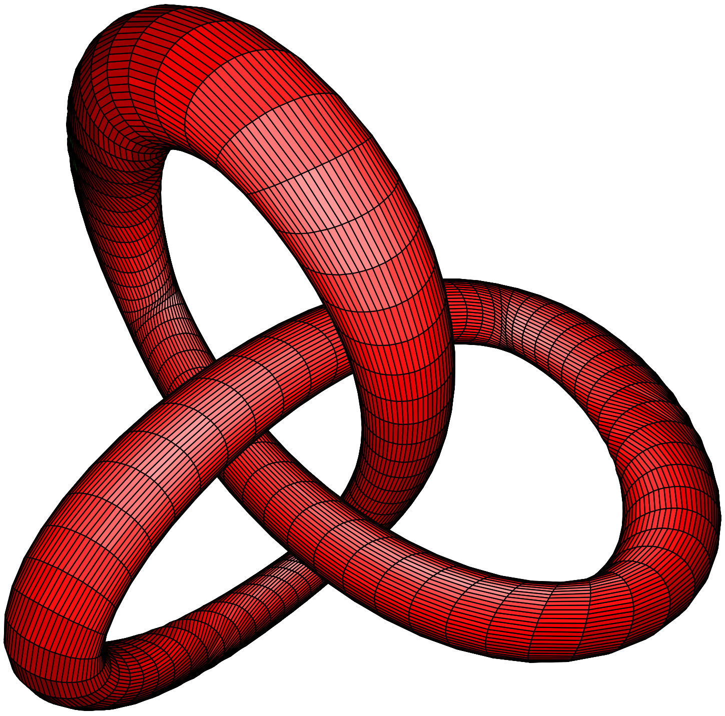
MLAB Graphics Examples

by

Daniel Kerner

Revision Date: May 28, 2014

© Civilized Software, Inc.
12109 Heritage Park Circle
Silver Spring, MD 20906 USA
Tel: (301)962-3711
Email: csi@civilized.com
URL: http://www.civilized.com



Preface

MLAB Graphics Examples is an updated version of the book **Some Easy Exercises to Teach You MLAB 2-D Graphics** written by Marvin Shapiro in 1979 and revised by Bruce Krulwich and Gary Knott in 1983 at the National Institutes of Health Division of Computer Research and Technology, Bethesda, MD.

The original book demonstrated how to create various graphs with the MLAB mathematical modeling computer program running on a DEC PDP-10 mainframe computer with a Tektronix 4012 display and Calcomp plotter. This book serves the same purpose for the MLAB program running on Microsoft DOS/Windows (MSWindows), Apple Macintosh OS7/8/9/X, and Linux with X-Windows computers with monochrome or color display and Hewlett Packard LaserJet or PostScript printer.

Chapter names in the Table of Contents, figure numbers in the text and List of Figures, and page numbers in the index appear in the color blue and provide clickable links to the respective content.

The examples of surface drawing with sweeping shown on the cover and in Chapter 13 were developed by Dr. Zhiping You at Civilized Software, Inc., in the early 1990's.

The author thanks Drs. Gary Knott, Barry Bunow, Zhiping You, and others working at Civilized Software, Inc., and Dr. Stephen Baylor at the University of Pennsylvania, for their help, corrections, and suggestions.

If the reader should find any errors in this text, please send an email to kerner@civilized.com describing the error so that corrections can be made to future editions.

Daniel Kerner
Silver Spring, MD
January, 2014

Contents

Chapter 1

Drawing in the default graphics window

In MLAB, making simple graphs is very easy. Yet very elaborate graphs can also be constructed, as we will demonstrate throughout this manual. The key to simplicity is using only those elements of MLAB that you wish to use, and letting the other optional elements take care of themselves. When a graphics command is given, generally appropriate defaults are assumed for any optional components of the command which are omitted. In this initial chapter, we will do our drawing in the default graphics window. The default graphics window is named `W`, but that, like most of its other properties, is conveniently hidden in many commands. Subsequently in this manual, MLAB commands which have been used and explained in previous examples will generally not be re-explained at each subsequent use unless some new feature is involved.

Pictures in MLAB are created using the `DRAW` statement. The simplest possible `DRAW` statement is

```
 * DRAW M
```

where `M` is a 2-column matrix. Each row of `M` contains the x- and y-coordinates of a point to be drawn, in columns 1 and 2, respectively.

The effects of this command are:

1. to create the default MLAB 2 dimensional graphics object named `W`. An MLAB graphics object is colloquially referred to as a graphics "window". It consists of a rectangular *frame* region; an *image* region, which is a rectangular region within the frame in which graphs are drawn; and a *window* clipping region, which is a rectangular region of the Cartesian (x, y)-plane where the curve defined by the points of `M` is drawn. The window region in the Cartesian plane is mapped into the image region on the display.

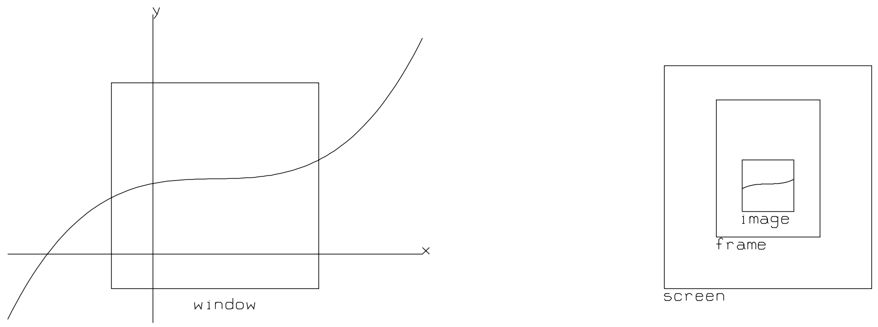


Figure 1.1: The MLAB window, image, frame, and screen rectangles.

2. to create a curve in that window. The curve will contain the points of `M`, connected one to the next by straight lines. The part of the curve within the window region will be drawn in the image region on the display.
3. to create x- and y-axes, each labeled with 6 numbers which show the domain and range of the curve.

Figure 1.1 illustrates the relationships between the window, image, frame, and screen rectangles in MLAB. Note that the curve is clipped to the window and the contents of the window appear in the image rectangle. The remaining pictures in this book will show the screen region containing the frame and image, however the border of the screen and frame will not be drawn.

Perhaps surprisingly, `DRAW` does *not* immediately show the picture on the display screen. In order to see the picture created by the `DRAW` command, the additional command `VIEW` is required. The `VIEW` command has different effects on different computer platforms. A DOS PC has two screen modes: text and graphics. Ordinarily, when MLAB statements are being entered on a DOS PC, text mode is used. To see pictures, we must switch to graphics mode by the `VIEW` statement. When MLAB is in graphics mode, new MLAB statements may not be typed without first returning to

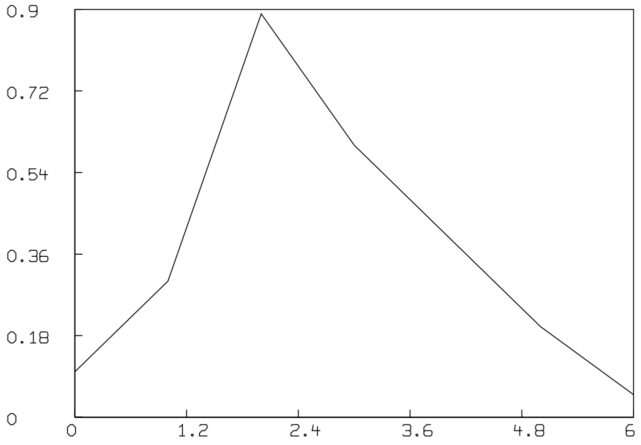


Figure 1.2: Data points connected by straight lines.

text mode. To return from graphics mode to MLAB command mode, a key—any key—must be struck.

In the case of windowing-based operating systems such as Macintosh, Linux with X-Windows, and Windows, the `VIEW` command will cause MLAB pictures to appear in a window of the underlying windowing system. This underlying window serves as a virtual screen. In this book we will give examples assuming the DOS operating system on a PC is being used. In the case of other platforms, the only important difference is that once a `VIEW` command is issued, the MLAB picture *remains* on the screen and no further `VIEW` command is required. The MLAB picture can be explicitly removed from the screen in the windowing system environment by typing the command `UNVIEW`; then a subsequent `VIEW` will be needed to redisplay MLAB graphics.

Example 1-1: Draw a graph on the display screen of the following data points connected by straight lines: (0,.10) (1,.30) (2,.89) (3,.60) (4,.40) (5,.20) (6,.05). The MLAB statements to make this graph appear below. The picture is shown in Figure 1.2.

```
* M COL 1 = 0:6
* M COL 2 = LIST(.1,.3,.89,.6,.4,.2,.05)
* DRAW M
* VIEW
```

In the above statements, `COL` is short for column, and the expression `0:6` denotes the 7-row, 1-column matrix (column vector) composed of the values {0,1,2,3,4,5,6}. The `LIST` command returns a column vector constructed from the list of numbers provided. A few MLAB operators

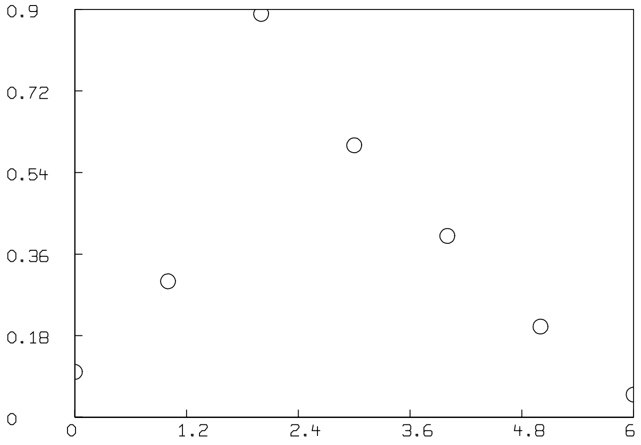


Figure 1.3: Data drawn with circles.

such as `:`, `LIST`, and `COL` are needed to do the numerical calculations and data rearrangements necessary to draw pictures with MLAB. Refer to the MLAB on-line Help system (e.g. type:`HELP LIST`) to get a short description of any MLAB operator. A complete discussion of every operator is found in the **MLAB Reference Manual**.

When a window is no longer needed, it may be removed (and the memory it occupies made available once more) by typing a `DELETE` command, e.g. `DELETE W`. This can also apply to functions and matrices. In each successive example below, we assume that any MLAB objects created previously have been deleted, unless otherwise specified.

Example 1-2: Draw the same data points as in Example 1-1, but mark the points with circles and omit the connecting lines. This may be accomplished with the MLAB commands:

```
* DELETE W
* DRAW M LINETYPE NONE POINTTYPE CIRCLE
* VIEW
```

The result is shown in Figure 1.3.

The `LINETYPE`-clause specifies the type of line used to connect the points of a graph. We wanted no connecting lines, so we used the clause `LINETYPE NONE`. (The default linetype is solid connecting lines, e.g. `LINETYPE SOLID`, as in Example 1-1 where no `LINETYPE`-clause appeared.) The keyword `LINETYPE` may be abbreviated to `LINE` or `LT`. The keyword `POINTTYPE` may be abbreviated to `PT`.

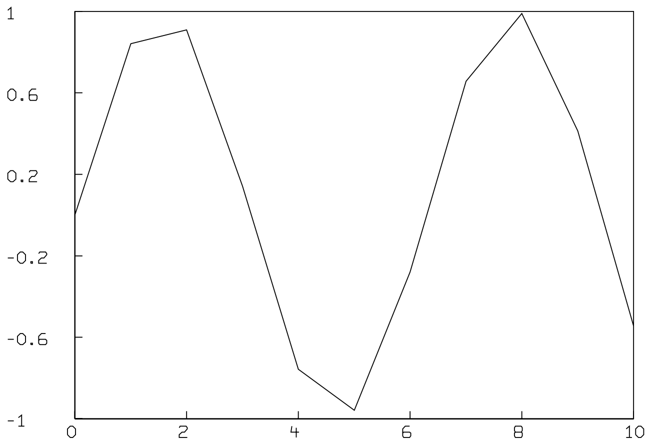


Figure 1.4: Sine curve sampled with 11 points.

Besides drawing data, MLAB may also be used to draw the graph of a function. The function is evaluated on a list of monotonically increasing (or decreasing) points covering a certain interval. The resulting (x, y) coordinate pairs are placed on the graph and connected by straight line segments. If the points are spaced closely enough, the resulting graph looks like a smooth curve. If the points are not spaced closely enough, the resulting graph will resemble the desired graph of the function less closely. A sufficiently and insufficiently sampled function are shown in the next example.

Example 1-3: Draw a graph of the function $f(x) = \sin(x)$ for x between 0 and 10. To begin, we will choose the abscissae $x =$ 0:10. The resulting curve is shown in Figure 1.4 using solid lines.

```
* DELETE M,W
* M = POINTS(SIN,0:10)
* DRAW M
* VIEW
```

Here we have used the `POINTS` operator to build an array `M` of (x, y)-coordinates which we then used in the `DRAW` statement. The first argument to `POINTS` is a function name, and the second argument is a list of abscissae. `POINTS` then returns a two column matrix with the abscissae (x-values) in column 1 and the function evaluated at the corresponding abscissae in column 2.

Evidently, this spacing of x-values is not sufficient to form a smooth curve. Next we choose the points $x =$ 0:10!100. This is the sequence of 100 numbers $0, .101, .202, \ldots, 10$. The MLAB statements appear below. The resulting curve is shown in Figure 1.5.

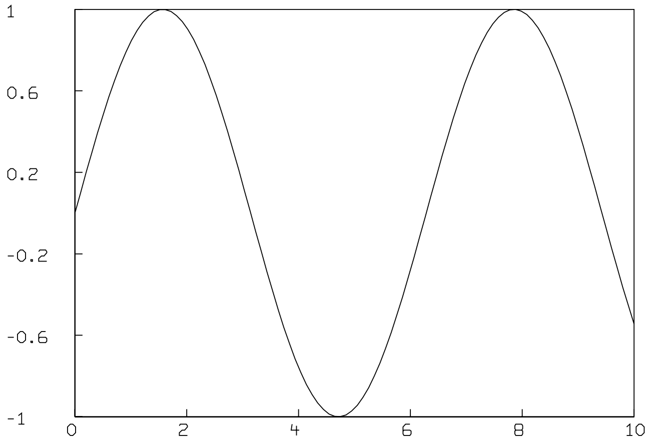


Figure 1.5: Sine curve sampled with 100 points.

```
* DELETE W
* DRAW POINTS(SIN,0:10!100)
* VIEW
```

Here we did not bother to construct the matrix of points in a separate statement.

Functions to be graphed in MLAB may be builtin, such as SIN in the previous example. However, the user may define and draw arbitrarily complicated functions (user functions) as well.

Example 1-4: Draw a graph of the function $f(x) = \sin(x) * \exp(-x/3)$, for x between 0 and 20. The resulting picture is shown in Figure 1.6. The MLAB commands to build this picture appear below.

```
* DELETE M,W
* FUNCTION F(X) = SIN(X)*EXP(-X/3)
* M = POINTS(F,0:20:.2)
* DRAW M
* VIEW
```

The FUNCTION statement is used to define an arbitrary user function. FUNCTION may be abbreviated as FCT.

Drawing the points given as the rows of a 2-column matrix in the default window W is about as easy as it possibly could be. The axes in W are automatically marked with tics and labeled with

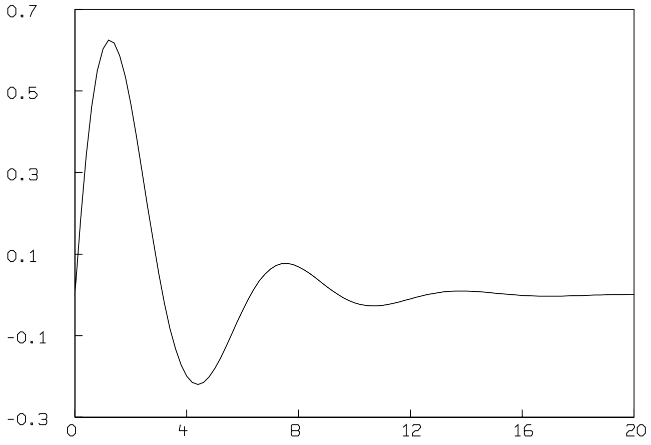


Figure 1.6: $\sin(x) * \exp(-x/3)$.

values of the x- and y-coordinates. The data points are scaled to fit into the image provided in W. Thus, a quick look at a graph of some function is a simple matter within MLAB.

Example 1-5: Here we draw the graph of several curves in one picture with different linetypes to distinguish each curve. Suppose we have a Fourier sine expansion of the odd periodic function $f(x) = -1$ for x in the interval $-\pi$ to 0, and $f(x) = 1$ for x in the interval 0 to π. We can express $f(x)$ in the form

$$f(x) = \sum_{j=1}^{\infty} a_j \sin(jx)$$

where

$$a_j = \frac{2(1 + (-1)^{j-1})}{\pi j}.$$

While the Fourier series for most functions is an infinite series, on a computer we must settle for a finite number of terms. Sometimes, most of the variability of a function is represented by the first few terms in the series. Let us examine the convergence of the partial sums of the series to $f(x)$ by graphing four partial sums of the series. We will establish the first seven coefficients in the 1-column matrix A $= a_k$. Note $a_k = 0$ when k is even. We will compute partial sums at each of 100 abscissae in the interval $-\pi$ to π which are stored in the 1-column matrix XL. For visibility, we will display each successive function raised two units above the previous one.

```
* DELETE M,W
* FCT F(N) = -2*((-1)^N-1)/(PI*N)
```

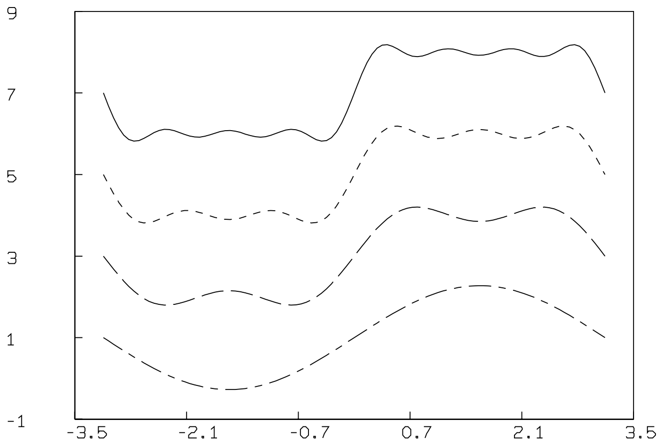


Figure 1.7: Partial sums of a sine Fourier series.

```
* A = F ON 1:7
* XL = -PI:PI!100
* FCT G(X) = SUM(J,1,I,A[J]*SIN(J*X))
* K = 4
* FOR I = 1:7:2 DO {
>    M = POINTS(G,XL);
>    M COL 2 = (M COL 2)+I;
>    DRAW M LT K;
>    K = K-1;
> }
* VIEW
```

The resulting picture is shown in Figure 1.7.

The `SUM` operator takes 4 arguments: a summation index, a lower limit for the summation index, an upper limit for the summation index, and a function expression, usually dependent on the summation index. The clause `LT K` in the `DRAW` statement causes each partial sum of the Fourier series to be drawn with a different linetype as `K` takes on the values 4,3,2, and 1.

Note that executing several `DRAW` statements results in each specified curve appearing together in the default window. When several curves are placed in the same picture, it is convenient to distinguish them by color. When each curve is drawn by a single `DRAW` statement, the `COLOR`-clause may be used to assign each curve a specific color. When the `DRAW` statement is in a `FOR`-loop, we can use the `FOR`-loop variable to specify the color.

Example 1-6: Draw several different color curves in one picture. Suppose we have a function with a parameter b, e.g. $f(x) = \sin(b * x) * \exp(-x/3)$, and we would like to draw a graph

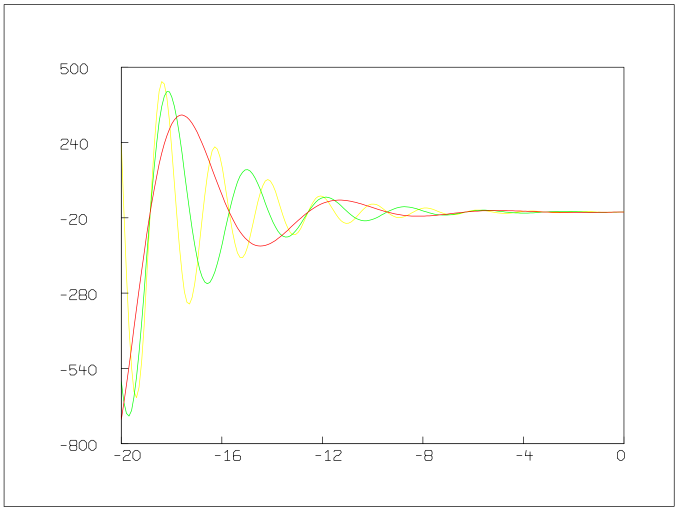


Figure 1.8: $\sin(b * x) * \exp(-x/3)$ for $b = 1, 2, 3$.

of $f(x)$ for x in the interval [-20,0] for several values of the parameter b: 1,2,3. The following MLAB commands can be used:

```
* DELETE W
* FCT F(X) = SIN(B*X)*EXP(-X/3)
* BV = 1:3
* XV = -20:0!200
* FOR I = 1:3 DO {B = BV[I]; DRAW POINTS(F,XV) COLOR COLORN(I);}
* VIEW
```

The resulting picture is shown in Figure 1.8.

BV and XV are 1-column vectors, with 3 and 200 elements, respectively. The FOR statement performs the MLAB commands within the braces {} repeatedly, with the scalar I successively taking on the values 1,2, and then 3. The function COLORN computes a color for each value of I. (This function is provided because consecutive intrinsic color numbers for the display are too similar in appearance.)

The curves of Example 1-6 all occupied the same region in the Cartesian plane. The region defined for the first curve was sufficient to contain the other curves as well. When this is not the case, the graph region must "grow" to hold all the curves which have been drawn.

Example 1-7: Draw several curves occupying different regions of the Cartesian plane. Suppose we wish to draw a graph of the Bessel function $J_0(x)$, but we don't know the range of x for

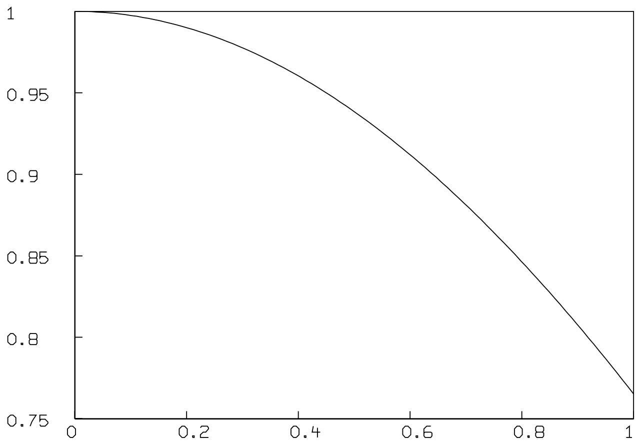


Figure 1.9: J_0(x) from 0 to 1.

which this function is “interesting”. We might first draw a piece of the function, e.g. from 0 to 1, and examine it. The MLAB commands appear below.

```
  * DELETE W
  * FCT J0(X) = BESSJ(X,0)
  * DRAW B0TO1 = POINTS(J0,0:1!100)
  * VIEW
```

The graph is shown in Figure 1.9.

Here we have given the curve a name. In fact MLAB curves are *always* given names, even if the user does not name them explicitly. The default names of curves are constructed by appending a number to the letter C. Curve names given by the user are most easily remembered if they have mnemonic content, e.g. B0TO1 for *B*essel-*0-to-1*. This is also useful if the user expects to refer to the curve again, e.g. to delete it, as will be described presently.

From the appearance of the graph, we conclude that perhaps there is still some “action” on the right hand side, so we draw another piece of the function, e.g. from 1 to 10 with the following MLAB commands.

```
  * DRAW B1TO10 = POINTS(J0,1:10!100) LT DASHED
  * VIEW
```

The graph is shown in Figure 1.10.

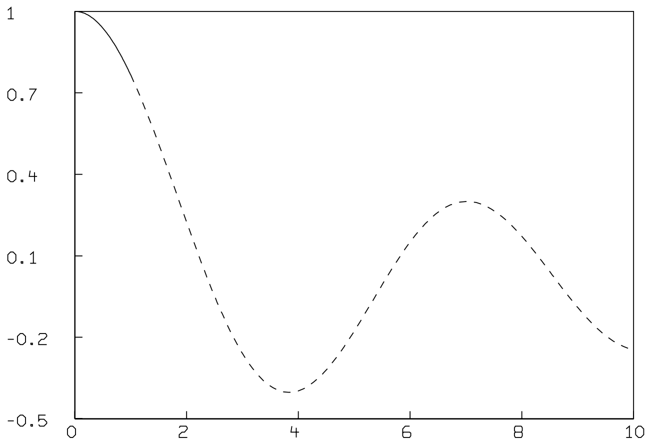


Figure 1.10: $J_0(x)$ from 0 to 10.

Here the additional part of the Bessel function has been drawn with a dashed line. Note that the x-range of the figure has grown automatically. By now we have the idea, and can see that a good range is from 0 to 100, so we finish the drawing with the MLAB statements below.

```
* DRAW B10TO100 = POINTS(J0,10:100!200)
* VIEW
```

The graph is shown in Figure 1.11.

Each time we gave a `DRAW` command, a new curve (which happens in this case to be a section of the same underlying mathematical curve) was put into the existing picture. Note in each case, the new curve went off the edge of the old graph. MLAB automatically detects this circumstance, and enlarges the existing window to hold the full extent of all the curves in that window.

What happens when a curve is removed? A curve may be removed from a graph using the MLAB `DELETE` command. We will delete the third portion of the curve, i.e. MLAB curve `B10TO100`.

```
* DELETE B10TO100
* VIEW
```

The resulting picture is shown in Figure 1.12.

Note that the window has shrunk down again so that its x-extent only goes from 0 to 10.

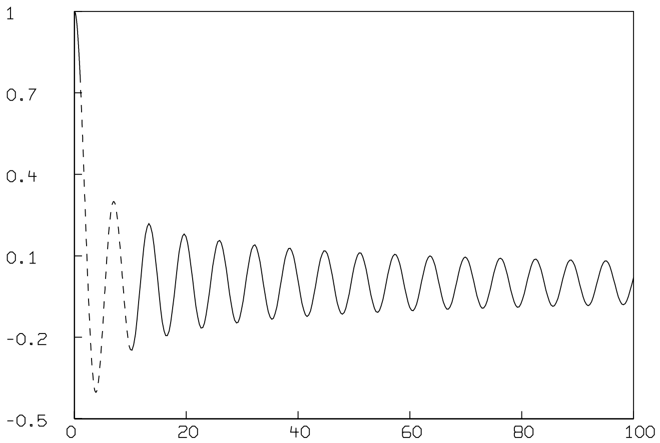


Figure 1.11: J_0(x) from 0 to 100.

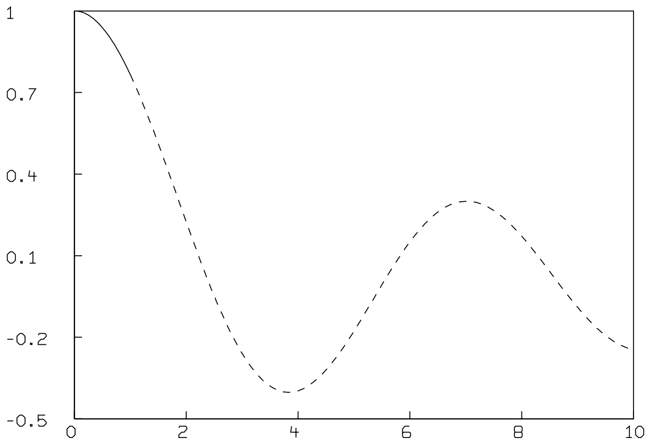


Figure 1.12: Shrunken window after deleting $J_0(x)$ from 10 to 100.

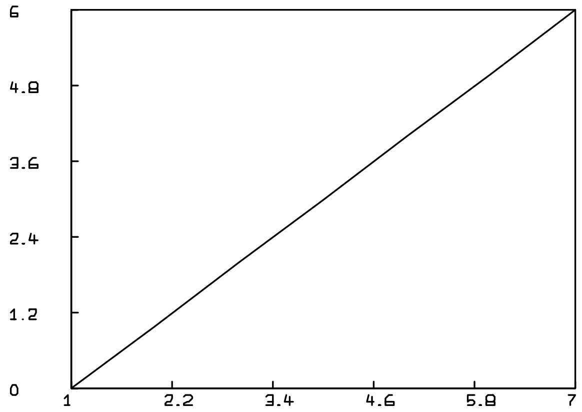


Figure 1.13: The MLAB default window after drawing 0:6.

If the data matrix containing the coordinates of points in a DRAW statement has one or three columns instead of two columns, MLAB will still create a default window and curve.

Example 1-8: Demonstrate the effect of the DRAW statement when the data matrix is a one column array.

```
* DELETE M,W
* M = 0:6
* DRAW M
* VIEW
```

The resulting picture is shown in Figure 1.13.

When the DRAW statement is given with a one column array as its first argument, MLAB treats the one column array as a list of ordinates (y-values) and supplies consecutive integers starting from 1 as the abscissae (x-values). It then draws the points connected by line segments in the MLAB default window, W. In this example, the line segments connecting the (x, y) ordered pairs $(1, 0)$, $(2, 1)$, $(3, 2)$, $(4, 3)$, $(5, 4)$, $(6, 5)$, and $(7, 6)$ are drawn.

See Chapter 12 for discussion and examples of what happens when the data matrix in a DRAW statement has three columns.

Chapter 2

Using linetypes and pointtypes

This chapter demonstrates some more of the options for the `LINETYPE` and `POINTTYPE` clauses of the `DRAW` statement.

As mentioned in Chapter 1, the `LINETYPE` clause controls the type of line used to connect data points in a graph. Currently, thirteen linetypes are available for 2 dimensional graphs in MLAB. Either the name or number may be used in the `LINETYPE` clause to designate a linetype. The linetype names and numbers are listed below.

`NONE`	(0)	no connecting lines.
`SOLID`	(1)	solid connecting line between successive points.
`DASH`	(2)	shorter dashed line connecting successive points.
`LDASH`	(3)	longer dashed line connecting successive points.
`DDASH`	(4)	alternating long and short dashed lines connecting successive points.
`ALTERNATE`	(5)	connect every other point by a solid line.
`MARKER`	(6)	solid line with marker skipping.
`VMARKER`	(7)	variable dashed line with marker skipping.
`DOTTED`	(8)	dotted line.
`DOTMARK`	(9)	dotted line with marker skipping.
`DASHMARK`	(10)	dashed line with marker skipping.
`SMARKER`	(11)	solid line with marker skipping using spline curve.
`SVMARKER`	(12)	variable dashed line with marker skipping using splines.

The next example demonstrates the use of the `ALTERNATE` linetype.

Example 2-1: Draw a graph of some differences between predicted and actual data, with

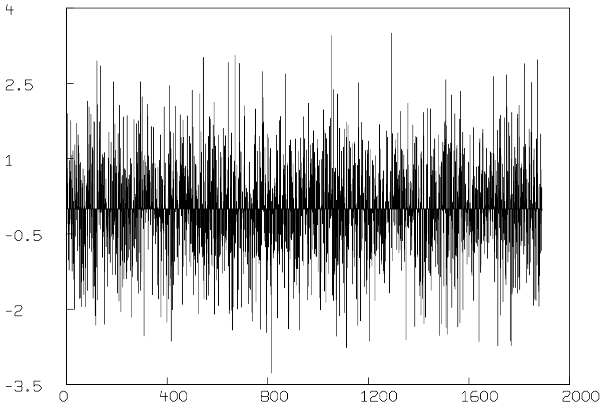


Figure 2.1: The simulated discrepancies between predicted and actual data.

vertical lines representing the residual errors. We simulate 1890 differences as zero-mean, unit variance, normal random variable values.

```
* E = NORMRAN ON 0^^1890
* V1 = 1:NROWS(E)
* E = MESH(V1&'0,V1&'E)
* DRAW E LT ALTERNATE
* VIEW
```

The resulting picture is shown in Figure 2.1.

The function NORMRAN computes a random number from a normal distribution (with zero mean and unit variance by default). When the first argument value is non-zero, it is the seed for the random number generator. When the first argument value is 0, it means take the next number in the current sequence. When a second argument is supplied, it specifies the mean of the distribution. The ON operator applies the specified function, NORMRAN in this case, to the scalar arguments given in each row of the matrix specified as the right argument of ON. Here the matrix is 0ˆˆ1890, which means a 1-column matrix with 1890 rows, all equal to zero. Thus, we have computed 1890 samples from a zero mean, unit variance normal distribution.

The column vector V1 is defined to contain the consecutive integers from 1 to NROWS(E), where NROWS(E) is the number of rows in E. The &' operator is the matrix column concatenation operator. The MESH operator transforms the vector E into a 2 column matrix with rows $1, 3, 5, \ldots$ taken from the 2 column matrix V1 &'0 (that is the 2 column matrix with V1 in column 1 and

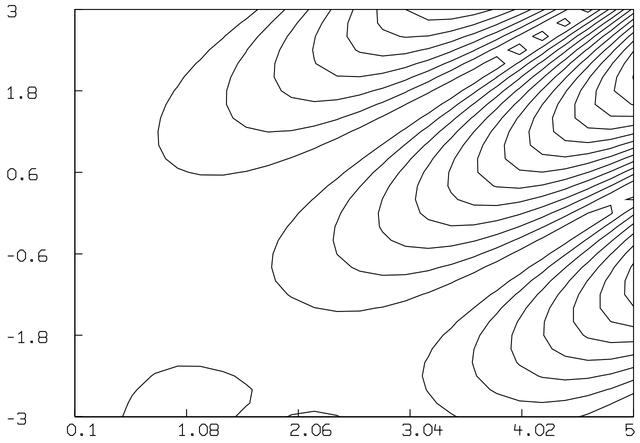


Figure 2.2: Contour map of the function $f(x,y) = (1 + x^2 + x * y) * ABS(\cos(x - y))$.

all zeros in column 2) and rows $2, 4, 6, \ldots$ taken from the 2 column matrix `V1 &'E` (that is the 2 column matrix with `V1` in column 1 and `E` in column 2). The `DRAW` statement with `LINETYPE ALTERNATE` then draws the points in `E` by connecting every other pair of points with a straight line segment.

The next example demonstrates the `MARKER` linetype.

Example 2-2: Draw a contour map of the function $f(x,y) = (1+x^2+x*y)*ABS(\cos(x-y))$. This is accomplished using the following MLAB commands:

```
* DELETE W
* FUNCTION F(X,Y) = (1+X^2+X*Y)*ABS(COS(X-Y))
* Q = CROSS(0:5:.5,-3:3:.2)
* M = POINTS(F,Q)
* C = CONTOUR(M,-3:36:2.75)
* DRAW C LT MARKER
* VIEW
```

The resulting picture is shown in Figure 2.2.

The `CROSS` function takes two one-column input matrices and returns a two-column matrix with elements from the first input matrix in the first column and elements of the second input matrix in the second column. The elements of the two matrices are paired in every possible way, so the number of rows in the resulting matrix will be the product of the numbers of rows in the two input matrices. Note that the resulting matrix specifies a grid of points on the plane.

The `CONTOUR` operator returns a matrix which contains the piecewise-linear contour curves for a surface corresponding to a function of two variables, $f(x, y)$. The first argument must be a 3-column matrix. The first two columns of the first argument must be a tensor product basis for a rectangular region and the third column contains the values of $f(x, y)$. The second argument is a column vector which contains the level values at which the contours are to be determined.

`LINETYPE MARKER` is designed for a special format of the matrix of points to be drawn. The `CONTOUR` operator produces such a matrix, but you can construct a `LINETYPE MARKER` matrix in many other ways. The special format for a `LINETYPE MARKER` matrix, `M`, is that `M[1,1]` is an arbitrary number called the marker value; often 1.E300 is used. All the rows following a row beginning with a marker value up to another row starting with a marker value (or the end of the matrix) form a group of points defining a polygonal curve to be drawn with successive straight-line segments. For each group, these line-segments are drawn with dashed lines with the dash length in inches given by the second value (in column 2) of the marker row leading the group. `M[1,2]` is such a dashed line parameter for the first group of points.

More elaborate linetypes can be drawn by specifying a 7-tuple of numbers designated line1, gap1, line2, tick1, tick2, thick1, and fillsw. These seven numbers determine the characteristics of the line. A line is composed of the following components: a 1st line segment, a gap, a 2nd line segment, and a 2nd gap the same size as the first gap. The middle of the 1st gap contains the first tick mark. The middle of the 2nd gap contains the 2nd tick mark. This pattern of line-segments, gaps and tick marks are repeated cyclicly. These components may be defined by means of the following seven scalars. They are listed in the order in which they must be given. These scalars have the following meanings:

line1 length of the 1st solid line segment

gap1 length of the gap between 1st 2nd solid line

line2 length of the 2nd solid line segment

tick1 length of the 1st tick mark

tick2 length of the 2nd tick mark

thick1 thickness of the boxes that form line segments and tickmarks

fillsw thick box fill switch (-1 means no filling; 0 means solid fill; $k > 0$ means use k horizontal fill lines)

The first six of these scalars have an associated unit which may be set with an optional units qualifier. If there is no units qualifier, the default unit is FFRACT. The options for units are

FFRACT horizontal frame fraction units: 1 = width of the frame

INCHES screen inches

FRACT horizontal screen fraction units: 1 = width of the screen

IFRACT horizontal image fraction units: 1 = width of the image

WORLD world vertical (y-axis) units

Some examples:

`(0.1,0,0,0.2,0.2,0,0)` defines tic-mark lines `0.2 FFRACT` units long placed every `0.1 FFRACT` units on a solid line.

`(1E-4,0.1,0,0.2,0.2,0,0)` defines alternating dots and tic-mark lines of length `0.2 FFRACT` units placed `0.05 FFRACT` units apart.

`(0,.1,0,0.2,.2,0,0)` defines tic-mark lines of length `0.2 FFRACT` units occurring every `0.1 FFRACT` units on an invisible line.

The `POINTTYPE`-clause specifies the pointtype symbol used to mark the points of the graph. Either the name or the number of a pointtype symbol may be used. In Figure 1.3 , we wanted circles, so we used the clause `POINTTYPE CIRCLE`. `POINTTYPE 13` might have been typed as well. (The default is no pointtype symbol, e.g. `POINTTYPE NONE`, as in Example 1-1 where no `POINTTYPE`-clause was used.) The keyword `POINTTYPE` may be abbreviated to `PT`. There are many pointtype symbols available in MLAB, which are listed in the table below.

`NONE`	(0)	no point symbol.
`VBAR`	(1)	vertical bar.
`CROSSPT`	(2)	+-shaped symbol.
`TRIANGLE`	(3)	triangle symbol.
`SQUARE`	(4)	box symbol.
`STAR`	(5)	a five-pointed star.
`HBAR`	(6)	horizontal bar.
`OCTAGON`	(7)	eight sided figure.
`XPT`	(8)	X-shaped symbol.
`LTICK`	(9)	leftward tic mark.
`RTICK`	(10)	rightward tic mark.
`UTICK`	(11)	upward tic mark.
`DTICK`	(12)	downward tic mark.
`CIRCLE`	(13)	variable resolution circle.
`ARROW`	(14)	vector field arrows.
`DOTPT`	(15)	a small dot.

`DIAMOND`	(16)	a diamond symbol.
`XERRBAR`	(17)	horizontal error bars.
`YERRBAR`	(18)	vertical error bars.
`XLOGTICK`	(19)	horizontal logarithmically-spaced tics.
`YLOGTICK`	(20)	vertical logarithmically-spaced tics.
`NBAR`	(21)	normal bar.
`TBAR`	(22)	tangent bar.
`RBAND`	(23)	solid band extending to the right image margin.
`LBAND`	(24)	solid band extending to the left image margin.
`UBAND`	(25)	solid band extending to the top image margin.
`DBAND`	(26)	solid band extending to the bottom image margin.
`DRBAND`	(27)	dotted band extending to the right image margin.
`DLBAND`	(28)	dotted band extending to the left image margin.
`DUBAND`	(29)	dotted band extending to the top image margin.
`DDBAND`	(30)	dotted band extending to the bottom image margin.
`ARROWTIP`	(31)	directed arrowhead.
`"c"`	–	the character `’c’` in a specified font.

Point types corresponding to convex shapes are drawn in outline. If filled point types are desired, simply negate the point type name or value. The following table summarizes values of point types resulting in filled shapes.

`-TRIANGLE`	(-3)	filled triangle symbol.
`-SQUARE`	(-4)	filled box symbol.
`-STAR`	(-5)	filled five-pointed star.
`-OCTAGON`	(-7)	filled eight sided figure.
`-CIRCLE`	(-13)	filled circle.
`-DIAMOND`	(-16)	filled diamond symbol.

Example 2-3: Draw a scatterplot of some data as in Example 1-1 using no lines between the points and small triangles to mark the data points.

```
* DELETE M,W
* M COL 1 = 0:6
* M COL 2 = LIST(.1,.3,.89,.6,.2,.05)
* DRAW M LT NONE PT TRIANGLE PTSIZE .04
* VIEW
```

The resulting picture is shown in Figure 2.3.

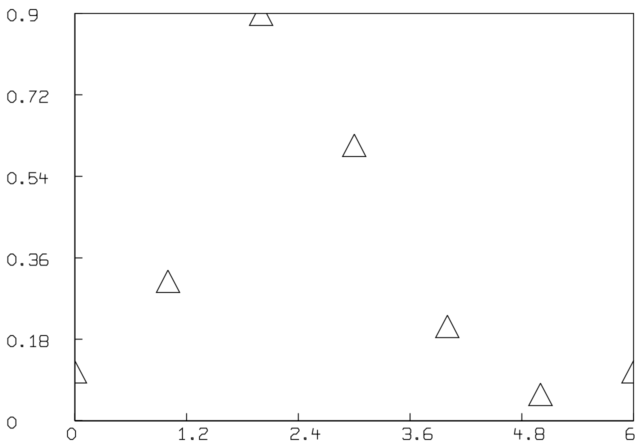


Figure 2.3: Data drawn with triangles.

To mark the data points in this example with solid filled triangles, the following MLAB commands can be used:

```
* DELETE W
* DRAW M LT NONE PT -TRIANGLE PTSIZE .04
* VIEW
```

The resulting picture is shown in Figure 2.4.

The PTSIZE-clause: PTSIZE .04 specifies the size of the pointtype symbols. The value .04 is in FFRACT (frame fraction) units by default, which means that the point symbol size will be 4% of the horizontal extent of the frame rectangle.

The PTSIZE-clause takes its argument in FFRACT or frame fraction units by default. Frame fraction units are units such that the full width of the frame is 1. By default, the frame is the full screen. The argument in the PTSIZE-clause can also be specified in IFRACT units or image fraction units–meaning units such that the full width of the image is 1; in FRACT or screen fraction units–meaning units such that the full width of the screen is 1; in INCHES units; and in WORLD units.

In Example 2-3, we made a scatterplot of a single data set. When more than one data set is to be displayed, we can use different pointtype symbols and/or different colors. The next two examples illustrate these two approaches.

Example 2-4: Draw a scatterplot of two data sets on the same graph using different pointtype symbols. Construct the two data sets in the matrices M1 and M2. The MLAB statements to do this are:

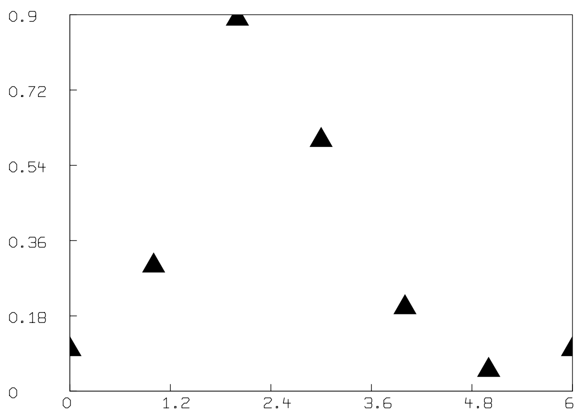


Figure 2.4: Data drawn with filled triangles.

```
* DELETE W
* M1 COL 1 = 0:6; M1 COL 2 = LIST(0,.1,.2,.3,.4,.5,.6)
* M2 COL 1 = 0:6; M2 COL 2 = LIST(0,.2,.4,.6,.8,1.,1.2)
* DRAW M1 LT NONE PT CIRCLE PTSIZE .03
* DRAW M2 LT NONE PT XPT PTSIZE .03
* VIEW
```

The resulting picture is shown in Figure 2.5.

This approach is satisfactory when there are not too many points in each data set. As the number of points grows, however, the use of a large pointtype symbol causes the picture to become confusing. With many data points, it is more effective to use a very small pointtype or a single dot for each point, contrasting the two data sets by the color used to plot each one. Colored dots are effective on color displays but not on monochrome displays or printers.

Example 2-5: Draw scatterplots of two large data sets with distinct pointtype symbols, also distinguishing between them by using different colors. The two data sets are defined by normal random numbers and placed in the matrices N1 and N2. The MLAB statements to do this are:

```
* DELETE W
* N1 COL 1 = 1:200; N1 COL 2 = NORMRAN ON 0^^200
* N2 COL 1 = 1:200; N2 COL 2 = NORMRAN ON (0&'1)^^200
* DRAW N1 LT NONE PT SQUARE PTSIZE .006 COLOR RED
* DRAW N2 LT NONE PT XPT PTSIZE .006 COLOR GREEN
* VIEW
```

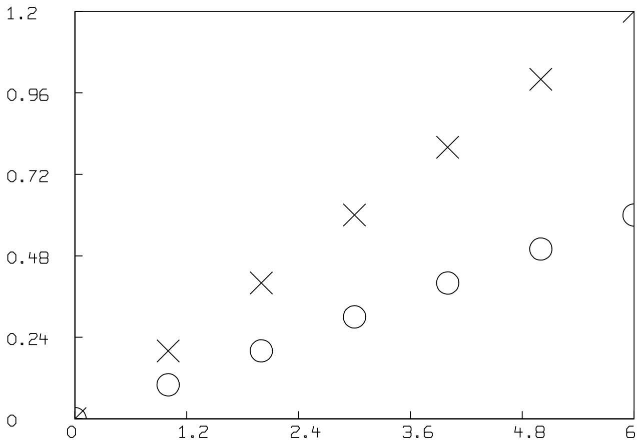


Figure 2.5: Two data sets drawn with different pointtype symbols.

The resulting picture is shown in Figure 2.6.

The `ON` operator acting on the built-in function `NORMRAN` with one vector argument was used and explained in an earlier example. The second use of the `ON` operator in this example, demonstrates the `ON` operator acting on the built-in function `NORMRAN` with a two column matrix argument. The first column specifies the seed value for the normal distribution generator. The second column specifies the mean of the normal distribution from which the random variable values are generated. The matrix (0&'1)^^200 defines a 200-row, 2-column matrix, each row of which is (0,1). Thus we have computed 200 samples from a normal distribution with unit mean (and unit variance by default).

The `COLOR`-clause: `COLOR RED` specifies that data from matrix `N1` is to be drawn using the color red. Similarly, the `COLOR`-clause: `COLOR YELLOW` specifies that data from matrix `N2` is to be drawn using the color yellow.

It is also possible to draw error bars on data plots, as the next two examples show.

Example 2-6: Draw a graph of some data containing replicate samples (y-values) for each value of x with y-error bars about the mean value to reflect the standard error of each group of replicates. Thus, the x-values correspond to group numbers. Suppose that the data is in the ASCII file `DATA.DAT` in 61 row by 2 column format, where the first few rows are:

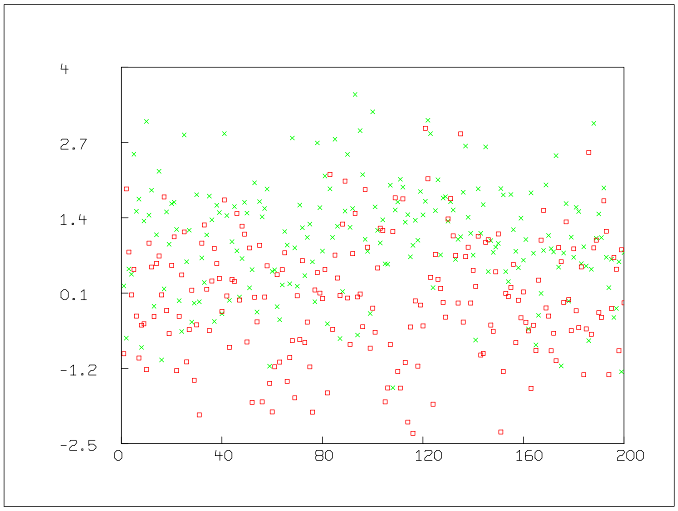


Figure 2.6: Two data sets drawn with 2 different pointtypes and colors.

0	.984
1	1.808
2	2.87
0	1.255
6	3.25
.	.
.	.
.	.

The data is read into a matrix `M` by the command

```
* M = READ(DATA,61,2)
```

An MLAB command sequence to produce the Figure 2.7 is given below.

```
* MS = SORT(M)
* N = MS[NROWS(MS),1]+1
* FOR I = 1:N DO {
>      G = MS[1,1];
>      MC = MS &' (MS COL 1=G);
>      MC = COMPRESS(MC,3);
>      MV ROW I = G &' MEAN(MC COL 2) &' STDDEV(MC COL 2);
>      MS = COMPRESS(MS,1,G);
>     }
```

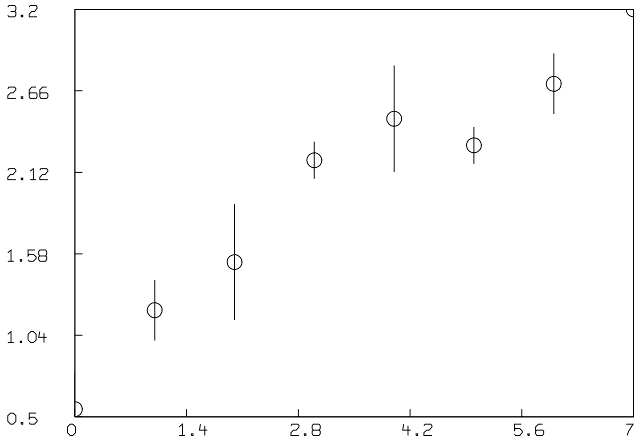


Figure 2.7: Data with y error bars.

```
* DELETE W
* DRAW MV COL 1:2 LT NONE PT CIRCLE
* DRAW MV COL 1:2 LT NONE PT YERRBAR PTSIZE MV COL 3
* VIEW
```

The `SORT` function produces a copy of the input matrix, sorted on column 1 in increasing order, by default. The `NROWS` function produces the number of rows of a matrix. The `FOR`-loop creates a three column matrix `MV`; the first column of `MV` contains the different x-variates, and the second and third columns of `MV` contain the mean and standard deviation of the corresponding y-variates. This is achieved by building an intermediate 3 column matrix, `MC`, the first two columns of which are a copy of the matrix `MS`. The elements of the third column of `MC` are 1 if the corresponding first column element equals the first element of `MS`; otherwise they are 0. The first call to `COMPRESS` removes from `MC` those rows for which the indicated column (3 in this case) has the value 0. The second call to `COMPRESS` removes from `MS` those rows for which column 1 has the value `MS[1,1]`.

The x- and mean-y pairs in `MV` are drawn as points on the graph, with pointtypes `CIRCLE` and `YERRBAR`. Note that the argument to the `PTSIZE`-clause for the error bars is a vector, `MV COL 3`. When the argument to `PTSIZE` is a vector, each element of the vector specifies the size of each successive pointtype. In this case, the elements of the vector are the standard deviations of the replicate groups and specify the lengths of the error bars.

In the previous example the x-values were replicated, corresponding to repeated measurements of the y-values, so we used y-error bars. Alternatively, if it were the y-values which were replicated, corresponding to repeated measurements of the x-values, then x-error bars would be appropriate.

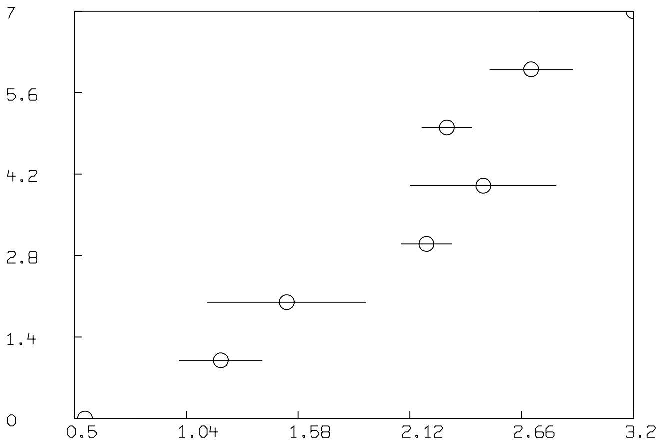


Figure 2.8: Data with x error bars.

Example 2-7: Draw a graph of the data derived from the previous example, with x-error bars. In the context of the previous example, we suppose that the roles of the 1^{st} and 2^{nd} columns of the matrix M have been interchanged. An MLAB command sequence to produce the Figure 2.8 is given below.

```
* DELETE W
* MS = SORT(M,2)
* N = MS[NROWS(MS),2]+1
* FOR I = 1:N DO {
>     G = MS[1,2];
>     MC = MS &' (MS COL 2=G);
>     MC = COMPRESS(MC,3);
>     MV ROW I = G &' MEAN(MC COL 1) &' STDDEV(MC COL 1);
>     MS = COMPRESS(MS,2,G);
>    }
* DRAW MV COL 1:2 LINE NONE PT CIRCLE
* DRAW MV COL 1:2 LINE NONE PT XERRBAR PTSIZE MV COL 3
* VIEW
```

The resulting picture is shown in Figure 2.8.

This series of commands is the same as that in the previous example except that we SORT on column 2 of the matrix M and DRAW with the XERRBAR pointtype symbol.

The previous 3 examples demonstrated how the PTSIZE clause can take either a scalar or vector argument. The next example shows how the PTSIZE clause with the ARROW pointtype can take a two column matrix as its argument and be used to draw vector fields.

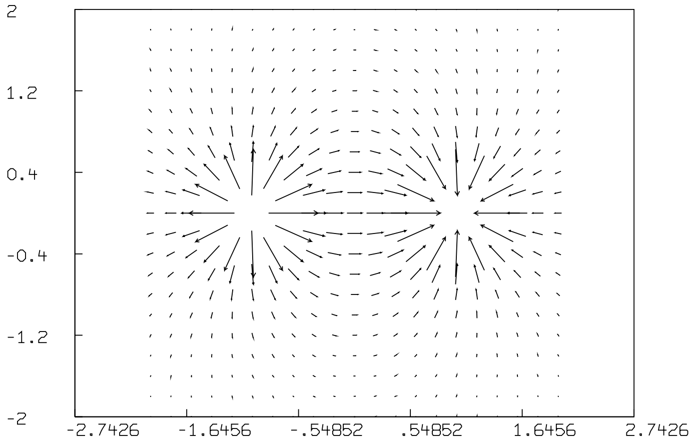


Figure 2.9: Force field of an electric dipole in 2 dimensions.

Example 2-8:

Draw the force field of an electric dipole in the $x - y$ plane. This is done as follows:

```
* DELETE W,I,M
* M = CROSS(-2:2!21,-2:2!21)
* FCT I(X,Y) = -(X-1)/((X-1)^2+Y^2)^1.5 + (X+1)/((X+1)^2+Y^2)^1.5
* FCT J(X,Y) = -Y/((X-1)^2+Y^2)^1.5 + Y/((X+1)^2+Y^2)^1.5
* FCT C(X,Y) = .01*SQRT(I(X,Y)^2+J(X,Y)^2)
* FCT D(X,Y) = ATAN2(J(X,Y),I(X,Y))*180/PI
* N COL 1 = C ON M
* N COL 2 = D ON M
* DEL M ROW 94:96,M ROW 115:117,M ROW 136:138,M ROW 304:306,\
:    M ROW 325:327,M ROW 346:348
* DEL N ROW 94:96,N ROW 115:117,N ROW 136:138,N ROW 304:306,\
:    N ROW 325:327,N ROW 346:348
* DRAW M LT NONE PT ARROW PTSIZE N
* WINDOW ADJUST WMATCH
* VIEW
```

The resulting picture is shown in Figure 2.9.

Here the electric dipole consists of a negative charge at the point (-1,0) and a positive charge at the point (1,0). We construct a grid of sample points with x and y each ranging from $-2, -1.8, -1.6, \ldots, 2$ by using the `CROSS` operator. The set of sample points is stored in matrix `M`. The function `I` represents the x-component of the electric force field at a point (x, y) in the plane, and the function J represents the y-component. We then define the functions C and D which are the magnitude and direction (in degrees) of the force field at a point, respectively. (An arbitrary constant of .01 has been introduced in the function C so that the magnitudes of the arrows fit in the figure without overlapping.) Applying functions C and D to matrix M, we build the matrix N. The vector field is then drawn by graphing the points in M with no connecting lines and with pointtype ARROW. The PTSIZE clause has matrix N as its argument. The numbers in the first column of N determine the size of the arrows and the numbers in the second column determine the angle (rotating from the horizontal in a counterclockwise fashion and expressed in degrees) of the arrow. We have deleted those elements of the grid near the point charges which have force field magnitudes so large that the resulting arrows are too large for the picture.

Chapter 3

Using the title statement

We have already seen two ways to modify the default window: adding a curve by using the DRAW command and removing a curve by using the DELETE command. This chapter introduces a third command, the TITLE command, which permits the addition of titles to any area in the frame.

Example 3-1: Place a title on a graph. Suppose we have drawn a graph of the function $f(x) = \mathrm{J}_{1/2}(x)$, (the J Bessel function of order 1/2, BESSJ(X,.5) in MLAB) for x between 0 and 10. In order to remind us of the function which was drawn, we will place the text Graph of BESSJ(X,.5) X = 0:10!100 above the graph as a title. The MLAB commands to make this graph are:

```
* TOP TITLE "Graph of BESSJ(X,.5), for X = 0:10!100"
* FCT JHALF(X) = BESSJ(X,.5)
* DRAW POINTS(JHALF,0:10!100)
* VIEW
```

The resulting picture is shown in Figure 3.1.

The TITLE command creates a title for a graph. The TOP qualifier specifies the placement of the title to be centered in the space above the top of the image. The string comprising the title appears in quotes.

Example 3-2: In this example we add titles to the axes of a graph. Suppose we have some bivariate data, e.g. heights and ages for a group of children. We will plot the weights on the horizontal axis against the heights on the vertical axis. The vertical y-axis will be labeled with HEIGHT IN INCHES. The horizontal x-axis will be labeled with WEIGHT IN POUNDS. The MLAB commands to make this graph are:

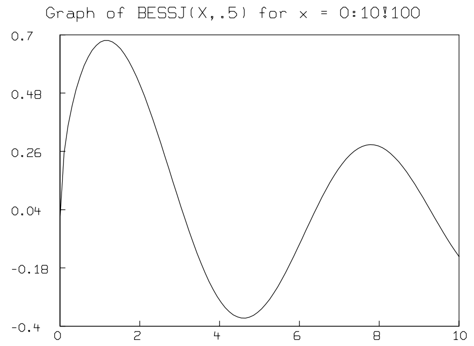


Figure 3.1: Titled graph of $\mathrm{J}_{1/2}(x)$ for $x = 0$ to 10.

```
* DELETE W
* M COL 1 = LIST(47,61,55,67,56,61,42,45,61,42)
* M COL 2 = LIST(27,52,43,56,82,91,87,38,92,84)
* DRAW M LT NONE PT CROSSPT
* LEFT TITLE "HEIGHT IN INCHES"
* BOTTOM TITLE "WEIGHT IN POUNDS"
* VIEW
```

The graph is shown in Figure 3.2.

The BOTTOM qualifier to the TITLE command specifies that the placement of the title is to be horizontally-centered in the frame and placed in the space below the bottom of the image. Because the image rectangle in W is shifted right in the frame in order to make room for axis labels and a possible left title, the bottom title shown above does not appear centered with respect to the image rectangle. The LEFT qualifier to the TITLE command specifies the placement of the title to be centered, running vertically up the left side of the image. The RIGHT qualifier to the TITLE command is also available. It places a centered title, running vertically up the right side of the image.

The next example shows how titles may be placed anywhere in the frame by using an AT clause in the TITLE statement.

Example 3-3: Use titles to label each of several curves in a window. Suppose we have drawn graphs of the three curves: $\mathrm{J}_0(x)$, $\mathrm{J}_1(x)$, and $\mathrm{J}_2(x)$, (J Bessel functions of order 0, 1, and 2) for x in [0,10]. We can label each curve with the order of the corresponding Bessel function. The MLAB statements to do this are:

```
* DELETE W
```

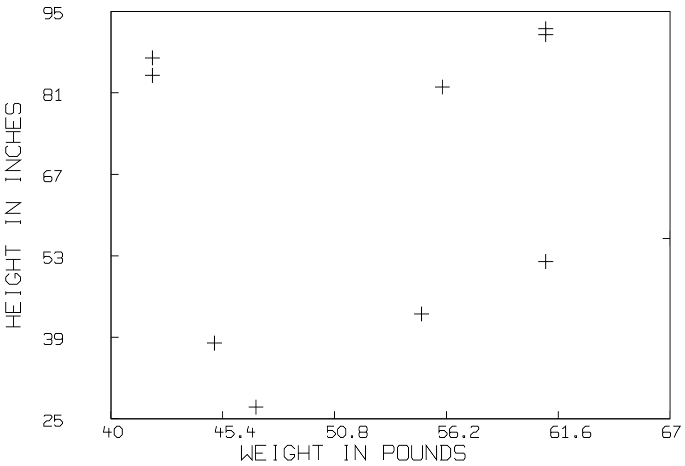


Figure 3.2: Graph with labeled axes.

```
* XL = 0:10!100
* DRAW XL &' (BESSJ ON XL &' 0) COLOR RED
* DRAW XL &' (BESSJ ON XL &' 1) COLOR GREEN
* DRAW XL &' (BESSJ ON XL &' 2) COLOR TURQUOISE
* TOP TITLE "J BESSEL FUNCTIONS"
* TITLE "order = 0" AT (.25,.8) COLOR RED
* TITLE "order = 1" AT (.35,.67) COLOR GREEN
* TITLE "order = 2" AT (.5,.55) COLOR TURQUOISE
* VIEW
```

The resulting graph is shown in Figure 3.3.

Here we have used the `AT`-clause to specify the position of a title. The `AT`-clause takes an ordered pair of numbers as its argument and an optional units designation. If no units are designated, the ordered pair is assumed to be in frame fraction units. If units are specified, the `AT`-clause permits specification of the position of the title in `IFRACT`, `FRACT`, `INCHES`, `FINCHES`, and `WORLD` coordinates. `IFRACT` coordinates refer to the image fraction coordinate system where (0,0) is the lower left corner of the image rectangle and (1,1) is the upper right corner of the image rectangle. `FRACT` coordinates refer to the frame fraction cooordinate system where (0,0) is the lower left corner of the frame rectangle and (1,1) is the upper right corner of the frame rectangle. `INCHES` coordinates refer to the screen coordinate system where (0,0) is the lower left corner of the screen rectangle and (`SXINCHES`,`SYINCHES`) is the upper right corner of the screen rectangle. Here `SXINCHES` and `SYINCHES` are the maximum x- and y-extents of the screen in inches. `FINCHES` coordinates refer to the frame inches coordinate system where (0,0) is the lower left corner of the frame and (`FXINCHES`,`FYINCHES`) is the upper right corner of the frame.

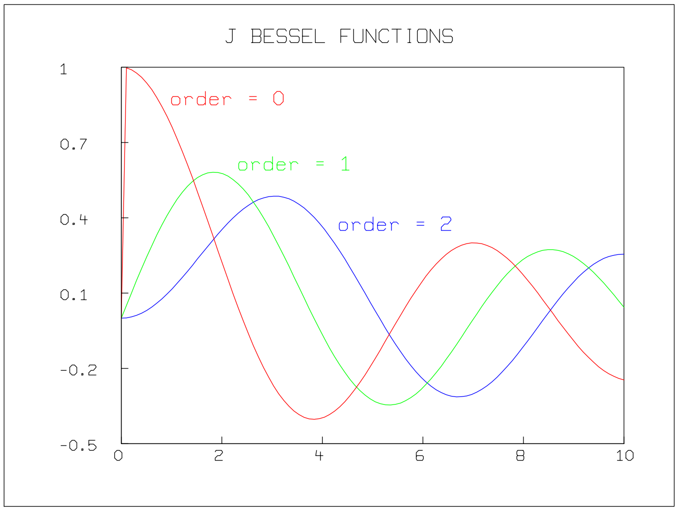


Figure 3.3: Titled curves.

Here FXINCHES and FYINCHES are the x- and y-extents of the frame in inches. Finally, WORLD coordinates refer to the Cartesian coordinate system which is mapped to the image rectangle.

The previous example also used COLOR-clauses. The COLOR-clause in the TITLE statement determines the color to be used when drawing the title text. It is convenient to use the 16 predefined MLAB colors, given in the table below.

Color Name	DOS and Linux with X-Windows	MSWindows and Macintosh OS9/X
BLACK	0	0
GREY	43	2185
WHITE	1	1
RED	49	3841
GREEN	13	241
YELLOW	61	4081
BLUE	4	16
VIOLET	36	2351
PURPLE	36	2351
ORANGE	57	4001
PINK	59	4029
AQUA	32	1006
CHARTREUSE	45	2289
BROWN	37	2115
ROSE	55	3947
TURQUOISE	31	1245

Colors are merely integer constants which specify a color. For Linux with X-Windows and DOS with EGA or VGA display, the values of the color symbols are listed in the table above in the center column. Values for Microsoft Windows and Macintosh with OS9 and OSX displays are shown in right column.

The following example demonstrates all of the predefined colors.

Example 3-4: Enlarge the image of an MLAB graphics window to the full frame and draw rectangles filled with each of the predefined MLAB colors and labeled with the filled color name.

```
* RESET
*
* /* draw a black filled and labeled rectangle at the lower left corner
*    of the window */
* A = 0; B = 0;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "black" COLOR WHITE AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR BLACK IN W
* DEL W.XAXIS, W.YAXIS /* delete the default axes */
* NO IMAGEBOX          /* omit the image border rectangle */
* IMAGE 0 TO 1, 0 TO 1 /* enlarge the image rectangle to the extent of the frame */
*
* /* draw a white filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 0; B = 1;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "white" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR WHITE
*
* /* draw a grey filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 0; B = 2;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "grey" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR GREY
*
* /* draw a red filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 0; B = 3;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "red" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR RED
*
* /* draw a green filled and labeled rectangle to the right of the first
*    rectangle defined above.*/
* A = 1; B = 0;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "green" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR GREEN
*
* /* draw a yellow filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 1; B = 1;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "yellow" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR YELLOW
*
* /* draw a blue filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 1; B = 2;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "blue" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR BLUE
*
* /* draw a violet filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 1; B = 3;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "violet" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR VIOLET
*
* /* draw a purple filled and labeled rectangle in at the bottom of the
*    third column.*/
* A = 2; B = 0;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "purple" COLOR BLACK AT AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR PURPLE
*
* /* draw an orange filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 2; B = 1;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "orange" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR ORANGE
*
* /* draw a pink filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 2; B = 2;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "pink" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR PINK
*
* /* draw an aqua filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 2; B = 3;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "aqua" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR AQUA
*
* /* draw a chartreuse filled and labeled rectangle at the bottom of a
*    column to the right of the previously defined column.*/
* A = 3; B = 0;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "chartreuse" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR CHARTREUSE
*
* /* draw a brown filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 3; B = 1;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "brown" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR BROWN
*
* /* draw a rose filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 3; B = 2;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "rose" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR ROSE
*
* /* draw a turquoise filled and labeled rectangle above the previously defined
*    rectangle.*/
* A = 3; B = 3;
* M = SHAPE(5,2,LIST(A,B,A+1,B,A+1,B+1,A,B+1,A,B))
* TITLE "turquoise" COLOR BLACK AT (A+.05,B+.05) WORLD FONT 7 SIZE .02 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR TURQUOISE
* VIEW
```

The resulting picture is shown in Figure 3.4.

The MLAB function `COLORX(R,G,B)` can be used as the argument of a `COLOR`-clause when drawing text, axes, or curves in colors other than those defined by predefined color names. `COLORX` takes three arguments, each in the range `0:1`, which specify the red, green, and blue components of the desired color. The following example demonstrates how `COLORX` is used to draw various shades of blue.

Example 3-5: Draw rectangles filled with various shades of blue and labeled with the `DCOLOR` clause that generates the color.

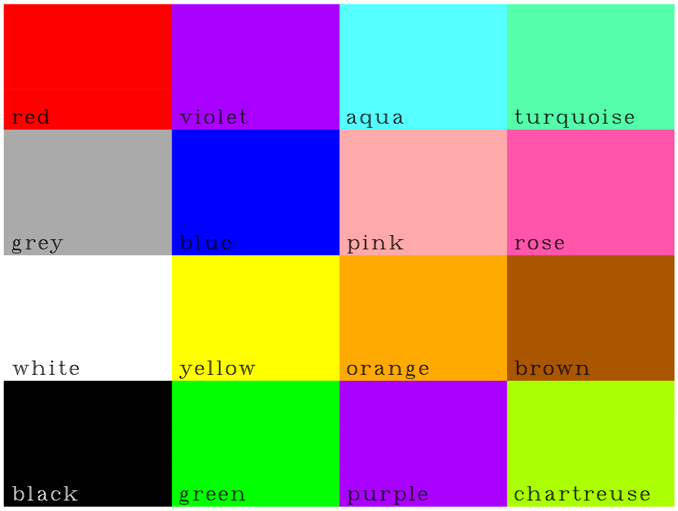


Figure 3.4: Rectangles filled with each of the predefined MLAB colors.

```
* RESET
* A = 0; B = 0;
* M = SHAPE(5,2,LIST(A,B,A+10,B,A+10,B+2.5,A,B+2.5,A,B))
* TITLE "colorx(0,0,1)" COLOR YELLOW AT (A+.15,B+.15) WORLD FONT 7 SIZE .03 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR COLORX(0,0,1) IN W
* DEL W.XAXIS, W.YAXIS
* NO IMAGEBOX
* IMAGE 0 TO 1, 0 TO 1
*
* B = 2.5;
* M = SHAPE(5,2,LIST(A,B,A+10,B,A+10,B+2.5,A,B+2.5,A,B))
* TITLE "colorx(0,0,.67)" COLOR YELLOW AT (A+.15,B+.15) WORLD FONT 7 SIZE .03 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR COLORX(0,0,.67)
*
* B = 5;
* M = SHAPE(5,2,LIST(A,B,A+10,B,A+10,B+2.5,A,B+2.5,A,B))
* TITLE "colorx(0,0,.33)" COLOR YELLOW AT (A+.15,B+.15) WORLD FONT 7 SIZE .03 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR COLORX(0,0,.33)
*
* B = 7.5
* M = SHAPE(5,2,LIST(A,B,A+10,B,A+10,B+2.5,A,B+2.5,A,B))
* TITLE "colorx(0,0,0)" COLOR YELLOW AT (A+.15,B+.15) WORLD FONT 7 SIZE .03 FFRACT
* DRAW FILL(M,300) LT ALTERNATE COLOR COLORX(0,0,0)
* VIEW
```

The resulting picture is shown in Figure 3.5.

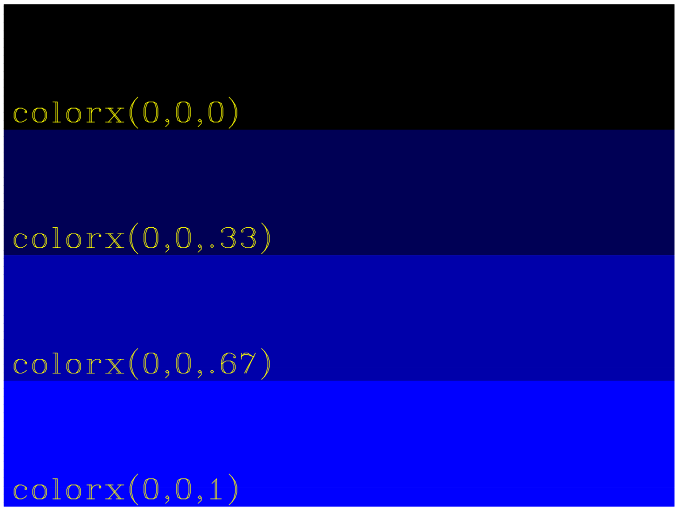


Figure 3.5: Rectangles filled with various shades of blue.

The characters in the `TITLE` string may be any combination of upper and lower case letters or numbers. In addition, there are embedded string control codes which can alter the properties of character placement. The next example illustrates how characters can be drawn in an arc using embedded string control codes.

Example 3-6: Draw the string “This was done using MLAB” in a semi-circular arc across the default window. The following MLAB commands do this:

```
* DEL W
* TITLE B = "'-7A'5TThis was done using '21TMLAB", ANGLE 90 AT (1,1)\
:     WORLD SIZE .5 WORLD
* VIEW
```

The resulting picture is shown in Figure 3.6.

Each embedded string control code begins with an apostrophe. Two adjacent apostrophes denote the actual character (`'`). The embedded string control code `'-7A` rotates the baseline of the first character `7` degrees clockwise from the initial direction in which the string is printed. (Because of the `ANGLE 90` clause, the initial direction for printing the string is in the vertical direction of increasing y.) The baseline of the second character is rotated `14` degrees, the baseline of the third character is rotated `21` degrees, etc. `'0A` stops this incremental rotation and `'nA` will begin rotating the text in a counterclockwise direction by `n` degrees.

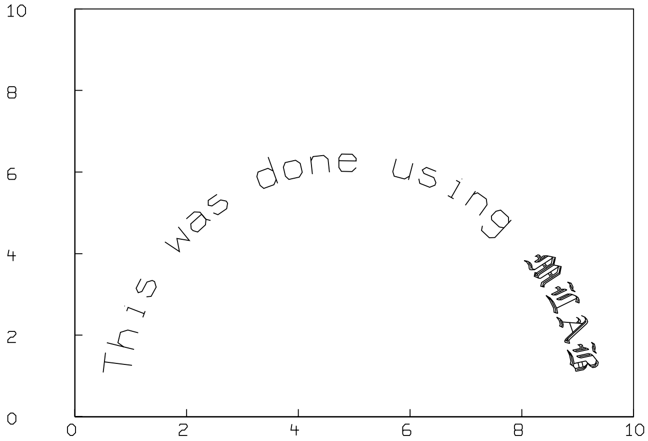


Figure 3.6: Letters forming a semi-circle using embedded string control codes.

The embedded string control code `'nT` changes the font table number for the text from the current font to font table number n. This code was used twice in the example to start the string in font table 5 and then again to change to font table 21.

MLAB offers more than fifteen additional embedded string control codes. The codes are:

`'nS` changes the size of all the following characters to be `n` times the baseline size.

`'nW` changes the relative width of the characters to be `n` times the baseline width, by keeping the height unchanged and modifying the width as required.

`'nH` changes the relative height of the characters to be `n` times the baseline height, by keeping the width unchanged and modifying the height as required.

`'nT` changes the font table number to be used from the current font to font table number `n`.

`'nU` shifts the following characters up off the baseline. The number `n` can be positive or negative where 1 unit of `n` represents the nominal height of a character. The `n` value need not be an integer. `'.5U` works fine for superscripts.

`'nD` shifts the following characters down from the baseline. The number `n` can be positive or negative and `'1.66667D` is equivalent to a linefeed.

`'nA` rotates each following character by `n` degrees incrementally, so that the second following character is rotated 2`n` degrees, and so on. The baseline is thus converted to a circular arc. `'0A` stops this incremental rotation and `'-nA` will go the opposite direction.

`’R` resets the above parameters to what they were at the start of the string. Note that this allows you to specify a particular font such as Script (font 19) in the middle of a text string and then revert back to whatever font you started with, no matter what the starting font table was.

`’Cx` converts the character “x” to a control character. For alphabetic characters, upper and lower case are equivalent; thus `’CB` and `’Cb` both produce the octal ASCII code 002 corresponding to CTRL-b. The defining formulas for the ASCII codes produced by `’Cx` are ASCII(′Cx) = ASCII(x) − 64 for ASCII(x) < 97, and ASCII(′Cx) = ASCII(x) − 96 for 96 < ASCII(x) < 123.

`’M` is equivalent to `’CM’CJ`. `’CM` is a CR (carriage return = octal 12), and `’CJ` is a LF (line feed = octal 15).

`’nB` is used to specify `n` sequential backspaces, and fractional backspacing, as in `2.5B`, is honored. Fractional forward spacing is also allowed, as in `’-1.33B`, by using `n`<0. `’B` is equivalent to `’1B`. Backspaces space back variable amounts according to the widths of the previous characters.

`’nF` is used to move forward `n` spaces. Fractional spaces, for example `’.3F`, are honored. `’nF` is equivalent to `-nB`.

`’X` is approximately the same as `’.5B`, but the amount backspaced is adjusted so that the text of the next character will be precisely centered on the text of the just-previous character.

`’Y` causes vertical bars to be inserted before each character, until another `’Y` is given.

`’nO` changes the color of the succeeding characters to be color `n`.

`’nI` iteratively repeats the next drawn character `n` times, for a total of `n`+1 occurrences.

`’E` extends to the boundary point just after the farthest character from the starting point and continues from there.

`’Z` underlines the following characters until another `’Z` is given.

Using only `TITLE` commands, MLAB becomes a powerful typesetting program to create posters, display text, and slides for presentations.

Example 3-7: Draw a title containing a string which defines $F(t)$ as the error function, i.e. the integral of a Gaussian from 0 to t, to demonstrate some more of the embedded control codes. This is accomplished using the MLAB commands:

```
* DEL W
* WINDOW 0 TO 4, 4 TO 6 IN W
* TITLE "F(t) = ’1U’Z 2 ’Z’4B’2D’25Tk’.6Uqqq’.6D’2.5B’13Tp ’1U’26T’2H’2SI’5T\
:     ’.5H’.5S’1B’2.5DO’6Ut’3.5De’.6U-x’.6U’.5S2’2S’1.2Ddx" AT (1,5) WORLD
* VIEW
```

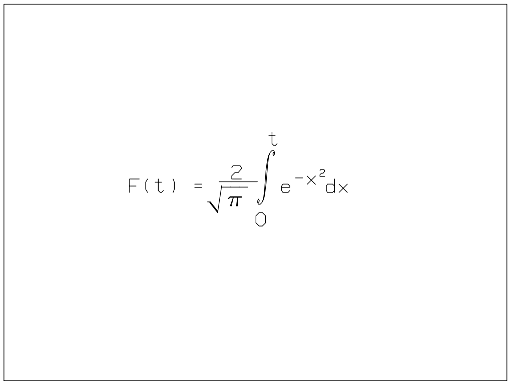


Figure 3.7: A title which demonstrates embedded string control codes.

The resulting picture is shown in Figure 3.7.

The first two embedded string codes in this example serve to shift the characters up 1 character height unit and begin underlining characters. Underlining creates the horizontal bar of the fraction. The second `'Z` turns underlining off. `'4B` backspaces 4 character positions, `'2D` shifts the characters for the denominator down 2 character heights, and `'25T` switches the font to be used for subsequent characters to font number 25. Note that lower case `k` in font 25 is the beginning of the square root sign. `'.6U` shifts the following characters up off the current baseline .6 units. Lower case `q` in font 25 is the extension of the square root sign. `'.6D` shifts the following character down .6 units and `'2.5B` backspaces underneath the square root sign. The current font is then changed to font number 13 in which the letter p is π. `'1U` shifts the following characters up 1 unit. `'26T` changes the font to font number 26. `'2H'2S` scales the relative height and width of the subsequent characters up by a factor of 2. The letter `I` in font number 26 is an integral sign. After resetting the current font table to table number 5 with `'5T` and resetting the scale of relative height and width of the characters to unity by `'.5H'.5S`, we backspace by typing `'1B` and shift down 2.5 character height units with `'2.5D` to print the lower limit of the integral. The upper limit of the integral is printed by shifting up 6 character units (`'6U`) and typing `t`. The integrand is then printed by shifting down 3.5 character units (`'3.5D`), typing `e`, shifting up six tenths of a character unit and typing `-x`, shifting up yet another six tenths of a character unit, reducing the relative size of the subsequent characters and typing `2`, returning the relative size of characters to unity by typing `'2S`, dropping down to the baseline by typeing `'1.2D`, and finally typing `dx`.

The next example illustrates some of the many different fonts available for drawing titles.

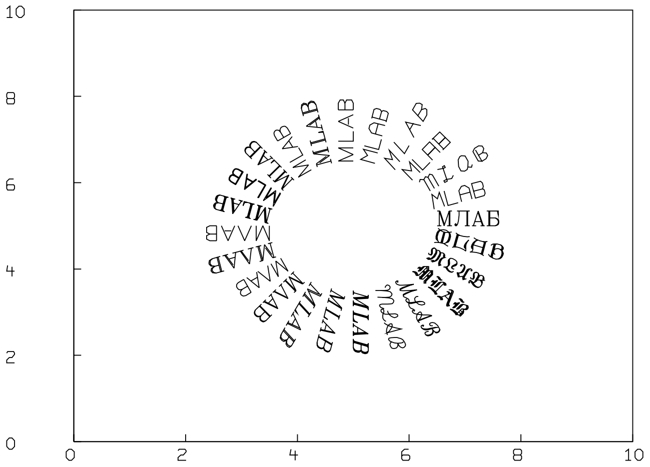


Figure 3.8: “MLAB” in 24 different Hershey fonts.

Example 3-8: Draw the word “MLAB” as the spokes of a wheel using a different Hershey font for the characters in each spoke.

This is accomplished using the following MLAB commands:

```
* DEL W,C
* FOR C = 1:24 DO {
>    AN = C*360/24;
>    TITLE "MLAB" AT (5.+1.5*COSD(AN),5.+1.5*SIND(AN)) WORLD \
>        ANGLE AN FONT C;
> }
* VIEW
```

The resulting picture is shown in Figure 3.8.

The AT-clause of the TITLE statement determines the starting position of each “MLAB” spoke in WORLD units. The ANGLE-clause of the TITLE statement determines the ray along which the title will be drawn. Starting with a horizontal ray directed toward increasing x, the ANGLE-clause specifies the angle the ray is rotated counterclockwise from the horizontal. The FONT-clause of the TITLE statement determines the font to be used in drawing the string in quotes.

The preceding example demonstrated text rendered in 24 of the 34 so-called *Hershey* fonts available in MLAB. These fonts are named after their original creator who developed them in the 1960’s. A complete table of the characters in the 34 MLAB Hershey fonts appears in Appendix A.

When MLAB graphics are displayed on PCs with Microsoft Windows or Macintosh OS9 or OSX operating systems, or plotted on a PostScript printer device, there are an additional 13 fonts available for use. These are the so-called *TrueType* fonts– a set of fonts developed in the 1980's and used in most word-processing programs. A complete table of the characters in each of the 13 TrueType fonts available in MLAB also appears in Appendix A.

TrueType fonts are not ordinarily available on PCs with Linux and X-Windows operating systems unless specifically installed. If a TrueType font is specified for text in an MLAB session on a system where TrueType fonts are not available, Hershey font 1 is used instead. Although a TrueType font may not be available on the display screen, it will be available on the output printer if the output printer is a PostScript printer.

The following table lists font numbers to be used in the `FONT`-clause of an MLAB `TITLE` statement and the name of the resulting TrueType font.

Font number	Microsoft Windows and Postscript	Macintosh OS9/X
-2	Courier New	Courier
-3	Courier New Bold	Courier Bold
-4	Courier New Italic	Courier Oblique
-5	Courier New Bold Italic	Courier Bold Oblique
-6	Times New Roman	Times Roman
-7	Times New Roman Bold	Times Bold
-8	Times New Roman Italic	Times Italic
-9	Times New Roman Bold Italic	Times Bold Italic
-10	Arial	Helvetica
-11	Arial Bold	Helvetica Bold
-12	Arial Italic	Helvetica Oblique
-13	Arial Bold Italic	Helvetica Bold Oblique
-14	Symbol	Symbol

These TrueType fonts are demonstrated in the following example.

Example 3-9: Draw the word "MLAB" as the spokes of a wheel using a different TrueType font table for each spoke.

```
* DEL W,C
* FOR C = 1:13 DO {
>    AN = C*360/13;
>    TITLE "MLAB" AT (5.+1.5*COSD(AN),5.+1.5*SIND(AN)) WORLD \
>        ANGLE AN FONT -(C+1);
> }
* VIEW
```

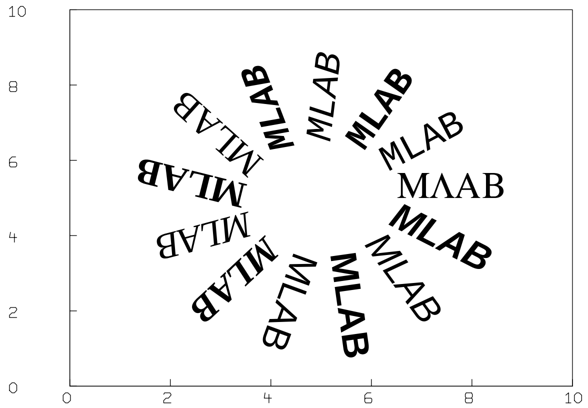


Figure 3.9: "MLAB" in 13 different TrueType fonts.

Note, the argument of the `FONT`-clause in this example is a negative number. The resulting picture is shown in Figure 3.9.

Chapter 4

Using the window and image statements

In the previous chapter, we attempted to draw the MLAB title as spokes of a circle. Due to the dimensions of the window and the image, the circle appeared to be elliptical. This is because the aspect ratio did not equal 1, i.e. the ratio of the range of x-coordinates in world units to the range of x-coordinates in inches did not equal the ratio of the range of y-coordinates in world units to the range of y-coordinates in inches. One consequence of the default image's aspect ratio not equalling one is that geometric shapes can be distorted. While this is not usually a problem, it can be annoying when a curve which is known to be a circle turns out to be an ellipse on the screen. In the next example we show how to re-image a window so that the aspect ratio is one.

Example 4-1: Draw a circle centered at $b = (0, 0)$, with radius 1 on the display so that it comes out looking circular. One way to do this is to use the `WINDOW` statement as shown below. The resulting picture is shown in Figure 4.1.

```
* FCT CIRX(T) = COS(2*PI*T)
* FCT CIRY(T) = SIN(2*PI*T)
* TL = 0:1!100
* M = (CIRX ON TL) &' (CIRY ON TL)
* DRAW M
* WINDOW ADJUST WMATCH
* VIEW
```

The functions `CIRX` and `CIRY` are the parametric equations for the x- and y-coordinates of the desired circle, as the parameter goes from 0 to 1. The `ADJUST`-clause in the `WINDOW` command: `ADJUST WMATCH`, specifies that the window dimensions should be adjusted so that the ratio of

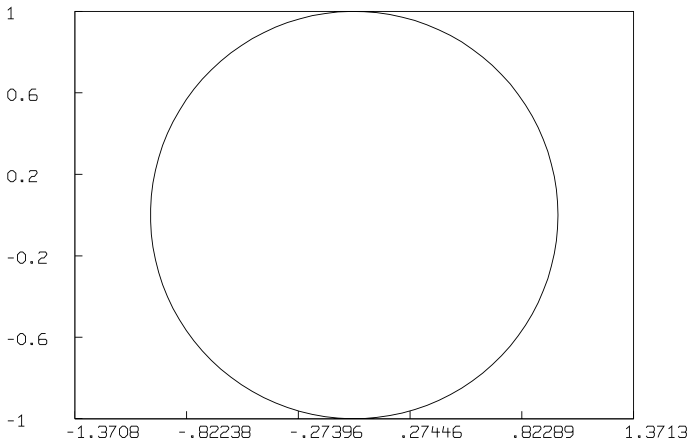


Figure 4.1: Graph with unit aspect ratio due to WINDOW ADJUST WMATCH.

user units to screen inches *matches*, i.e. is the same for both the x- and y-directions. This changes the default aspect ratio of the image.

An alternative approach to solving the same problem may be found by using the IMAGE command. Instead of using the WINDOW command above, we could have written, for example,

```
* IMAGE .5 TO 4.5, .5 TO 4.5 INCHES
```

prior to the VIEW command, and obtained the picture shown in Figure 4.2.

The IMAGE command here sets a square image region, four inches on a side with its left edge half an inch from the left edge of the MLAB graphics window and its bottom edge half an inch from the bottom of the MLAB graphics window. Since the WINDOW extends from -1 to 1 in both x and y, the resulting figure has unit aspect ratio.

Note, if the MLAB graphics window frame has been resized and is less than 4.5 inches in width or 4.5 inches in height, the width of the image can not be made larger than the frame and the preceding command will fail with an MLAB error.

The previous example demonstrated one way of modifying the x- and y-limits of the default window by using the ADJUST clause with the WMATCH option in a WINDOW statement. The ADJUST clause has seven predefined options which control how the window limits will vary when curves are added or deleted:

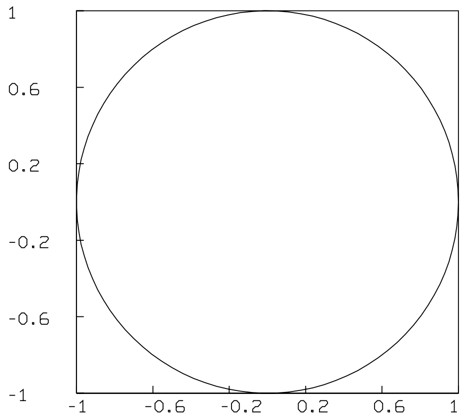


Figure 4.2: Graph with aspect ratio of 1 achieved by re-imaging the window.

`WFIX` (0) neither expand nor shrink the window limits when adding or deleting curves.

`WSHRINK` (1) decrease window limits, if necessary, when deleting curves.

`WEXPAND` (2) increase window limits, if necessary, when adding curves.

`WFLOAT` (3) shrink or expand window limits if necessary to accomodate added or deleted curves.

`WSLACK` (7) increase window limits by 5 percent of the window extent beyond that determined by minimum shrink extent.

`WNICE` (11) increase window limits by as much as 10 percent of the minimum shrink extent to obtain round numbers on axes.

`WMATCH` (15) use the larger of minimum x- or y-shrink extent to achieve unit aspect ratio.

Either numbers or names may appear in the `ADJUST` clause. By default, a window's `ADJUST` option is set to `WNICE`, which means that the upper and lower limits of the x- and y-axes of the window are automatically chosen to capture all of the curves in the window with rounded numbers appearing on the axes. We demonstrated this option in the previous chapter in Example 1-7.

The next example demonstrates how to rewindow a graph.

Example 4-2: Suppose we are drawing the probability density function for the normal distribution with mean zero, for several different values of the variance, `V`. The MLAB commands for creating this graph are:

```
* DELETE W
```

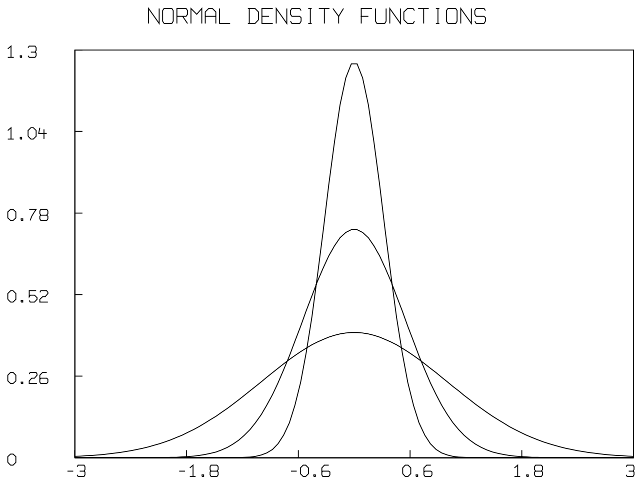


Figure 4.3: Normal density functions in a window with fixed x and y extents.

```
* WINDOW -3 TO 3, 0 TO 1.3
* FCT NORMD(X) = GAUSSD(X,0,V)
* VL = LIST(1.,.3,.1)
* FOR I = 1:3 DO {
>    V = VL[I];
>    DRAW POINTS(NORMD,-3:3!100) COLOR COLORN[I];
>   }
* TOP TITLE "NORMAL DENSITY FUNCTIONS"
* VIEW
```

The resulting picture is shown in Figure 4.3.

The `WINDOW` statement here fixes the x and y dimensions for the graph. The x-dimension of this graph is fixed from -3 to 3, and the y-dimension is fixed from 0 to 1.3. Note that if a curve with maximum magnitude greater than 1.3 is added to the window, parts of the curve will not be visible in this window.

Pictures drawn using the default window always have the same image rectangle, which leaves enough space in the frame for axis numbers and titles. Sometimes, however, it is convenient to have a different image, leaving room for a legend, for example.

Example 4-3: Re-image the picture in Example 4-2 in order to place a legend to the right of the image. We will write a descriptive title for each curve in the picture, and place it to the right of the image. The MLAB statements for this are shown below and the resulting picture is displayed in Figure 4.4.

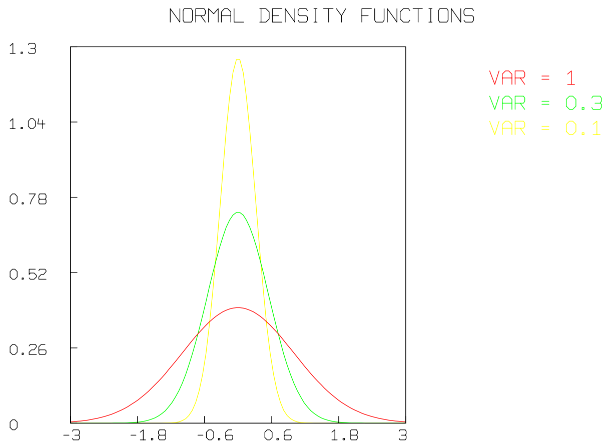


Figure 4.4: A graph with a legend.

```
* IMAGE .125 TO .625, .125 TO .875
* TITLE "VAR = "+VL[1] AT (.75,.8) COLOR COLORN(1)
* TITLE "VAR = "+VL[2] AT (.75,.75) COLOR COLORN(2)
* TITLE "VAR = "+VL[3] AT (.75,.7) COLOR COLORN(3)
* VIEW
```

The units of the TO-clauses in the IMAGE statement are FFRACT (frame fraction) units by default. These are the horizontal units of the coordinate system for which the lower left corner of the frame is at (0,0), and the upper right corner of the frame is at (1,1). Thus, this image occupies one half of the frame horizontally, displaced to left of center. It is somewhat longer than it is wide. The three TITLE statements draw titles in the region to the right of the image, with each title drawn in the color of the curve to which it applies. The AT-clause specifies the placement of the string, in FFRACT units.

Chapter 5

Drawing point labels

This chapter describes the LABEL clause of the DRAW statement which can be used to label points on a curve. The main uses of this clause are in the AXIS statement to build your own axes as discussed in the next chapter, however, point-labels can be used with ordinary curves as well.

Example 5-1: Draw the function $f(x) = 3 * \exp(x)/2$ over a range of 11 to 20, labeling each point with an integer-valued x-coordinate with its x-coordinate in “Gothic-English” numbers (font table 21). This is done as follows:

```
* FCT F(X) = 3*EXP(X)/2
* DRAW POINTS(F,11:20:.1) LABEL (11:20&'1:101:10) LABELFONT 21
* VIEW
```

The resulting picture is shown in Figure 5.1.

Here, the LABEL clause assigns the strings 11,12,13,. . . ,20 to the $1^{st}, 11^{th}, 21^{st}, \ldots, 101^{st}$ points of the curve. The strings are drawn using font number 21 due to the use of the LABELFONT clause.

Example 5-2: Draw a sine wave and a cosine wave, labeling the points of both curves which have integral x-values with the cosine wave point labels twice as big as the sine wave point labels.

The MLAB commands to do this are:

```
* DELETE W
* DRAW POINTS(SIN,0:6.28:.1) LABEL (0:6&'1:63:10) LABELSIZE .01 \
```

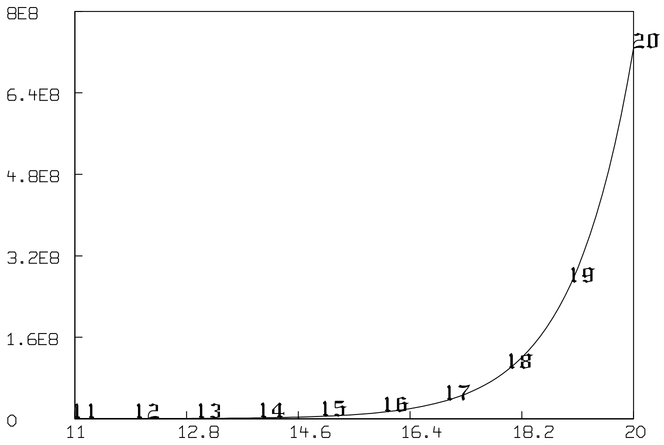


Figure 5.1: The function $f(x) = 3 * \exp(x)/2$ from $x = 11$ to 20 with point labels.

```
:    OFFSET(.01,.01)
* DRAW POINTS(COS,0:6.28:.1) LABEL (0:6&'1:63:10) LABELSIZE .02 \
:    OFFSET(.01,.01)
* VIEW
```

The resulting picture is shown in Figure 5.2.

The size of the point labels is determined by the `LABELSIZE` clause. The default size of a point label, if the `LABELSIZE` clause is omitted, is .02 frame fraction units. If the `LABELSIZE` clause is included, it takes one argument and an optional units specification. If no units are specified, the argument is in frame fraction units by default. Otherwise, the options for units are `FFRACT` (frame fraction), `IFRACT` (image fraction), `INCHES` (screen inches), `FRACT` (screen fraction), and `WORLD` (world vertical axis units).

The `OFFSET` clause determines the placement of the label relative to the point on the curve. It takes two arguments; the first is the x-displacement and the second is the y-displacement. Like the arguments to the `LABELSIZE` clause, the arguments are in `FFRACT` units by default, but a units qualifier may be included.

Example 5-3: Draw the function $t(x) = 5 * \log(x)$ over a range of 1 to 10, labeling 5 evenly-spaced points with their x-coordinates. Each label should have one digit on each side of the decimal point. Here are the MLAB commands to do this:

```
* DELETE W
```

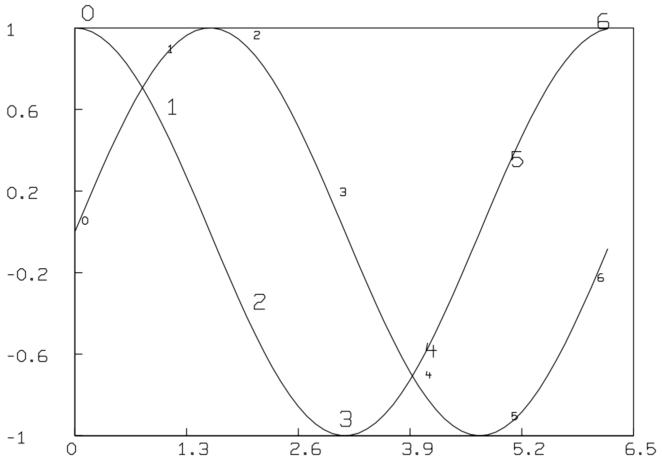


Figure 5.2: Sine and cosine waves labeled at integer x-values with different sized labels.

```
* FCT T(X) = 5*LOG(X)
* DRAW POINTS(T,1:10:.1) LABEL (1:10:2.&'1:91:20) PLACE (RIGHT,BOTTOM) \
:     LABELFONT 16 LABELSIZE .05 FORMAT(1,1,1,3,0,0)
* VIEW
```

The resulting picture is shown in Figure 5.3.

Here, the $1^{st}, 21^{st}, 41^{st}, 61^{st}$, and 81^{st} points are labeled with the strings `1.0`, `3.0`, `5.0`, `7.0`, and `9.0`, respectively, written with font 16 and in size .05 frame fract units. The `PLACE` clause is used to define the position of the rectangle enclosing a numeric label string relative to the offset from the point being labeled. `PLACE` takes two arguments; the options for the first argument are `LEFT` (2), `CENTER` (0), or `RIGHT` (3) and the options for the second argument are `TOP` (1), `CENTER` (0), and `BOTTOM` (4). In this example, the clause `PLACE (RIGHT,BOTTOM)` means the right bottom point of the rectangle enclosing the label string is to be placed on the label point, since there is no offset.

The `FORMAT` clause specifies the format of the numeric labels. The `FORMAT` clause takes six scalar arguments: `H,K,A,B,R,` and `E` where:

`H` is the number of digits to the left of the decimal point or precision code if `H` is less than 0.

`K` is the number of digits to the right of the decimal point or the total number of digits when `H` is less than 0.

`A` is the format code for the integer part.

`B` is the format code for the fraction part.

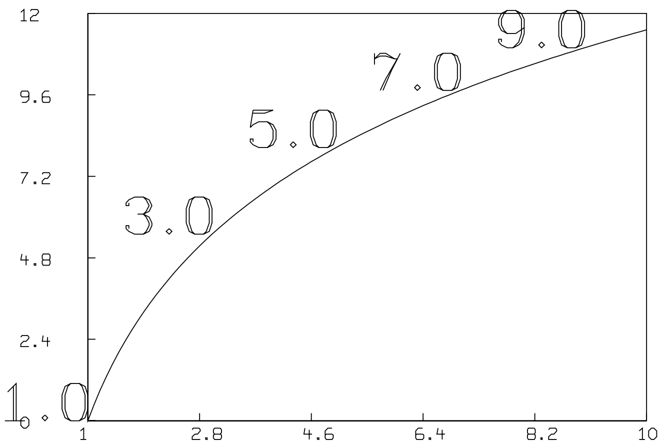


Figure 5.3: The function $t(x) = 5 * \log(x)$ from $x = 1$ to 10 with point labels.

`R` is the required number of significant digits for numbers with negative exponents.

`E` is the mandatory exponent code (0 for no exponent).

For a complete explanation of precision, format, and exponent codes, see the description of the `NFORMAT` clause of the `TYPE` statement in the **MLAB Reference Manual**.

In this example, the first and second arguments to the `FORMAT` clause are 1 and specify that 1 digit is to be shown to the right and to the left of the decimal point. The third through sixth arguments are codes for specific formats for the integer and decimal parts of the number.

Chapter 6

Using the axis statement

In chapters 1 through 5, all drawings were made in the default window W which provides automatic axes and axis labeling. In this chapter drawings are made in windows defined by the user. Generally, when drawing in a user-defined window, the XAXIS and YAXIS statements must be used to draw and label axes, and the clause IN <window name> must be added to each MLAB DRAW, TITLE, WINDOW, IMAGE and AXIS statement. The next example demonstrates how to use the XAXIS and YAXIS statements.

Example 6-1: Draw a pair of coordinate axes which intersect at the origin. Each axis should be numbered from −5 to 5. This is achieved as follows using MLAB:

```
* WINDOW -5 TO 5, -5 TO 5 IN WXY
* NO IMAGEBOX IN WXY
* WINDOW ADJUST WMATCH IN WXY
* AY = 0 &’ (-5:-1&1:5)
* YAXIS YA = AY IN WXY PT CROSSPT LABEL (-5:-1&1:5) OFFSET (.02,0) FFRACT \
:    PLACE(LEFT,CENTER)
* AX = (-5:-1&1:5) &’ 0
* XAXIS XA = AX IN WXY PT CROSSPT LABEL (-5:-1&1:5) OFFSET (0,-.02) FFRACT \
:    PLACE(CENTER,TOP)
* VIEW WXY
```

The resulting picture is shown in Figure 6.1.

Like the DRAW statement, XAXIS and YAXIS statements take POINTTYPE, LINETYPE, LABEL, OFFSET, and PLACE clauses.

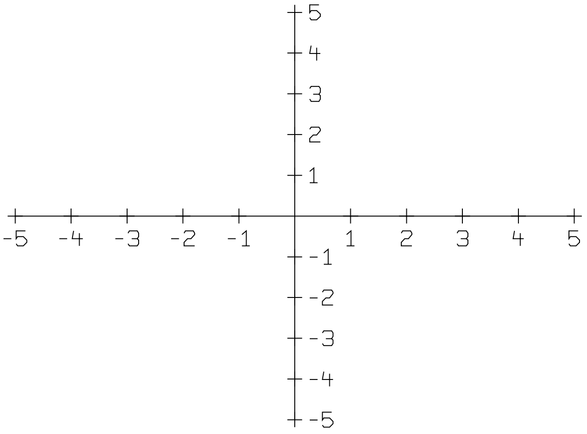


Figure 6.1: Cartesian coordinate axes.

Example 6-2: Draw a graph of the function $y(x) = x^3 - 3 * x^2 - 9 * x + 22$ for x = -2 to 2 with x- and y-axes, axis labels, and title in a newly-created window WP. This is accomplished by the following MLAB commands:

```
* DEL WXY
* WINDOW -2 TO 2, 0 TO 27 IN WP
* NO IMAGEBOX IN WP
* IMAGE 1 TO 5, 1 TO 5 INCHES IN WP
* FCT Y(X) = X^3-3*X^2-9*X+22
* DRAW C1 = POINTS(Y,-2:2:.1) IN WP
* XAXIS -2:2&’0 PT DTICK LABEL -2:2 LABELSIZE .015 FFRACT \
:     OFFSET(-.01,-.05) IN WP
* YAXIS -2 &’ 0:27:9 PT LTICK LABEL 0:27:9 LABELSIZE .015 FFRACT \
:     OFFSET(-.09,-.01) IN WP
* TITLE "ABSCISSA" AT (.3,.15) FFRACT IN WP
* TITLE "ORDINATE" AT (.06,.45) FFRACT ANGLE 90 IN WP
* TITLE "x^3-3*x^2-9*x+22 vs. x" AT (.2,.9) FFRACT IN WP
* VIEW WP
```

The result is shown in Figure 6.2.

The clause IN WP appears in every statement, thereby specifying the user-window WP as the target window rather than the default window W.

The first three statements define the overall characteristics of the window: the Cartesian coordinates in the window have x ranging from -2 to 2 and y ranging from 0 to 27. The default

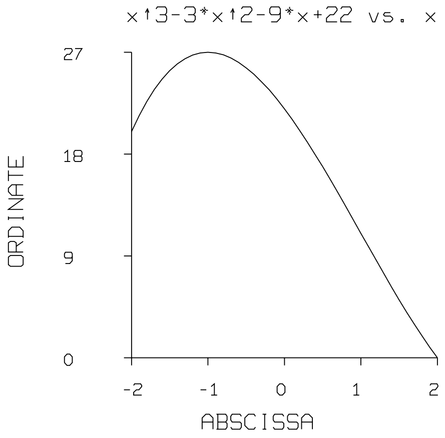


Figure 6.2: A simple polynomial in a user-defined window.

drawing of a box around the image rectangle is suppressed by the `NO IMAGEBOX` statement. The `IMAGE` statement sets the image rectangle to be 4 inches square and places it with its lower left corner positioned 2 inches above and 2 inches to the right of the lower left corner of the display.

The next two statements define the curve which will appear in the window. The curve is given the name `C1`.

The `XAXIS` and `YAXIS` statements determine how the coordinate axes are to be drawn and labeled. The first argument to both statements is a 2 column matrix containing the world coordinates of points where the axis tic marks appear. The x-tic marks appear at (-2,0), (-1,0), (0,0), (1,0), and (2,0) and the y-tic marks appear at (-2,0), (-2,9), (-2,18), and (-2,27). No linetype is specified so the tic marks are connected by a solid line by default. The pointtype specifies the symbol for each tic; `DTICK` is for a downward-directed tic mark and `LTICK` is for a left-directed tic mark. Each tic mark is also labeled with a number by the `LABEL`-clause. In this example, `LABEL` takes a one column matrix of numbers as its argument, and these numbers are the labels of successive points on the axis. The `LABELSIZE`-clause determines the size of the labels drawn, in this case expressed in `FFRACT` (frame fraction) units. The `OFFSET`-clause specifies where the label appears with respect to the axis point. By default, the offset is in `FFRACT` (frame fraction) units.

The title statements produce three titles with positions determined by the `AT`-clause. Here the coordinates of the position are expressed in `FFRACT` (frame fraction) units. In the ordinate title, the `ANGLE`-clause causes a 90 degree counterclockwise rotation of the title.

Example 6-3: Draw each of the inverse hyperbolic trigonometric functions in a different color in the specially-created window `IHT`. Here are the MLAB commands which do this:

```
* DEL WP
* WINDOW -2.1 TO 2.6, -2.6 TO 2.6 IN IHT
* NO IMAGEBOX IN IHT
* IMAGE .5 TO 5., .5 TO 5. INCHES IN IHT
* XAXIS -2:2.5:.5&'2.6 LABEL -2:2.5:.5 LABELSIZE .01 FFRACT \
:       OFFSET(-.02,0.02) LT 0 PT DDBAND IN IHT
* YAXIS 2.6&'-2.5:2.5:.5 LABEL -2.5:2.5:.5 LABELSIZE .01 FFRACT \
:       OFFSET(.02,-.01) LT 0 PT DLBAND IN IHT
* DRAW C1 = POINTS(ASINH,-2.0:2.5!200) COLOR RED IN IHT
* TITLE "asinh(x)" AT (.05,.30) COLOR RED SIZE .01 FFRACT IN IHT
* DRAW C2 = POINTS(ACOSH,1.:2.5!100) COLOR GREY IN IHT
* TITLE "acosh(x)" AT (.45,.50) COLOR GREY SIZE .01 FFRACT IN IHT
* DRAW C3 = POINTS(ATANH,-.999:.999!100) COLOR YELLOW IN IHT
* TITLE "atanh(x)" AT (.25,.35) COLOR YELLOW SIZE .01 FFRACT IN IHT
* DRAW C4 = POINTS(ASECH,.01:.999!100) COLOR PINK IN IHT
* TITLE "asech(x)" AT (.27,.60) COLOR PINK SIZE .01 FFRACT IN IHT
* DRAW C5 = POINTS(ACOTANH,-2.0:-1.01!50) COLOR BLUE IN IHT
* DRAW C6 = POINTS(ACOTANH,1.01:2.5!50) COLOR BLUE IN IHT
* TITLE "acotanh(x)" AT (.45,.8) COLOR BLUE SIZE .01 FFRACT IN IHT
* DRAW C7 = POINTS(ACOSECH,-2.0:-.01!100) COLOR GREEN IN IHT
* DRAW C8 = POINTS(ACOSECH,.01:2.5!100) COLOR GREEN IN IHT
* TITLE "acosech(x)" AT (.3,.2) COLOR GREEN SIZE .01 FFRACT IN IHT
* TITLE "INVERSE HYPERBOLIC FUNCTIONS" AT (.1,.9) FONT 8 IN IHT
* VIEW IHT
```

The ranges of the x-coordinates at which each function is evaluated are chosen to avoid singularities and branch points. The result is shown in Figure 6.3.

We have seen most of the statements in this example before. The first three statements serve to define the overall characteristics of the window. The Cartesian coordinates of the window range from -2.1 to 2.6 in x and -2.6 to 2.6 in y. The drawing of borders around the image rectangle is suppressed by the `NO IMAGEBOX` statement. The image is located with its lower left corner half an inch above and half an inch to the right of the lower left corner of the display. The image box measures five and a half inches on a side.

The `XAXIS` and `YAXIS` statements include an `OFFSET` clause which determines where a label will be placed with respect to the points specified in the `LABEL` matrix. The `OFFSET` is expressed in `FRACT` units by default. The `XAXIS` and `YAXIS` statements also use the pointtype symbols: `DDBAND` and `DLBAND`. `DDBAND` produces a dotted line in a downward direction from the x-tic mark position and `DLBAND` produces a dotted line to the left of the y-tic mark position. The `TITLE` statement which defines the main title of the graph shows the use of the `FONT` clause. The characters of a title may be drawn in any one of the 34 MLAB Hershey fonts or one of the 13 MLAB TrueType fonts appearing in Appendix A.

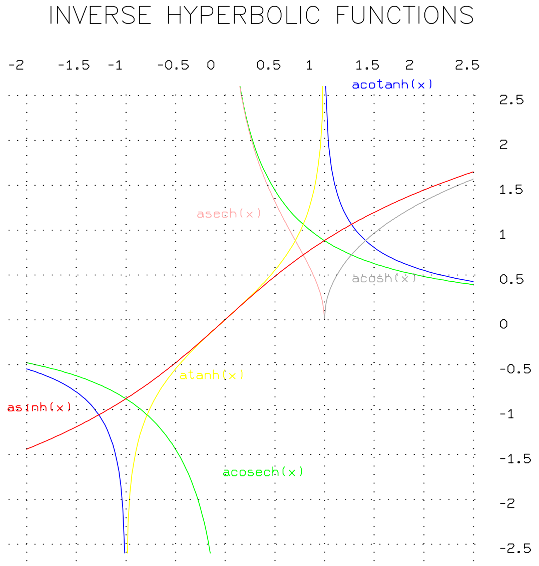


Figure 6.3: The inverse hyperbolic trigonometric functions in a user-defined window.

Chapter 7

Making special graphs

Not all graphs made using MLAB need be linear-linear Cartesian coordinate graphs. MLAB can be used to produce graphs in polar coordinates or graphs with one or two logarithmic scales, as the examples in this chapter show.

Example 7-1: Draw the function $f(t) = 4 * \cos(3 * t)$ in polar coordinates. The MLAB commands to do this are:

```
* A = 4
* WINDOW -A TO A, -A TO A IN POLAR
* NO IMAGEBOX IN POLAR
* IMAGE .5 TO 5., .5 TO 5. INCHES IN POLAR
* U = 0:150:30
* FCT FX(T) = A*COSD(T)
* FCT FY(T) = A*SIND(T)
* DRAW MESH((FX ON U)&'(FY ON U),(FX ON U+180)&'(FY ON U+180)) LT 5 \
:    IN POLAR
* U = 0:330:30
* DRAW (FX ON U)&'(FY ON U) LT 0 LABEL U LABLESIZE .01 FFRACT IN POLAR
* U = 0:360:5
* FOR A = 1:3 DO {DRAW (FX ON U)&'(FY ON U) IN POLAR;}
* FCT R(T) = 4*COS(3*T)
* DRAW TPOLAR(R,0:PI:.05) LT 2 IN POLAR
* TITLE "4*cos(3*t) vs. t" AT (.2,.9) FFRACT IN POLAR
* VIEW POLAR
```

The resulting graph is shown in Figure 7.1.

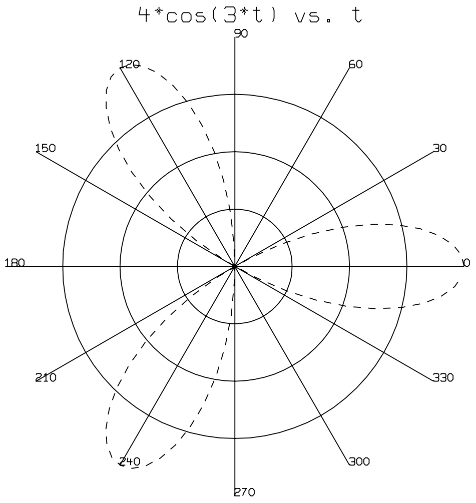


Figure 7.1: Example of a polar coordinate graph in a user-defined window.

The first three statements serve to define the characteristics of the window. The Cartesian coordinates of the window range from -4 to 4 in both x and y. The borders of the image box are not drawn. The image box measures 6 inches on a side and is located .2 inches above and .2 inches to the right of the lower left corner of the screen.

The next eight statements cause the polar coordinate axes to be drawn and labeled. The `MESH` operator is used to create the matrix of coordinates that determine the endpoints of axes at angles of $0, 30, 60, 90, \ldots, 330$ degrees. It returns a 2 column matrix in which the rows of the input matrices are shuffled together. In other words, the first row of the output matrix is the first row of the first argument matrix; the second row of the output matrix is the first row of the second argument matrix; the third row of the output matrix is the second row of the first argument matrix; the fourth row of the output matrix is the second row of the second argument matrix; etc. The second `DRAW` statement labels the axes at each angle. The `FOR`-loop serves to produce 3 concentric circles for the radial axes.

The `TPOLAR` operator constructs a two column matrix of (x, y) coordinates suitable for drawing. The first argument is the name of a function which yields radius as a function of angle. The second argument is a vector of polar coordinate angle values in radians.

MLAB also has two functions called `RPOLAR` and `PPOLAR` which simplify drawing angular functions of radius and parametric polar coordinate functions, respectively. See the **MLAB Reference Manual** for more details.

Example 7-2: There are more than twenty Global Positioning System (GPS) satellites in orbit about the earth at a radius of roughly 22000 kilometers. The orbits lie in planes which are inclined to the earth's equator at an angle of 55 degrees; a satellite completes an orbit in 12 hours. With some GPS receivers, the positions of satellites above the horizon at a given time are given in terms of an azimuth and angle of elevation. Draw a polar graph of such data over a 14 hour period, showing the observation start, observation stop, rise, and/or set times for each satellite track. The data file `gps.dat` appears in Appendix D.

```
* /* read and shape (time,elevation,azimuth) data from file = gps.dat */
* M = READ("gps.dat",1000,1)
* N = NROWS(M)/3
* M = SHAPE(N,3,M)
*
* /* define functions for converting (elevation,azimuth) to (x,y) */
* FCT X(E,A) = COSD(E)*SIND(A)
* FCT Y(E,A) = COSD(E)*COSD(A)
*
* /* transform data to Cartesian coordinates */
* P = (X ON M COL (2,3)) &' (Y ON M COL (2,3))
*
* /* set marker rows */
* FOR I = 1:N DO {IF P[I,1] = 0 AND P[I,2] = 1 THEN {P[I,1] = MAXPOS}}
*
* /* draw unit circle */
* DRAW (COS ON 0:(2*PI)!200) &' (SIN ON 0:(2*PI)!200) IN W
* WINDOW ADJUST WMATCH
* NO IMAGEBOX
*
* /* label azimuth circle */
* TITLE "North" AT (0,1) WORLD PLACE (CENTER,BOTTOM)
* TITLE "East" AT (1,0) WORLD PLACE (LEFT,CENTER)
* TITLE "South" AT (0,-1) WORLD PLACE (CENTER,TOP)
* TITLE "West" AT (-1,0) WORLD PLACE (RIGHT,CENTER)
*
* /* draw satellite paths */
* DRAW P LT MARKER
*
* /* define observation time string array */
* FCT MN(T) = INT(100*T-100*INT(T))
* T = (INT ON M COL 1) &' (MN ON M COL 1)
*
* /* label endpoints of satellite paths with observation times as hour:minute */
* S = T[2,1]+(IF T[2,2] < 10 THEN ":0" ELSE ":")+T[2,2]
* TITLE S AT (P[2,1],P[2,2]) WORLD SIZE .01 PLACE (CENTER,CENTER)
* FOR I = 3:N DO {\
: IF P[I,1] = MAXPOS THEN {
> S = T[I+1,1]+(IF T[I+1,2] < 10 THEN ":0" ELSE ":")+T[I+1,2]
> TITLE S AT (P[I+1,1],P[I+1,2]) WORLD SIZE .01 PLACE (CENTER,CENTER)
> S = T[I-1,1]+(IF T[I-1,2] < 10 THEN ":0" ELSE ":")+T[I-1,2]
```

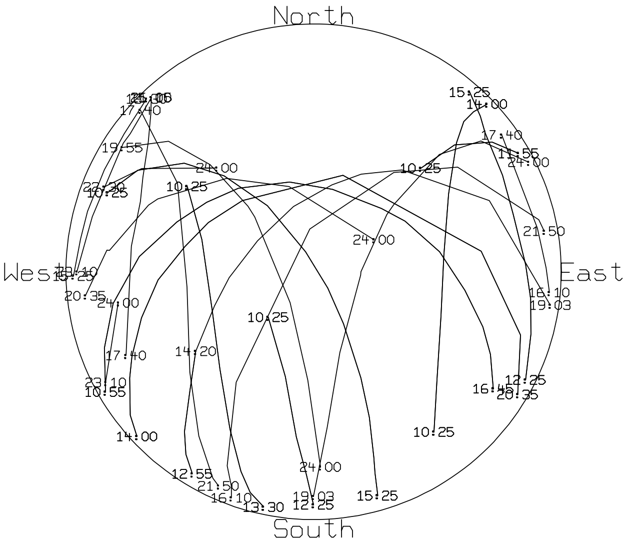


Figure 7.2: GPS satellite tracks in the sky over 14 hours.

```
> TITLE S AT (P[I-1,1],P[I-1,2]) WORLD SIZE .01 PLACE (CENTER,CENTER)
> }
> }
* S = T[N,1]+(IF T[N,2] < 10 THEN ":0" ELSE ":")+T[N,2]
* TITLE S AT (P[N,1],P[N,2]) WORLD SIZE .01 PLACE (CENTER,CENTER)
* VIEW
```

The resulting picture is shown in Figure 7.2.

MLAB can also make linear-log plots, as the next example shows.

Example 7-3: Draw linear-linear and linear-log plots of simulated radioactivity counts versus time data with an exponential curve fitted to the data.

```
* M = EXPRAN ON (0&’10)^^500
* D = HISTO(M,25)
* XAXIS 0:10!6&’0 LABEL 0:10!6 LABELSIZE .015 FFRACT OFFSET(-.01,-.025) \
:    FFRACT FORMAT(-3,5,0,0,2,0) PT UTICK IN DECAY
* YAXIS 0&’0:10!6 LABEL 0:10!6 LABELSIZE .015 FFRACT OFFSET (-.09,-.01) \
:    FFRACT FORMAT(-3,5,0,0,2,0) PT RTICK IN DECAY
* DRAW BARGRAPH(D) IN DECAY
* WINDOW 0 TO MAXV(D COL 1), 0 TO MAXV(D COL 2) IN DECAY
```

The data consists of 500 exponential random numbers with mean 10 sorted into 25 equal sized bins. The data is generated by applying the exponential random number generator EXPRAN to a 500 row, 2 column matrix with zeros in column 1 and tens in column 2. Then the function HISTO is used to make a 2 column matrix histogram with center-of-bin values in column 1 and number of counts in column 2. The XAXIS and YAXIS statements serve to make 6 equally spaced tic marks with labels on the x and y axes. The function BARGRAPH generates the x and y coordinates necessary to draw a bar graph of the histogram matrix D. The bargraph is then set in a window named DECAY with x ranging from 0 to MAXV(D COL 1)–the maximum value appearing in the first column of matrix D, and with y ranging from 0 to MAXV(D COL 2)–the maximum value appearing in the second column of matrix D.

An exponential function is fit to the data as follows:

```
* FCT F(X) = A*EXP(-X/B)
* A = 1; B = 1;
* FIT (A,B), F TO D
```

The FIT operation causes the following output from MLAB:

```
final parameter values
      value               error                dependency    parameter
    145.6208046         3.029833482          0.3380746002   A
    10.94275774         0.377410294          0.3380746002   B
9 iterations
CONVERGED
best weighted sum of squares = 2.841914e+02
weighted root mean square error = 3.515132e+00
weighted deviation fraction = 3.980574e-02
R squared = 9.914948e-01
```

The FCT command defines the function to be fit. The function includes two parameters to be fit, A and B, which are both guessed to be 1. The FIT command then initiates a search for A and B values which minimize a sum of squares criterion using the function F and the data in

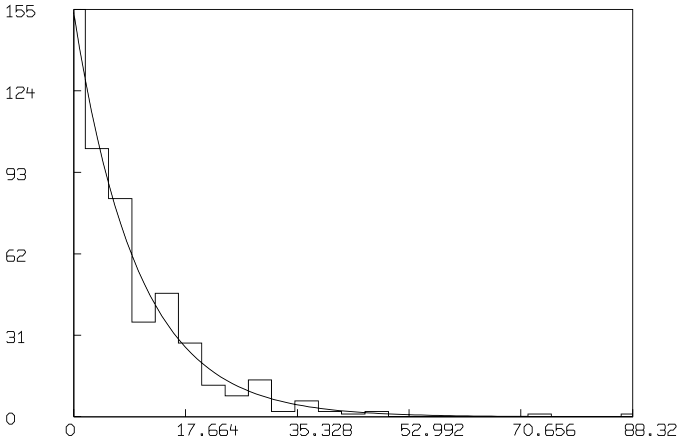


Figure 7.3: Linear-linear graph of decay time versus counts.

matrix D. The MLAB output indicates that the search converged in 14 iterations. Note that the original value of 10 used for the mean of the exponential random numbers is within the error of the value found for B.

We continue by plotting the fitted function with the following commands:

```
* DRAW POINTS(F,0:MAXV(M)!100) IN DECAY
* VIEW DECAY
```

The results in Figure 7.3.

A linear-log curve of the above data and the fitted function is plotted in the user-defined window DECAY2 with the following MLAB commands:

```
* DELETE DECAY,Y
* DD = COMPRESS(D,2,0)
* DELETE D
* XAXIS 0:10!6&'0 LABEL 0:10!6 LABELSIZE .015 FFRACT OFFSET(-.01,-.025) \
:    FFRACT PT UTICK IN DECAY2
* WINDOW 0 TO 100, -2 TO 3 IN DECAY2
* Y = LIST(.01,.1,1,10,100,1000)
* YY = -2:3
* YAXIS 0&'YY LABEL Y LABELSIZE .015 FFRACT OFFSET(-.09,-.01) \
:    FFRACT IN DECAY2
* YYY = LOG10 ON (.01:.09:.01 & .1:.9:.1 & 1:9 & 10:90:10 & \
```

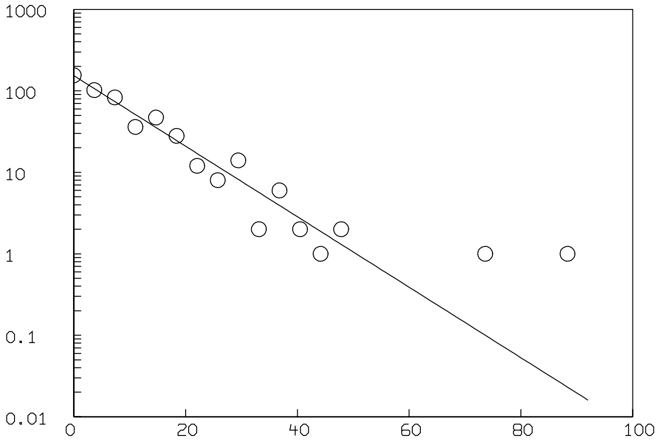


Figure 7.4: Linear-log graph of decay time versus counts.

```
 :    100:900:100)
 * AXIS 0&'YYY PT RTICK IN DECAY2
 * DRAW LINLOG(DD) LT NONE PT CIRCLE IN DECAY2
 * DRAW LINLOG(F,0:MAXV(M)!100) IN DECAY2
 * VIEW DECAY2
```

The result of these commands is shown in Figure 7.4.

The COMPRESS operator is used to censor, i.e. delete, those rows of the histogram array D which have zeros in the second column. These values will otherwise cause problems when the LOG10 operator is applied to the second column of D. The XAXIS and YAXIS statements mark the x- and y-axis with six equally spaced tic marks and label them appropriately. The AXIS statement adds decade tic marks to the y-axis.

The LINLOG statement constructs a 2 column matrix suitable for semilog drawing, where the x coordinate is linear and the y coordinate is base-10 logarithmic. The two calls to LINLOG demonstrate the two input cases. In the first call, the input is a 2 column matrix. The result is a 2-column matrix containing the points in the input matrix transformed such that column 1 of the resultant matrix is the same as column 1 of the input matrix and column two of the resultant matrix is LOG10 on column 2 of the input matrix.

The second LINLOG statement demonstrates the second input case. Here the input is a function and a 1-column matrix. The result is a 2-column matrix with the 1-column input matrix in column 1 of the output matrix and LOG10 on the function applied to the input 1-column matrix in the second column of the output matrix.

The cumulative distribution function of the simulated decay data in matrix M can also be viewed with the corresponding mean-10 exponential distribution function overlaid. This is done in the default window W as follows:

```
* DELETE DECAY2,DD
* N = CDF(M)
* DELETE M
* DRAW STEPGRAPH(N)
* FCT G(X) = 1.-C*EXP(-X/D)
* C = 1; D = 1;
* FIT (C,D), G TO N
```

The function CDF constructs a matrix which contains values corresponding to the cumulative distribution function of the multi-dimensional data. At this point MLAB performs the curve fit, responding with:

```
final parameter values
      value               error                dependency    parameter
   0.9900066378      0.001256493198           0.4723679752   C
    10.76204882       0.02895993028           0.4723679752   D
5 iterations
CONVERGED
best weighted sum of squares = 7.032791e-02
weighted root mean square error = 1.188363e-02
weighted deviation fraction = 1.605038e-02
R squared = 9.983121e-01
```

We then draw the fitted exponential curve, shown in Figure 7.5, as follows:

```
* DRAW POINTS(G,0:100!100) LT DASHED
* VIEW W
```

Finally, we can use the WINDOW statement to zoom in on the region of the graph where the fitted function and the cumulative distribution function do not match as closely as elsewhere. This is done as follows:

```
* WINDOW 15 TO 40, .7 TO 1.
* VIEW W
```

The resulting picture is shown in Figure 7.6.

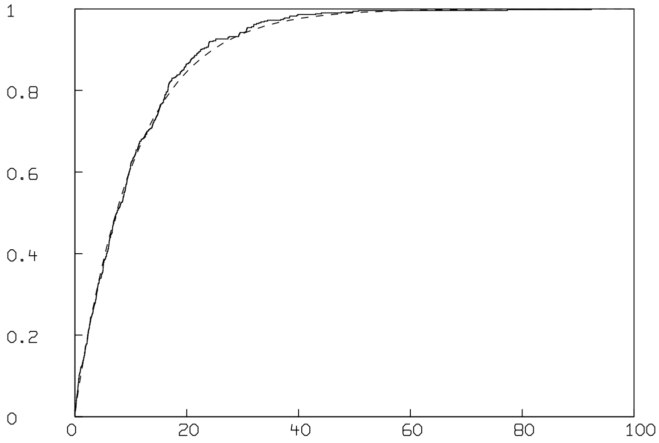


Figure 7.5: CDF of data (solid) and best fit exponential function (dashed).

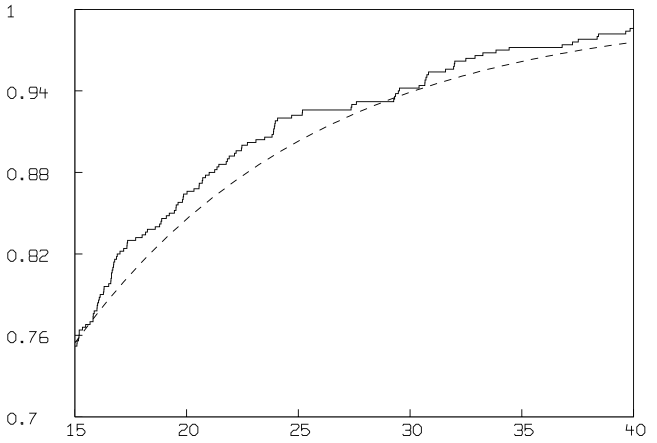


Figure 7.6: Magnified CDF of data (solid) and best fit exponential function (dashed).

MLAB also has operators LOGLIN and LOGLOG which perform the transformations necessary to create log-linear and log-log plots, respectively. See the **MLAB Reference Manual** for further details.

Example 7-4: Draw a log-log plot of 50 random data points. Show 6 decades in both x and y coordinate axes.

```
* RESET
* /* store the independent variable values in tl */
* LOWPOW10 = -9
* NCYCLES = 6
* TL = 0'
* FOR I=LOWPOW10:(LOWPOW10+NCYCLES-1) DO {T=(10^I):(9*10^I)!9; TL=TL&T;}
* TL = TL&(10^(I+1))
* DEL TL ROW 1
*
* /* generate log-log plot of random data with x,y both ranging */
* /* from 1E(lowpow10+1) to 1E(lowpow10+ncycles) */
* WINDOW LOWPOW10 TO (LOWPOW10+NCYCLES),LOWPOW10 TO (LOWPOW10+NCYCLES) IN LL
* IMAGE .5 TO 4.5, .5 TO 4.5 INCHES IN LL
* IMAGEBOX IN LL
*
* /* x1 and y1 are the tic marks */
* XAXIS X1 = (LOG10 ON TL)&'LOWPOW10 PT DTICK IN LL
* YAXIS Y1 = LOWPOW10&'(LOG10 ON TL) PT LTICK IN LL
*
* /* x2 and y2 are the exponents in the decade labels */
* STEPS = LOWPOW10:(LOWPOW10+NCYCLES)
* XAXIS X2 = STEPS &' LOWPOW10 LABEL STEPS LABELSIZE .01 OFFSET \
: (-.01,-.025) FFRACT IN LL
* YAXIS Y2 = LOWPOW10 &' STEPS LABEL STEPS LABELSIZE .01 OFFSET \
: (-.04,.01) FFRACT IN LL
*
* /* x3 and y3 are the 10's in the decade labels */
* TENS = 10^^(NCYCLES+1)
* XAXIS X3 = STEPS &' LOWPOW10 LABEL TENS LABELSIZE .017 OFFSET \
: (-.03,-.05) FFRACT IN LL
* YAXIS Y3 = LOWPOW10 &' STEPS LABEL TENS LABELSIZE .017 OFFSET \
: (-.06,-.01) FFRACT IN LL
*
* /* draw 50 random data points as small green circles */
* FUNCTION F(X) = RAN(0)*10^(IRAN(0,LOWPOW10,LOWPOW10+NCYCLES))
* M = (F ON 1:50) &' (F ON 1:50)
* DRAW LOGLOG(M) LT NONE PT CIRCLE COLOR GREEN PTSIZE .01 IN LL
*
* VIEW
```

The resulting picture is shown in Figure 7.7.

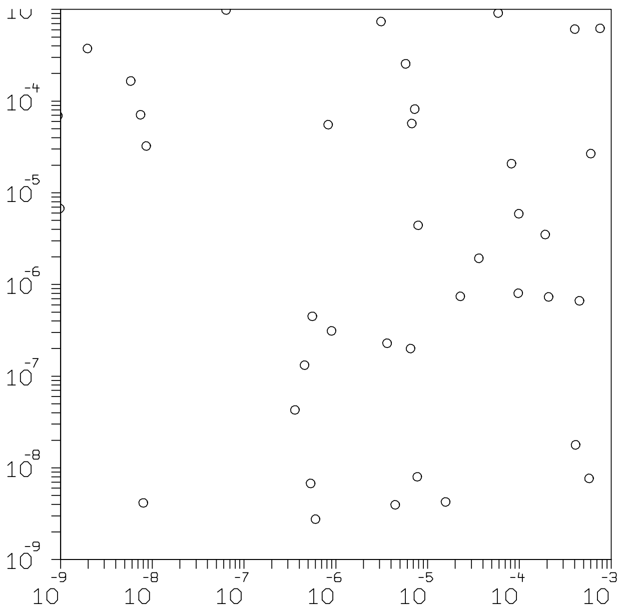


Figure 7.7: Log-log plot of 50 random points.

Chapter 8

Displaying several graphs simultaneously

In this chapter we will learn to use the default window to build a number of different graphs, some or all of which may be shown on the display, either sequentially or simultaneously.

Example 8-1: Build two graphs and show them sequentially. The first graph will be of bivariate uncorrelated normal random data. The second graph will be the same bivariate normal data modified to be correlated with a specified correlation matrix. Here are the MLAB statements to build these graphs:

```
* R = SHAPE(100,2,NORMRAN ON 0^^200)
* C = SHAPE(2,2,LIST(1,.5,.5,2))
* FCT Q1(I) = C[1,1]*R[I,1]+C[1,2]*R[I,2]
* FCT Q2(I) = C[2,1]*R[I,1]+C[2,2]*R[I,2]
* D COL 1 = Q1 ON 1:100
* D COL 2 = Q2 ON 1:100
* DRAW D LINE NONE PT "a" PTSIZE .01
* TOP TITLE "CORRELATED BIVARIATE NORMAL RANDOM DATA"
* WINDOW ADJUST WMATCH
* W1 = W
* TOP TITLE "UNCORRELATED BIVARIATE NORMAL RANDOM DATA"
* DRAW R LINE NONE PT "c" PTSIZE .01
* WINDOW ADJUST WMATCH
* VIEW W1
* VIEW W
```

The resulting pictures are shown in Figure 8.1 and Figure 8.2.

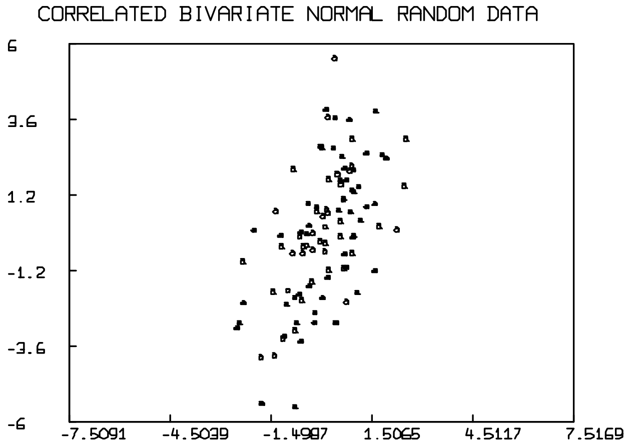


Figure 8.1: First of two sequential graphs.

The important point above is the window renaming assignment statement: `W1 = W`. This statement changes the name of `W` to be `W1`. Now `W` does not exist and it can be recreated.

Here we have specified the name of the window to be viewed in the `VIEW` statement. If no window names are specified in the `VIEW` statement, all windows are displayed, in unpredictable order. Figure 8.1 shows the correlated data, which is in window `W1`. Figure 8.2 shows the uncorrelated data, which is in window `W`. Note that the uncorrelated data is spread over a circular region where the correlated data is spread over an elliptical region.

The `SHAPE` function builds a matrix. Its first argument is the number of rows in the resultant matrix; its second argument is the number of columns in the resultant matrix. The matrix `R` is constructed to be a list of 100 bivariate normal random vectors, with the x-component in column 1 and the y-component in column 2. The matrix `C` is the specified covariance matrix. The functions `Q1` and `Q2` are used to build the x- and y-components of the correlated normal random vectors.

It would be more convenient to view both of these graphs simultaneously, since it is the contrast between them which is of interest. In the next example we show how to display two graphs side by side.

Example 8-2: Build two graphs and display them side by side. Here we take an example from mechanics, showing the displacement versus time trajectory of a damped 1-dimensional oscillator in one graph and the velocity versus displacement trajectory in the other graph. The equations of motion are integrated and the results graphed by the following MLAB commands:

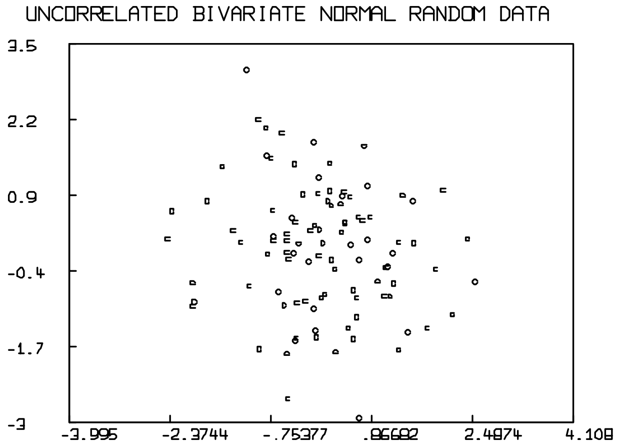


Figure 8.2: Second of two sequential graphs.

```
* DEL W,C,W1,D
* FCT X DIFF T(T) = V
* FCT V DIFF T(T) = -(A*V+K*X)/M
* M = 1
* A = .3
* K = 1
* INITIAL X(0) = 1
* INITIAL V(0) = 0
* N = INTEGRATE(X,V,0:50:.1)
* DRAW C = N COL(1,2)
* DRAW (0&’0) LT NONE PT CIRCLE
* TOP TITLE "DISPLACEMENT VS. TIME"
* FRAME .01 TO .5,.25 TO .75
* FRAMEBOX
* W1 = W
* DRAW D = N COL(2,3)
* DRAW (0&’0) LT NONE PT CIRCLE
* TOP TITLE "VELOCITY VS. DISPLACEMENT"
* FRAME .5 TO .99, .25 TO .75
* FRAMEBOX
* VIEW
```

The result is shown in Figure 8.3.

The first eight commands define the equations of motion, constants, and initial conditions for a one-dimensional damped oscillator. Working in cgs units, the mass M, the spring constant K,

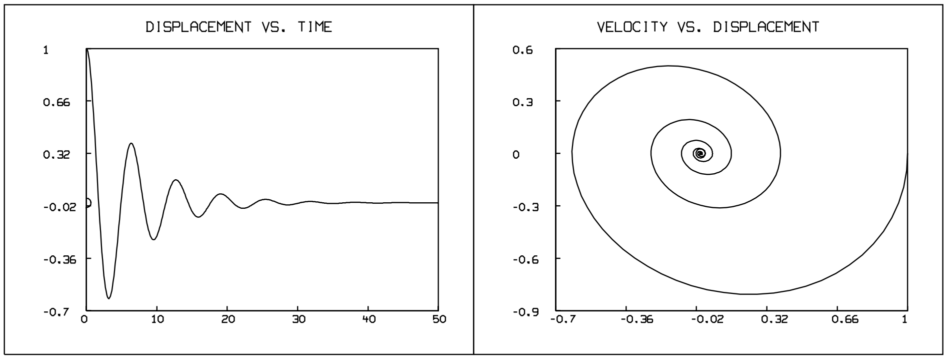


Figure 8.3: Trajectories of damped oscillator side by side.

drag coefficient A, initial displacement, and initial velocity are arbitrarily set equal to 1 gram, 1 dyne/cm, .3 dyne*sec/cm, 1 cm, and 0 cm/sec, respectively. The velocity and displacement are initialized with INITIAL statements because they are initial values of functions.

The INTEGRATE command takes the names of the functions and derivatives which are to be integrated and a vector of values of the dependent variable as arguments and returns a matrix containing the dependent variable values in column one, the integrated function values in subsequent even numbered columns, and the derivative of the integrated function values in subsequent odd numbered columns. In this case, the dependent variable is time and it takes the values of 0 to 50 in steps of .1. The INTEGRATE command returns a 5 column by 501 row matrix N which contains the time values in column 1, displacement as a function of time in column 2, velocities in columns 3 and 4, and accelerations in column 5.

The first FRAME command reframes the original picture, shrinking it approximately by half to occupy the left half of the display screen, and centering it vertically. The units of the TO-clauses in the FRAME command are FRACT (screen fraction) by default. The FRACT coordinate system has the lower left corner of the display screen at (0,0), and the upper right corner at (1,1). Then this picture, which has the name W by default–since we did not give it a name–is renamed W1. Now the original window W *no longer exists*. Next, we draw the displacement versus velocity graph, in the default window, which is thereby newly recreated. In the second FRAME command, we have reframed the new picture, shrinking it again approximately by half to occupy the right half of the display screen, and centering it vertically. The FRAMEBOX commands draw a box

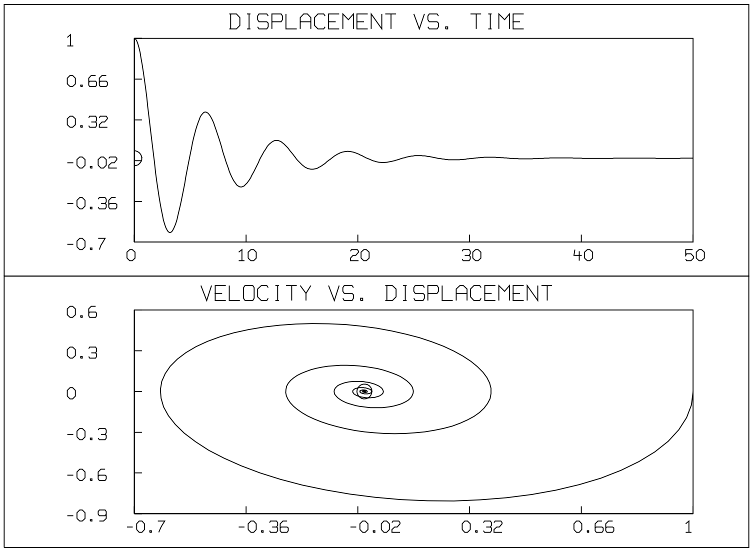


Figure 8.4: Trajectories of damped oscillator one above the other.

surrounding the frame of each window.

Pictures can be displayed one above the other, as well as side-by-side, as the next example shows.

Example 8-3: Display the previous two pictures one above the other instead of side by side. Note that this changes the aspect ratio. The MLAB statements for this are shown below and the resulting picture is displayed in Figure 8.4.

```
* FRAME 0 TO 1, 0 TO .5 IN W
* FRAME 0 TO 1, .5 TO 1 IN W1
* VIEW
```

The `IN`-clauses in these `FRAME` statements cause them to apply to the pictures in the windows `W` and `W1`, respectively.

In the previous examples, the pictures were non-overlapping. It is sometimes convenient to have two pictures shown so that one appears to lie on top of the other. This is shown in the next example.

Example 8-4: Draw two graphs where one is inset on the other. The first graph will be of the dependence of reaction rate on substrate concentration for a simple enzyme-catalyzed reaction. The inset graph will show the transformation of this graph to double reciprocal form.

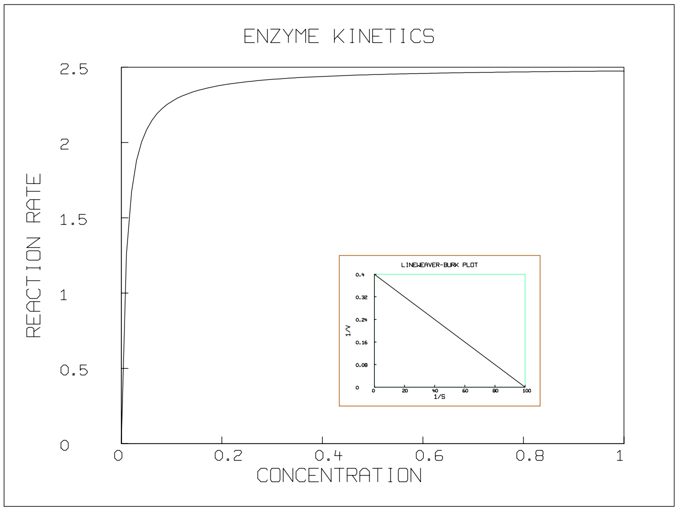


Figure 8.5: Lineweaver-Burk graph inset on enzyme kinetics graph.

```
* DEL W,W1
* FCT V(S) = VM*S/(KM+S)
* VM = 2.5; KM = .01
* DRAW POINTS(V,0:1!100)
* TOP TITLE "ENZYME KINETICS"
* BOTTOM TITLE "CONCENTRATION"
* LEFT TITLE "REACTION RATE"
* W1 = W
* DRAW (0&'(1/VM)&((1/KM)&'0))
* TOP TITLE "LINEWEAVER-BURK PLOT"
* LEFT TITLE "1/V"
* BOTTOM TITLE "1/S"
* FRAME COLOR BROWN
* IMAGE COLOR TURQUOISE
* FRAME .5 TO .8, .2 TO .5 PRIORITY 1
* VIEW
```

The result is shown in monochrome in Figure 8.5.

The new features here are the `COLOR`-clause and `PRIORITY`-clause of the `FRAME` command and the `COLOR`-clause of the `IMAGE` command. By default, all pictures have priority 0 and are displayed in an unpredictable order. When one picture has a higher priority than another picture, the picture with the lower priority is drawn first. Since a picture created using the default window `W` has an *opaque* frame and image because of the image and frame colors, when it is displayed, it obscures whatever was previously on the screen in the same place. While this is convenient for insets as in this example, it causes unexpected effects when the user did not intend for the pictures to overlap, and they all occupy the full screen. Each picture in the list specified by the `VIEW` command is drawn (or all pictures in no list was specified) one after another. Each subsequent picture obscures the previous picture, and finally only the last picture to be drawn is visible.

Chapter 9

Getting information about 2D graphics objects

Once a window has been defined, MLAB provides several commands for obtaining information about the window and its constituent curves, axes, and titles. This chapter discusses these commands in the context of an example from astronomy: the Hertzsprung-Russell diagram (see [Jas] in the bibliography).

Example 9-1: Draw the log-log plot of Surface Temperature versus Absolute Luminosity for 172 stars in the galaxy. Label the x- and y-axes and identify the main sequence with a title. The MLAB commands for the diagram follow. See Figure 9.1 for the result. The data file, `hr.dat` containing (surface temperature, absolute luminosity) ordered pairs appears in Appendix D.

```
* M = READ("hr.dat",172,2)
* M = MAPPLY(LOG10,M)
* M COL 1 = -M COL 1
* WINDOW -LOG10(40000) TO -LOG10(2000), -5 TO 6 IN HR
* DRAW M PT CIRCLE LT 0 PTSIZE .005 IN HR
* IMAGE .5 TO 5, .5 TO 5 INCHES IN HR
* TITLE "Hertzsprung-Russell Diagram" AT (.075,.95) FFRACT SIZE .02 IN HR
* TITLE "Absolute luminosity (luminosity of sun = 1)" AT (.05,.25)\
:     FFRACT ANGLE 90 SIZE .01 IN HR
* TITLE "Surface temperature of star (degrees Kelvin)" AT (.125,.025)\
:     FFRACT SIZE .01 IN HR
* XT = 2000:10000:1000 & 12000:20000:2000 & 25000:40000:5000
* XXT = XT
* FOR I = 1:NROWS(XT) DO {XT[I] = -LOG10(XT[I]);}
* XT COL 2 = -5
* XAXIS X1 = XT PT DTICK IN HR
```

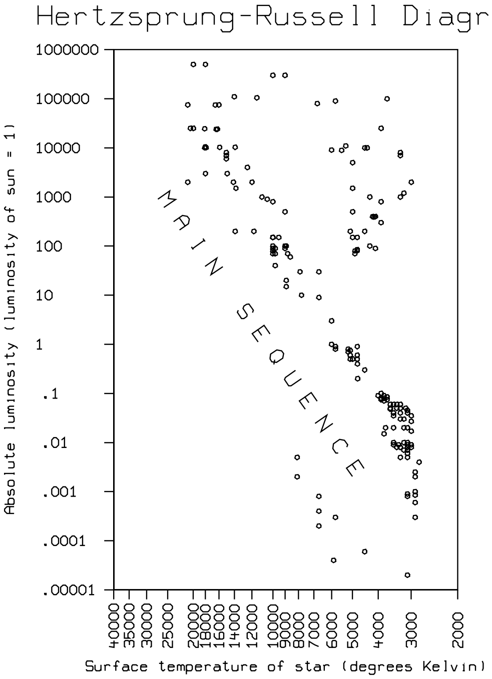


Figure 9.1: Hertzsprung-Russell diagram.

```
* XAXIS XT LABEL XXT LABELSIZE .01 ANGLE 90 OFFSET (.005,-.06) IN HR
* YYT = LIST(.00001,.0001,.001,.01,.1,1,10,100,1000,10000,100000,1000000)
* YT COL 2 = -5:6
* YT COL 1 = -LOG10(40000)
* YAXIS Y1 = YT PT LTICK IN HR
* YAXIS YT LABEL YYT LABELSIZE .01 FORMAT(-3,9,6,0,2,0) \
:      OFFSET (-.08,-.005) IN HR
* TITLE "M A I N   S E Q U E N C E" AT (.2,.7) FFRACT SIZE .015 \
:      ANGLE -55 IN HR
* VIEW HR
```

The ordered pair data are read into the matrix `M` . The logarithm (base 10) is applied to all elements of `M` by use of the `MAPPLY`-operator which takes a built-in function name as its first argument and a matrix name as its second argument.

Since the abscissae of the Hertzsprung-Russell diagram increase from right to left, we negate the first column of `M`, set the range of window x-coordinates to $[log_{10}(40000):log_{10}(2000)]$ and the range of window y-coordinates to $[log_{10}.000001 = -5:log_{10}1000000 = 6]$, and plot circles at the resulting negative abscissae and positive ordinates.

The image is set equal to a 4.5 by 4.5 inch rectangle with lower left corner at (.5,.5) inches with respect to the lower left corner of the display. The top title and axes titles are then defined. Then the x-axis and y-axis are drawn and labeled with positive label numbers.

Example 9-2: Obtain a list of all currently defined windows. This is achieved by the command

```
* WINDOWS
```

to which MLAB responds,

```
currently defined WINDOWS
HR
```

Example 9-3: Find the limits of the window `HR`. This is determined by the MLAB command `WINDOWM` which takes a window name as its first argument,

```
* WINDOWM(HR)
```

This causes the following output to be printed:

```
    : a  4 by 1 matrix

  1: -4.60205999
  2: -3.30103
  3: -5
  4: 6
```

The first and second elements of this matrix are the range of x- coordinates in the window: $[-4.60205999 : -3.30103]$; the third and fourth elements of this matrix are the range of y-coordinates in the window: $[-5 : 6]$.

Example 9-4: Find the x- and y-extents of the frame in the `HR` window in both `FRACT` and `INCH` units. The extents of the frame may be obtained by using the `FRAMEM` command. `FRAMEM` takes a window name as its first argument and an optional number specifying the units as a second argument. If the second argument is 1 the extents are returned in `FRACT` units, and if the second argument is 2 the extents are returned in `INCH` units. 2 is the default value of the second argument if it is omitted. Therefore,

```
* FRAMEM(HR,1)
```

results in

```
      : a  4 by 1 matrix

    1: 0
    2: 1
    3: 0
    4: 1
```

and

```
  * FRAMEM(HR)
```

or

```
  * FRAMEM(HR,2)
```

results in

```
      : a  4 by 1 matrix

    1: 0
    2: 9.75
    3: 0
    4: 7.11
```

This means that the extent of the frame of the window HR expressed in FRACT units is [0:1] in both x and y; and expressed in INCH units is [0:9.75] in x and [0:7.11] in y.

Example 9-5: Find the x- and y-extents of the image. The extents of the image can be determined using the IMAGEM command. The first argument of the IMAGEM command specifies the name of the window and the second argument determines the units: 1 - FRACT, 2 - INCHES, 3 - FFRACT, and 4 - FINCHES. The second argument is optional and defaults to FRACT units if it is not included. Therefore,

```
  * IMAGEM(HR)
```

or

```
  * IMAGEM(HR,1)
```

results in

```
      : a  4 by 1 matrix

    1: .153846154
    2: .512820513
    3: .140646976
    4: .914205345
```

and

```
  * IMAGEM(HR,2)
```

results in

```
      : a  4 by 1 matrix

    1: 1.5
    2: 5
    3: 1
    4: 6.5
```

and

```
  * IMAGEM(HR,3)
```

results in

```
      : a  4 by 1 matrix

    1: .153846154
    2: .512820513
    3: .140646976
    4: .914205345
```

and

```
  * IMAGEM(HR,4)
```

results in

```
    : a  4 by 1 matrix

  1: 1.5
  2: 5
  3: 1
  4: 6.5
```

These results tell us that the x-range of the image is [.153846154 : .512820513] in `FRACT` and `FFRACT` units and [1.5:5] in `INCH` and `FINCH` units, and that the y-range of the image is [.140646976 : .914205345] in `FRACT` and `FFRACT` units and [1 : 6.5] in `INCH` and `FINCH` units.

Example 9-6: Obtain even more comprehensive information about the window `HR`. This can be achieved in just one command, by entering

```
* TYPE HR
```

or simply the name of the window,

```
* HR
```

This will cause MLAB to print the following:

```
window: HR (UNBLANKED) priority: 0
window clipping limits (world units)wxmin:-4.60206 wxmax:-3.30103 wymin:-5 wymax:6
adjustment codes: WFIX, WFIX, WFIX, WFIX
frame  color: CLEAR (-2)  NO FRAMEBOX
frame in screen fraction units(width,height = 1):
x:0 to 1, y:0 to 1

frame in screen inches:x:0 to 9.75, y:0 to 7.11

image  color: CLEAR (-2) IMAGEBOX,  color: WHITE (1)
image in frame inches:x:1.5 to 5, y:1 to 6.5
image in frame fraction units:x:0.153846 to 0.512821, y:0.140647 to 0.914205
image in screen fraction units:x:0.153846 to 0.512821, y:0.140647 to 0.914205
image in screen inches:x:1.5 to 5, y:1 to 6.5
CURVES: C0
AXES: HR.YAXIS,Y1,HR.XAXIS,X1
TITLES: T4,T3,T2,T1
```

This command reveals all of the window, frame, and image limits information that can be found from the `WINDOWM`, `FRAMEM`, and `IMAGEM` commands. It also tells us the `ADJUST` options for the x-minimum, x-maximum, y-minimum, and y-maximum window limits, and the number and names of all curves (one named `C0`), axes (four named `HR.YAXIS, Y1, HR.XAXIS,` and `X1`), and titles (four named `T4, T3, T2,` and `T1`).

Further information about the curve, axes, or titles may be obtained by typing the name of the specific curve, axis, or title.

Example 9-7: Find out more about the curve C0 in the window HR. This can be achieved by entering

```
* TYPE C0
```

or simply the curve name,

```
* C0
```

Both of these commands will produce the following information about curve C0:

```
curve: C0, (UNBLANKED) in window: HR
curve data matrix: 172 rows
points: pointtype: 13, font: 5, size: 0.005 FFRACT color: WHITE (1)

lines: linetype: 0,  color: WHITE (1)
no label matrix
labels: font: 5, size: 0.02, FFRACT color: WHITE (1)
format: (-3,5,0,0,2,0), angle: 0
 place (left, bottom) xoffset: 0, yoffset: 0 FFRACT
initial axis window: -4.60206, -3.30103, -5, 6
wclipsw: 1
```

This tells us the name of the curve; the window in which the curve appears; the dimension of the curve's data array; the size, color, and pointtype symbol of the curve if the curve is drawn with a pointtype; the linetype and color of the curve if it is draw with a linetype; the existence of an associated curve label; the font, point size, color, format, angle, and offset of the associated curve label if it exists; and the initial axis window.

Example 9-8: Obtain a list of the points forming the curve C0. This is accomplished by entering the command:

```
* CURVEM(C0)
```

This causes the complete curve matrix to be printed:

```
      : a  172 by 2 matrix

   1: -4.30103      5.69897
   2: -4.25527251   5.69897
   3: -4            5.47712125
   4: -3.95424251   5.47712125
   .      .            .
   .      .            .
   .      .            .
 171: -3.51851394   -1.52287875
 172: -3.50514998   -1.52287875
```

Example 9-9: Find out information about the ”MAIN SEQUENCE” title in the window `HR`. This is done by entering

```
 * TYPE T4
```

or simply the name of the title,

```
 * T4
```

and results in the following description:

```
 title name: T4 (UNBLANKED) in HR
 title text: M A I N   S E Q U E N C E
 font: 5, size: 0.015 FFRACT units
 angle: 305
 at: 0.2, 0.7 in FFRACT units,  place (left, bottom)
 color: 1, shift: 0
```

This tells us the title name; the window in which it is found; the text of the title; and the font, point size, angle, position, and color of the title.

Example 9-10: Find out more about the axis named `HR.AXIS` in the window `HR`. This is accomplished by entering

```
 * TYPE HR.YAXIS
```

or simply the axis name,

```
 * HR.YAXIS
```

This will produce the following:

```
yaxis: HR.YAXIS, (UNBLANKED) in window: HR
curve data matrix: 12 rows
points: pointtype: 0, font: 5, size: 0.02 FFRACT color: WHITE (1)

lines: linetype: 1,  color: WHITE (1)
label matrix: 12 rows
labels: font: 5, size: 0.01, FFRACT color: WHITE (1)
format: (-3,9,6,0,2,0), angle: 0
 place (left, bottom) xoffset: -0.08, yoffset: -0.005 FFRACT
initial axis window: -4.60206, -3.30103, -5, 6
wclipsw: 0
```

This tells us the name of the axis; the window in which the axis is found; the dimension of the curve's data matrix; the pointtype, font, point size, and color of the points if there is a pointtype; the linetype and color if there is a linetype; the dimension of the label matrix; the font, point size, and color of the labels if there is a label matrix; the format, angle, and offset if there are labels; and the limits of the window in which the axis is located.

Example 9-11: List the points where the labels for the `HR.YAXIS` are found. The `LABELM` function can obtain these labels directly.

```
* LABELM(HR.YAXIS)
```

results in

```
     : a  12 by 1 matrix

   1: 0.00001
   2: 0.0001
   3: 0.001
   4: 0.01
   5: 0.1
   6: 1
   7: 10
   8: 100
   9: 1000
  10: 10000
  11: 100000
  12: 1000000
```

Example 9-12: Find the point size of the labels on the `HR.YAXIS` axis. We can do this by using the `PTSIZEM` command.

```
  * PTSIZEM(HR.YAXIS)
```

This results in

```
      : a  1 by 1 matrix

    1: 0.02
```

This tells us that the point size of the left tic marks on the y-axis are .02 frame fraction units.

Chapter 10

Saving, plotting, and reusing graphics windows

Once an MLAB graphics window has been created on the display, MLAB commands are available to plot, save, and use that window and/or its contents. In this chapter we review these commands.

MLAB supports the following types of output files for graphics:

`PSL`	PostScript language, landscape orientation
`PSP`	PostScript language, portrait orientation
`PSCL`	Color PostScript language, landscape orientation
`PSCP`	Color PostScript language, portrait orientation
`LJ2L`	HP LaserJet Printer Control Language (PCL), landscape orientation
`LJ2P`	HP LaserJet Printer Control Language (PCL), portrait orientation

PostScript is a graphics language developed by Adobe Systems. PostScript output files consist of English commands with numerical arguments that can be sent to a printer, edited, embedded in word processor documents, or converted to other file types, such as .pdf files, using other graphics programs.

HP LaserJet Printer Control Language is another graphics language developed by Hewlett Packard. Unlike the PostScript language, HP LJ PCL files consist of hexadecimal codes for commands and are usually inscrutable when opened with file editor programs. These files are usually sent directly to the printer.

landscape orientation means the output is printed on an 8.5 x 11 inch page with the larger dimension of the page oriented horizontally; *portrait* orientation means the output is printed on an 8.5 x 11 inch page with the larger dimension of the page oriented vertically.

In MLAB, the type of output file is specified by assigning one of the string variable names listed in the table above to the MLAB variable PLOTDEV. The default value of PLOTDEV is PSL.

Once the desired output file-type is specified, the MLAB command PLOT <windowname> is used to generate the actual output file depicting the named window. The command PRINT FILE <filename> can then be used to send the file to the printer. These commands are demonstrated in the following example.

Example 10-1: Draw a simple graph in window WS, set the output file type to PostScript with portrait orientation, generate an output file named smplpic.ps containing the simple graph in the PostScript language, and send the output file to the printer.

```
* DRAW 1:2 IN WS
* PLOTDEV = PSP
* PLOT WS IN "smplpic.ps"
PLOT file is "smplpic.ps"
* PRINT FILE "smplpic.ps"
file printed: smplpic.ps
```

Note in the preceding example, MLAB responded to the PLOT command with a message indicating the plot file was created; and MLAB responded to the PRINT FILE command with a message indicating that the file was sent to the printer. If for some reason the plot file is not created or the printer fails to print the named file, MLAB will print an error message. Also, when executing a PLOT command, if a file with the specified filename already exists in the MLAB working directory, then MLAB will ask the user if the named plot-file should be overwritten or kept, and wait for a response.

The most general form of the PLOT command is

```
* PLOT [<windowname1>[,<windowname2>,...]] IN <filename>
```

where <windowname1>, <windowname2>, ... are designated window names, and <filename> is the name of the output file. If no window names are specified, all visible windows are included by default. For PLOTDEV = PSL, PSP, PSCL, and PSCP, if no output filename is specified in the PLOT command, then <filename> defaults to the file named mlabp # .ps , where # is the lowest consecutive integer starting from 0 that yields a unique filename.

The next example shows how a Hewlett Packard LaserJet PCL file is created using the PLOT command and printed using the PRINT FILE command.

Example 10-2: Create a Hewlett Packard LaserJet PCL file depicting the window of the previous example, in landscape mode using default arguments.

This is done by entering:

```
* PLOTDEV = LJ2L
* PLOT
PLOT file is "mlabp0.lj"
* PRINT FILE "mlabp0.lj"
file printed: mlabp0.lj
```

For plotting devices `LJ2L` and `LJ2P` the default output file of the `PLOT` command is the file named `mlabp # .lj`, where # is the lowest consecutive integer starting from 0 that does not yield an existing filename.

The colors used in an MLAB `PLOT` are selected so that black is plotted as white (which will not show on the plot), and white is plotted as black (which will show on the plot), and all other colors are plotted as various shades of grey. Actually, they are specified as colors in Postscript and converted to grey-scale shades if the PostScript printer is not a color device. Only dots and lines (including the lines comprising text characters) are plotted; filled regions, i.e. the screen, frame, and image colors, are ignored.

Once one or more windows have been defined in MLAB, they may be saved in a file on disk so as to be used in another MLAB session. This is demonstrated in the following example.

Example 10-3: Save the default window with its constituent titles, axes, and curves in a file for use in another MLAB session.

This is achieved by

```
* SAVE <windowname> IN <filename>
```

Here `<windowname>` is the name of the window to be stored. If no name is supplied, the default window `W` is saved. The `IN`- clause must be included and a name of a file must be supplied.

The saved window can be restored in another MLAB session by typing

```
* USE <filename>
```

where `<filename>` is the name of the file used in the `SAVE` command.

Example 10-4: Redefine the default window, `W`. The default window `W` used by MLAB has been pre-defined by the following MLAB commands:

```
* WINDOW 0 TO 10, 0 TO 10 ADJUST WNICE IN W
* XAXIS 0:10!6&'0 LABEL 0:10!6 LABELSIZE .015 FFRACT OFFSET (-.01,-.025)\
:    FFRACT FORMAT(-3,5,0,0,2,0) PT UTICK IN W
* YAXIS 0&'0:10!6 LABEL 0:10!6 LABELSIZE .015 FFRACT OFFSET (-.09,-.01)\
:    FFRACT FORMAT(-3,5,0,0,2,0) PT RTICK IN W
* FRAME COLOR VIOLET IN W
* SAVE W IN DW
```

This description of the window `W` is saved in the file `DW.SAV` which appears in the MLAB directory. Whenever MLAB needs a default window, it secretly executes `USE DW`. Any of the characteristics of this window may be changed at will by making your own version of the `DW.SAV` file. For example, 4 labels will appear on the yaxis instead of 6 labels if the following commands are given:

```
* REUSE DW.SAV
* DELETE W.YAXIS
* YAXIS 0&'0:10!4 LABEL 0:10!4 LABELSIZE .015 FFRACT OFFSET \
:    (-.09,-.01) FFRACT FORMAT(-3,5,0,0,2,0) PT RTICK IN W
```

Then the command

```
* RESAVE W IN DW.SAV
```

will save the revised description of the default window to the MLAB directory and the revised default window will be used for all subsequent drawing in the default window.

Apart from MLAB, most operating systems provide a means for capturing the contents of the screen or a window on the screen as a bitmap or pdf file.

On PCs with Microsoft Windows, holding the Alt-key and hitting the PrintScrn key on the keyboard (the "PrintScrn"-key is at the top of the keyboard) "grabs" the "active" window and saves it as a bitmap in the "clipboard". If you want the whole screen to be grabbed as a bitmap, then you just hit the "PrintScrn"-key by itself. Then the contents of the clipboard can be pasted to a document, saved to a file, or imported to a graphics application as needed.

On Macintosh PCs, the screen-capture key combination (Command-Shift-3) or the window-capture key combination (Command-Shift-4) can be used to generate PDF file images of the screen or MLAB graphics window, respectively. Such PDF files are stored on the disk in the directory: `/Users/[Username]/Desktop` with names such as "Picture 1.pdf", "Picture 2.pdf", etc.

On Linux with X-Windows PCs, the screen-capture and window-capture functions differ according to the Desktop Manager in use. For Linux with X-Windows and the Gnome Desktop Manager, the gnome-screenshot utility is provided in the Gnome–Accessories menu. For Linux with X-Windows and the KDE Desktop Manager, a screen-capture application is found in the KDE Menu under Applications–Graphics–Ksnapshot.

Chapter 11

Miscellaneous 2D graphics examples

In this chapter we show some more 2d graphics examples using MLAB.

Example 11-1: Space-filling curves were first discovered by mathematicians seeking an example of a curve that is continuous but nowhere differentiable. A space-filling curve also has the property that it completely fills a two dimensional region. A space-filling curve that has these properites can be found as the limit of a sequence of curves that do not have these properties. Draw curves which in the infinite limit yield the Hilbert space-filling curve.

```
* N=0; S=1; E=2; W=3
* FCT MOVE(A) = \
:      (C[I+1,1]_C[I,1]+(IF A=E THEN H ELSE IF A=W THEN -H ELSE 0))+\
:      (C[I+1,2]_C[I,2]+(IF A=N THEN H ELSE IF A=S THEN -H ELSE 0))+\
:      (I_I+1);
* FCT HILB(R,D,L,U,LEV) = IF LEV<2 THEN (MOVE(R)+MOVE(D)+MOVE(L)) ELSE\
:      (HILB(D,R,U,L,LEV-1)+MOVE(R)+HILB(R,D,L,U,LEV-1)+MOVE(D)+\
:      HILB(R,D,L,U,LEV-1)+MOVE(L)+HILB(U,L,D,R,LEV-1));
* WINDOW 0. TO 1., 0. TO 1. IN WNDW
* NO IMAGEBOX IN WNDW
* IMAGE .5 TO 5., .5 TO 5. INCHES IN WNDW
* I = 1
* LEV = 5
* H = 1./(2^LEV-1)
* C = LIST(0,0)'
* C[2^(2*LEV),2] = 0
* RES = HILB(E,N,W,S,LEV)
* DRAW HC = C IN WNDW
* VIEW WNDW
```

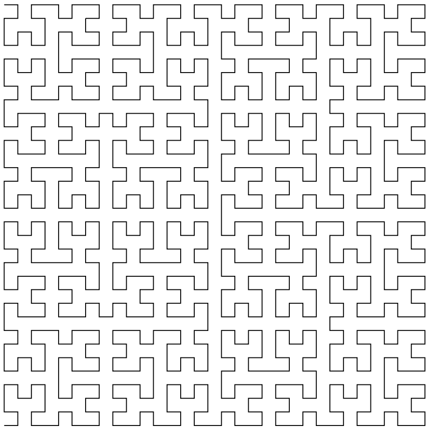


Figure 11.1: Hilbert space-filling curve at recursion level 5.

The resulting picture is shown in Figure 11.1.

The variables N,S,E, and W are used as constants to represent four different directions. They can have any values as long as they are all different. The function MOVE computes the coordinates for the $(i+1)^{st}$ row of the matrix C from the i^{th} row as a side effect. All steps are of length H. The function HILB builds the curve matrix, C, by calling the function MOVE repeatedly. Try this example with different values for the variable LEV.

Example 11-2: Draw a finite recursive approximation to the Dragon fractal at a prespecified level of recursion. The approximations to the Dragon fractal are generated by the following process: Fold a long strip of paper, right half over left, and open it out to a right angle. Viewed edge-on, this is the first order approximation to the dragon fractal. Now close the strip and fold it in half again, in the same direction as the first fold, and open it out again so that each fold is a right angle. Repeat this process. The results, again viewed edge on, are the second and third order approximations to the Dragon fractal. The Dragon fractal is the continuous but non-differentiable limiting curve.

Figure 11.2 shows the tenth order approximation to the Dragon fractal. It was generated as follows using MLAB:

```
* RESET
* P = 10; PION2 = PI/2; PON2 = P/2;
* H = 2^(-PON2); S = 0;
* FCT G(X) = IF X = 1 THEN 1 ELSE IF MOD(X,2) = 0 THEN G(X/2)\
:          ELSE IF MOD(X,4) = 1 THEN 1 ELSE -1;
```

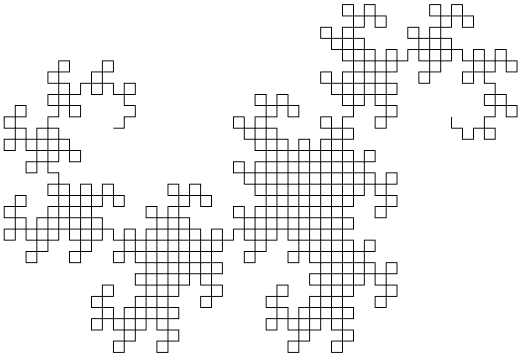


Figure 11.2: Dragon fractal at recursion level 10.

```
* C[1,1] = H*COS(P*PI/4)
* C[1,2] = H*SIN(P*PI/4)
* FOR N = 1:(2^P-2) DO {
>     S = S+G(N); F = (S-PON2)*PION2;
>     C[N+1,1] = C[N,1]+H*COS(F); C[N+1,2] = C[N,2]+H*SIN(F);
>    }
* DRAW C IN W1
* WINDOW ADJUST WMATCH IN W1
* NO IMAGEBOX IN W1
* VIEW W1
```

The variable P defines the level of recursion. Once P is defined, we know that the approximation of the fractal will consist of 2**P-1 line segments each of length H = 2**(-P/2), where ** is the exponentiation operator, such that X**Y equals X^Y. The coordinates of the endpoints of the segments are collected in the matrix C. Try other values for P.

Example 11-3: Given (x, y) coordinates (1,1),(2,3),(3,4),(4,3.2), and (5,3.5) plot the points using an X, draw a smooth curve through the points using the INTERPOLATE operator to generate a smooth curve, and draw a step-function interpolation for the points using the BARGRAPH operator. Create a window WI for the picture. The MLAB commands follow:

```
* RESET
* M = 1:5 &' LIST(1,3,4,3.2,3.5)
* P = INTERPOLATE(M,.5:5.5:.1)
* WINDOW 0 TO 6, 0 TO 4.5 IN WI
```

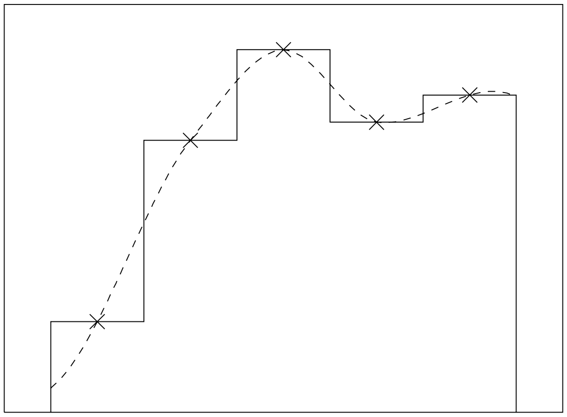


Figure 11.3: Interpolation on 5 points in the plane with bargraph.

```
 * DRAW M IN WI, LT NONE PT 8
 * DRAW P IN WI, LT 2
 * DRAW BARGRAPH(M) IN WI
 * VIEW WI
```

The resulting picture is shown in Figure 11.3.

The INTERPOLATE operator performs a cubic spline interpolation of the data in M at the values .5,.6,.7,. . . ,5.5. The BARGRAPH function will return an appropriate step-function approximation of the curve given by the input matrix. The input matrix must be in sort by column one for this to work.

Example 11-4: Draw splines through 10 random points in the plane, demonstrating the four different options of the SPLINE operator.

```
 * DEL M,N,W
 * M COL 1 = 1:10
 * M COL 2 = NORMRAN ON 0^^10
 * DRAW M LT NONE PT CIRCLE
 * N = 0:1!150
 * FOR I = 0:3 DO {DRAW SPLINE(M,N,I) COLOR COLORN(I);}
 * VIEW W
```

The resulting picture is shown in Figure 11.4.

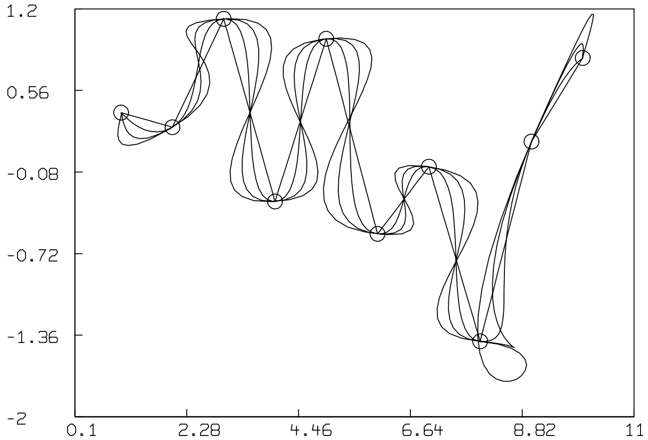


Figure 11.4: Four options of the spline function in connecting 10 random points.

The SPLINE command constructs a matrix which contains the result of interpolation into a plane or space curve which is implicitly parameterized by relative chordal arc-length ranging from 0 to 1. The first argument in this example is a 2 column matrix containing the curve data. The second argument is a 1-column matrix, which holds the interpolation chordal arc-length parameters. The third argument is an optional flatness parameter which defaults to 1. As the third argument becomes smaller, the interpolating curve becomes increasingly flat. In the limit, as the third argument goes to 0, the interpolating curve becomes piecewise straight between the input data points.

Example 11-5: Fill an arbitrary polygon with forty fill lines drawn at an angle of 45 degrees. This is achieved using MLAB as follows:

```
* DEL M,N,W
* M = SHAPE(20,2,RAN ON 0^^40)
* DRAW M PT CIRCLE
* N ROW 1 = M ROW 1
* N ROW 2 = M ROW 20
* DRAW N
* DRAW FILL(M,40,45) LT ALTERNATE COLOR RED
* VIEW W
```

The resulting picture is shown in Figure 11.5.

The FILL operator generates an array of (x, y) coordinates which are endpoints for parallel line segments that fill a polygon. The vertices of the polygon are provided in the first argument to

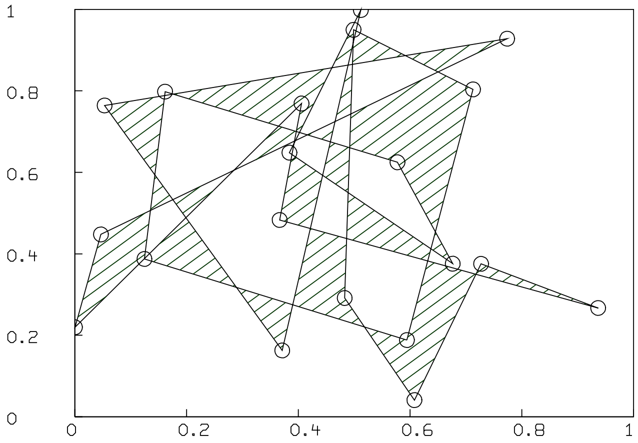


Figure 11.5: A polygon of 20 vertices filled with 40 lines drawn at an angle of 45 degrees.

FILL which is a 2 column matrix. The second argument to FILL determines the number of fill lines to be drawn and the third argument determines the angle with respect to the horizontal at which the lines are to be drawn. The resultant line segment endpoints are suitable for drawing with the ALTERNATE linetype.

Example 11-6: Draw a contour plot of the Hénon-Heiles potential energy surface and draw a bound, low energy trajectory. Then draw a Poincaré surface of section in the $x = 0$ plane.

The contour plot and trajectory are shown in Figure 11.6 and generated as follows:

```
* RESET
* FCT V(X,Y) = (X^2+Y^2)/2 + X^2*Y - Y^3/3
* Q = CROSS(-2:2:.2,-2:2:.2)
* Q COL 3 = V ON Q
* C = CONTOUR(Q,LIST(-.15,0,.05,.1,.15,1/6,.2,.3,.4,.6,.8,1.0))
* DRAW C, LT 7
* FRAME .01 to .5, .25 to .75
* FRAMEBOX
```

Here V is the potential function introduced by Hénon and Heiles in the study of orbits of stars through a galaxy (see [Hen] in the bibliography. This potential can be thought of as a perturbed 2 dimensional harmonic oscillator potential with a three-fold symmetry. The potential is zero at the origin and becomes unbounded for large values of the coordinates. For energies less than 1/6 the trajectories are bound within an equilateral triangle.

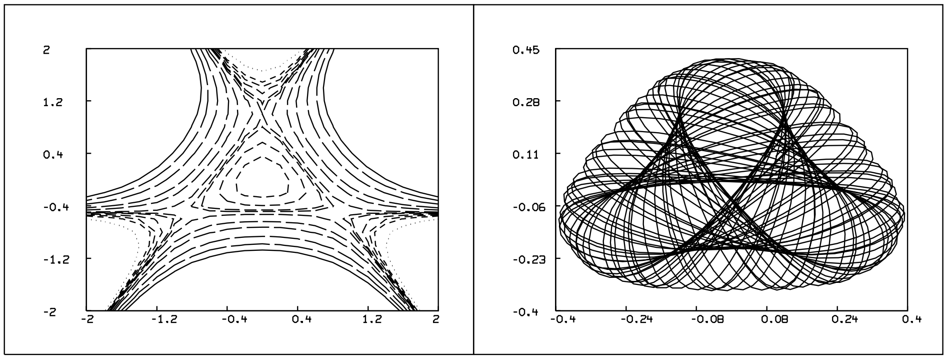


Figure 11.6: The Hénon-Heiles potential energy surface and a low energy trajectory.

```
* FCT H(X,Y,PX,PY) = (PX^2+PY^2)/2 + V(X,Y)
* FCT PX'T(T) = -H'X(X,Y,PX,PY)
* FCT PY'T(T) = -H'Y(X,Y,PX,PY)
* FCT X'T(T) = H'PX(X,Y,PX,PY)
* FCT Y'T(T) = H'PY(X,Y,PX,PY)
* T0 = 0
* TMAX = 300
* NSTEP = 1500
* INITIAL X(T0) = .1
* INITIAL Y(T0) = .1
* INITIAL PX(T0) = .2
* INITIAL PY(T0) = .3
* M = INTEGRATE(X,Y,PX,PY,T0:TMAX!NSTEP)
* W1 = W
* DRAW M COL(2,4)
* FRAME .5 to .99, .25 to .75
* FRAMEBOX
* VIEW
```

The function `H` is the Hamiltonian or total energy of the system. The next four functions `PX'T`, `PY'T`, `X'T`, and `Y'T` define Hamilton's equations of motion for the system. There are four initial conditions which have been selected so that the trajectory will be bound. Note that the

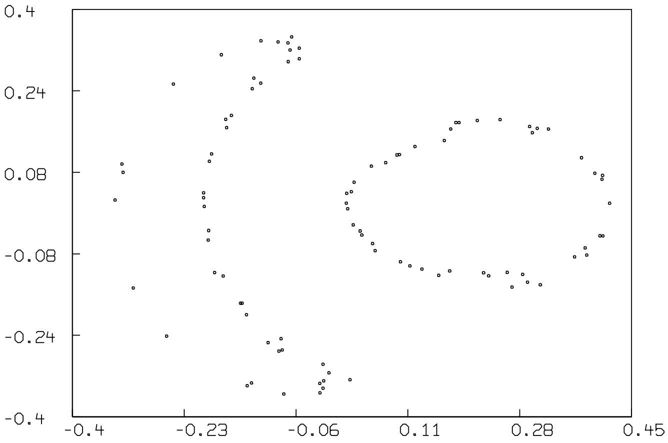


Figure 11.7: Poincaré surface of section for the trajectory in previous figure.

trajectory is drawn on a different scale than the potential surface and that it is limited to the region in the largest triangle of the contour plot.

An approximate Poincaré surface of section in the $x = 0$ plane can be generated by plotting those (y, py)-coordinate pairs at which $x = 0$. This is accomplished as follows:

```
* DELETE W,W1
* K = 1
* L = NROWS(M)-1
* FOR I = 1:L DO {
>   IF (SIGN(M[I,2])NOT=SIGN(M[I+1,2])) THEN {MM ROW K = M ROW I; K = K+1;}
> }
* DRAW MM COL(4,5) LT NONE PT "o" PTSIZE .005
* VIEW
```

Here we search the differential equation solution matrix M for the rows where $x(t)$ changes sign (i.e. passes through $x = 0$) and plot a point $(y(t), y'(t))$ at that specific time. The collection of such points is the $x = 0$ Poincaré surface of section.

The resulting picture is shown in Figure 11.7.

Example 11-7: Draw the Voronoi regions, the corresponding Delaunay triangulation, and the circles which circumscribe the Delaunay triangles for a set of thirty random points. The Voronoi region for a given point in a set of points in the plane is that region of the plane which is closer to the given point than any other point in the set. These regions are either bounded by irregular

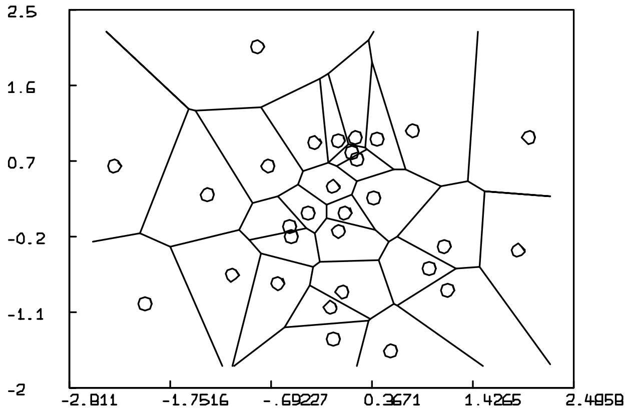


Figure 11.8: Voronoi regions for 30 random points in the plane.

polygons or unbounded. The Delaunay triangles for a given set of points in the plane are those triangles whose sides bisect the edges of the Voronoi regions. The Delaunay triangles have the property that their circumscribing circles do not contain any of the set points in their interiors.

This is done with MLAB as follows:

```
  * RESET
  * PTS30 = SHAPE(30,2,RAN ON 0^^60)
  * DRAW PTS30 LT NONE PT CIRCLE
  * DRAW VORCURVE(PTS30) LT MARKER
  * WINDOW ADJUST WMATCH
  * VIEW
```

The resulting picture is shown in Figure 11.8.

```
  * DELETE W
  * DRAW DELCURVE(PTS30) LT MARKER
  * DRAW PTS30 LT NONE PT CIRCLE
  * WINDOW ADJUST WMATCH
  * VIEW
```

The resulting picture is shown in Figure 11.9.

The MLAB operator `DCIRCLES` computes and draws the circumscribing Delaunay triangles, as shown in Figure 11.10, with the following commands:

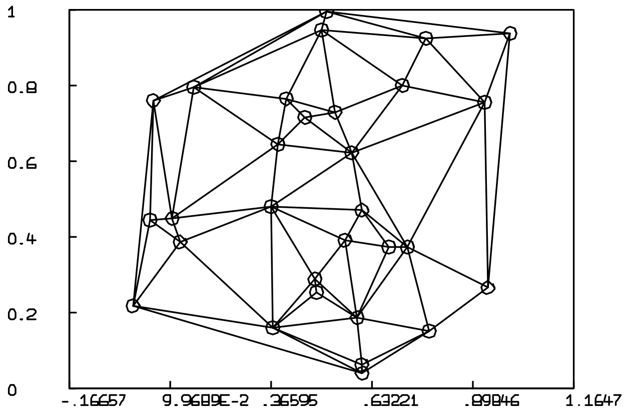


Figure 11.9: Delaunay triangles for 30 random points in the plane.

```
* DELETE W
* DRAW DCIRCLES(PTS30) LT MARKER
* WINDOW -.166657 TO 1.1647, 0 TO 1
* VIEW
```

Example 11-8: Demonstrate the dendrogram (tree) cluster analysis functions in MLAB. Begin by generating 3 groups of 5, 2-dimensional random variates and storing them in the matrix `A`.

```
* RESET
* NUM = 5
* NORMRAN(0)
    = 0.38611908
* A = SHAPE(NUM,2,NORMRAN ON 0^^(2*NUM))
* U = PI/6
* A = A*(DIAG(1:2))*(SHAPE(2,2,LIST(COS(U),-SIN(U),SIN(U),COS(U))))+ \
:     (2&'0)^^NUM
* B = SHAPE(NUM,2,NORMRAN ON 0^^(2*NUM))
* U = PI/4
* B = B*(DIAG(2:1))*(SHAPE(2,2,LIST(COS(U),-SIN(U),SIN(U),COS(U))))+ \
:     (-2&'5)^^NUM
* C = SHAPE(NUM,2,NORMRAN ON 0^^(2*NUM))
* U = 2*PI/3
* C = C*(DIAG(2&.5))*(SHAPE(2,2,LIST(COS(U),-SIN(U),SIN(U),COS(U))))+  \
:     (-1.5&'-4)^^NUM
* A = A&B&C
```

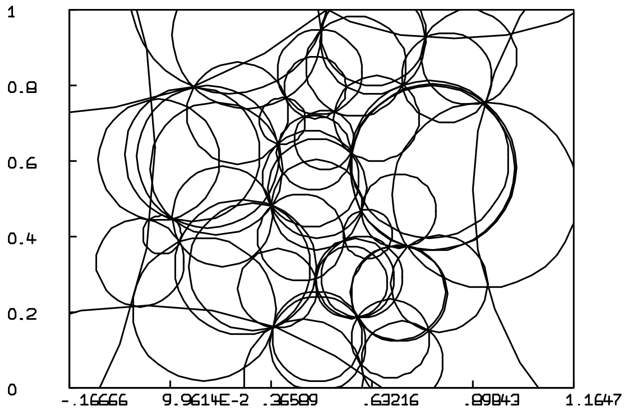


Figure 11.10: Circles circumscribing Delaunay triangles for 30 random points in the plane.

Next, draw the 15 random variates in the plane and the minimum spanning tree. The minimum spanning tree for a set of points is a way of connecting all of the points in the set with straight line segments such that the sum of the lengths of the line segments is a minimum.

```
* D = DISTS(A)
* E = MST(D)
* F = TREECURVE(E,1,(1:(3*NUM))&'A)
* DRAW F COL 1:2 LT ALTERNATE LABEL F COL 3 LABELSIZE 0.02 COLOR RED
* TOP TITLE "MINIMAL SPANNING TREE" SIZE .04
* W6 = W
* FRAME 0 TO .334, .5 TO 1 COLOR YELLOW IN W6
* NO IMAGEBOX IN W6
```

The single link, complete link, average link, centroid linkage, and Ward's linkage dendrograms, are drawn as follows:

```
* T = SLINK(D)
* N = DENCURVE(T)
* FRAME .334 TO .667, .5 TO 1 COLOR BLUE IN W1
* DRAW N COL 1:2 LABEL N COL 3:4 LINETYPE MARKER LABELSIZE .015 IN W1
* YAXIS A1 = 0 &' 0:1!6 LABEL 0:1!6 LABELSIZE .015 PT RTICK \
: OFFSET (-0.09,-0.01) IN W1
* NO IMAGEBOX IN W1
* TOP TITLE "SINGLE LINK" SIZE .04 IN W1
* T = ALINK(D)
* N = DENCURVE(T)
* FRAME .667 TO 1, .5 TO 1 COLOR RED IN W2
* DRAW N COL 1:2 LABEL N COL 3:4 LINETYPE MARKER LABELSIZE .015 \
: COLOR AQUA IN W2
* NO IMAGEBOX IN W2
* TOP TITLE "COMPLETE LINK" SIZE .04 IN W2
* T = CLINK(D)
* N = DENCURVE(T)
* FRAME 0 TO .334, 0 TO .5 COLOR ROSE IN W3
* DRAW N COL 1:2 LABEL N COL 3:4 LINETYPE MARKER LABELSIZE .02 \
: COLOR TURQUOISE IN W3
* YAXIS A2 = A1 IN W3
* NO IMAGEBOX IN W3
* TOP TITLE "AVERAGE LINKAGE" SIZE .04 IN W3
* T = CENTROID(D)
* N = DENCURVE(T)
* FRAME .334 TO .667, 0 TO .5 COLOR PINK IN W4
* DRAW N COL 1:2 LABEL N COL 3:4 LINETYPE MARKER LABELSIZE .02 \
: COLOR VIOLET IN W4
* NO IMAGEBOX IN W4
* TOP TITLE "CENTROID LINKAGE" SIZE .04 IN W4
* T = WARD(D,1)
* N = DENCURVE(T)
* FRAME .667 TO 1, 0 TO .5 COLOR GREEN IN W5
* DRAW N COL 1:2 LABEL N COL 3:4 LINETYPE MARKER LABELSIZE .02 \
: COLOR BLACK IN W5
* NO IMAGEBOX IN W5
* TOP TITLE "WARD’’S LINKAGE" SIZE .04 IN W5
* VIEW
```

The diagram shown in Figure 11.11 shows the results.

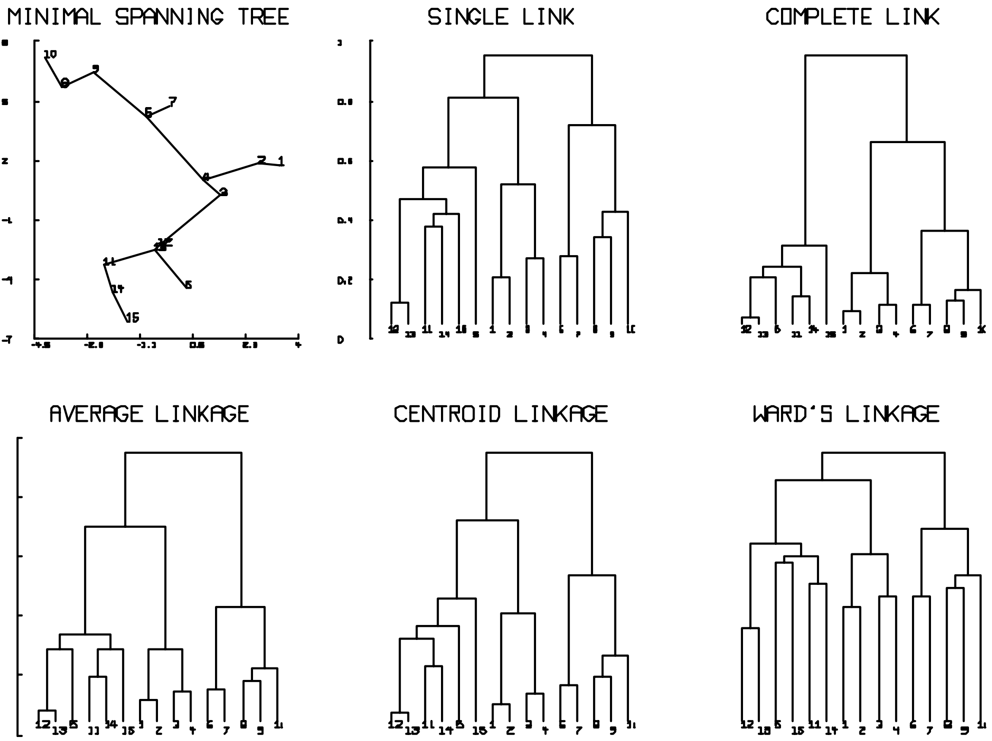


Figure 11.11: Minimum spanning tree and 5 dendrograms for normally distributed data.

Chapter 12

3D Graphics

MLAB also has facilities for drawing two dimensional perspective views of three dimensional data. These facilities are based on a camera paradigm in which camera position, view direction, and scale are controlled by explicit commands. This chapter demonstrates some of MLAB's three dimensional graphics capabilities.

Example 12-1: Demonstrate the effect of the DRAW statement when the data is a three column array and demonstrate some of the MLAB 3-D View commands.

Consider the commands:

```
* FCT F(X,Y) = X+Y
* M = CROSS(0:1!10,0:1!10)
* M COL 3 = F ON M
* DRAW M
* VIEW
```

The first three statements build a matrix called M which contains the coordinates of 100 points $(x, y, x+y)$ where x and y take on all pairwise combinations of the ten numbers $(0, .909, \ldots, 10)$. The VIEW statement results in the view shown if Figure 12.1.

When the data array given in the DRAW statement has 3 columns, as in this example, the effects of the DRAW statement are:

1. to create the default MLAB 3 dimensional graphics object named W3. Like the default 2 dimensional graphics window W, the default 3 dimensional graphics window W3 consists of a rectangular *frame* region; an *image* region; and a *window* clipping region. In addition

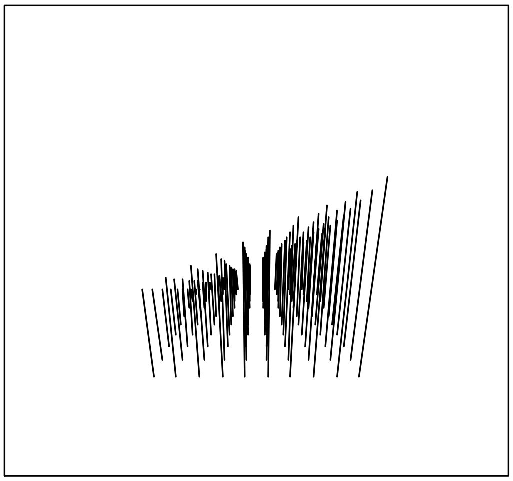


Figure 12.1: Needles drawn for the function $(x + y)$ at 100 points in the unit square.

to those elements of `W`, `W3` also includes a *clipping pyramid* region. The clipping pyramid is a volume of Cartesian (x, y, z)-space characterized by a view point, view direction, and view angle—similar to the volume of 3-space visible to a camera. The clipping pyramid is perspectively projected onto the 2 dimensional plane containing the window clipping region, and the window clipping region is mapped into the image region on the display.

2. to create needles in 3 dimensional space at the points in the 3 column data array. In MLAB, a *needle* at a point (x, y, z) is a line segment connecting the points (x, y, z) and $(x, y, 0)$.
3. to set the view point, view direction, and view angle of the clipping pyramid so that all of the needles are visible. All of the needles at the points of the data array are enclosed by a bounding box $(xmin \le x \le xmax; ymin \le y \le ymax; zmin \le z \le zmax)$ which measures $x1 = xmax - xmin$, $y1 = ymax - ymin$, and $z1 = zmax - zmin$ on the side parallel to the x, y, and z-axis, respectively. The center of the box is at the position $(xcenter = xmin + x1/2, ycenter = ymin + y1/2, zcenter = zmin + z1/2)$. The view point is set to $(xcenter, ymax + y1, zmax)$. The view direction is set to the direction which results from rotating a unit vector—initially pointing from the origin along the positive z-axis—by 105 degrees about the x-axis. The view angle is set to 90 degrees. From this perspective, the view is outside the box ”looking” at the box's center.

When the `VIEW` statement is given, the view of the needles is shown from a default position. A 3D-view command prompt `>>` is printed on the command line. The 3D-view command prompt indicates that MLAB will accept any one of a number of commands for changing the view position, view direction, angle of view, or object position where the object is the set of needles being viewed. Here we demonstrate five 3D-view commands.

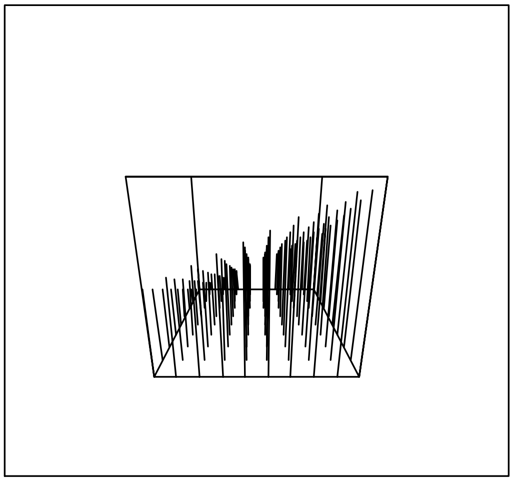


Figure 12.2: 100 needles with the enclosing box.

The BOX command draws the edges of the box that encloses the needles. For example,

```
>> BOX
```

results in Figure 12.2.

The RAISE command translates the camera view point along the z-direction. For example,

```
>> RAISE .5
```

translates the camera view point one half the height of the enclosing box in the direction of the positive z-axis. The resulting picture is shown in Figure 12.3.

The TRACK command changes the camera view direction so that it points from the current view position to the center of the enclosing box.

```
>> TRACK
```

The resulting picture is shown in Figure 12.4.

The TWIST command rotates the camera about the view direction axis. For example,

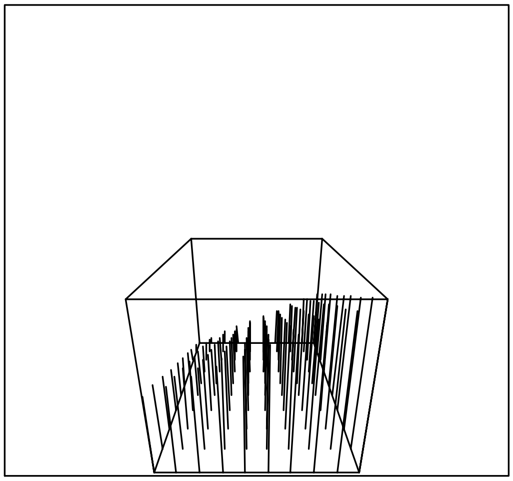


Figure 12.3: Raise the camera half a side of the enclosing box.

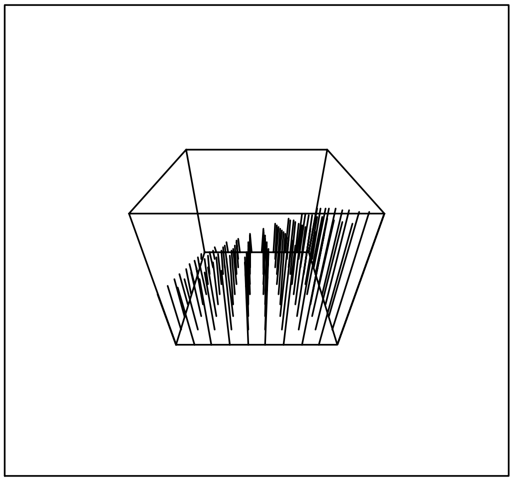


Figure 12.4: Redirect the camera toward the center of the enclosing box.

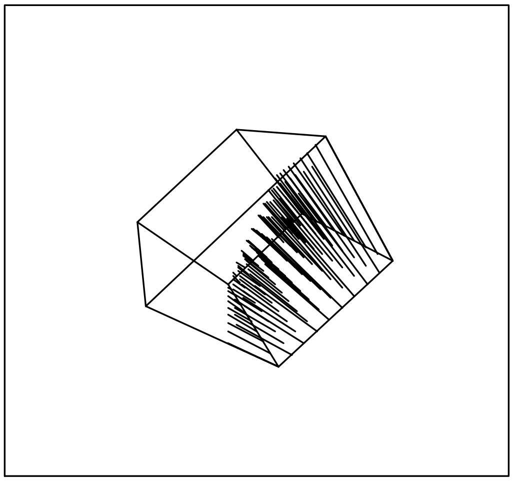


Figure 12.5: Rotate the camera 45 degrees about the view direction.

```
>> TWIST 45
```

rotates the view 45 degrees counterclockwise about the view direction. The result is shown in Figure 12.5.

Finally, the AXES command draws the coordinate axes of the data. For example,

```
>> AXES
```

The result is shown in Figure 12.6.

The 3D-view command dialog is terminated by striking the Return/Enter key at the 3D-view command prompt, >>.

A complete list of the 3D-view commands is given in Appendix C.

There are ten linetypes for 3D curves. These include:

NONE	(0)	no connecting lines.
NET	(1)	net.
HBAND	(2)	horizontal bands.
VBAND	(4)	vertical bands.

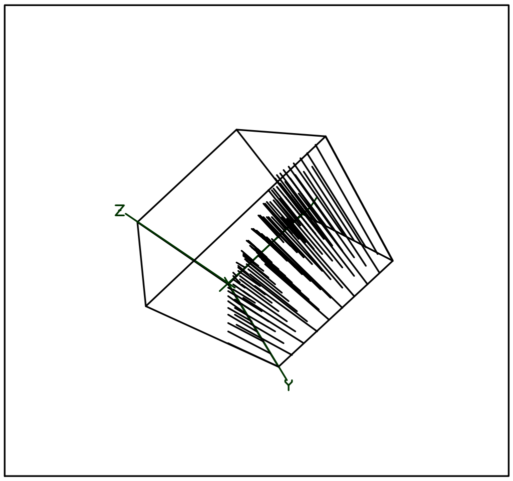


Figure 12.6: Add the coordinate axes.

`ALTERNATE`	(5)	half-sequence.
`MARKER`	(6)	solid line with marker skipping.
`NEEDLES`	(8)	needles.
	(9)	needles and net.
`HIDDEN`	(16)	hidden-line net.
`SEQUENCE`	(32)	sequence.

The next example demonstrates the use of the `SEQUENCE` linetype.

Example 12-2: Draw a perspective view of a 3 dimensional spiral.

The following MLAB commands define parametric functions for the Cartesian coordinates of a spiral, evaluate the coordinates for 400 values of the parameter, and draw the perspective view.

```
* DELETE M,W3
* FCT X(T) = COS(T)/T
* FCT Y(T) = SIN(T)/T
* FCT Z(T) = 3*T
* M COL 1 = X ON 1:20!400
* M COL 2 = Y ON 1:20!400
* M COL 3 = Z on 1:20!400
* DRAW M LT SEQUENCE
* CMD3D("RAISE -.5")
```

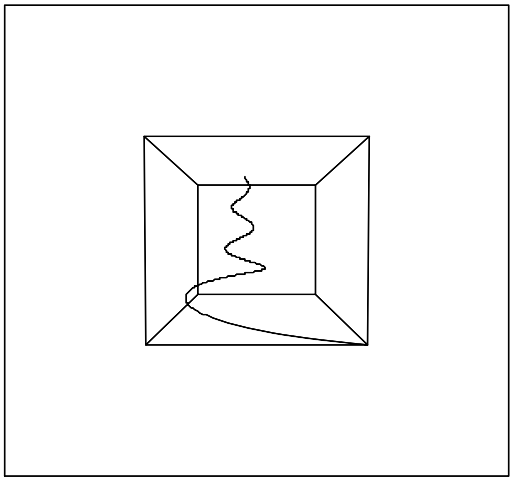


Figure 12.7: Spiral drawn with SEQUENCE linetype.

```
* CMD3D("TRACK")
* CMD3D("BOX")
* VIEW
```

The functions X(T), Y(T), and Z(T) are the parametric functions defining the (x, y, z) coordinates of the spiral. The numerical values of the coordinates are stored in the three column matrix M. The DRAW statement creates the default 3D graphics window with the curve defined by the matrix M—each point connected in sequence to the next with a solid line segment. Note that the CMD3D statements cause the 3D-view control instructions to be executed from the MLAB command prompt before the MLAB 3D command prompt (>>) is printed. The result is shown in Figure 12.7.

The following table defines the pointtypes available for 3D graphs:

NONE	(0)	no pointtype.
DOT3D	(1)	small dot.
CROSS3D	(2)	small 3D-crosses.
"x"	(4)	terminal-generated character x at the smallest size.

The following example demonstrates the DOT3D pointtype.

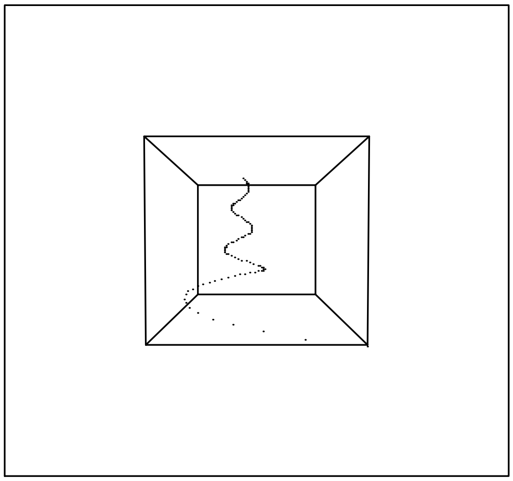


Figure 12.8: Spiral drawn with 3D dot at each point and no connecting lines.

Example 12-3: Draw the spiral in the previous example, sampled at 100 points, with the DOT3D pointtype at each point and do not connect each point with a line segment.

This is done as follows:

```
* DELETE W3
* FCT X(T) = COS(T)/T
* FCT Y(T) = SIN(T)/T
* FCT Z(T) = 3*T
* M COL 1 = X ON 1:20!100
* M COL 2 = Y ON 1:20!100
* M COL 3 = Z on 1:20!100
* DRAW M LT NONE PT DOT3D
* CMD3D("RAISE -.5")
* CMD3D("TRACK")
* CMD3D("BOX")
* VIEW
```

The result is shown in Figure 12.8.

The NET linetype can be used to draw surfaces which are sampled over evenly spaced grids.

Example 12-4: Draw a perspective view of the Hénon-Heiles potential energy surface in the square [-2,2]x[-2,2]. The Hénon-Heiles potential energy function–defined in [Hen] (see bibliography)– is evaluated and drawn with the following MLAB commands:

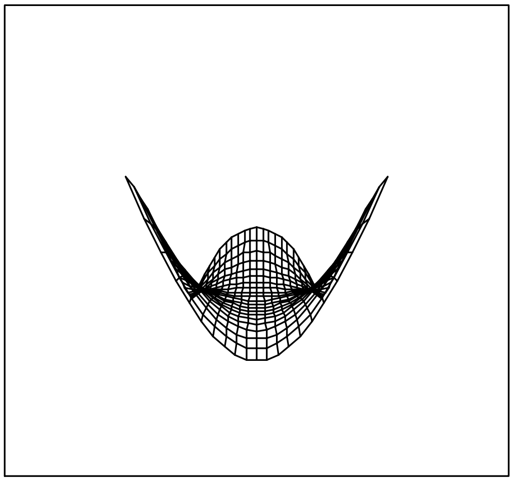


Figure 12.9: Hénon-Heiles potential energy surface with x and y in [-2,2].

```
* RESET
* FCT F(X,Y) = (X^2+Y^2)/2+X^2*Y-Y^3/3
* M = CROSS(-2:2.1,-2:2:.2)
* M COL 3 = F ON M
* DRAW M LT NET
* VIEW
```

This results in Figure 12.9.

Depending on the position of the camera with respect to the surface, near parts of the surface may obscure far parts of the surface. In that case, the HIDDEN linetype should be used instead of the NET linetype.

Example 12-5: Draw the top hat function and the real and imaginary parts of the two dimensional Fourier transform of the top hat function using the HIDDEN linetype in a single window.

```
* RESET
* /* generate points on the top hat surface */
* FCT F(X,Y) = IF (X^2+Y^2 >= 1) THEN 0 ELSE 1
* T = CROSS(-2:2!32,-2:2!32)
* T COL 3 = F ON T
* T COL 4 = 0^^1024
```

```
*
* /* set s to the 2D FFT of t */
* SR = DFT(T)
* SI = SR
*
* /* set si to the imaginary part, sr to the real part of the 2D FFT of t */
* DELETE SI COL 3
* DELETE SR COL 4
* DELETE T COL 4
*
* /* draw the 2d signal, the real part of ft, and the imaginary part of ft */
* DRAW SI IN W1 LT HIDE
* DRAW SR IN W2 LT HIDE
* DRAW T IN W3 LT HIDE
* FRAME 0 TO .5, 0 TO 1 IN W3
* FRAME .5 TO 1, .5 TO 1 IN W2
* FRAME .5 TO 1, 0 TO .5 IN W1
* VIEW
>> WINDOW3D W1
>> RAISE .5
>> TRACK
>> DOLLY .5
>> COLORLIST 0,1
>> WINDOW3D W2
>> RAISE .5
>> TRACK
>> DOLLY .5
>> COLORLIST 0,1
>> WINDOW3D W3
>> RAISE .5
>> TRACK
>> DOLLY .5
```

In this example, three 3D window frames exist in the MLAB window. The `WINDOW3D` statement specifies to which window frame the camera commands will be applied. The resulting picture is shown in Figure 12.10.

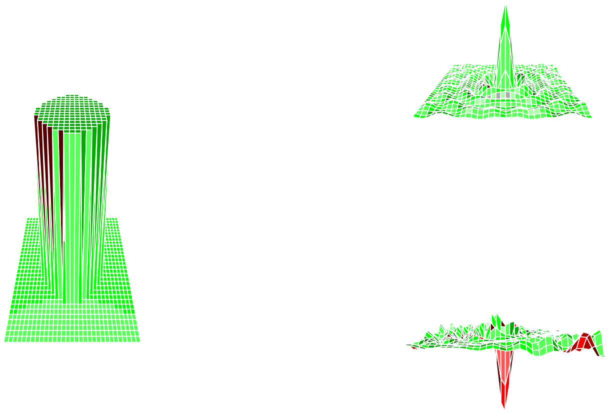


Figure 12.10: The tophat function and the real and imaginary parts of its 2D Fourier transform..

Chapter 13

Surface Drawing with Sweeping

[Note: this chapter is based on the chapter of the same name appearing in the **MLAB Applications Manual** written by Dr. Zhiping You.]

A surface Q in 3-dimensional space may be represented in two parameter form as a function $Q(s,t) = (x(s,t), y(s,t), z(s,t))$. A special case of this is the surface of a function. The parameter form of a function is just $(x, y, z(x, y))$. In this chapter, we will discuss a special case of surfaces in 3 dimensions, *i.e.* tube-like-surfaces that are generated by sweeping a space curve along another central space curve.

Given a space curve $r(t) = (x(t), y(t), z(t))$, at each point, there are three associated directions: the *tangent* direction, the *normal* direction, and the *bi-normal* direction. The unit *tangent* vector–denoted by T, is defined as $T(t) = r'(t)/\|r'(t)\|$; the unit *normal* vector–denoted by N, is defined as $N(t) = T'(t)/\|T'(t)\|$, and the unit *bi-normal* vector–denoted by B, is defined as $B(t) = T(t) \times N(t)$ (where $\times$ means cross product).

Given $r(t), T(t), N(t)$ and $B(t)$, a tube-like surface can be expressed as:

$$Q(s,t) = r(t) + d \cdot (\cos(s)N(t) - \sin(s)B(t))$$

where d is a parameter corresponding to the radius of rotation. In general, d can be a function of t. For fixed t, when s runs from 0 to 2π, we have a circle around the point $r(t)$ in the N, B plane. As we change t, this circle moves along the space curve r, thereby generating a tube-like surface along r.

Example 13-1: A simple example is the torus, where r is a circle, $r(t) = (\cos(t), \sin(t), 0)$. In this case, $T(t) = (-\sin(t), \cos(t), 0)$, $N(t) = (-\cos(t), -\sin(t), 0)$ and $B(t) = (0, 0, 1)$. Thus, the parametric form of the torus is:

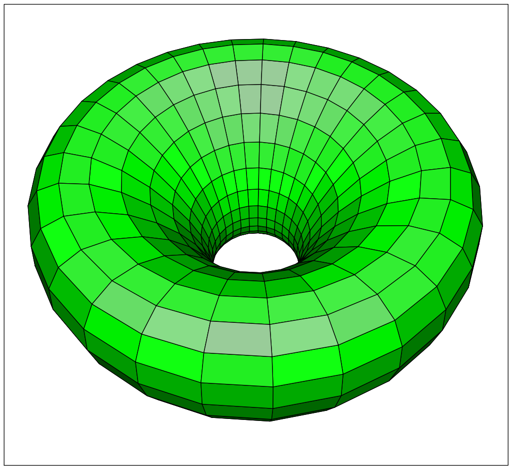


Figure 13.1: A torus with central curve r(t) = (cos(t), sin(t), 0), and radius 0.5.

$$\begin{aligned} x(s,t) &= \cos(t) - d \cdot \cos(s)\cos(t) \\ y(s,t) &= \sin(t) - d \cdot \cos(s)\sin(t) \\ z(s,t) &= d \end{aligned}$$

We construct and draw a torus in MLAB by defining and evaluating the x, y and z functions on a regular grid of (s,t)-points. Using the functions x, y and z, above, this is done as follows:

```
* FCT X(S,T) = COS(T)-D*COS(S)*COS(T)
* FCT Y(S,T) = SIN(T)-D*COS(S)*SIN(T)
* FCT Z(T) = D
* GRID = CROSS((0:(2*PI)!30, 0:(2*PI)!30)
* D = 0.5
* M = (X ON GRID) &' (Y ON GRID) &' (Z ON GRID)
* DRAW M LT HIDE
* CMD3D("raise 2.5")
* CMD3D("track")
* CMD3D("dolly 2")
* VIEW
```

The resulting figure is shown in Figure 13.1.
Note, in this case, that the MLAB `DRAW` statement is able to correctly discover the underlying grid structure in the matrix `M` automatically!

A general do-file for drawing tube-like surfaces named `TUBE.DO` follows. If one defines functions for the x, y, and z components of the central curve, a grid of parameter values, and a function *d* for the radius of the sweeping curve, then executing the do-file `TUBE.DO` computes the required vector components and draws the curve.

```
/* TUBE.DO */
/* generate a tube of radius D(T) around the curve: R(T) = (X(T),Y(T),Z(T)) */
/* The central space curve component functions X(T), Y(T) and Z(T) must be */
/* predefined. The radius function D(T) must also be predefined, and */
/* finally the (S,T) grid matrix called GRID must also be predefined. */

/* Construct unit tangent vector T(t) */
/* nd(t) is the norm of the derivative */
FCT ND(T) = ((X’T(T))^2+(Y’T(T))^2+(Z’T(T))^2)^0.5
/* A(T), B(T) and C(T) are the component of T(T) */
FCT A(T) = X’T(T)/ND(T)
FCT B(T) = Y’T(T)/ND(T)
FCT C(T) = Z’T(T)/ND(T)

/* Construct unit normal vector N(T) */
/* NDUT(T): norm of the derivative of the unit tangent vector */
FCT NDUT(T) = ((A’T(T))^2+(B’T(T))^2+(C’T(T))^2)^0.5
/* AN(T), BN(T) and CN(T) are the components of N(T) */
FCT AN(T) = A’T(T)/NDUT(T)
FCT BN(T) = B’T(T)/NDUT(T)
FCT CN(T) = C’T(T)/NDUT(T)

/* Construct the unit binormal vector B(T) */
/* AB(T), BB(T), and CB(T) are the components of B(T) */
FCT AB(T) = B(T)*CN(T)-C(T)*BN(T)
FCT BB(T) = C(T)*AN(T)-A(T)*CN(T)
FCT CB(T) = A(T)*BN(T)-B(T)*AN(T)

/* Now, construct the space tube */
/* M(S,T) = R(T) + D*(COS(S)*N(T) + SIN(S)*B(T)) */
/* D is the radius of the tube along the central curve */
/* MX(S,T), MY(S,T), MZ(S,T) are the components of M(S,T) */
FCT MX(S,T) = X(T) + D(T)*(COS(S)*AN(T)+SIN(S)*AB(T))
FCT MY(S,T) = Y(T) + D(T)*(COS(S)*BN(T)+SIN(S)*BB(T))
FCT MZ(S,T) = Z(T) + D(T)*(COS(S)*CN(T)+SIN(S)*CB(T))

/* now generate the surface */
MM = (MX ON GRID) &’ (MY ON GRID) &’ (MZ ON GRID)
DRAW MM LINETYPE NET
VIEW
/* end of tube.do */
```

The next example demonstrates how `TUBE.DO` can be used to generate the previous figure of a torus.

Example 13-2: Use `TUBE.DO` to draw a torus by defining functions `X(T)`, `Y(T)`, and `Z(T)` which determine the central curve as the unit circle in the x, y plane, the 2-column matrix `GRID`, and the function `D(T)`, which in this case, is a constant.

```
* RESET
* /* the central line curve is (cos t, sin t, 0) */
* /* define the center space curve r(t) = (x(t), y(t), z(t)) */
* FCT X(T) = COS(T)
* FCT Y(T) = SIN(T)
* FCT Z(T) = 0
* GRID = CROSS((0:(2*PI)!30),(0:(2*PI)!30))
* FCT D(T) = 1
* DO TUBE.DO
* CMD3D("raise 2.5")
* CMD3D("track")
* CMD3D("dolly 2")
* VIEW
```

One can generate more interesting surfaces with other central curves. Here are two examples with two different central curves.

Example 13-3: Use the MLAB do-file `TUBE.DO` to draw a torus with a sweeping circle of radius that varies between 0.25 to 0.75.

```
* RESET
* FCT X(T) = COS(T)
* FCT Y(T) = SIN(T)
* FCT Z(T) = 0
* GRID = CROSS(0:(2*PI)!30,0:(2*PI)!30)
* FCT D(T) = .5+.25*SIN(3*T)
* DO TUBE.DO
* CMD3D("raise 2.5")
* CMD3D("track")
* CMD3D("dolly 2")
* VIEW
```

The Figure 13.2 is the result.

Example 13-4: Use the MLAB do-file `TUBE.DO` to draw a descending spiral central curve with a sweeping circle that increases in radius.

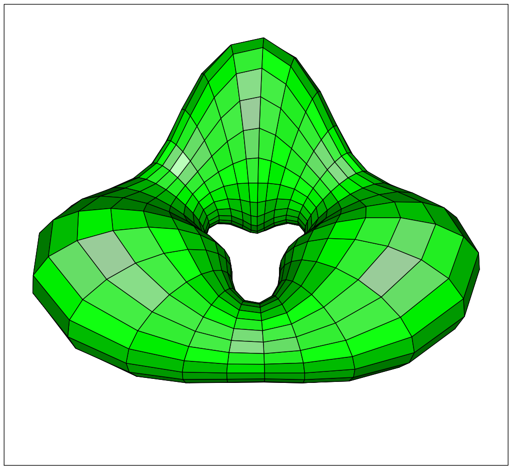


Figure 13.2: Central curve r(t) = (cos(t), sin(t), 0) with a variable radius.

```
* RESET
* /* draw descending central curve and ascending radius */
* FCT X(T) = .12*T*COS(T)
* FCT Y(T) = .12*T*SIN(T)
* FCT Z(T) = -.18*T
* GRID = CROSS(0:(2*PI)!16,0:(6*PI)!60)
* FCT D(T) = .1*T
* DO TUBE.DO
* CMD3D("raise 1")
* CMD3D("track")
* CMD3d("dolly 1")
* VIEW
```

The resulting picture is shown in Figure 13.3.

Example 13-5: Use the MLAB do-file `TUBE.DO` to draw a view of a 3-dimensional trefoil, i.e. a self-crossing central curve with constant radius sweeping circle.

```
* RESET
* /* draw 3D trefoil = self crossing central curve with constant radius */
* FCT X(T) = COS(3*T)*COS(T)+P*SIN(2*(T-Q))
* FCT Y(T) = COS(3*T)*SIN(T)+P*COS(2*(T-Q))
* FCT Z(T) = SIN(6*T)
* P = 0.2;     /* amplitude of oscillation */
* Q = PI/4;    /* phase shift of oscillation */
```

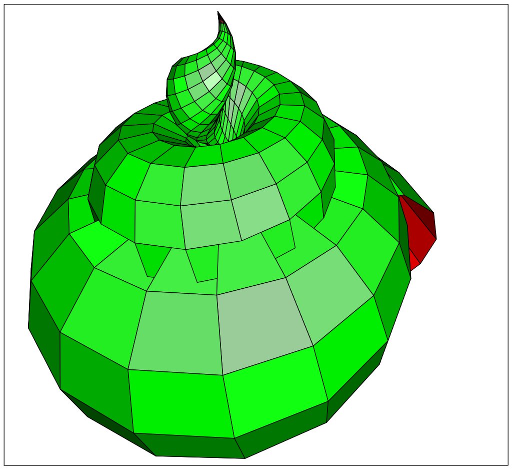


Figure 13.3: Descending central curve and increasing radius.

```
* GRID = CROSS(0:(2*PI)!16,0:PI!60)
* FCT D(T) = 0.1;  /* radius of tube */
* DO TUBE.DO
* CMD3D("xrotate 90")
* CMD3D("xscale 1.5")
* CMD3D("yscale 1.5")
* CMD3D("raise .1")
* VIEW
```

The resulting figure is shown on the cover of this book and in Figure 13.4.

Generally speaking, our sweeping curve does not need to be a circle, it can be any 2-dimensional curve $c(s,t) = (p(s,t), q(s,t))$. Thus, instead of having:

$$Q(s,t) = r(t) + d \cdot (\cos(s)N(t) - \sin(s)B(t))$$

as the sweeping surface, we can have:

$$Q(s,t) = r(t) + p(s,t) \cdot N(t) - q(s,t) \cdot B(t)$$

In particular, when the curve c is a straight line segment, the sweeping surface becomes a band. In the next two examples, two such bands are demonstrated: one with a fixed line segment, and one with a rotating line segment. The latter surface is the famous *Mobius* band.

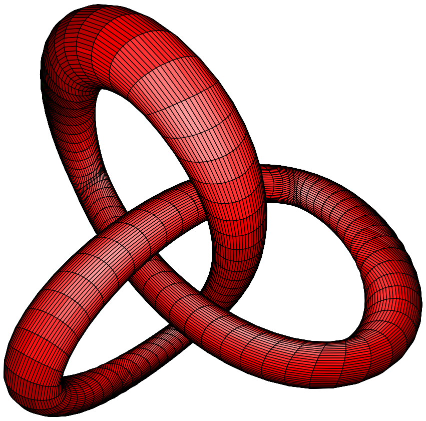


Figure 13.4: 3D Trefoil: a self crossing central curve and constant radius.

Example 13-6: Use the MLAB do-file `TUBE.DO` to draw a regular band with fixed radius.

```
* RESET
* /* draw regular band with fixed radius */
* FCT X(T) = COS(T)
* FCT Y(T) = SIN(T)
* FCT Z(T) = 0
* GRID = CROSS((PI/2):(3*PI/2)!2,0:(2*PI)!30)
* FCT D(T) = .5
* DO TUBE.DO
* CMD3D("raise 1")
* CMD3D("track")
* CMD3D("dolly .5")
* VIEW
```

The regular band is shown in Figure 13.5.

Example 13-7: Draw a Mobius band.

```
* RESET
* /* draw a Mobius band */
* FCT X(S,T) = (2-T*SIN(S/2))*SIN(S)
* FCT Y(S,T) = (2-T*SIN(S/2))*COS(S)
* FCT Z(S,T) = T*COS(S/2)
```

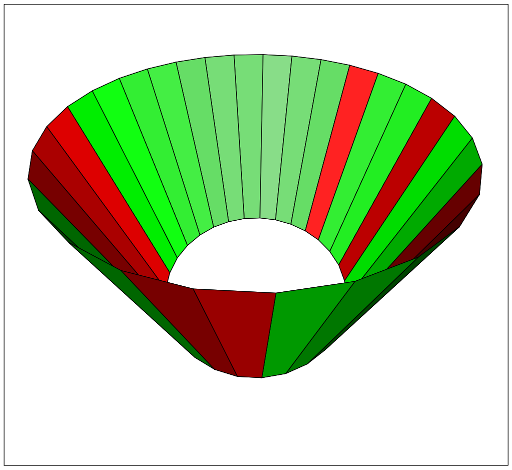


Figure 13.5: Regular band with fixed radius and direction.

```
* M = CROSS(0:(2*PI)!30,-1:1!4)
* M = (X ON M) &' (Y ON M) &' (Z ON M)
* DRAW M LT HIDDEN
* CMD3D("surface zrotate -90")
* CMD3D("raise 2")
* CMD3D("track")
* CMD3D("dolly 1.5")
* VIEW
```

The resulting Mobius band is shown in Figure 13.6.

One can easily see that if we adjust the radius, direction, number of points on each circle, and other parameters, we can generate many interesting surfaces.

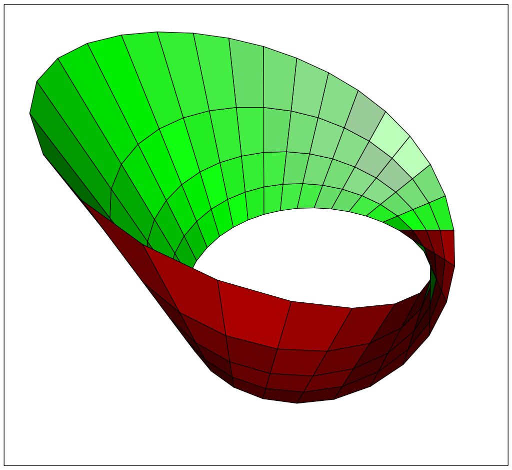


Figure 13.6: Mobius band.

Appendix A

Font Tables

The tables on the following pages show the printable characters in each of the thirty four Hershey fonts (specified by font numbers equal to postitive integers $1, 2, ..., 34$) and in each of the thirteen TrueType fonts (specified by font numbers equal to the negative integers $-2, -3, ..., -14$).

Hershey fonts are supported on all systems.

TrueType fonts are supported only on Microsoft Windows and Apple Macintosh displays and PostScript printer devices. TrueType fonts are not supported on Unix and Linux with X-Windows displays or Hewlett-Packard Printer Control Language (PCL) printer devices. The following list shows the MLAB font numbers and the names of the corresponding TrueType fonts.

Font number	Microsoft Windows and Postscript	Macintosh OS9/X
-2	Courier New	Courier
-3	Courier New Bold	Courier Bold
-4	Courier New Italic	Courier Oblique
-5	Courier New Bold Italic	Courier Bold Oblique
-6	Times New Roman	Times Roman
-7	Times New Roman Bold	Times Bold
-8	Times New Roman Italic	Times Italic
-9	Times New Roman Bold Italic	Times Bold Italic
-10	Arial	Helvetica
-11	Arial Bold	Helvetica Bold
-12	Arial Italic	Helvetica Oblique
-13	Arial Bold Italic	Helvetica Bold Oblique
-14	Symbol	Symbol

ASCII characters 0 to 31 do not appear for Hershey fonts numbered 6 to 34 or for any of the TrueType fonts because they do not correspond to displayable characters.

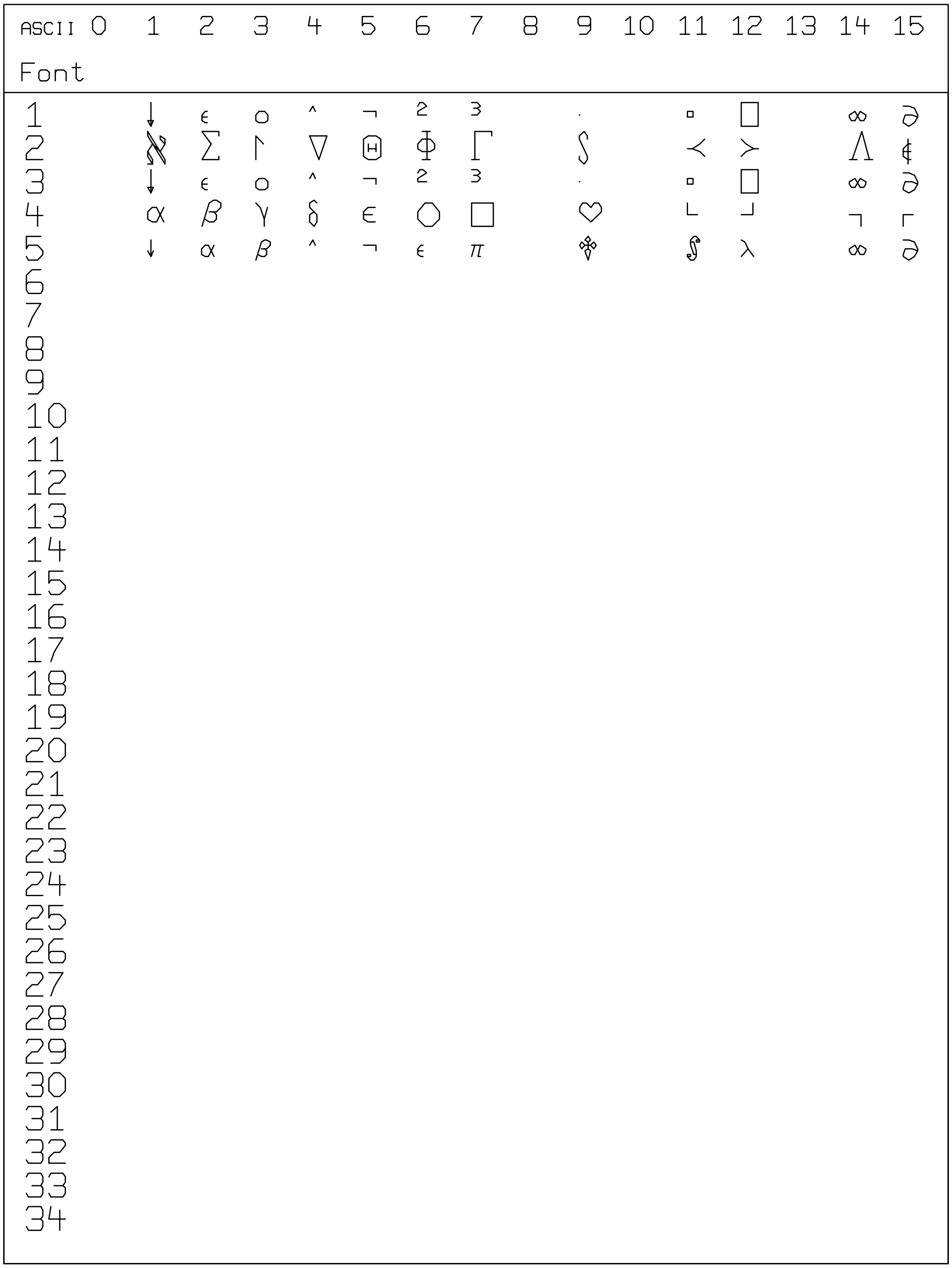


Figure A.1: ASCII character numbers 0 to 15 in Hershey fonts 1 to 5.

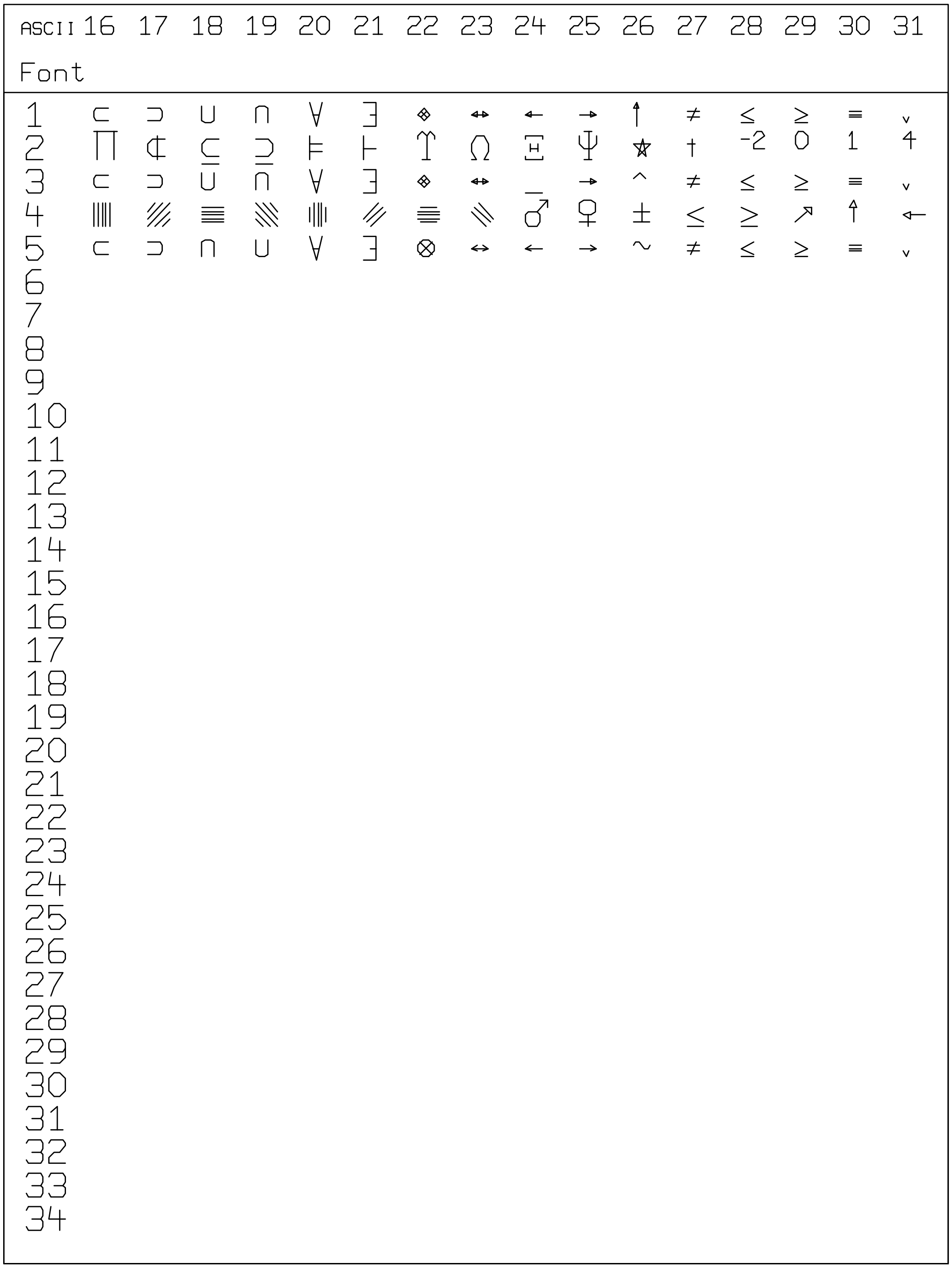

ASCII 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Font
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

Figure A.2: ASCII characters 16 to 31 in Hershey fonts 1 to 5.

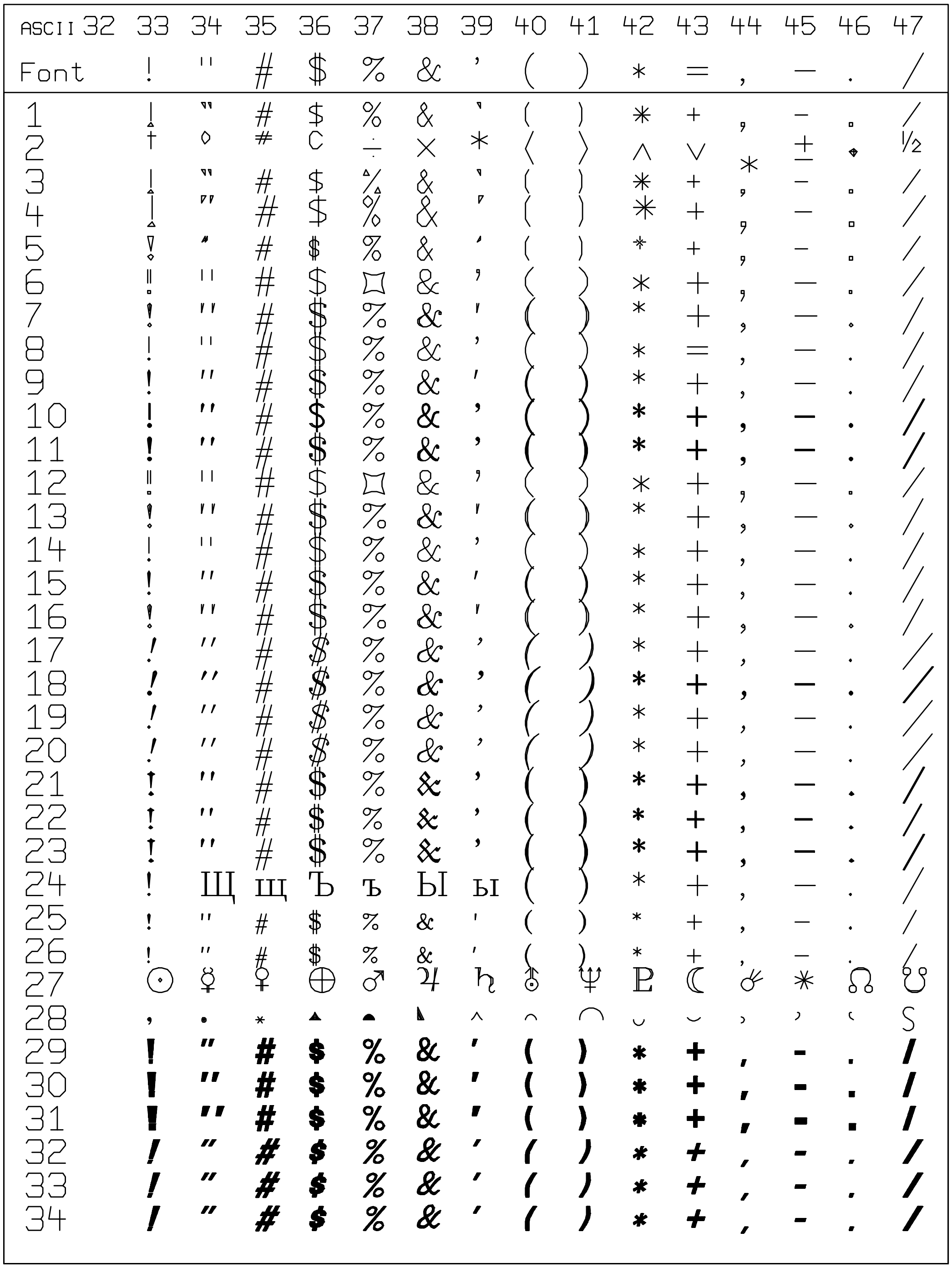

ASCII 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
Font ! " # $ % & ' ( ) * = , — . /
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

Figure A.3: ASCII characters 32 to 47 in Hershey fonts 1 to 34.

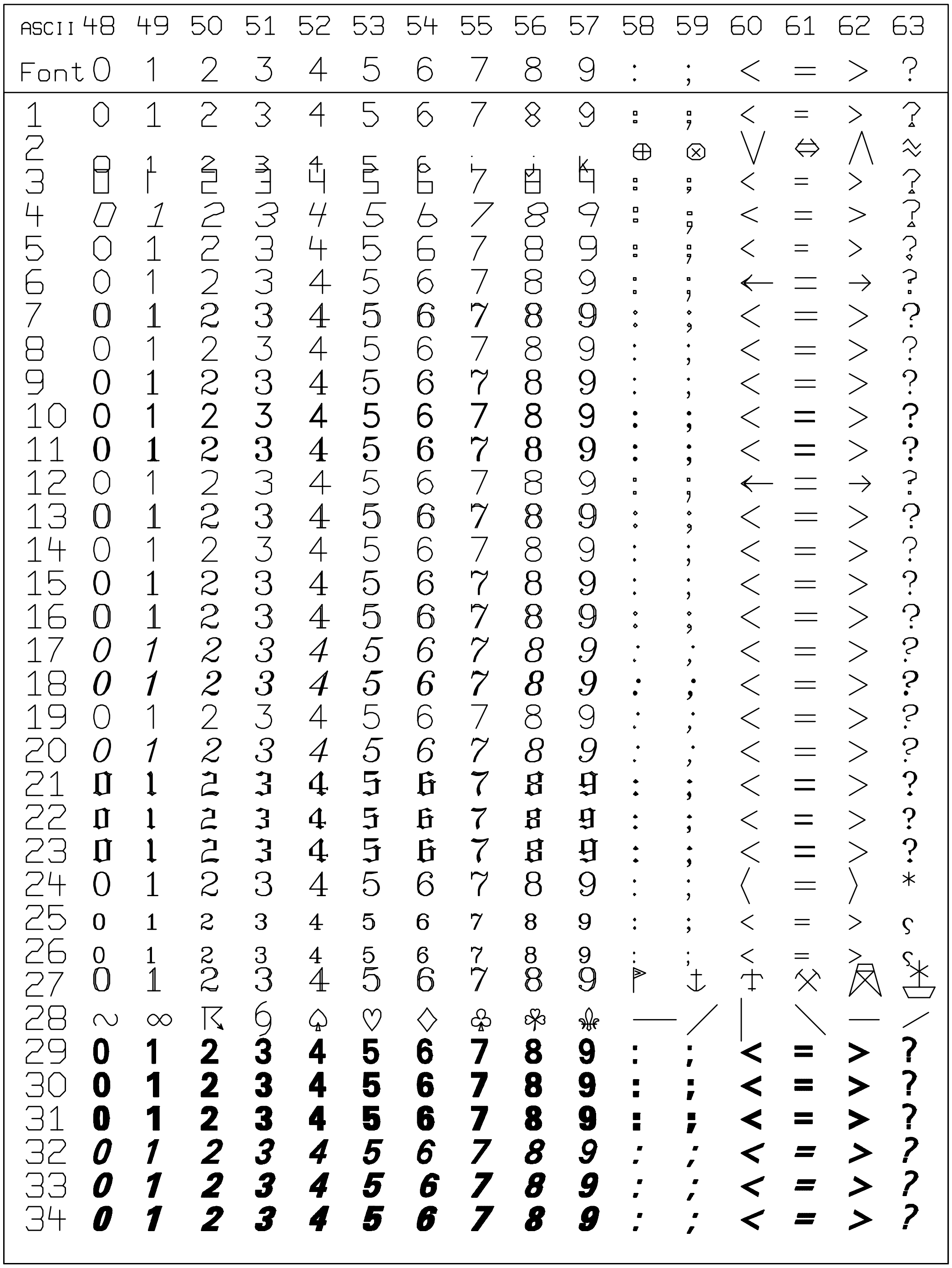


Figure A.4: ASCII characters 48 to 63 in Hershey fonts 1 to 34.

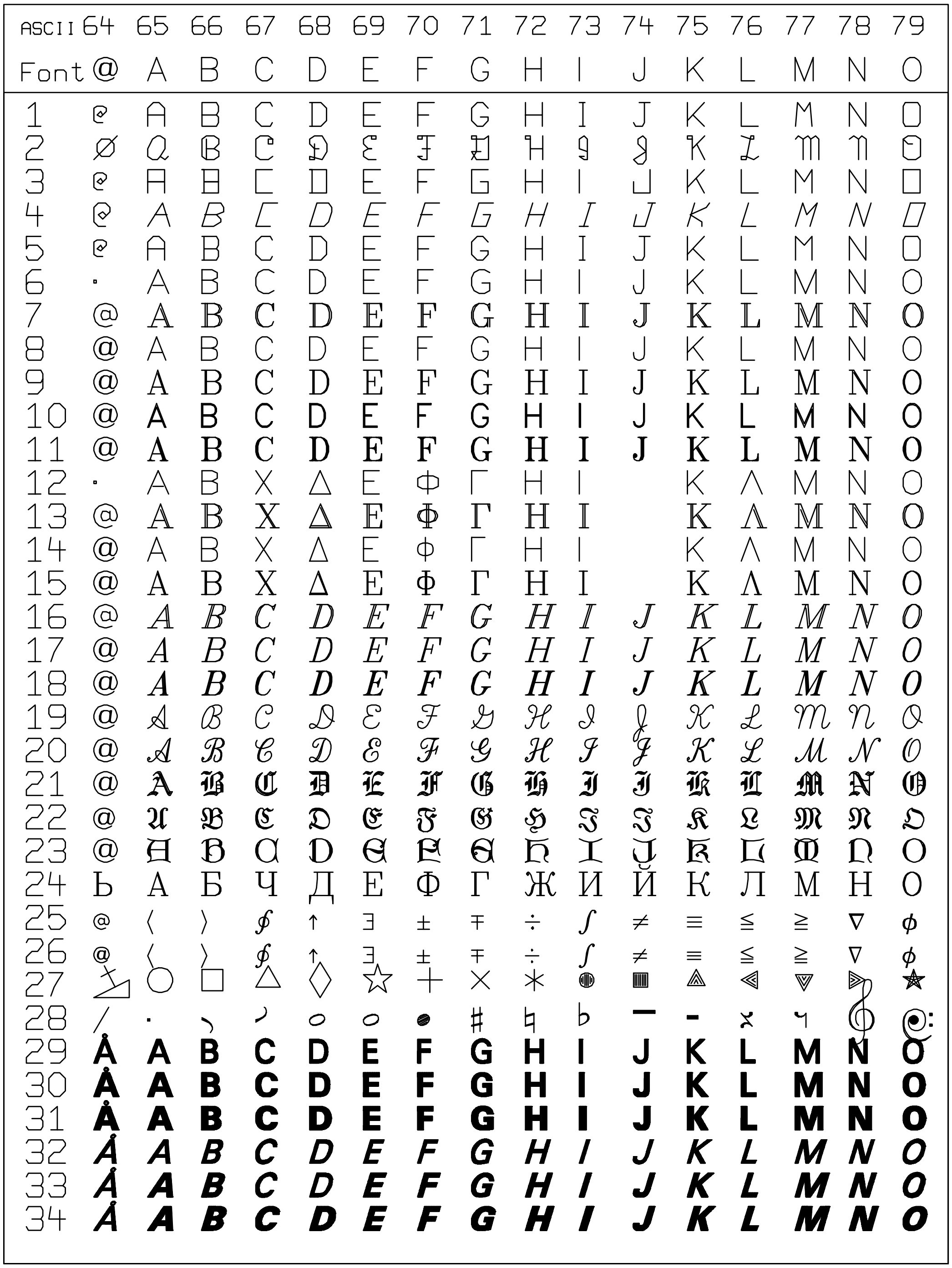

ASCII 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
Font @ A B C D E F G H I J K L M N O
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

Figure A.5: ASCII characters 64 to 79 in Hershey fonts 1 to 34.

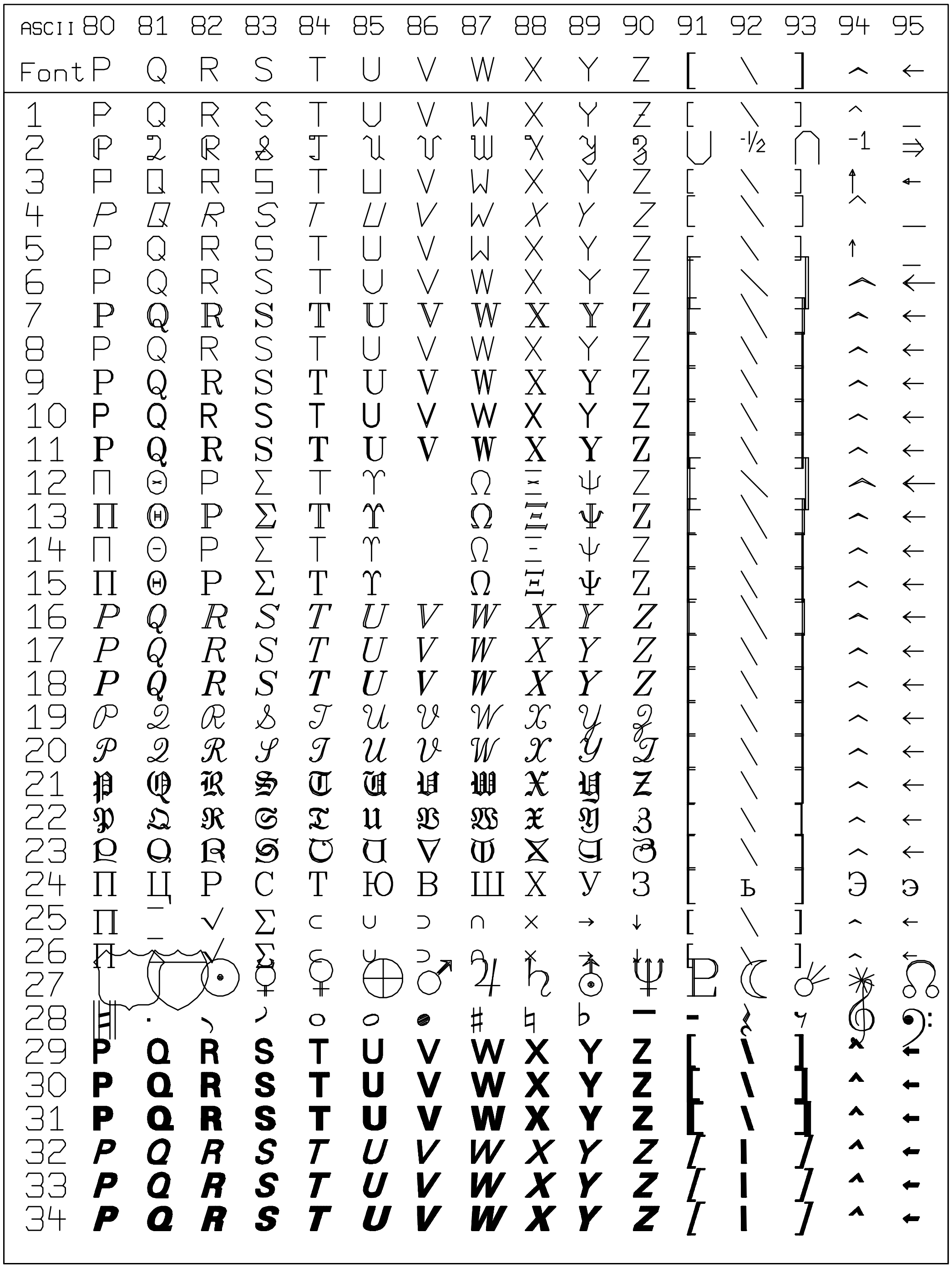

ASCII 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
Font P Q R S T U V W X Y Z [\ ] ^ ←
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34

Figure A.6: ASCII characters 80 to 95 in Hershey fonts 1 to 34.

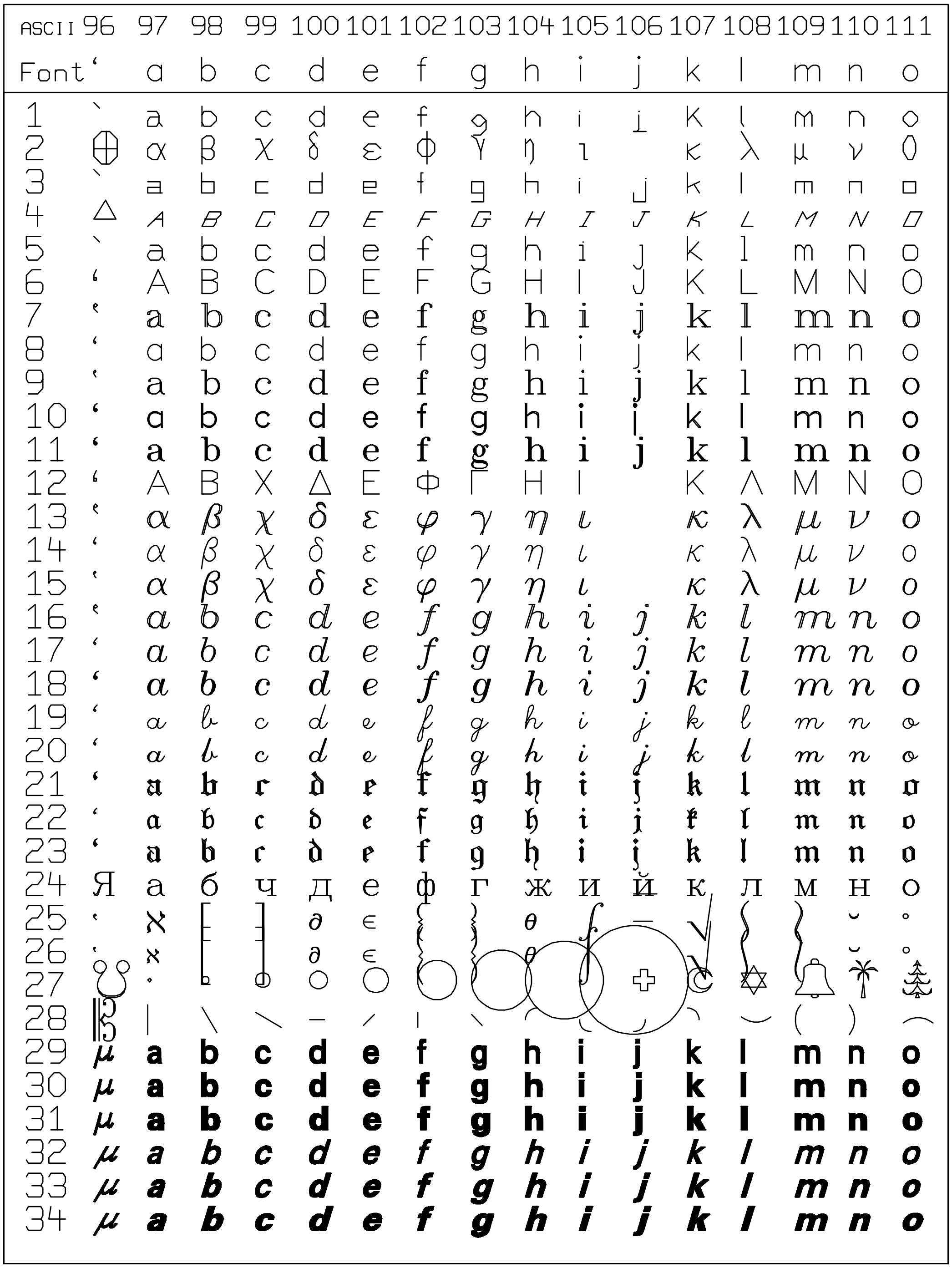

ASCII 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
Font ‘ a b c d e f g h i j k l m n o
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

Figure A.7: ASCII characters 96 to 111 in Hershey fonts 1 to 34.

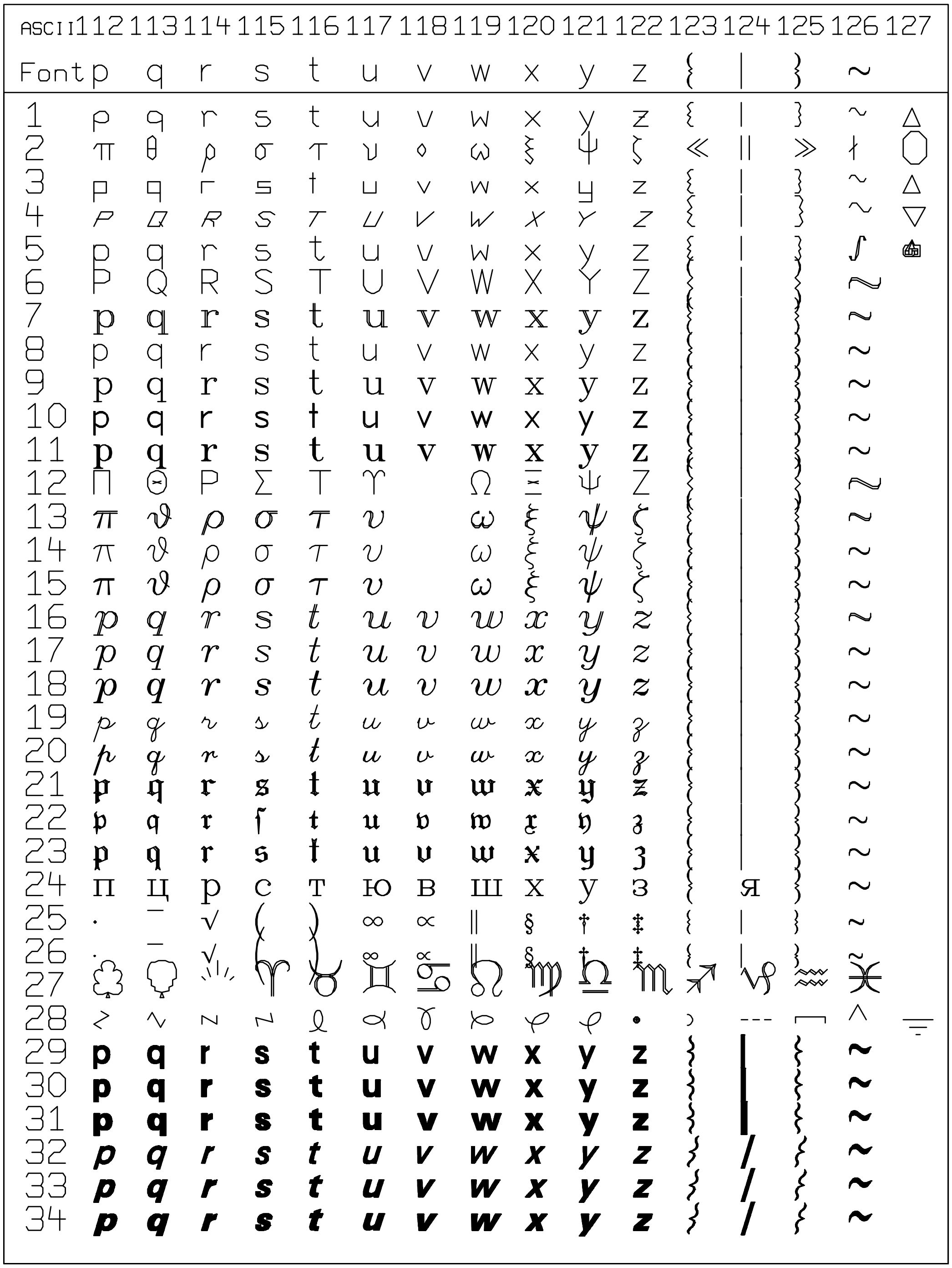
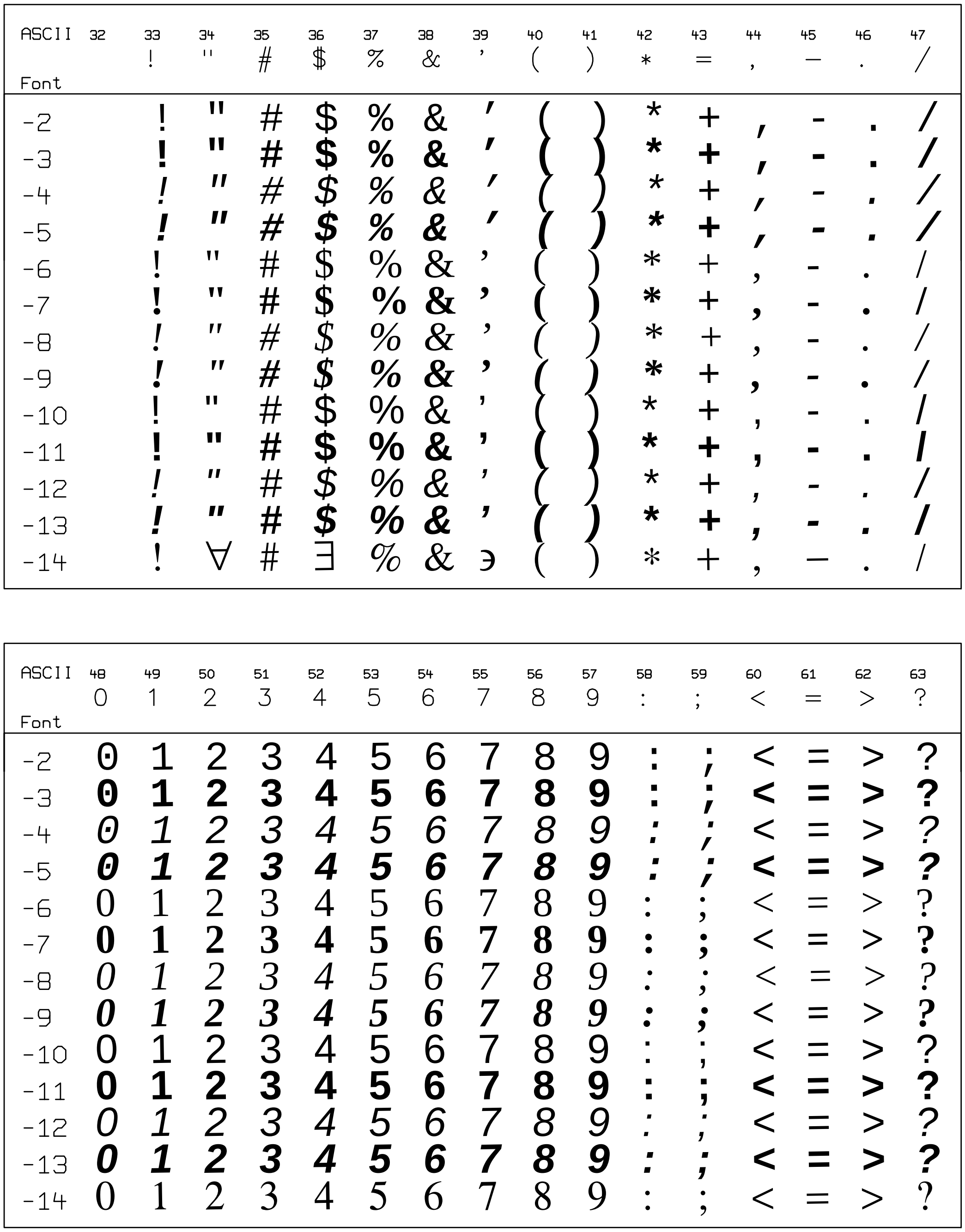

ASCII 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
Font p q r s t u v w x y z { | } ~
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34

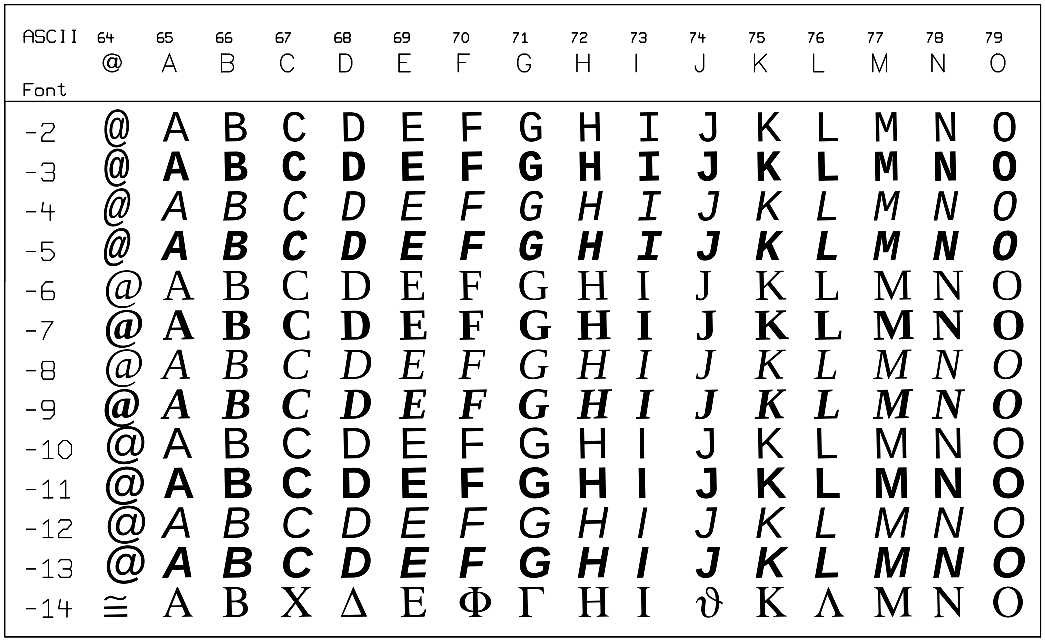
Figure A.8: ASCII characters 112 to 127 in Hershey fonts 1 to 34.



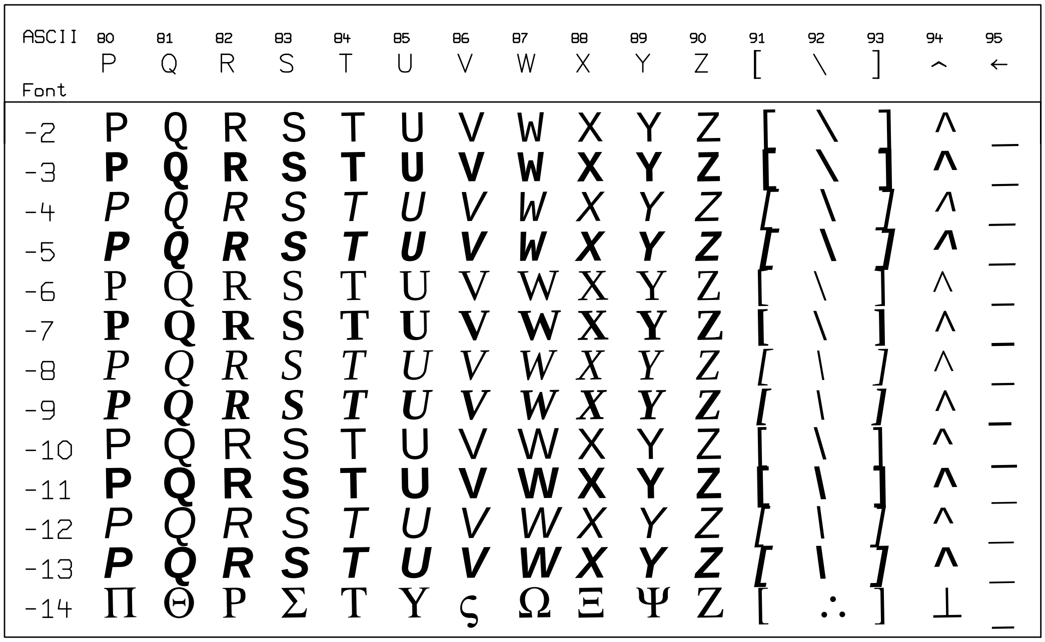
ASCII	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Font		!	''	#	$	%	&	'	(	)	*	=	,	—	.	/
-2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-3		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-4		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-5		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-6		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-7		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-8		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-9		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-10		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-11		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-12		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-13		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
-14		!	∀	#	∃	%	&	∋	(	)	∗	+	,	−	.	/

ASCII	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Font	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-2	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-4	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-5	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-6	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-7	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-8	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-9	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-10	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-11	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-12	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-13	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
-14	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?

Figure A.9: ASCII characters 32 to 63 in TrueType fonts -2 to -14.

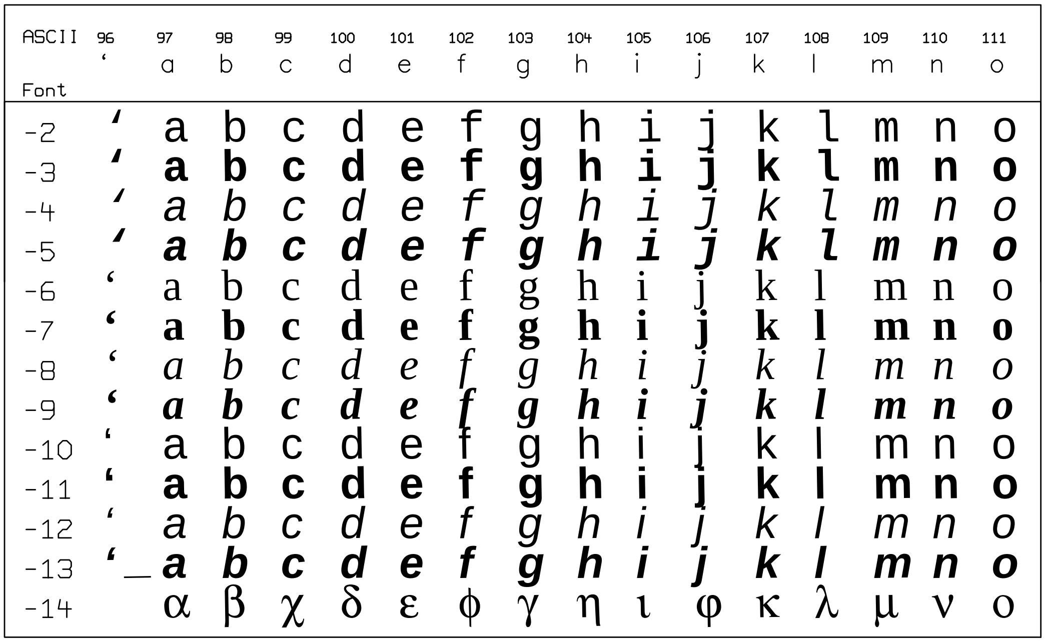


ASCII	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Font																
-2	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-3	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-5	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-6	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-7	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-8	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-9	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-10	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-11	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-12	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-13	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
-14	≅	Α	Β	Χ	Δ	Ε	Φ	Γ	Η	Ι	ϑ	Κ	Λ	Μ	Ν	Ο

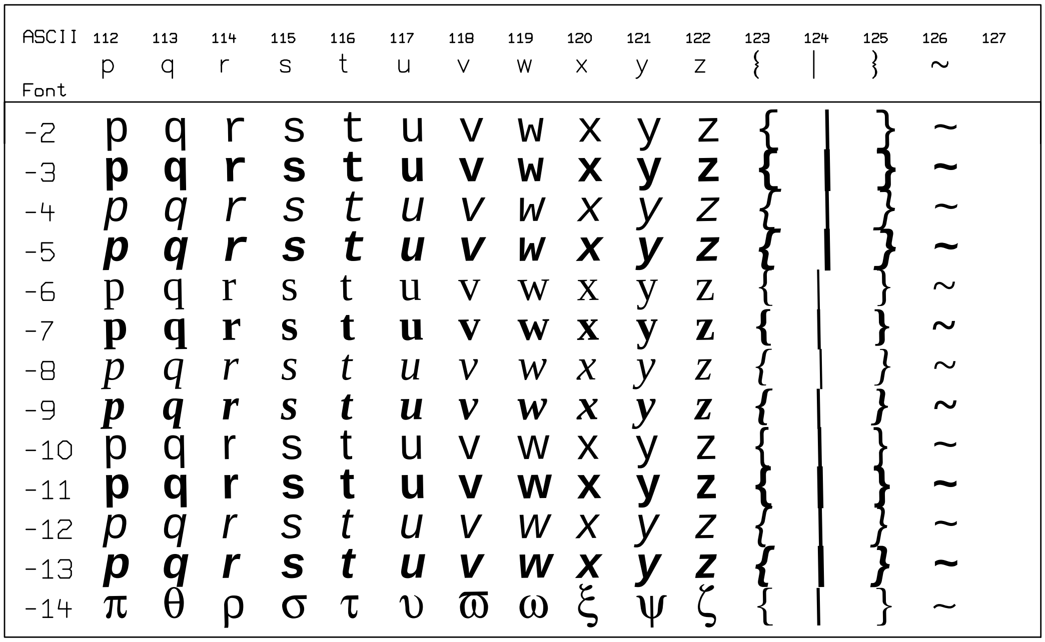


ASCII	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	←
Font																
-2	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-3	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-4	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-6	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-7	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-8	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-9	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-10	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-11	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-12	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-13	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
-14	Π	Θ	Ρ	Σ	Τ	Υ	ς	Ω	Ξ	Ψ	Ζ	[	∴	]	⊥	_

Figure A.10: ASCII characters 64 to 95 in TrueType fonts -2 to -14.



ASCII	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
Font	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-2	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-3	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-4	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-5	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-7	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-8	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-9	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-10	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-11	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-12	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-13	‘_	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
-14		α	β	χ	δ	ε	ϕ	γ	η	ι	φ	κ	λ	μ	ν	ο



ASCII	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
Font	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-2	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-3	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-4	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-5	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-6	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-8	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-9	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-10	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-11	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-12	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-13	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
-14	π	θ	ρ	σ	τ	υ	ϖ	ω	ξ	ψ	ζ	{	|	}	~	

Figure A.11: ASCII characters 96 to 127 in TrueType fonts -2 to -14.

Appendix B

Syntax for MLAB 2D graphics statements

W is a window name. C is a curve name. S is a string. T is a title name. M is a curve matrix. id1,id2,... denote windows (if id is W) and curves (if id is C). Bracketed items ([ ]) denote clauses that can be omitted. <A|B> means A or B. Color numbers, denoted Q may be a predefined color name, such as YELLOW, RED, BLUE; or the color number returned by calling COLORN(I), with I equal to an integer in the inverval [1:16]; or the color returned by calling COLORX(R,G,B) where R,G, and B are red, green, and blue component values, each in the interval [0:1].

Commands to specify display and plotter devices:

```
DISPLAY = <displaytype>
```

where <displaytype> is the quoted string MSWDEV for Microsoft Windows systems, XDEV for Linux systems with an X-Windows display, MACD for Apple Macintosh systems, EGAC for DOS systems with 640x350 color enhanced graphics adapter, EGAM for DOS systems with 640x350 monochrome enhanced graphics adapter, VGAC for DOS systems with 640x480 color video graphics array, VGAM for DOS systems with 640x480 monochrome video graphics array, or DPS for NeXT systems with PostScript display device.

```
PLOTDEV = <plottertype>
```

where <plottertype> is the quoted string PSL for PostScript Laser printers in landscape mode, PSP for PostScript Laser printers in portrait mode, PSCL for PostScript Color Laser printers in landscape mode, PSCP for PostScript Color Laser printers in portrait mode, LJ2L for Hewlett Packard LaserJet printers using Printer Control Language (PCL) in landscape mode, or LJ2P for Hewlett Packard LaserJet printers using Printer Control Language (PCL) in portrait mode. (“landscape mode” means the output is printed on an 8.5 x 11 inch page with the larger dimension oriented horizontally; "portrait mode" means the output is printed on an 8.5 x 11 inch page with the larger dimension oriented vertically.) The default plotter type is `PSL`.

Commands to create and modify windows, images, frames, image boxes, and frame boxes:

`WINDOW [A TO B, C TO D] [ADJUST <P|(P,Q,R,S)>] [IN W]`

where `A TO B` defines the range of x-coordinates and `C TO D` defines the range of y-coordinates to be shown in the window; `P,Q,R,` and `S` are `WFIX`, `WSHRINK`, `WEXPAND`, `WFLOAT`, `WSLACK`, `WNICE`, or `WMATCH`. The default window is 0 to 10 in both x and y. The default `ADJUST` type is `WNICE`.

`IMAGE [A TO B, C TO D [UNITS]] [COLOR Q] [IN W]`

Expressed in frame fraction units, the default image is .175 to .925 in x and .125 to .875 in y. `UNITS` may be `INCHES,FRACT,FINCHES,` or `FFRACT`; the default is `FFRACT`. The `COLOR Q` clause determines the background color of the image region. The default image background color is blue on displays, white on plots printed on white paper.

`FRAME [A TO B, C TO D [UNITS]] [COLOR Q] [PRIORITY P] [IN W]`

Expressed in `FRACT` units, the default frame is 0 to 1 in x and 0 to 1 in y. `UNITS` may be `INCHES` or `FRACT`; the default is `FRACT`. The default `PRIORITY` is 0. The `COLOR Q` clause determines the background color of the frame. The default background frame region color is purple on displays, white on plots printed on white paper.

`[NO] <IMAGEGBOX|FRAMEBOX> [COLOR Q] [IN W]`

The `COLOR Q` clause determines the color of the image and frame outline.

Commands to create and modify curves:

`DRAW <C|[C =]M> <optional clauses>`

where `<optional clauses>` include:

`LINETYPE <A|(L1,L2,L3,L4,L5,L6)>`

`POINTTYPE A`

`PTFONT A`

where `A` is an integer in [1:34] that selects a Hershey font, or an integer in [-2:-14] that selects a TrueType font.

`PTSIZE <A|M> [UNITS]`

where `UNITS` may be `FFRACT, INCHES, IFRACT, FRACT`, or `WORLD`; the default is `FFRACT`.

`LABEL M`

`OFFSET (A,B) [UNITS]`
: where `UNITS` may be `FFRACT, INCHES, FRACT, IFRACT,` or `WORLD`; the default is `FFRACT`.

`PLACE (A,B)`
: where `A` may be `LEFT, CENTER,` or `RIGHT` and `B` may be `TOP, CENTER,` or `BOTTOM`.

`LABELFONT A`
: where `A` is an integer in [1:34] that selects a Hershey font, or an integer in [-2:-14] that selects a TrueType font.

`LABELSIZE A [UNITS]`
: where `UNITS` may be `FFRACT, IFRACT, INCHES, FRACT,` or `WORLD`. The default is 0.015 `FFRACT`.

`ANGLE A`
: where angle `A` is in right-hand screw degree units.

`FORMAT (H,K,A,B,P,Q)`

`COLOR Q`
: The `COLOR Q` clause determines the color of the curve. The default curve color is white on displays, black on plots.

`IN W`

Commands to create and modify axes:

`<AXIS|XAXIS|YAXIS> <A|M|A = M|A1 = A2> (optional clauses)`
where `optional clauses` include:

`LABEL M`

`OFFSET(X,Y) [UNITS]`
: where `UNITS` may be `FFRACT, INCHES, IFRACT, FRACT,` or `WORLD`; the default is `FFRACT`.

`PLACE(A,B)`
: where `A` may be `LEFT, CENTER,` or `RIGHT` and `B` may be `TOP, CENTER,` or `BOTTOM`.

`LABELFONT X`
: where `X` is an integer in [1:34] that selects a Hershey font, or an integer in [-2:-14] that selects a TrueType font.

`LABELSIZE A [UNITS]`
: where `UNITS` may be `FFRACT, INCHES, IFRACT, FRACT,` or `WORLD`; the default is `FFRACT`.

`FORMAT(H,K,A,B,P,Q)`

COLOR Q

The COLOR Q clause determines the color of the axis lines and tic marks. The default color is white on displays, black on plots.

LINETYPE <A|(L1,L2,L3,L4,L5,L6,L7)>

POINTTYPE A

PTFONT A

where A is an integer in [1:34] that selects a Hershey font, or an integer in [-2:-14] that selects a TrueType font.

PTSIZE A [UNITS]

where UNITS may be FFRACT, INCHES, IFRACT, FRACT, or WORLD; the default is FFRACT.

IN W1

Commands to create and modify titles:

[TOP|LEFT|RIGHT|BOTTOM] TITLE <T = S|T|S|T1 = T2> (optional clauses)

where (optional clauses) include

AT (X,Y) [UNITS] where UNITS may be FFRACT, INCHES, IFRACT, FRACT, FINCHES, or WORLD units; the default is FFRACT.

SIZE A [UNITS]

where UNITS may be FFRACT, INCHES, IFRACT, FRACT, FINCHES, or WORLD units; the default is FFRACT.

PLACE (A,B)

where A may be LEFT, CENTER, or RIGHT and B may be TOP, CENTER, or BOTTOM.

SHIFT F

ANGLE B

where angle B is in right-hand screw degree units.

FONT A

where A is an integer in [1:34] that selects a Hershey font, or an integer in [-2:-14] that selects a TrueType font.

COLOR Q

The COLOR Q clause determines the color of the text. The default is white on displays, black on plots.

IN W1

Commands to delete, blank, and unblank windows, axes, curves, and titles:

`DELETE <W|C|T>`

deletes the window, curve, and/or title from memory and display.

`BLANK <W|C|T>`

hides the window, curve, and/or title from display.

`UNBLANK <W|C|T>`

shows the previously hidden window, curve, and/or title.

Commands to view windows and unview (i.e. remove window) from the display screen:

```
VIEW [W1,...]
```

```
UNVIEW
```

Commands to create plot files on disk and print plot files on printer:

```
PLOT [W1,...]  [IN F]
```

```
PRINT FILE F
```

Commands to get information about windows, curves, axes, or titles:

```
[TYPE] WINDOWS
```

```
[TYPE] CURVES
```

```
[TYPE] AXES
```

```
[TYPE] TITLES
```

```
[TYPE] <W|C|T>
```

```
CURVEM(C)
```

```
LABELM(C)
```

```
PTSIZEM(C)
```

```
WINDOWM(W)
```

```
FRAMEM(W[,X])
```

```
IMAGEM(W[,X])
```

Appendix C

3D-view control statements

3D-view control statements may be issued at the 3D-view control statement prompt, which is `>>`, or they may be issued in `CMD3D` statements at the ordinary MLAB prompt by typing

```
  * CMD3D("<3D-view control command>")
```

where the double quotes are included and `<3D-view control command>` is a 3D-view control command.

3D-view control statements control (1) the camera of a 3D-window, or (2) the "structural format" of a 3D-window, involving scaling, the appearance of axes, boxes, floors, etc., or (3) the orientation, positioning, and scaling of a particular surface in a 3D-window. All these types of statements may be intermingled as desired in one input string which applies to a specified surface, or to all the surfaces in a specified 3D-window.

A surface is displayed as a camera would view it. All the surfaces in a 3D-space are bounded by a box ($xmin \leq x \leq xmax$; $ymin \leq y \leq ymax$; $zmin \leq z \leq zmax$) which measures $x1 = xmax - xmin$, $y1 = ymax - ymin$, and $z1 = zmax - zmin$ on the side parallel to the x, y, and z-axis, respectively. The center of the box is at the position ($xcenter = xmin + x1/2$, $ycenter = ymin + y1/2$, $zcenter = zmin + z1/2$). $xmin$, $xmax$, $ymin$, $ymax$, and $zmin$, $zmax$ are the given ranges of the data. This box is adjusted if necessary so that $x1 > 0$, $y1 > 0$, and $z1 > 0$.

Initially the camera is positioned at $(xcenter, ymax + y1, zmax)$, with its lens pointed in the direction which results from rotating a unit vector which is initially parallel to the z-axis by 105 degrees about the x-axis (right hand screw). The camera's initial viewing angle is 90 degrees. A number of commands may then be issued from the keyboard to vary the camera viewing position, camera viewing direction, 3D-curve orientation, and 3D-curve position.

The 3D-window to which the camera belongs is initially the default 3D-window W3. The 3D-window can be changed by the command “WINDOW3D Q” which makes Q the active 3D-window.

The camera parameters of the active 3D-window are controlled by the following commands. (All the various command words used in 3D-view commands may be abbreviated to any unique prefix string, e.g. PER may be used in place of PERSPECTIVE.

DOLLY K: the camera is moved K box units in the viewing direction.

TRUCK K: the camera is moved k box units horizontal to the viewing directions.

RAISE K: the camera is moved K box units vertical to the viewing direction.

PAN K: the viewing direction is rotated K degrees in the x-z plane of the camera coordinate system. This is a left-hand screw rotation about the raise-vector.

TILT K: the viewing direction is rotated K degrees in the y-z plane of the camera coordinate system. This is a right-hand screw rotation about the truck-vector.

TWIST K: the camera is right-hand-screw rotated K degrees about the viewing direction.

TRACK [AT A,B,C]: The camera is rotated so as to point to the point (A,B,C). If no “AT A,B,C”-clause is given then the camera is rotated to point to the box-enter of the box about the various surfaces.

TURN K: the camera is right-hand-screw rotated K degrees about the box-center of the window-box about the various surfaces in the plane that contains the camera and the box center and is normal to the raise vector. The camera’s direction of view is modified to maintain the same deviation between the direction of view and the vector to the box center as existed before the turn operation.

ZOOM K: the viewing angle is set to K degrees. (Photographers specify this angle in terms of its cotangent, but we use the actual angle.)

ORIGIN: the camera is moved to 0,0,0 looking up the z-axis with 0 degrees of twist.

RESET: the camera is “automatically” positioned so as to provide the initially-established overall view.

PERSPECTIVE: draws the 3D window with the $1/z$-perspective transformation.

ORTHOGONAL: draws the 3D window without the $1/z$-perspective trans-formation.

Except for the viewing angle, the various camera parameters are taken as incremental, so that DOLLY, TRUCK, RAISE, PAN, TILT, and TWIST arguments are added to the current values to obtain new quantities which determine the new picture to be displayed.

The arguments to DOLLY, TRUCK, and RAISE are relative box units. Suppose the box surrounding all the surface objects has an absolute size in scaled units of a by b by c. Then one unit of camera motion along the X-direction displaces the camera a scaled units, one unit of motion along the y-direction displaces the camera b scaled units, and one unit of motion along the z-direction displaces the camera c scaled units. One unit of transverse motion displaces the camera a certain combination of a, b, and c scaled units as determined by the Pythagorean theorem.

Actually a real camera has additional degrees of freedom. In particular, one can control the height of the lens relative to the film and the angular position of the film plane relative to the direction of the lens-film vector. These parameters are used to control perspective. Moreover an actual camera has a focus parameter which determines a plane or sphere of focus. All points on the plane of focus are seen as point images in the developed picture. Points which are off the plane of focus are seen as small disks whose radius is proportional to the distance they occur from the plane of focus. The intensity of light in such disks falls off from the center in a symmetric distribution. Thus a true camera causes more or less thick bands with fuzzy boundaries to occur where lines should be. For programming purposes it would suffice to map each point to a circle of appropriate radius. Actually however, the infinite depth of field provided by the simple graphic simulation in MLAB is to be preferred.

There are also clauses for modifying the particular 3D-window established as the current active 3D-window that holds various surfaces.

WINDOW3D Q: establish Q as the current 3D-window to which all further 3D view control commands apply

SPACE XSCALE K: x-coordinates of space are scaled by K. The camera and all surfaces in the active 3D-window are affected.

SPACE YSCALE K: y-coordinates of space are scaled by K. The camera and all surfaces in the active 3D-window are affected.

SPACE ZSCALE K: z-coordinates of space are scaled by K. The camera and all surfaces in the active 3D-window are affected.

SPACE SCALE S : scale x, y, and z space coordinates by S.

AUTOWBOX X Y Z : scale all the surfaces with a common set of x, y, and z scales to make them jointly fit in a box of size X by Y by Z.

FLOOR [AT K]: A grid in the $x - y$ plane is drawn at Z=K. If K is not given, K is taken to be 0.

NO FLOOR: the floor grid, if any, disappears from the window.

AXES [AT A,B,C]: axes are drawn from the minimum to the maximum value of each coordinate variable, taken over all surfaces, and intersecting at (A,B,C). If no values for A,B,C are entered, (0,0,0) is implied.

NO AXES: the axes and labels disappear from the window.

XNUMBER [X0 X1]: the x-axis is marked and numbered at four places with the range X0 to X1.

YNUMBER [Y0 Y1]: the y-axis is marked and numbered at four places with the range Y0 to Y1.

ZNUMBER [Z0 Z1]: the z-axis is marked and numbered at four places with the range Z0 to Z1.

NO XNUMBER: x-axis marks and numbers disappear from the window.

NO YNUMBER: y-axis marks and numbers disappear from the window.

NO ZNUMBER: z-axis marks and numbers disappear from the window.

AXESNUMBER N: the x, y, and z axis are each marked at N places.

TICS [K]: draw 3D-tic marks on the axes

NO TICS: remove 3D tic-marks from the axes

BOX: draws a box around the surface objects.

NO BOX: the box disappears from the 3D-window.

HALFBOX: draw the three faces of the box which contain the vertex furthest from the camera.

NO HALFBOX: erase the three faces of the box which contain the vertex furthest from the camera.

BOXNUMBER: draw numeric labels along the edges of the halfbox

NO BOXNUMBER: remove numeric labels along the edges of the halfbox

GRID [K]: establish the floor and the halfbox sides to be $K \times K$ grids when drawn.

NO GRID: establish the floor and the halfbox sides to have no interior grid lines when drawn.

TITLES: enable any XTITLE, YTITLE, and ZTITLE titles

NO TITLES: disable any XTITLE, YTITLE, and ZTITLE titles

XTITLE S: make S the title-string for the x-axis

YTITLE S: make S the title-string for the y-axis

ZTITLE S: make S the title-string for the z-axis

WINDOWCOLOR S: set the color for drawing the 3D-window box, axes, axis labels, and tic marks to color S.

FLOORCOLOR S: set the color for the 3D-window floor to color S.

BOXCOLOR S: set the color for drawing the 3D-window box to color S.

AXESCOLOR S: set the color for drawing the 3D-window axes to color S.

BOXNUMBERCOLOR S: set the color for drawing the 3D-window halfbox numeric labels to color S.

GRIDCOLOR S: set the color for drawing the floor and halfbox-side grids to color S.

TICCOLOR S: set the color for drawing 3D-axis tic marks to color S.

TITLECOLOR: set the color for drawing the x, y, and z-axis title strings to color S.

AXESNUMBERCOLOR S: set the color for drawing the numeric labels on the x, y, and z-axes to color S.

Note: when space is scaled, the apparent camera directions will be affected. When the camera is at the origin looking at the point (1,0,0) and we have executed "SPACE XSCALE 3", then the command "PAN 45" will cause the camera to look at the point (1,1,0), but (1,1,0) is not on the 45 degree radial from the origin! Thus scaling individual surfaces may be preferable to scaling space.

Finally there are some statements for modifying various surfaces individually, including "moving them" in space before "looking through the camera" at the entire scene. These statements apply to the current active surface in the current active 3D window. The particular surface to which the above clauses apply is initially taken to be the first surface in the active 3D-window. It can be changed by the statement "CURVE3D C". When a surface is drawn it becomes the current active surface. The current active status active surface can be explicitly selected by the CURVE3D statement: CURVE3D C. For any given surface established as the current active surface, these statements are as follows:

XSCALE A: the surface is scaled by A in its x-coordinates.

YSCALE A: the surface is scaled by A in its y-coordinates.

ZSCALE A: the surface is scaled by A in its z-coordinates.

SCALE S: x, y, and z surface coordinates are all scaled by S

CBOX X Y Z: scale the surface to fit in a box of size X $\times Y \times Z$

XTRANSLATE A: the surface is translated A units in the x direction.

YTRANSLATE A: the surface is translated A units in the y direction.

ZTRANSLATE A: the surface is translated A units in the z direction.

SYSTEM XROTATE A, XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the underlying "system" coordinate system. Note the default rotation qualifier which is assumed when none is given is "SYSTEM".

SYSTEM YROTATE A:, YROTATE A: the surface is right-hand screw rotated A degrees about the "system" y-axis.

SYSTEM ZROTATE A:, ZROTATE A: the surface is right-hand screw rotated A degrees about the "system" z-axis.

AXES XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the user-specified coordinate system defined by the axes command.

AXES YROTATE A: the surface is right-hand screw rotated A degrees about the user-specified y-axis.

AXES ZROTATE A: the surface is right-hand screw rotated A degrees about the user-specified z-axis.

BOX XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the coordinate system centered in the box enclosing the entire group of objects. This box always has its sides parallel to the system coordinate axes.

BOX YROTATE A: the surface is right-hand screw rotated A degrees about the box y-axis.

BOX ZROTATE A: the surface is right-hand screw rotated A degrees about the box z-axis.

SURFACE XROTATE A: the surface is right-hand screw rotated A degrees about the x-axis of the coordinate system which is centered in the surface object itself, in the sense that the origin for this set of coordinates is the center of the smallest box which has its sides parallel to the system coordinate axes and which encloses the surface object.

SURFACE YROTATE A: the surface is right-hand screw rotated A degrees about the individual surface y-axis.

SURFACE ZROTATE A: the surface is right-hand screw rotated A degrees about the individual surface z-axis.

SEQUENCE: draw the curve by connecting successive points with solid line segments.

NO SEQUENCE: erase the curve by connecting successive points with solid line segments.

ALTERNATE: draw the curve by connecting successive points with alternating solid and invisible line segments.

NO ALTERNATE: erase the curve by connecting successive points with alternating solid and invisible line segments.

NET: draw the surface with a net. This can only be done on a regular rectangular lattice of points.

NO NET: erase the surface net.

HBAND: draw horizontal bands

NO HBAND: erase horizontal bands

VBAND: draw vertical bands

NO VBAND: erase vertical bands

NEEDLES: draw the surface with line segments emanating from the x-y plane to each point.

NO NEEDLES: erase the surface with line segments emanating from the x-y plane to each point.

HIDE: draw a hidden line view of a surface using depth-sort algorithm with quadrilaterals.

NO HIDE: erase a hidden line view of a surface using depth-sort algorithm with quadrilaterals.

POINTS [C]: draw 3D pointtype symbols of type C (0: none, 1: simple dots, 2: 3D-crosses, > 2: the corresponding ASCII character). If C is omitted, 2 is assumed.

PTCOLOR C: point symbols, if any, are drawn in color C.

LCOLOR S: line segments, if any, are drawn in color S

TOPNETCOLOR S: sets the color for the net viewed from above to colornumber S. All other surface drawing (i.e. net, needles, dots) is done in this color, too, by default.

BOTNETCOLOR S: sets the color of a surface's net viewed from below to color number S.

COLORLIST C1 C2 [C3...C16]: change the current 3D-curve's list of colors to be used for shading surfaces. When the surface is drawn using a hidden line algorithm, surface elements (either quadrilaterals or triangles) are shaded according to the magnitude of the angle that is formed between the view direction and the vector normal to the surface element. The magnitude of this angle ranges from 0 to 180 degrees. If there are $n >= 2$ colors in the colorlist, the interval [0,180] is divided into n subintervals and color c_1 is used to shade surface elements with angles are within the first interval, c_2 to shade surface elements with angles within the second interval, etc. Note that the first arguments correspond to smaller angles and the last arguments correspond to larger angles.

BOXCLIP [X1, X2, Y1, Y2, Z1, Z2]: the 3D-clipping box associated with the current active 3D-window is established as the box $[X1, X2] \times [Y1, Y2] \times [Z1, Z2]$. Also, boxclipping is enabled. If the arguments $X1, X2, Y1, Y2, Z1, Z2$ are missing, the enabled 3D-clipping box is the 3D-clipping box that was previously established for the current active 3D-window.

NO BOXCLIP: 3D-box clipping is disabled for the current active 3D-window

Appendix D

Data files

The following list (with comments in quotes) shows the Global Positioning Satellite (time,elevation,azimuth) data triples in the file `gps.dat` used in Chapter 7. Rows containing `0 0 0` are used as marker rows to separate data sets for each satellite.

```
"file: gps.dat"
"times of observation are in column 1"
"elevations from horizon in degrees are in column 2"
"azimuths in degrees are in column 3"
"GPS satellite (time, elevation, azimuth)'s on 6/2/96"
"satellite 24"
0 0 0
14.00 13 227
14.20 20 232
14.45 31 240
15.00 36 245
15.25 44 255
15.45 51 267
16.10 58 281
16.25 62 295
16.45 66 315
17.40 66 17
19.03 47 83
19.55 29 107
20.35 16 121

"satellite 31"
0 0 0
10.25 53 46
10.55 40 48
11.25 27 54
```

```
11.55 17 60

"satellite 27"
0 0 0
10.25 52 304
10.55 62 286
11.25 64 249
11.55 57 224
12.25 44 208
12.55 31 200
13.15 22 196
13.30 14 192

"satellite 19"
0 0 0
10.25 75 225
10.55 64 195
11.25 48 186
11.55 35 182
12.25 20 180

"satellite 15"
0 0 0
10.25 36 143
10.55 48 132
11.25 57 109
11.55 58 83
12.25 51 60
12.55 39 48
13.15 31 45
13.30 24 45
14.00 13 46

"satellite 2"
0 0 0
10.25 27 291
10.55 35 300
11.25 47 310
11.55 58 317
12.25 71 325
12.55 85 338
13.15 85 139
13.30 76 150
14.00 62 156
14.20 51 159
14.45 38 162
15.00 32 163
15.25 20 164
```

```
"satellite 7"
0 0 0
10.55 14 240
11.25 26 250
11.55 35 261
12.25 45 275
12.55 54 290
13.15 59 303
13.30 63 318
14.00 68 345
14.20 70 13
14.45 68 47
15.00 65 61
15.25 59 81
15.45 52 97
16.10 44 108
16.25 38 115
16.45 30 123

"satellite 14"
0 0 0
12.25 16 117
12.55 24 106
13.15 28 97
13.30 31 88
14.00 33 74
14.20 31 63
14.45 27 51
15.00 23 47
15.25 16 41

"satellite 4"
0 0 0
12.55 18 211
13.15 26 215
13.30 34 219
14.00 46 226
14.20 55 236
14.45 65 251
15.00 70 266
15.25 75 299
15.45 74 343
16.10 69 12
16.25 64 26
16.45 57 41
17.40 40 68
19.03 15 98
```

```
"satellite 9"
0 0 0
13.30 14 316
14.00 19 306
14.20 20 296
14.45 19 285
15.00 18 278
15.25 14 269

"satellite 5"
0 0 0
15.00 16 317
15.25 24 314
15.45 32 308
16.10 38 299
16.25 41 292
16.45 42 278
17.40 34 246

"satellite 18"
0 0 0
16.10 17 95
16.25 20 89
16.45 23 79
17.40 20 54

"satellite 16"
0 0 0
16.10 14 200
16.25 22 201
16.45 31 204
17.40 57 215
19.03 80 355
19.55 59 39
20.35 44 54
21.45 21 77
21.50 19 80

"satellite 6"
0 0 0
17.40 17 313
19.03 49 303
19.55 59 263
20.35 50 231
21.10 36 215
21.15 35 214
21.45 22 206
```

```
21.50 19 204

"satellite 26"
0 0 0
19.03 25 180
19.55 51 175
20.35 70 162
21.10 79 94
21.15 79 90
21.45 68 52
21.50 65 50
22.30 46 47
23.10 31 52
24.00 13 63

"satellite 17"
0 0 0
19.55 23 303
20.35 38 312
21.10 53 316
21.15 55 316
21.45 68 315
21.50 71 313
22.30 81 217
23.10 64 183
24.00 38 178

"satellite 23"
0 0 0
20.35 22 264
21.10 33 276
21.15 34 276
21.45 44 292
21.50 46 295
22.30 59 317
23.10 69 344
24.00 74 62

"satellite 1"
0 0 0
23.10 18 242
24.00 37 261

"satellite 28"
0 0 0
21.15 16 317
21.45 22 307
21.50 22 304
```

```
22.30 23 284
23.10 17 270

"satellite 21"
0 0 0
22.30 24 292
23.10 36 301
24.00 55 317
```

The following list (with comments in quotes) shows the data file `hr.dat` used in Chapter 9. The list contains 172 (surface temperature, absolute luminosity) ordered pairs for stars in our galaxy, taken from a figure in Jastrow and Thomson (see bibliography).

```
"file: hr.dat"
"surface temperatures in column 1"
"absolute luminosity in column 2"
20000 500000
18000 500000
10000 300000
9000 300000
3700 100000
11500 105000
14000 110000
5800 90000
6800 80000
21000 75000
16000 75000
16500 74500
20000 25000
20500 25000
3900 25000
18100 24500
16200 24000
16400 23900
5300 11000
18000 10500
18100 10100
17900 10100
15900 10200
13900 10300
4500 10000
4400 10000
15000 8000
15000 7000
15000 6000
18000 3000
```

```
21000 2000
14900 3000
12500 4000
6000 9000
5500 8900
5000 5000
3300 8000
3300 7000
3000 2000
14100 2000
13800 1500
12000 2000
5000 1500
11000 1000
4300 1000
3200 1200
3300 1000
10500 900
10000 800
9000 500
3900 800
5000 500
4200 400
4100 400
4150 390
13900 200
11800 200
5100 200
3900 300
4500 200
5000 150
10000 150
10000 150
9500 150
4800 150
10000 100
9000 100
4300 100
10000 90
10000 80
10000 70
9800 90
9800 70
9800 40
9000 90
8900 100
8900 20
8900 15
```

```
8800 70
8600 60
8100 .005
8100 .002
7900 30
7800 10
6700 30
6700 9
6700 .0008
6700 .0004
6700 .0002
4100 90
4800 85
4800 80
4900 80
4900 70
3100 .00002
5900 .00004
4500 .00006
5800 .0003
2900 .0003
2900 .0006
3100 .0008
2900 .00085
3100 .0009
6000 1
6000 3
5800 .9
5800 .8
5200 .8
5100 .75
5200 .7
5100 .6
5100 .5
5000 .5
4800 .9
4800 .6
4800 .5
4790 .4
4780 .2
4500 .3
3900 .1
2900 .001
2900 .002
2900 .0025
2800 .004
4000 .09
3800 .09
```

```
3700 .085
3800 .08
3900 .08
3700 .075
3800 .07
3900 .075
3600 .06
3500 .06
3400 .06
3300 .06
3300 .05
3400 .05
3600 .05
3100 .045
3150 .045
3600 .048
3800 .01
3750 .02
3100 .005
3300 .005
3100 .006
3000 .008
3000 .009
3000 .01
3000 .02
3000 .03
3100 .04
3100 .02
3100 .01
3100 .009
3100 .008
3100 .007
3500 .04
3300 .04
3500 .03
3500 .02
3500 .01
3500 .009
3400 .008
3200 .007
3300 .008
3350 .009
3200 .01
3200 .02
3300 .03
3200 .03
```

Appendix E

List of Figures

List of Figures

Appendix F

Bibliography

[1] Abramowitz, Milton, and Stegun, Irene, eds., **Handbook of Mathematical Functions**, (US Government Printing Office: Washington, DC, 1972).

[2] Cushing, James T., **Applied Analytical Mathematics for Physical Scientists**, (John Wiley and Sons Inc.: New York, 1975).

[3] Hénon, Michel, and Heiles, Carl, *The Applicability of the Third Integral of Motion: Some Numerical Experiments*, **The Astronomical Journal**, **69** (1964) 73-79.

[4] Jastrow, Robert, and Thompson, Malcolm, **Astronomy Fundamentals and Frontiers**, (John Wiley and Sons Inc.: New York, 1976).

[5] Kerner, Daniel, **MLAB Beginner's Guide**, (Civilized Software, Inc.: Silver Spring, MD, 2013).

[6] Kirsch, John Keats, *An MLAB Example: Space Filling Curves*, (NIH Division of Computer Research and Technology: Bethesda, MD, 1983).

[7] Knott, Gary, **MLAB Applications Manual**, (Civilized Software, Inc.: Silver Spring, MD, 1998).

[8] Knott, Gary, and Kerner, Daniel, **MLAB Reference Manual**, (Civilized Software, Inc.: Silver Spring, MD, 2002).

[9] Koonin, Steve E., and Meredith, Dawn C., **Computational Physics, Fortran Version**, (Addison-Wesley Publishing Company: New York, 1990).

[10] Lauwerier, Hans, **Fractals**, (Princeton University Press: Princeton, New Jersey, 1991).

[11] Segel, Irwin H., **Enzyme Kinetics: Behavior and Analysis of Rapid Equilibrium and Steady State Enzyme Systems**, (John Wiley and Sons Inc.: New York, 1975).

[12] Shapiro, Marvin, **Some Easy Exercises to Teach You MLAB 2-D Graphics**, (NIH Division of Computer Research and Technology: Bethesda, MD, 1983).

[13] Wells, David, **The Penguin Dictionary of Curious and Interesting Geometry**, (Penguin Books: London, England, 1991).

Appendix G

Index

The index begins on the next page. Words and symbols defined and reserved by MLAB commands appear in **`BOLD TYPEWRITER FONT`**.

Index