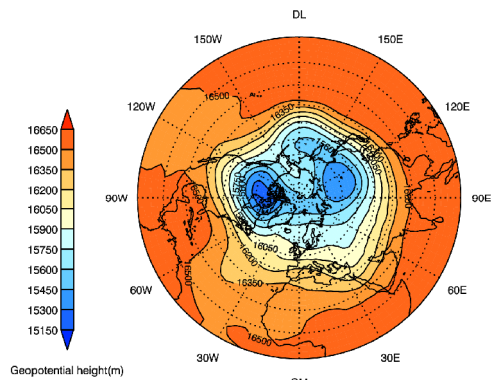
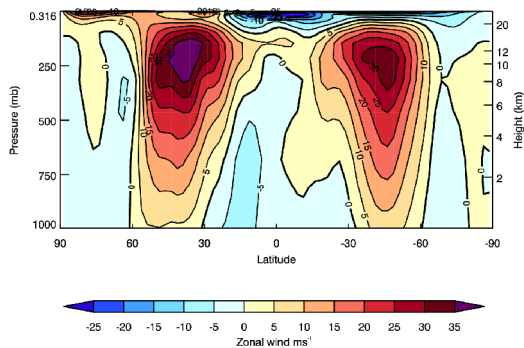
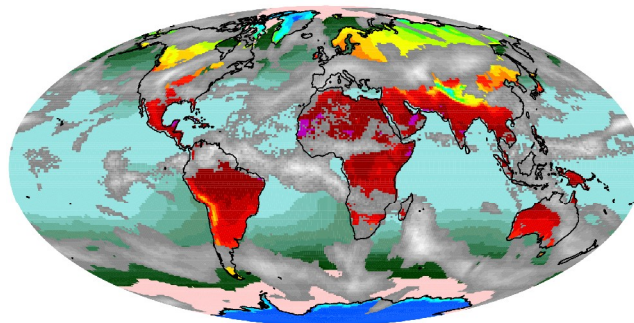
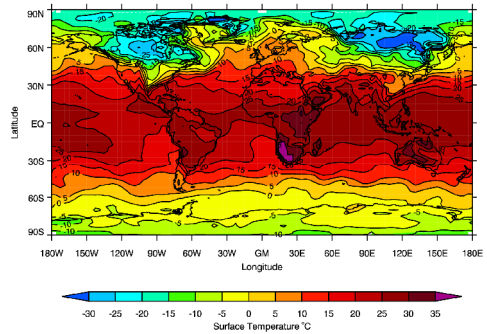
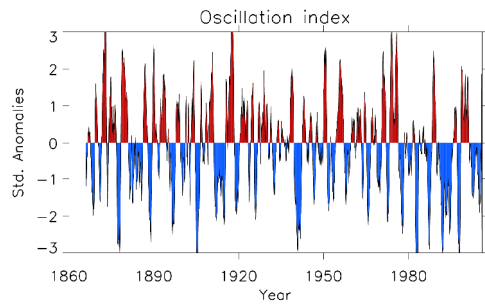


An Introduction to Using IDL in Meteorology

Version 3



Andy Heaps
a.j.heaps@reading.ac.uk
NCAS-Climate,
Department of Meteorology,
University of Reading
© NCAS, October 2006

Scope and purpose

This is a guide to using IDL as a visualisation tool in the Meteorology department at the University of Reading. It has an emphasis on visualisation and the type of plots that meteorologists commonly use.

This guide doesn't cover data manipulation. A Met Office PP model output data analysis and visualisation guide for IDL written by Jonathan Gregory is available at:

<http://ncas-cms.nerc.ac.uk> , under visualisation on the left hand side click on Guide to the Met Office PV-WAVE/IDL library.

Jonathan's guide is only available to people who have signed a Met Office collaboration agreement.

Please contact j.m.gregory@reading.ac.uk if you wish to use this software on Met Office model output.

Notation used in this guide


The guide text is in the Times New Roman font.

Unix commands are in the **bold Times New Roman font**.

IDL commands and keywords are in the **BOLD UPPER-CASE COURIER FONT**.

IDL parameters are in the **bold courier font**.

Optional IDL commands and statements are in the *bold italic Courier font*.

Programs available within this guide have the symbol  .

Good ideas to note in IDL are outlined in red.

Acknowledgements

Version 1: Many thanks are due to the group of students whose questions on IDL during the course of 2004 inspired, and formed the greater part of, this guide. In particular, Saraju Baidya, Alistair Hind and Evangelos Tyrlis are due special thanks for many interesting discussions about IDL.

Version 2: Many thanks to Mauro Dall'Amico for his excellent suggestions for improving this guide.

Version 3: Many thanks to NCAS-Climate for their probing questions leading to this version.

Contents

	Page
1. What is IDL	1
2. Adding IDL to your setup	2
3. IDL resources	2
4. Using IDL	3
5. A sample program	3
6. Compiling and running programs	4
7. Setting IDL preferences	4
8. Creating and manipulating variables	4
9. Creating and manipulating arrays	6
10. Program control statements	8
11. Scientific routines	9
12. ASCII and binary file input/output	10
13. NetCDF input/output	11
14. Postscript output	13
15. Formatting text	15
16. Colour scales	16
17. Lines, text and polygons	18
18. Graphs	18
19. Maps and contours	21
20. Maps and pattern filled contours	23
21. Maps and colour filled contours	24
22. Block fill plots	26
23. Hovmuller plots	26
24. Zonal plots	27
25. Vector plots	28
26. Interpolation	30
27. Animation	31
28. Codelets	32
29. Running IDL in Unix batch mode	33
30. Calling external objects	33
31. A Mollweide plot	34
32. Map distributions	34
33. Writing style	35
34. Irregular grids	35

1. What is IDL?

Interactive **D**ata **L**anguage, IDL, is a commercial software package for array based data analysis, visualisation and cross-platform application development. IDL is a very powerful language and, although not difficult to learn, will require a certain amount of application to get to grips with. IDL is widely used in meteorological institutes around the world, and as such, your ability to program in it will be a sound investment.

1.1 Where did IDL come from?

The founder of IDL, David Stern, started work in the late 1970s on what would become IDL while working with data from NASA's Mars Mariner 7 and 9 missions.

In September of 2000 Eastman Kodak acquired Research Systems Inc., RSI (IDL).

In March 2004 RSI (IDL) was acquired by ITT industries and is now called ITT Visual Information Systems.

IDL is widely used in Meteorology: NOAA, NASA, NIWA, ECMWF, Max Planck Institute for Meteorology, Australian Bureau of Meteorology to name a few.

The Met Office is currently using PVWAVE.

IDL are based in Boulder, Colorado, USA. American spelling is used within the IDL language so be aware that color has meaning while colour doesn't.

1.2 What is PV-WAVE and how is it related to IDL?

Precision Visuals Inc. received a copy of IDL as it stood in 1990 and enhanced it to make PV-WAVE. Since then PV-WAVE and IDL have been developing their codes separately but have the same initial source tree. While some simple code might work in both packages it can take a considerable amount of time to migrate a piece of complex code from one package to the other.

PV-WAVE have since merged with IMSL and are now called Visual Numerics Inc.

1.3 What platforms is IDL available on?

The current shipping version is 6.3 and is available on:

Unix – Sun, HP, IBM, Linux (x86 – 32 bit)

Windows NT 4.0, 2000, XP

Mac OS X

The department has 50 floating user licenses. Primarily IDL is used on the Sun Unix system although it is possible to setup a different platform to use one of the floating user licenses.

1.4. The IDL student edition

The IDL student edition is available for qualifying students for \$99 (US Dollars) and is based on IDL 6.2. The student edition is available for Windows, Linux and Mac OSX.

More information is available from: <http://www.ittvis.com>

2. Adding IDL to your setup

The programs in this guide can be accessed by doing the following in your home directory when within the department:

```
tar xf /home/cweb/live/graphics/idl_guide.tar
```

(External users can download the tarfile from <http://ncas-cms.nerc.ac.uk> Under visualisation on the left hand side click on NCAS CMS IDL Guide)

Then add the lines:

```
setup idl  
export IDL_PATH=$IDL_PATH:~/idl_guide
```

and type

```
. ~/.kshrc
```

to your .kshrc file to make all the programs in the guide available for your use.

Please exit from IDL when you are not using it as we have a limited number of licenses for use within the department.

3. IDL resources

3.1 Local online manuals

IDL have an excellent document system, which is available locally as PDF files.

At the IDL command prompt type:

```
?
```

to enter the help system. Click on view a manual and a list of available manuals will appear. Two good ones to start with are the 'Getting Started' and 'Using IDL' manuals.

The reference guide, at some 4090 pages, is for the more advanced user.

Searching for keywords and routines can be done by typing:

```
? map_set (gives the manual page for the map_set procedure)
```

```
index (gives an index of the commands available)
```

at the IDL command prompt.

3.2 Local manuals

Three good manuals are available to borrow from me in room 2L45:

'Practical IDL programming' by Liam Gumley.

'IDL programming techniques' by David Fanning.

'An Introduction to Programming with IDL' by Kenneth Bowman.

3.3 The NCAS-Climate IDL script repository

NCAS-Climate maintain a local repository of useful IDL scripts at the web address

<http://ncas-cms.nerc.ac.uk> Under services on the left hand side, click on utilities and then scripts repository.

There is a large library of software, particularly appropriate for manipulating and plotting data from Met Office models. A guide covering this library, written by Jonathan Gregory, is available at:

<http://ncas-cms.nerc.ac.uk> Under visualisation on the left hand side click on Guide to the Met Office PV-WAVE/IDL library.

Jonathan's guide is only available to people who have signed a Met Office collaboration agreement.

Please contact j.m.gregory@rdg.ac.uk if you wish to use this software.

3.4 Web resources

David Fanning's website, <http://www.dfanning.com>, is an excellent online source of links and tips. Research Systems Inc., the makers of IDL, have a set of technical tips that are useful for the more experienced user – see <http://www.rsinc.com> under the support menu.

The department has an IDL user group, which you can send questions to and tap into IDL expertise within the department. There is also a useful discussion archive for this group. Both are available at: <http://www.met.rdg.ac.uk/Lists>

3.5 The IDL demo program

Typing:

iddemo

at the Unix command prompt gives a good overview of the functionality that IDL is capable of.

4. Using IDL

4.1 Typing at the command line

IDL is an interactively compiled language i.e. each command that is typed at the command-line is compiled and run when the return key is pressed.

To use the command-line interface type:

idl

To scroll through the command history you can use the up and down keyboard keys.

4.2 Using program files

Using this mode you can use program files that you edit in your favourite editor. After a change to the program source code recompilation is needed before executing your changed program.

5. A sample program

The first program, when learning a new language, is traditionally one which writes “Hello World” on the screen. The code to do this in IDL is:

```
PRO hello
  PRINT, 'Hello World'
END
```

In the IDL language, programs start with the **PRO myname** statement and finish with the **END** statement. The **PRINT** command is used to display on the terminal text or variables, in this case the string ‘Hello World’.

Comments in IDL are prefaced by a semi-colon, **;**.

Command sometimes get too long to effectively fit on one line and can be continued to the next line using the dollar sign, **\$**.

5.1 Some useful IDL commands

PRINT, myvar – display the current contents of myvar

HELP, myvar – provide basic information on the myvar parameter

@myfile.pro – include the file myfile.pro at this point

6. Compiling and running programs

To compile and run the above program use:

```
.compile hello  
hello
```

In IDL commands prefaced by a dot are called executive commands. These commands can only be used from the IDL command-line and not within a program.

Other executive commands you might use are:

.continue

continues the execution of a program that has stopped because of an error, a stop statement or a keyboard interrupt.

.reset_session

resets much of the state of an IDL session without requiring the user to exit and restart the IDL session. See also the **.full_reset_session** command in the reference manual for a fuller description of what each command resets.

7. Setting IDL preferences

The default IDL preference file is called `.idlrc` and resides in your home directory.

We won't use this file in this guide, but be aware that such a file exists and that you might want to use it as your knowledge of IDL increases.

7.1 Adding directories to your IDL path

The environment variable `IDL_DIR` contains the search path for IDL routines. If you had a program directory `$HOME/idl_prog` that you wished to add to your search path then adding the line

```
export IDL_PATH=$IDL_PATH:$HOME/idl_prog
```

in your `.kshrc` will do this. (Don't forget to logout and back in again to start using the new directory path)

Any programs in your `$HOME/idl_prog` directory that have the matching routine name and program name can now be called no matter which Unix directory you are currently in.

8. Creating and manipulating variables

Commonly used variable types are:

Variable	bits	zero value
BYTE	8	0B
INT	16	0
LONG	32	0L
LONG64	64	0LL
FLOAT	32	0.0
DOUBLE	64	0.0D
STRING	up to 32,767 characters	' '

The default size for signed integers is 2 bytes i.e. on a Sun workstation this will be +/-32,768. Specifying a number out of this range can produce strange results.


```

a=320*160
HELP, a
A      INT      = -14336

```

To ask for a 4 byte integer add the L symbol after the variable definition as in the table above, i.e.

```

a=320*160L
HELP, a
A      LONG     =   51200

```

Also, note that integer arithmetic gives different results from floating point arithmetic.

```

PRINT, 11/2, 11.0/2
5    5.50000

```

Float values have seven significant digits and can range from 10^{-37} to 10^{38} (positive or negative). The generation of a 16 byte double float can be accomplished with

```

x=3.0D
y=3.0
HELP, x,y
X      DOUBLE   =   3.0000000
Y      FLOAT    =   3.00000

```

Formatting of strings can be done in a similar manner to FORTRAN.

```

a=-40.0
b=STRING(a, FORMAT='(f10.1)')
help, a,b
A      FLOAT    =  -40.0000
B      STRING   = ' -40.0'

```

Some useful rounding routines are:

ROUND – round to the closest integer

FLOOR – round to the nearest integer less than or equal to the argument

CEIL - round to the nearest integer greater than or equal to the argument

```

a=5.6
b=-5.6
PRINT, ROUND(a), ROUND(b)      6, -6
PRINT, FLOOR(a), FLOOR(b)     5, -6
PRINT, CEIL(a), CEIL(b)       6, -5

```

8.2 Variable conversion

Occasionally it is useful to convert one variable type into another, this can readily be accomplished using:

BYTE(myvar) - converts myvar into a byte
FIX(myvar) - converts myvar to an integer
FLOAT(myvar) - converts myvar to floating point
STRING(myvar) - converts myvar to a string
DOUBLE(myvar) - converts myvar into a double

For example:

```
a=10
b=FLOAT(a)
HELP, a, b
A      INT      =    10
B      FLOAT    =   10.0000
```

8.3 String manipulation

String manipulation in IDL is very easy and powerful. To add two strings together simply use the + operator

```
a='X axis'
b=' is the temperature'
c=a+b
HELP, c
C      STRING   = 'X axis is the temperature'
```

Other string operators that you might use are

```
b=STRTRIM(a,2) - delete the white space before and after the last characters of a.
b=STRLEN(a) - return the length of string a in b.
b=STRPOS(a,'x') - return in b the first occurrence of the string 'x' in a.
STRPUT, mystr, substr, pos - place the string substr in the string mystr starting at the position pos.
```

Using our previous example in 8.1 we can remove the white space before as below:

```
a=-40.0
b=STRTRIM(STRING(a, FORMAT='(f10.1)'),2)
help, a, b
A      FLOAT    =  -40.0000
B      STRING   = '-40.0'
```

9. Creating and manipulating arrays

9.1 Creating arrays

There are a similar set of routines to variable creation for array creation:

Variable	Array creation
BYTE	BYTARR
INT	INTARR
LONG	LONARR
LONG64	LON64ARR
FLOAT	FLTARR
DOUBLE	DBLARR
STRING	STRARR

If you remember one thing from this guide is should be that array indices in IDL start at zero.

Array indices start at zero in IDL. Forgetting this is probably the most common mistake people make when starting to use IDL.

Creating the array initializes it to zero.

```
a=INTARR(10)
PRINT, a
  0  0  0  0  0  0  0  0  0  0
```

Another way is to specify the individual array elements

```
n=[1,2,3,4]
PRINT, n, n[1:2]
  1  2  3  4
  2  3
```

9.2 Manipulating arrays

The IDL routines **INDGEN** and **FINDGEN** create sequences of numbers for integers and floating point numbers:

```
ypts1=INDGEN(6)
ypts2=FINDGEN(6)
PRINT, ypts1, ypts2
  0  1  2  3  4  5
0.00000  1.00000  2.00000  3.00000  4.00000
5.00000
```

A good use of **INDGEN** and **FINDGEN** is in the creation of regularly space sequences such as a set of latitude points.

```
ypts=90.0-INDGEN(7)*30.0
PRINT, ypts
90.0000  60.0000  30.0000  0.00000  -30.0000  -
60.0000  -90.0000
```

We could have used **INDGEN** here as an integer multiplied by a real will give a real number.

Other common examples of array manipulation:

a=a*10.0 - multiply all elements of a by 10.0

a=a-273.15 - take 273.15 off all array points

pos=WHERE(a LT 0.0) - find the positions of all points where the value is less than zero

maxval=MAX(arr, index) - Find the maximum value of arr, put this in maxval and store the index of this point in the variable index.

minval=MIN(arr, index) - Find the minimum value of arr, put this in minval and store the index of this point in the variable index.

a=REVERSE(b, dimension index) - reverse the order of one dimension of an array.

b=TOTAL(a, dimension index) - sum array over one dimension.

t2=CONGRID(t, X, Y, Z) - shrink or expand an array of points to a new size.

f=REFORM(e) - change the dimensions of an array without changing the number of elements.

Useful for removing the degenerate leading dimensions of size 1.

TRANSPPOSE(a) - returns the transpose of the array a.

ROTATE(a, direction) - rotate/transpose an array.

c=a#b - multiply the columns of the array a by the rows of b.

c=a##b - multiply the rows of the array a by the columns of b.

a=[a, a(0,*)] - add a column to the end of a.

a=[[a(*,0)], [a]] - add a row before a.

9.3 Use of the temporary function

As your use of IDL progresses you might have need of the temporary function to save memory space.

The code

```
t=FLTARR(1000000)
t=t-273.15
```

uses 1 million * 4 bytes = 4 million bytes for the first statement. The second statement creates a new array to hold the result of the subtraction and then frees the memory used by the old array version. The memory used in this code is 8 million bytes.

```
t=TEMPORARY(t)-273.15
```

uses just 4 million bytes. The temporary function can only be used where the array in question appears once on the right hand side of the equation. Multiple commands might be needed for more complex array manipulations but can substantially reduce the memory usage of your code.

10. Program control statements

Each language has its own syntax for the control of a program which although similar have slight differences. These are examples of the common ones that you are likely to use in IDL:

```
IF x LT 12.0 THEN y=0.8
IF (x GT 5) THEN BEGIN
    idl statements
ENDIF
FOR I=0,9 DO BEGIN
    idl statements
ENDFOR
REPEAT BEGIN
    idl statements
ENDREP UNTIL (x EQ 27)
IF (x LT 12.0) THEN BEGIN
    idl statements
ENDIF ELSE BEGIN
    idl statements
ENDELSE
WHILE (x LE 0) DO BEGIN
    idl statements
ENDWHILE
```

Here is an example of a loop creating a filename that can be used to read ERA40 data from a machine called charney:

```
FOR year=1979,1985 DO BEGIN
stryear=STRTRIM(STRING(year),2)
myfile='/export/charney/data-01/era-40/monthly_means/'+stryear+'
    '/ggapjan'+stryear+'.nc'
PRINT, 'reading ', year, myfile
ENDFOR
```



Gives the screen output

```
reading 1979 /export/charney/data-01/era-40/monthly_means/1979/ggapjan1979.nc
reading 1980 /export/charney/data-01/era-40/monthly_means/1980/ggapjan1980.nc
reading 1981 /export/charney/data-01/era-40/monthly_means/1981/ggapjan1981.nc
reading 1982 /export/charney/data-01/era-40/monthly_means/1982/ggapjan1982.nc
reading 1983 /export/charney/data-01/era-40/monthly_means/1983/ggapjan1983.nc
reading 1984 /export/charney/data-01/era-40/monthly_means/1984/ggapjan1984.nc
reading 1985 /export/charney/data-01/era-40/monthly_means/1985/ggapjan1985.nc
```

A second example is presented in this code that lists in order the filenames for 6 hourly data in files for the 850 mb zonal wind for the months May to October and the years 1979 to 1980. This shows the flexibility of using IDL to correctly name files for reading in and doing running means on real data. (Neither the program or output are listed here for brevity.)



10.1 Speeding up loops

Array expressions execute faster than loops and conditional statements. If your program is running slowly, you might like to consider using the **WHERE** array operator.

The example below shows this in action on an array of 100 million elements. It checks to see if the element is below zero and if so sets the corresponding mask value to be 1. This example could be a calculation of the land-sea mask, for instance.

```
ix=10000
iy=10000
FOR i=0,ix-1 DO BEGIN
FOR j=0,iy-1 DO BEGIN
  IF a(i,j) GT 0 THEN b(i,j)=1
ENDFOR
ENDFOR
```



Using the **WHERE** function this code can be simplified to:

```
land=WHERE(a GT 0.)
b(land)=1
```

On a reasonably fast machine the top code took 65 seconds and that on the bottom took just 4 seconds. There are variations on a theme here with other methods being **b=a GT 0** and **b(*)=a GT 0**.

Use array based operators to get the fastest code.

11. Scientific routines

IDL comes with the normal set of scientific functions, which can be called as listed below.

- SIN(x)**, **COS(x)**, **TAN(x)** - sine etc.
- ASIN(x)**, **ACOS(x)**, **ATAN(x)** - arcsine etc.
- SINH(x)**, **COSH(x)**, **TANH(x)** - hyperbolic sine etc.
- SQRT(x)** - returns the square root of x.
- a^b** - a raised to the b power.
- CEIL(x)** - returns the closest integer greater than or equal to the argument.
- FLOOR(x)** - returns the closest integer less than or equal to the argument.
- ROUND(x)** - rounds the argument to the closest integer.
- EXP(x)** - returns the natural exponential function of x.
- ALOG(x)**, **ALOG10(x)** - return the natural logarithm and logarithm to the base 10 of x.
- !PI**, **!DPI** - single and double precision values for π .
- mag = SQRT(u^2.+v^2.)** - gives the magnitude of the wind.

11.1 Other routines

There are a host of other routines in IDL to do correlations etc. Care is needed so that you understand how your data is being changed. IDL core routines such as mean, fft etc. can cause unexpected results if you have a routine of your own with the same name.

Don't give your routine the same name as an IDL core routine.

12. ASCII and binary file input/output

IDL recognizes both free and explicit format ASCII files.

12.1 ASCII input

There are three commands for reading ASCII data

READ – read free format data

READF – read fixed format data

READS – read free format data from a string variable

A typical sequence of opening, reading data from and closing a file would be:

```
OPENR, 1, 'ascii.dat'  
READF,1, mydata  
CLOSE,1
```

This example explicitly uses logical unit number 1 for reading the data.

A more general method would be:

```
OPENR,lun, 'ascii.dat', /GET_LUN  
READF,lun, mydat  
CLOSE, lun  
FREE_LUN, lun
```

The **/GET_LUN** switch picks a free logical unit number from the pool of free units and places it in the integer variable lun. If mydata is undefined then it will just read the first variable from the file ascii.dat whether this is integer or floating point and place it in the variable mydata.

As an example of reading some data the file comp.txt contains data in the form:

```
year   value1   value2  
1880 -0.0228571 -0.0600000  
1881 -0.0600000 0.0900000 etc.
```

To read this data we create three IDL arrays, form a loop over the rows of data to be input, read the data into dummy variables and place the data from these into the IDL. Reading ASCII data directly into the arrays can give confusing results and should be avoided as is shown in read_row.pro.

```
OPENR,1,'comp.txt'  
mystr=""  
READF,1,mystr ; read titles  
mytime=INTARR(121)  
temp1=FLTARR(121)  
temp2=FLTARR(121)  
FOR i=0,120 DO BEGIN
```



read_row.pro

```

READF, 1, val1, val2, val3
mytime(i)=val1
temp1(i)=val2
temp2(i)=val3
ENDFOR
CLOSE, 1

```

12.2 ASCII output

There are two ways of opening files for writing in IDL,
OPENW – open for write (if the file exists overwrite it)
OPENU – open a file for update.

To open and write a data file an example would be:

```

OPENW, lun, 'myfile.dat', /GET_LUN
PRINTF, lun, mydat
CLOSE, lun

```

12.3 Binary files

Binary files are read and written in the same manner as ASCII files but the read/write statements are changed to **READU** / **WRITEU**.

```

OPENR, lun, 'binary.dat', /GET_LUN, /F77_UNFORMATTED
READU, lun, data
CLOSE, lun

```

Other file opening switches that you might find useful are:

```

/SWAP_ENDIAN
/SWAP_IF_LITTLE_ENDIAN
/CLOBBER
/NOCLCLOBBER

```

Sun SPARC computers and SGI Origin systems are big endian.
SUN AMD, PCs and SGI Altix systems are little endian.

12.4 Met Office PP files

Met Office PP files are in binary format but we won't cover them here, as there is an extensive PP file IDL library available already. The guide to these routines, written by Jonathan Gregory, is available at: <http://ncas-cms.nerc.ac.uk> Under visualisation on the left hand side click on Guide to the Met Office PV-WAVE/IDL library.

Jonathan's guide is only available to people who have signed a Met Office collaboration agreement. Please contact j.m.gregory@reading.ac.uk if you wish to use this software on Met Office model output.

13. NetCDF input/output

The Network Common Data Format is a common way for scientists to exchange data. This format uses XDR (eXternal Data Format) to make this self describing binary format machine independent.

All IDL NetCDF routines begin with **NCDF_** .

13.1 Finding out what is in your NetCDF file

Typing:

ncdump -c file.nc

on the Unix command-line will print off the header and coordinate information for the NetCDF file. You will be able to see the names, dimensions and units of data variables within your NetCDF file.

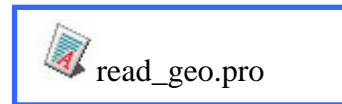
If you want to see the full contents then type:

ncdump file.nc > myoutput

13.2 Reading NetCDF data

The following code reads an array of geopotential height data (96x73x22 points) from the file ukmo.nc and stores it in the array geo. It also reads in the pressure levels and prints them out.

```
fid=NCDF_OPEN('ukmo.nc')
varid=NCDF_VARID(fid,'geopotential')
NCDF_VARGET, fid, varid, geo
varid=NCDF_VARID(fid,'p')
NCDF_VARGET, fid,varid, pressure
PRINT, 'pressure levels are', pressure
NCDF_CLOSE, fid
```



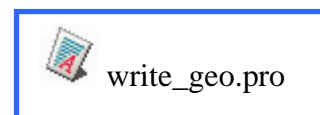
Some NetCDF files have a scale and offset parameter and these should obviously be read and used to get the correct data values.

13.3 Writing NetCDF data

The following code writes out a NetCDF file of geopotential height data in the Meteorological Office assimilated data format:

Note that no spaces are allowed in variable names.

```
geo=FLTARR(96,73,22)
lats=90-FINDGEN(73)*2.5
longs=FINDGEN(96)*3.75-180.0
levs = ['1000.','681.','464.','316.','215.','146.','100.','68.','46.','31.6','21.5'$
        ',14.6','10.0','6.81','4.64','3.16','2.15','1.46','1.0','0.68','0.464','0.316']
fid=NCDF_CREATE('ukmo.nc', /CLOBBER)
longid=NCDF_DIMDEF(fid,'longitude',96)
latid=NCDF_DIMDEF(fid,'latitude',73)
levid=NCDF_DIMDEF(fid,'level',22)
geovid=NCDF_VARDEF(fid,'geopotential', [longid,latid,levid],/FLOAT)
longvid=NCDF_VARDEF(fid,'longitude', [longid],/FLOAT)
latvid=NCDF_VARDEF(fid,'latitude', [latid],/FLOAT)
levvid=NCDF_VARDEF(fid,'level', [levid],/FLOAT)
NCDF_CONTROL, fid, /ENDEF
NCDF_VARPUT, fid, longvid, longs
NCDF_VARPUT, fid, latvid, lats
NCDF_VARPUT, fid, levvid, levs
NCDF_VARPUT, fid, geovid, geo
NCDF_CLOSE, fid
```



13.4 IDL NetCDF routines

Reading NetCDF Files:

The following commands should be used to read data from a NetCDF file:

- NCDF_OPEN** - Open an existing NetCDF file.
- NCDF_INQUIRE** - Find the format of the NetCDF file.
- NCDF_DIMINQ** - Retrieve the names and sizes of dimensions in the file.
- NCDF_VARINQ** - Retrieve the names, types, and sizes of variables in the file.
- NCDF_ATTNAME** - Optionally, retrieve attribute names.
- NCDF_ATTINQ** - Optionally, retrieve the types and lengths of attributes.
- NCDF_ATTGET** - Optionally, retrieve the attributes.
- NCDF_VARGET** - Read the data from the variables.
- NCDF_CLOSE** - Close the file.

Creating NetCDF files

The following commands should be used to write data to a NetCDF file:

- NCDF_CREATE** - Create a new file. The new file is put into define mode.
- NCDF_DIMDEF** - Create dimensions for the file.
- NCDF_VARDEF** - Define the variables to be used in the file.
- NCDF_ATTPUT** - Optionally, use attributes to describe the data.
- NCDF_CONTROL, /ENDEF** - Call **NCDF_CONTROL** and set the **ENDEF** keyword to leave define mode and enter data mode.
- NCDF_VARPUT** - Write the appropriate data to the NetCDF file.
- NCDF_CLOSE** - Close the file.

14. Postscript output

Some IDL users like to view their output on the screen to debug their code more quickly; others like to work entirely in postscript. I recommend you use just postscript and a line such as

```
SPAWN, 'display myplot.ps'
```

to the end of your IDL program to use the ImageMagick command **display** to view the file on the screen.

14.1 The basics

To get postscript output from your program have the following lines near the top of your program:

```
SET_PLOT, 'ps'  
DEVICE, FILE='output.ps'  
and at the bottom:  
DEVICE, /CLOSE
```

Remember to use the **DEVICE, /CLOSE** command to properly close your postscript file.

If your program doesn't reach the **DEVICE, /CLOSE** statement then the postscript file will be partially formed and won't print properly.

14.2 device options that you might find useful

/COLOR - (note spelling) turn on colour postscript output (off by default).

/BITS_PER_PIXEL=8 - use 256 colours as per Sun display, useful for displaying images in postscript.

/ENCAPSULATED - produce encapsulated postscript for import in LaTeX etc. Files so produced will not print on a printer.

/LANDSCAPE - use landscape mode instead of the default portrait mode.

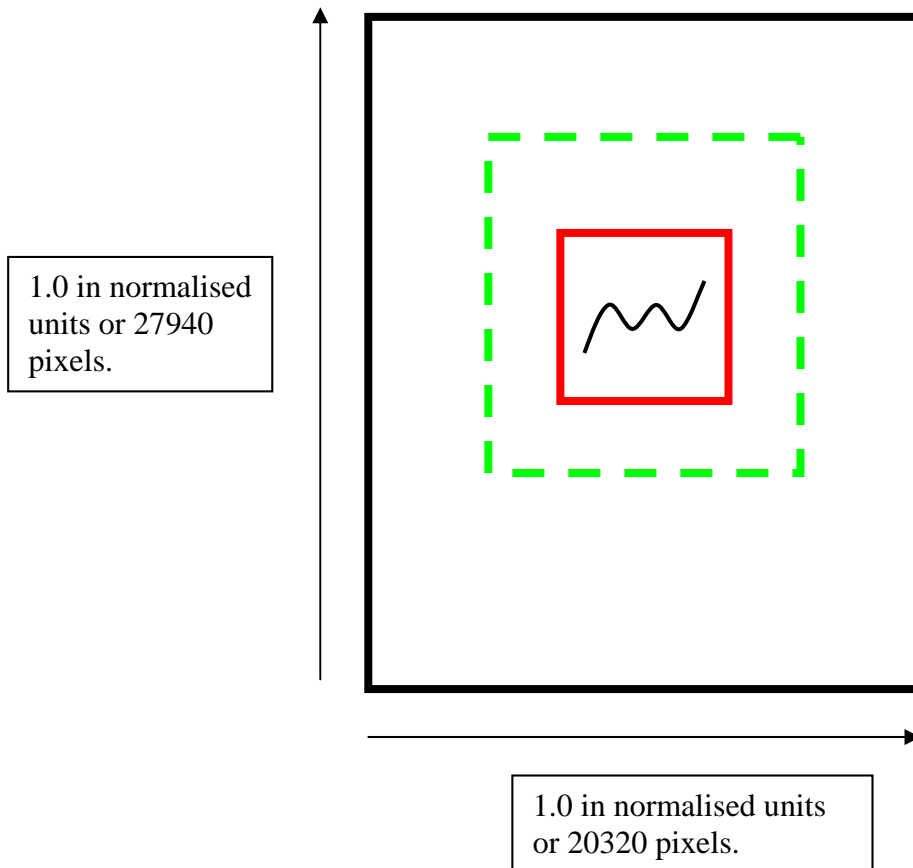
The default postscript output size is 7"x5".

A set of options to print to an A4 portrait colour postscript file would be:

```
DEVICE, FILE='output.ps',, /INCHES, XSIZE=8.27, YSIZE=11.69, $  
XOFFSET=0.5, YOFFSET=0.5, /COLOR
```

14.3 Coordinate systems

Using the above A4 settings our coordinate systems look as follows:



The green area is **!P.REGION** with its space for axes and annotation. Of more use is the actual plot area outlined in red, the plot region **!P.POSITION**. The plot position is always set in normalised units using **!P.POSITION** and then the data values are assigned to it using the plotting process. This will become more apparent later when we make our first plot.

The three coordinate systems you will use are device, normalised and data coordinates. The **/DEVICE**, **/NORMAL** and **/DATA** switches to many IDL plotting commands specify which coordinate system you would like IDL to use.

To convert between the different coordinate systems use the **CONVERT_COORD** command:

```
d=CONVERT_COORD([0,3.75], [0,0], /DATA, /TO_DEVICE)
```

will convert the data coordinate points point1=(0,0) and point2=(3.75,0) into device coordinates. The keywords **/DATA** and **/TO_DEVICE** can be changed to any of the three coordinate systems so giving a very flexible command.

14.3 Setting the plot area

The plot area is set using the `!P.POSITION=[x0,y0,x1,y1]` command.

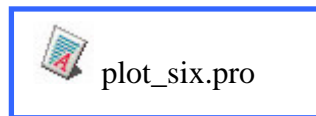
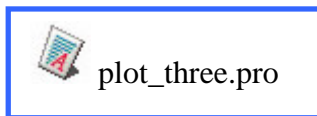
For example: A single stereographic plot is needed.

We specify an A4 sized postscript file (as above) and get the x and y sizes of the paper using `!D.X_VSIZE` and `!D.Y_VSIZE`. We then select a large plot size of 18000 pixels for our plotting area. As both the x and y plotting sizes are the same the final plot will have a circular stereographic projection.

```
xsize=!D.X_VSIZE ;Paper x size
ysize=!D.Y_VSIZE ;Paper y size
xstart=(1000.)/xsize
xfinish=(1000.+18000.)/xsize
ystart=(1000.)/ysize
yfinish=(1000.+18000.)/ysize
!P.POSITION=[xstart, ystart, xfinish, yfinish]
```

14.4 Multiple plots on a page

Multiple plots on a page can be made using the above method. Examples of three and six plots on a page are available in this guide.



IDL has another way of doing plot positioning via the `!P.MULTI` command. This often confuses the most seasoned IDL programmers and it is recommended you avoid using this command.

14.5 Importing figures into documents

The best way to do this is to produce a postscript file and then to use the Unix command `convert` to produce a png image file. Image files can easily be loaded into Word, PowerPoint and LaTeX. If you use a reasonable resolution (say 250 dots per inch) then the image is virtually indistinguishable from an incorporated postscript file. All the plots in this guide are png files at 250 dots per inch.

`convert -density 250 myplot.ps myplot.png`

[You might need to add `-page A4` to the options list sometimes to see the full image. Use `xv` to find the coordinates of the cropping region and then feed these back into the previous `convert` command]

Encapsulated postscript files won't load directly into PowerPoint and can cause screen refresh problems in Word if they are very complicated. Journals will, however, ask you for postscript copies of your files when you publish a paper.

15. Formatting text

Text formatting is available in IDL to make up various characters that you might use in your plots i.e. superscripts - 12°C, subscripts – CO₂, ñ as in El Niño.

The following text operators are those you will probably use:

!E – superscript mode.

!I – subscript mode.
!N – Return to normal mode.
!X – return to the entry font.
!! – display the ! symbol.
!C – Begin a new line of text.

Further information is available in the fonts chapter in the IDL reference manual.

Superscripts and subscripts are straightforward to use:

```
mydeg="!Eo!N"  
myco2= CO!I2!N
```

To get other characters such as ñ in postscript output is more difficult. First of all you will have to set the font to be isolatin 1 to put IDL in Unicode compatible mode. Then you have to look for the relevant Unicode symbol at <http://www.unicode.org/charts>. To display a ñ in postscript:

```
DEVICE, /ISOLATIN1  
XYOUTS, 0.5,0.7, "E1 Ni!Z(00f1)ο", FONT=1,COLOR=1, /NORMAL
```

The degree symbol above is not a true degree symbol and you might want to select the relevant Unicode symbol.

David Fanning has a more complete discussion on fonts on his website at http://www.dfanning.com/graphics_tips/lesign.html.

Unless you are labelling axes etc. you might find it easier to add unusual characters in the word processing package you are using for the final document.

16. Colour scales

IDL colour scale can easily be defined in terms of red, green and blue vectors on an integer scale of 0 to 255. Keeping the first two colours as white and black will mean that your scales are interchangeable and that text always comes out as black when specifying **COLOR=1**. This can be set through the whole program by using **!P.COLOR=1** so that you don't have to specify the text or line colour on commands.

16.1 Defining a colour scale

A commonly used colour scale is:

```
r=[255,0,0,0,0,0,66,184,245,255,255,255,230,191,148,105,161,184]  
g=[255,0,84,199,255,255,255,255,255,209,135,28,0,0,0,0,0,0]  
b=[255,0,255,255,255,178,0,0,0,0,0,0,0,0,0,0,20,135,186]
```

This sets the number of colours to be 18 with the first being white and the second black.

We can load this colour scale into IDL using:

```
TVLCT, r,g,b
```

Colour printers don't give an accurate rendition of the screen colours so testing your colour scale on a colour printer or projector is a good idea.

Choosing a colour past the last defined colour in your colour scale will cause the colours to cycle through your colours again leading to strange effects.

The ColorBrewer webpage <http://www.colorbrewer.org> is an excellent starting point for choosing a colour scale.

There are also some nice scientific colour tables available at http://geography.uoregon.edu/datagraphics/color_scales.htm.

Picking your own colours in rgb can also be done using the rgb values in the image file rgb.png.

You might like to define your colourscale in a more readable fashion as in the code snippet mycols.pro and include them in your IDL program with `@mycols.pro`. This makes adding and removing colours easy as well as the code more readable. See mycols.pro for more details.

16.2 Plotting your colour scale

A simple colourbar routine is included in this guide and is called colbar.

colbar.pro has the following calling parameters:

`COLBAR, coords, levs, title, /LEFT, /RIGHT, /UP, /DOWN, /TEXTPOS, /ALT`

`coords` is an array [`x0, y0, x1, y1`] where `x0, y0` are the lower left and `x1, y1` are the upper right of the colour bar in normalised coordinates.

`levs` is an array of levels.

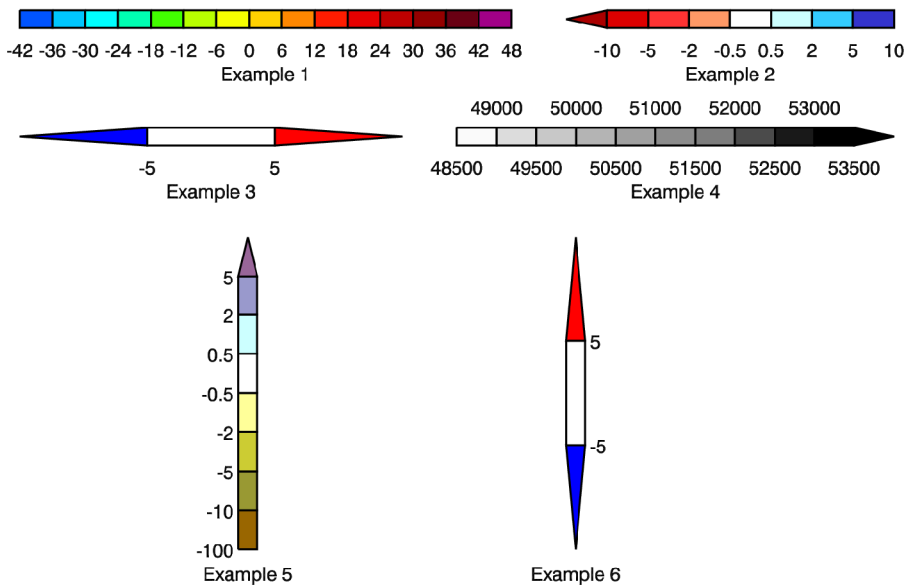
`title` is the title for the colour bar.

`/LEFT, /RIGHT, /UP, /DOWN` are the scale extensions (the triangles on the colourbar ends).

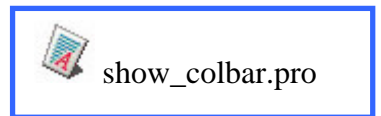
`/TEXTPOS` switches the text to be on the right hand side for vertical colourbars.

`/ALT` alternates the side of the text in horizontal colour bars

A selection of colourbars made using the `COLBAR` procedure are shown below.



colbar.pro



show_colbar.pro

17. Lines, text and polygons

Lines, text and polygons are the basics of making good plots.

17.1 Lines

Lines are drawn with the `plots` procedure, which has the syntax:

PLOTS, x, y

x and **y** are vectors containing the x and y coordinates to be connected.

Other keywords you might use are:

COLOR=n – use the nth colour from your colour table.

THICK=n – change the line thickness, default is 1.0.

LINestyle=n – Change the line style, default is 0, meaning a solid line. Dotted lines have the value 1 and dashed lines that of 2.

17.2 Text

Text is drawn using the `xyouts` procedure, which has the syntax

XYOUTS, x, y, string

x and **y** are the x and y coordinates of the test position to place the text in **string**.

Other keywords you might use are:

ALIGNMENT=a - 0.0 left justifies, 0.5 centres and 1.0 right justifies the text.

CHARSIZE=a – character size, 1.0 is the default.

CHARTHICK=a – character thickness, the default is 1.0.

COLOR=n – use the nth colour from your colour table.

17.3 Polygons

Filled polygons are drawn with the `polyfill` procedure, which has the syntax:

POLYFILL, xpts, ypts, COLOR=n

xpts and **ypts** are the x and y values to be joined together.

n is the nth colour from the colourtable.

Other keywords you might use are:

/LINE_FILL – fill polygon with a set of lines.

/SPACING – spacing between the lines.

18. Graphs

Now we know the basics of IDL we are ready to make our first plot. This is how I would recommend that you'd make a plot:

Open a postscript file.

Read in the data.

Make the plot without axes.

Add the axes.

Close the postscript file.

Here we will plot surface temperature against latitude along the Greenwich Meridian. When setting up the plot we use **XSTYLE=5**, **YSTYLE=5** to force exact axis range to be what we have defined and to turn off plotting of the axes.

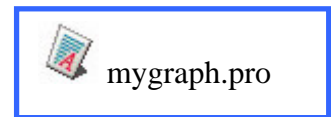
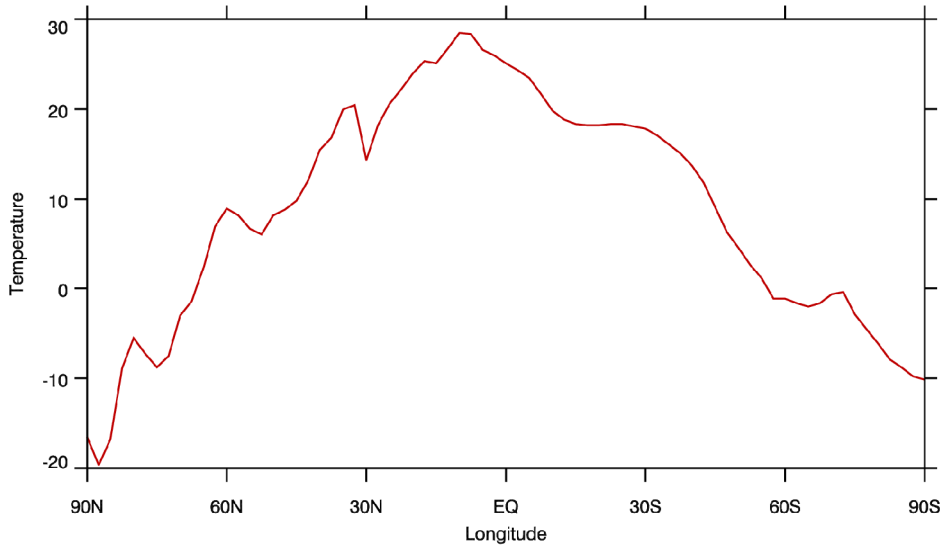
Graphs are plotted using the IDL procedure `PLOT`.

```

;Read in the data
fid=NCDF_OPEN('ukmo.nc')
varid=NCDF_VARID(fid,'temp')
NCDF_VARGET, fid, varid, data
NCDF_CLOSE, fid

;Select data from North pole to south pole on the equator
;and at the surface. Also convert to Celsius.
mytemp=REFORM(data(0,*,0))-273.15
PLOT, INDGEN(73), mytemp, XSTYLE=5, YSTYLE=5, COLOR=13,$
      YRANGE=[-20.0, 30.0]

```



Axes are drawn with the **AXIS** command. Drawing the bottom x axis for example is done using:

```

myticks=['90N','60N','30N','EQ','30S','60S','90S']
myticklen=-0.03
mytickvals=[0,12,24,36,48,60,72]
AXIS, XAXIS=0, XTITLE='Longitude', TICKLEN=myticklen,$
      XTICKS=6, XTICKNAME=myticks, XTICKV=mytickvals

```

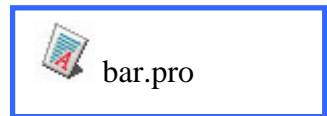
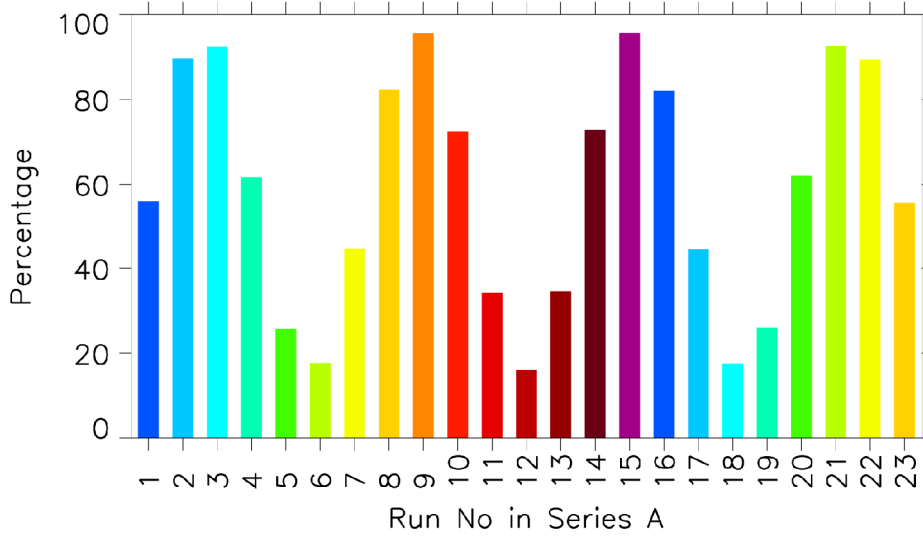
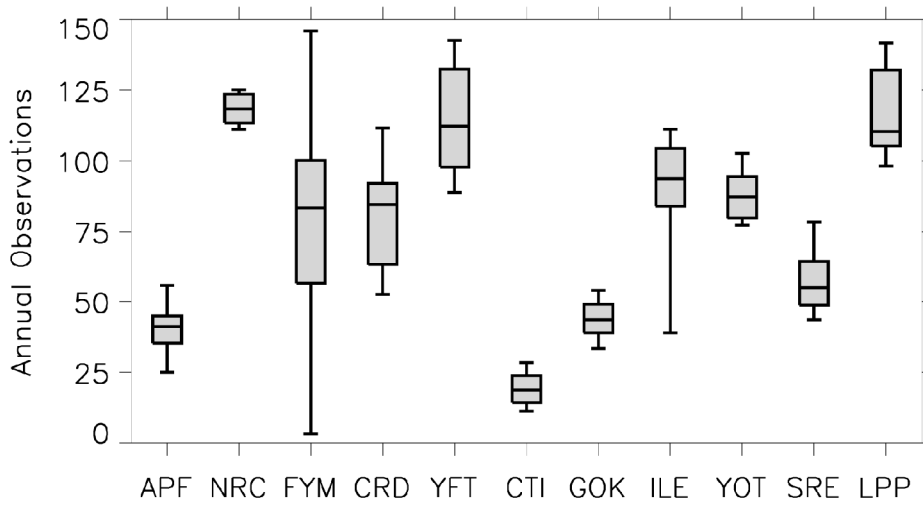
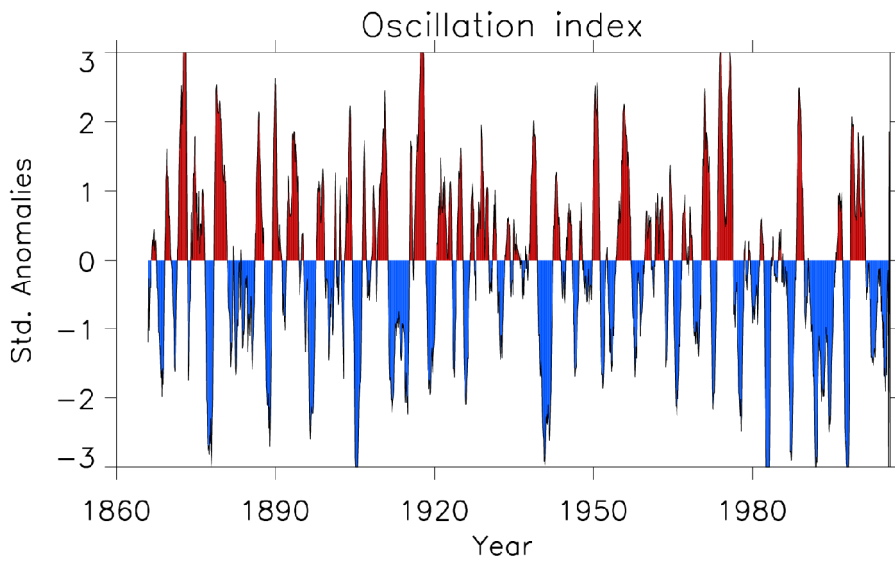
Setting the tick length to be negative makes the tick marks come out of the plot area giving a cleaner plot. The tick annotation (**myticks**) are then plotted at the tick values (**mytickvals**) along the axis.

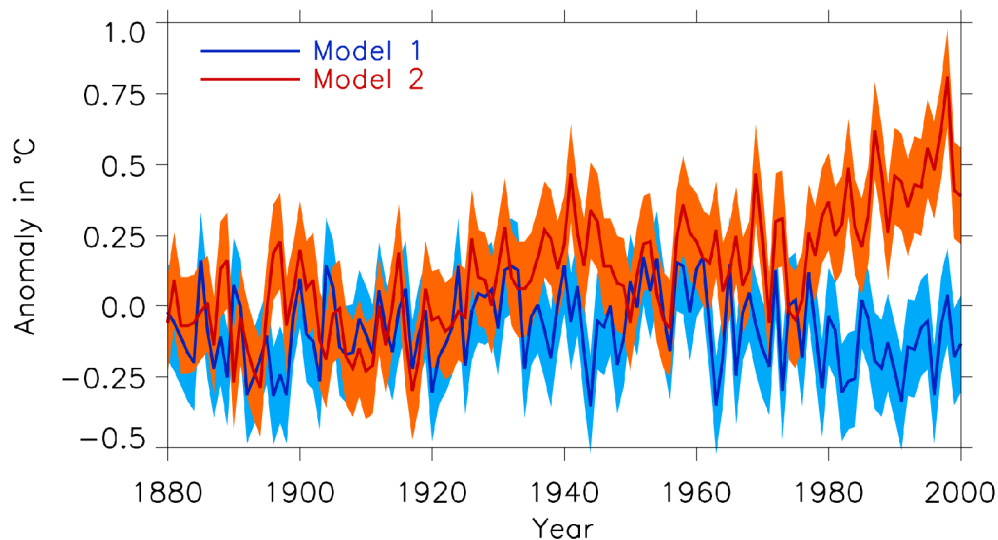
The xticks or yticks value is the number of tick intervals not the number of ticks – often a source of confusion.

Other keywords to the **PLOT** command you might use are:

- PSYM=n** – add a symbol to each plotted point, 1=plus, 2=dot, 3=diamond, 4=triangle etc. If n is negative then the line is drawn as well as the symbol.
- SYMSIZE=a** - size of the symbol
- LINestyle=n** – linestyle, 0=solid, 1=dotted, 2=dashed, 3=dash dot etc.
- XRANGE=[x0, x1], YRANGE=[y0, y1]**- specify the range of the data
- /XLOG, /YLOG** – use a logarithmic x and y axis.

There are many sorts of graphs that can be created using lines, text and filled polygons. Here are a few more examples.





The following sections show how to produce common meteorological plots using IDL. If you are using Met Office PP data please see the guide written by Jonathan Gregory available at: <http://ncas-cms.nerc.ac.uk> Under visualisation on the left hand side click on Guide to the Met Office PV-WAVE/IDL library. Contact j.m.gregory@rdg.ac.uk if you wish to use this software on Met Office model output.

19. Maps and contours

The most common plot in meteorology is the cylindrical projection so let's concentrate on this to start with. An example NetCDF dataset from a Met Office climate model run has the 2m temperature data in Kelvin on the grid 90°N to 90°S in steps of 2.5° and from 0°E to 356.25°E in steps of 3.75°. We can read this data in with the following code:

```
fid=NCDF_OPEN('ukmo.nc')
varid=NCDF_VARID(fid,'temp')
NCDF_VARGET, fid, varid, data
NCDF_CLOSE, fid
```

To select surface temperature and convert from Kelvin to Celsius we use:
`temp=data(*,*,0)-273.15`

IDL likes the data to be wrapped around the Greenwich Meridian. We can do this for this dataset using:
`temp=[temp,temp(0,*)]`
 We also add an extra point to the array of longitudes as shown in the program code.

Wrap your data around the Greenwich Meridian or you'll have a strip of plot without contours just to the west of your starting longitude.

Setting the map projection is done with the IDL procedure
`MAP_SET, lat, lon, LIMIT=[lat0, lon0, lat1, lon1], /CYLINDRICAL`

`lat, lon` are the latitude and longitude of the centre point of the plot and
`[lat0, lon0, lat1, lon1]` are the lower left and upper right limits on the viewing box.

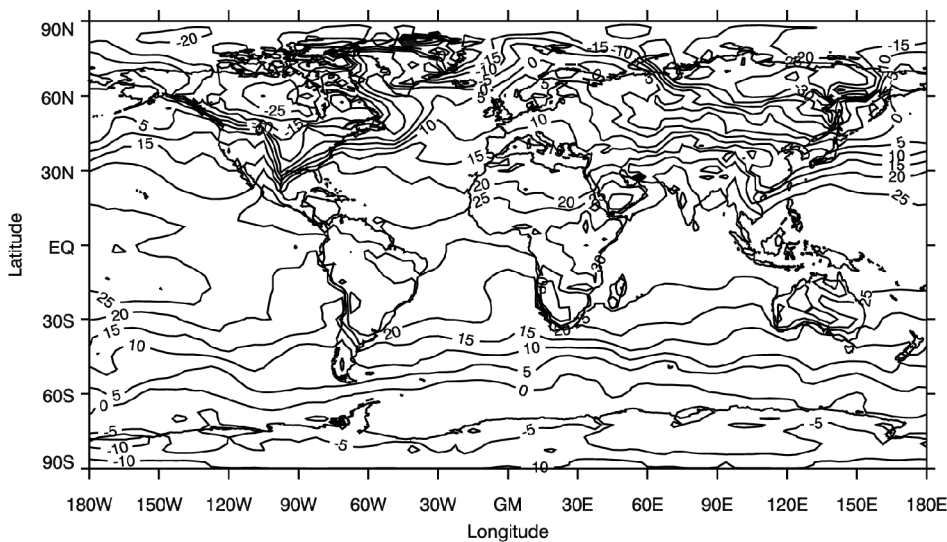
Always set the latitude mid-point of your data to be 0.0 in cylindrical plots.
If you don't do this, plots from say 30N to 90N will have curved latitude lines.

```
levs=-40+FINDGEN(16)*5.0 ;specify the contour levels  
labs=STRTRIM(STRING(levs),2) ;specify the labels
```

Contour labels are controlled by the `C_ANNOTATION` parameter. In our plot we have 16 levels so to plot the labels for each level our contour statement will be:

```
CONTOUR, temp, longs, lats, LEVELS=levs, XSTYLE=5, YSTYLE=5,$  
C_ANNOTATION=labs, /OVERPLOT
```

The `/OVERPLOT` keyword to the contour procedure preserves the mapping that we have set and plots the contours over the top. Setting `XSTYLE` and `YSTYLE` to be 5 plots the data exactly to the limits specified by the map projection. We then add the axes as before using the `AXIS` command as before.

 `cylin1.pro`

Other useful keywords are:

- `C_ANNOTATION` - annotation to be drawn on each contour (array).
- `C_COLORS` - colour of each contour line (array).
- `C_THICK` - thickness of each contour (array).
- `C_LINestyle` - the linestyle of each contour (array).

Avoid the using `NLEVS` and `C_LABELS` in the `CONTOUR` command as these can produce confusing results.

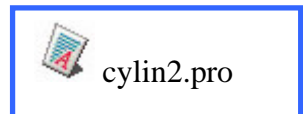
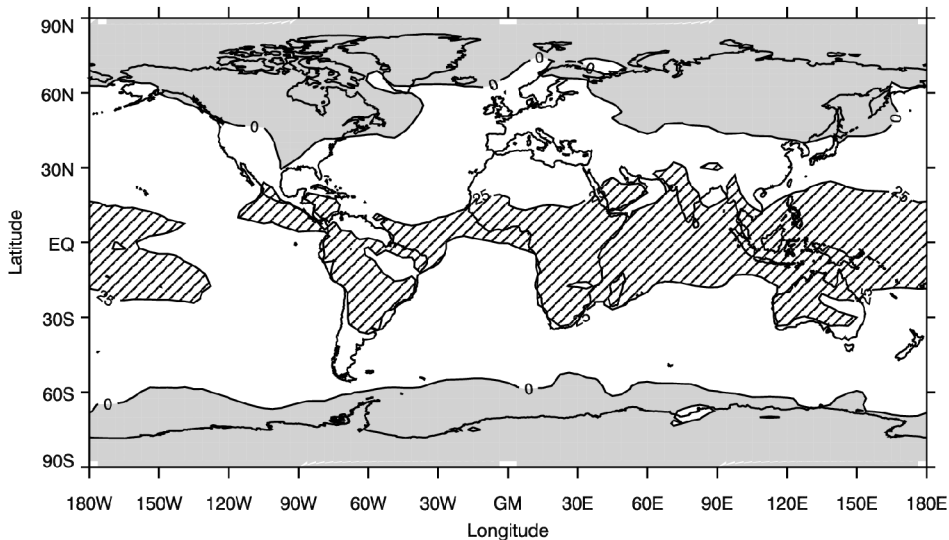
20. Maps and pattern filled contours

A good way to highlight a part of the plot without using colour (journals often charge for colour pages) is to use either a pattern or grey fill. The following code shows the use of this facility in IDL in shading the temperature between 25 and 60 degrees Celsius with a pattern and below zero degrees Celsius with a grey colour. A second call to **CONTOUR** is made to add the contours lines and labels.

```
levs=[-100, 0]
labs=STRTRIM(STRING(levs),2)
CONTOUR, temp, longs, lats, LEVELS=levs, XSTYLE=5, YSTYLE=5,$
        /CELL_FILL, C_COLORS=[2,3], /OVERPLOT
CONTOUR, temp, longs, lats, LEVELS=levs, XSTYLE=5, YSTYLE=5,$
        C_ANNOTATION=labs, /OVERPLOT
```

```
levs=[25,60]
labs=STRTRIM(STRING(levs),2)
CONTOUR, temp, longs, lats, LEVELS=levs, XSTYLE=5, YSTYLE=5,$
        C_ORIENTATION=45.0, /CELL_FILL, /OVERPLOT
CONTOUR, temp, longs, lats, LEVELS=levs, XSTYLE=5, YSTYLE=5,$
        C_ANNOTATION=labs, /OVERPLOT
```

```
MAP_CONTINENTS          ;overlay the continents
```



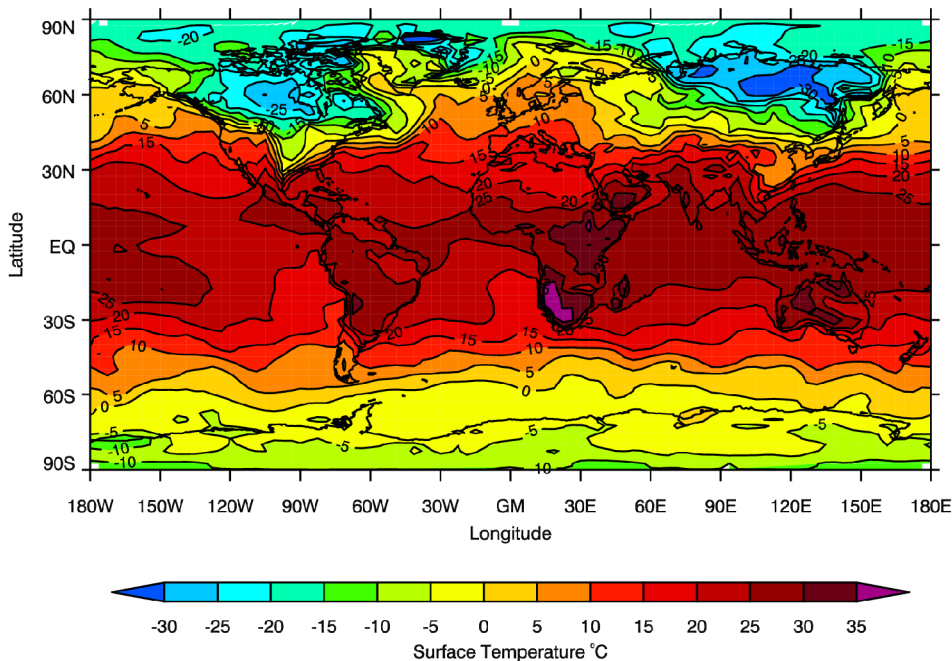
A keyword associated with pattern fill is **C_SPACING** – the distance in cm between the lines used to fill the contours.

21. Maps and colour filled contours

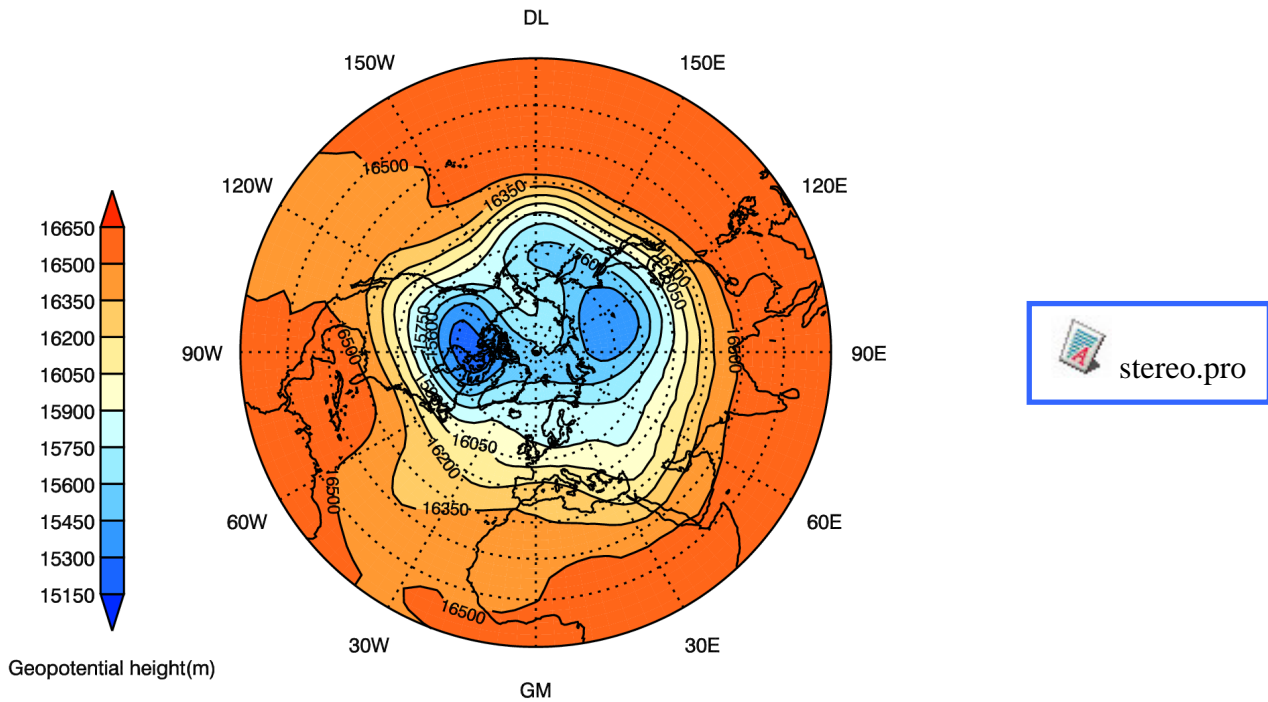
Colour filled contours as achieved in the same manner as grey fill in the above example but using more levels.

```
n=15
levs=INDGEN(n)*5-35
levs(0)=-100 ; Extend minimum
levs(n-1)=100 ; and maximum as we are plotting an extended colourbar
labs=STRTRIM(STRING(levs),2)
CONTOUR, temp, longs, lats, LEVELS=levs, XSTYLE=5, YSTYLE=5,$
      /CELL_FILL, C_COLORS=INDGEN(n)+2, /OVERPLOT
CONTOUR, temp, longs, lats, LEVELS=levs, XSTYLE=5, YSTYLE=5,$
      C_ANNOTATION=labs, /OVERPLOT
```

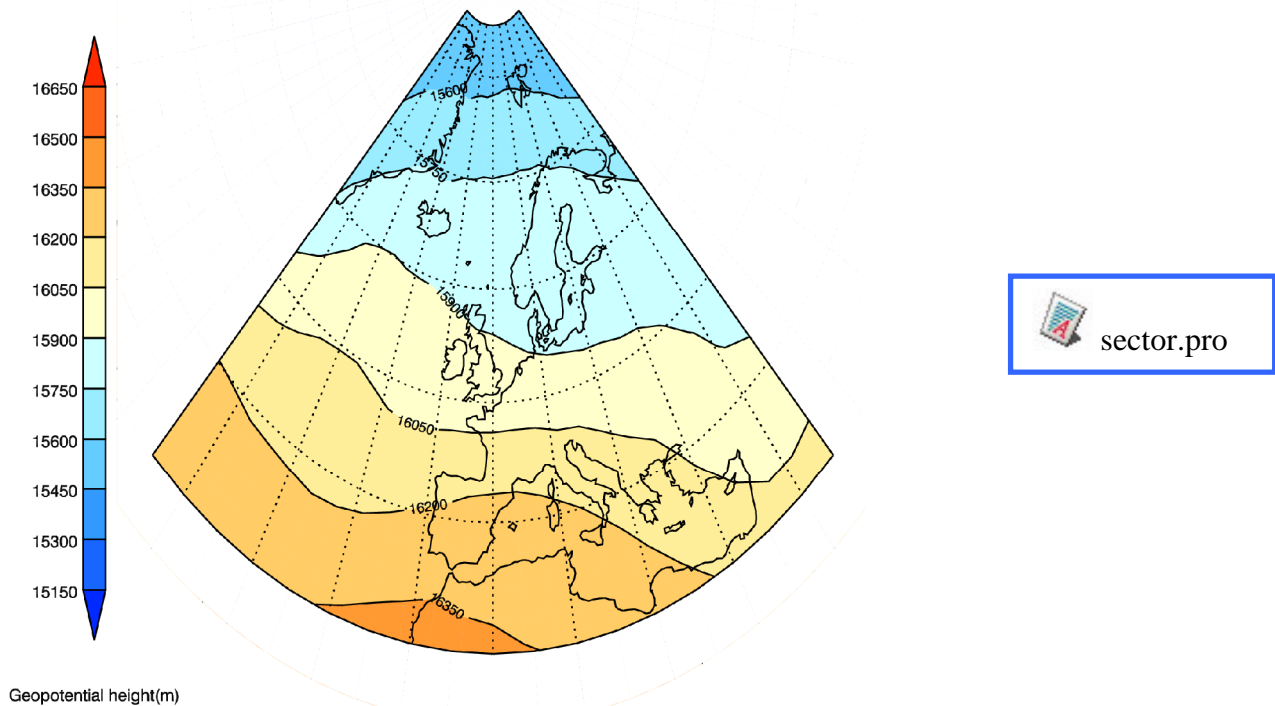
Using /CELL_FILL gives the correct filled colours.
/FILL can give strange results and should be avoided.



If you are interested in the polar regions then the MAP_SET has the /STEREOGRAPHIC and /ORTHOGRAPHIC keywords. The stereographic projection has equally spaced latitudes whereas orthographic emphasises the pole more. In the plot below we use the stereographic projection. Labelling the axes is a little different here. To do this we swap into device coordinates and use XYOUTS to draw the labels in a circle, as can be seen in the code stereo.pro.

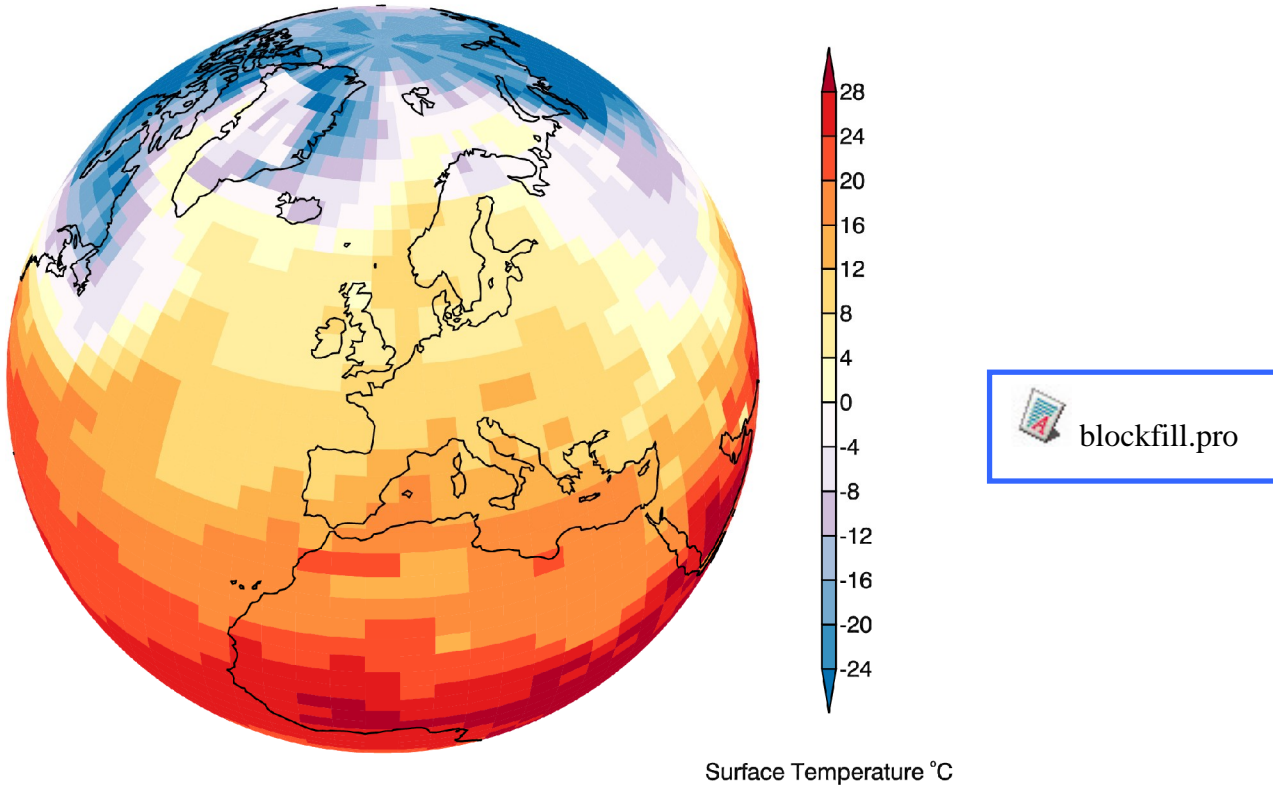


A variation of the stereographic plot is the sector plot. Here we select an area of the plot with the **LIMIT=[latmin, longmin, latmax, longmax]** keyword to the **MAP_SET** routine and then blank off the surrounding areas that we don't wish to see with calls to **POLYFILL**. Choosing the centre of the plot is again similar to the cylindrical centre issue. Here the latitude centre is always 90°N and the longitude centre in this case is 0.0° . See the code in `sector.pro` for the details.



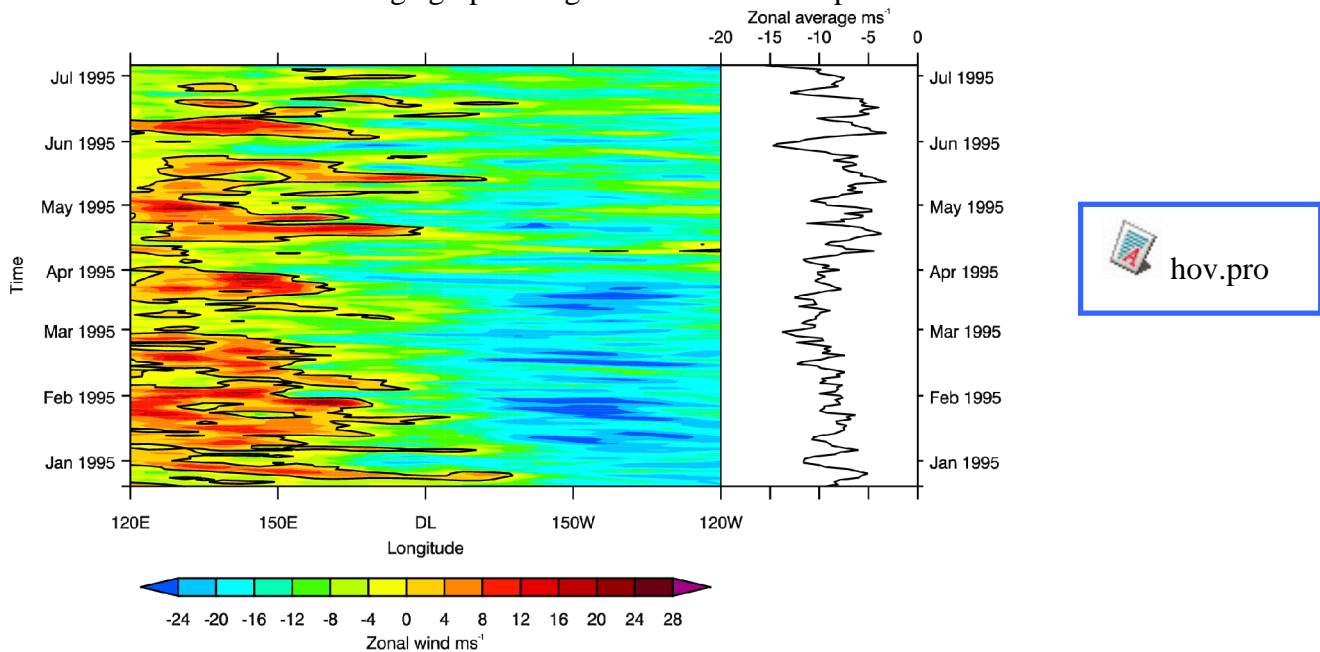
22. Block fill contour plots

This type of plot is quite easily made using the IDL `POLYFILL` routine. The basic idea is to circle over each latitude and longitude position and fill that rectangle in with the appropriate colour.



23. Hovmuller plots

Hovmuller plots are the temporal variation of a field with longitude or a time-longitude plot. In this example we read in 200 days of global model zonal wind data, reduce this down to a mean between the latitudes 2.5°N to 2.5°S and plot the data between 120°E and 120°W. Here, we use the `!P.POSITION` command to add a zonal average graph alongside the Hovmuller plot.



24. Zonal plots

First we will plot a zonal mean zonal wind plot that is linear in pressure on the y-axis. We read in the zonal wind and pressure levels with:

```
fid=NCDF_OPEN('ukmo.nc')
varid=NCDF_VARID(fid,'u')
NCDF_VARGET, fid, varid, data
varid=NCDF_VARID(fid,'p')
NCDF_VARGET, fid, varid, plevs
NCDF_CLOSE, fid
```

```
u=TOTAL(data,1)/96.0 ; compute zonal mean
```

Pressure and height are interchangeable in the stratosphere using the relation:

$$p=p_s \exp^{(-H/H_s)}$$

p =pressure

p_s =pressure at the surface (1013hPa)

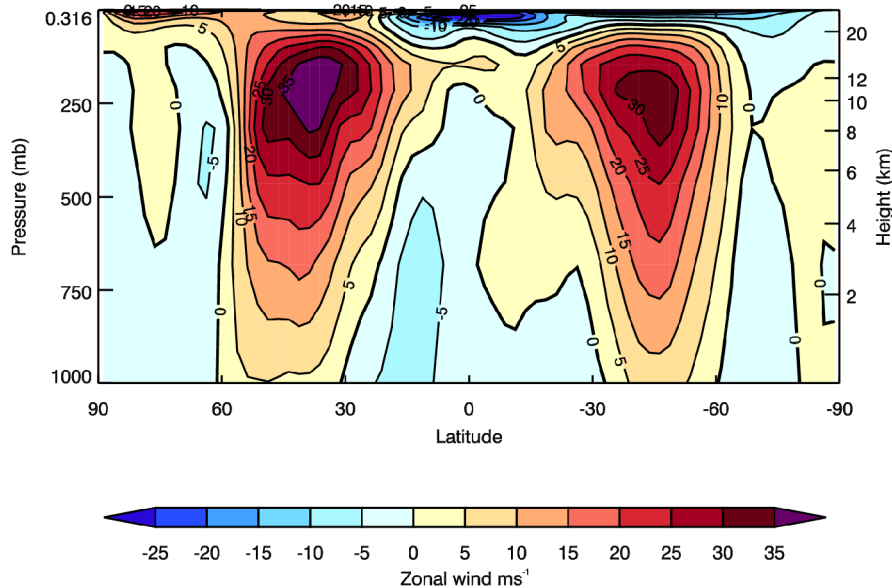
H =height in km

H_s =scale height which is 7.0 within the stratosphere.

The IDL code to convert pressure to height using the above relation is:

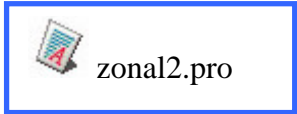
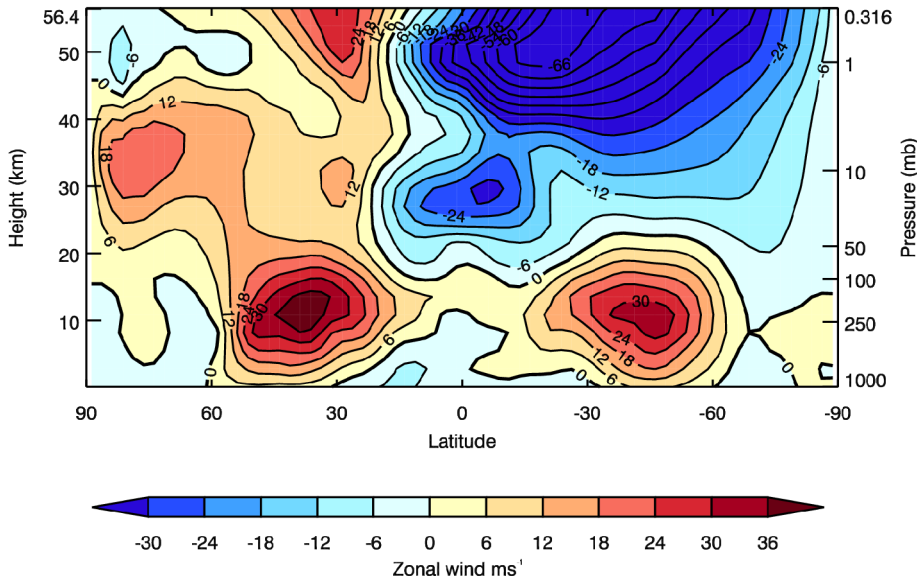
```
levskm=FLTARR(22)
FOR iz=0,21 DO BEGIN
  levskm(iz)=7.0*(ALOG(1013.)-ALOG(plevs(iz)))
  PRINT, 'levels ', levskm(iz), plevs(iz)
ENDFOR
```

We use this relation to make a height axis on the right hand side of the plot while plotting linear pressure on the left axis.

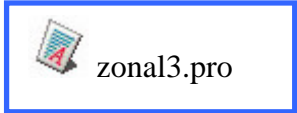


In the second plot we change the pressure levels into their corresponding height levels and plot this as the linear y axis. We could also do the same plot using a \log_{10} pressure conversion and plotting this as

the y-axis. In zonal3.pro we use the `/YLOG` keyword to contour and set `YRANGE=[1000.0, 0.316]` to get the atmosphere the correct way up. The picture is identical to that produced by zonal2.pro



zonal2.pro



zonal3.pro

25. Vector plots

25.1 Regularly spaced vector plots

The **VECTOR** routine included with this guide is a new vector plotting routine and has the following calling sequence:

VECTOR, u, v, x, y

u and **v** are the wind components (both 2d arrays)

x and **y** are the positions of the wind components (1d arrays and can be a single point)

Optional calling parameters are:

LENGTH=n is the length of the vector

ALIGN=0.0 – vector alignment, 0.0=tail, 0.5=centre and 1.0=head.

STRIDE=n - only plot every nth vector from **u** and **v**.

TYPE – vector head type, 0-4 as follows \longrightarrow \longrightarrow \longrightarrow \longrightarrow \longrightarrow

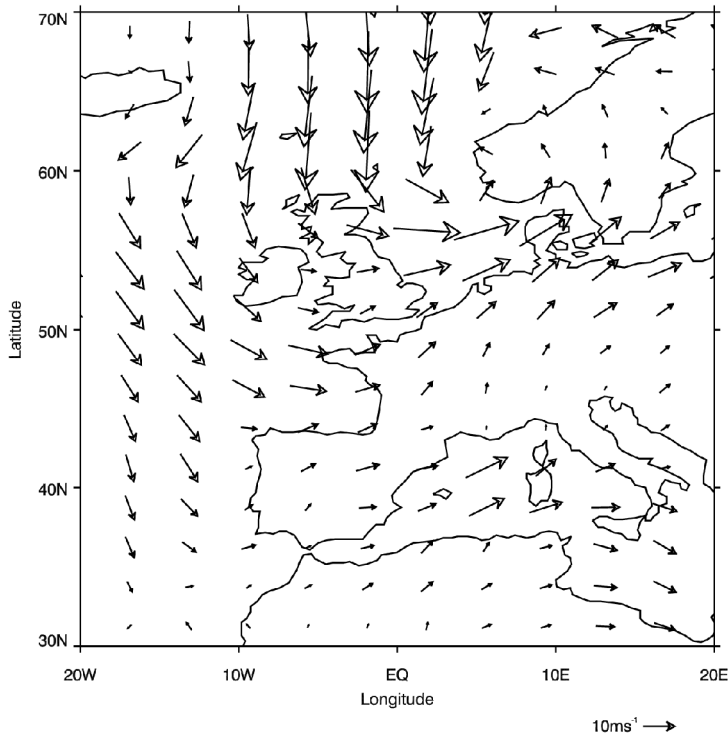
REF_MAG – magnitude of the reference vector.

REF_POS=[xpos, ypos] – Plot a reference vector at normal coordinates xpos,ypos.


REF_TEXT='mytext' – Label reference vector with the text 'mytext'.


ANGLE – angle of the head, default=22.5°

HEAD_LEN – length of the head relative to shaft, default=0.3.



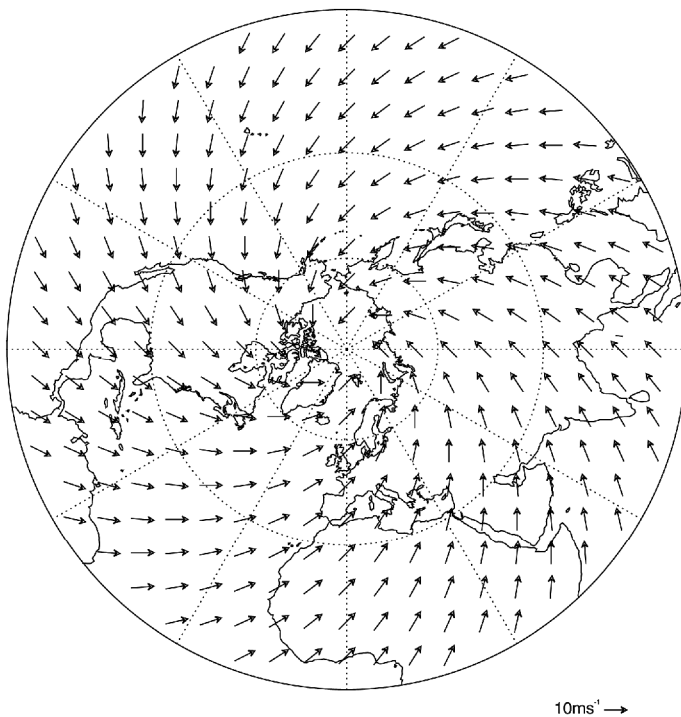
 winds.pro


 vector.pro

 plot_vector.pro

25.2 Polar vector plots

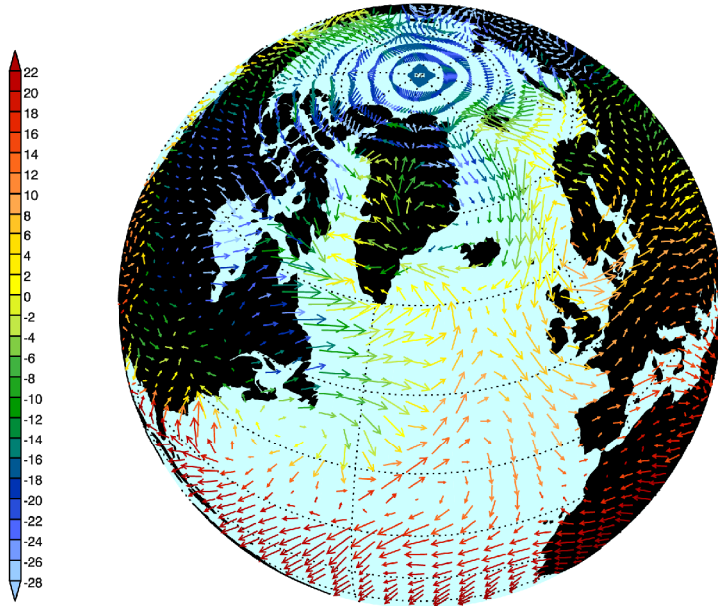
Viewing polar vector plots when using regular latitude-longitude grids give concentric circles of vectors, which are difficult to read properly. To get a more realistic view of the wind field some interpolation is needed. In this program bilinear interpolation is used to get an equally spaced grid in device space. In this example we use the standard Met-Office climate grid and set the u and v components to give a wind speed of 10ms^{-1} .



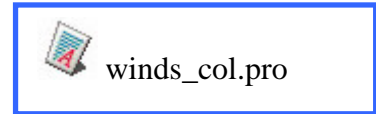
 polar_winds.pro

25.3 Coloured vectors

We can also colour each of the vectors with the magnitude of a field at that point – temperature in the case below.



Surface Temperature °C



26. Interpolation

A simple regridding technique is to use bilinear interpolation. This technique is described in more detail on Wikipedia at http://en.wikipedia.org/wiki/Bilinear_interpolation. A longitude-latitude regridding program is provided with this guide and is called regrid.pro.

The calling parameters for this program are:

REGRID, fieldin, longs, lats, fieldout, longs2, lats2

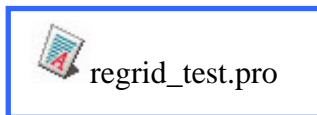
fieldin is the original grid on the grid spacing **longs, lats**.

fieldout is the output grid on the grid spacing **longs2, lats2**.

longs2 and **lats2** can be single or multi-value so you can get the value(s) at a point or on a new grid.

Examples of how to use the program are given in regrid_test.pro.

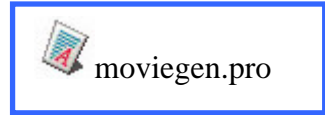
This program is used in polar_winds.pro and moll.pro.



27. Animation

Animation can be a difficult subject to find a way through. Generally I'd recommend starting by making postscript files and then converting these with the Unix command **convert** to PNG files.

An IDL program is provided with this guide to make movie png files from a file containing daily geopotential height data at 100mb. This generates the PNG files in the guide subdirectory called geomovie.



There are three common ways you might like to show your animation.

27.1 Using an animated GIF file

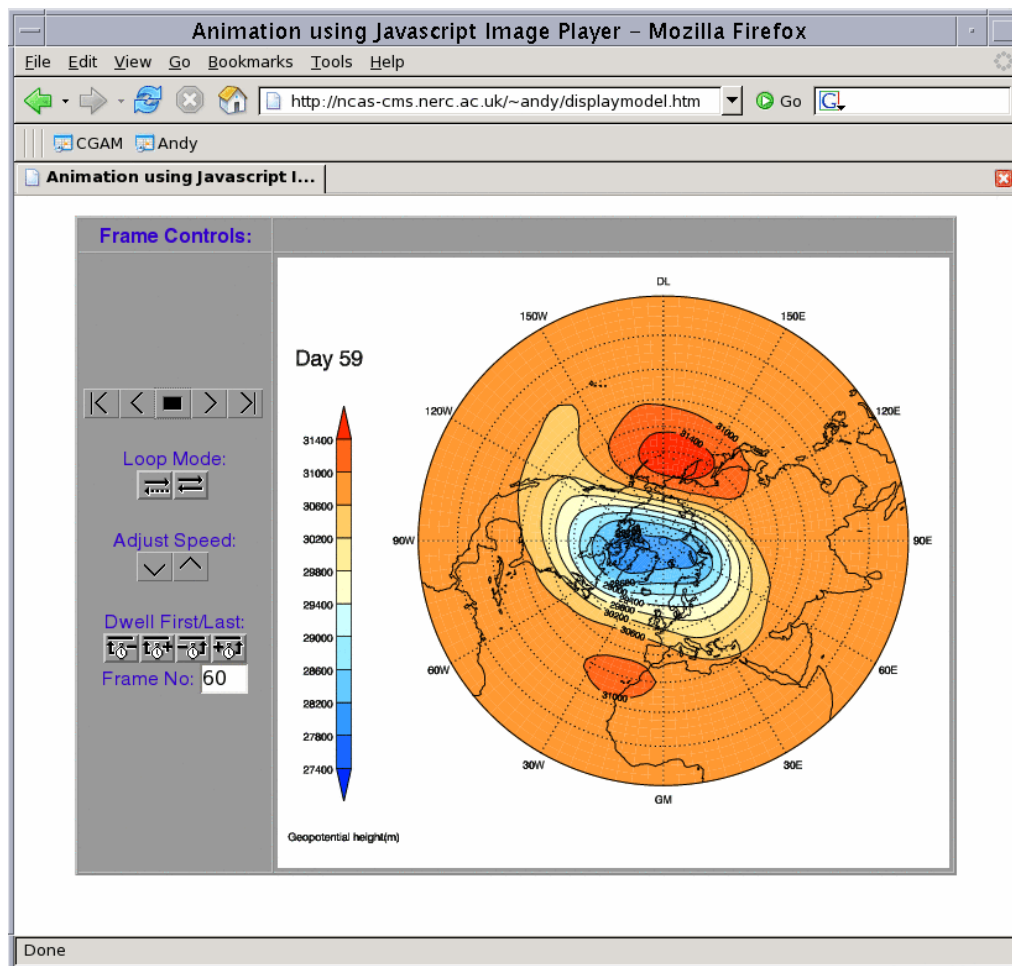
The PNG movie files can be converted to an animated GIF file with the **convert** command.
convert *.png movie.gif

The animated GIF file can then be played in **xanim** on Unix systems or viewed with most web browsers.

27.2 A web page JavaScript movie viewer

Writing your own JavaScript viewer is a time consuming process and there are many movie viewers already available on the web. The code for this JavaScript viewer originally came from <http://www.rap.ucar.edu/weather/model/displayMod.php>.

The movie viewer for the 100mb geopotential height PNG files generated above is available at <http://ncas-cms.nerc.ac.uk/~andy/displaymodel.htm>.



The html code is available in the guide as displaymodel.htm. Looking at the top of the code you will see that the only lines that need modifying are the list of images that you are viewing as a movie. A small IDL program is provided to generate the lines needed in this section.



The final piece needed is the image files for the movie control buttons that are available in the directory movie_buttons.

27.3 Making an AVI file

AVI files can be played within PowerPoint or externally using RealPlayer and so are convenient when you are giving talks. A simple and effective AVI maker is makeavi and is available from Sourceforge at <http://sourceforge.net/projects/makeavi>. This is a PC program so you will need a PC or laptop to make your AVI file. Unix AVI makers are in very short supply and I haven't found a good one yet.

Remember to copy both the PowerPoint and AVI files to your memory stick as the movie isn't embedded within the PowerPoint file.

28. Codelets

Codelets are re-usable snippets of code to make programming quicker. An example is cylin4.pro, which produces the same output as cylin3.pro previously shown.

```
PRO cylin4
;show the use of code snippets

outfile='temp.ps'
datafile='ukmo.nc'
level='1000.00'
field='temp'
offset=273.15 ;convert to Celcius
levels=INDGEN(16)*5-35
legend_title='Surface Temperature !Eo!NC'

@read_data.inc
@open_ps.inc
@plot_latlon_std.inc
@close_ps.inc

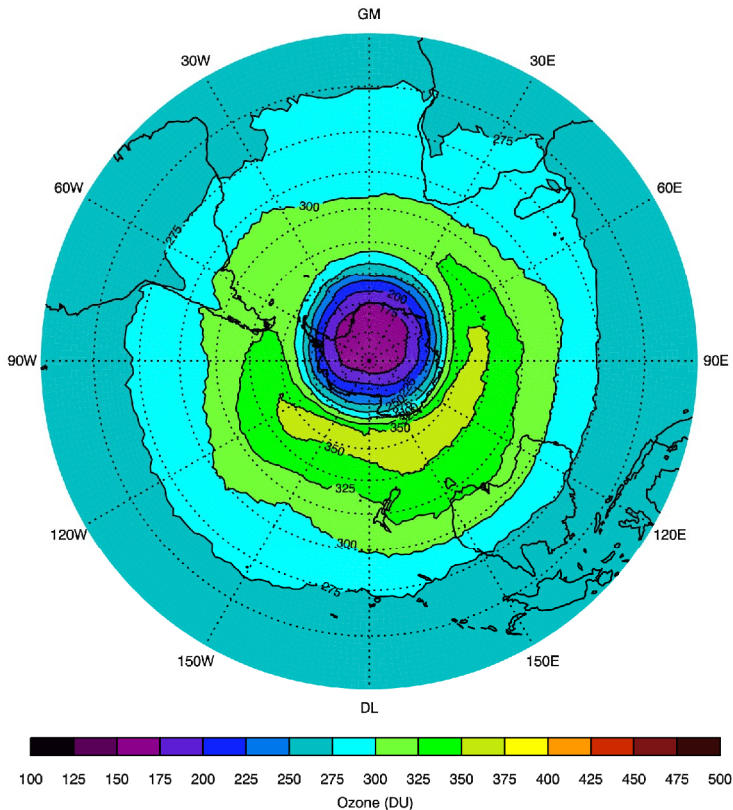
END
```



Using codelets is a very powerful way to easily develop your own code library.

29. Running IDL in Unix batch mode

IDL can easily be run in Unix batch mode so that scripts can run overnight or to plot data when it becomes available on a remote server. A simple script to download and plot some ozone data is available in the program `get_data`.



You can set a script such as this to run each evening using a crontab entry - type **man crontab** for more details. You can also use `at` to run the script just once at a time when the computers are less used – type **man at** for more details.

30. Calling external objects

IDL allows the integration of externally compiled code into your IDL programs. Here a simple example is given of an IDL program calling a Fortran compiled shared object which multiplies two numbers together. The result is then passed back to IDL and the result is printed on the screen.

Use the following to setup the Fortran compiler and make the shared object.

```
setup studio11
f90 -G -pic -c ext.f
f90 -G ext.o -o ext.so
```



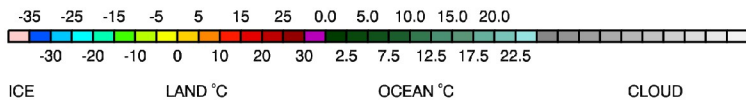
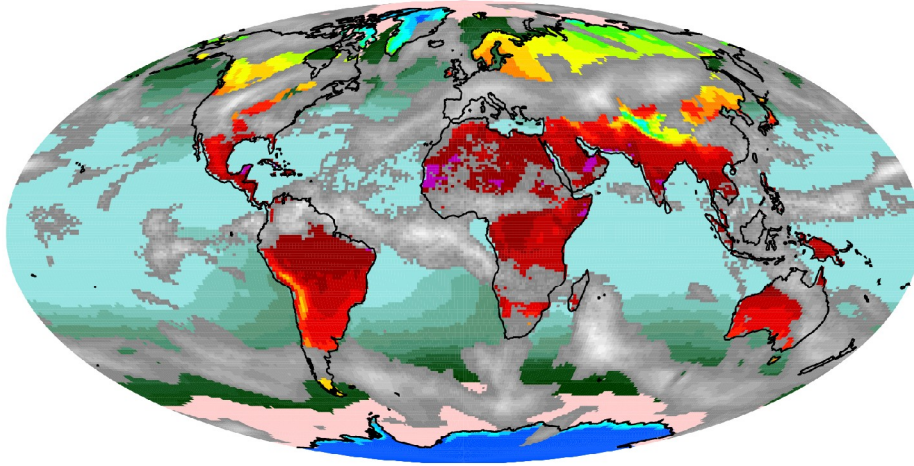
Care is needed to match the data sizes in IDL and Fortran. Using `f95` produces errors so using `f90` is recommended until probably the next release of the Fortran compiler.

This is a very simple example to help you understand the basics of calling external objects. Detailed notes are available in the IDL External Development Guide manual.

31. A Mollweide plot

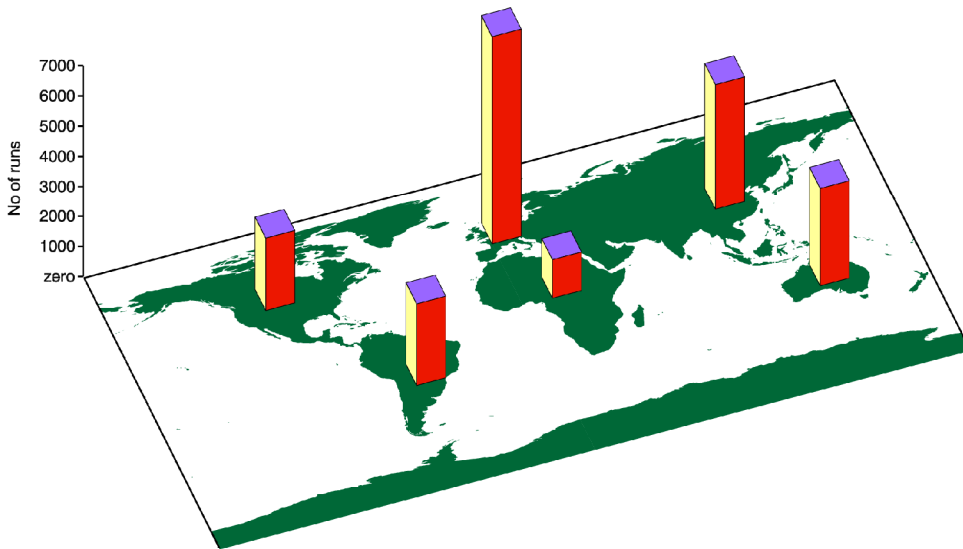
Here we read in the temperature, ice, cloud and land-sea mask and make a plot like those found on the University of Wisconsin-Madison composite images website:

<http://www.ssec.wisc.edu/data/composites.html>. Extensive use of **WHERE** is used in this routine.



32. Map distributions

This plot shows the use of **POLYFILL** again to create a distribution map.



33. Writing style

A simple program style is advisable for clarity and ease of later modification:

- 1) Use uppercase for IDL commands and reserved keywords.
- 2) Include a comment block at the top of your program describing the program function. Comment sections to allow easy understanding of the major program functions.
- 3) Add spacing to allow easy reading and indentation so loops are easily picked out.

Appendix A of Kenneth Bowman's 'An Introduction to programming with IDL' is a good style guide and well worth a read.

34. Irregular grids

Irregular grids can be contoured by first creating the triangulation data and then passing this to the **CONTOUR** routine.

```
TRIANGULATE, x, y, tri  
CONTOUR, z, x, y, TRIANGULATION=tri
```

In this example a tri-polar grid is read in and a point is plotted at each data point. The sea surface temperature is then plotted using the **TRIANGULATION** keyword to **CONTOUR**.

