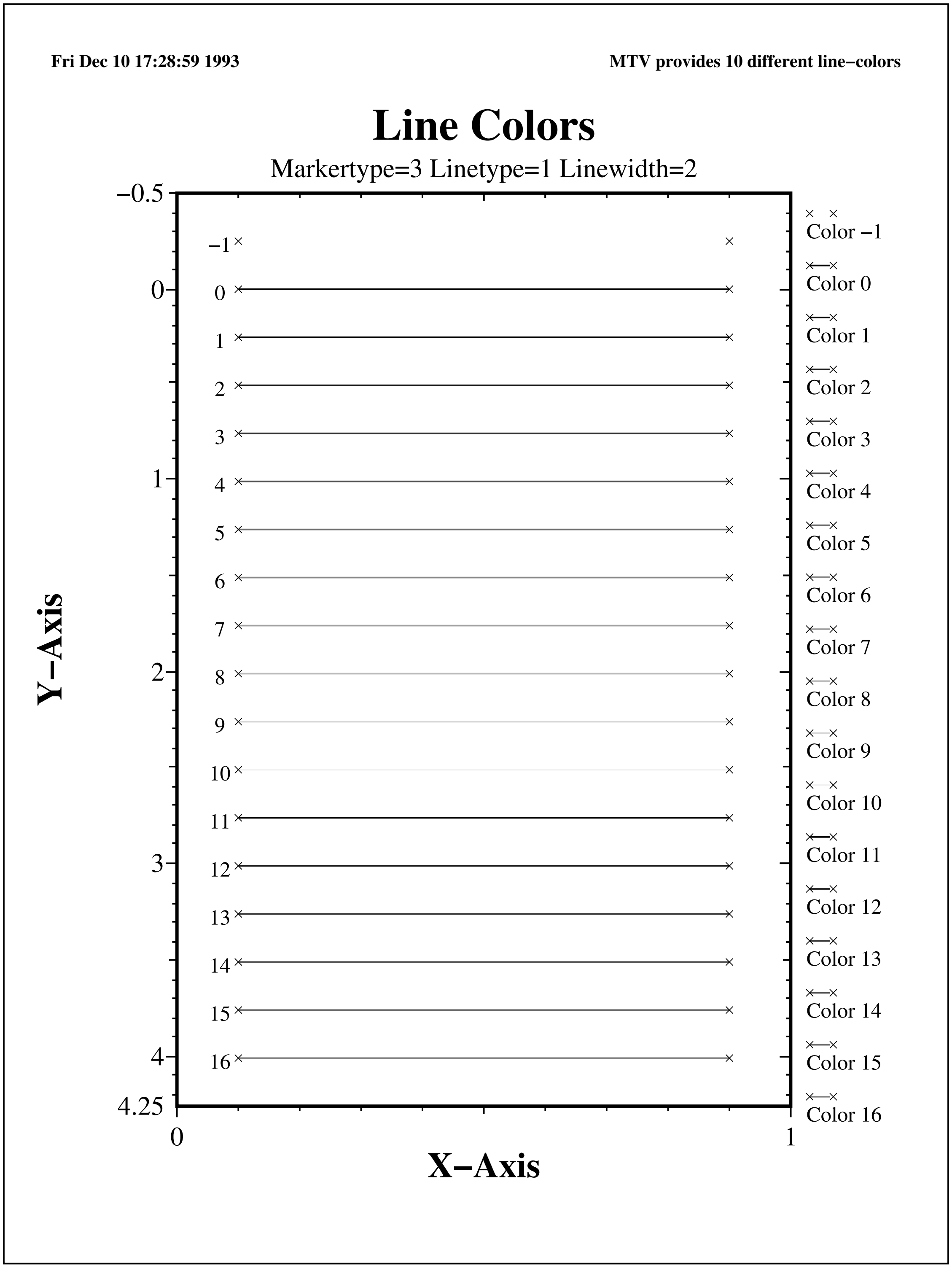
dfilltype Specifies the filltype to be used in drawing the curve. The default is 0, in which case only the outline of the curve is drawn.
Note: The **dfilltype** is ignored if the Dataset Resource **applyfill** is **False**.

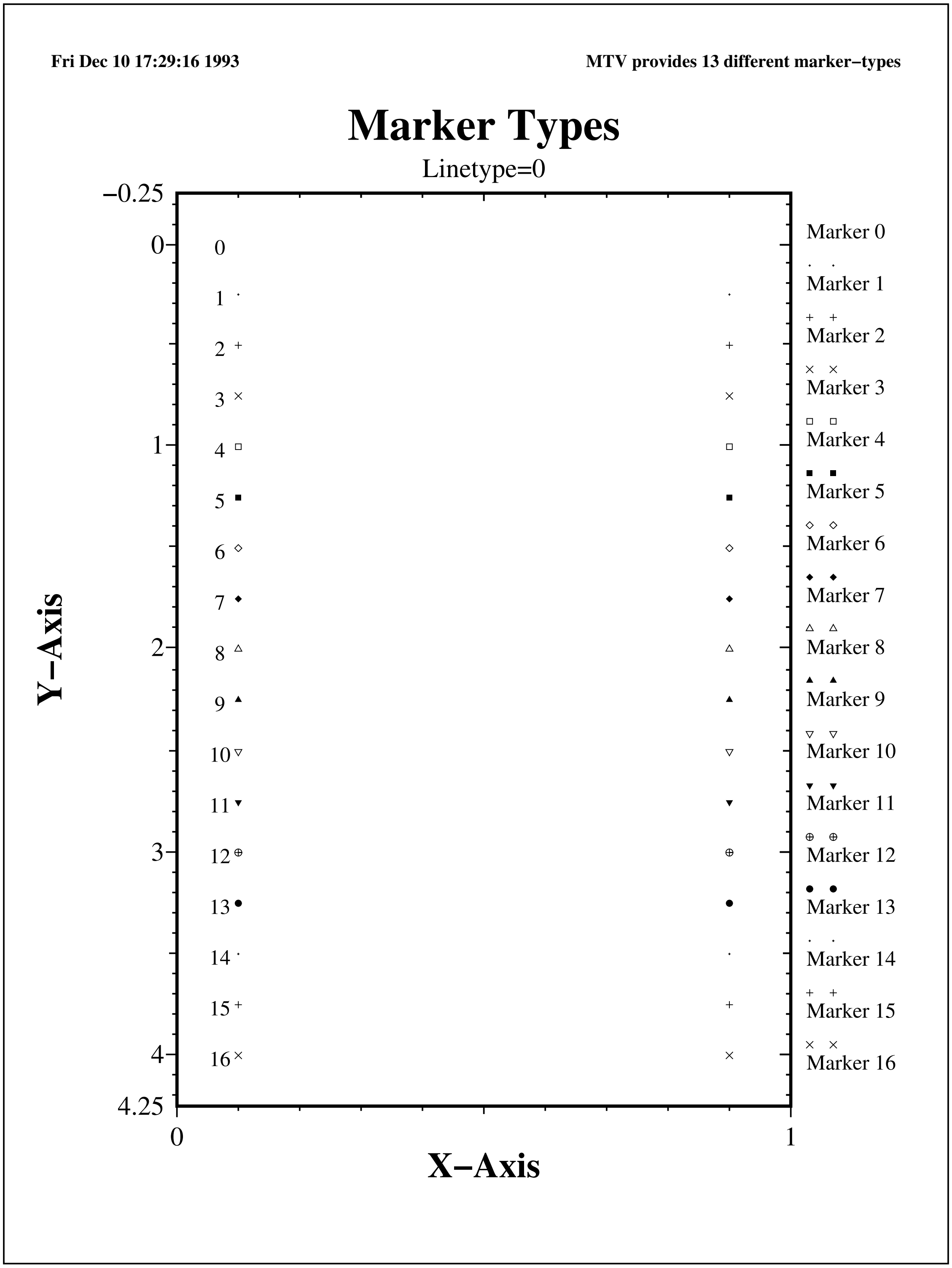
dfillcolor Specifies the color to be used in the fill-pattern. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

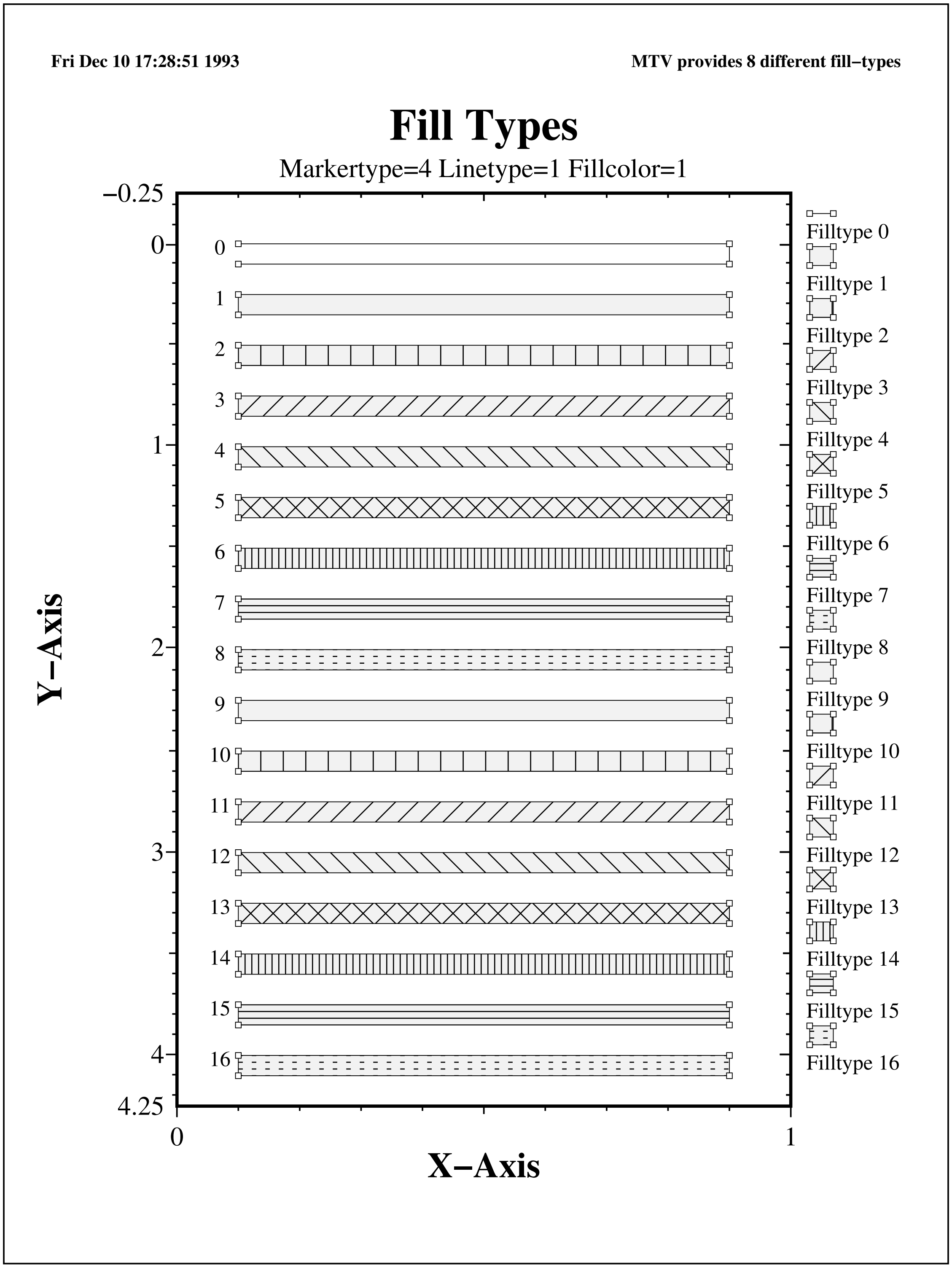
7 CURVE RESOURCE EXAMPLES

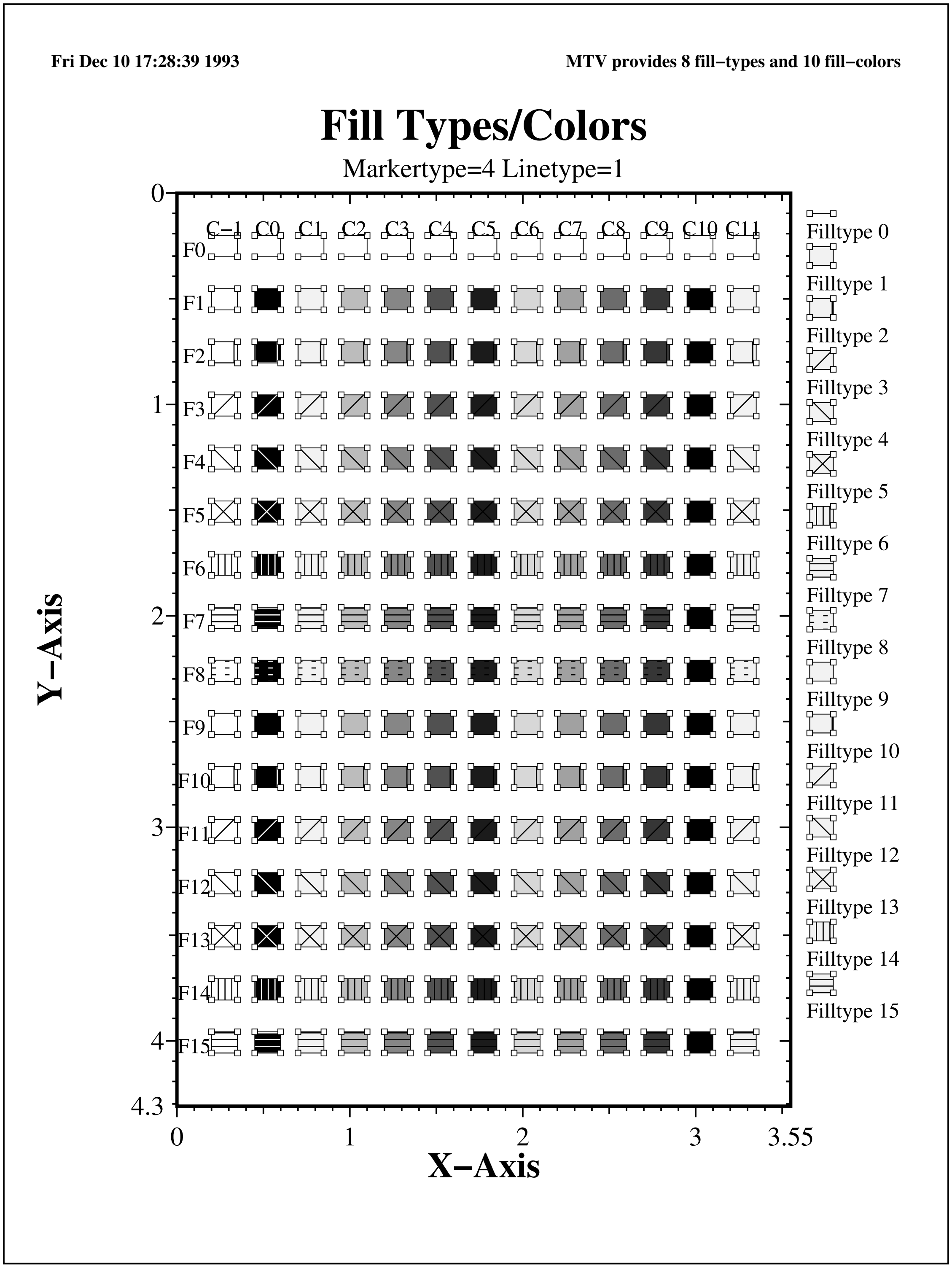
The example plots shown on the following pages illustrate the many different line, marker and fill types available as Curve Resources.


Fri Dec 10 17:29:08 1993
MTV provides 10 different line−types
Line Types
Markertype=4 Linecolor=1
−0.25
0
1
2
3
4
4.25
0
1
X−Axis
Y−Axis
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Line 0
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
Line 11
Line 12
Line 13
Line 14
Line 15
Line 16


Fri Dec 10 17:28:59 1993
MTV provides 10 different line−colors
Line Colors
Markertype=3 Linetype=1 Linewidth=2
−0.5
0
1
2
3
4
4.25
0
1
X−Axis
Y−Axis
−1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Color −1
Color 0
Color 1
Color 2
Color 3
Color 4
Color 5
Color 6
Color 7
Color 8
Color 9
Color 10
Color 11
Color 12
Color 13
Color 14
Color 15
Color 16


Fri Dec 10 17:29:16 1993
MTV provides 13 different marker−types
Marker Types
Linetype=0
−0.25
0
1
2
3
4
4.25
0
1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Marker 0
Marker 1
Marker 2
Marker 3
Marker 4
Marker 5
Marker 6
Marker 7
Marker 8
Marker 9
Marker 10
Marker 11
Marker 12
Marker 13
Marker 14
Marker 15
Marker 16
Y−Axis
X−Axis


Fri Dec 10 17:28:51 1993
MTV provides 8 different fill−types
Fill Types
Markertype=4 Linetype=1 Fillcolor=1
−0.25
0
1
2
3
4
4.25
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Filltype 0
Filltype 1
Filltype 2
Filltype 3
Filltype 4
Filltype 5
Filltype 6
Filltype 7
Filltype 8
Filltype 9
Filltype 10
Filltype 11
Filltype 12
Filltype 13
Filltype 14
Filltype 15
Filltype 16
Y−Axis
X−Axis



Fri Dec 10 17:28:39 1993
MTV provides 8 fill−types and 10 fill−colors
Fill Types/Colors
Markertype=4 Linetype=1
C−1 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11
F0
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
Filltype 0
Filltype 1
Filltype 2
Filltype 3
Filltype 4
Filltype 5
Filltype 6
Filltype 7
Filltype 8
Filltype 9
Filltype 10
Filltype 11
Filltype 12
Filltype 13
Filltype 14
Filltype 15
0
1
2
3
4
4.3
0
1
2
3
3.55
X−Axis
Y−Axis

APPENDIX B

MTV Annotations

1 ANNOTATIONS

1.1 Overview

The MTV data format allows plot annotations to be attached to sets of data. Annotations are specified in the data-file or data-stream as strings beginning with the "@" character. Multiple annotations can be attached in each dataset. Furthermore, each annotation can be specified in data (world) coordinates or in plot coordinates.

The format for annotations is as follows:

MTV Annotation Types

First Character	Keyword	Arguments	
@	rectangle	x1=<> y1=<> [z1=<>] x2=<> y2=<> [z2=<>]	<annotation-properties>
@	line	x1=<> y1=<> [z1=<>] x2=<> y2=<> [z2=<>]	<annotation-properties>
@	arrow	x1=<> y1=<> [z1=<>] x2=<> y2=<> [z2=<>]	<annotation-properties>
@	point	x1=<> y1=<> [z1=<>]	<annotation-properties>
@	text	x1=<> y1=<> [z1=<>] label=<>	<annotation-properties>

The following is a simple example of a CURVE2D dataset containing a single arrow annotation.

```
$DATA=CURVE2D

# Plotset resources
% toplabel=”Arrow Annotation Example”

# Curve specification
0 0
0.1 0.01
0.2 0.04
0.3 0.09
0.4 0.16
0.5 0.25
0.6 0.36
0.7 0.49
0.8 0.64
0.9 0.81
1.0 1.0

# Arrow Annotation
@arrow x1=0.60 y1=0.25 x2=0.52 y2=0.25 label=”y=x*x” fn=20

$END
```

Annotation Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Line Options				
linelabel label ll	string	NULL	N/A	Curve label
linewidth lw	integer	1	0 — 32	Curve width
linetype lt	integer	1	0 — 10	Curve type 0=None 1=Solid 2=Dashed 3=Dotted 4=Dot-Dash 5=Long-Dots 6=Double-Dot 7=Long-Dash 8=Sparse Dot-Dash 9=Triple-Dot 10=Dot-Dot-Dash
linecolor lc	integer	1	-1 — 10	Line color -1=Background color[a] 0=Foreground color[b] 1=Yellow 2=Blue 3=Green 4=Red 5=Dark Blue 6=Orange 7=Pink 8=Light Pink 9=Cyan 10=Brown

Annotation Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Marker Options				
markertype mt	integer	0	0 — 13	Marker type 0=None 1=Dot 2=Cross 3=X 4=White Square 5=Black Square 6=White Diamond 7=Black Diamond 8=White Triangle 9=Black Triangle 10=White Inverted Triangle 11=Black Inverted Triangle 12=White Circle 13=Black Circle
markercolor mc	integer	1	-1 — 10	Marker color Refer to linecolors
markersize ms	integer	1	1 — 10	Marker size
Fill Options				
filltype ft	integer	0	0 — 8	Fill type 0=None 1=Solid 2=Square Crosshatch 3=Left-to-right Diagonal 4=Right-to-left Diagonal 5=Crossed Diagonals 6=Vertical Lines 7=Horizontal Lines 8=Dotted Horizontal Lines
fillcolor fc	integer	1	-1 — 10	Fill color Refer to linecolors
Miscellaneous Options				
absolute	Boolean	False	T/F	Plot in world/plot coordinates

Annotation Property Argument List				
Name	**Type**	**Default**	**Range**	**Description**
Miscellaneous Options (continued)				
clip	Boolean	True	T/F	Clip against plot boundary
fontsize fn	integer	10	1 — 30	Text annotation font-size (in dpi)
scale	Boolean	False	T/F	Scale the text annotation font-size relative to the size of the plot.

a. The background color is typically blue on an X11 plot. On a Postscript plot, the background color is always white.

b. The foreground color is typically white on an X11 plot. On a Postscript plot, the foreground color is always black.

1.2 Resource Specification

Lines

linelabel

label

ll Specifies a label (legend) to be placed to the left of the plot. For contour-curves, the label is placed at various locations on the curve itself. Default is a blank string.

linewidth

lw Specifies the width of the line to be used in drawing the curve. Default is 1.

linetype

lt Specifies the type of line-pattern to be used in drawing the curve. Line-patterns include solid lines, dashed lines and dotted lines. Default is 1 (solid line).

linecolor

lc Specifies the color to be used in drawing the curve. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

Markers

markertype

mt Specifies the markertype to be used in drawing the points on the curve. The default value is 0, in which case markers are not plotted.

markercolor

mc Specifies the color to be used in drawing the markers. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

markersize

ms Specifies the size of markers on the curve. The default value is 1, which is the smallest size.

Fills

filltype

ft Specifies the filltype to be used in drawing the curve. The default is 0, in which case only the outline of the curve is drawn.
Note: The **filltype** is ignored if the Dataset Resource **applyfill** is **False**.

fillcolor

fc Specifies the color to be used in the fill-pattern. Colors are approximated by gray-scales for black-and-white Postscript plots. Default is 1 (yellow).

Miscellaneous

absolute Specifies if the position of the annotation is specified in world coordinates or in plot coordinates. Defaults to **False**, in which case world coordinates are assumed. If **absolute=True**, then the annotations are specified by their position on a page - the coordinates must be given such that `0<x<1` and `0<y<1`; the actual position of the annotation is obtained from `x` × `width`, `y` × `height`, where `width` and `height` are the dimensions of the page/plot. This latter scheme however has the disadvantage that the position of an annotation in an X11 plot is slightly different from that on a Postscript plot.

clip Specifies if the annotation is to be clipped against the plot boundaries. Defaults to **True**, in which case, the annotation is not drawn if the point(s) to which it is attached fall outside the plot boundaries.

fontsize

fn Specifies the font-size for text annotations. This font-size applies to the text-labels in all the annotation types. The font-size is specified in dpi (dots-per-inch) and has a default value of 10.0. The text annotation font-size remains constant as the plot is zoomed in or zoomed out, i.e., regardless of the plot-size, the text-labels are drawn with a constant size by default. Use the **scale** option to change this behavior.

scale Specifies if the font-size of the text-annotations scales with the size of the plot. If set to **True**, then the text annotations are drawn with a larger font-size as the plot is zoomed in, and with a smaller font-size as the plot is zoomed out. Default is **False**.

1.3 Example

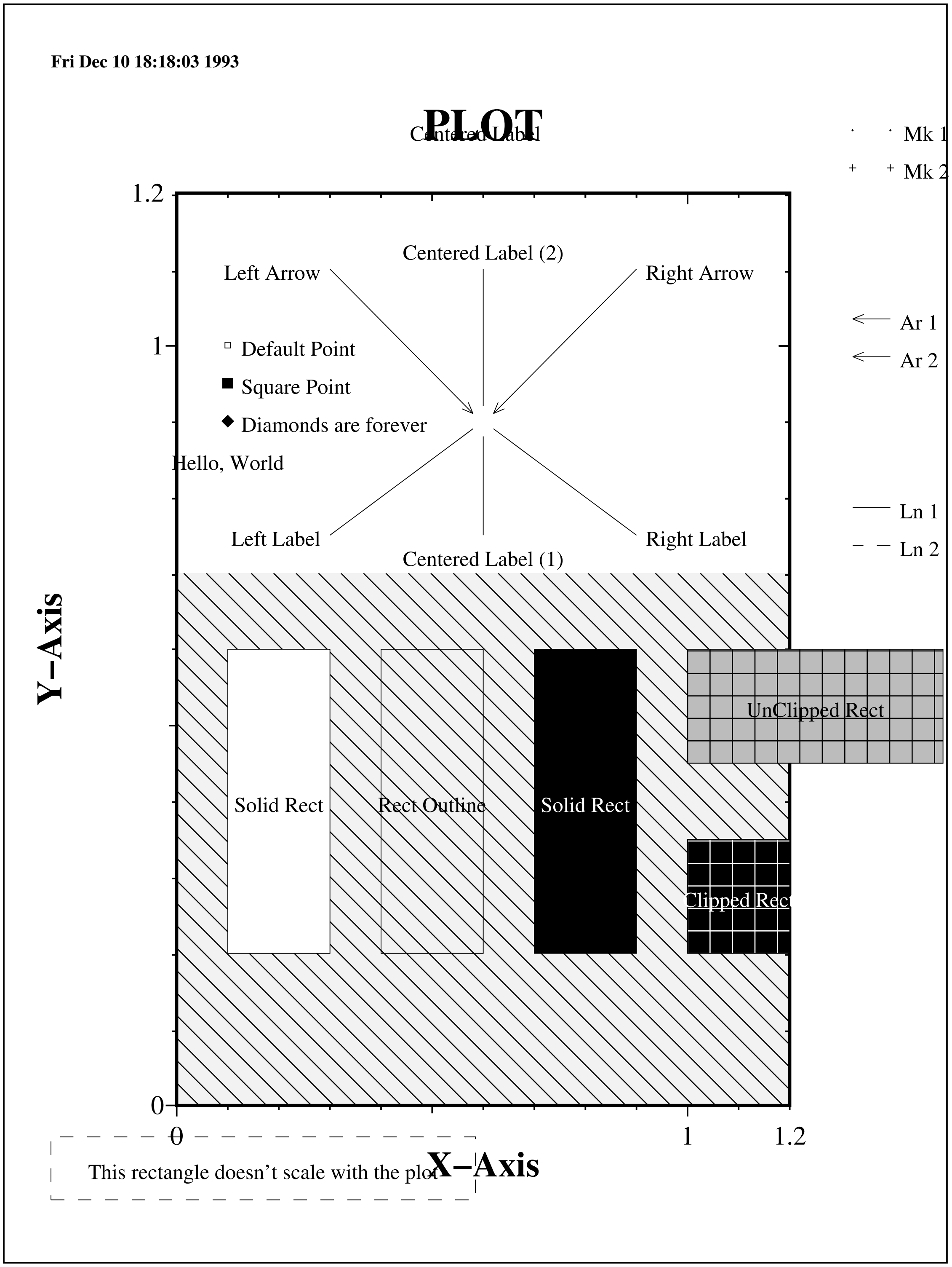
The following shows an example file containing a CURVE2D dataset containing one curve and several annotations. The resultant plot is shown on the following page,.

```
$ DATA=CURVE2D

# This data-file illustrates the use of annotations

% vymax=1.2

# Define fake data
% linetype=0 filltype=4
0.0 0.0
1.2 0.0
1.2 0.7
0.0 0.7
0.0 0.0

# define annotations
#
# By default, annotations are defined in data coordinates
#
# Annotations can also be specified by their position on a page (plot)
# in which case the coordinates are given such that 0<x<1 and 0<y<1;
# the actual position of the annotation is obtained from x*width,y*height
# This scheme however has the disadvantage that the position of an
# annotation in an X plot is slightly different from that on a Postscript
# plot.
#

#
# Rectangle annotations
#
# syntax:
#   @ rectangle P1 P2 [properties]
#
#   P1=lower left corner
#   P2=upper right corner
#   The linelabel is placed at the center of the rectangle
#

# rectangle with background color (white in Postscript) and solid fill
@ rectangle x1=0.1 y1=0.2 z1=0.0 x2=0.3 y2=0.6 z2=0.0 \
fillcolor=-1 filltype=1 linelabel="Solid Rect"

# rectangle with no fill (outline) - this is the default
@ rectangle x1=0.4 y1=0.2 z1=0.0 x2=0.6 y2=0.6 z2=0.0 \
fillcolor=0 filltype=0 linelabel="Rect Outline"

# rectangle with foreground color (black in Postscript) and solid fill
@ rectangle x1=0.7 y1=0.2 z1=0.0 x2=0.9 y2=0.6 z2=0.0 \
fillcolor=0 filltype=1 linelabel="Solid Rect"
```

```
# rectangle with striped fill, extends outside data-boundaries (default clip)
@ rectangle x1=1.0 y1=0.2 z1=0.0 x2=1.5 y2=0.35 z2=0.0 \
filltype=2 linelabel="Clipped Rect"

# rectangle with striped fill, extends outside data-boundaries (no clip)
@ rectangle x1=1.0 y1=0.45 z1=0.0 x2=1.5 y2=0.6 z2=0.0 \
fillcolor=2 filltype=2 linelabel="UnClipped Rect" doclip=false

# Plot this rectangle in absolute coordinates (x=0..1, y=0..1)
@ rectangle x1=0.05 y1=0.05 x2=0.50 y2=0.1 linetype=2 absolute=true \
linelabel="This rectangle doesn't scale with the plot"

#
# Line annotations
#
# syntax:
#   @ line P1 P2 [properties]
#
#              P1------------------P2
#            label
#
#   P1=starting point
#   P2=ending point
#   The linelabel is placed at the starting-point (P1)
#   and left-justified if P1.x > P2.x, and right-justified if P1.x < P2.x,
#   and center-justified if P1.x = P2.x
#

# default line
@ line      x1=0.90 y1=0.75 z1=0.0 x2=0.62 y2=0.89 z2=0.0 \
linelabel="Right Label"

# Label centered at P2
@ line      x1=0.60 y1=0.75 z1=0.0 x2=0.60 y2=0.88 z2=0.0 \
linelabel="Centered Label (1)"

# Label centered at P2
@ line      x1=0.60 y1=1.10 z1=0.0 x2=0.60 y2=0.92 z2=0.0 \
linelabel="Centered Label (2)"

# Label to the left of the line
@ line      x1=0.30 y1=0.75 z1=0.0 x2=0.58 y2=0.89 z2=0.0 \
linelabel="Left Label"

# Labels at right corner of plot
@ line      x2=0.90 y2=0.60 x1=0.94 y1=0.60 \
linetype=1 absolute=True linelabel="Ln 1"
@ line      x2=0.90 y2=0.57 x1=0.94 y1=0.57 \
linetype=2 absolute=True linelabel="Ln 2"
```

```
#
# Arrow annotations
#
# syntax:
#   @ arrow P1 P2 [properties]
#
#               P1--------------->P2
#             label
#
#   P1=starting point
#   P2=ending point
#   The linelabel is placed at the starting-point (P1)
#   and left-justified if P1.x > P2.x, and right-justified if P1.x < P2.x,
#   and center-justified if P1.x = P2.x
#

# default line
@ arrow     x1=0.90 y1=1.10 z1=0.0 x2=0.62 y2=0.91 z2=0.0 \
linelabel="Right Arrow"

# Label to the left of the line
@ arrow     x1=0.30 y1=1.10 z1=0.0 x2=0.58 y2=0.91 z2=0.0 \
linelabel="Left Arrow"

# Labels at right corner of plot
@ arrow     x2=0.90 y2=0.75 x1=0.94 y1=0.75 \
linetype=1 linecolor=1 absolute=True linelabel="Ar 1"
@ arrow     x2=0.90 y2=0.72 x1=0.94 y1=0.72 \
linetype=1 linecolor=2 absolute=True linelabel="Ar 2"

#
# Marker(point) annotations
#
# syntax:
#   @ point P1 [properties]
#
#   P1=coordinates
#   The linelabel is left-justified and on the right of P1
#

# single point
@ point     x1=0.1 y1=1.00 z1=0.0 linelabel="Default Point"

# another point
@ point    x1=0.1 y1=0.95 z1=0.0 markertype=5 markersize=3 linelabel="Square
Point"

# another point
@ point     x1=0.1 y1=0.90 z1=0.0 markertype=7 markersize=3 linelabel="Dia-
monds are forever"

# Set of points at the left side of the plot
@ point     x1=0.90 y1=0.90 markertype=1 absolute=True
@ point     x1=0.94 y1=0.90 markertype=1 absolute=True linelabel="Mk 1"
@ point     x1=0.90 y1=0.87 markertype=2 absolute=True
@ point     x1=0.94 y1=0.87 markertype=2 absolute=True linelabel="Mk 2"
```

```
#
# Text (label) annotations
#
# syntax:
#   @ text P1 [properties]
#
#   P1=coordinates of center of label
#   The linelabel is centered around P1
#

# label
@ text      x1=0.1 y1=0.85 z1=0.0 linelabel="Hello, World"

# label in absolute coordinates
@ text      x1=0.5 y1=0.9 z1=0.0 linelabel="Centered Label" absolute=True

$END
```


Fri Dec 10 18:18:03 1993
PLOT
Centered Label
Mk 1
Mk 2
Ar 1
Ar 2
Ln 1
Ln 2
Left Arrow
Centered Label (2)
Right Arrow
Default Point
Square Point
Diamonds are forever
Hello, World
Left Label
Centered Label (1)
Right Label
Solid Rect
Rect Outline
Solid Rect
UnClipped Rect
Clipped Rect
Y−Axis
X−Axis
1.2
1
0
0
1
1.2
This rectangle doesn't scale with the plot