

Introduction to scientific visualisation  
Part 2

Petra Heil  
Australian Antarctic Division  
&  
Antarctic Climate and Ecosystems CRC  
University of Tasmania  
Hobart, TAS 7001  
Australia

October 13, 2006

This course is designed to introduce students of environmental science studies (e.g., oceanography, atmospheric sciences, environmental science) to *state of the art* computer-based graphic tools for scientific analysis as well as digital presentation of derived results. The course emphasis is on presenting available software tools and how to use and apply these in academic research where the focus is on computational fluid dynamics. Not all software packages currently available have been included in this course, instead the it is attempted to provide the student with an overview of the various types of software available and their potential benefit within fluid dynamics applications.

Course objectives are to:

- analyse and interpret large data sets derived either from experiments or computer simulations;
- visually summarise multi-dimensional properties (e.g., extrema, eigenvalues) of the data set;
- understand application and virtue of discussed software tools;
- prepare a well-structured project using the student's data set.

## Syllabus

### Part 1:

- 1. Background and introduction.
- 2. Graphical presentation of data.
- 3. Basic visualisation algorithms.
- 4. Volume visualisation.

### Part 2:

- 5. Visualisation systems: a) Matlab.
- 6. Visualisation systems: b) IDL.

### Part 3:

- 7. Visualisation systems: c) GMT.
- 8. Visualisation systems: d) Ferret.
- 9. Visualisation systems: e) Vis5d+.
- 10. Other application examples and useful data sources.

# Chapter 1

## Specialized visualisation packages

Visualisation software can be divided into three basic groups:

- Modular visualisation environments (MVE), complete environments for the development of visualisation.
- Software toolkits, such as libraries, that can be helpful when creating visualisation applications.
- Specialised visualisation software.

### *Modular visualisation environments*

The user has direct control of the visualisation process, often by interacting with a visual programmable interface. Software in this category are:

- AVS/Express
- OpenDX
- IRIX Explorer (from NAG)

### *Toolkits*

The user can use toolkits to develop a number of different visualisation applications by using various visualisation techniques. An example is VTK<sup>1</sup>, an open-source visualisation toolkit for 3D computer graphics, image processing, and visualisation.

### *Specialised visualisation software*

These specialised softwares are generally produced for a certain type of data visualisation, and application might be limited.

---

<sup>1</sup><http://public.kitware.com/VTK/>

- Vis5d
- PCmodel

## 1.1 Matlab or Scilab

Matlab<sup>2</sup>, a product of *The MathWorks*, is described as the language of technical computing. The name Matlab is an abbreviation for *matrix laboratory*. Matlab is software to carry out vector- and matrix-based numerical computation, data visualisation, and provides an easy programming environment. Matlab consists of a set of essential commands and a number of extensions, also called toolboxes. E.g., the *Symbolic Math* toolbox contains symbolic expressions. Programs (M-files) and session data (MET-files) can be saved for later use.

Scilab<sup>3</sup> is the free scientific software package with similar capabilities as Matlab.

### 1.1.1 M-files

Most Matlab functions are available in form of M-files, which are files with the extension *.m*. The user can create his or her own M-files, which contain Matlab code. In the Matlab user environment M-files are then run by simply typing the name (without extension) of the M-file. For this to work the M-file must be in the same directory as the active Matlab session. Alternative M-files can be placed in a directory, whose *path* is included in the user-environment.

User-created M-files may be:

- Scripts to perform an operation on data in the Matlab workspace.
- Functions that ingest one or more user arguments and return a set of output. Variables used within the function are local only unless they are declared as global.

### 1.1.2 The Matlab user environment

Matlab provides a number of tools that allow the user to manage the workspace and program variables and the import and export of data. M-files and MET-files enable storage of the contents of the work environment for later use.

Depending on the version of Matlab installed, one will obtain a single window (or xterm) with Matlab running at the command line, or starting with version 6 a set of *gui*'s will open up, one of these will operate at the Matlab command line. The option *-nojvm* when calling Matlab from the Unix command line will open Matlab without any *gui*'s.

---

<sup>2</sup><http://www.mathworks.com/>

<sup>3</sup><http://www.scilab.org>

The Matlab command line is marked with ">>".

The user may quit Matlab by typing "exit" at the Matlab command line.

In Matlab comments are preceded by a "%", and the rest of the line is ignored by Matlab.

### 1.1.3 Starting Matlab and the help function

To run Matlab under the Unix operating system, type **matlab** at the command-line prompt. The Matlab prompt is generally given by the ">>". To exit out of Matlab type "quit" at the Matlab prompt.

Matlab provides an on-line help system. Typing "help" in the command line produces a list of topics for which on-line help is available. More specific help is provided by typing "help *command*" for any given *command*, e.g. "help *plot*".

A listing of all variables used within a Matlab session is obtained by typing "who", while the array size of each variable is also displayed when typing "whos". Results of operations without assigning a variable name are stored as a temporary variable called "ans". Only the most recent operation is kept in the temporary variable "ans".

In Matlab the data display can be modified using the **format** command: For example

```
x=[2/3 4.54545e-5]

format short      x= 0.6667          0.0000
format short g    x= 0.67          4.5454e-05
format long       x= 0.666666666666667 0.00004545450000
may be displayed as: format long g    x= 0.67          4.54545e-05
format rat        x= 2/3           45/990001
format bank       x= 0.67          0.00
format hex        x= 3fe5555555555555 3f07d4ccba4727f7
```

Notes:

- Undesired screen output can be omitted by ending input lines with a semi-colon.
- The command **format compact** eliminates blank lines from the screen output, and is useful to retain more data within the screen display.
- Command lines may be continued over 2 or more lines by typing three periods then the return-key, and continuing the statement in the next line.

## 1.1.4 I/O in Matlab

Importing data from Matlab:

- The **load** command:  
The load command reads binary or formatted files of existing matrices. Text files need to be structured as rectangular, non-sparse matrices of numbers separated by blanks.

- Loading a NetCDF file:

```
nc = netcdf('test_in.nc', 'nowrite');           % Open NetCDF file.
description = nc.description(:)                 % Global attribute.
variables = var(nc);                             % Get variable data.
for i = 1:length(variables)
    disp([name(variables{i}) ' =']), disp(' ')
    disp(variables{i}(:))
end
nc = close(nc);                                 % Close the file.
```

- NetCDF data files are loaded into Matlab using the **nload** command. See the CSIRO matlab/netCDF interface<sup>4</sup> for further information.

Exporting data from Matlab:

- The **save** command:  
By itself **save** stores all workspace variables in a binary format in the current directory in a file named matlab.mat. MAT-files are double-precision, binary, Matlab format files. They can be created on one machine and later read by Matlab on another machine with a different floating-point format, retaining as much accuracy and range as the different formats allow. The data are retrieved via the **load** command.  
Other options include:

- save('filename')
- save('filename', 'var1', 'var2', ...)

---

<sup>4</sup><http://www.marine.csiro.au/sw/matlab-netcdf.html>

- save(..., 'format'), where the following standards are supported:
  - append                   appends data to the specified existing MAT-file
  - ascii                    8-digit ASCII format
  - ascii -double           16-digit ASCII format
  - ascii -tabs             tab-delimited 8-digit ASCII format
  - ascii -double -tabs    tab-delimited 16-digit ASCII format
  - mat                     binary MAT-file form

Example:

```
p = rand(1, 10);
q = ones(10);
save('/home/user/matlab1/test.out', 'p', 'q', '-ASCII')
```

- Matlab I/O functions:

This includes the use of fopen, fwrite, and fclose to generate a data file. Files can be opened in binary mode (the default) or in text mode.

- fopen:
  - 'r'    read
  - 'w'    write (create if necessary)
  - 'a'    append (create if necessary)
  - 'r+'   read and write (do not create)
  - 'w+'   truncate or create for read and write
  - 'a+'   read and append (create if necessary)
  - 'W'    write without automatic flushing
  - 'A'    append without automatic flushing

Example:

```
fid = fopen('output.tst1','w')
```

- fwrite:

Example:

```
cnt = fwrite(fid,var1,var2,'%I4,%I4\n')
```

- fclose:

Example:

```
error = fclose (fid)
```

ST = FCLOSE(FID) closes the file with file identifier FID, which is an integer obtained from an earlier FOPEN. FCLOSE returns 0 if successful and -1 if not.

, where error = 0 signals a successful operation and error = -1 an unsuccessful operation.

- Translate a MAT-output file into other formats:  
Matlab provides Fortran and C-libraries to translate a MAT-file into other formats. See their manual<sup>5</sup> for details.
- Develop a MEX-file<sup>6</sup> to export data, if required by user application.
- Create a NetCDF file within Matlab:

```

nc = netcdf('test.nc', 'clobber');           % Create 'nc', an
                                           % interface to file.

nc.description = 'NetCDF Create test file'; % Set global attributes.
nc.author = 'Petra Heil';
nc.date = '15 June 2004';

nc('lat_dimension') = 10;                   % Define dimensions.
nc('lon_dimension') = 24;

nc{'latitude'} = 'lat_dimension';          % Define variables.
nc{'longitude'} = 'lon_dimension';
nc{'hour'} = {'lon_dimension'};

nc{'latitude'}.units = 'degrees';           % Set attributes.
nc{'longitude'}.units = 'degrees';
nc{'hour'}.units = 'hours';

latitude = [0 10 20 30 40 50 60 70 80 90]; % Create data.
longitude = [0 15 30 45 60 75 90 105 120 135 150 165 180 ...
            195 210 225 240 255 270 285 300 315 330 345];
hour = -12. + longitude/15;

nc{'latitude'}(:) = latitude;               % Write data.
nc{'longitude'}(:) = longitude;
nc{'hour'}(:) = hour;

nc = close(nc);                             % Close netCDF file.

```

---

<sup>5</sup>[http://www.mathworks.com/access/helpdesk/help/techdoc/matlab\\_external/ch01int6.html](http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/ch01int6.html)

<sup>6</sup>[http://www.mathworks.com/access/helpdesk/help/techdoc/matlab\\_external/ch03cre2.html](http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_external/ch03cre2.html)

### 1.1.5 The Matlab language and its elements

Matlab is based on a language components such as functions, data structures, input/output, control-flow structure, and object-oriented programming.

*Note:* Matlab is case-sensitive.

#### Key commands in Matlab

- clear
- load
- plot
- poly
- polyfit and polyval
- roots
- conv and deconv
- inv
- eig

#### Key expressions in Matlab

Type or dimension declarations are not required in Matlab but are assigned automatically when a new variable name is introduced. The dimension of variables can be adjusted while in use.

#### Vectors and matrices

Vectors and matrices form the base for Matlab operations. A vector can be defined explicitly

$$x = [1 \ 1 \ 2 \ 2 \ 3 \ 3]$$

which in Matlab is displayed as:

$$x = \ 1 \ 1 \ 2 \ 2 \ 3 \ 3$$

or derived implicitly by providing an initial value (minimum), a step size and an end value (maximum), e.g.:

$$x = 1 : 2 : 11$$

which in Matlab is displayed as:

$$x = 1 \ 3 \ 5 \ 7 \ 9 \ 11$$

In Matlab matrices are represented as multi-dimensional vectors. When entered explicitly the different dimensions are separated by a semicolon or a keyboard "return":

$$X = [1 \ 1 \ 1; \ 2 \ 2 \ 2; \ 3 \ 3 \ 3; \ 4 \ 4 \ 4]$$

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix}$$

Matrices may be transposed:

$$Y = X'$$

for which Matlab will return:

$$Y = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

Matrices may be multiplied:

$$Z = X * Y$$

for which Matlab will return:

$$Z = \begin{bmatrix} 3 & 6 & 9 & 12 \\ 6 & 12 & 18 & 24 \\ 9 & 18 & 27 & 36 \\ 12 & 24 & 36 & 48 \end{bmatrix}$$

or :

$$Z = X * Y$$

for which Matlab will return:

$$Z = \begin{bmatrix} 30 & 30 & 30 \\ 30 & 30 & 30 \\ 30 & 30 & 30 \end{bmatrix}$$

We define a symmetric matrix M:

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

The inverse of a symmetric matrix is calculated by the Matlab function *inv*:

$$\text{inv}(M)$$

for which Matlab will return:

$$\text{ans} = \begin{array}{cc} -2.0000 & 1.0000 \\ 1.5000 & -0.5000 \end{array}$$

and the eigenvalues of a symmetric matrix are calculated by the Matlab function *eig*:

$$\text{eig}(M)$$

for which Matlab will return:

$$\text{ans} = \begin{array}{c} -0.3723 \\ 5.3723 \end{array}$$

## Functions

Several mathematical functions, e.g., *abs*, *bessel*, *cos*, *sin*, *sqrt*, are available within Matlab. Information on a number of function groups are available via the **help** information:

*help elfun*

*help elmat*

*help specfun*

Special functions provide values of useful constants:

pi	$\pi$ (3.14159265)
i	Imaginary unit (-1)
j	Imaginary unit (-1)
eps	Floating point relative precision (2 - 52)
realmin	Smallest floating point number (2 - 1022)
realmax	Largest floating point number (2 - 21023)
Inf	Infinity
NaN	<i>Not-a-number</i>

*Note:* Function names are not write-protected and can be overwritten.

## Polynomials

Matlab represents a polynomial by a vector, which has the length of the power of the polynomial plus one. E.g.,

$$y(x) = Ax^3 + Bx^2 + Cx^1 + D$$

is represented by a vector:

$$yy = [A \ B \ C \ D]$$

and is expanded into a polynomial at the value  $x$  by the calling the Matlab function *polyval*:

$$y = polyval(yy, x)$$

Two polynomials can be multiplied in Matlab by using the command *conv*, and any two polynomials may be divided in Matlab by using the command *deconv*.

The Matlab command to find the roots of a polynomial is *roots*.

### 1.1.6 Flow control

- *if* statements  
Evaluation of logical expressions.  
Related keywords: *else*, *elseif* and *end*.
- *for* loops  
Related keyword: *end*.
- *while* loops  
Related keyword: *end*.
- *switch* statements  
Related keywords: *case*, *otherwise* and *end*.
- *break* statements  
Enables early exit from a *for* or *while* loop.

### 1.1.7 Graphics properties

*set* and *get*

The *set* functions allows the user to define the properties of any object to be used in the following application, e.g.:

```
set(h, 'FontSize', [14])
```

A listing of the adjustable properties is obtained by typing *set(h)* at the Matlab prompt:

```
set(h)
```

This yields:

```
Color
EraseMode: [ {normal} | background | xor | none ]
LineStyle: [ {-} | -- | : | -. | none ]
LineWidth
Marker:     [ + | o | * | . | x | square | diamond | v | ^ | > |
             < | pentagram | hexagram | {none} ]
MarkerSize
MarkerEdgeColor: [ none | {auto} ] -or- a ColorSpec.
MarkerFaceColor: [ {none} | auto ] -or- a ColorSpec.
...
XData
YData
ZData
```

A listing of all current properties can be obtained by typing *get(h)* after the **handles** for a Matlab operation have been marked, e.g., by typing *h = plot( ... )*.

### 1.1.8 Graphics generation

Graphics are usually generated into a *figure* environment activated within Matlab. Digital output of the visuals can be generated in a variety of output formats (see section *Digital output*). Matlab offers a range of 2D and 3D-plotting routines, these include:

```
plot
bar
contour
surf
mesh
patch
fill
```

These can be used together with the following display options:

```
subplot
shade
```

```
view
colormap
axis
```

Analytical tools include:

```
fitfun
fminsearch
fft
```

## Line plotting

Simple 2D line plots can be easily generated in Matlab:

```
x=0:0.1:2*pi;
y=1/15.*sin(x/2*pi);
hold off
hline=plot(x,y,'green');
hold
hline=plot(x,y,'k*');
title('Sine curve')
xlabel('Time (hours)')
ylabel('Signal (ms-1)')
```

A set of predefined line types ('-', '--', ':', '-.', '.') and line symbols ('.', 'o', 'x', '+', '\*', 'd', 's', 'h', 'p', '^', 'v', '<', '>') are available.

Line colours may be chosen from blue ('b'), green ('g'), red ('r'), magenta ('m'), cyan ('c'), yellow ('y') or black ('k').

Line width and symbol size may be set explicitly:

```
hline=plot(x,y,'*');
set(hline,'LineWidth',1)
set(hline,'MarkerSize',9)
```

To plot multiple lines in one plot:

```
hline=plot(x,y,'*',xx,yy,'r - -');
```

## 2D plots

In addition to line plots two-dimensional data may be displayed in bar charts, staircase plot, semilogarithmic or logarithmic scaled plot, polar plot, stem plot, or with error bars.

```
x=0:0.1:2*pi;
y=1/15.*sin(x/2*pi);
subplot(3,2,1)
hold off
plot(x,y,'green');
hold
e=erf(y);
errorbar(x,y,e/3);
axis([0,2*pi,-0.1,0.2])
title('Line plot')
subplot(3,2,2)
bar(x,y);
axis([0,2*pi,-0.1,0.2])
title('Bar chart')
subplot(3,2,3)
stairs(x,y);
axis([0,2*pi,-0.1,0.2])
title('Stairstep plot')
subplot(3,2,4)
semilogx(x,y,'green');
axis([0,2*pi,-0.1,0.2])
title('Semilogarithmic-scaled plot')
subplot(3,2,5)
polar(x,y);
title('Polar plot')
subplot(3,2,6)
stem(x,y);
axis([0,2*pi,-0.1,0.2])
title('Stem plot')
```

Figure 1.1 shows the results of running the above Matlab script.

## 3D surface plots

Three-dimensional data may be visualised on 2D via a number of techniques.

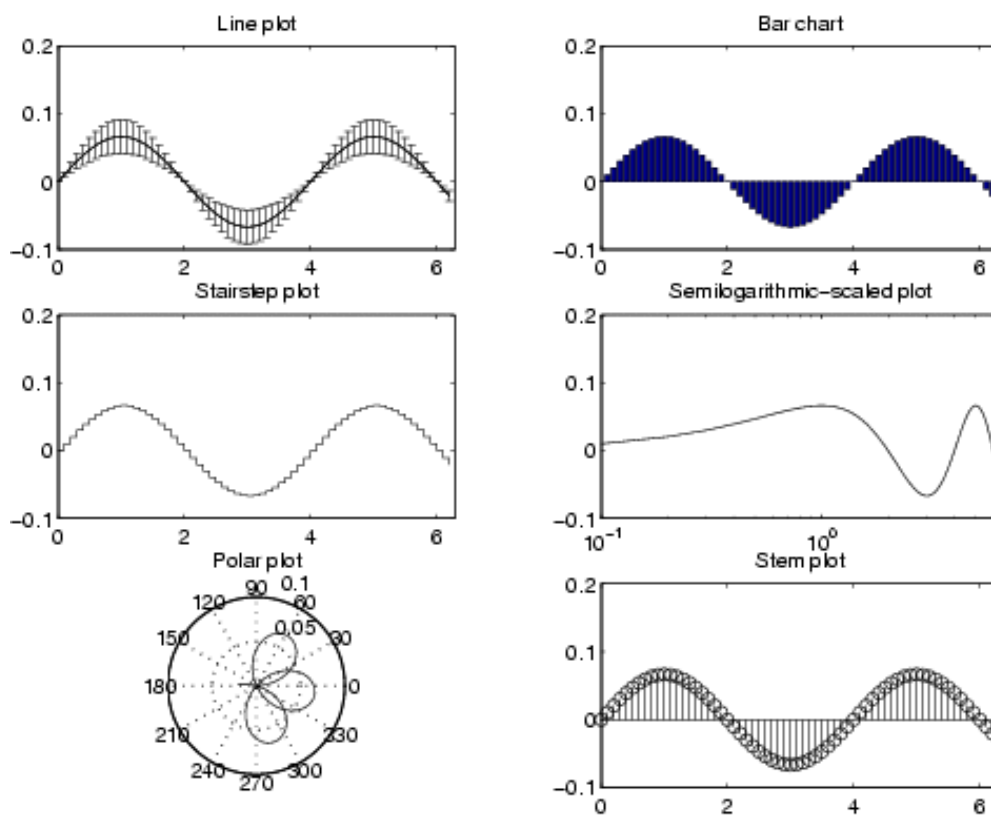


Figure 1.1: 2D plot examples.

```

subplot(3,2,1)
t=0.2:0.2:100;
plot3(exp(-2./t).*sin(t),exp(-2./t).*cos(t),t)
title('3D line-plot')
subplot(3,2,2)
for i=1:25
for j=1:25
for k=1:25
x(i,j) = i;
y(i,j) = j;
z2(i,j)=exp(-2./i)*sin(1/25*pi*i)*cos(1/25*pi*j);
z3(i,j,k)=exp(-2./k)*sin(1/25*pi*i)*cos(1/25*pi*j);
end
end
end
mesh(z2);
title('Mesh plot')
axis([0 25 0 25 -0.6 0.6])
subplot(3,2,3)
surf(z2)
shading interp;
colormap(hot);
title('Surf plot with shading')
subplot(3,2,4)
contour(z2,[-0.7:0.05:0.7])
axis([0 25 0 25 -0.6 0.6])
title('Contour plot')
subplot(3,2,5)
[px,py] = gradient(z2,.1,.1);
quiver(x,y,px,py,2);
axis([0 25 0 25])
title('Vector plot')
subplot(3,2,6)
slice(20*z3,[5 22],10,[20,10])
title('Plot of 3D slices')
print -deps plots_3d

```

Figure 1.2 shows the results of running the above Matlab script.

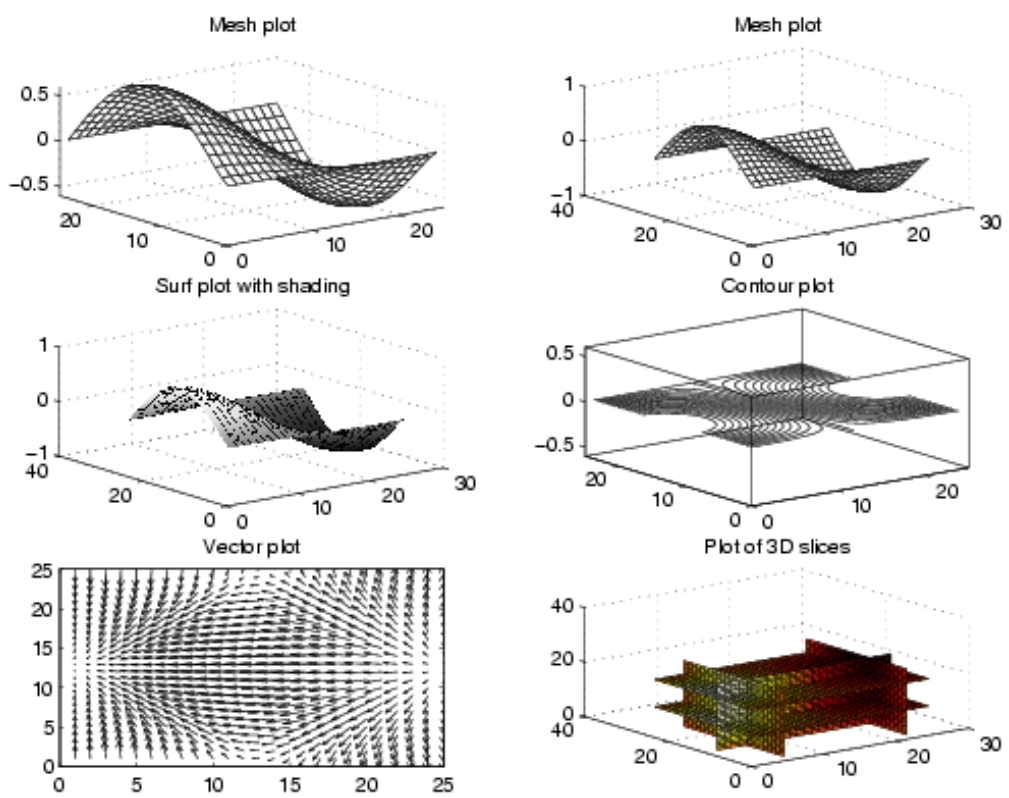


Figure 1.2: 3D plot examples.

### 1.1.9 Mapping with Matlab

Matlab provides facilities to prepare geographical maps and to display data embedded in those maps. Maps are initialised with the **m\_map** command. Note that Matlab assumes that the x-coordinate represents longitude and the y-coordinate represents latitude.

Supported projection types include:

- Stereographic
- Orthographic
- Azimuthal Equal-area
- Azimuthal Equidistant
- Gnomonic
- Satellite
- Albers Equal-Area Conic
- Lambert Conformal Conic
- Mercator
- Miller Cylindrical
- Equidistant Cylindrical
- Oblique Mercator
- Transverse Mercator
- Sinusoidal
- Gall-Peters
- Hammer-Aitoff
- Mollweide
- UTM

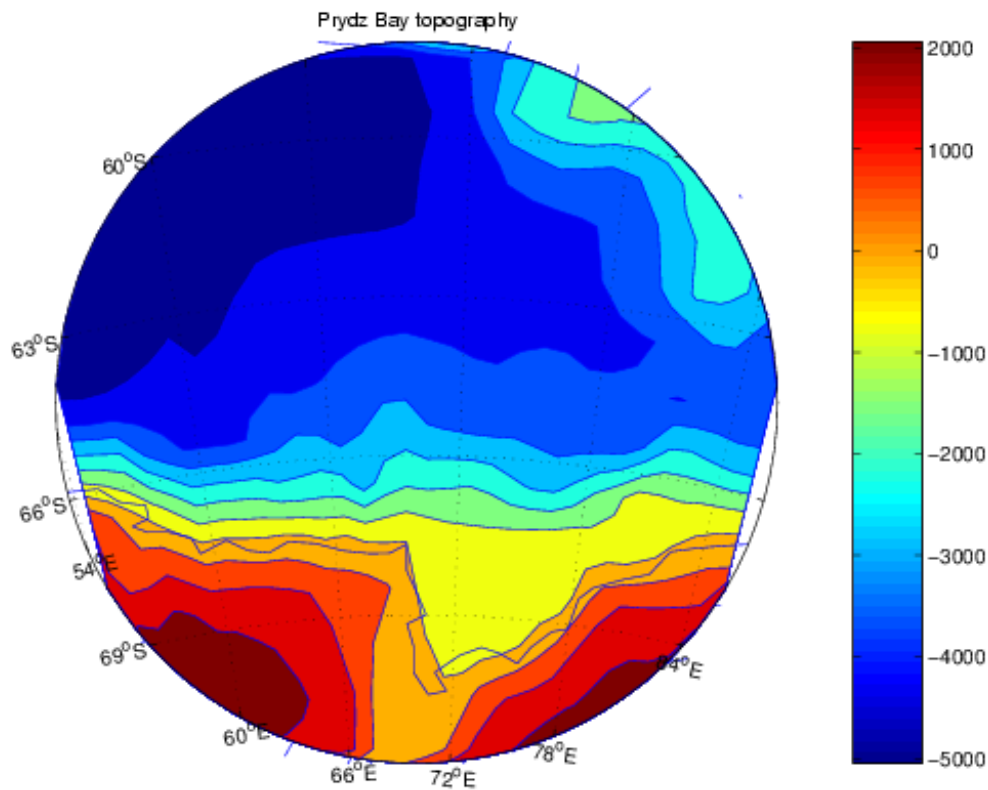


Figure 1.3: Coastline and topography around Prydz Bay, East Antarctica.

### Coastline and topography options

The **m\_map** data base provides a  $0.25^\circ$  resolution for the coastlines via the **m\_coast** command, and a  $1^\circ$  global elevation database for map topography via the **m\_elev** command. Caution needs to be exercised when using both the coastline and the topography data sets within the same figure, as the horizontal resolution differs between the two data sets, as shown in figure 1.3.

High-resolution coastline data may be supplied to **m\_map** by the user. This involves selecting a subset of high-resolution data from the user-supplied data set and to convert it to Matlab display coordinates (using the **m\_ll2xy** command).

The following example depicts the topography of the Prydz Bay region using the TerrainBase<sup>7</sup> data set. The results are shown in figure 1.4.

```
m_proj('lambert','lon',[50 90],'lat',[-73 -57]);
```

<sup>7</sup><ftp://ncardata.ucar.edu/datasets/ds759.2/data/tbase.Z>

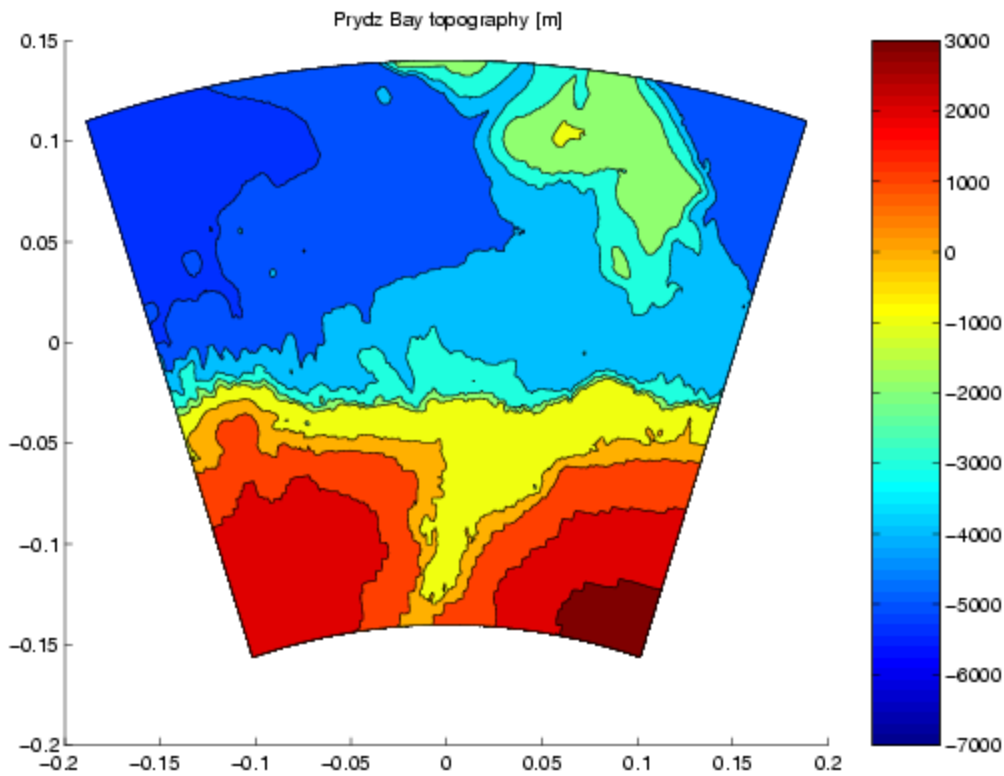


Figure 1.4: Detailed topography around Prydz Bay, East Antarctica.

```
m_tbase('contourf');
m_grid('linestyle','none','tickdir','out', ...
'linewidth',2,title('Prydz Bay topography [m]'));
colorbar
```

## Data overlay

Map projections and topography information are often used to support analysis or interpretation of the user's own data. Hence the graphics software needs to provide the facility to overlay these data on the map:

```
m_proj('lambert','lon',[50 90],'lat',[-73 -57]);
m_tbase('contourf');

geo_lon =[ 50, 60, 70, 80, 90];
%10deg ice-edge position from
```

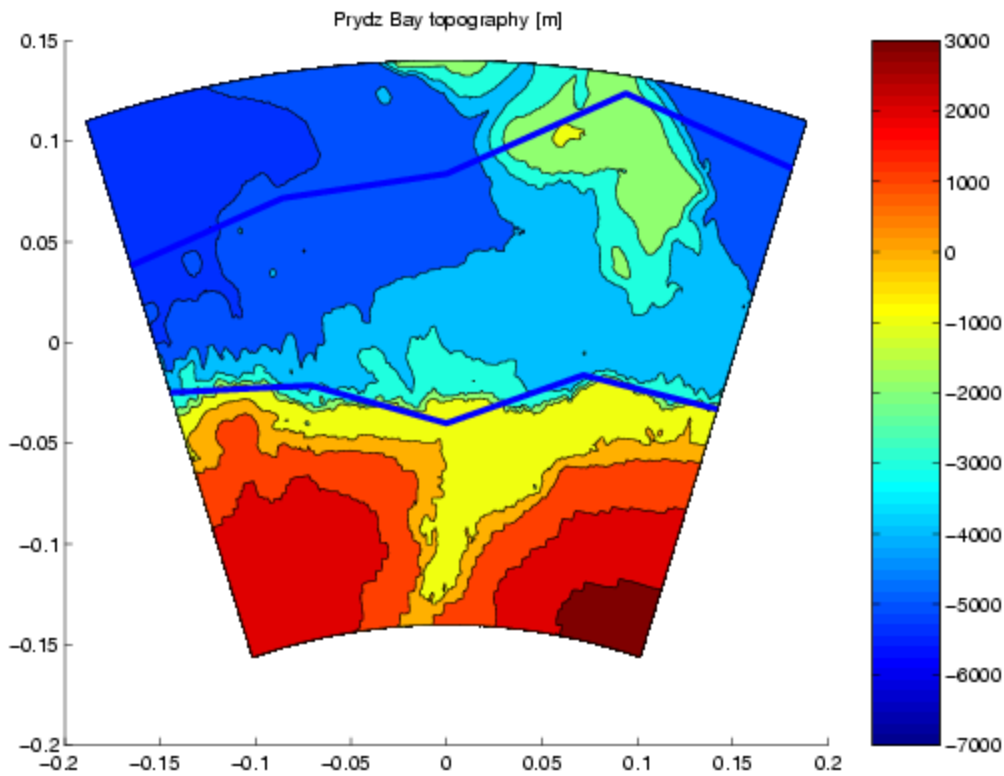


Figure 1.5: Prydz Bay topography with overlaid tracks.

```
%http://www.antcrc.utas.edu.au/~jacka/seaice.html
geo_latfeb=-1.0*[ 65.1, 65.9, 67.3, 65.6, 65.6];
geo_latsep=-1.0*[ 61.3, 60.5, 60.2, 57.5, 58.4];

m_plot(geo_lon,geo_latfeb,'linewidth',3);
m_plot(geo_lon,geo_latsep,'linewidth',3);
colorbar
```

The results are shown in figure 1.5.

Unfortunately in Matlab the filled polygon overwrites all underlying data. E.g., by continuing the script above with the following lines:

```
geo_lon_p =[ 50, 60, 70, 80, 90, 90, 80, 70, 60, 50];
geo_lat_p =-1.0*[ 65.1, 65.9, 67.3, 65.6, 65.6 58.4, 57.5, 60.2, 60.5, 61.3];
m_patch(geo_lon_p,geo_lat_p,[0.3 0.3 0.3]);
```

and figure 1.6 is obtained.

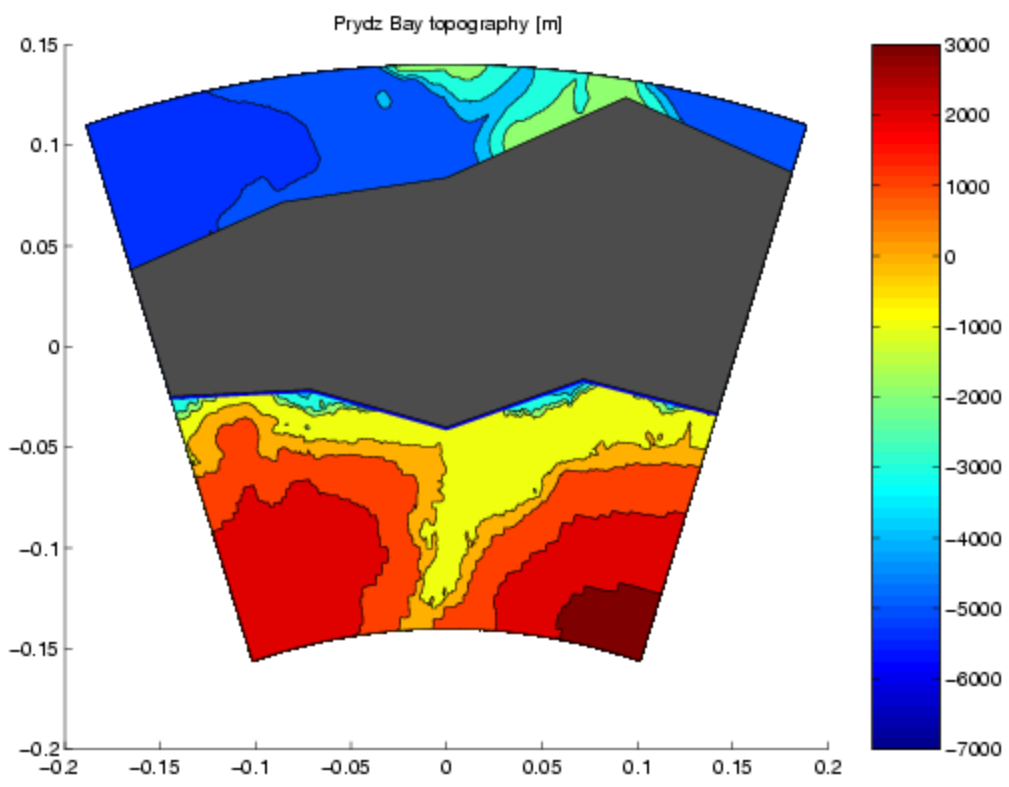


Figure 1.6: Prydz Bay topography with overlaid tracks.

### 1.1.10 Animated graphics

Several options are available to produce animated graphics.

A fast approach is provided by the *movie* option, where the number of frames is determined on the outside of the Matlab routine.

The first step is decide the number of frames required for the animation:

```
nk = 30;
```

Then a plot is created:

```
xx=[1:50];
yy=1/50*sin(xx);
h = plot(xx,yy,'-');
set(h,'MarkerSize',18);
A = moviein(nk);

for i=1:nk
yy(i)=i/50*sin(xx(i));
set(h,'XData',xx,'YData',yy)
M(:,i) = getframe;
end
```

To play the movie type

```
movie(M,NN)
```

where *NN* is the number of repeat cycles of the movie.

### Graphics interfaces

The Matlab graphics facilities can be used to build user interfaces:

```
button = uicontrol('Style','pushbutton','Units','normalized',...
'Position',[.45 .45 .2 .1],'String','User interface');
```

which can be treated as a Matlab object.

To activate the displayed push button both the execution command and the *callback* are required:

```
buttonpos = 'set(button, 'position"', [.8*rand .9*rand .2 .1])';  
set(button, 'Callback', buttonpos)
```

To move the button randomly across the graphic type

```
eval(buttonpos)
```

### 1.1.11 Digital output

Figures and animations may be directly streamed to a printing device or exported in a variety of formats using the *print* command.

Examples for exporting are:

```
print -dwinc          (Prints current figure to current printer in colour)  
print(1, '-deps', 'out1')  (Prints figure No. 1 to out1.eps)  
print(2, '-djpeg', 'out2') (Prints figure No. 2 to out2.jpg)
```

Matlab also allows the user to save the complete Matlab working space onto disk for use in a later Matlab session. Typing

```
save
```

generates a binary file with default name "*matlab.mat*".

Selective saves of the workspace are also possible:

```
save outfile xx yy
```

and *ASCII* output may also be generated:

```
save outfile2 xx yy -ASCII          (8-digit ASCII)  
save outfile2 xx yy -ASCII -double (16-digit ASCII)
```

### 1.1.12 Further examples

Matlab provides a set of examples, which can be activated by typing at the Matlab prompt:

```
demo
```

An internet search on Matlab tutorials is also recommended.

### 1.1.13 Useful links

- The USGS Woods Hole Science Center<sup>8</sup> hosts SEA-MAT<sup>9</sup>, which provides Matlab tools for oceanographic analysis.
- The OCEANS toolbox<sup>10</sup> contains a collection of M-files (including the UNESCO parameterisations) useful to oceanographers.
- The USGS Woods Hole Science Center<sup>11</sup> hosts GridGen<sup>12</sup>, a MATLAB-based tool to construct orthogonal curvilinear grids.
- A/Prof. Rich Pawlowicz's own Matlab site<sup>13</sup> with a mapping and other oceanographic toolboxes. Useful is the Matlab mapping package<sup>14</sup>.

A useful reference to Matlab may be found in:

Hunt, B.R., R.L. Lipsman, and J.M. Rosenberg, A Guide to MATLAB: for Beginners and Experienced Users, Cambridge University Press, ISBN 0521-00859-X, 2001.

---

<sup>8</sup><http://woodshole.er.usgs.gov>

<sup>9</sup><http://woodshole.er.usgs.gov/operations/sea-mat/index.html>

<sup>10</sup><ftp://acoustics.whoi.edu/pub/Matlab/oceans>

<sup>11</sup><http://woodshole.er.usgs.gov>

<sup>12</sup><http://woodshole.er.usgs.gov/operations/modeling/gridgen>

<sup>13</sup><http://www2.ocgy.ubc.ca/~rich>

<sup>14</sup><http://www2.ocgy.ubc.ca/~rich/map.html>

## 1.2 IDL

The Interactive Data Language (IDL<sup>15</sup>), a product of *Research Systems Inc.*, is a data analysis and visualisation software. It includes tools for mathematics, statistics, graphics, image processing, mapping and general data manipulation. It is often used for processing and displaying large arrays of data, such as satellite imagery.

Summary of IDL features:

- Flexible data I/O allows analysis of data from various digital sources.
- OpenGL support allows hardware accelerated 2D and 3D graphics.
- Provides object-oriented graphics technology.
- Provides image processing features (e.g. region-of-interest).

### 1.2.1 IDL programs, functions and batch files

IDL routines are presented in form of program (\*.pro) files. As with other data languages the user may create his/her own routines and functions and utilise them from within IDL by calling the appropriate user-defined program files.

Program files need to be "compiled" (see tool bar function compile, or by typing

```
.compile filename.pro
```

at the IDL command line. Program files may be executed either via the "run" button in the tool bar, or by typing

```
filename
```

at the IDL command line.

In program files comment lines need to start with a ";". \*.pro files need to contain defining elements and statements.

Example of a program file "summation.pro" generated with a text editor:

---

<sup>15</sup><http://www.rsinc.com>

```

; Header information
pro summation, a, b
; program declaration and list of variables passed into the program.

sum = a + b

print, sum;

return ; Return to the program that called program summation.
end

```

To compile in IDL:

```
.compile summation
```

To execute in IDL:

```
summation, 10, 4
```

for which IDL will return:

```
14
```

IDL supports functions to be stored in \*.pro files. These also require compilation through IDL and can be called from within other IDL program files:

Example of a function file "f\_summation.pro" generated with a text editor:

```

; Header information
function f_summation, a, b
; program declaration and of variables passed into the program.

f_sum = a + b

return, f_sum; Return to the program that called program summation.
end

```

To compile in IDL:

```
.compile f_summation
```

To execute in IDL:

```
value = f_summation(10,4)
```

for which IDL will store the result of this function in the variable "value".

For further information please refer to section 1.2.5 or to the online examples<sup>16</sup>.

Batch files are created using a text editor, and executed by IDL in a manner as if the text contained in a batch files was typed line by line at the IDL prompt. These kinds of files are extremely useful to repeat a series of IDL commands to rerun or to apply some small modification in the input. Batch files are activated at the IDL prompt via:

```
IDL> @batch_file.bat
```

The `@` symbol indicates to IDL to run the filename.

Example:

```
print, for i=1,10 do begin $ f = sin(i!*pi) & $ print, f $ end
```

## 1.2.2 Starting IDL and the help function

To run IDL under the Unix operating system, type **idlde** at the command line prompt. The IDL gui with its development environment will open on your monitor. IDL's multiple-document interface includes built-in editing and debugging tools. The menu on the top of the gui provides a guide through the IDL options.

The IDL command line is the bottom-most active field within the gui.

IDL provides an interactive help system, which is activated from the menu bar.

IDL is also available as a command line interface. To start IDL type **idl** at the Unix prompt.

Exit out of IDL by typing **exit**, or use the menu option.

## 1.2.3 The IDL user environment

IDL uses a command line interface to facilitate data input and output, graphics generation, annotation and output.

Upon starting, IDL will open a graphics interface, the "IDL development environment". This interface consists of multiple documents, including a tool bar, and interactive editing window (see figure 1.7).

Useful notes:

---

<sup>16</sup><http://www.rsinc.com/codebank/index.asp?list=bycategory&product=IDL>

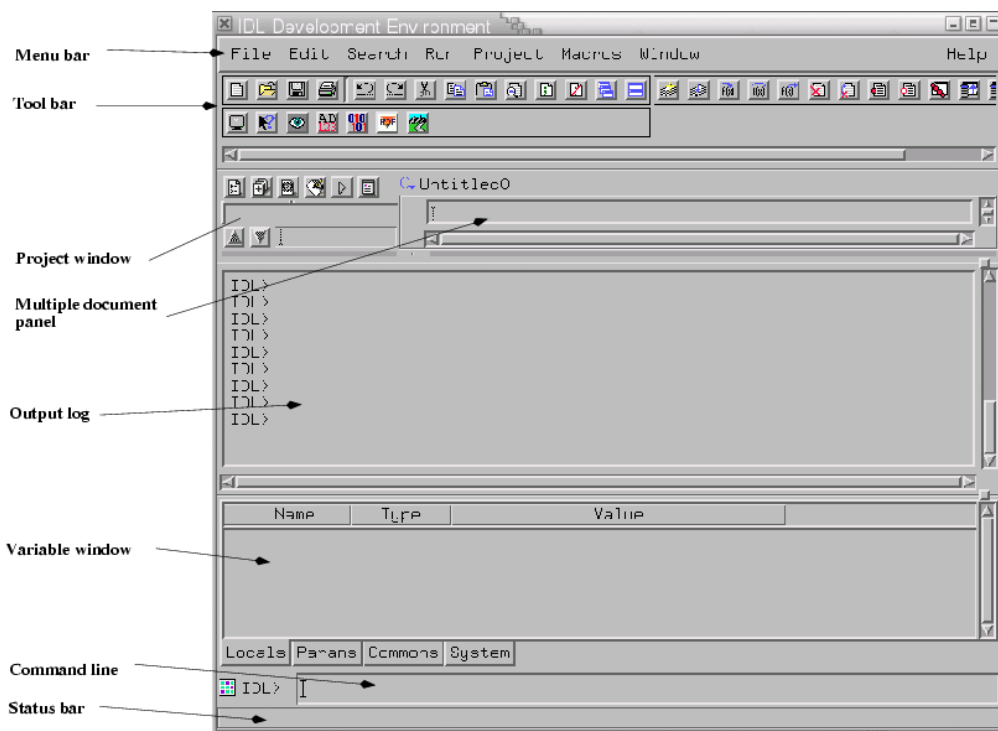


Figure 1.7: The IDL Development Environment (IDLDE).

- In IDL command phrases are followed by a comma before the parameter list.
  - Aborting IDL:
    - IDL programs can be stopped using CTRL-C.
    - IDL can be aborted immediately using CTRL-\. (Data variables are lost; opened files will be be damaged).
  - Retaining content of graphics windows:
    - ”Backing store” can be switched on or off (for reasons of speed and memory):
    - In IDL backing store is initialised via the command **retain**:
    - \* RETAIN = 0 implies no backing store,
    - \* RETAIN = 1 IDL issues a command to the window system to perform backing store,
    - \* RETAIN = 2 IDL does it
- Example:

```
window,0,retain=2,xsize=200,ysize=200
```

#### 1.2.4 I/O in IDL

IDL supports a range of data formats:

- ASCII
- Binary
- CDF
- DICOM
- DXF
- HDF
- HDF-EOS
- netCDF
- WAV
- XDR

Example:

```
nyny=read_tiff('/home/user/images/image.tif')
```

To write a file in IDL three steps are required:

```
openw,nid,'filename.dat'    ; Open the file with write permission.  
printf,nid,data1,data2     ; Print data to file.  
close,nid                  ; Close the file.
```

## 1.2.5 The IDL language and its elements

### The ! command

The character (!) is used to indicate constant pre-defined values.

### findgen

Creates a floating point array of the specified dimension, where each element of the array is set to the value of its one-dimensional subscript.

### Data\_dims

Set this keyword to a scalar or array of up to eight elements specifying the size of the data to be read and returned.

Example:

```
(data_dims=[360,180])
```

### Data\_type

Set this keyword to an IDL code that specifies the type of data to be read.

IDL code	Data type
0	Undefined
1	Byte
2	Integer
3	Longword Integer
4	Floating point
5	Double-precision floating
6	Complex
7	String
8	Structure
9	Double-precision complex
10	Pointer
11	Object reference
12	Unsigned Integer
13	Unsigned Longword Integer
14	64-Bit Integer
15	Unsigned 64-Bit Integer

Example:  
(data\_type=4)

### **Endian**

Setting the **endian** keyword to one of three string values "big", "little", or "native" specifies the byte ordering of the file.

Example:  
(endian='little')

Example for full read command:

To read a file containing binary data of sea-surface temperature:

```
sst=read_binary('/home/user/data/ocean_model/seasurface.dat', $
               data_dims=[360,180],data_type=4,endian='little')
```

### **where**

This command is useful in analysing array data.

```
x = findgen(10,10)
index = where(x lt 30)
y = x[index]
```

In this example y will contain all the values of x that have values less than 30.

## **1.2.6 Flow control**

There are several IDL commands that allow the user to control the flow of a program:

### **FOR loops**

Repeated of IDL commands may be coded using the FOR loop:

```
for i = start_index,stop_index
do statement
```

The stepping size may be user determined, if no value is given an increment of "1" is assumed:

```
for i = start_index,stop_index,increment
do statement
```

## WHILE loops

WHILE loops provide the user with the looping of statement which is conditional to another program statement without the need to specify the number of loop iterations required.

```
while condition
  do statement
```

or in a more complex construct a block statement may be used:

```
while condition
  do begin
    statement #1
    statement #2
    statement #3
  endwhile
```

A typical use of WHILE loops is for the line-wise reading of an input file, whose length is unknown:

```
indata=fltarr(10,2000) ;Define array large enough to hold unknown
                    ;input data.
inline=fltarr(10)    ;Define vector to hold data input from each
                    ;line of the file.
cnt=0                ;Set line count to Zero.

openr,nid,'input.dat' ;Open input file with read permission.

; Loop to read in data from file:
while not eof(nid) do begin ;Check if end of file is reached.
  readf,nid,inline
  indata[:,cnt]=inline      ;Move file info onto IDL data array.
  cnt=cnt+1                ;Increase line counter by one.
endwhile

indata=indata[:,0:cnt-1]   ;Truncate array to actual length of
                    ;input file.

close,nid

plot,indata[1,:],indata[2,:] ;Generate a plot using two input data
                    ;columns.

end
```

## REPEAT loops

REPEAT loops are similar to WHILE loops except that the condition is checked at the end of the loop:

```
repeat begin
  statement
endrep until condition
```

## IF statements

IF statements are used for conditional action, e.g., when the execution of one (or more) commands depends on the condition of another variable:

```
if (i gt 2) then x = sqrt(i) else x = i
```

or a more complex conditional execution may be created using block statements:

```
if (i gt 2) then begin
  x = sqrt(i)
  y = i/2.
  z = sqrt(x^2 + y^2)
endif else begin
  x = i
  y = i/4.
  z = sqrt(x^2 + y^2)
endelse
```

In IDL the conditional statements are driven by boolean and relational operators:

Boolean Operator	IDL symbol
And	and
Not	not
Or	or
Exclusive OR	xor

Relational Operator	IDL symbol
Equal to	eq
Not equal to	ne
Less than or equal to	le
Less than	lt
Greater than or equal to	ge
Greater than	gt

## Other flow control commands

IDL also supports:

- goto statements
- case statements

## 1.2.7 Graphics properties

### Windows

In IDL the user can open graphics windows with the **window** command:

```
window,0
```

Graphics windows can be limited to a certain number of colours:

```
window,0,colors=64
```

### Colour

IDL provides the user with a suite of colour schemes. These are loaded with the **loadct** command:

```
loadct,10
```

IDL also allows user-defined colours and colour schemes, and these may be uploaded from \*.pro files. The **tvlct** command is used to set colour values:

```
tvlct, [0,1,1,0.5,1]*255, [0,1,0,0,0,1,1,1]*255, [0,0.4,0,1,1]*255
```

where:

```
red= [0,1,0,0,0]*255  
green=[0,0,1,0,1]*255  
blue= [0,0,0,1,1]*255
```

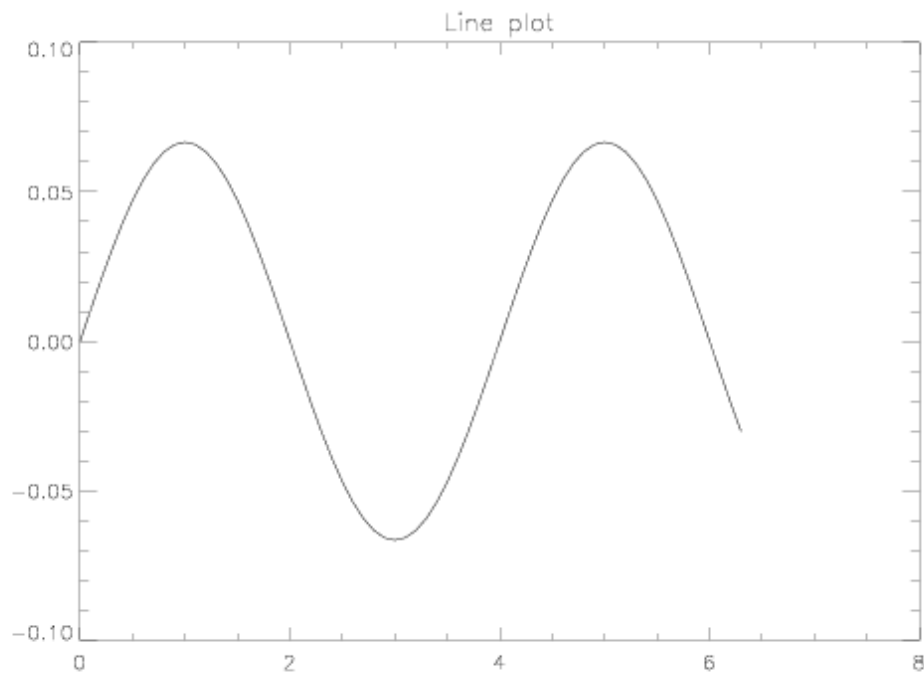


Figure 1.8: Simple line plot in IDL.

## 1.2.8 Graphics generation

### Line plotting

This is a simple example for plotting a line in IDL:

```
x=findgen(64)/10;
y=1/15.*sin(x/2!*pi);
plot,x,y,xrange=[0,6.6],title='Line plot';
```

The results are shown in figure 1.8.

### 3D surface plots

Three-dimensional data may be visualised on 2D via a number of techniques.

To generate some 2D data arrays, and auxiliary data structures do the following:

```
t=findgen(500)/5
x=findgen(25)+1
```

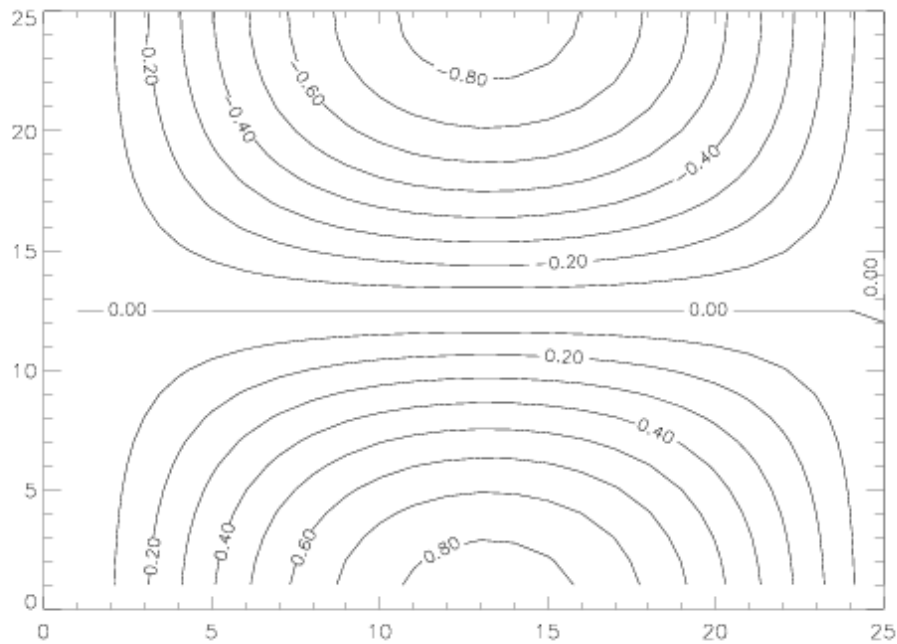


Figure 1.9: Contour plot example.

```

y=findgen(25)+1
z2=fltarr(25,25)
z3=fltarr(25,25,25)
for i=1,25 do for j=1,25 $
    do z2[i-1,j-1]=exp(-2./i)*sin(1./25*!pi*i)*cos(1./25*!pi*j)
for i=1,25 do for j=1,25 do for k=1,25 $
    do z3[i-1,j-1,k-1]=exp(-2./k)*sin(1./25*!pi*i)*cos(1./25*!pi*j)

```

We then generate a contour plot by using the **contour** command:

```
contour,z2,x,y,levels=(findgen(41)-20)/10,/follow
```

or a filled contour plot:

```
contour,z2,x,y,levels=(findgen(41)-20)/10,/cell_fill
```

Both contour plots are shown in figures 1.9 and 1.10 respectively.

To generate a surface plot we use the **surface** command:

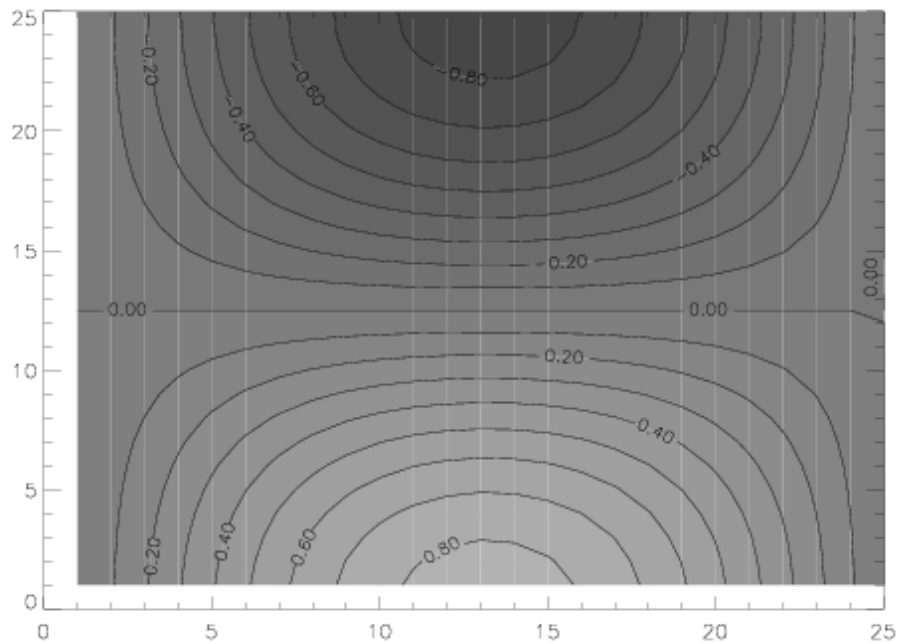


Figure 1.10: Filled contour plot example.

```
surface,z2,x,y,charsize=3.0,xtitle=('i-axis'),ytitle=('j-axis'), $
    ztitle=('k-axis'),az=35,ax=65
```

The results are shown in figure 1.11.

Note that other than **ax** and **ay** there are many attributes to change the appearance of surface plots.

The equivalent shaded surface is generated with:

```
shade_surf,z2,x,y,charsize=3.0,xtitle=('i-axis'),ytitle=('j-axis'), $
    ztitle=('k-axis'),az=35,ax=65
```

and looks like figure 1.12.

## 1.2.9 Mapping with IDL

IDL provides facilities to prepare geographical maps and to display data embedded in those maps. Maps are initialised with the **map\_set** command. Note that IDL

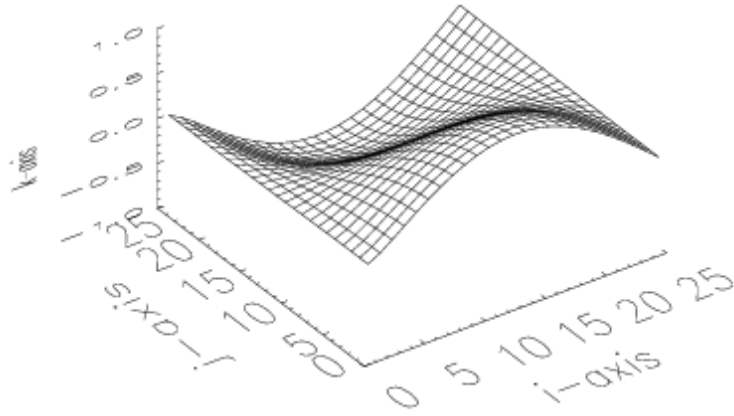


Figure 1.11: Surface plot example.

assumes that the x-coordinate represents longitude and the y-coordinate represents latitude.

Supported projection types include:

- Albers
- Azimuthal
- Conic
- Cylindrical
- Gnomonic
- Hammer
- Lambert
- Mercator
- Mollweide

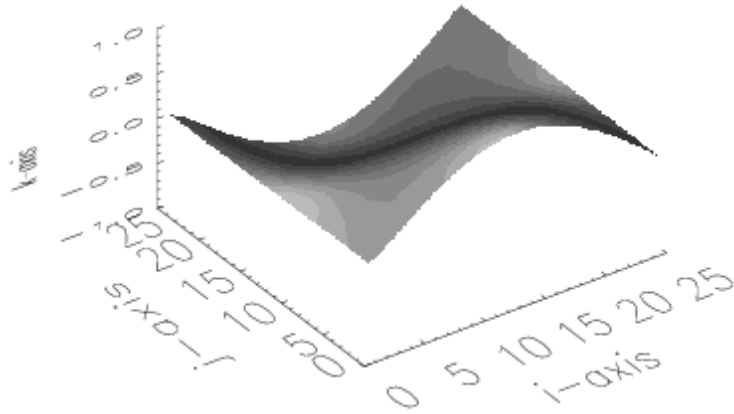


Figure 1.12: Shaded surface plot example.

- Orthographic
- Satellite
- Stereographic
- Transverse\_mercator

```

;Draw continental outlines, initialise with mercator projection, and
;plot an isotropic grid:
map_set, /continents, /lambert, /isotropic

longitude=findgen(360)-180
latitude=45*sin(longitude*!pi/180.)
oplot,longitude,latitude

```

Contour plots and filled contour maps, may also be embedded in maps.  
The map produced from the IDL script above is shown in figure 1.13:

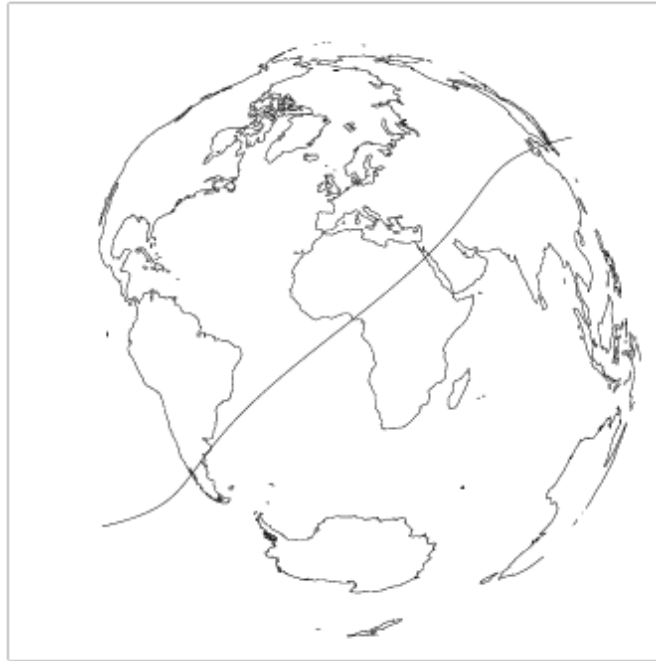


Figure 1.13: Example map plot in IDL.

**An example map:** The following IDL script (based on a script courtesy of J.L. Innis [AAD]) has been used to generate a plot of maximum and minimum seasonal ice extent for the Prydz Bay region (see figure 1.14):

```

pro idlmap_davis, hard=hard

if keyword_set(hard) then begin
  mydevice = !d.name      ;The name field of the !d system variable
                          ;contains the name of the current device.
  set_plot, 'PS'         ;Set plotting to PostScript.
  device, filename='idlmap_davis.ps'
endif

map_set,/azimu,/iso, -15,/grid,$
       londe=5,latdel=5, /conti,$
       /horiz,limit=[-70., 50., -57., 90.], /label, $
       title='Mean sea-ice edge (1973 - 1996)', $
       e_continents={fill:1}, con_color=180, mlinestyle=1, $
       mlinethick=3, lonlab=-71., latlab=49.8

```

```

; plot filled square at Mawson
  oplot, [62.857, 62.857], [-67.605, -67.605], psym=-6, symsi=0.6, $
    thick=5
  xyouts, 58.6, -68.6, 'Mawson', charsi=1.5
; plot filled square at Davis
  oplot, [78, 78], [-68.6, -68.6], psym=-6, symsi=0.6, thick=5
  xyouts, 76.3, -68.9, 'Davis', charsi=1.5

geo_lon   =[ 50, 60, 70, 80, 90];
;10deg ice-edge position from
;http://www.antcrc.utas.edu.au/~jacka/seaice.html
geo_latfeb=-1.0*[ 65.1, 65.9, 67.3, 65.6, 65.6];
geo_latsep=-1.0*[ 61.3, 60.5, 60.2, 57.5, 58.4];

; plot Feb and Sep ice edge:
oplot, geo_lon, geo_latfeb, thick=1.5, lines=2
oplot, geo_lon, geo_latsep, thick=1.5, lines=0
xyouts, 90.2, -67.0, 'February', charsi=1.5
xyouts, 90.7, -59.6, 'September', charsi=1.5

; plot a hatched area representing approximate wave source region
polyfill, [50, 60, 70, 80, 90, 90, 80, 70, 60, 50], $
  -1.0*[65.1, 65.9, 67.3, 65.6, 65.6, 58.4, 57.5, 60.2, 60.5, 61.3], $
  /line_fill

if keyword_set(hard) then begin
  device, /close          ;DEVICE closes the PostScript file.
  set_plot, mydevice      ;Return plotting to the original device.
  print
  print, 'Postscript file "idlmap_davis.ps" was written'
endif

end

```

*Acknowledgement:*

Sea-ice data are from J. Jacka's WWW page of Antarctic sea-ice extent:  
<http://www.antcrc.utas.edu.au/~jacka/seaice.html>

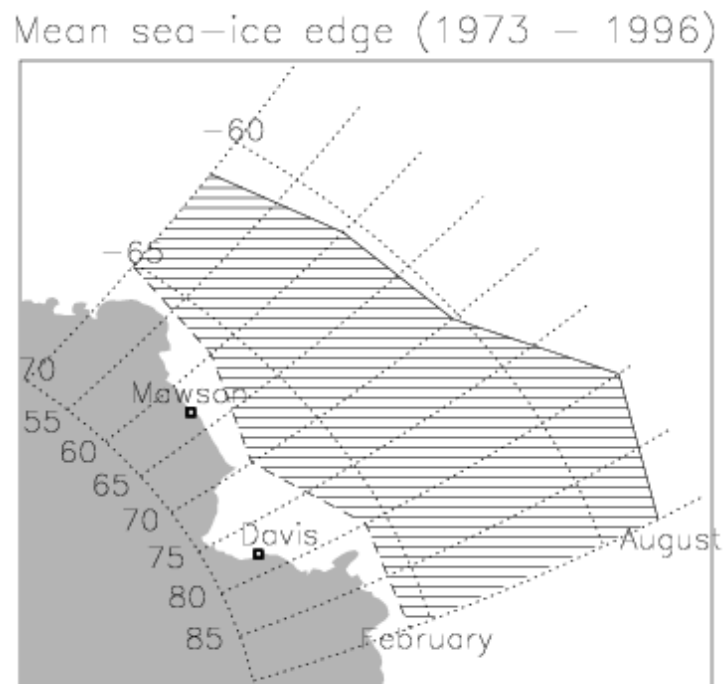


Figure 1.14: Maximum and minimum seasonal ice extent for the Prydz Bay region.

### 1.2.10 Digital output

Digital output is supported for the following formats by IDL:

- Bitmap
- BMP
- GEO TIFF
- Interfile
- JPEG
- NRIF
- PICT
- PNG
- PPM

- SRF
- TIFF
- XWD
- X11

IDL also includes routines for reading the graphic formats listed above.

Graphics may be saved in the format of a postscript file by setting the graphics device to be a postscript device and closing the device once the graphic is completed:

```
IDL> set_plot,'PS'  
IDL> device, filename='out_filename.ps'  
IDL> ... plotting commands ...  
IDL> device, /close
```

**Copyright Notice**

All information and data (including graphics) provided by the University and its staff on this Web server are, unless otherwise noted, copyright by the University of Tasmania, Australia. Unlimited distribution of University copyright material is permitted, but only if textual and graphic content is not altered and the source is acknowledged. The copyright in other material found on this server may be owned by other people, and you should get permission from them before distributing their material. Copyright 2004, University of Tasmania.