D I S L I N 11.6

A Data Plotting Extension

for the

Programming Language

Ruby

by

Helmut Michels

© Helmut Michels, Göttingen 1997 - 2026
All rights reserved.

Contents

1 Overview **1**

1.1 Introduction . . . 1
1.2 Dislin Features . . . 1
1.3 Passing Parameters from Ruby to Dislin Routines . . . 2
1.4 FTP Site, Dislin Home Page . . . 2
1.5 Reporting Bugs . . . 2

A Short Description of Dislin Routines **3**

A.1 Initialization and Introductory Routines . . . 3
A.2 Termination and Parameter Resetting . . . 4
A.3 Plotting Text and Numbers . . . 5
A.4 Colours . . . 5
A.5 Fonts . . . 6
A.6 Symbols . . . 6
A.7 Axis Systems . . . 7
A.8 Secondary Axes . . . 8
A.9 Modification of Axes . . . 8
A.10 Axis System Titles . . . 9
A.11 Plotting Data Points . . . 9
A.12 Legends . . . 10
A.13 Line Styles and Shading Patterns . . . 11
A.14 Cycles . . . 12
A.15 Base Transformations . . . 12
A.16 Shielding . . . 12
A.17 Parameter Requesting Routines . . . 12
A.18 Elementary Plot Routines . . . 14
A.19 Conversion of Coordinates . . . 15
A.20 Utility Routines . . . 15
A.21 Binary File I/O . . . 16
A.22 Date Routines . . . 16
A.23 Cursor Routines . . . 17
A.24 Transparency . . . 17
A.25 Bar Graphs . . . 17
A.26 Pie Charts . . . 18
A.27 Coloured 3-D Graphics . . . 19
A.28 3-D Graphics . . . 20
A.29 Geographical Projections . . . 22
A.30 Contouring . . . 23
A.31 Image Routines . . . 24
A.32 Window Routines . . . 25
A.33 Widget Routines . . . 25
A.34 Dislin Quickplots . . . 28

A.35 Using Threads 28
A.36 Reading FITS Files 28
A.37 MPS Logo 28

B Examples **29**
B.1 Demonstration of CURVE 30
B.2 Symbols 32
B.3 Logarithmic Scaling 34
B.4 Interpolation Methods 36
B.5 Line Styles 38
B.6 Legends 40
B.7 Shading Patterns (AREAF) 42
B.8 Vectors 44
B.9 3-D Colour Plot 46
B.10 Surface Plot 48
B.11 Surface Plot 50
B.12 Polar Plots 52
B.13 Contour Plot 55
B.14 Shaded Contour Plot 58
B.15 Pie Charts 60
B.16 World Coastlines and Lakes 62

Chapter 1

Overview

1.1 Introduction

This manual describes a data plotting extension for the interpreted, programming language Ruby. The plotting extension is based on the data plotting library Dislin that is available for several C, Fortran 77 and Fortran 90/95 compilers.

Dislin is a high-level plotting library that contains subroutines and functions for displaying data graphicallly as curves, bar graphs, pie charts, 3-D colour plots, surfaces, contours and maps. The library contains about 700 plotting and parameter setting routines which are now available from Ruby.

1.2 Dislin Features

The following features are supported by Dislin:

- Several output formats can be selected such as X11, PostScript, PDF, CGM, WMF, PNG, BMP, PPM, GIF, TIFF and HPGL.
- 9 software fonts are available where each font provides 6 alphabets and special european characters. Hardware fonts for PostScript printers and X11 and Windows displays can also be used.
- Plotting of two- and three-dimensional axis systems. Axes can be linearly or logarithmically scaled and labeled with linear, logarithmic, date, time, map and user-defined formats.
- Plotting of curves. Several curves can appear in one axis system and can be differentiated by colour, line style and pattern. Multiple axis systems can be displayed on one page.
- Plotting of legends.
- Elementary plot routines for lines, vectors and outlined or filled regions such as rectangles, circles, arcs, ellipses and polygons.
- Shielded regions can be defined.
- Business graphics.
- 3-D colour graphics.
- 3-D graphics.
- Elementary image routines.
- Geographical projections and plotting of maps.
- Contouring.

1.3 Passing Parameters from Ruby to Dislin Routines

The passing of parameters from Ruby to Dislin routines is not so strict as in other programming languages. The following rules are applied:

- Parameters can be passed from Ruby to Dislin routines as variables, constants and expressions.
- String constants must be enclosed in a pair of either apostrophes or quotation marks.
- Floating point parameters can be passed from Ruby to Dislin as integer and floating point numbers.
- Integer parameters can be passed from Ruby to Dislin as integer and floating point values. If a floating point value is passed for an integer parameter, the fractional part of the floating point value will be truncated.
- Multi-dimensional arrays can be passed to Dislin as one-dimensional arrays with row major ordering.
- The names of callback routines (user-defined Ruby functions) must be passed to Dislin as a string.
- Memory must be allocated for arrays in a parameter list which are used as output parameters from Dislin. Arrays can be created with the Ruby command Array.new.

Note: Normally, the number and meaning of parameters passed to Dislin routines are identical with the syntax description of the routines in the Dislin manual. Dislin routines which return more than one scalar are implemented in Ruby as functions that return an array of scalars. For example, the statement 'array = getpag ()' returns the page width and height in array[0] and array[1].

1.4 Dislin Home Page

The Dislin software is available from the site:

https://www.dislin.de

The Ruby home page is:

https://www.ruby-lang.org

1.5 Reporting Bugs

Dislin is well tested by many users and should be very bug free. However, no software is perfect. If you have any problems with Dislin, contact the author:

Helmut Michels
Dislin Software
Am Hachweg 10
37083 Göttingen, Germany
Email: michels@dislin.de

Appendix A

Short Description of Dislin Routines

This appendix presents a short description of all Dislin routines that can be called from Ruby. A complete description of the routines can be found in the Dislin manual or via the online help of Dislin. For parameters, the following conventions are used:

- integer variables begin with the character N or I;
- strings begin with the character C;
- other variables are floating point numbers;
- one-dimensional arrays end with the keyword 'ray', two-dimensional arrays with the keyword 'mat'.

A.1 Initialization and Introductory Routines

Routine	Meaning
bmpmod(n, cval, copt)	defines the physical resolution of BMP files.
bufmod(cmod, ckey)	modifies the behaviour of the output buffer.
cgmbgd(xr, xg, xb)	defines the background colour for CGM files.
cgmpic(cstr)	sets the picture ID for CGM files.
disenv(cenv)	defines the Dislin environment.
disini()	initializes Dislin.
erase()	clears the screen.
errdev(cdev)	defines the error device.
errfil(cfil)	sets the name of the error file.
errmod(ckey, copt)	modifies the printing of error messages.
filbox(nx, ny, nw, nh)	defines the position and size of included metafiles.
filmod(cmode)	defines the file creation mode.
filopt(copt, ckey)	modifies rules for creating file versions.
[nw, nh, iret] = filsiz(cfil)	returns the size on an image file.
iret = filtyp(ctype)	returns the type of a file.
filwin(nx, ny, nw, nh)	defines a rectangle of the image that will be included by INCFIL.
gifmod(cmod, ckey)	enables transparency for GIF files.
hpgmod(cmod, ckey)	defines options for HPGL files.
hworig(nx, ny)	defines the origin of the PostScript hardware page.
hwpage(nw, nw)	defines the size of the PostScript hardware page.
hwscal(xscl)	modifies the scale operator in PostScript files.
imgfmt(copt)	defines the format of image files.
incfil(cfil)	includes metafiles into a graphics.

Routine	Meaning
metafl(cfmt)	defines the plotfile format.
newpag()	creates a new page.
origin(nx, ny)	defines the origin.
page(nw, nh)	sets the page size.
pagera()	plots a page border.
pagfll(iclr)	fills the page with a colour.
paghdr(c1, c2, iopt, idir)	plots a page header.
pagmod(copt)	selects a page rotation.
pagorg(copt)	defines the origin of the page.
cbuf = pdfbuf()	copies a PDF file to a string.
pdfmod(cmod, ckey)	defines PDF options.
pdfmrk(cstr, copt)	defines bookmarks for PDF files.
pngmod(cmod, ckey)	enables transparency for PNG files.
psmeta (cinf, copt)	adds comment lines to PostScript files.
sclfac(x)	defines a scaling factor for the entire plot.
sclmod(copt)	defines a scaling mode.
scrmod(copt)	swaps back- and foreground colours.
sendbf()	flushes the output buffer.
setfil(cfil)	sets the plotfile name.
setpag(copt)	selects a predefined page format.
setxid(id, copt)	defines an external X Window or pixmap.
symfil(cdev, cstat)	sends a plotfile to a device.
tifmod(n, cval, copt)	defines the physical resolution of TIFF files.
unit(nu)	defines the logical unit for messages.
units(copt)	defines the plot units.
wmfmod(cmod, ckey)	modifies the format of WMF files.

Figure A.1: Initialization and Introductory Routines

A.2 Termination and Parameter Resetting

Routine	Meaning
delglb()	frees space allocated for global parameters.
disfin()	terminates Dislin.
endgrf()	terminates an axis system and sets the level to 1.
reset(copt)	resets parameters to default values.

Figure A.2: Termination and Parameter Resetting

A.3 Plotting Text and Numbers

Routine	Meaning
angle(n)	defines the character angle.
chaang(x)	defines an inclination angle for characters.
chaspc(x)	affects character spacing.
chawth(x)	affects the width of characters.
fixspc(x)	sets a constant character width.
frmess(nfrm)	defines the thickness of text frames.
height(n)	defines the character height.
messag(cstr, nx, ny)	plots text.
mixalf()	enables control signs in character strings for plotting indices and exponents.
newmix()	defines an alternate set of control characters for plotting indices and exponents.
n = nlmess(cstr)	returns the length of character strings in plot coordinates.
number(x, ndig, nx, ny)	plots floating point numbers.
numfmt(copt)	determines the format of numbers.
numode(c1, c2, c3, c4)	modifies the appearance of numbers.
rlmess(cstr, x, y)	plots text.
rlnumb(x, ndig, xp, yp)	plots numbers.
setbas(xfac)	determines the position of indices and exponents.
setexp(xfac)	sets the height of indices and exponents.
setmix(char, cmix)	defines global control signs for plotting indices and exponents.
texmod(cmode)	enables TeX mode for plotting mathematical formulas.
texopt(copt, ctype)	defines TeX options.
texval(x, copt)	defines TeX values.
txtbgd(nclr)	defines a background colour for text and numbers.
txtjus(copt)	defines the alignment of text and numbers.

Figure A.3: Plotting Text and Numbers

A.4 Colours

Routine	Meaning
color(color)	defines colours.
[xr,xg,xb]= hsvrgb(xh, xs, xv)	converts HSV to RGB coordinates.
n = indrgb(xr, xg, xb)	calculates a colour index.
n = intrgb(xr, xg, xb)	calculates an explicit colour value.
myvlt(rray, gray, bray, n)	changes the current colour table.
[xh,xs,xv] = rgbhsv(xr, xg, xb)	converts RGB to HSV coordinates.
setclr(nclr)	defines colours.

Routine	Meaning
setind(i, xr, xg, xb)	changes the current colour table.
setrgb(xr, xg, xb)	defines colours.
setvlt(cvlt)	selects a colour table.
vltfil(cfil, cmod)	store or loads a colour table.

Figure A.4: Colours

A.5 Fonts

Routine	Meaning
basalf(calph)	defines the base alphabet.
bmpfnt(cfont)	defines a bitmap font.
chacod(copt)	defines the character coding.
complx()	sets a complex font.
duplx()	sets a double-stroke font.
disalf()	sets the default font.
eushft(cnat, char)	defines a shift character for European characters.
gothic()	sets a gothic font.
helve()	sets a shaded font.
helves()	sets a shaded font with small characters.
helvet()	sets a shaded font with thick characters.
hwfont()	sets a standard hardware font.
psfont(cfont)	sets a PostScript font.
psmode(cmode)	enables Greek and Italic characters in PostScript fonts.
serif()	sets a complex shaded font.
simplx()	sets a single-stroke font.
smxalf(calph, c1, c2, n)	defines shift characters for alternate alphabets.
triplx()	sets a triple-stroke font.
ttfont(cfont)	loads a TrueType font.
winfnt(cfont)	sets a TrueType font.
x11fnt(cfont, copt)	sets an X11 font.

Figure A.5: Fonts

A.6 Symbols

Routine	Meaning
hsymbl(n)	defines the height of symbols.
mysymb(xray, yray, n, isym, iflag)	defines an user-defined symbol.
rlsymb(nsym, x, y)	plots symbols for user coordinates.
symbol(nsym, nx, ny)	plots symbols.
symrot(xang)	defines a rotation angle for symbols.

Figure A.6: Symbols

A.7 Axis Systems

Routine	Meaning
addlab(cstr, v, itic, cax)	plots additional single labels.
ax2grf()	suppresses the plotting of the upper X- and left Y-axis.
ax3len(nxl, nyl, nzl)	defines axis lengths for a coloured 3-D axis system.
axsbgd(iclr)	defines the background colour.
axsers()	erases the contents of an axis system.
axslen(nxl, nyl)	defines axis lengths for a 2-D axis system.
axsorg(nx, ny)	determines the position of a crossed axis system.
axspos(nxp, nyp)	determines the position of axis systems.
axstyp(ctype)	select rectangular or crossed axis systems.
axgit()	plots the lines X = 0 and Y = 0.
box2d()	plots a border around an axis system.
center()	centres axis systems.
cross()	plots the lines X = 0 and Y = 0 marked with ticks.
endgrf()	terminates an axis system.
frame(nfrm)	defines the frame thickness of axis systems.
frmclr(nclr)	defines the colour of frames.
[a,b,or,stp,ndig] = gaxpar(v1, v2, copt, cax)	calculates axis parameters.
grace(ngrace)	affects the clipping margin of axis systems.
graf(xa, xe, xor, xstp, ya, ye, yor, ystp)	plots a two-dimensional axis system.
graf3(xa, xe, xor, xstp, ya, ye, yor, ystp, za, ze, zor, zstp)	plots an axis system for colour graphics.
grafp(xe, xor, xstp, yor, ystp)	plots a polar axis system.
grafr(xray, n, yray, m)	plots an axis system for a Smith chart.
grdpol(nx, ny)	plots a polar grid.
grid(nx, ny)	overlays a grid on an axis system.
gridim(zimg, zre1, zre2, n)	plots a grid line with a constant imaginary part in a Smith chart.
gridre(zre, zimg1, zimg2, n)	plots a grid line with a constant real part in a Smith chart.
noclip()	suppresses clipping of user coordinates.
nograf()	suppresses the plotting of an axis system.
polmod(cpos, cdir)	modifies the appearance of polar labels.
setgrf(c1, c2, c3, c4)	suppresses parts of an axis system.
setscl(xray, n, cax)	sets automatic scaling.
title()	plots a title over an axis system.
xaxgit()	plots the line Y = 0.
xcross()	plots the line Y = 0 and marks it with ticks.
yaxgit()	plots the line X = 0.
ycross()	plots the line X = 0 and marks it with ticks.

Figure A.7: Axis Systems

A.8 Secondary Axes

Routine	Meaning
xaxis(xa, xe, xor, xstp, nl, cstr, it, nx, ny)	plots a linear X-axis.
xaxlg(xa, xe, xor, xstp, nl, cstr, it, nx, ny)	plots a logarithmic X-axis.
yaxis(ya, ye, yor, ystp, nl, cstr, it, nx, ny)	plots a linear Y-axis.
yaxlg(ya, ye, yor, ystp, nl, cstr, it, nx, ny)	plots a logarithmic Y-axis.
ypolar(ya, yb, yor, ystp, cstr, ndist)	plots a polar Y-axis.
zaxis(za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a linearly scaled colour bar.
zaxlg(za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a logarithmically scaled colour bar.

Figure A.8: Secondary Axes

A.9 Modification of Axes

Routine	Meaning
axclrs(nclr, copt, cax)	defines colours for axis elements.
axends(copt, cax)	suppresses certain labels.
axsscl(copt, cax)	defines the axis scaling.
hname(nh)	defines the character height of axis names.
intax()	defines integer numbering for all axes.
labdig(ndig, cax)	sets the number of decimal places for labels.
labdis(ndis, cax)	sets the distance between labels and ticks.
labels(copt, cax)	selects labels.
labjus(copt, cax)	defines the alignment of axis labels.
labmod(ckey, cval, cax)	modifies date labels.
labpos(copt, cax)	determines the position of labels.
labtyp(copt, cax)	defines vertical or horizontal labels.
logtic(copt)	modifies the appearance of logarithmic ticks.
mylab(cstr, itic, cax)	sets user-defined labels.
namdis(ndis, cax)	sets the distance between axis names and labels.
name(cstr, cax)	defines axis titles.
namjus(copt, cax)	defines the alignment of axis titles.
noline(cax)	suppresses the plotting of axis lines.
rgtlab()	right-justifies labels.
rvynam()	defines an angle for Y-axis names.
ticks(ntics, cax)	sets the number of ticks.
ticlen(nmaj, nmin)	sets the length of ticks.

Routine	Meaning
ticmod(copt, cax)	modifies the plotting of ticks at calendar axes.
ticpos(copt, cax)	determines the position of ticks.
timopt()	modifies time labels.

Figure A.9: Modification of Axes

A.10 Axis System Titles

Routine	Meaning
htitle(nh)	defines the character height of titles.
lfttit()	left-justifies title lines.
linesp(xfac)	defines line spacing.
titjus(copt)	defines the alignment of titles.
title()	plots axis system titles.
titlin(cstr, ilin)	defines text lines for titles.
titpos(copt)	defines the position of titles.
vkytit(nshift)	shifts titles in the vertical direction.

Figure A.10: System Titles

A.11 Plotting Data Points

Routine	Meaning
bars(xray, y1ray, y2ray, n)	plots a bar graph.
bars3d(xray, yray, z1ray, z2ray, xwray, ywray, icray, n)	plots 3-D bars.
chnatt()	changes curve attributes.
chncrv(copt)	defines attributes changed automatically by CURVE.
color(color)	defines the colour used for text and lines.
crvmat(zmat, n, m, ixpts, iypts)	plots a coloured surface.
curve(xray, yray, n)	plots curves.
curve3(xray, yray, zray, n)	plots coloured rectangles.
curvx3(xray, y, zray, n)	plots rows of coloured rectangles.
curvy3(x, yray, zray, n)	plots columns of coloured rectangles.
errbar(xray, yray, e1ray, e2ray, n)	plots error bars.
fbars(x1ray, x2ray, x3ray, x4ray, x5ray, n)	plots financial bars.
field(x1ray, y1ray, x2ray, y2ray, n, ivec)	plots a vector field.
gapcrv(xgap)	defines gaps plotted by CURVE.
inccrv(ncrv)	defines the number of curves plotted with equal attributes.
licpts(xvmat, yvmat, nx, ny, itmat, wmat)	calculates a Line Integral Convolution image of a vector field.

Routine	Meaning
incmrk(nmrk)	selects symbols or lines for CURVE.
licmod(cmod, ckey)	sets modes for the LIC algorithm.
[a,b,r] = linfit(xray, yray, n, copt)	plots a fitted line.
marker(nsym)	sets the symbols plotted by CURVE.
mrkclr(nclr)	defines the colour of symbols plotted by CURVE.
nancrv(copt)	enables handling of NaN values in curves.
nochek()	suppresses listing of out of range data points.
piegrf(cbuf, nlin, xray, n)	plots a pie chart.
polcrv(copt)	defines the interpolation method used by CURVE.
resatt()	resets curve attributes.
setres(nx, ny)	sets the size of coloured rectangles.
shdcrv(x1ray, y1ray, n1, x2ray, y2ray, n2)	plots shaded areas between curves.
splmod(ngrad, npts)	modifies spline interpolation.
stmmod(cmod, ckey)	sets streamline modes.
stmopt(n, ckey)	defines integer options for streamlines.
n = stmpts(xvmat, yvmat, nx, ny, xpray, ypray, x0, y0, xray, yray, nmax)	generates a streamline.
stmtri(xvray, yvray, xpray, ypray, n, i1ray, i2ray, i3ray, ntri, xsray, ysray, nray)	plots streamlines from triangulated data.
stmval(x, ckey)	defines floating point options for streamlines.
stream(xvmat, yvmat, nx, ny, xpray, ypray, xsray, ysray, n)	plots streamlines.
thkcrv(nthk)	defines the thickness of curves.
itmat = txture(nx, ny)	generates a texture array for LICPTS.
vecfld(xvray, yvray, xpray, ypray, n, ivec)	plots a vector field.
vecmat(xvmat, yvamt, nx, ny, xpray, ypray, ivec)	plots a vector field on a regular grid.

Figure A.11: Plotting Data Points

A.12 Legends

Routine	Meaning
frame(nfrm)	sets the frame thickness of legends.
legbgd(nclr)	defines the background colour of legends.
legend(cbuf, ncor)	plots legends.
legini(cbuf, nlin, nmaxln)	initializes legends. cbuf is a dummy parameter for Ruby. The text of legend lines is stored in an internal buffer.

Routine	Meaning
leglin(cbuf, cstr, ilin)	defines text for legend lines.
legopt(xf1, xf2, xf3)	modifies the appearance of legends.
legpat(ityp, ithk, isym, iclr, ipat, ilin)	stores curve attributes.
legpos(nxp, nyp)	determines the position of legends.
legsel(nray, n)	selects legend lines.
legtbl(n, copt)	sets the number of columns in table legends.
legtit(ctitle)	defines the legend title.
legtyp(ctype)	defines horizontal or vertical legend lines.
legval(x, copt)	modifies the appearance of legends.
linesp(xfac)	affects line spacing.
mixleg()	enables multiple text lines in legends.
nxl = nxlegn(cbuf)	returns the width of legends in plot coordinates.
nyl = nylegn(cbuf)	returns the height of legends in plot coordinates.

Figure A.12: Legends

A.13 Line Styles and Shading Patterns

Routine	Meaning
chndot()	sets a dotted-dashed line style.
chndsh()	sets a dashed-dotted line style.
color(color)	sets a colour.
dash()	sets a dashed line style.
dashl()	sets a long-dashed line style.
dashm()	sets a medium-dashed line style.
dot()	sets a dotted line style.
dotl()	sets a long-dotted line style.
hwmode(cmod, ckey)	enables or disables hardware features for line styles and shading patterns.
linclr(nray, n)	defines colours for line styles.
lintyp(itype)	defines a line style.
linwid(nwidth)	sets the line width.
lncap(copt)	sets the line cap parameter.
lnjoin(copt)	sets the line join parameter.
lnmlt(xfac)	sets the miter limit parameter.
myline(nray, n)	sets a user-defined line style.
mypat(iang, ityp, idens, icross)	defines a global shading pattern.
penwid(nwidth)	sets the pen width.
shdfac(xfac)	modifies the distance of scan lines for software shading.
shdpat(ipat)	selects a shading pattern.
solid()	sets a solid line style.

Figure A.13: Line Styles and Shading Patterns

A.14 Cycles

Routine	Meaning
clrcyc(index, iclr)	modifies the colour cycle.
lincyc(index, itype)	modifies the line style cycle.
patcyc(index, ipat)	modifies the pattern cycle.

Figure A.14: Cycles

A.15 Base Transformations

Routine	Meaning
tr3axs(x, y, z, a)	defines a rotation about an arbitrary axis.
tr3res()	resets 3-D base transformations.
tr3rot(xrot, yrot, zrot)	affects the 3-D rotation of plot vectors.
tr3scl(xscl, yscl, zscl)	affects the 3-D scaling of plot vectors.
tr3shf(xshf, yshf, zshf)	affects the 3-D shifting of plot vectors.
trfres()	resets base transformations.
trfrot(xang, nx, ny)	affects the rotation of plot vectors.
trfscl(xscl, yscl)	affects the scaling of plot vectors.
trfshf(nx, ny)	affects the shifting of plot vectors.

Figure A.15: Base Transformations

A.16 Shielding

Routine	Meaning
shield(carea, cmode)	defines automatic shielding.
shlcir(nx, ny, nr)	defines circles as shielded areas.
shldel(id)	deletes shielded areas.
shlell(nx, ny, na, nb, t)	defines ellipses as shielded areas.
id = shlind()	returns the index of a shielded area.
shlpie(nx, ny, nr, a, b)	defines pie segments as shielded areas.
shlpol(nxray, nyray, n)	defines polygons as shielded areas.
shlrct(nx, ny, nw, nh, t)	defines rotated rectangles as shielded areas.
shlrec(nx, ny, nw, nh)	defines rectangles as shielded areas.
shlres(n)	deletes shielded areas.
shlvis(id, cmode)	enables or disables shielded areas.

Figure A.16: Shielding

A.17 Parameter Requesting Routines

Routine	Meaning
calf = getalf()	returns the base alphabet.
n = getang()	returns the current angle used for text and numbers.

Routine	Meaning
[nx,ny,nw,nh] = getclp()	returns the currents clipping window.
n = getclr()	returns the current colour number.
[nx,ny,nz] = getdig()	returns the number of decimal places used in labels.
cdsp = getdsp()	returns the terminal type.
cfil = getfil()	returns the current plotfile name.
[a,b,or,stp] = getgrf(cax)	returns the scaling of the current axis system.
n = gethgt()	returns the current character height.
n = gethnm()	returns the character height of axis titles.
[xr,xg,xb] = getind(i)	returns the RGB coordinates for a colour index.
[cx,cy,cz] = getlab()	returns the current labels.
[nx,ny,nz] = getlen()	returns the current axis lengths.
n = getlev()	returns the current level.
n = getlin()	returns the current line width.
cmfl = getmfl()	returns the current file format.
c = getmix(copt)	returns shift characters for indices and exponents.
[nx,ny] = getor()	returns the current origin.
[nx,ny] = getpag()	returns the current page size.
n = getpat()	returns the current shading pattern.
n = getplv()	returns the patch level of Dislin.
[nx,ny] = getpos()	returns the position of the axis system.
[nx,ny] = getran()	returns the range of colour bars.
[nx,ny] = getres()	returns the size of points used in 3-D colour graphics.
[xr,xb,xg] = getrgb()	returns the RGB coordinates of the current colour.
[nx,ny,nz] = getscl()	returns the current axis scaling.
[nx,ny,nz] = getscm()	informs if automatic scaling is enabled.
[nw,nh] = getscr()	returns the screen size in pixels.
c = getshf(copt)	returns shift characters for European characters.
[nx,ny,nz] = getsp1()	returns the distance between axis ticks and labels.
[nx,ny,nz] = getsp2()	returns the distance between axis labels and names.
[nsym,nh] = getsym()	returns the current symbol number and height.
[nmaj,nmin] = gettcl()	returns the current tick lengths.
[nx,ny,nz] = gettic()	returns the number of ticks plotted between labels.
n = gettyp()	returns the current line style.
n = getuni()	returns the current unit used for messages.
x = getver()	returns the Dislin version number.
[nytit,nxbar,nybar] = getvk()	returns the current lengths used for shifting.
cvlt = getvlt()	returns the current colour table.
n = getwid()	returns the width of colour bars.
[nx,ny,nw,nh] = getwin()	returns the position and size of the graphics window.
id = getxid('WINDOW')	returns the X window ID.
[c1,c2,n] = gmxalf(copt)	returns shift characters for additional alphabets.

Figure A.17: Parameter Requesting Routines

A.18 Elementary Plot Routines

Routine	Meaning
arcell(nx, ny, na, nb, alpha, beta, theta)	plots elliptical arcs.
areaf(nxray, nyray, n)	plots polygons.
circle(nx, ny, nr)	plots circles.
connpt(x, y)	plots a line to a point.
ellips(nx, ny, nr1, nr2)	plots ellipses.
line(nx, ny, nu, nv)	plots lines.
noarln()	suppresses the outline of geometric figures.
pie(nx, ny, nr, a, b)	plots pie segments.
point(nx, ny, nb, nh, nc)	plots coloured rectangles where the position is defined by the centre point.
recfll(nx, ny, nw, nh, nc)	plots coloured rectangles.
rectan(nx, ny, nw, nh)	plots rectangles.
rndrec(nx, ny, nw, nh, iopt)	plots a rectangle with rounded corners.
rlarc(x, y, r1, r2, a, b, t)	plots elliptical arcs for user coordinates.
rlarea(xray, yray, n)	plots polygons for user coordinates.
rlcirc(x, y, r)	plots circles for user coordinates.
rlell(x, y, r1, r2)	plots ellipses for user coordinates.
rline(x, y, u, v)	plots lines for user coordinates.
rlpie(x, y, r, a, b)	plots pie segments for user coordinates.
rlpoin(x, y, nw, nh, nc)	plots coloured rectangles for user coordinates.
rlrec(x, y, xw, xh)	plots rectangles for user coordinates.
rlrnd(x, y, xw, xh, iopt)	plots for user coordinates a rectangle with rounded corners.
rlsec(x, y, r1, r2, a, b, ncol)	plots coloured pie sectors for user coordinates.
rlvec(x1, y1, x2, y2, ivec)	plots vectors for user coordinates.
rlwind(x, xp, yp, nw, a)	plots wind speed symbols for user coordinates.
sector(nx, ny, nr1, nr2, a, b, ncol)	plots coloured pie sectors.
strtpt(x, y)	moves the pen to a point.
triflc(xray, yray, icray, n)	plots solid filled rectangles.
trifll(xray, yray)	plots solid filled rectangles.
vecclr(nclr)	defines coulour for arrow heads.
vecopt(xopt, ckey)	defines vector options.
vector(nx, ny, nu, nv, ivec)	plots vectors.
windbr(x, nx, ny, nw, a)	plots wind speed symbols.
xmove(x, y)	moves the pen to a point.
xdraw(x, y)	plots a line to a point.

Figure A.18: Elementary Plot Routines

A.19 Conversion of Coordinates

Routine	Meaning
iray = colray(zray, n)	converts Z-coordinates to colour numbers.
[xp,yp] = getico(x, y)	converts a complex reflection factor to an impedance.
[xp,yp] = getrco (x, y)	converts a complex impedance to a reflection factor.
n = nxpixl(ix, iy)	converts X plot coordinates to pixel.
n = nxposn(x)	converts X-coordinates to plot coordinates.
n = nypixl(ix, iy)	converts Y plot coordinates to pixel.
n = nyposn(y)	converts Y-coordinates to plot coordinates.
n = nzposn(z)	converts Z-coordinates to colour numbers.
trfco1(xray, n, cfrom, cto)	converts one-dimensional coordinates.
trfco2(xray, yray, n, cfr, cto)	converts two-dimensional coordinates.
trfco3(xray, yray, zray, n, cfr, cto)	converts three-dimensional coordinates.
trfrel(xray, yray, n)	converts X- and Y-coordinates to plot coordinates.
x = xinvrs(nx)	converts X plot coordinates to user coordinates.
x = xposn(x)	converts X-coordinates to real plot coordinates.
y = yinvrs(ny)	converts Y plot coordinates to user coordinates.
y = yposn(y)	converts Y-coordinates to real plot coordinates.

Figure A.19: Conversion of Coordinates

A.20 Utility Routines

Routine	Meaning
bezier(xray, yray, n, xpray, ypray, np)	calculates a Bezier interpolation.
n = bitsi4(nbits, ninp, iinp, nout, iout)	allows bit manipulation on 32 bit variables.
[xm, ym, r] = circ3p(x1, y1, x2, y2, x3, y3)	calculates a circle specified by 3 points.
cstr = fcha(x, ndig)	converts floating point numbers to character strings.
n = flen(x, ndig)	calculates the number of digits for floating point numbers.
nh = histog(xray, n, xhray, yhray)	calculates a histogram.
cstr = intcha(nx)	converts integers to character strings.
n = intlen(nx)	calculates the number of digits for integers.
cstr = intutf(iray, n)	converts Unicode numbers to an UTF8 string.
n = nlmess(cstr)	returns the length of character strings in plot coordinates.
n = nlnumb(x, ndig)	returns the length of numbers in plot coordinates.
n = polclp(xray, yray, n, x2ray, y2ray, nmax, xv, cedge)	clips a polygon.
sortr1(xray, n, copt)	sorts floating point numbers.
sortr2(xray, yray, n, copt)	sorts points in the X-direction.

Routine	Meaning
npt = spline(xray, yray, n, xsray, ysray)	returns splined points as calculated in CURVE.
iray = swapi4(iray, n)	swaps the bytes of 32 bit integer variables.
zmat2 = trfmat(zmat, nx, ny, nx2, ny2)	converts matrices.
ntri = triang(xray, yray, n, i1ray, i2ray, i3ray, nmax)	calculates the Delaunay triangulation.
n = trmlen(cstr)	calculates the number of characters in character strings.
cstr = upstr(cstr)	converts a character string to uppercase letters.
iray = utfint(cstr)	converts an UTF8 string to Unicode numbers.

Figure A.20: Utility Routines

A.21 Binary File I/O

Routine	Meaning
istat = closfl(nu)	closes a file.
istat = openfl(cfil, nu, irw)	opens a file for binary I/O.
istat = posifl(nu, nbyte)	skips to a certain position relative to the start.
iray = readfl(nu, nbyte)	reads a given number of bytes.
istat = skipfl(n, nbyte)	skips a number of bytes from the current position.
n = tellfl(nu)	returns the file position.
n = writfl(nu, iray, nbyte)	writes a given number of bytes.

Figure A.21: Binary File I/O

A.22 Date Routines

Routine	Meaning
basdat(id, im, iy)	defines the base date.
n = incdat(id, im, iy)	returns incremented days.
n = nwkday(id, im, iy)	returns the weekday of a date.
[id,im,iy] = trfdat(n)	converts incremented days to a date.

Figure A.22: Date Routines

A.23 Cursor Routines

Routine	Meaning
n = csrkey()	returns a character key.
[nx1, ny1, nx2, ny2] = csrlin()	returns the end points of a line.
n = csrmov(nxray, nyray, nmax)	collects cursor movements.
n = csrpol(nxray, nyray, nmax)	returns collected cursor positions.
[nx,ny,nkey] = csrpos()	sets and returns the cursor position.
[nx,ny] = csrpt1()	returns a pressed cursor position.
n = csrpts(nxray, nyray, nmax)	collects cursor positions.
[nx1, ny1, nx2, ny2] = csrrec()	returns opposite corners of a rectangle.
csrtyp(copt)	selects the cursor type.
csruni(copt)	selects the unit of returned cursor positions.
setcsr(copt)	defines the cursor type of the graphics window.

Figure A.23: Cursor Routines

A.24 Transparency

Routine	Meaning
tprfin()	terminates alpha blending.
tprini()	initializes alpha blending.
tprmod(cmode, ckey)	modifies alpha blending.
tprval(x)	sets the alpha value.

Figure A.24: Transparency

A.25 Bar Graphs

Routine	Meaning
barbor(iclr)	defines the colour of bar borders.
barclr(ic1, ic2, ic3)	defines bar colours.
bargrp(ngrp, gap)	affects clustered bars.
barmod(copt, ckey)	enables variable bars.
baropt(xf, ang)	modifies the appearance of 3-D bars.
barpos(copt)	selects predefined positions for bars.
bars(xray, y1ray, y2ray, n)	plots bar graphs.
bartyp(copt)	selects vertical or horizontal bars.
chnbar(copt)	modifies the appearance of bars.
labclr(nclr, 'BARS')	defines the colour of bar labels.
labdig(ndig, 'BARS')	defines the number of decimal places in bar labels.
labels(copt, 'BARS')	defines bar labels.
labpos(copt, 'BARS')	defines the position of bar labels.

Figure A.25: Bar Graphs

A.26 Pie Charts

Routine	Meaning
chnpie(copt)	defines colour and pattern attributes for pie segments.
labclr(nclr, 'PIE')	defines the colour of segment labels.
labdig(ndig, 'PIE')	defines the number of decimal places in segment labels.
labels(copt, 'PIE')	defines pie labels.
labpos(copt, 'PIE')	defines the position of segment labels.
labtyp(copt, 'PIE')	modifies the appearance of segment labels.
piebor(iclr)	defines the colour of pie borders.
piecbk(Routine)	defines a callback routine for PIEGRF.
pieclr(ic1ray, ic2ray, n)	defines pie colours.
pieexp()	defines exploded pie segments.
piegrf(cbuf, nlin, xray, n)	plots pie charts.
pielab(clab, cpos)	sets additional character strings plotted in segment labels.
pieopt(xf, ang)	modifies the appearance of 3-D pies.
pierot(angle)	sets a rotation angle for 2-D pie charts.
pietyp(copt)	selects 2-D of 3-D pie charts.
pieval(x, ckey)	modifies parameters for pie charts.
pievec(ivec, copt)	modifies the arrow plotted between labels and segments.

Figure A.26: Pie Charts

A.27 Coloured 3-D Graphics

Routine	Meaning
ax3len(nx, ny, nz)	defines axis lengths.
colran(nx, ny)	defines the range of colour bars.
crvmat(zmat, n, m, ixp, iyp)	plots a coloured surface.
crvqdr(xray, yray, zray, n)	plots coloured quadrangles.
crvtri(xray, yray, zray, n, i1ray, i2ray, i3ray, ntri)	plots the coloured surface of an Delaunay triangulation.
curve3(xray, yray, zray, n)	plots coloured rectangles.
curvx3(xray, y, zray, n)	plots rows of coloured rectangles.
curvy3(x, yray, zray, n)	plots columns of coloured rectangles.
erase()	erases the screen.
frmbar(nfrm)	defines the thickness of frames around colour bars.
graf3(xa, xe, xor, xstp, ya, ye, yor, ystp, za, ze, zor, zstp)	plots a coloured axis system.
jusbar(copt)	defines the alignment of colour bars.
nobar()	suppresses the plotting of colour bars.
nobgd()	suppresses the plotting of points which have the same colour as the background.
n = nzposn(z)	converts a Z-coordinate to a colour number.
point(nx, ny, nb, nh, nc)	plots coloured rectangles.
posbar(copt)	sets the position of colour bars.
recfll(nx, ny, nw, nh, nc)	plots coloured rectangles.
rlpoin(x, y, nw, nh, nc)	plots coloured rectangles for user coordinates.
rlsec(x, y, r1, r2, a, b, ncol)	plots coloured pie sectors for user coordinates.
sector(nx, ny, nr1, nr2, a, b, ncol)	plots coloured pie sectors.
setres(nx, ny)	defines the size of coloured rectangles.
spcbar(nspc)	sets the space between colour bars and axis systems.
vkxbar(nshift)	shifts colour bars in the X-direction.
vkybar(nshift)	shifts colour bars in the Y-direction.
widbar(nw)	defines the width of colour bars.
zaxis(za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a linearly scaled colour bar.
zaxlg(za, ze, zor, zstp, nl, cstr, it, id, nx, ny)	plots a logarithmically scaled colour bar.

Figure A.27: Coloured 3-D Graphics

A.28 3-D Graphics

Routine	Meaning
[xp,yp] = abs3pt(x, y, z)	converts absolute 3-D coordinates to plot coordinates.
axis3d(x, y, z)	defines the lengths of the 3-D box.
bars3d(xray, yray, z1ray, z2ray, xwray, ywray, icray, n)	plots 3-D bars.
box3d()	plots a border around the 3-D box.
cone3d(xm, ym, zm, r, h1, h2, n, m)	plots a cone.
conn3d(x, y, z)	plots a line to a point in 3-D space.
conshd3d(xray, n, yray, m, zmat, zlvray, nlev)	plots 3-D contours.
curv3d(xray, yray, zray, n)	plots curves or symbols.
curv4d(xray, yray, zray, wray, n)	plots coloured 3-d symbols.
cyli3d(xm, ym, zm, r, h, n, m)	plots a cylinder.
dbffin()	terminates a depth sort.
iret = dbfini()	initializes a depth depth sort.
dbfmod(cmod)	can disable the depth sort.
disk3d(xm, ym, zm, r1, r2, n, m)	plots a disk.
field3d(x1ray, y1ray, z1ray, x2ray, y2ray, z2ray, n, ivec)	plots a vector field.
flab3d()	disables the suppression of axis labels.
nclr = getlit(xp, yp, zp, xn, yn, zn)	calculates colour values.
getmat(xray, yray, zray, n, zmat, nx, ny, zv)	calculates a function matrix from randomly distributed data points.
graf3d(xa, xe, xor, xstp, ya, ye, yor, ystp, za, ze, zor, zstp)	plots an axis system.
grffin()	terminates a projection into 3-D space.
grfimg(cfil)	includes an image into 3-D space.
grfini(x1, y1, z1, x2, y2, z2, x3, y3, z3)	initializes projections in 3-D space.
grid3d(nx, ny, copt)	plots a grid.
hsym3d(xh)	sets the height of 3-D symbols.
n = isopts(xray, nx, yray, ny, zray, nz, wmat, wlev, xtri, ytri, ztri, nmax)	calculates isosurfaces.
labl3d(copt)	modifies the appearance of labels on the 3-D box.
light(cmode)	turns lighting on or off.
litmod(id, cmode)	turns single light sources on or off.
litop3(id, xr, xg, xb, ctype)	modifies light parameters.
litopt(id, xval, ctype)	modifies light parameters.
litpos(id, xp, yp, zp, copt)	sets the position of light sources.
matop3(xr, xg, xb, ctype)	modifies material parameters.
matopt(xval, ctype)	modifies material parameters.
mdfmat(ix, iy, w)	modifies the algorithm used in GETMAT.

Routine	Meaning
mshclr(iclr)	defines the colour of surface meshes.
mshcrv(n)	sets the resolution of meshes for 3-D curves.
nohide()	disables the hidden-line algorithm.
pike3d(x1, y1, z1, x2, y2, z2, r, nsk1, nsk2)	plots a cone.
plat3d(xm, ym, zm, xl, copt)	plots a Platonic solid.
plyfin(cfil, cobj)	terminates output of polygons to a PLY format.
plyini(copt)	initializes output of polygons to a PLY format.
[xp,yp,zp] = pos3pt(x, y, z)	converts user coordinates to absolute 3-D coordinates.
pyra3d(xm, ym, zm, xl, h1, h2, n)	plots a pyramid.
quad3d(xm, ym, zm, xl, yl, zl)	plots a quad.
[xp,yp] = rel3pt(x, y, z)	converts user coordinates to plot coordinates.
rot3d(a, b, c)	defines rotation angles for symbols and solids.
setfce(copt)	sets a face side for defining material parameters.
shlsur()	protects surfaces from overwriting.
sphe3d(xm, ym, zm, r, n, m)	plots a sphere.
n = stmpts3d(xv, yv, zv, nx, ny, nz, xpray, ypray, zpray, x0, y0, z0, xray, yray, zray, nmax)	generates a streamline.
stream3d(xv, yv, zv, nx, ny, nz, xpray, ypray, zpray, xsray, ysray, n)	plots streamlines.
strt3d(x, y, z)	moves the pen to a point.
surclr(itop, ibot)	selects surface colours.
surfce(xray, nx, yray, ny, zmat)	plots the surface of a function matrix.
surfcp(cfunc, a1, a2, astp, b1, b2, bstp)	plots the surface of a parametric function.
surfun(cfunc, ixp, xdel, iyp, ydel)	plots the surface grid of a function.
suriso(xray, nx, yray, ny, zray, nz, wmat, wlev)	plots isosurfaces.
surmat(zmat, nx, ny, ixpts, iypts)	plots the surface of a function matrix.
surmsh(copt)	enables grid lines for surfcp and surshd.
suropt(copt)	suppresses surface lines for surfce.
surshc(xray, nx, yray, ny, zmat, wmat)	plots a coloured surface.
surshd(xray, nx, yray, ny, zmat)	plots a coloured surface.
surtri(xray, yray, zray, n, i1ray, i2ray, i3ray, ntri)	plots the surface of an Delaunay triangulation.
survis(copt)	determines the visible part of surfaces.
symb3d(n, xm, ym, zm)	plots a 3-D symbol.
thkc3d(nthk)	defines the thickness of 3-D curves.
torus3d(xm, ym, zm, r1, r2, h, a1, a2, n, m)	plots a torus.
tube3d(x1, y1, z1, x2, y2, z2, r, n, m)	plots a tube.

Routine	Meaning
vang3d(ang)	defines the field of view.
vecf3d(xvray, yvray, zvray, xpray, ypray, zpray, n, ivec)	plots a vector field.
vectr3(x1, y1, z1, x2, y2, z2, ivec)	plots vectors in 3-D space.
vecmat3d(xv, yv, zv, nx, ny, nz, xpray, ypray, zpray, ivec)	plots a vector field on a regular grid.
vfoc3d(x, y, z, copt)	defines the focus point.
view3d(x, y, z, copt)	defines the viewpoint.
vscl3d(xfac)	sets a scaling factor for orthographic view.
vtx3d(xray, yray, zray, n, copt)	plots faces from vertices.
vtxc3d(xray, yray, zray, icray, n, copt)	plots faces from vertices.
vtxn3d(xray, yray, zray, xnray, ynray, znray, n, copt)	plots faces from vertices.
vup3d(ang)	defines the camera orientation.
zbfers()	erases the frame buffer of a Z-buffer.
zbffin()	terminates the Z-buffer.
iret = zbfini()	allocates space for a Z-buffer.
zbflin(x1, y1, z1, x2, y2, z2)	plots lines.
zbfmod(cmod)	can disable the Z-buffer.
zbfres()	resets the Z-buffer.
zbfscl(x)	scales the internal image for PDF output.
zbftri(xray, yray, zray, iray)	plots triangles.
zscale(zmin, zmax)	defines a Z-scaling for coloured surfaces.

Figure A.28: 3-D Graphics

A.29 Geographical Projections

Routine	Meaning
curvmp(xray, yray, n)	plots curves or symbols.
grafmp(xa, xe, xor, xstp, ya, ye, yor, ystp)	plots a geographical axis system.
gridmp(nx, ny)	plots a grid.
mapbas(copt)	defines a base map.
mapdir(cdir)	defines a search directory for map files.
mapfil(cfil, copt)	defines an external map file.
mapimg(cfil, x1, x2, x3, x4, x5, x6)	plots a BMP or GIF map image.
maplab(copt, ckey)	defines label options.
maplev(copt)	specifies land or lake plotting.
mapmod(copt)	modifies the connection of points used in CURVMP.
mapopt(copt, ckey)	defines map options.
mappol(xpol, ypol)	defines the map pole used for azimuthal projections.
mapref(ylw, yup)	defines two latitudes used for conical projections.
[xp,yp] = pos2pt(x, y)	converts user coordinates to plot coordinates.

Routine	Meaning
projct(copt)	selects a projection.
[xp,yp] = pt2pos(x, y)	converts plot coordinates to user coordinates.
shdafr(inray, ipray, icray, n)	shades African countries.
shdasi(inray, ipray, icray, n)	shades Asiatic countries.
shdaus(inray, ipray, icray, n)	shades Oceanic countries.
shdeur(inray, ipray, icray, n)	shades European countries.
shdmap(copt)	shades continents.
shdnor(inray, ipray, icray, n)	shades states of North and Central America.
shdsou(inray, ipray, icray, n)	shades states of South America.
shdusa(inray, ipray, icray, n)	shades USA states.
world()	plots coastlines and lakes.
xaxmap(xa, xe, xor, xstp, cstr, nt, ny)	plots a secondary X-axis.
yaxmap(ya, ye, yor, ystp, cstr, nt, nx)	plots a secondary Y-axis.

Figure A.29: Geographical Projections

A.30 Contouring

Routine	Meaning
conclr(ncray, n)	defines colours for shaded contours.
concrv(xray, yray, n, z)	plots generated contours.
confll(xray, yray, zray, n, i1ray, i2ray, i3ray, ntri, zlvray, nlev)	plots filled contours of an Delaunay triangulation.
congap(xfac)	affects the spacing between contour lines and labels.
conlab(copt)	defines a character string used for contour labels.
conmat(zmat, nx, ny, z)	plots contours.
conmod(xfac, xquot)	affects the position of contour labels.
ncrv = conpts(xray, n, yray, m, zmat, zlev, xpray, ypray, maxpts, iray, maxcrv)	generates contours.
conshd(xray, nx, yray, ny, zmat, zlray, n)	plots shaded contours.
conshd2(xmat, ymat, zmat, nx, ny, zlray, n)	plots shaded contours.
contri(xray, yray, zray, n, i1ray, i2ray, i3ray, ntri, zlev)	plots contours of an Delaunay triangulation.
contur(xray, nx, yray, ny, zmat, zlev)	plots contours.
contur2(xmat, ymat, zmat, nx, ny, zlev)	plots contours.

Routine	Meaning
labclr(nclr, 'CONT')	defines the colour of contour labels.
labdis(ndis, 'CONT')	defines the distance between labels.
labels(copt, 'CONT')	defines contour labels.
shdmod(copt, 'CONT')	sets the algorithm for shaded contours.
ncrv = tripts(xray, yray, zray, n, i1ray, i2ray, i3ray, ntri, zlev, xpray, ypray, maxpts, iray, maxcrv)	generates contours from triangulated data.

Figure A.30: Contouring

A.31 Image Routines

Routine	Meaning
expimg(cfil, copt)	copies an image from memory to a file.
imgbox(nx, ny, nw, nh)	defines a rectangle for PostScript/PDF output.
imgclp(nx, ny, nw, nh)	defines a clipping rectangle.
imgfin()	terminates transfering of image data.
imgini()	initializes transfering of image data.
imgmod(cmod)	selects index or RGB mode.
imgsiz(nw, nh)	defines an image size for PostScript/PDF output.
imgtpr(nclr)	defines a transparency colour for images.
cbuf = rbfpng()	stores an image as PNG file in a buffer.
rbmp(cfil)	stores an image as a BMP file.
rgif(cfil)	stores an image as a GIF file.
rimage(cfil)	copies an image from memory to a file.
iclr = rpixel(ix, iy)	reads a pixel from memory.
cray = rpixls(ix, iy, nw, nh)	reads image data from memory.
rpng(cfil)	stores an image as a PNG file.
rppm(cfil)	stores an image as a PPM file.
cray = rpxrow(nx, ny, n)	reads a row of image data from memory.
rtiff(cfil)	stores an image as a TIFF file.
tiforg(nx, ny)	defines the position of TIFF files copied with WTIFF.
tifwin(nx, ny, nw, nh)	defines a clipping window for TIFF files.
wimage(cfil)	copies an image from file to memory.
wpixel(ix, iy, iclr)	writes a pixel to memory.
wpixls(cray, ix, iy, nw, nh)	writes image data to memory.
wpxrow(cray, nx, ny, n)	write a row of image data to memory.
wtiff(cfil)	copies a TIFF file created by Dislin to memory.

Figure A.31: Image Routines

A.32 Window Routines

Routine	Meaning
clswin(id)	closes a window.
hidwin(id, copt)	defines whether a window is visible or not.
opnwin(id)	opens a window for graphics output.
pagwin(nxp, nyp)	defines page formats for windows.
selwin(id)	selects a window for graphics output.
winapp(capp)	defines a window or console application.
wincbk(crout, copt)	defines a callback routine for the windows size.
window(nx, ny, nw, nh)	defines the position and size of windows.
winico(cstr)	loads an icon for the windows title bar.
id = winid()	returns the ID of the currently selected window.
winjus(copt)	defines the position of the graphics window.
winkey(ckey)	defines a key that can be used for program continuation in DISFIN.
winmod(copt)	affects the handling of windows in DISFIN.
winsiz(nw, nh)	defines the size of windows.
wintit(cstr)	sets the title of the currently selected window.
wintyp(copt)	sets the type of the graphics window.
x11mod(copt)	enables backing store.

Figure A.32: Window Routines

A.33 Widget Routines

Routine	Meaning
doevnt()	processes pending events.
ival = dwgbut(cstr, ival)	displays a message that can be answered with 'Yes' or 'No'.
iret = dwgerr()	returns a status for dialog widget routines.
cfil = dwgfil(clab, cfil, cmask)	creates a file selection box.
isel = dwglis(clab, clis, isel)	gets a selection from a list of items.
dwgmsg(cstr)	displays a message.
cstr = dwgtxt(clab, cstr)	prompts an user for input.
n = gwgatt(id, copt)	requests widget attributes.
n = gwgbox(id)	requests the value of a box widget.
n = gwgbut(id)	requests the status of a button widget.
cfil = gwgfil(id)	requests the value of a file widget.
x = gwgflt(id)	requests the value of a text widget as real number.
n = gwggui()	returns the used GUI.
n = gwgint(id)	requests the value of a text widget as integer.
n = gwglis(id)	requests the value of a list widget.
x = gwgscl(id)	requests the value of a scale widget.

Routine	Meaning
[nw,nh] = gwgsiz(id)	returns the size of widgets.
x = gwgtbf(id, irow, icol)	requests the value of a table cell as real number.
ix = gwgtbi(id, irow, icol)	requests the value of a table cell as integer.
xray = gwgtbl(id, n, idx, copt)	requests the values of table cells.
cstr = gwgtbs(id, irow, icol)	requests the value of a table cell as a string.
cstr = gwgtxt(id)	requests the value of a text widget.
n = gwgxid(id)	requests the windows ID of a widget.
clis = itmcat(clis, n, citem)	concatenates an element to a list string.
n = itmcnt(clis)	calculates the number of elements in a list string.
citem = itmstr(clis, n)	extracts an element from a list string.
msgbox(cstr)	displays a message.
swgatt(id, catt, copt)	sets widget attributes.
swgbgd(id, xr, xg, xb)	changes the background colour of widgets.
swgbox(id, isel)	changes the selection of a box widget.
swgbut(id, ival)	changes the status of a button widget.
swgcb2(id, routine)	connects a callback routine with a table widget.
swgcbk(id, routine)	connects a callback routine with a widget.
swgclr(xr, xg, xb, copt)	sets widget colours.
swgdrw(xf)	defines the height of draw widgets.
swgfgd(id, xr, xg, xb)	changes the foreground colour of widgets.
swgfil(id, cfil)	changes the value of a file widget.
swgflt(id, x, ndig)	changes the value of text widgets.
swgfnt(cfnt, npts)	defines widget fonts.
swgfoc(id)	sets the keyboard focus.
swghlp(cstr)	sets a character string for the Help menu.
swgint(id, n)	changes the value of text widgets.
swgiop(i, copt)	sets integer options for widgets.
swgjus(cjus, class)	defines the alignment of label widgets.
swglis(id, isel)	changes the selection of a list widget.
swgmix(char, cmix)	defines control characters.
swgmrg(ival, cmrg)	defines widget margins.
swgopt(copt, ckey)	sets a center option for parent widgets.
swgpop(copt)	modifies the appearance of the popup menubar.
swgpos(nx, ny)	defines the position of widgets.
swgray(xray, n, copt)	defines the width of columns in table widgets.
swgscl(id, xval)	changes the value of a scale widget.
swgsiz(nw, nh)	defines the size of widgets.
swgspc(xspc, yspc)	modifies the spaces between widgets.
swgstp(xstp)	defines a step value for scale widgets.
swgtbf(id, x, ndig, i, j, copt)	changes the values of table cells.
swgtbi(id, ix, i, j, copt)	changes the values of table cells.
swgtbl(id, xray, n, ndig, idx, copt)	changes the values of table cells.

Routine	Meaning
swgtbs(id, cstr, i, j, copt)	changes the values of table cells.
swgtit(cstr)	sets a title for the main widget.
swgtxt(id, cval)	changes the value of a text widget.
swgtyp(ctype, class)	modifies the appearance of widgets.
swgval(id, xval)	changes the value of progress bars.
swgwin(nx, ny, nw, nh)	defines the position and size of widgets.
swgwth(nwth)	sets the default width of widgets.
id = wgapp(ip, clab)	creates an entry in a popup menu.
id = wgappb(ip, iray, nw, nh)	uses an image as entry in a popup menu.
id = wgbas(ip, copt)	creates a container wdiget.
id = wgbox(ip, clis, isel)	creates a list widget where the list elements are displayed as toggle buttons.
id = wgbut(ip, cval, ival)	creates a button widget.
id = wgcmd(ip, clab, cmd)	creates a push button widget for a system command.
id = wgdlis(ip, clis, isel)	creates a dropping list widget.
id = wgdraw(ip)	creates a draw widget.
id = wgfil(ip, clab, cfil, cmask)	creates a file widget.
wgfin()	terminates widget routines.
id = wgicon(ip, clab, nw, nh, cfl)	creates a label widget with an icon as label.
id = wgimg(ip, clab, iray, nw, nh)	creates a label widget with an image as label.
id = wgini(copt)	creates a main widget and initalizes widget routines.
id = wglab(ip, cstr)	creates a label widget.
id = wglis(ip, clis, isel)	creates a list widget.
id = wgltxt(ip, clab, cstr, nwth)	creates a labeled text widget.
id = wgok(ip)	creates an OK push button widget.
id = wgpbar(ip, x1, x2, xstp)	creates a progress bar.
id = wgpbut(ip, clab)	creates a push button widget.
id = wgpicon(ip, clab, nw, nh, cfl)	creates a push button with an icon as label.
id = wgpimg(ip, clab, iray, nw, nh)	creates a push button with an image as label.
id = wgpop(ip, cstr)	creates a popup menu.
id = wgpopb(ip, iray, nw, nh)	uses an image as a popup menu.
id = wgquit(ip)	creates a Quit push button widget.
id = wgscl(ip, clab, xmin, xmax, xval, ndez)	creates a scale widget.
id = wgsep(ip)	creates a separator widget.
id = wgstxt(ip, nsize, nmax)	creates a scrolled text widget.
id = wgtbl(ip, nrows, ncols)	creates a table widget.
id = wgtxt(ip, cstr)	creates a text widget.

Figure A.33: Widget Routines

A.34 Dislin Quickplots

Routine	Meaning
qplbar(xray, n)	plots a bar graph.
qplclr(zmat, n, m)	plots a coloured surface.
qplcon(zmat, n, m, nlv)	makes a contour plot.
qplcrv(xray, yray, n, copt)	plots multiple curves.
qplot(xray, yray, n)	plots a curve.
qplpie(xray, yray, n)	plots a pie chart.
qplsca(xray, yray, n)	makes a scatter plot.
qplscl(a, e, or, step, copt)	sets a user-defined scaling.
qplsur(zmat, n, m)	plots a surface.

Figure A.34: Dislin Quickplots

A.35 Using Threads

Routine	Meaning
thrfin()	terminates threads.
thrini(n)	enables threads.

Figure A.35: Using Threads

A.36 Reading FITS Files

Routine	Meaning
fitscls()	closes a FITS file.
x = filtsflt(ckey)	returns the floatingpoint value of a key.
n = fitsimg(iray)	copies a FITS image to an array.
istat = fitsopn(cfil)	opens a FITS file for reading.
cval = fitsstr(ckey)	returns the string value of a key.
n = fitstyp(ckey)	returns the type of a key.
n = fitsval(ckey)	returns the integer value of a key.

Figure A.36: Reading FITS Files

A.37 MPS Logo

Routine	Meaning
mpslogo(nx, ny, nsize, copt)	plots the MPS logo.

Figure A.37: MPS Logo

Appendix B

Examples

This appendix presents some examples of the Dislin manual in Ruby coding. They can be found in the Dislin subdirectory ruby.

B.1 Demonstration of CURVE

```
#!/usr/bin/ruby
require 'dislin'

n = 101
pi = 3.1415926
f = pi / 180.0
step = 360.0 / (n - 1)

xray = Array.new(n)
y1ray = Array.new(n)
y2ray = Array.new(n)
for i  in 0..n-1
  xray[i] = i * step
  x = xray[i] * f
  y1ray[i] = Math.sin(x)
  y2ray[i] = Math.cos(x)
end

Dislin.metafl('xwin')
Dislin.scrmod('revers')
Dislin.disini()
Dislin.complx()
Dislin.pagera()

Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')
Dislin.axspos(450, 1800)
Dislin.axslen(2200, 1200)

Dislin.labdig(-1, 'X')
Dislin.ticks(10, 'XY')
Dislin.titlin('Demonstration of CURVE', 1)
Dislin.titlin('SIN (X), COS (X)', 3)

ic = Dislin.intrgb(0.95, 0.95, 0.95)
Dislin.axsbgd(ic)
Dislin.graf(0.0, 360.0, 0.0, 90.0, -1.0, 1.0, -1.0, 0.5)
Dislin.setrgb(0.7, 0.7, 0.7)
Dislin.grid(1,1)

Dislin.color('fore')
Dislin.box2d()
Dislin.height(50)
Dislin.title()

Dislin.color('red')
Dislin.curve(xray, y1ray, n)
Dislin.color('green')
Dislin.curve(xray, y2ray, n)
Dislin.disfin()
```

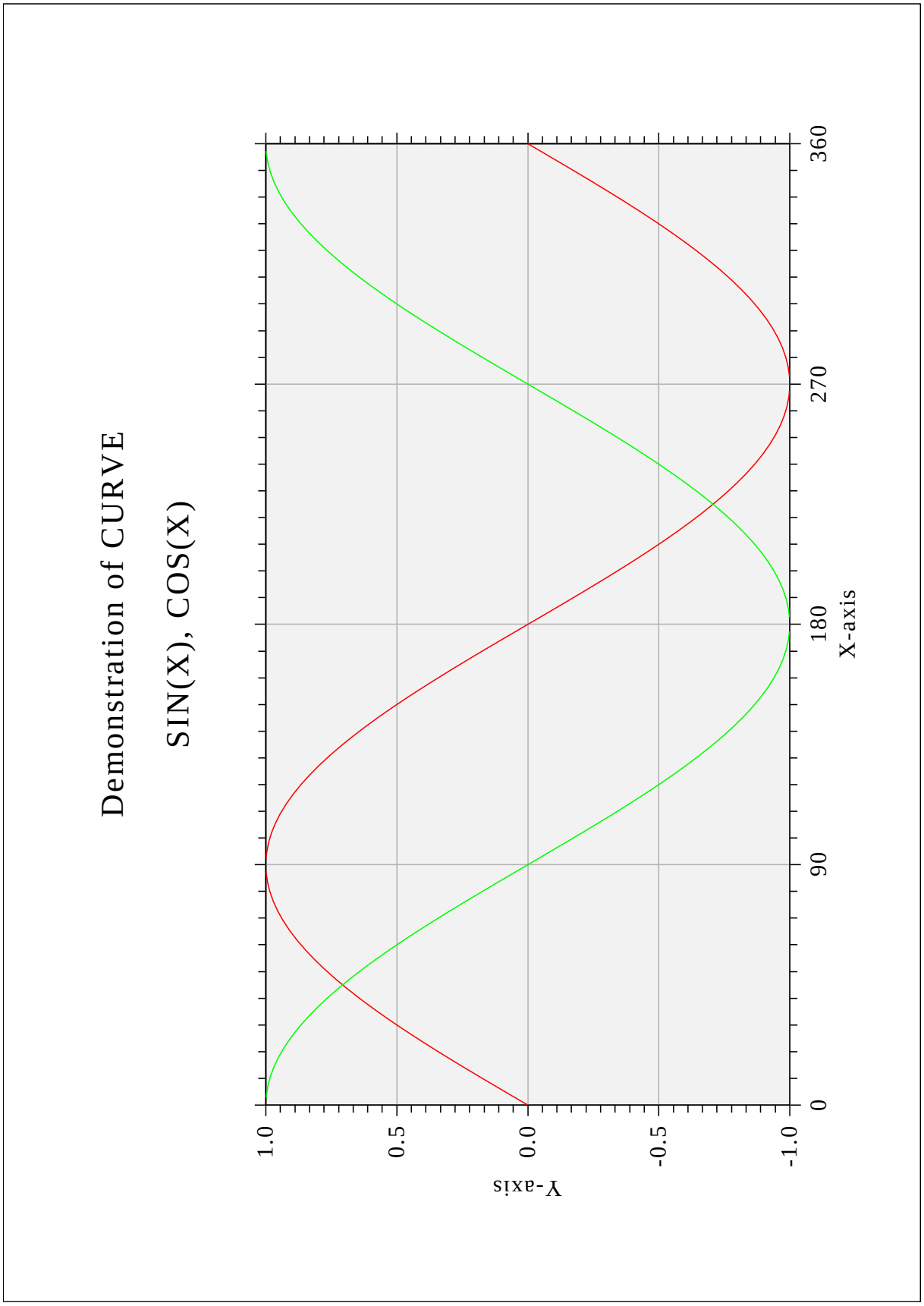

Figure B.1: Demonstration of CURVE

B.2 Symbols

```
#!/usr/bin/ruby
require 'dislin'

ctit = 'Symbols'

Dislin.setpag('da4p')
Dislin.metafl('cons')
Dislin.disini()
Dislin.complx()
Dislin.pagera()

Dislin.paghdr('H. Michels  (', ')', 2, 0)

Dislin.height(60)
nl = Dislin.nlmess(ctit)
Dislin.messag(ctit, (2100 - nl)/2, 200)

Dislin.height(50)
Dislin.hsymbl(120)

ny = 150
for i in 0..21
  if (i % 4) == 0
    ny = ny + 400
    nxp = 550
  else
    nxp = nxp + 350
  end

  nl = Dislin.nlnumb(i, -1)
  Dislin.number(i, -1, nxp - nl/2, ny + 150)
  Dislin.symbol(i, nxp, ny)
end
Dislin.disfin()
```

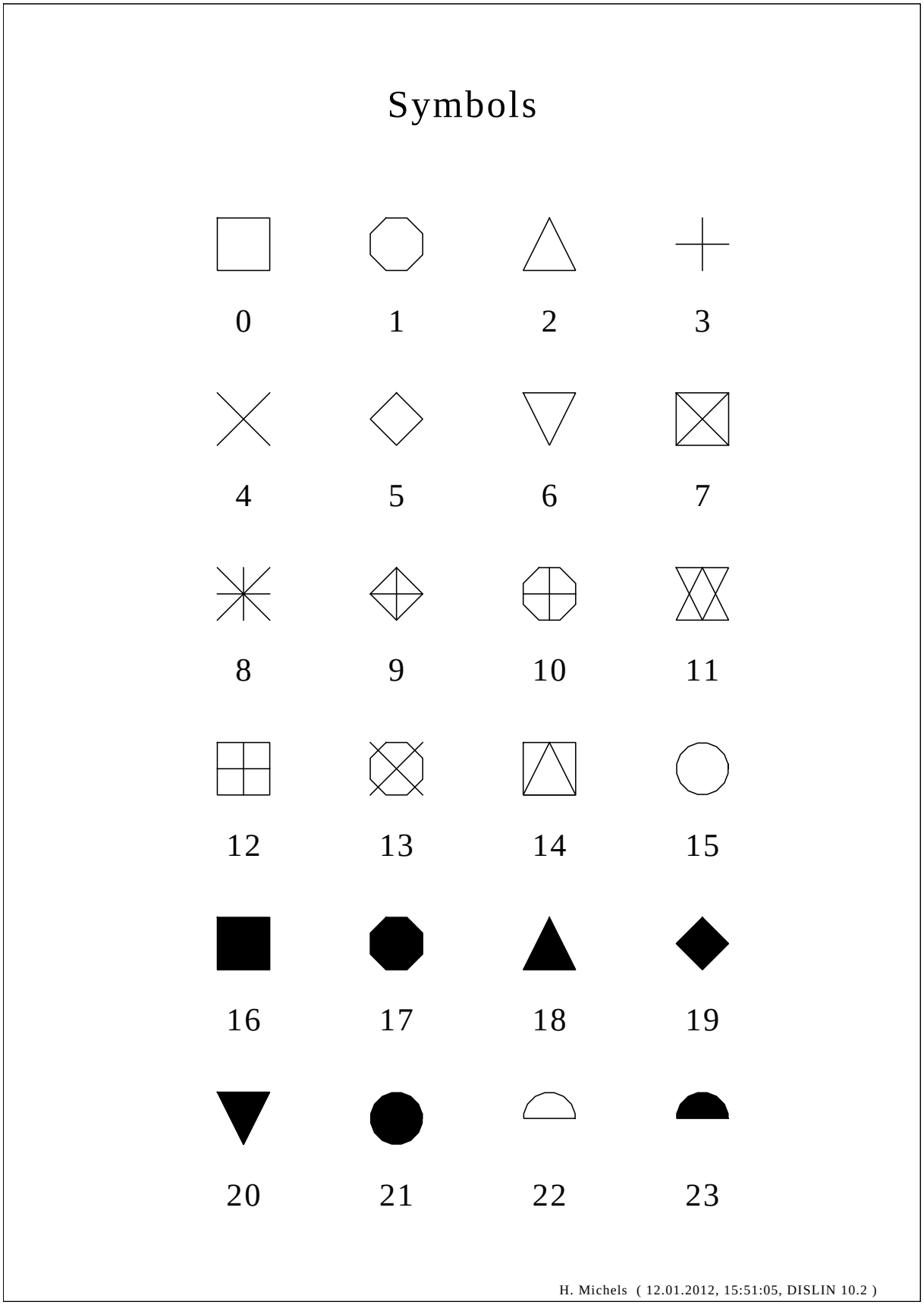

Figure B.2: Symbols

B.3 Logarithmic Scaling

```
#!/usr/bin/ruby
require 'dislin'

ctit = 'Logarithmic Scaling'
clab = ['LOG', 'FLOAT', 'ELOG']

Dislin.setpag('da4p')
Dislin.metafl('cons')

Dislin.disini()
Dislin.pagera()
Dislin.complx()
Dislin.axslen(1400, 500)

Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')
Dislin.axsscl('LOG', 'XY')

Dislin.titlin(ctit, 2)

for i in 0..2
  nya = 2650 - i * 800
  Dislin.labdig(-1, 'XY')
  if i == 1
    Dislin.labdig(1, 'Y')
    Dislin.name(' ', 'X')
  end

  Dislin.axspos(500, nya)
  Dislin.messag('Labels: ' + clab[i], 600, nya - 400)
  Dislin.labels(clab[i], 'XY')
  Dislin.graf(0.0, 3.0, 0.0, 1.0, -1.0, 2.0, -1.0, 1.0)
  if i == 2
    Dislin.height(50)
    Dislin.title()
  end

  Dislin.endgrf()
end

Dislin.disfin()
```

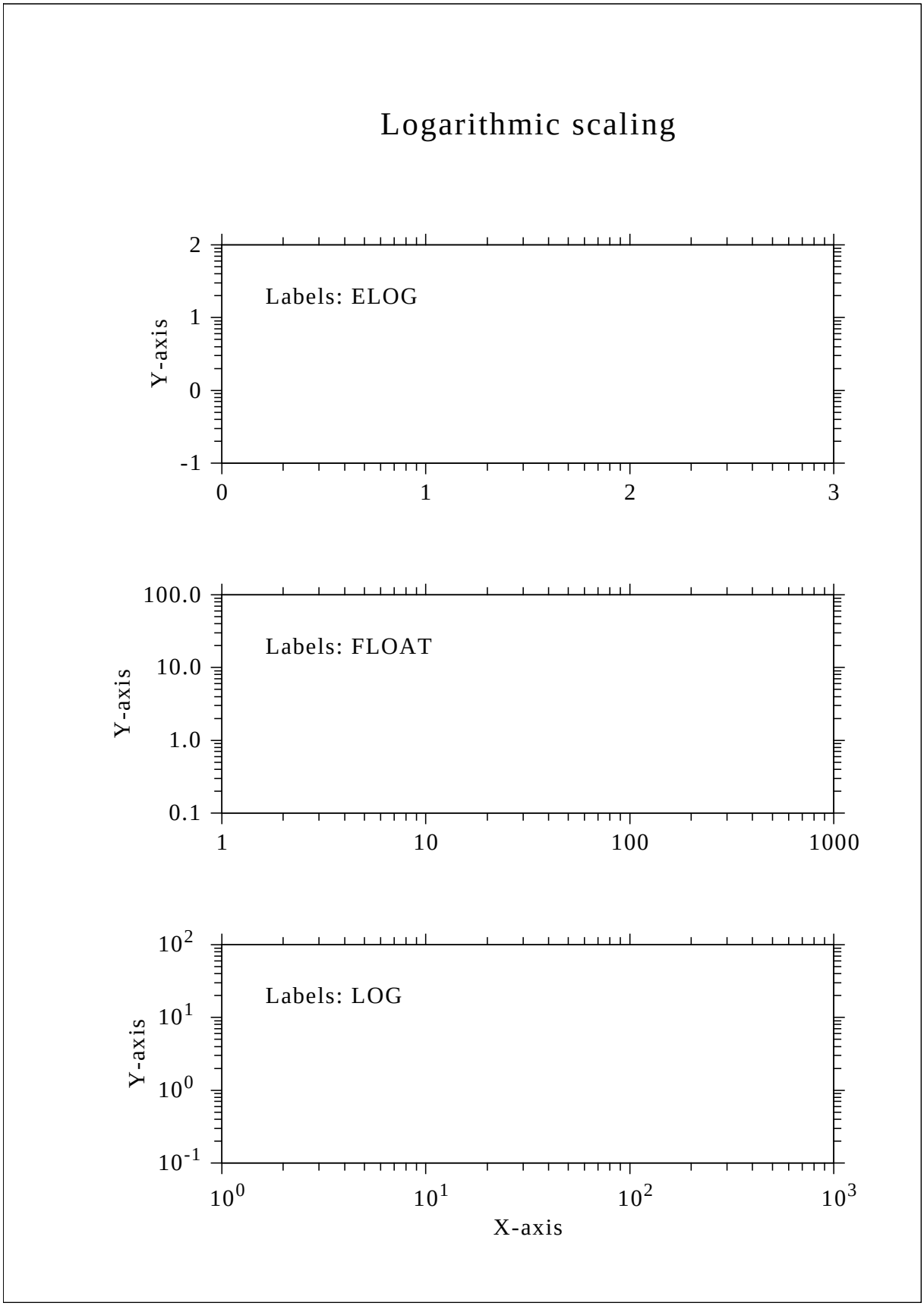

Figure B.3: Logarithmic Scaling

B.4 Interpolation Methods

```
#!/usr/bin/ruby
require 'dislin'

ctit = 'Interpolation Methods'

xray = [0.0, 1.0, 3.0, 4.5, 6.0, 8.0, 9.0, 11.0, 12.0, 12.5,
        13.0, 15.0, 16.0, 17.0, 19.0, 20.0]
yray = [2.0, 4.0, 4.5, 3.0, 1.0, 7.0, 2.0, 3.0, 5.0, 2.0, 2.5,
        2.0, 4.0, 6.0, 5.5, 4.0]
cpol = ['SPLINE', 'STEM', 'BARS', 'STAIRS', 'STEP', 'LINEAR']

Dislin.setpag('da4p')
Dislin.metafl('cons')

Dislin.disini()
Dislin.pagera()
Dislin.complx()

Dislin.incmrk(1)
Dislin.hsymbl(25)
Dislin.titlin(ctit, 1)
Dislin.axslen(1500, 350)
Dislin.setgrf('LINE', 'LINE', 'LINE', 'LINE')

nya = 2700
for i in 0..5
  Dislin.axspos(350, nya - i * 350)
  Dislin.polcrv(cpol[i])
  Dislin.marker(0)
  Dislin.graf(0.0, 20.0, 0.0, 5.0, 0.0, 10.0, 0.0, 5.0)
  nx = Dislin.nxposn(1.0)
  ny = Dislin.nyposn(8.0)
  Dislin.messag(cpol[i], nx, ny)
  Dislin.curve(xray, yray, 16)

  if i == 5
    Dislin.height(50)
    Dislin.title()
  end

  Dislin.endgrf()
end

Dislin.disfin()
```

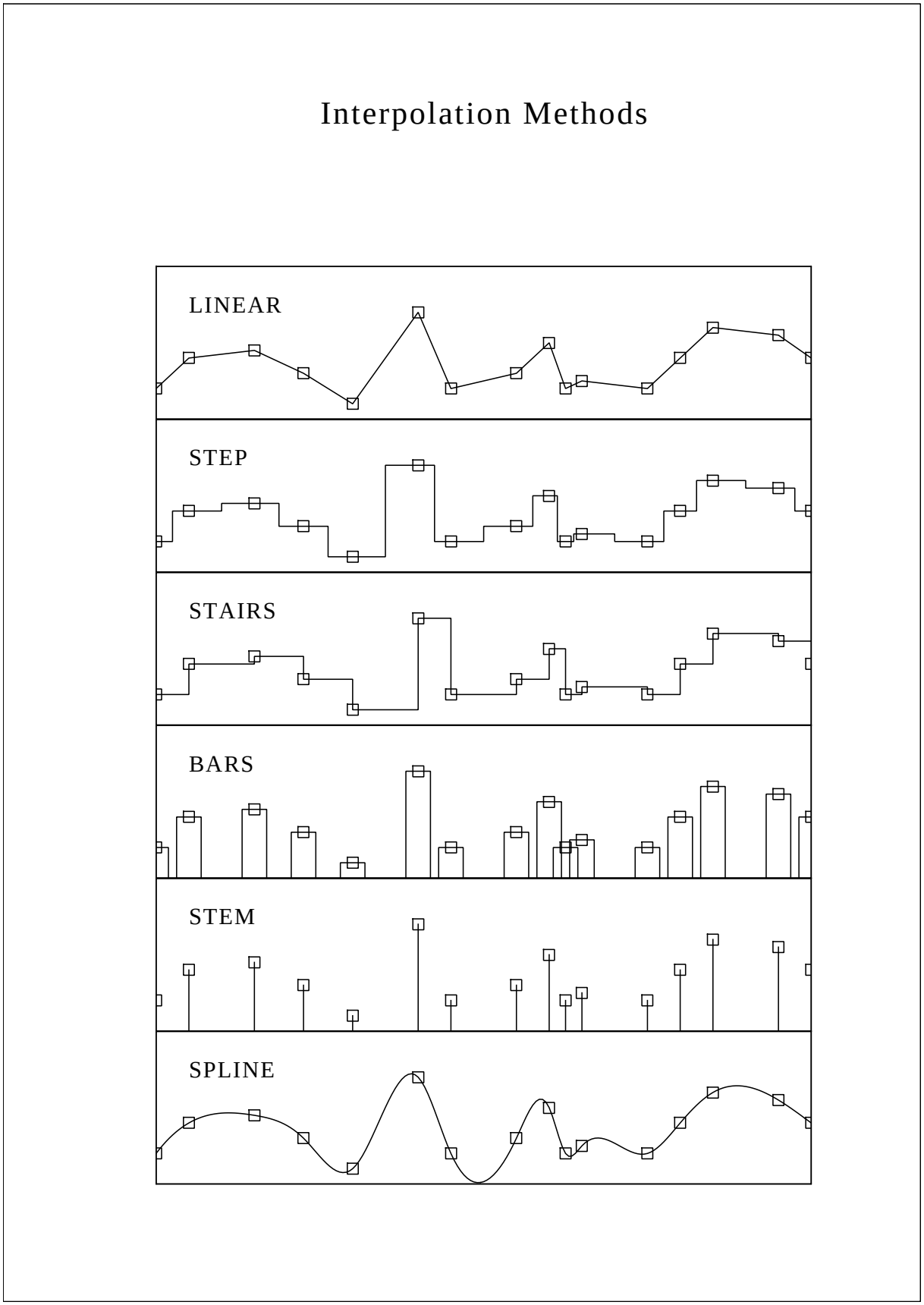

Figure B.4: Interpolation Methods

B.5 Line Styles

```
#!/usr/bin/ruby
require 'dislin'

ctit1 = 'Demonstration of CURVE'
ctit2 = 'Line Styles'

ctyp = ['SOLID', 'DOT', 'DASH', 'CHNDSH',
        'CHNDOT', 'DASHM', 'DOTL', 'DASHL']
x = [3.0, 9.0]
y = [0.0, 0.0]

Dislin.metafl('cons')
Dislin.setpag('da4p')

Dislin.disini()
Dislin.pagera()
Dislin.complx()
Dislin.center()

Dislin.chncrv('BOTH')
Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')

Dislin.titlin(ctit1, 1)
Dislin.titlin(ctit2, 3)

Dislin.graf(0.0, 10.0, 0.0, 2.0, 0.0, 10.0, 0.0, 2.0)
Dislin.title()

for i in 0..7
  y[0] = 8.5 - i
  y[1] = 8.5 - i
  nx = Dislin.nxposn(1.0)
  ny = Dislin.nyposn(y[0])
  Dislin.messag(ctyp[i], nx, ny - 20)
  Dislin.curve(x, y, 2)
end

Dislin.disfin()
```

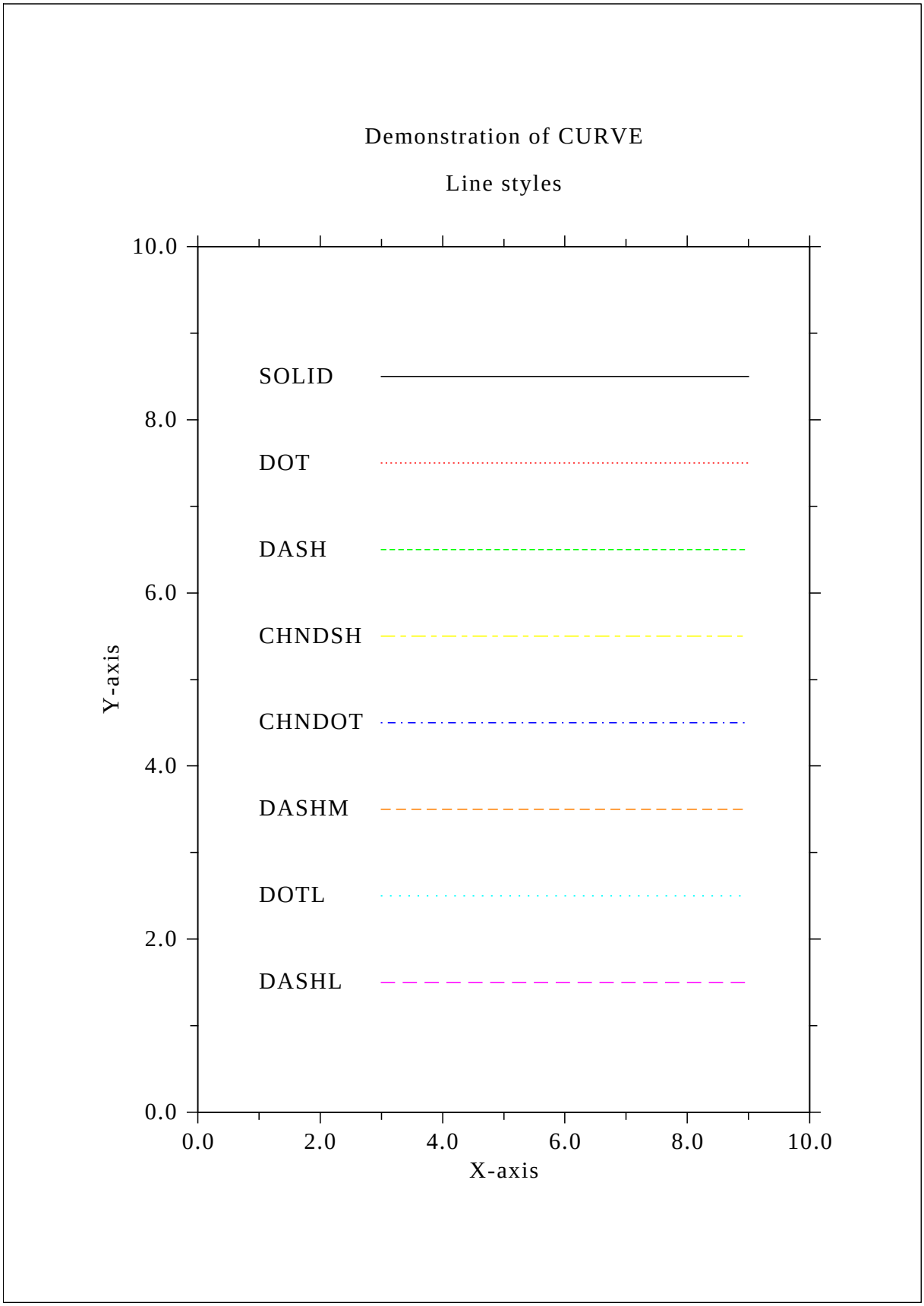

Figure B.5: Line Styles

B.6 Legends

```
#!/usr/bin/ruby
require 'dislin'

n = 101
f = 3.1415926 / 180.0
step = 360.0 / (n - 1)
xray = Array.new(n)
y1ray = Array.new(n)
y2ray = Array.new(n)

for i in 0..n-1
  xray[i] = i * step
  x = xray[i] * f
  y1ray[i] = Math.sin(x)
  y2ray[i] = Math.cos(x)
end

Dislin.metafl('xwin')
Dislin.disini()
Dislin.complx()

Dislin.axspos(450, 1800)
Dislin.axslen(2200, 1200)
Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')

Dislin.labdig(-1, 'X')
Dislin.ticks(10, 'XY')
Dislin.titlin('Demonstration of CURVE', 1)
Dislin.titlin('Legend', 3)

Dislin.graf(0.0, 360.0, 0.0, 90.0, -1.0, 1.0, -1.0, 0.5)
Dislin.title()
Dislin.xaxgit()
Dislin.chncrv('BOTH')
Dislin.curve(xray, y1ray, n)
Dislin.curve(xray, y2ray, n)

cbuf = ' '
Dislin.legini(cbuf, 2, 7)     # cbuf is a dummy parameter for python
nx = Dislin.nxposn(190.0)
ny = Dislin.nyposn(0.75)
Dislin.leglin(cbuf, 'sin(x)', 1)
Dislin.leglin(cbuf, 'cos(x)', 2)
Dislin.legpos(nx, ny)
Dislin.legtit('Legend')
Dislin.legend(cbuf, 3)
Dislin.disfin()
```

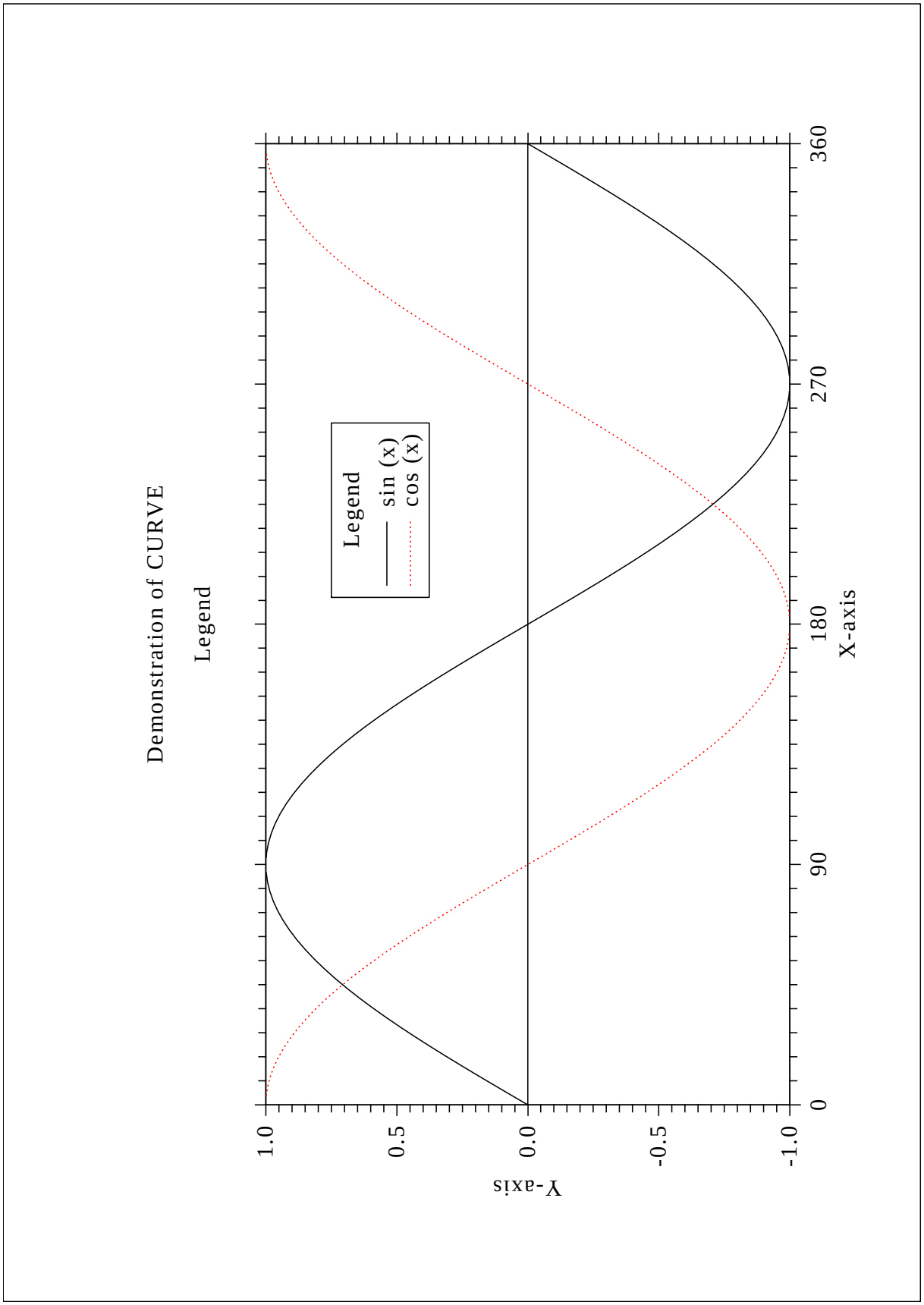

Figure B.6: Legends

B.7 Shading Patterns (AREAF)

```
#!/usr/bin/ruby
require 'dislin'

ix  = [0, 300, 300,   0]
iy  = [0,   0, 400, 400]
ixp = [0, 0, 0, 0]
iyp = [0, 0, 0, 0]

Dislin.metafl('cons')
Dislin.disini()
Dislin.setvlt('small')
Dislin.pagera()
Dislin.complx()

Dislin.height(50)
ctit = 'Shading patterns (AREAF)'
nl = Dislin.nlmess(ctit)
Dislin.messag(ctit, (2970 - nl)/2, 200)

nx0 = 335
ny0 = 350

iclr = 0
for i in 0..2
  ny = ny0 + i * 600

  for j in 0..5
    nx = nx0 + j * 400
    ii = i * 6 + j
    Dislin.shdpat(ii)
    iclr = iclr + 1
    iclr = iclr % 8
    if iclr == 0
      iclr = 8
    end
    Dislin.setclr(iclr)
    for k in 0..3
      ixp[k] = ix[k] + nx
      iyp[k] = iy[k] + ny
    end

    Dislin.areaf(ixp, iyp, 4)
    nl = Dislin.nlnumb(ii, -1)
    nx = nx + (300 - nl) / 2
    Dislin.color('foreground')
    Dislin.number(ii, -1, nx, ny + 460)
  end
end

Dislin.disfin()
```

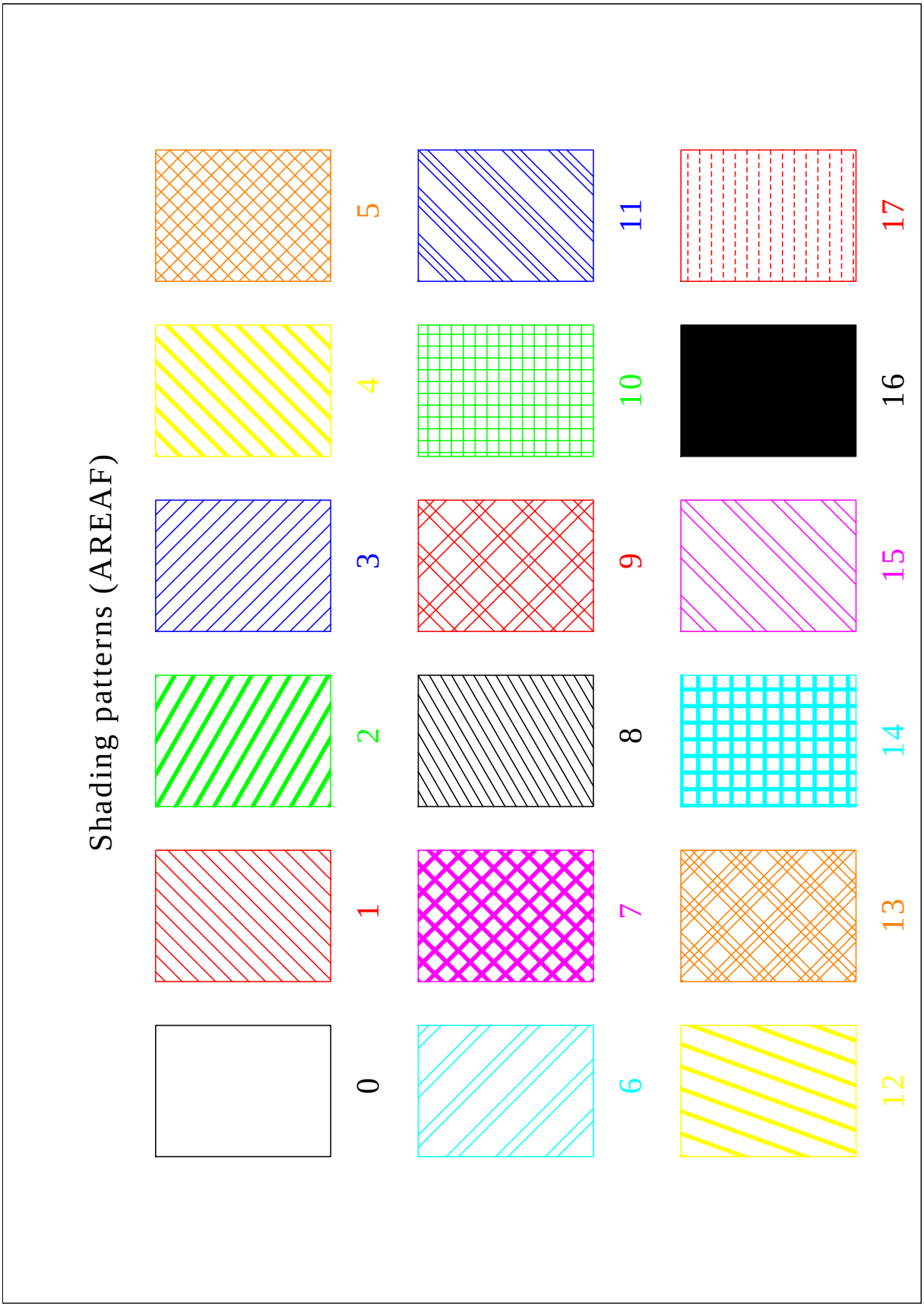

Figure B.7: Shading Patterns

B.8 Vectors

```
#!/usr/bin/ruby
require 'dislin'

ivec  = [0, 1111, 1311, 1421, 1531, 1701, 1911,
          3111, 3311, 3421, 3531, 3703, 4221, 4302,
          4413, 4522, 4701, 5312, 5502, 5703]

ctit = 'Vectors'

Dislin.metafl('cons')
Dislin.disini()
Dislin.pagera()
Dislin.complx()

Dislin.height(60)
nl = Dislin.nlmess(ctit)
Dislin.messag(ctit, (2970 - nl)/2, 200)

Dislin.height(50)
nx = 300
ny = 400

for i in 0..19
  if i == 10
    nx = nx + 2970 / 2
    ny = 400
  end

  nl = Dislin.nlnumb(ivec[i], -1)
  Dislin.number(ivec[i], -1, nx - nl, ny - 25)

  Dislin.vector(nx + 100, ny, nx + 1000, ny, ivec[i])
  ny = ny + 160
end

Dislin.disfin()
```

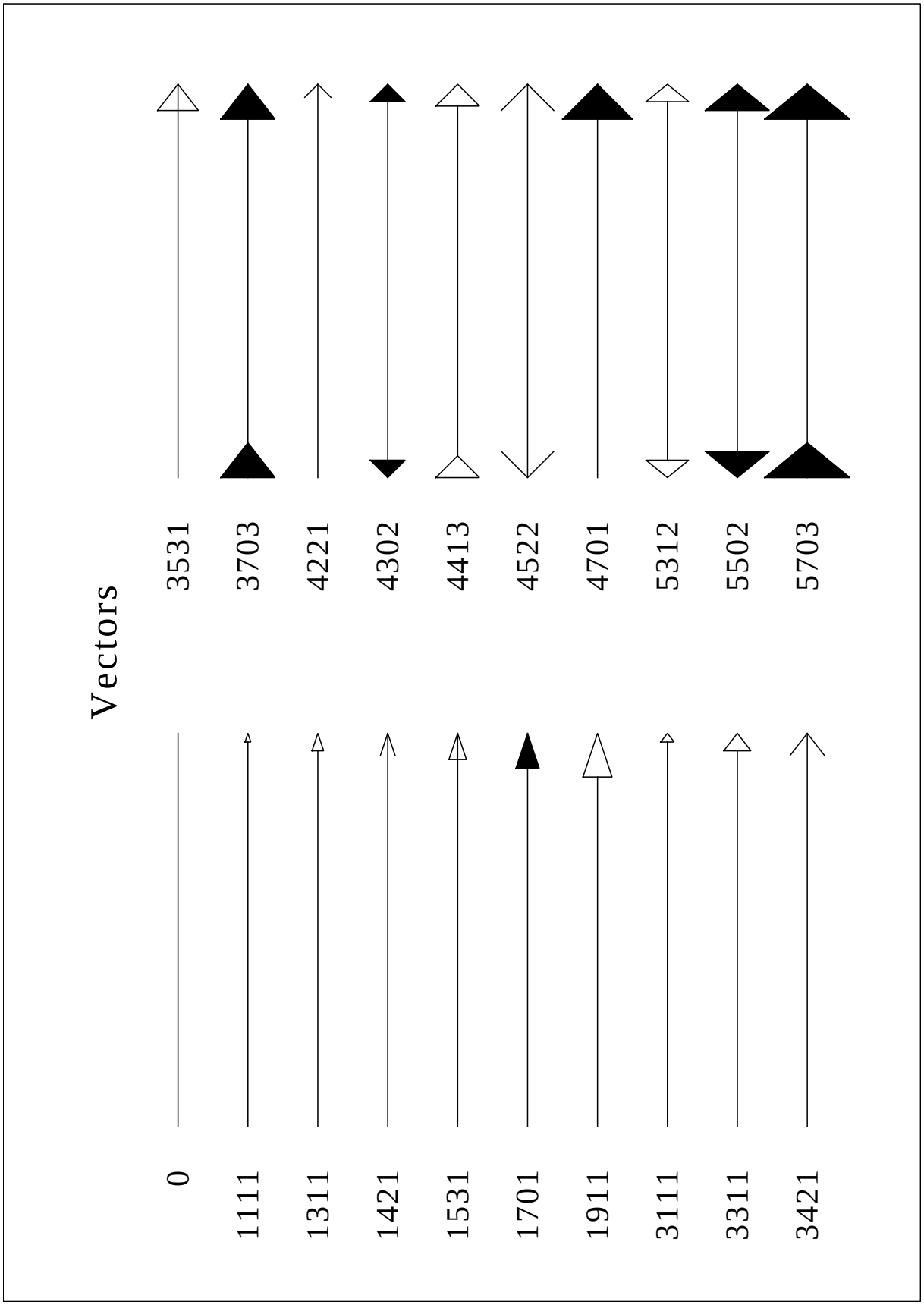

Figure B.8: Vectors

B.9 3-D Colour Plot

```
#!/usr/bin/ruby
require 'dislin'

ctit1 = '3-D  Colour Plot of the Function'
ctit2 = 'F(X,Y) = 2 * SIN(X) * SIN (Y)'

n = 50
m = 50
zmat = Array.new(n*m)

fpi  = 3.1415927 / 180.0
stepx = 360.0 / (n - 1)
stepy = 360.0 / (m - 1)

for i in 0..n-1
  x = i * stepx
  for j in 0..m-1
    y = j * stepy
    zmat[i*m+j] = 2 * Math.sin(x * fpi) * Math.sin(y * fpi)
  end
end

Dislin.metafl('xwin')
Dislin.disini()
Dislin.pagera()
Dislin.complx()

Dislin.titlin(ctit1, 1)
Dislin.titlin(ctit2, 3)

Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')
Dislin.name('Z-axis', 'Z')

Dislin.intax()
Dislin.autres(n, m)
Dislin.axspos(300, 1850)
Dislin.ax3len(2200, 1400, 1400)

Dislin.graf3(0.0, 360.0, 0.0, 90.0, 0.0, 360.0, 0.0, 90.0,
                -2.0, 2.0, -2.0, 1.0)
Dislin.crvmat(zmat, n, m, 1, 1)
Dislin.height(50)
Dislin.title()
Dislin.disfin()
```

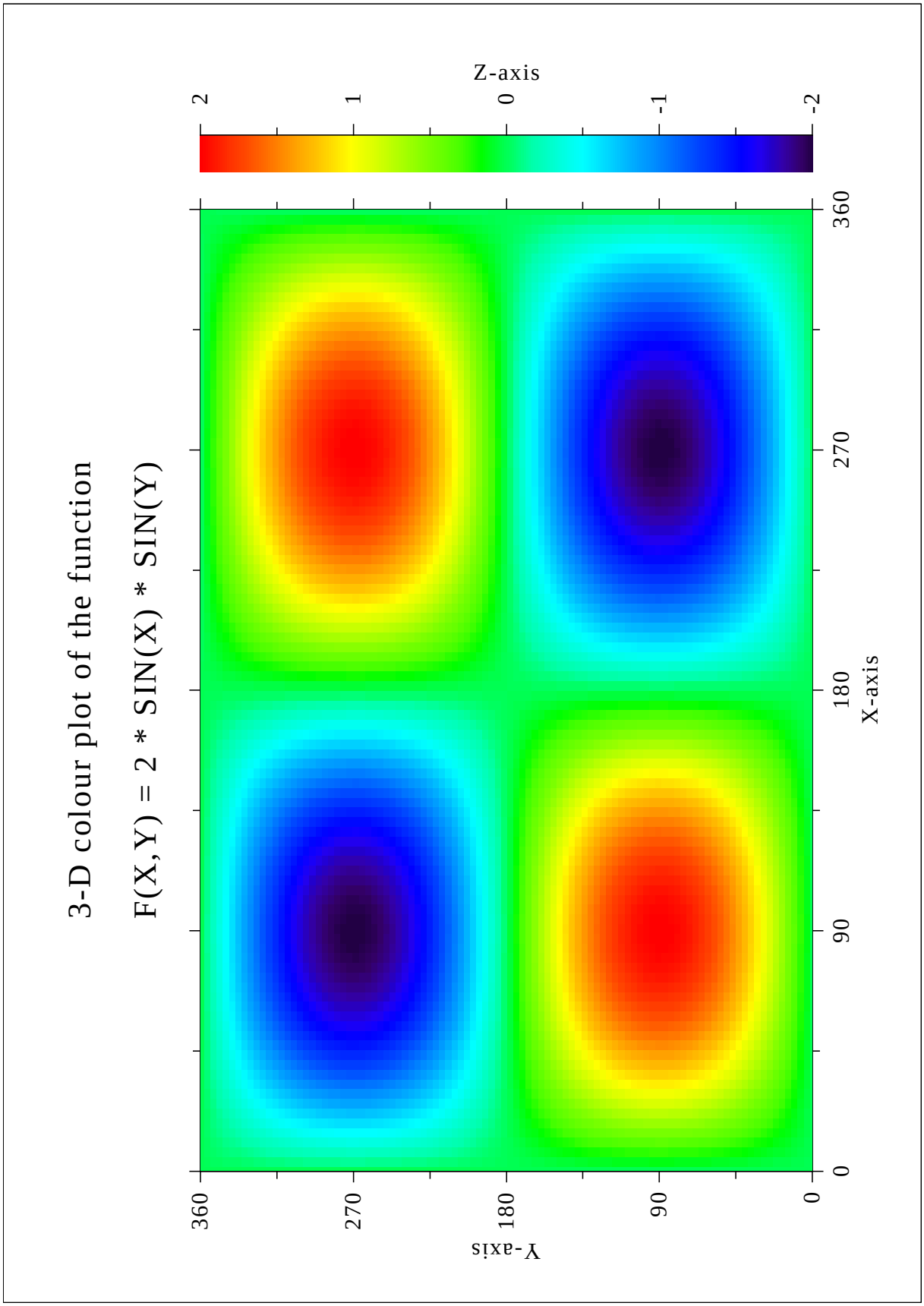

Figure B.9: 3-D Colour Plot

B.10 Surface Plot

```
#!/usr/bin/ruby
require 'dislin'

ctit1 = 'Surface Plot of the Function'
ctit2 = 'F(X,Y) = 2 * SIN(X) * SIN(Y)'

n = 50
m = 50
zmat = Array.new(n*m)

fpi  = 3.1415927 / 180.0
stepx = 360.0 /(n - 1)
stepy = 360.0 /(m - 1)

for i in 0..n-1
  x = i * stepx
  for j in 0..m-1
    y = j * stepy
    zmat[i*m+j] = 2 * Math.sin(x * fpi) * Math.sin(y * fpi)
  end
end

Dislin.metafl('cons')
Dislin.setpag('da4p')
Dislin.disini()
Dislin.pagera()
Dislin.complx()

Dislin.titlin(ctit1, 2)
Dislin.titlin(ctit2, 4)

Dislin.axspos(200, 2600)
Dislin.axslen(1800, 1800)

Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')
Dislin.name('Z-axis', 'Z')

Dislin.view3d(-5.0, -5.0, 4.0, 'ABS')
Dislin.graf3d(0.0, 360.0, 0.0, 90.0, 0.0, 360.0, 0.0, 90.0,
                -3.0, 3.0, -3.0, 1.0)
Dislin.height(50)
Dislin.title()

Dislin.color('green')
Dislin.surmat(zmat, n, m, 1, 1)
Dislin.disfin()
```

Surface plot (SURMAT)

F(X,Y) = 2*SIN(X)*SIN(Y)

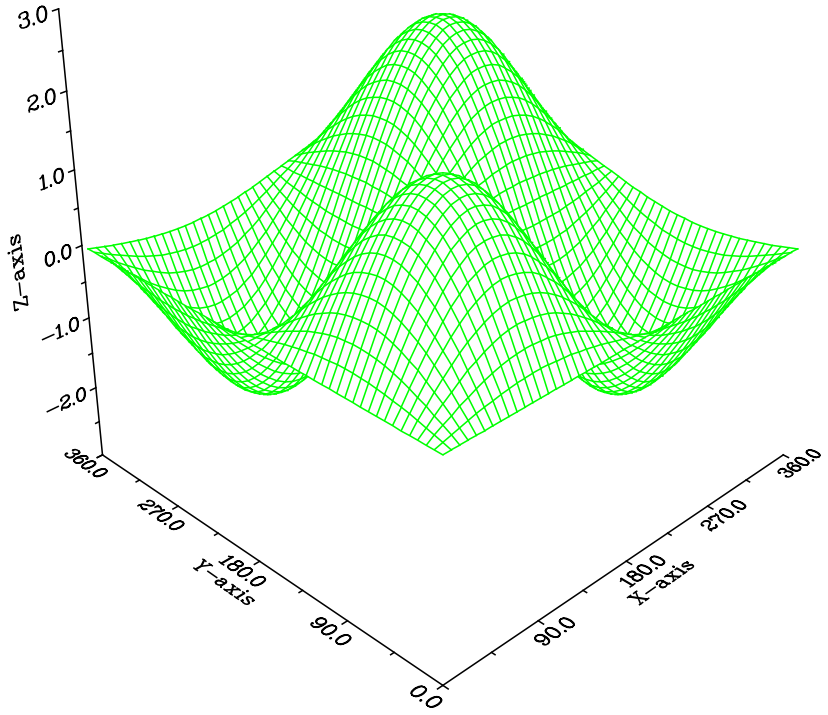

Figure B.10: Surface Plot

B.11 Surface Plot

```
#!/usr/bin/ruby
require 'dislin'

def myfunc(x, y, iopt)
  if iopt == 1
     xv = Math.cos(x)*(3+Math.cos(y))
  elsif iopt == 2
     xv = Math.sin(x)*(3+Math.cos(y))
  else
     xv = Math.sin(y)
  end

  return xv
end

ctit1 = 'Surface Plot of the Parametric Function'
ctit2 = '[COS(t)*(3+COS(u)), SIN(t)*(3+COS(u)), SIN(u)]'

pi  = 3.1415927

Dislin.metafl('cons')
Dislin.setpag('da4p')
Dislin.disini()
Dislin.pagera()
Dislin.complx()

Dislin.titlin(ctit1, 2)
Dislin.titlin(ctit2, 4)

Dislin.axspos(200, 2400)
Dislin.axslen(1800, 1800)

Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')
Dislin.name('Z-axis', 'Z')
Dislin.intax()

Dislin.vkytit(-300)
Dislin.zscale(-1.0,1.0)
Dislin.surmsh('on')

Dislin.graf3d(-4.0,4.0,-4.0,1.0,-4.0,4.0,-4.0,1.0,-3.0,3.0,-3.0,1.0)
Dislin.height(40)
Dislin.title()

step = 2 * pi / 30.0
Dislin.surfcp('myfunc', 0.0, 2*pi, step, 0.0, 2*pi, step)
Dislin.disfin()
```

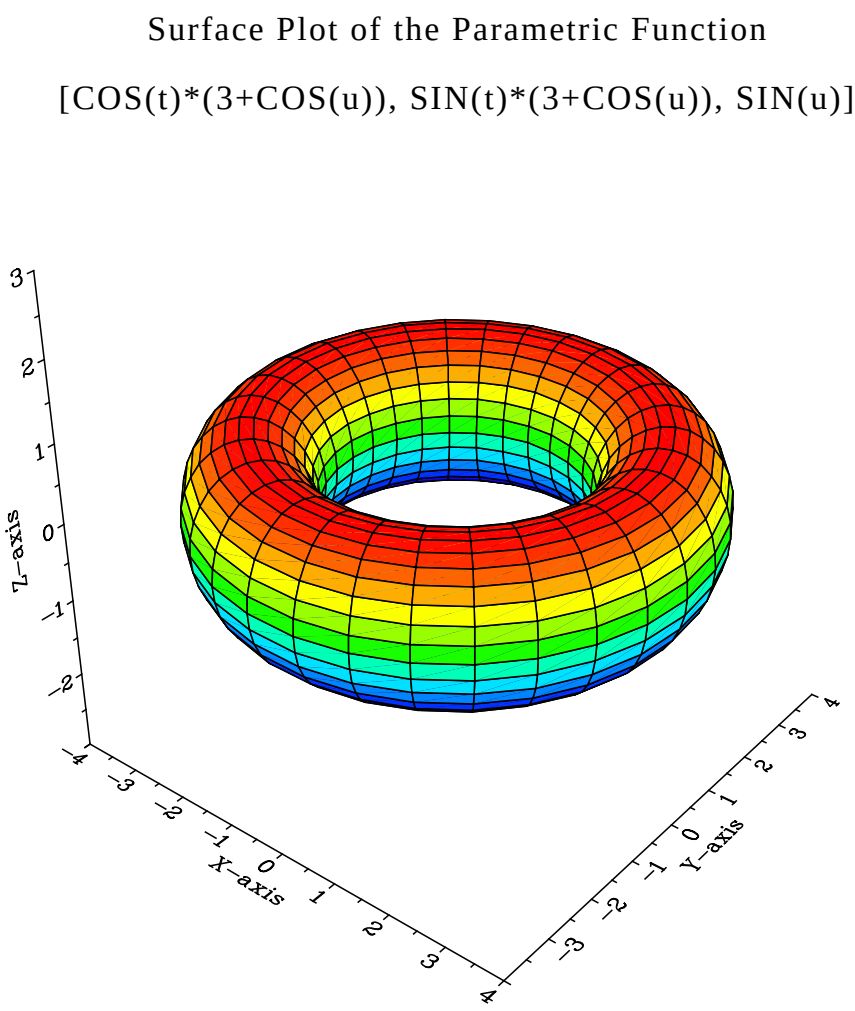

Figure B.11: Surface Plot of a Parametric Function

B.12 Polar Plots

```
#!/usr/bin/ruby
require 'dislin'

n = 300
m = 10
pi = 3.1415926
f = pi / 180.0
step = 360.0 / (n - 1)

xray = Array.new(n)
x1 = Array.new(n)
y1 = Array.new(n)

x2 = Array.new(m)
y2 = Array.new(m)

for i  in 0..n-1
  a = (i * step) * f
  xray[i] = i * step
  x = xray[i] * f
  y1[i] = a
  x1[i] = Math.sin(5 * a)
end

for i  in 0..m-1
  x2[i] = i + 1
  y2[i] = i + 1
end

Dislin.setpag('da4p')
Dislin.scrmod('revers')
Dislin.metafl('cons')
Dislin.disini()
Dislin.complx()
Dislin.pagera()

Dislin.titlin('Polar Plots', 2)
Dislin.ticks(3, 'Y')
Dislin.axends('NOENDS', 'X')
Dislin.labdig(-1, 'Y')
Dislin.axslen(1000, 1000)
Dislin.axsorg(1050, 900)

ic = Dislin.intrgb(0.95,0.95,0.95)
Dislin.axsbgd(ic)
Dislin.grafp(1.0, 0.0, 0.2, 0.0, 30.0);
Dislin.color('blue')
Dislin.curve(x1, y1, n)
Dislin.color('fore')
Dislin.htitle(50)
```

```
Dislin.title()
Dislin.endgrf()

Dislin.labdig(-1, 'X')
Dislin.axsorg(1050, 2250)
Dislin.labtyp('VERT', 'Y')
Dislin.grafp(10.0, 0.0, 2.0, 0.0, 30.0)
Dislin.barwth(-5.0)
Dislin.polcrv('FBARS')
Dislin.color('blue')
Dislin.curve(x2, y2, m)

Dislin.disfin()
```

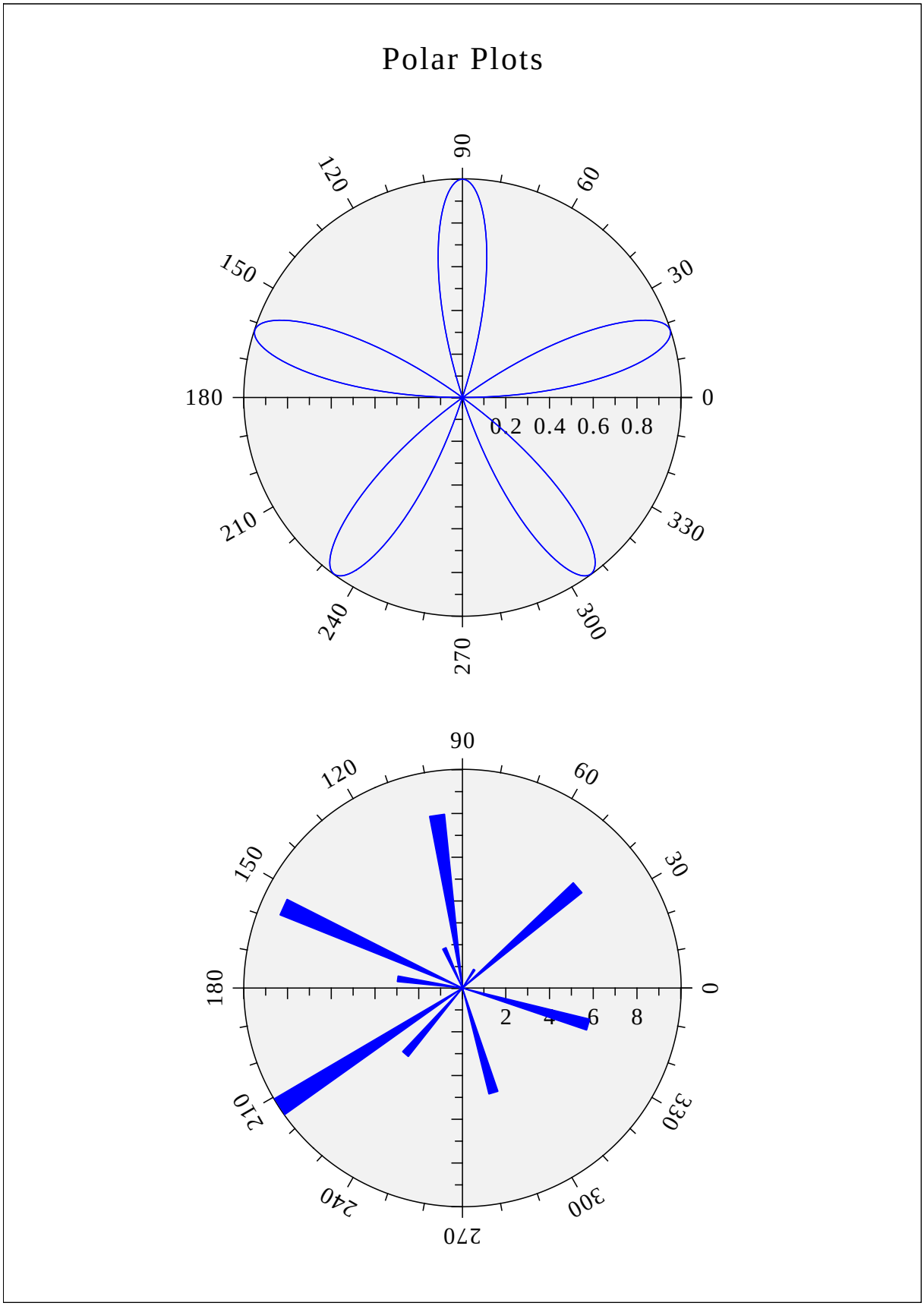

Figure B.12: Polar Plots

B.13 Contour Plot

```
#!/usr/bin/ruby
require 'dislin'

ctit1 = 'Contour Plot'
ctit2 = 'F(X,Y) = 2 * SIN(X) * SIN(Y)'

n = 50
m = 50
xray = Array.new(n)
yray = Array.new(m)
zlev = Array.new(12)
zmat = Array.new(n * m)

fpi  = 3.1415927 / 180.0
stepx = 360.0 /(n - 1)
stepy = 360.0 /(m - 1)

for i in 0..n-1
  xray[i] = i * stepx
end

for i in 0..m-1
  yray[i] = i * stepy
end

for i in 0..n-1
  x = xray[i] * fpi
  for j in 0..m-1
    y = yray[j] * fpi
    zmat[i*m+j] = 2 * Math.sin(x) * Math.sin(y)
  end
end

Dislin.metafl('cons')
Dislin.setpag('da4p')
Dislin.disini()
Dislin.pagera()
Dislin.complx()

Dislin.titlin(ctit1, 1)
Dislin.titlin(ctit2, 3)

Dislin.intax()
Dislin.axspos(450, 2650)

Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')

Dislin.graf(0.0, 360.0, 0.0, 90.0, 0.0, 360.0, 0.0, 90.0)
Dislin.height(50)
```

```
Dislin.title()

Dislin.height(30)
for i in 0..8
  zlev = -2.0 + i * 0.5
  if i == 4
    Dislin.labels('NONE', 'CONTUR')
  else
    Dislin.labels('FLOAT', 'CONTUR')
  end

  Dislin.setclr((i+1) * 28)
  Dislin.contur(xray, n, yray, m, zmat, zlev)
end

Dislin.disfin()
```

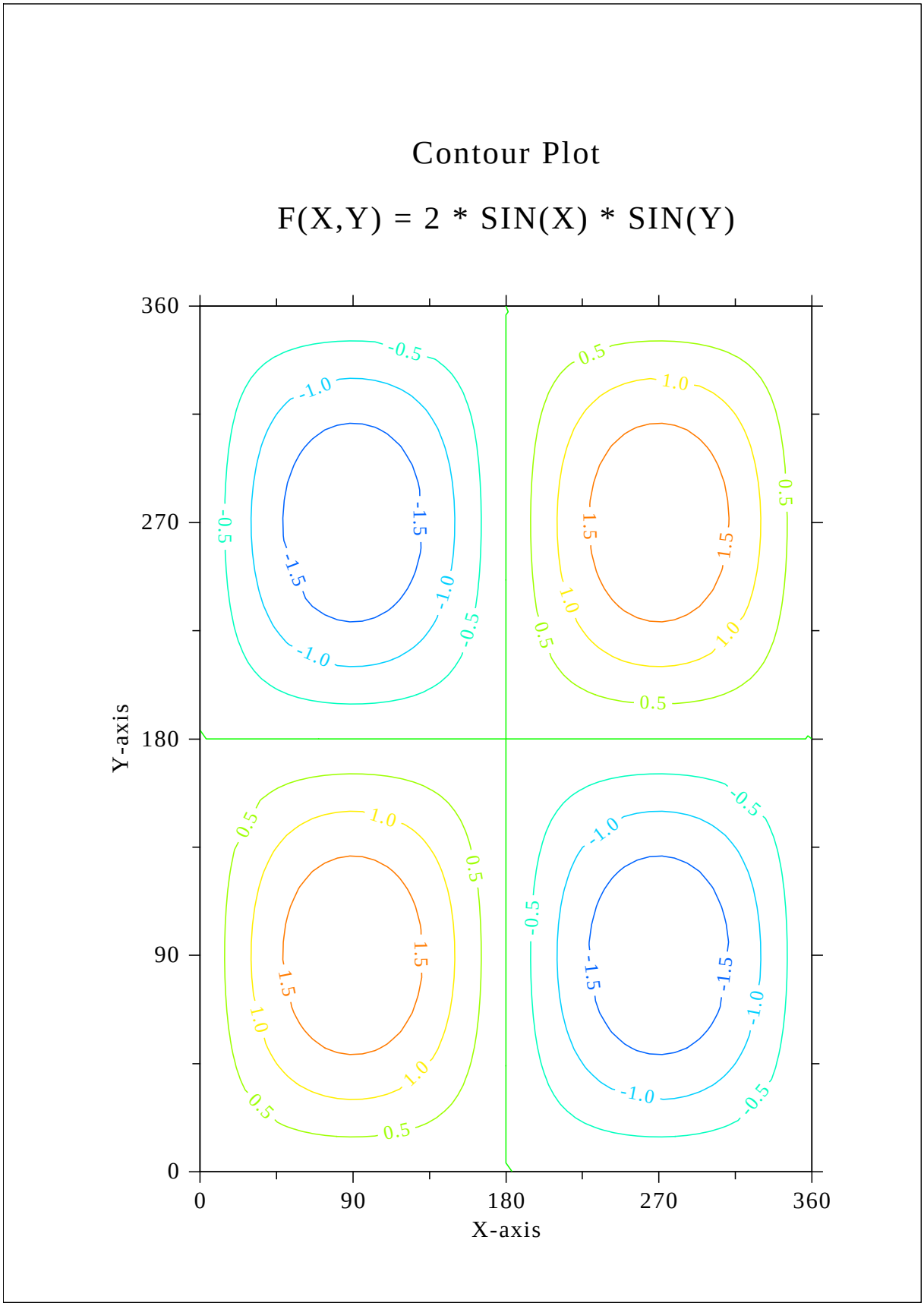

Figure B.13: Contour Plot

B.14 Shaded Contour Plot

```
#!/usr/bin/ruby
require 'dislin'

n = 50
m = 50
xray = Array.new(n)
yray = Array.new(m)
zlev = Array.new(12)
zmat = Array.new(n * m)
stepx = 1.6 /(n - 1)
stepy = 1.6 /(m - 1)

for i in 0..n-1
  xray[i] = i * stepx
end
for i in 0..m-1
  yray[i] = i * stepy
end
for i in 0..n-1
  x = xray[i] * xray[i] - 1.0
  x = x * x
  for j in 0..m-1
    y = yray[j] * yray[j] - 1.0
    zmat[i*m+j] = x + y * y
  end
end

Dislin.metafl('cons')
Dislin.setpag('da4p')
Dislin.disini()
Dislin.pagera()
Dislin.complx()
Dislin.mixalf()

Dislin.titlin('Shaded Contour Plot', 1)
Dislin.titlin('F(X,Y) =(X[2$ - 1)[2$ +(Y[2$ - 1)[2$', 3)
Dislin.name('X-axis', 'X')
Dislin.name('Y-axis', 'Y')
Dislin.axspos(450, 2670)
Dislin.shdmod('poly', 'contur')
Dislin.graf(0.0, 1.6, 0.0, 0.2, 0.0, 1.6, 0.0, 0.2)

for i in 0..11
  zlev[11-i] = 0.1 + i * 0.1
end
Dislin.conshd(xray, n, yray, m, zmat, zlev, 12)

Dislin.height(50)
Dislin.title()
Dislin.disfin()
```

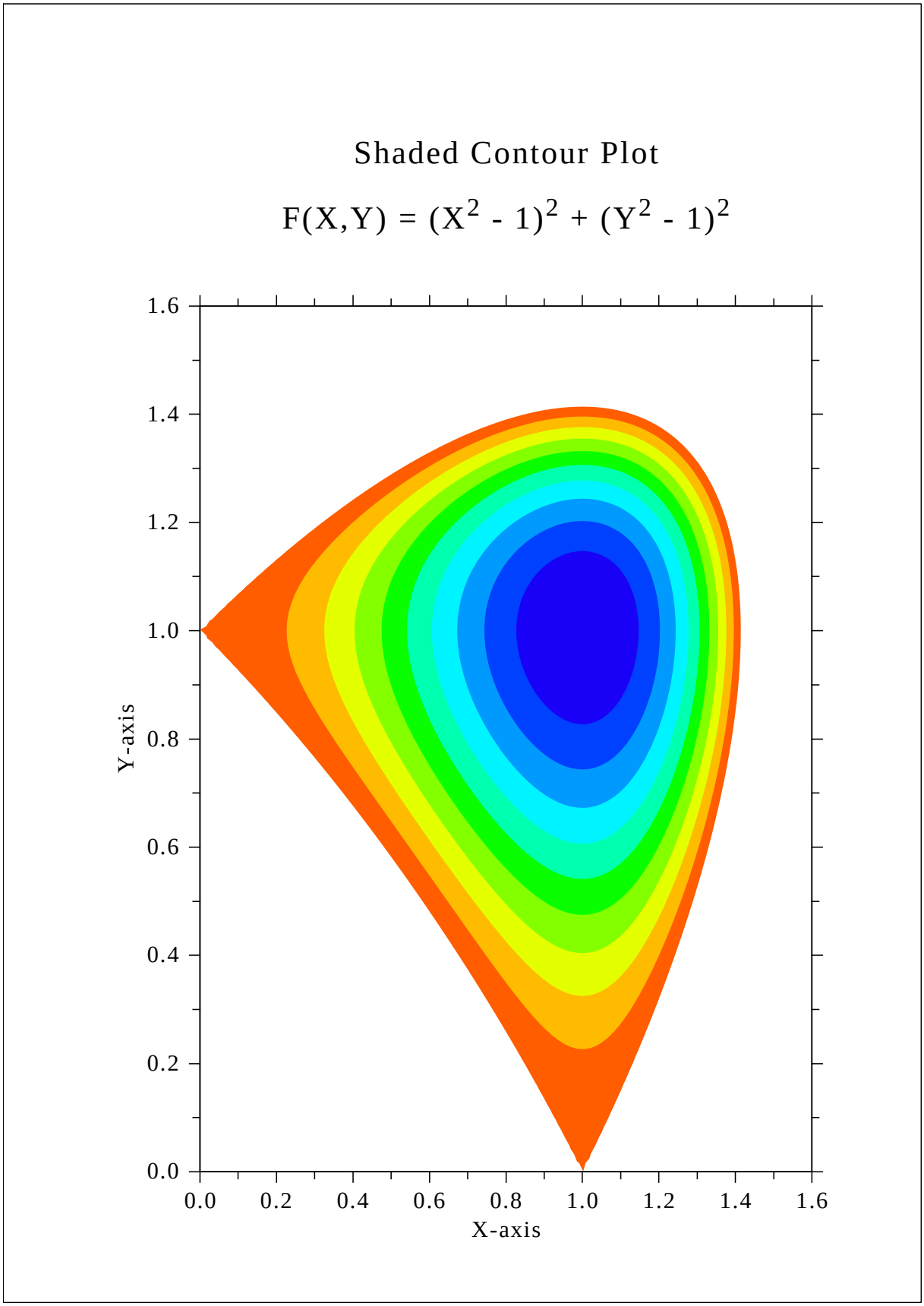

Figure B.14: Shaded Contour Plot

B.15 Pie Charts

```
#!/usr/bin/ruby
require 'dislin'

xray = [1.0, 2.5, 2.0, 2.7, 1.8]

ctit = 'Pie Charts(PIEGRF)'

Dislin.setpag('da4p')
Dislin.metafl('cons')
Dislin.disini()
Dislin.pagera()
Dislin.complx()
Dislin.chnpie('BOTH')

Dislin.axslen(1600, 1000)
Dislin.titlin(ctit, 2)

cbuf = ' '
Dislin.legini(cbuf, 5, 8)
Dislin.leglin(cbuf, 'FIRST',  1)
Dislin.leglin(cbuf, 'SECOND', 2)
Dislin.leglin(cbuf, 'THIRD',  3)
Dislin.leglin(cbuf, 'FOURTH', 4)
Dislin.leglin(cbuf, 'FIFTH',  5)

# Selecting shading patterns
Dislin.patcyc(1, 7)
Dislin.patcyc(2, 4)
Dislin.patcyc(3, 13)
Dislin.patcyc(4, 3)
Dislin.patcyc(5, 5)

Dislin.axspos(250, 2800)
Dislin.piegrf(cbuf, 1, xray, 5)
Dislin.endgrf()

Dislin.axspos(250, 1600)
Dislin.labels('DATA', 'PIE')
Dislin.labpos('EXTERNAL', 'PIE')
Dislin.piegrf(cbuf, 1, xray, 5)

Dislin.height(50)
Dislin.title()
Dislin.disfin()
```

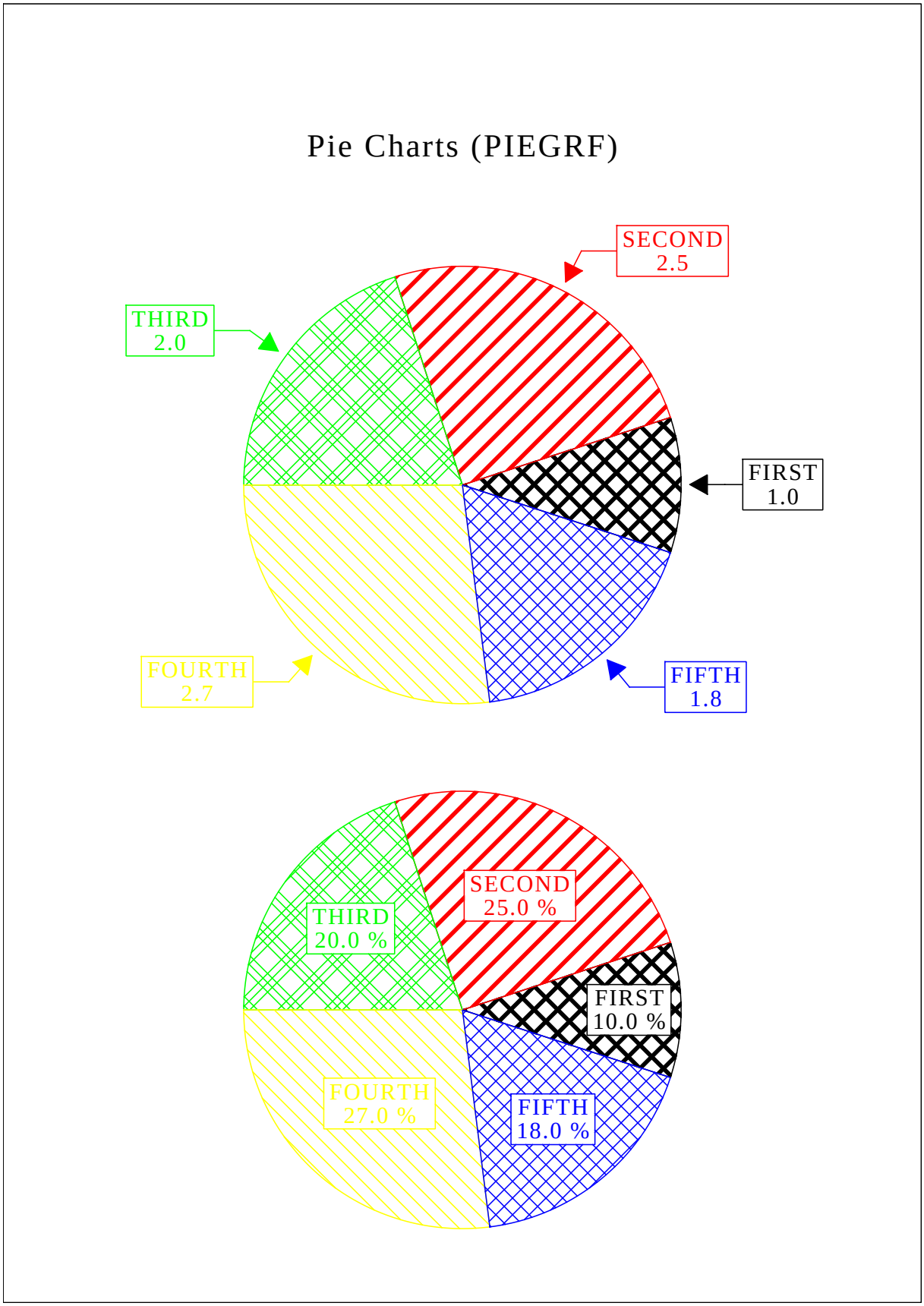

Figure B.15: Pie Charts

B.16 World Coastlines and Lakes

```
#!/usr/bin/ruby
require 'dislin'

Dislin.metafl('cons')
Dislin.disini()
Dislin.pagera()
Dislin.complx()

Dislin.axspos(400, 1850)
Dislin.axslen(2400, 1400)

Dislin.name('Longitude', 'X')
Dislin.name('Latitude',  'Y')
Dislin.titlin('World Coastlines and Lakes', 3)

Dislin.labels('MAP', 'XY')
Dislin.labdig(-1, 'XY')
Dislin.grafmp(-180.0, 180.0, -180.0, 90.0, -90.0, 90.0, -90.0, 30.0)

Dislin.gridmp(1, 1)
Dislin.color('green')
Dislin.world()

Dislin.color('foreground')
Dislin.height(50)
Dislin.title()

Dislin.disfin()
```

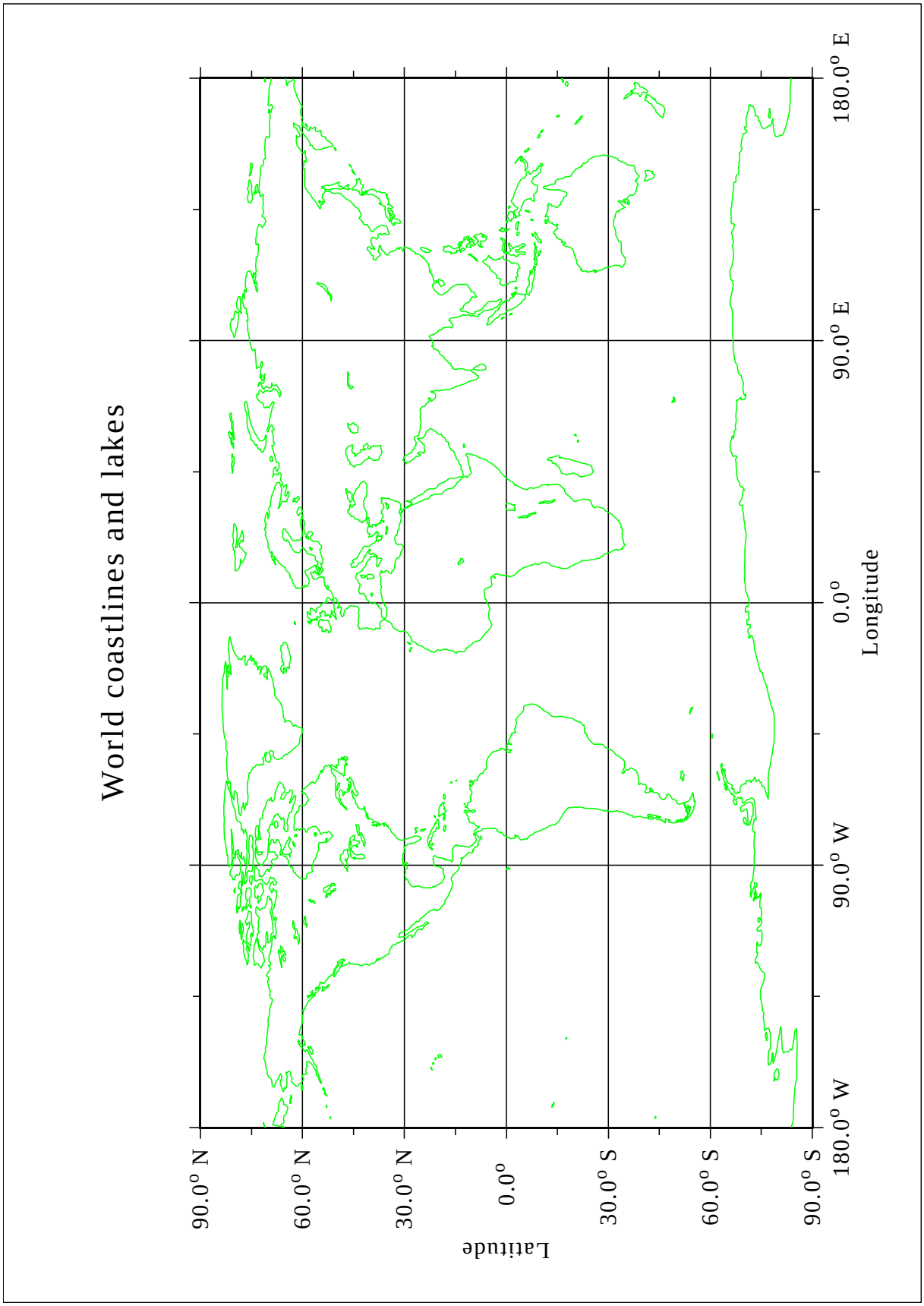

Figure B.16: World Coastlines and Lakes