Notes of IDL Programming V1.0

Faxiang Zheng

July 13, 2016

1 Before you start

Assuming that you have IDL (version greater than or equal to 7.1) installed on your Linux, MacOS or FreeBSD, there's some additional configuration that I recommend.

1.1 install Coyote and Catalyst Library

Open your terminal (with bash or zsh), then copy and paste following commands

```
mkdir -p ~/lib/idl/idl-coyote   # ~/lib/idl can be anywhere else
cd ~/lib/idl/idl-coyote
git init
git remote add origin https://github.com/davidwfanning/idl-coyote.git
echo "coyote" >> .git/info/sparse-checkout
echo "catalyst" >> .git/info/sparse-checkout
git config core.sparsecheckout true
git pull origin master
```

1.2 install AstroLib

```
cd ~/lib/idl
git clone https://github.com/wlandsman/IDLAstro.git astrolib
```

1.3 install Motley Library

```
cd ~/Downloads
wget http://www.idlcoyote.com/hadfield/idl/motley.tar.gz
mkdir ~/lib/idl/motley
tar -C ~/lib/idl/motley -zxvf motley.tar.gz
```

1.4 install Gumley's IDL routines

```
cd ~/Downloads
wget https://svn.ssec.wisc.edu/repos/geoffc/IDL/Gumley/PIP_programs.tar.Z
mkdir ~/lib/idl/gumley
tar -C ~/lib/idl/gumley -zxvf ~/Downloads/PIP_programs.tar.Z
```

1.5 install Slug's IDL routines

```
cd ~/Downloads
wget http://slugidl.pbworks.com/f/slug_idl.tar
mkdir ~/lib/idl/slug
tar -C ~/lib/idl/slug -xvf slug_idl.tar
```

1.6 install TeXtoIDL

```
cd ~/Downloads
wget http://physics.mnstate.edu/craig/textoidl/textoidl-2-1-2.tar
tar -C ~/lib/idl -xvf textoidl-2-1-2.tar
```

1.7 setting PATH

Add the following to your ~/.profile (bash user) or ~/.zshenv (zsh user)

```
# IDL setting
export IDL_LIB_DIR=$HOME/lib/idl
export IDL_PATH='<IDL_DEFAULT>'
export IDL_PATH=$IDL_PATH:+'$IDL_LIB_DIR/idl-coyote/coyote'
export IDL_PATH=$IDL_PATH:+'$IDL_LIB_DIR/idl-coyote/catalyst'
export IDL_PATH=$IDL_PATH:+'$IDL_LIB_DIR/astrolib/pro'
export IDL_PATH=$IDL_PATH:+'$IDL_LIB_DIR/motley'
export IDL_PATH=$IDL_PATH:+'$IDL_LIB_DIR/gumley'
export IDL_PATH=$IDL_PATH:+'$IDL_LIB_DIR/slug'
export IDL_PATH=$IDL_PATH:+'$IDL_LIB_DIR/textoidl'
```

1.8 startup file

You can, of course, name the start-up file anything you like and locate it anywhere in your directory structure. For example, add the following to your ~/.profile (bash user) or ~/.zshenv (zsh user)

```
export IDL_STARTUP=$HOME/.idlstartup.pro
```

I put something like

```
print, 'Hello, ' + getenv('USER') + '! ' + 'IDL is ready :-)'
```

in my startup file.

1.9 editor

Emacs, with its embedded *IDLWave-mode* and *IDLWave-shell*, is a little slice of heaven for dealing with IDL source file and command line. You may have a look at my Emacs configuration.

If your IDL version is 7.1, you might have to do something to fix a bug.

- open your terminal and type

```
sudo mkdir /usr/local/itt/idl71/help/online_help
sudo ln -s /usr/local/itt/idl71/help/idl_catalog.xml \
     /usr/local/itt/idl71/help/online_help/
sudo ln -s /usr/local/itt/idl71/help/idlithelp.xml \
     /usr/local/itt/idl71/help/online_help/
sudo ln -s /usr/local/itt/idl71/help/template_help.xml \
     /usr/local/itt/idl71/help/online_help/
```

- open your Emacs and *M-x idl-rescan-asynchronously*

Happy hacking!

1.10 resources

- http://www.idlcoyote.com/
- http://www.idlcoyote.com/gallery/index.html
- http://www.harrisgeospatial.com/docs/routines-1.html
- http://www.harrisgeospatial.com/docs/PDF_Guides.html
- http://idlastro.gsfc.nasa.gov/
- http://astroplotlib.stsci.edu/
- http://slugidl.pbworks.com/w/page/28913791/Plotting%2520Examples
- http://www.astrobetter.com/wiki/tiki-index.php?page=Python+Switchers+Guide
- url: https://pan.baidu.com/s/1eSlZ7v0 passwd: t4x3
- url: https://pan.baidu.com/s/1mhSk4YW passwd: h83i

2 Call system commands

2.1 spawn

```
spawn, 'echo hello IDL!'
spawn, 'ls ~/',result
print, result
```

2.2 $

```
$cd ~
$ls
```

3 Fundamentals of syntax

3.1 data types

3.1.1 get data type and size

1. *help*

   help procedure is convenient to use in interactive mode

   ```
   a = dist(256)
   help, a
   ```

2. *size*

```
a = [[0, 1, 1], [2, 2, 1]]
print, 'Number of dimensions:', size(a, /n_dimensions)
print, 'Size of each dimension:', size(a, /dimensions)
print, 'Number of elements:', size(a, /n_elements)
print, 'Data type name:', size(a, /tname)
print, 'Data type index:', size(a, /type)
```

If no keywords are set, *SIZE* returns a vector of integer type. The first element is equal to the number of dimensions of Expression. This value is zero if Expression is scalar or undefined. The next elements contain the size of each dimension, one element per dimension (none if Expression is scalar or undefined). After the dimension sizes, the last two elements contain the type code (zero if undefined) and the number of elements in Expression, respectively.

```
data = dist(32)
result = size(data)
ndims = result[0]
if (ndims gt 0) then dims = result[1:dims] else dims = -1L
type = result[ndims + 1]
nele = result[ndims + 2]
```

The *n_elements* is commonly used within IDL programs to test whether a variable (array or otherwise) is defined.

3.1.2 type codes and names

Table 1: IDL Type Code and Names

Data type	Type code	Type name
Undefined	0	'UNDEFINED'
Byte	1	'BYTE'
Integer	2	'INT'
Long Integer	3	'LONG'
Float	4	'FLOAT'
Double precision	5	'DOUBLE'
Complex float	6	'COMPLEX'
String	7	'STRING'
Structure	8	'STRUCT'
Complex double precision	9	'DCOMPLEX'
Pointer	10	'POINTER'
Object reference	11	'OBJECT'
Unsigned integer	12	'UINT'
Unsigned long integer	13	'ULONG'
64-bit integer	14	'LONG64'
Unsigned 64-bit integer	15	'ULONG64'

3.1.3 data type and generate function

Table 2: Data types

Type	Byte number	Create	Function
byte	1	Var=0B	thisVar=Byte(variable)
int16	2	Var=0	thisVar=Fix(variable)
int32	4	Var=0L	thisVar=Long(variable)
int64	8	Var=0LL	thisVar=Long64(variable)
uint16	2	Var=0U	thisVar=UInt(variable)
uint32	4	Var=0UL	thisVar=ULong(variable)
uint64	8	Var=0ULL	thisVar=Ulong64(variable)
float	4	Var=0.0	thisVar=Float(variable)
double	8	Var=0.0D	thisVar=Double(variable)
complex	8	Var=Complex(0.0,0.0)	thisVar=Complex(variable)
double complex	16	Var=Dcomplex(0.0D,0.0D)	thisVar=DComplex(variable)
string	0-32767	Var=”,Var=””	thisVar=String(variable)
pointer	4	Var=Ptr_New()	None
object	4	Var=Obj_New()	None

3.1.4 data type and array/index generation

Table 3: Array/Index generation

Type	Array	Index
byte	BytArr	BIndGen
int16	IntArr	IndGen
int32	LonArr	LIndGen
int64	Lon64Arr	L64IndGen
uint16	UIntArr	UIndGen
uint32	ULon64Arr	ULIndGen
uint64	ULon64Arr	UL64IndGen
float	FltArr	FIndGen
double	DblArr	DIndGen
complex	ComplexArr	CIndGen
double complex	DCompexArr	DCIndGen
string	StrArr	SIndGen
pointer	PtrArr	None
object	ObjArr	None

3.1.5 vector/array slicing

```
vector1 = IndGen(4)
vector2 = vector1[0:2]
print, vector2
```

different from python

```
import numpy as np
vector1 = np.arange(4)
```

```
vector2 = vector1[0:2]
print("vector2 = ", vector2)
```

array slicing is just the same you can easily get a row of the array or a subarray through it's indices

3.1.6 ceil, floor, round and fix

1. ceil

   The *CEIL* function returns the closest integer greater than or equal to its argument.

```
print, ceil(1.3)
print, ceil(-1.1)
print, ceil(3000000000.1D, /L64)
```

2. floor

   The *FLOOR* function returns the closest integer less than or equal to its argument.

```
print, floor(1.4)
print, floor(-1.2)
print, floor(3000000000.1D, /L64)
```

3. round

   The *ROUND* function rounds the argument to its closest integer.

```
print, round(5.5)
print, round(1.4)
print, round(-1.5)
print, round(3000000000.1D, /L64)
```

4. fix The *fix* function returns the closest integer whose absolute value is less than or equal to its argument's.

```
print, fix(1.5)
print, fix(-1.5)
```

3.2 operators

3.2.1 operators and precedence

Table 4: IDL operators

Operator	Meaning	Precedence level (1 is highest)
()	parentheses	1
*	pointer dereference	2
^	exponentiation	2
*	scalar multiplication	3
#	array multiplication	3
##	matrix multiplication	3
mod	modulus	3
+	addition	4
-	subtraction and negation	4
<	minimum	4
>	maximum	4
not	boolean negation	4
eq	equal to	5
ne	not equal to	5
le	less than or equal to	5
lt	less than	5
ge	greater then or equal to	5
gt	greater than	5
and	boolean AND	6
or	boolean OR	6
xor	Boolean exclusive OR	6
?:	ternary operator	7
=	assignment	8

3.2.2 minimum and maximum operators

```
a = [2, 4, 6, 8]
b = [3, 4, 5, 6]
print, a < b
print, 3 > a
print, -3 < (-indgen(9))
```

3.2.3 array and matrix multiplication operators

```
a = indgen(3, 2) + 1
b = indgen(2, 3) - 1
print, a
print, b
c = a # b
print, c
print, transpose(c)
d = a ## b
print, d
```

3.2.4 relational and boolean operators

```
a = 10.0
b = 20.0
help, a gt b
help, a lt b
```

Table 5: True/false definitions

Data Type	True	False
Integer	Odd nonzero values	Zero and even values
Floating point	Nonzero values	Zero
String	Any string that is not null	Null string

```
a = [2, 4, 6, 8]
b = [3, 4, 5, 6]
print, a gt b
```

```
arr = byte([0, 1, 2, 3, 4])
print, arr
print, not arr
arr = long([0, 1, 2, 3, 4])
print, not arr
```

```
a = [0, 0, 1, 1, 2]
b = [0, 1, 1, 2, 2]
print, a or b
print, float(a) or float(b)
print, float(b) or float(a)
```

```
print, 4, format = '(b08)'
print, 5, format = '(b08)'
print, 4 xor 5, format = '(b08)'
```

3.2.5 ternary operator

```
x = 10
y = 5
z = (x lt y) ? x : y
print, z
```

3.3 arrays

3.3.1 array generation

1. general method

```
array = [[1, 2, 3], [4, 5, 6]]
```

the same as

```
vector = IndGen(6) + 1
array = Reform(vector, 3, 2)
```

2. *make_array*

```
array1 = make_array(3, 3, /integer, /index)
array2 = make_array(3, 3, /integer, value=8)
```

3.3.2 properties

Table 6: Functions for array properties

Function name	Return
n_elements()	number of elements
size()	array size and type information
min()	minimum array value
max()	maximum array value
mean()	mean of array values
variance()	variance of array values
stddev()	standard deviation of array values
moment()	mean, variance, skew, kurtosis, standard deviation, mean absolute deviation
total()	sum of array values

the *min* function can also return the maximum value via the *max* keyword

```
arr = dist(32)
print, min(arr), max(arr)
minval = min(arr, max=maxval)
print, minval, maxval
maxval = max(arr, index)
print, index
```

The *moment* function returns the mean, variance, skew, and kurtosis of an array

```
result = moment(arr)
print, result
```

If the optional *sdev* and *mdev* keyword are supplied, *moment* also returns the standard deviation and mean absolute deviation

```
result = moment(arr, sdev=sdev, mdev=mdev)
print, sdev, mdev
```

The *total* function returns the total of an array

```
arr = indgen(3, 3)
print, arr
print, total(arr)
print, total(arr, 1)
print, total(arr, 2)
print, total(arr, /cumulative)
```

3.3.3 location values within an array

```
arr = indgen(9) * 10
print, arr
index = where(arr gt 35)
print, arr[index]

result = arr gt 35
```

```
print, result
index = where(result)
print, index
print, arr[index]

index1 = where((arr gt 35) and ((arr mod 4) eq 0))
print, arr[index1]
```

You should always account for the possibility that where will not find any non zero values in the input array or expression (i.e., no values meet the prescribed conditions). In this case, where returns a scalar long equal to -1

```
index2 = where(arr lt -10000.0, count)
help, index2
print, count
index3 = where(arr gt 2.0, count)
print, count
```

3.3.4 reform an array

```
arr = indgen(6, 2)
print, arr
arr = reform(arr, 3, 4)
print, arr
col = arr[0, *]
col = reform(col)
help, col
```

To conserve the memory, the *overwrite* keyword may be used with *reform* to change the dimension information of an array without making a copy of the array

```
arr = dist(32)
help, arr
arr = reform(arr, 32L * 32L, /overwrite)
help, arr
```

3.3.5 reversing array elements

```
arr = indgen(5)
print, arr
print, reverse(arr)

arr = indgen(3, 3)
print, arr
print, reverse(arr)
print, reverse(arr, 1)
print, reverse(arr, 2)
```

3.3.6 rotating arrays

```
arr = indgen(4, 3)
print, arr
print, rotate(arr, 1)
print, rotate(arr, 2)
print, rotate(arr, 3)
```

The direction flags 0, 1, 2, 3 specify rotations of 0, 90, 180 and 270 degrees, respectively.

3.3.7 transposing arrays

```
arr = indgen(4, 3)
print, arr
help, transpose(arr)
print, transpose(arr)

arr = indgen(2, 4, 6)
help, transpose(arr)
help, transpose(arr, [1, 2, 0])
```

3.3.8 shifting arrays

```
arr = indgen(5)
print, shift(arr, 1)
print, shift(arr, -2)

arr = indgen(3, 3)
print, arr
print, shift(arr, 0, 1)
```

3.3.9 sorting arrays

```
arr = [5, 3, 9, 4, 0, 2]
index = sort(arr)
print, index
print, arr[index]

print, sort([1, 1, 1, 1, 1, 1])
print, sort([0, 0, 1, 1, 2, 2])
```

3.3.10 finding unique array values

```
arr = [20, 30, 20, 40, 40, 30, 50, 60, 10]
arr = arr[sort(arr)]
arr = arr[uniq(arr)]
print, arr
```

If you want to extract unique values form an array while preserving the order of the elements.

```
arr = [20, 30, 20, 40, 40, 30, 50, 60, 10]
sort_index = sort(arr)
uniq_index = uniq(arr[sort_index])
index = sort_index[uniq_index]
index = index[sort(index)]
print, arr[index]
```

3.3.11 array resizing

1. functions

Table 7: Functions for array resizing

Function Name	Purpose	Algorithm
rebin()	Resize an n-dimensional array by an integer multiple or factor	Default is linear interpolation for enlarging and neighborhood averaging for shrinking. Nearest-neighbor sampling is optional.
congrid()	Resize a one-, two-, or three-dimensional array to an arbitrary size (user-friendly version)	Default is nearest-neighbor sampling. Linear or cubic convolution interpolation is optional.
interpolate()	Resize a one-, two-, or three-dimensional array to an arbitrary size (generalized version).	Default is linear interpolation. Cubic convolution interpolation is optional.

2. resizing by an integer factor

```
arr = [20, 40, 60]
print, rebin(arr, 9)
print, rebin(arr, 9, /sample)
```

Setting the *sample* keyword (sample or sample = 1) causes nearest neighbor sampling to be used for enlarging or shrinking. If the *sample* keyword is not set (sample absent or sample = 0), then linear interpolation is used for enlarging, yielding quite different results.

When shrinking an array, *rebin* uses neighborhood averaging.

```
arr = indgen(9)
print, arr
print, rebin(arr, 3)
```

The *rebin* function is useful when you wish to create multidimensional grid arrays.

```
v = indgen(5)
print, v
print, rebin(v, 5, 3, /sample)
print, rebin(reform(v, 1, 5), 3, 5, /sample)
print, rebin(reform(v, 1, 5), 3, 5, 3, /sample)
```

3. resizing to arbitrary size

The *congrid* function enlarges or shrinking a one-, two-, or three- dimensional array to an arbitrary size. The default is to use nearest neighbor sampling.

```
arr = [20, 40, 60]
print, congrid(arr, 4)
print, congrid(arr, 5)
print, congrid(arr, 6)
print, congrid(arr, 7)
print, congrid(arr, 8)
print, congrid(arr, 9)

ix = 3
nx = 9
```

```
index = (float(ix) / float(nx)) * findgen(nx)
print, index, format='(9f6.2)'
index = round(index)
print, index, format='(9i6)'
print, arr[index], format='(9i6)'
```

If the optional *minus_one* keyword is set, then *congrid* uses a slightly different algorithm to compute the resampled index array. You may set *interp* keyword to use linear interpolation.

```
print, congrid(arr, 9, /interp)
print, congrid(arr, 9, /interp, /minus_one)
```

4. resizing to arbitrary size with customized interpolation

   The i nterpo1 ate function differs from congri d in that with interpolate, you specify the exact locations for which you desire interpolated values

```
arr = [10.0, 20.0, 30.0]
loc = [0.0, 0.5, 1.0, 1.5, 2.0]
print, interpolate(arr, loc)
```

To resize a two-dimensional array (such as an image) with *interpolate*, you must choose an algorithm to compute the locations for the interpolated array values. The *grid* keyword allows the location arrays to have a different size in each output array dimension.

```
a = dist(32, 32)
nx = 500
ny = 500
dims = size(a, /dimensions)
xloc = (findgen(nx) + 0.5) * (dims[0] / float(nx)) - 0.5
yloc = (findgen(ny) + 0.5) * (dims[0] / float(ny)) - 0.5
b = interpolate(a, xloc, yloc, /grid)
help, b
tvscl, b
c = interpolate(a, round(xloc), round(yloc), /grid)
tvscl, c
```

3.3.12 removing rows or columns

```
arr = findgen(10, 1000)
delrow = [16, 200, 545, 762, 998]
dims = size(arr, /dimensions)
nrows = dims[1]
help, nrows
index = replicate(1L, nrows)
index[delrow] = 0L
keeprow = where(index eq 1)
arr = arr[*, keeprow]
help, arr
```

3.4 structures

3.4.1 anonymous structures

```
image = {name:'Test Image', valid_range:[0.0, 100.0], $
         data:dist(256)}
help, image
help, image, /structure
help, image.(0)
```

```
image.data = image.data * (1.0 / max(image.data))
image.name = 'Normalized Test Image'
image.valid_range = [0.0, 1.0]
```

3.4.2 arrays of structures

```
images = replicate(image, 10)
help, images
help, images[0]
```

3.4.3 named structures

```
rec = {nav_record, time:0.0, lat:0.0, lon:0.0, $
       heading:0.0}
help, rec
a = {nav_record}
a.time = 12.25
b = rec
b.time = 13.50
data = replicate({nav_record}, 100)
help, data
c = [a, b]
help, c
```

3.4.4 working with structures

```
image = {name:'Test Image', valid_range:[0.0, 100.0], $
         data:dist(256)}
print, n_tags(image)
print, tag_names(image)
```

```
image = create_struct('data', dist(256), $
                      'valid_range', [0.0, 100.0])
image = create_struct(image, 'date', systime())
help, image.data
help, image.date
```

3.5 pointers

3.5.1 functions and procedures

Table 8: Functions and procedures for pointers

Name	Purpose
ptr_new()	return a new pointer
ptrarr()	return an array of new pointers
ptr_free	free an array of new pointers
ptr_valid()	check the validity of a pointer

3.5.2 creating pointers

```
p = ptr_new(1.0)
help, p
print, *p
arr = indgen(5)
ptr1 = ptr_new(arr)
help, *ptr1
```

```
ptr2 = ptr_new(/allocate_heap)
help, *ptr2
*ptr2 = dist(256)
help, *ptr2
```

to create a pointer array

```
ptr = ptrarr(10, /allocate_heap)
*ptr[0] = dist(256)
*ptr[1] = indgen(10)
help, *ptr[0:2]
```

3.5.3 freeing pointers

```
ptr = ptr_new(10.0)
ptr_free, ptr
```

3.5.4 checking pointer validity

```
ptr = ptr_new(10.0)
print, ptr_valid(ptr)
ptr_free, ptr
print, ptr_valid(ptr)
```

3.5.5 pointer dereferencing

```
ptr = ptr_new(indgen(4, 4))
print, *ptr
print, (*ptr)[0, *]
```

if a pointer is included in a structure, the dereference operator is placed immediately before he structure name

```
rec = {flag:1L, data_ptr:ptr_new(indgen(5))}
print, *rec.data_ptr
struct_ptr = ptr_new(rec)
print, *(*struct_ptr).data_ptr
```

3.5.6 avoiding pointer problems

To see a list of all variables in memory that are currently referenced by a pointer, or were previously referenced by a pointer

```
help, /heap_variables
```

To guard against memory leakage, you can test the proposed pointer variable to see if it is already a valid pointer

```
if (ptr_valid(ptr) eq 1) then ptr_free, ptr
ptr = ptr_new(10.0)
```

To perform garbage collection, call the *heap_gc* procedure with the *ptr* and *verbose* keyword set

```
ptr = ptr_new(dist(256))
ptr = ptr_new(indgen(10))
ptr = ptr_new(2.0)
heap_gc, /ptr, /verbose
```

4 Basic I/O

4.1 standard input and output

4.1.1 procedures and functions

Table 9: procedures and functions for standard I/O

Name	Purpose
print	write formatted data to standard output
read	read formatted data from standard input
reads	read formatted data from a string
string()	write formatted data to a string

4.1.2 format

`https://www.exelisvis.com/docs/format_codes.html`

The syntax of an IDL format code is:

```
[n]FC[+][-][width]
```

where:

Table 10: format code syntax

Code	Description
n	the number of times the format code should be processed
FC	the format code
+	a + prefix before positive numbers that only valid for numeric format
-	left-justified
width	width specification

Table 11: format codes

Format Code	Description
a	character
:	terminates processing
$	suppresses newlines in output
F,D,E,G	floating-point
B,I,O,Z	integer (B:Binary,I:Decimal,O:Octal,Z:Hexadecimal)
Q	the number of characters
quoted string and H	output string directly
T	specifies the absolute position within a record
TL	moves the position with a record to the left
TR and X	move the position within a record to the right
C()	transfers calendar data
C printf-style	an alternative syntax for specifying the format of an output
/	newline

examples:

```
print, format='(a)', 'hello world'
print, format='(a7)', 'hello world'
print, format='(i12)', 300
print, format='(i12.8)', 300
print, format='(i08)', 300
print, format='(b)', 7
print, format='(o)', 7
print, format='(z)', 12
print, format='(3f12.3)', findgen(3) * 500
print, format='(f12.3, f10.2, f10.1)', findgen(3) * 500
print, format='(3f12.3)', findgen(9) * 500
print, format='(3(f12.3, :, ", "))', findgen(9) * 500
print, format='(3(e12.3, :, ", "))', findgen(9) * 500
print, format='(3(e+12.3, :, "$"))', findgen(9) * 500 - 1000
print, format='(g12.4)', 10
print, format='(g12.4)', 10000000
```

```
.run
pro format_test
  name = ''
  age = 0
  print, format='($, "Enter your name")'
  read, name
  print, format='("Enter your age")'
  read, age
  print, format='("You are ", i0, " years old, ", a0)', age, name
end

format_test
```

```
read, format='(q)', charnumber
print, charnumber
```

```
print, format='("Value: ", I0)', 23
print, format='(7HValue: , I0)', 23
```

```
print, format='("First", 20X, "Last", T10, "Middle")'
print, format='("First", 20X, "Last", TL15, "Middle")'
print, format='("First", TR15, "Last")'
```

```
print, systime()
print, systime(/utc)
print, systime(1)
print, systime(/julian)
curtime = systime(/julian)
print, format='(c())', curtime
print, format='(c(CYI, "-", CMOI, "-", CDI))', curtime
print, format='(c(CHI, ":", CMI, ":", CSI))', curtime
```

4.2 working with files

4.2.1 procedures and functions

Table 12: procedures and functions for working with files

Name	Purpose
openr	open an existing file for read only
openw	open a new file for read and write,overwrite if existed
openu	open an existing file for updating,i.e.,read and write
findfile()	return names of files in the current directory
dialog_pickfile()	graphical file selector
fstat()	return information about an open file
eof()	check for end-of-file
close	close a file
free_lun	free a logical unit number and close a file

4.2.2 opening files

Read and write operations are performed using the logical unit number to reference the file.

a specific logical unit number, which may be set to a scalar long value in the range of 1 to 99

```
lun = 20l
openw, lun, 'test.dat'
free_lun, lun
```

```
openr, 5, filename
close, 5
```

assign a free logical unit, an unused logical unit number in the range of 100 to 128 will be assigned

```
openw, lun, 'test.dat', /get_lun
free_lun, lun
```

4.2.3 selecting a file

```
list = findfile()
print, list
```

```
list = findfile('*.dat')
list = findfile('*.dat', count=numfiles)
```

```
file = dialog_pickfile()
file = dialog_pickfile(filter='*.dat')
file = dialog_pickfile(filter='*.pro', /multiple)
directory = dialog_pickfile(/directory)
```

most of the time, if we include a *Dialog_Pickfile* in a program, we just check for the null string returned from the dialog and return out of the program

```
file = dialog_pickfile(file='*.jpg', path=!dir)
if file eq '' then return
```

4.2.4 *filepath*

```
galaxy = filepath('galaxy.dat', subdirectory=['examples', 'data'])
```

```
cd, current=thisdir
galaxy = filepath('galaxy.dat', root_dir=thisdir, $
                  subdirectory='coyote')
```

4.2.5 obtaining information about files

1. specific lun

   ```
   file = filepath('hurric.dat', subdir='examples/data')
   openr, lun, file, /get_lun
   info = fstat(lun)
   help, info.size, info.read, info.name
   free_lun, lun
   ```

2. all luns

   ```
   help, /files
   ```

   if you want to close all files

   ```
   close, /all
   ```

4.2.6 reading and writing formatted (ASCII) files

1. reading a formatted file

   if the size data is known

   ```
   openr, lun, 'input.dat', /get_lun
   data = lonarr(10,100)
   readf, lun, data, format='(10i6)'
   free_lun, lun
   ```

if you want to read a formatted file that has a known format but an unknown length,e.g.,in the following format:

```
22:02:13.34UTC -29.93455 -116.0234W 234.2
22:02:14.33UTC -29.9357S -116.0232W 234.1
22:02:15.34UTC -29.9362S -116.0230W 234.1
22:02:16.35UTC -29.9367S -116.0228W 234.2
22:02:17.36UTC -29.9373S -116.0226W 234.2
22:02:18.34UTC -29.9378S -116.0224W 234.3
```

```
FUNCTION READ_POSITION, FILE, MAXREC=MAXREC

  ;; check argument
  if (n_elements(file) eq 0) then $
     message, 'Argument FILE is undefined'
  if (n_elements(maxrec) eq 0) then $
     maxrec = 10000L

  ;; open input file
  openr, lun, file, /get_lun

  ;; define record structure and create array
  fmt = '(2(i2, 1x), f5.2, 4x, f8.4, 2x, f9.4, 2x, f5.1)'
  record = {hour:0L, min:0L, sec:0.0, $
            lat:0.0, lon:0.0, head:0.0}
  data = replicate(record, maxrec)

  ;; read record until end-of-file reached
  nrecords = 0L
  recnum = 1L
  while (eof(lun) ne 1) do begin

     ;; read this record (jumps to bad_rec: on error)
     on_ioerror, bad_rec
     error = 1
     readf, lun, record, format=fmt
     error = 0

     ;; store data for this record
     data[nrecords] = record
     nrecords = nrecords + 1

     ;; check if maximum record count exceeded
     if (nrecords eq maxrec) then begin
        free_lun, lun
        message, 'Maximum record reached: increase MAXREC'
     endif

     ;; check for bad input record
     bad_rec:
     if (error eq 1) then $
        print, 'Bad data at record', recnum
     recnum = recnum + 1

  endwhile

  ;; close input file
```

```
  free_lun, lun

  ;; trim data array and return it to caller
  data = data[0 : records - 1]
  return, data

END
```

```
result = read_position('position.dat')
print, result[0].sec, result[0].lat
```

The *on_ioerror* procedure is used to handle any errors when reading from the input file, such as a record that with an invalid format.

2. writing a formatted file

   The *printf* procedure writes data to a formatted file that has been opened for writing or updating.

```
data = lindgen(10, 100)
openw, lun, 'output.data', /get_lun
printf, lun, data, format='(10i6)'
free_lun, lun
```

If you use *printf* (instead of *print*) in your program, it's easy to send the printed output to the screen by default, or to an output file if required. This is possible because the logical unit number -1 is reserved for standard output.

```
PRO REDIRECT, OUTFILE=OUTFILE

  ;; Select logical unit for output
  if (n_elements(outfile) eq 1) then begin
     openw, outlun, outfile, /get_lun
  endif else begin
     outlun = -1
  endelse

  ;; printf is used in the program for output
  printf, outlun, 'Starting program execution at', system()

  ;; Body of the program goes here

  ;; Close the output file if needed
  if (outlun gt 0) then free_lun, outlun

END
```

Formatted output to a file is limited to 80 columns by default. To increase the default output width, use the *width* keyword when the *openw* or *openu* procedure is called

```
openw, lun, 'wide.dat', /get_lun, width=150
printf, lun, findgen(100), format='(10f10.2)'
free_lun, lun
```

4.2.7 reading and writing unformatted (binary) files

1. procedures and functions

Table 13: procedures and functions for unformatted I/O

Name	Purpose
readu	read data from an unformatted file
point_lun	reposition the file pointer in an open file
writeu	write data to an unformatted file
assoc()	associate an logical unit number with a variable
save	save variable in a portable IDL-specific format
restore	restore variables from a save file

2. reading an unformatted file (single data type)

   If an unformatted file contains only one kind of binary data (e.g., byte), and the file size is known in advance, then reading the contents of the file is straightforward.

```
file = filepath('ctscan.dat', subdir='examples/data')
openr, lun, file, /get_lun
data = bytarr(256, 256)
readu, lun, data
free_lun, lun
tvscl, data
```

   If the size of the file is not known in advance, but the size of each record is known

```
file = filepath('hurric.dat', subdir='examples/data')
openr, lun, file, /get_lun
info = fstat(lun)
ncols = 440L
nrows = info.size / ncols
data = bytarr(ncols, nrows)
readu, lun, data
free_lun, lun
tvscl, data
```

3. reading an unformatted file (mixed data type)

   Unformatted data files may contain more than one data type per record.

```
FUNCTION READ_WEATHER, FILE

  ;; Check argument
  if (n_elements(file) eq 0) then $
     message, 'Argument FILE is undefined'

  ;; Open the input file
  openr, lun, file, /get_lun

  ;; Define record structure, and create data array
  record = {id:0L, year:0, month:0, day:0, hour:0, minute:0, $
            pres:0.0, temp:0.0, dewp:0.0, speed:0.0, dir:0.0}
  data = replicate(record, 24L * 60L)

```

```
   ;; Read records from the file until EOF
   nrecords = 0
   while (eof(lun) ne 1) do begin
      readu, lun, record
      data[nrecords] = record
      nrecords = nrecords + 1
   endwhile

   ;; Close the input file
   free_lun, lun

   ;; Trim the data array, and return it to caller
   data = data[0 : nrecords - 1]
   return, data

END
```

4. reading a Fortran-77 unformatted file

   Files that are written in Fortran-77 unformatted sequential access mode have extra information attached to the beginning and end of each data record. For example, consider the following Fortran-77 program:

```
      program unformatted_test
!     Fortran-77 program to produce unformatted output
      implicit none
      integer :: a(10)
      real :: b(5)
      integer :: index
      do index = 1, 10
         a(index) = index
      end do
      do index = 1, 5
         b(index) = real(index * 10)
      end do
      open(30, file='unformatted_test.dat', form='unformatted')
      write(30) a
      write(30) b
      close(30)
      end
```

   to read the contents of the output file

```
openr, lun, 'unformatted_test.dat', /get_lun, /f77_unformatted
a = lonarr(10)
b = fltarr(5)
readu, lun, a
readu, lun, b
free_lun, lun
print, a, format='(10i4)'
print, b, format='(5f5.1)'
```

5. repositioning the file pointer

   The file pointer points to the position in the file where the next read or write request will begin. Every time a read or write request is issued, the file pointer is moved the appropriate number of bytes further along in the file.

```
openr, lun, 'tapefile.dat', /get_lun
point_lun, lun, 1024L
data = bytarr(512, 512)
readu, lun, data
free_lun, lun
```

to get the current position of the file pointer

```
info = fstat(lun)
position = info.cur_ptr
```

6. writing binary data to an unformatted file

   When writing binary data to a file, it can be helpful to first write a header record that records the size and type of the data contained in the file.

```
data = dist(256)
openw, lun, 'dist.dat', /get_lun
info = size(data)
info_size = n_elements(info)
header = [info_size, info]
writeu, lun, header
writeu, lun, data
free_lun, lun
```

   to read the file, you would first read the header, and then use the *make_array* function to create an array of the correct size and type to hold the data:

```
openr, lun, 'dist.dat', /get_lun
info_size = 0L
readu, lun, info_size
info = lonarr(info_size)
readu, lun, info
data = make_array(size=info)
readu, lun, data
free_lun, lun
```

7. programs to write and read portable binary data

   The *binread* procedure allows you to read one or more arrays from a binary unformatted file created by *binwrite* in first-in/first-out order. The ”magic” value allows *binread* to determine whether or not byte swapping is required

```
PRO BINWRITE, LUN, DATA

  ;; Check arguments
  if (n_elements(lun) eq 0) then $
     message, 'LUN is undefined'
  if (n_elements(data) eq 0) then $
     message, 'DATA is undefined'

  ;; Check that file is open
  result = fstat(lun)
  if (result.open eq 0) then $
     message, 'LUN does not point to an open file'

  ;; Check that data type is allowed
```

```
   type_name = size(data, /tname)
   if (type_name eq 'STRING') or $
      (type_name eq 'STRUCT') or $
      (type_name eq 'POINTER') or $
      (type_name eq 'OBJREF') then $
         message, 'DATA type is not supported'

   ;; Check that DADA is an array
   if (size(data, /n_dimensions) lt 1) then $
      message, 'DATA must be an array'

   ;; Create header array
   magic_value = 123456789L
   info = size(data)
   info_size = n_elements(info)
   header = [magic_value, info_size, info]

   ;; Write header and data to file
   writeu, lun, header
   writeu, lun, data

END

PRO BINREAD, LUN, DATA

   ;; Check arguments
   if (n_elements(lun) eq 0) then $
      message, 'LUN is undefined'
   if (arg_present(data) eq 0) then $
      message, 'DATA cannot be modified'

   ;; Check that file is open
   result = fstat(lun)
   if (result.open eq 0) then $
      message, 'LUN does not point to an open file'

   ;; Read magic value
   magic_value = 0L
   readu, lun, magic_value

   ;; Devode magic value to see if byte swapping is required
   swap_flag = 0
   if (magic_value ne 123456789L) then begin
      magic_value = swap_endian(magic_value)
      if (magic_value eq 123456789L) then begin
         swap_flag = 1
      endif else begin
         message, 'File was not written with BINWRITE'
      endelse
   endif

   ;; Read the header, swapping if necessary
   info_size = 0L
   readu, lun, info_size
   if swap_flag then info_size = swap_endian(info_size)
   info = lonarr(info_size)
   readu, lun, info
   if swap_flag then info = swap_endian(info)
```

```

  ;; Read the data, swapping if necessary
  data = make_array(size=info)
  readu, lun, data
  if swap_flag then data = swap_endian(temporary(data))

END
```

example of usage

```
a = dist(256)
b = lindgen(1000)
openw, lun, 'test.dat', /get_lun
binwrite, lun, a
binwrite, lun, b
free_lun, lun
openr, lun, 'test.dat'
binread, lun, a
binread, lun, b
free_lun, lun
help, a, b
```

8. reading binary data via an associated variable

```
file = filepath('people.dat', subdir='examples/data')
openr, lun, file, /get_lun
chunk = bytarr(192, 192)
image = assoc(lun, chunk)
tvscl, image[0]
tvscl, image[1]
free_lun, lun
```

5 Basic programming

5.1 defining and compiling programs

5.1.1 procedures

- A main procedure

```
print, 'Hello world'
END
```

- A procedure with no arguments:

```
PRO HELLO
  print, 'hello world'
END
```

- A procedure with three parameters

```
PRO READ_IMAGE, IMAGE, DATA, TIME
  ;; statements...
END
```

- A procedure with three parameters and two optional keywords

```
PRO PRINT_IMAGE, IMAGE, DATA, TIME, LANDSCAPE=LANDSCAPE, $
                 COLOR=COLOR
  ;; statements
END
```

5.1.2 functions

- A function with one parameter

```
FUNCTION F_TO_C, DEG_F
  return, (deg_f - 32.0) * (5.0 / 9.0)
END
```

- A function with three parameters and two optional keywords

```
FUNCTION GET_BOUNDS, IMAGE, LAT, LON, POLAR= POLAR, $
                     NORMAL=NORMAL
  ;; statements...
END
```

5.1.3 naming source files

The standard format for naming an IDL source file is the procedure or function name followed by a . pro extension (e.g., imdisp.pro).

5.1.4 manual compilation

```
.compile hello.pro
.compile hello.pro imdisp.pro sds_read.pro
```

It is also possible to manually compile procedures of functions with the executive commands *.run* or *.rnew*. If the source file named in the *.run* or *.rnew* command is a ”main program”, the ”main program” will be executed.

5.1.5 automatic compilation

IDL automatically compiles procedures and functions as they are needed at runtime.

5.1.6 returning to the main level after an error

If an error in an IDL program causes execution to stop, by default, IDL halts inside the procedure that causes the error.

for example

```
FUNCTION SUM, ARG1, ARG2
  return, arg1 + arg2
END

PRO HELLO, A , B
  print, 'Hello world'
  print, 'The sum of A and B is ', sum(a, b)
END
```

To compile an run the program

```
.compile hello
a = 3
b = 15
hello, a, b
```

However, if the program is called when an argument is undefined

```
hello, a, bbb
```

The error message shows that IDL halted inside the *product* function at line 2 of the source file (hello.pro).

Because execution halted inside the *product* function, only the variables defined in this function are now available at the command line.

```
help, arg1
help, a
```

In addition, you will see warning messages if you try recompiling the source file after an execution halt.

After examining any variables in the failed function or procedure, use the *retall* command to return to the amin level.

```
retall
help, a
help, arg1
```

5.2 control statements

5.2.1 IDL control statements

Table 14: IDL control statements

Statement	Purpose
if	If a condition is true, execute statement(s)
case	Select one case to execute from a list of cases
for	For a specified number of times, execute a statement loop
while	While a condition is true, execute a statement loop
repeat	Repeat a statement loop until a condition is true
return	Return control to the calling function or procedure
goto	Go to a label
switch	Branch to a case in a list of cases
break	Break out of a loop, *case* statement, or *switch* statement
continue	Continue execution on the next iteration of a loop

5.2.2 *if* statement

```
if condition then statement
```

```
if condition then begin
  statement(s)
endif
```

```
if condition then statement else statement
```

```
if condition then begin
   statement(s)
endif else begin
   statement(s)
endelse
```

In all forms , *condition* is a scalar expression that evaluates to true or false.

5.2.3 *case* statement

```
case expression of
  exp1:
  exp2: statement
  exp3: begin
     statement(s)
  end
  else: statement
endcase
```

Here, *expression*, *exp1*, *exp2* and *exp3* are scalar expression.

The *case* statement can also be used in conjunction with relational and Boolean operators.

```
case 1 of
  cond1: statement
  cond2: begin
     statement(s)
  end
  else: statement
endcase
```

Here *cond1* and *cond2* are expressions containing relational operators that evaluate to either true or false, such as (x gt 100).

5.2.4 *for* statement

```
for i = v1, v2 do statement
```

```
for i = v1, v2, inc do statement
```

```
for i = v1, v2, inc do begin
  statements(s)
endfor
```

If the default *int* type is used for the loop start variable, large loop end values can cause problems

```
for i = 0, 40000, 1000 do print, i
```

The loop end value of 40,000 is automatically interpreted as a *long* since it is greater than 32,767, and therefore a mismatch exists between the types of the loop start and end values. For this reason, *long* loop control variables are recommended.

```
for i = 0L, 40000L, 1000L do print, i
```

Floating-point loop variables are permitted, but are not recommended. If you need to create floating-point variables inside a loop, the preferred approach is to create a float variable from the loop control variable.

```
for i = 1L, 5L, 1L do print, 0.1 * float(i)
```

5.2.5 *while* statement

```
while condition do statement
```

```
while condition do begin
  statement(s)
endwhile
```

5.2.6 *repeat* statement

```
repeat statemnt until condition
```

```
repeat begin
  statement(s)
endrep until condition
```

5.2.7 *return* statement

In general, you should **try** to limit each function to one return statement.

5.2.8 *goto* statement

Jump to a specific location

```
goto, label
```

The destination is specified by *label*

```
label:
```

5.2.9 *switch* statement

```
switch expression of
  exp1:
  exp2: statement
  exp3: begin
     statement(s)
  end
  else: statement
endswitch
```

Here, *expression*, *exp1*, *exp2* and *exp3* are scalar expressions.

5.2.10 *break* statement

The break statement causes an immediate exit from a *for*, *while* , or *repeat* loop, or a case or switch statement. Control is transferred to the next statement after the end of the loop, or an immediate exit from a case or switch statement.

5.2.11 *continue* statement

The continue statement causes the next iteration of a for, while, or repeat loop to be executed. Any statements remaining in the current loop iteration are skipped.

5.3 arguments and keywords

5.3.1 argument passing

Table 15: Arguments passed by reference versus value

Passed by reference	Passed by value
Scalars	Constants
Arrays	Indexed subarrays
Structures	Structure elements
Undefined variables	System variables
	Expressions

5.3.2 extra keywords

When keywords are passed to a procedure or function, it is sometimes necessary to pass these keywords to another routine from within the called routine.

```
PRO EPLOT, X, Y, NAME=NAME, _EXTRA=EXTRA_KEYWORDS

  ;; Check arguments
  if (n_params() ne 2) then $
     message, 'Usage: EPLOT, X, Y'
  if (n_elements(x) eq 0) then $
     message, 'Argument X is undefined'
  if (n_elements(y) eq 0) then $
     message, 'Argument Y is undefined'
  if (n_elements(name) eq 0) then name = 'Faxiang Zheng'

  ;; Plot the data
  plot, x, y, _extra=extra_keywords

  ;; Print name and date on plot
  date = systime()
  xyouts, 0.0, 0.0, name, align=0.0, /normal
  xyouts, 1.0, 0.0, date, align=1.0, /normal

END
```

then in your IDL shell,

```
x = findgen(200) * 0.1
eplot, x, sin(x), xrange=[0, 15], $
       xtitle='X', ytitle='SIN(X)'
```

5.3.3 extra keywords precedence

A keyword passed to a procedure or function via the extra keyword method takes precedence when a keyword with the same name is passed in the argument list.

```
PRO KPLOT, _EXTRA=EXTRA_KEYWORDS

  plot, indgen(10), psym=2, _extra=extra_keywords

END
```

When this procedure is called with no keywords, the *plot* procedure uses plot symbol 2. If the *psym* keyword is passed when *kplot* is called, the *plot* procedure uses the value of *psym* passed via the extra keyword structure and ignore the explicit setting of *psym=2* in the argument list.

```
kplot
kplot, psym=4
```

If you wish to use a keyword inside a procedure or function that would otherwise be passed via the extra keyword method, you must include the keyword in the procedure or function definition.

```
PRO KPLOT, PSYM=PSYM, _EXTRA=EXTRA_KEYWORDS

  if (n_elements(psym) eq 0) then psym = 2
  print, 'PSYM = ', psym
  plot, indgen(10), psym=psym, _extra=extra_keywords

END
```

5.3.4 extra keyword passing

If the *_ref_extra* keyword is used instead of *_extra* in the procedure or function definition, then the keywords are passed by reference and therefore can be modified by the routine to which they are passed.

```
FUNCTION SQUARE, ARG, POSITIVE=POSITIVE
  result = arg ^ 2
  if (arg gt 0) then positive = 1 else positive = 0
  return, result
END

PRO TEST_VAL, INPUT, OUTPUT, _EXTRA=EXTRA_KEYWORDS
  output = square(input, _extra=extra_keywords)
END

PRO TEST_REF, INPUT, OUTPUT, _REF_EXTRA=EXTRA_KEYWORDS
  output = square(input, _extra=extra_keywords)
END
```

```
.compile test_extra
test_val, 2.0, output, positive=pos_flag
help, pos_flag

test_ref, 2.0, output, positive=pos_flag
help, pos_flag
```

5.4 checking parameters and keywords

5.4.1 routines

Table 16: Routines for checking arguments

Name	Purpose
n_params()	Returns number of parameters passed (not including keywords)
n_elements()	Returns number of elements in a variable (zero means variable is undefined)
size()	Returns size and type information about a variable
arg_present()	Returns true if an argument was present and was passed by reference
message	Print a message and halts execution

5.4.2 checking input parameters and keywords

Consider a hypothetical plotting procedure named *myplot.pro*, which is defined as follows

```
MYPLOT, X, Y, POSITION=POSITION
```

The parameters *x* and *y* are one-dimensional vectors. Both parameters are required, and they must be the same size. The *position* keyword must be one-dimensional vector with four elements if supplied; otherwise the default value is [0.0, 0.0, 1.0, 1.0]. The following checks are appropriate for this routine.

1. Was the correct number of input parameters passed?

```
if (n_params() ne 2) then $
   message, 'Usage: MYPLOT, X, Y'
```

2. Is each input parameter defined?

```
if (n_elements(x) eq 0) then $
   message, 'X is undefined'
if (n_elements(y) eq 0) then $
   message, 'Y is undefined'
```

3. Does each input parameter have the correct number of dimensions?

```
if (size(x, /n_dimensions) ne 1) then $
   message, 'X must be a 1D array'
if (size(y, /n_dimensions) ne 1) then $
   message, 'Y must be a 1D array'
```

4. Are the input parameters the same size?

```
if (n_elements(x) ne n_elements(y)) then $
   message, 'X and Y must be same size'
```

5. If a keyword was not passed, is a default value set?

```
if (n_elements(position) eq 0) then $
   position = [0.0, 0.0, 1.0, 1.0]
```

6. If a keyword was passed, does it have the correct number of elements?

```
if (n_elements(position) ne 4) then $
   message, 'POSITION must have 4 elements'
```

5.4.3 don't modify input parameters

You should avoid modifying input parameters.

5.4.4 checking boolean keywords

Boolean keywords are checked using the *keyword_set* function, which returns true if the argument is defined, **and** is a nonzero scalar, **or** an array of any size, type, or value. For example, the *myplot* procedure shown previously could an *erase* keyword which causes the screen to be erased if it is set to true.

```
if keyword_set(erase) then erase
```

or

```
erase = keyword_set(erase)
if (erase) then erase
```

In this case, if erase is a nonzero scalar, or an array of any size, type, or value, it will be converted to a scalar int equal to 1, which will be returned to the caller.

Another case where you may wish to use a Boolean keyword within a procedure or function is to pass it to another routine that accepts a Boolean keyword with the same name.

```
plot, x, y, erase=keyword_set(erase)
```

5.4.5 checking output parameters and keywords

Output arguments must be passed by reference if they are to be created or modified within a called routine and returned to the caller. The *arg_present* function is used to check for arguments that can be modified inside a called routine. If a parameter or keyword was present in the calling sequence, and the parameter or keyword was passed by reference, *arg_present* returns true.

```
if arg_present(xrange) then $
   xrange = [min(x) - 1.0, max(x) + 1.0]
```

In the case of mandatory output parameters, *arg_present* allows you to detect parameters that cannot be modified.

```
if (arg_present(slope) ne 1) then $
   message, 'SLOPE cannot be modified'
```

5.5 scripts

The simplest IDL program is writing multiple command lines to a file, just the same as the commands used in the interactive IDL shell. For example, put the following commands into *hello.pro*

```
print, 'hello world'
print, 'I love IDL'
for index = 0, 10 do print, index
```

Then, you can call the command in the file by

```
@hello
```

in your IDL shell.

However, the following multiple lines for loop is not permitted in a script because it contains a statement block

```
for index = 0, 10 do begin
  print, index
endfor
```

It is possible to bypass this behavior by including multiple statements on one line separated by the & character

```
for index = 0, 10 do begin & print, index & endfor
```

This method of defining a loop should only be used in a script or on the command line.

5.6 include files

An include file contains a sequence of IDL statements that are inserted in a procedure or function at compile time. Include files may contain any statement that is legal in a procedure or function, including multiline statement blocks. For example, the following physical constants could be contained in an include file named *fundamental_constants.inc*

```
planck_constant = 6.6260755d-34     ; Joule second
light_speed = 2.9979246d+8          ; meters per second
boltzmann_constant = 1.380658d-23   ; Joules per Kelvin
rad_c1 = 2.0d0 * planck_constant * light_speed^2
rad_c2 = (planck_constant * light_speed) / boltzmann_constant
```

To include the file in a function named *planck.pro* that computes monochromatic Planck radiance given wavenumber and temperature

```
FUNCTION PLANCK, V, T
  @fundamental_constants.inc
  vs = 10d2 * v
  return, vs^3 * ((rad_c1 * 10D5) / $
                  (exp(rad_c2 * (vs / t)) - 1.0D0))
END
```

The include file is silently inserted into the function at compile time

5.7 journal

A journal of an IDL session can be created by calling the *journal* procedure

```
journal
```

The default journal file *idlsave.pro* is created in the current directory. The *journal* procedure accepts one optional parameter specified the path and name of the journal file.

```
journal, 'mydemo.pro'
```

Journal ends when *journal* is called without any arguments, or when the IDL session ends. To replay a journal file, simply use the 'at' character

```
@mydemo
```

5.8 global variables

5.8.1 read-only system variables

Table 17: Commonly used read-only system variables

System variable	Meaning
!dtor	Degrees to radians conversion factor
!radeg	Radians to degrees convolution factor
!pi	π (single precision)
!dpi	π (double precision)
!version.release	IDL version number (string)
!version.os_family	Platform identifier
!d.flags	Bitmask containing information about the current graphics device
!d.n_colors	Number of possible colors in palette
!d.name	Name of the current graphics device
!d.table_size	Size of current graphics window
!d.window	Index of current graphics window
!d.x_vsize	Viewable horizontal size of current graphics device (in pixels)
!d.y_vsize	Viewable vertical size of current graphics device (in pixels)
!x.window	Horizontal coordinates of current plot (normalized units)
!y.window	Vertical coordinates of current plot (normalized units)

5.8.2 writable system variables

Table 18: Commonly used writable system variables

System variables	Meaning
!p.multi	Configures multipanel plots
!p.font	Selects default font for plotted characters
!order	Controls image display order (bottom up or top down)
!path	The current search path for procedures and functions
!except	Controls math error behavior

5.8.3 user-defined system variables

```
defsysv, 'plot_info', 10.0
help, !plot_info
```

By default, system variables created by *defsysv* are writable, but their type and size cannot be changed

```
!plot_info = -32.0
help, !plot_info
```

However, read-only system variables can be created by adding an optional nonzero argument to *defsysv*, as shown in the following example, where the speed of light is defined in meters per second

```
defsysv, '!speed_of_light', 2.9979246d+8, 1
help, !speed_of_light
!speed_of_light = 123.45
```

To remove user-defined system variables, you can either exit IDL and start a new session, or issue the *.reset_session* executive command at the IDL command line to reset the session.

```
.reset_session
help, !plot_info
```

To check for the existence of a system variable, the *exist* keyword is used in conjunction with *defsysv* to return a logical variable

```
defsysv, '!plot_info', 20.0
defsysv, '!plot_info', exist=exist
help, exist
defsysv, '!plot_nothing', exist=exist
help, exist
```

5.9 error handling

5.9.1 *on_error*

When IDL encounters a fatal error in a procedure or function, the default behavior is to stop execution in the routine that cause the error. This default behavior may be modified by calling the *on_error* procedure, which accepts a single scalar argument specify the default error behavior.

Table 19: Arguments values for *on_error*

Value	Behavior when error occurs
0	Stop the routine that cause the error (default)
1	Return to the main level
2	Return to the caller of the routine that cause the error
3	Return to the routine that cause the error

5.9.2 intercepting errors

If you wish to manually intercept error and provide your own error- handling code, the *catch* procedure can be used.

The call to ca tch establishes the error handler and creates the variable *error_status* , which is set to zero. In the event of a subsequent error, *error_status* is set to a nonzero error code.

```
FUNCTION NCDF_ISNCDF, FILE

  ;; Check arguments
  if (n_params() ne 1) then $
     message, 'Usage: RESULT = NCDF_ISNCDF(FILE)'
  if (n_elements(file) eq 0) then $
     message, 'Argument FILE is undefined'

  ;; Establish error handler
  catch, error_status
  if (error_status ne 0) then return, 0

  ;; Attempt to open the file
  ncid = ncdf_open(file)

  ;; Close the file and return to caller
  ncdf_close, ncid
  return, 1

END
```

If you wish to obtain further information about an error that was handled by *catch*

```
if (error_status ne 0) then begin
  help, /last_message, output=errtext
  print, errtext[0]
  return, 0
endif
```

5.9.3 math errors

IDL is able to detect math errors such as overflow, underflow, and divide-by-zero.In these cases, IDL substitutes the special floating-point values *NaN* (not a number) or *Inf* (infinity) for the value that caused the error

```
print, sqrt(-1.0)
print, 1.0/0.0
```

The default behavior is to continue execution and print a warning message when control is returned to the command line. To change the default behavior, you can change the system variable *!except*,

Table 20: Values for system variable !except

Value	Behavior when math error occurs
0	Do not print a warning
1	Print a warning when control returns to the command line (default)
2	Print a warning when the exception occors

If you wish to check the math error status without printing a warning message, set *!except* to zero and call the *check_math* function

The *finite* function may be used to check for the presence of *NaN* and *Inf*.

```
arr = [0.0, 1.0, sqrt(-1.0), 4.0, 1.0 / 0.0]
print, finite(arr)
```

To locate non-finite values and set them to a missing value, use the *finite* function in conjunction with *where* to return the indices of the non-finite numbers

```
missing = -999.9
index = where(finite(arr) eq 0, count)
if (count gt 0) then arr[index] = missing
print, arr
```

5.10 efficient programming

5.10.1 conserving memory

The *temporary* function returns a temporary copy of a variable, and sets the original variable to "undefined". This function can be used to conserve memory when performing operations on large arrays, as it avoids making a new copy of results that are only temporary. In general, the *temporary* routine can be used to advantage whenever a variable containing an array on the left hand side of an assignment statement is also referenced on the right hand side.

```
;; generate an array
a = dist(500, 500)
;; normal way, requires more space
a = a + 1.0
;; better way, requires no additional space
a = temporary(a) + 1.0
```

If multiple operations are performed on the array, they can be performed one step at a time to conserve memory

```
;; normal way
a = a * (1.0 / max(a))
;; better way
maxval = max(a)
a = temporary(a) / maxval
```

When an array is replaced by a sub-array

```
;; generate an array
a = dist(256)
;; normal way
a = a[0:63, 0:63]
;; better way
a = (temporary(a))[0:63, 0:63]
```

5.10.2 using efficient methods

If you want to find the sum of all elements greater than 100.0 in a frequency distribution array with 500 columns and 1,000 rows.

```
;; generate an array
a = dist(500, 1000)
;; normal way
sum = 0.0
for i = 0L, n_elements(a) - 1L do $
   if (a[i] gt 100.0) then sum = sum + a[i]
;; a better way
index = where(a gt 100.0, count)
if (count gt 0) then sum = total(a[index])
;; another better way
sum = total(a * (a gt 100.0))
```

To time an operation

```
;; measure time
t0 = systime(1) & sum = total(a * (a gt 100.0)) & $
   t1 = systime(1)
print, t1 - t0
```

6 Window

6.1 create window

```
window
window, 10
window, /free
```

6.2 current window number

```
print, !D.Window
```

6.3 set specific window to current

```
wset, 10
```

6.4 delete windows

6.4.1 delete a window

```
wdelete, 10
```

6.4.2 delete all windows

```
WHILE !D.Window NE -1 DO Wdelete, !DWindow
```

6.5 window size and position

```
window, 1, xsize=200, ysize=300, xpos=75, ypos=150
```

6.6 show window

```
wshow, 1
```

6.7 window title

```
window, title='fuck'
```

6.8 erase current window

```
erase
```

6.9 device independent graphics windows

```
cgdisplay, 500, 400
cgdisplay, 500, 400, wid=2
cgdisplay, 500, 400, /free
```

6.10 resizable windows

```
cgwindow, 'plot', findgen(11)
```

```
loadct, 33, rgb_table=palette
cgPlot, cgdemodata(1), layout=[2, 2, 3], color='red', /window
cgContour, cgdemodata(2), nlevels=12, layout=[2, 2, 1], $
           xstyle=1, ystyle=1, color='dodger blue', /addcmd
cgSurf, cgDemoData(2), /elevation, layout=[2, 2, 4], $
        palette=palette, /addcmd
cgImage, cgDemoData(19), multimargin=4, /axes, $
        layout=[2, 2, 2], /addcmd
```

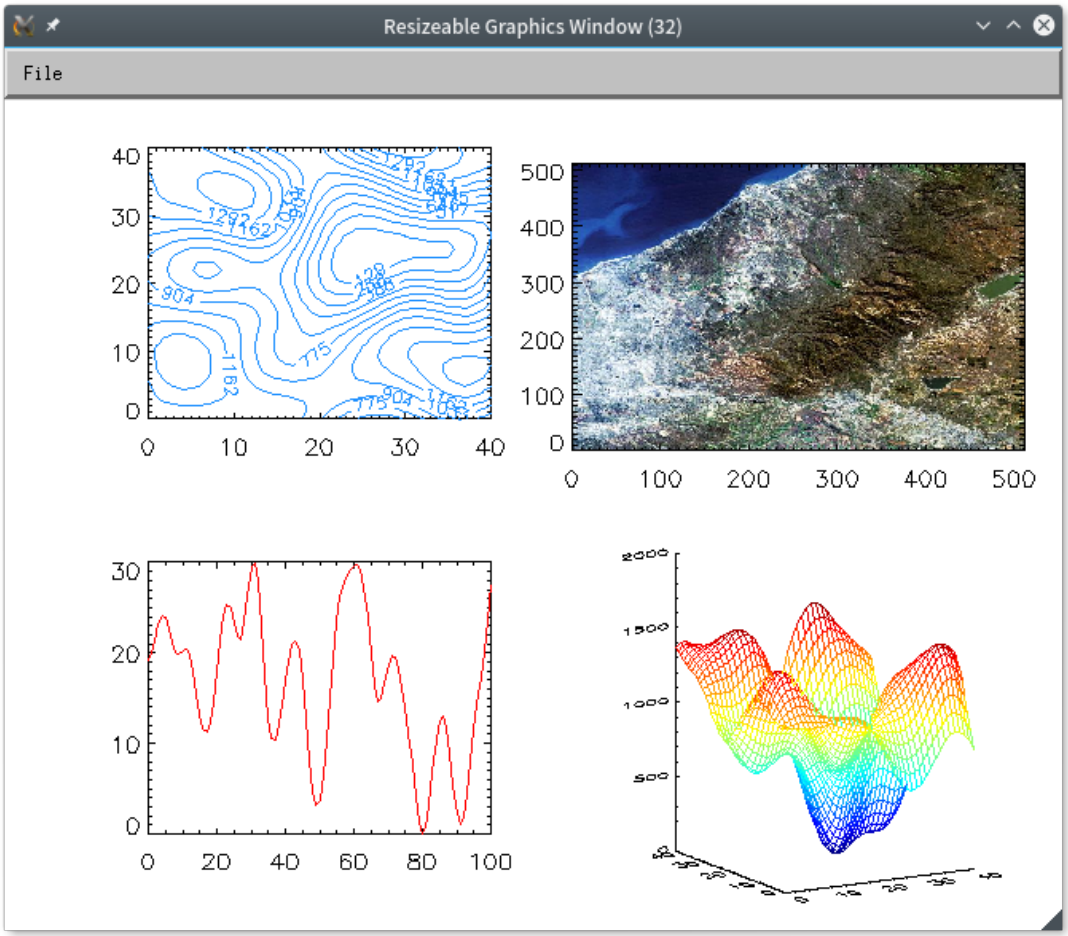

Figure 1: Resizable Window

7 Color

7.1 information about your graphic device

```
window
device,get_visual_name=theVisual,$
    get_visual_depth=theDepth
print,theVisual,theDepth
```

Normally, you will get 'TrueColor' and '24'. **If not**, add following content to your ~/.Xdefaults

```
idl.gr_visual: TrueColor
idl.gr_depth: 24
```

Or,add following command to your IDL startup file

```
Device,TrueColor=24
```

7.2 color models

There are two color modes in IDL, i.e., *decomposed color* (default) and *indexed color*

```
Device, Decomposed=1 ; Selects Decomposed Color Model.
Device, Decomposed=0 ; Selects Indexed Color Model.
```

Every color in IDL is represented, ultimately, as a three-element byte vector of red, green, and blue values, in which each value can vary between 0 and 255. We call this vector a *color triple*. Totally, we can get 256*256*256=16777216 kinds of different color.This is also known as *true color*.

In *decomposed* color model,

```
device,decomposed=1
clr=[R,G,B]
DecomposedColor=clr[0]+clr[1]*2L^8+clr[2]*2L^16
```

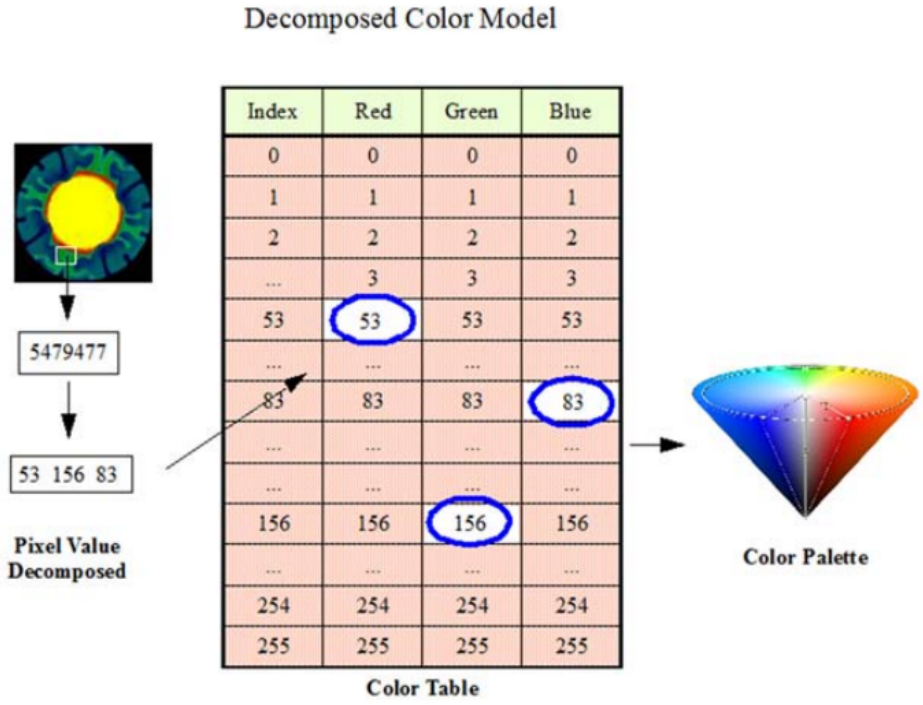

Figure 2: decomposed color, *credit: David W. Fanning*

Things differ in *indexed* color model.Suppose we load a at color index 100 with the *TVLCT* command, like this

```
device,decomposed=0
TVLCT,R,G,B,100
IndexColor=100
```

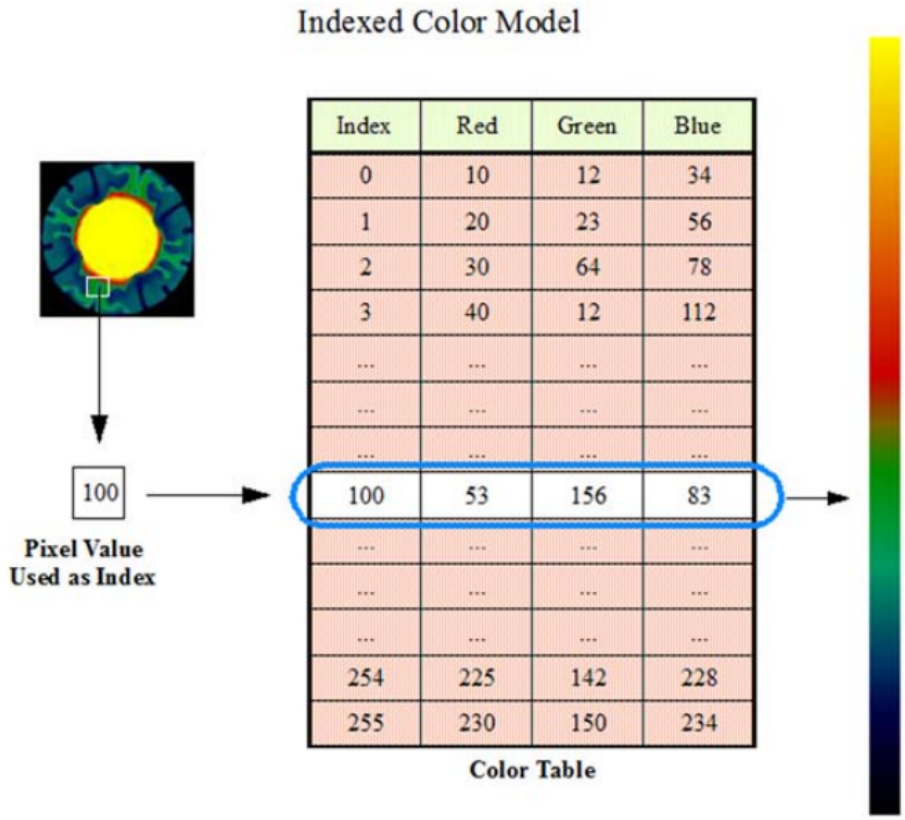

Figure 3: indexed color, *credit: David W. Fanning*

Once a color index is loaded, the color palette is limited to 256, but it is enough in most of cases.

7.3 specify a color by name

use *cgColor* from the *Coyote Library*, or simply use *PickColorName* program

7.4 load color table indices

7.4.1 *TvLCT*

```
1 TvLCT,red,green,blue,position
```

use *CIndex* command to see the result

7.4.2 *LoadCT*

used to load standard color tables come with IDL

7.4.3 *XloadCT*

7.4.4 *Xpalette*

8 Line plot

8.1 simple curve

```
curve=cgdemodata(1)
time = (findgen(100) + 1) * 6.0 / 100
plot,time,curve,xtitle='time axis',ytitle='signal strength',$
     title='experiment 35M',charsize=1.5, color=cgcolor('black'), $
     background=cgcolor('white')
```

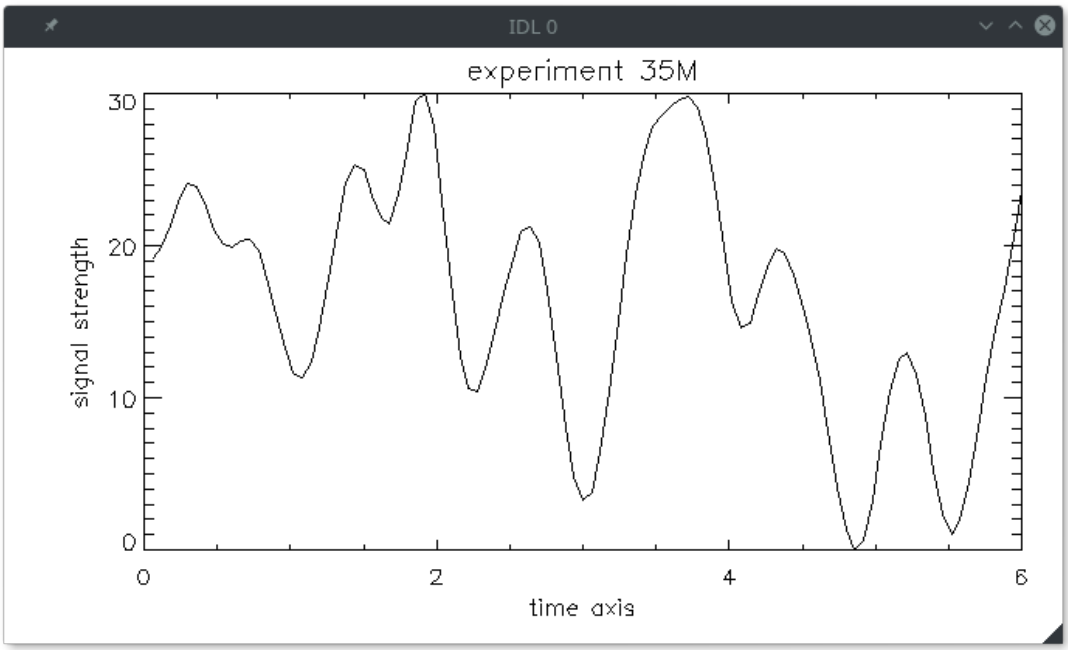

Figure 4: simple plot

set defualt char size

```
!P.CharSize=1.5
```

8.2 OPlot

plot multiple curves on a single window

```
plot, curve, color=cgcolor('black'), background=cgcolor('white')
oplot, curve/2.0, linestyle=1, color=cgcolor('black')
oplot, curve/5.0, linestyle=2, color=cgcolor('black')
```

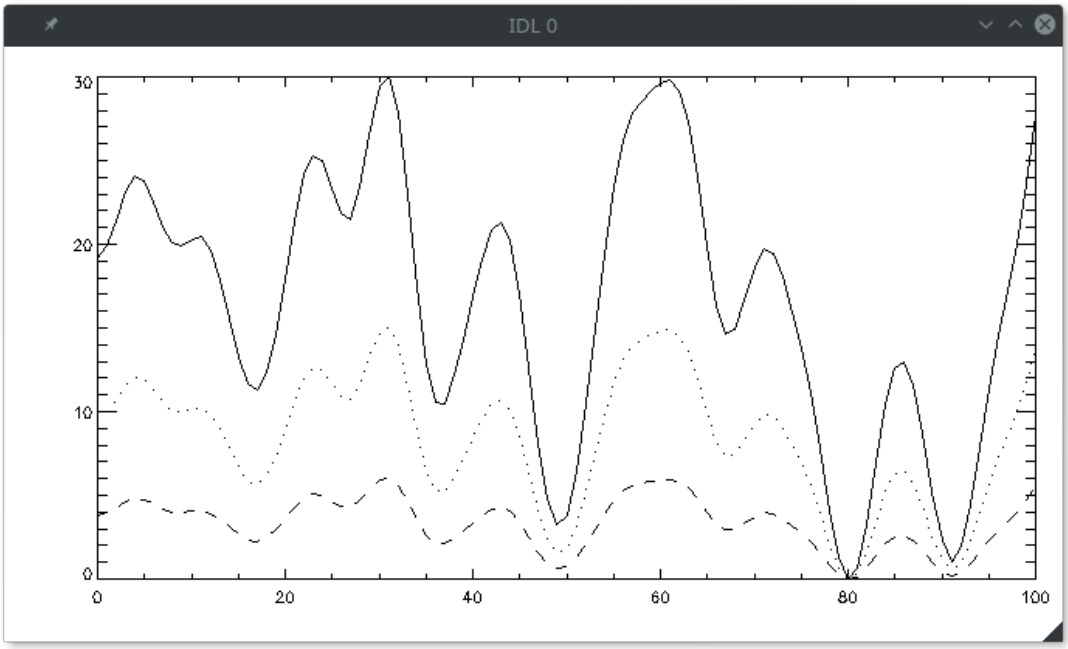

Figure 5: oplot

we can create second y axis by *save* keyword

```
plot, curve, ystyle=8, ytitle='solid line', $
      position=[0.13,0.15,0.85,0.95], color=cgcolor('black'), $
      background=cgcolor('white'), charsize=1.5
axis, yaxis=1, yrange=[0,max(curve*5+1)], /save, $
      ytitle='dashed line', color=cgcolor('black'), charsize=1.5
oplot, curve*5, linestyle=2, color=cgcolor('black')
```

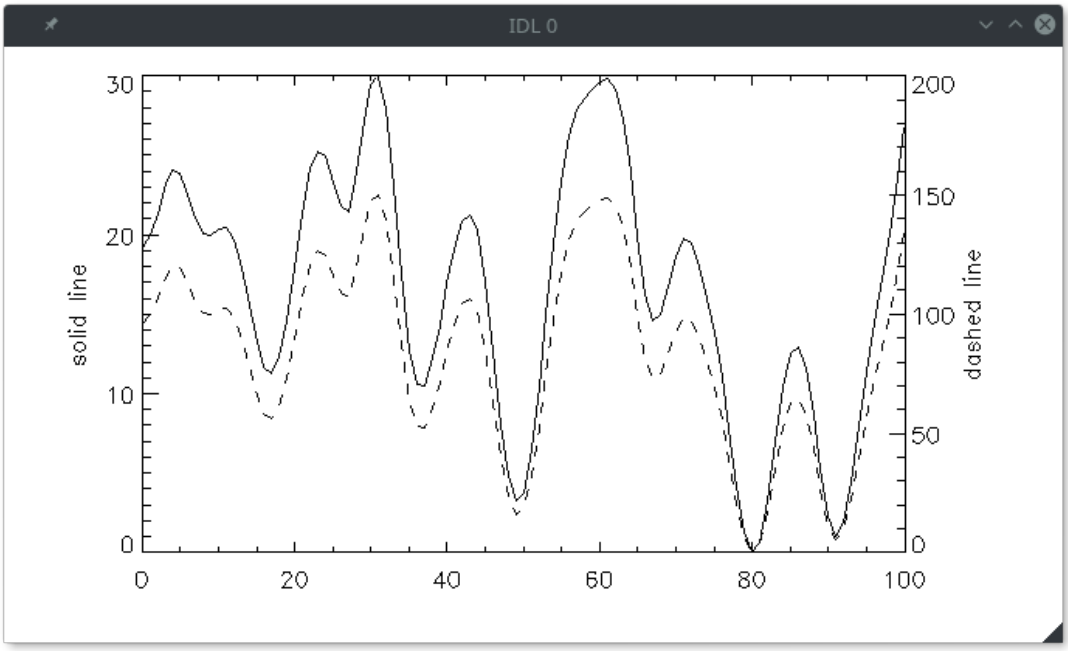

Figure 6: oplot two y axis

8.3 modify your plot

8.3.1 thick

A floating-point value between 1.0 (the default) and 10.0

8.3.2 line style

Table 21: LineStyle

Value of linestyle	LineStyle
0	solid line
1	dotted
2	dashed
3	dash dot
4	dash dot dot dot
5	long dash
6	no line draw

8.3.3 symbol

Table 22: Psym

Value of Psym	symbol
0	no symbol
1	plus sign
2	asterisk
3	period
4	diamond
5	triangle
6	square
7	X
8	UserSym
9	unused
10	ladder
-Psym	symbols connected by lines

We can create our user symbol

```
;; define symbol
x = [0.0, 0.5, -0.8, 0.8, -0.5, 0.0]
y = [1.0, -0.8, 0.3, 0.3, -0.8, 1.0]
usersym, x, y, /fill
;; plot
vector = randomu(seed, 10) * 10
plot, vector, color=cgcolor('black'), background=cgcolor('white'), $
      xstyle=1
oplot, vector, color=cgcolor('red'), psym=8, $
       symsize=4.0, noclip=1
```

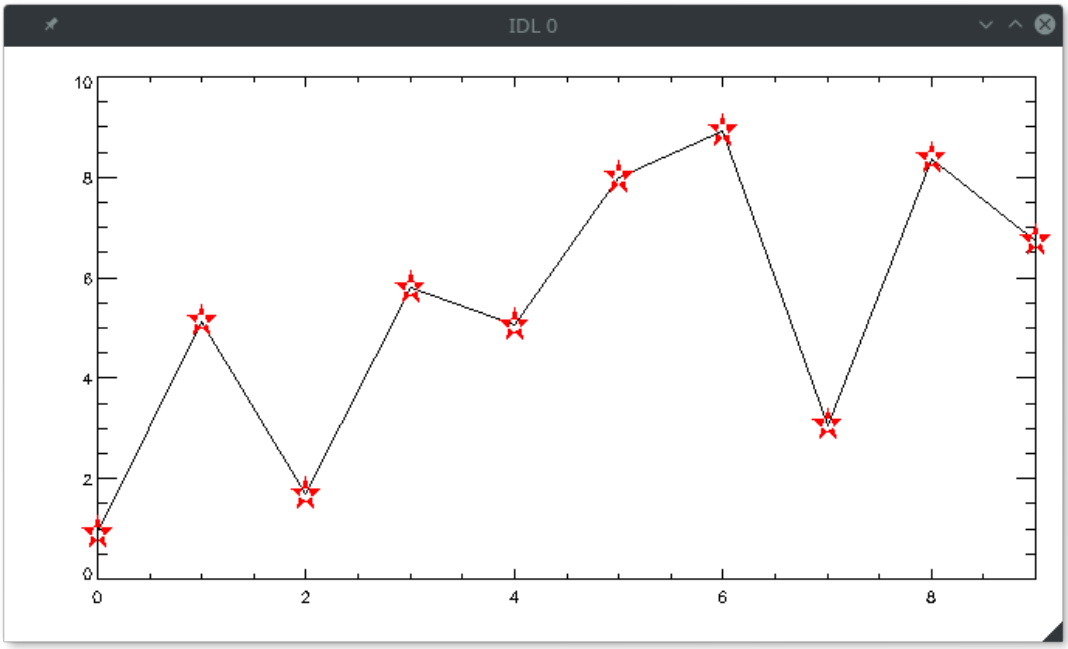

Figure 7: plot user symbol

8.3.4 color

```
vector = randomu(seed, 10) * 10
plot, vector, color=cgcolor('black'), background=cgcolor('yellow'), $
      /nodata
oplot, vector, color=cgcolor('red')
oplot, vector, color=cgcolor('forest green'), $
       psym=2, symsize=3.0
```

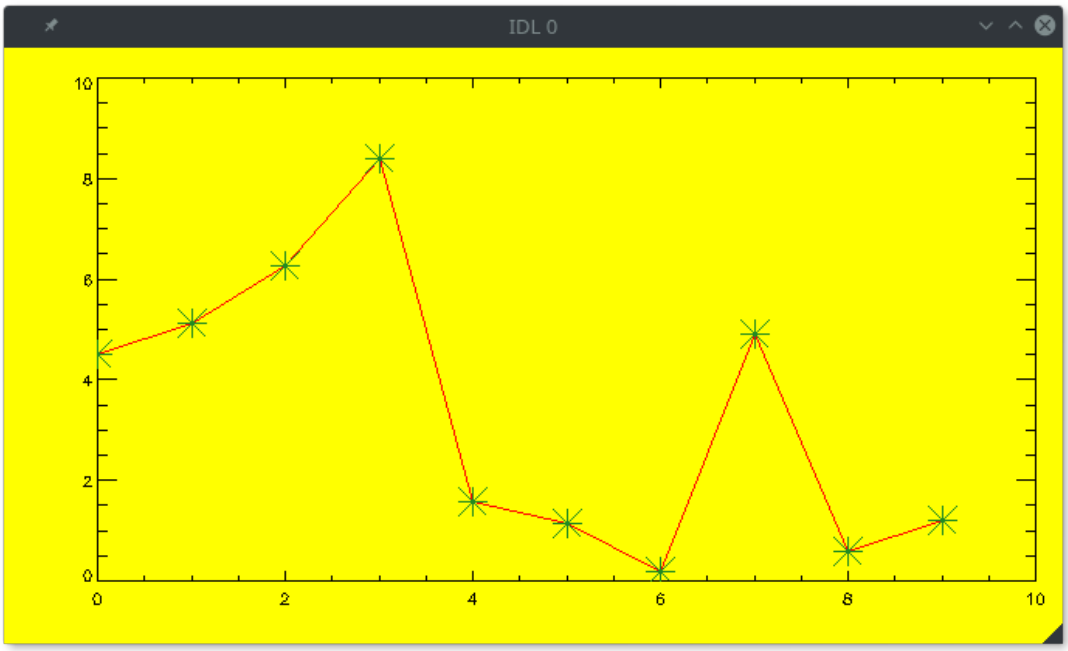

Figure 8: plot color

8.3.5 range

```
plot, time, curve, xrange=[2,4], yrange=[-10,40]
```

8.3.6 [XYZ]Style

Table 23: [XYZ]Style

value of [xyz]style	result
1	accurate axis range
2	expanded axis range
4	no axis
8	no fram
16	Y axis don't start from 0

all are cumulative

```
plot, time, curve, xstyle=9, ystyle=9, color=cgcolor('black'), $
      background=cgcolor('white')
```

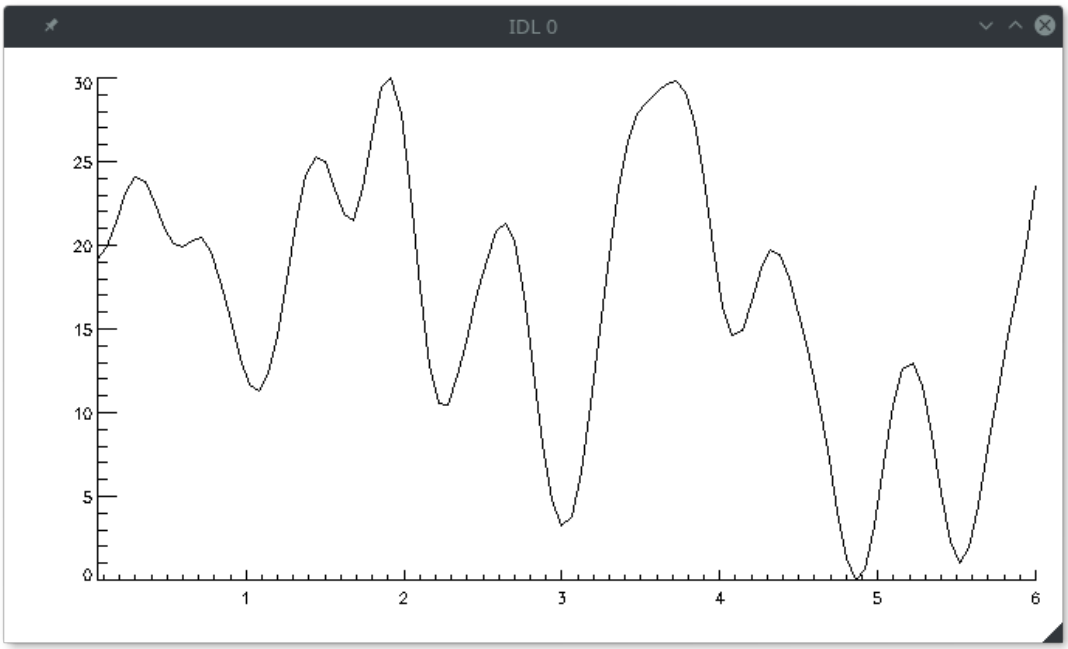

Figure 9: plot [xy]style

8.3.7 grid

when [xy]ticklen=1

```
plot, time, curve, xstyle=9, ystyle=1, xticklen=1, xgridstyle=1,$
      yticklen=1, ygridstyle=1, color=cgcolor('black'), $
      background=cgcolor('white')
```

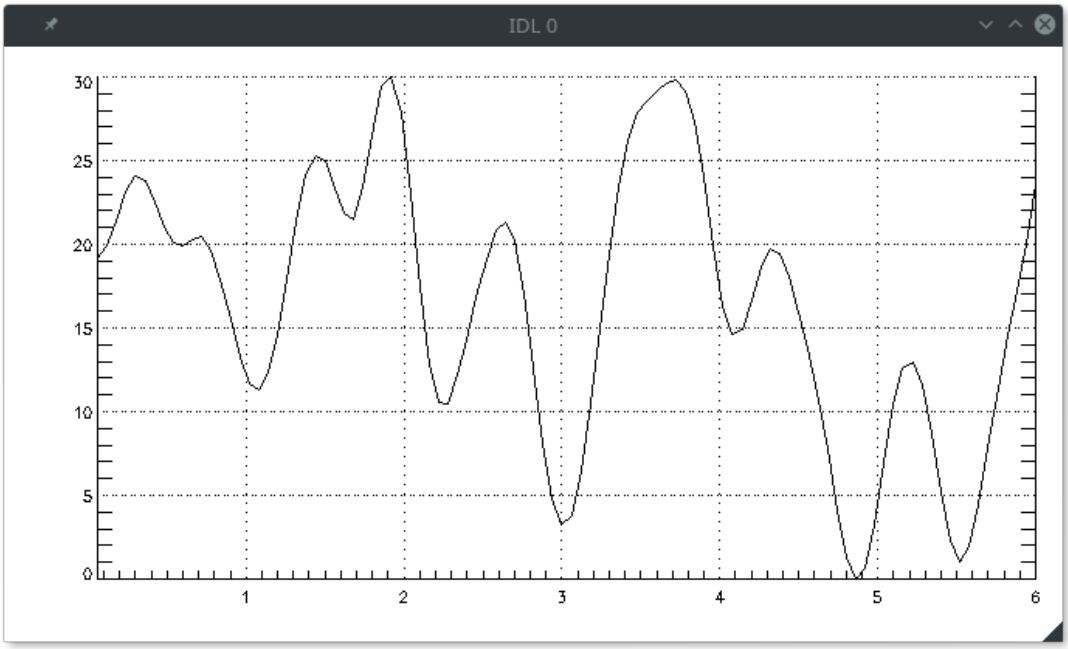

Figure 10: plot grid

8.3.8 tick

1. when *[xy]ticklen* or *ticklen* is smaller than 0

```
plot, time, curve, xstyle=9, ystyle=9, xticklen=-0.03, $
      yticklen=-0.02, color=cgcolor('black'), $
      background=cgcolor('white')
```

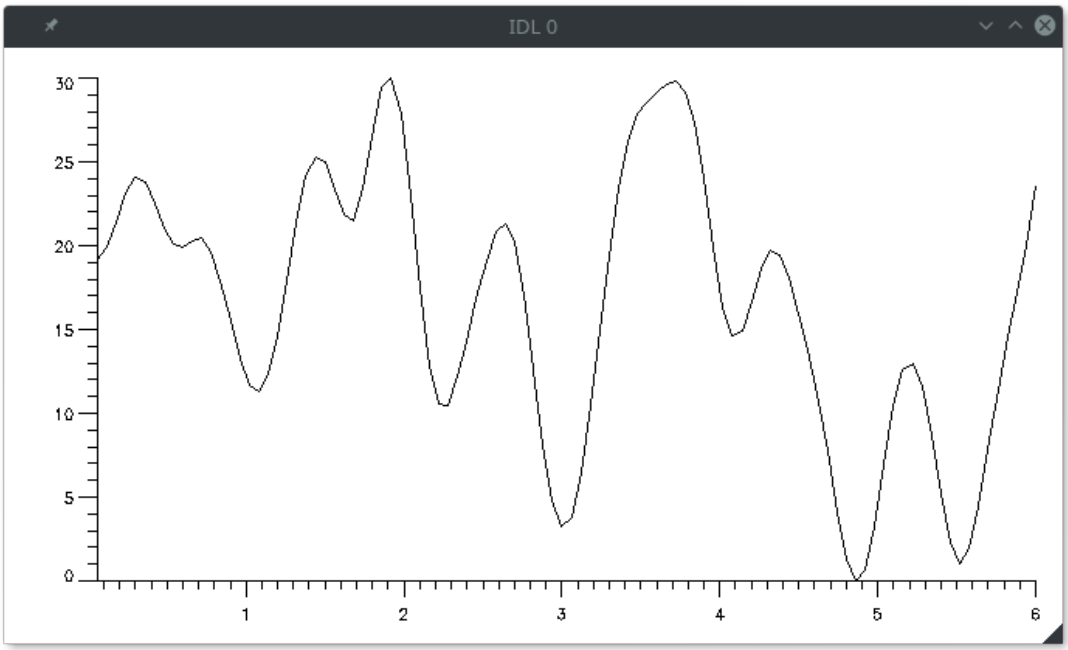

Figure 11: tick1

2. use *[xy]ticks* and *[xy]minor*

   the whole axis is divided into *[xy]ticks* parts and each part will be divided into *[xy]minor* parts

```
plot, time, curve, xstyle=1, xticks=2, xminor=10, yticks=3, $
      yminor=5, xticklen=-0.03, yticklen=-0.02, $
      color=cgcolor('black'), background=cgcolor('white')
```

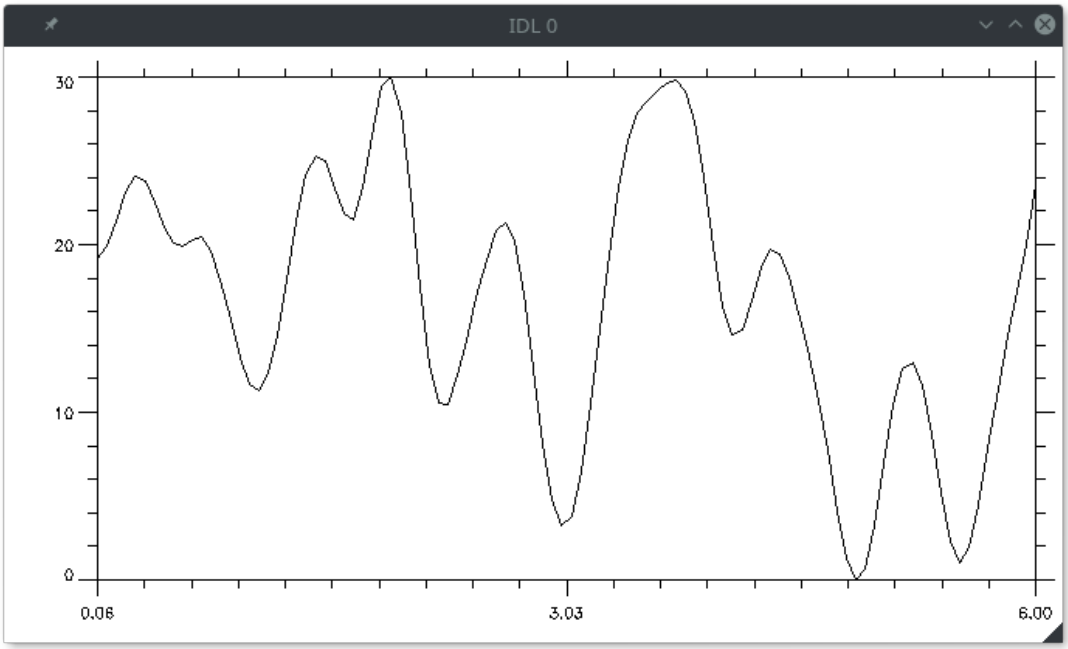

Figure 12: tick2

8.3.9 axis labels

1. no axis labels

```
plot, time, vector, xtitle='Measurements', $
      ytitle='Signal Strength', title='Experiment 1A', $
      charsize=1.5, xcharsize=1.0, ycharsize=1.0, $
      ystyle=1, xtickformat='(A1)', $
      color=cgcolor('black'), background=cgcolor('white')
```

2. creating labels

```
time = findgen(6) * 15.0 / 5.0
vector = randomu(seed, 6) * 100
ticklabels = ['1st', '2nd', '3rd', $
              '4th', '5th', '6th']
plot, time, vector, xtitle='Measurements', $
      ytitle='Signal Strength', title='Experiment 1A', $
      charsize=1.5, xcharsize=1.0, ycharsize=1.0, $
      xtickname=ticklabels, xticks=5, ystyle=1, $
      color=cgcolor('black'), background=cgcolor('white')
```

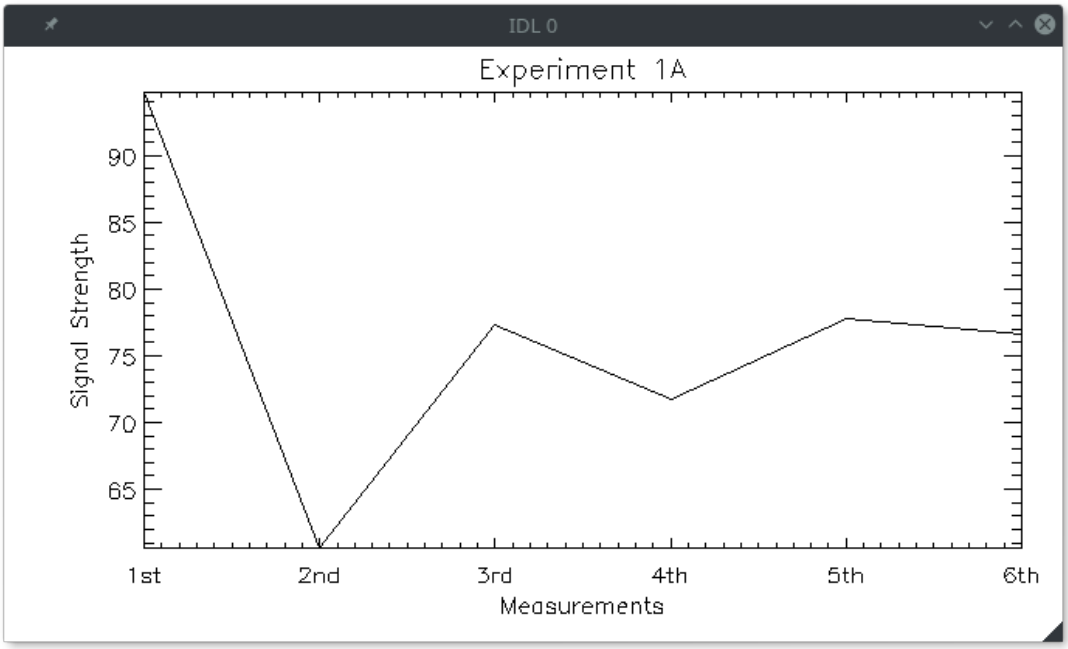

Figure 13: Axis Labels

3. label format

```
plot, time, vector, xtitle='Elapsed Time', $
      ytitle='Signal Strength', title='Experiment 1A', $
      xtickformat='(F0.2)', ystyle=1, color=cgcolor('black'), $
      background=cgcolor('white'), charsize=1.5, $
      xcharsize=1.0, ycharsize=1.0
```

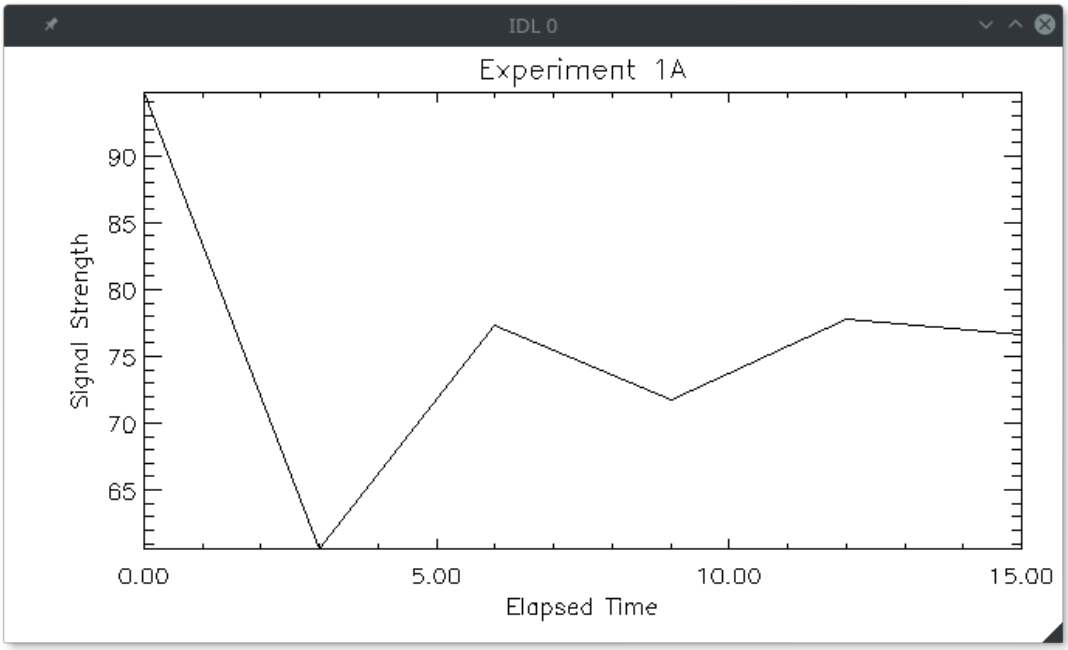

Figure 14: Label Format

4. calendar dates as axis labels

```
vector = randomu(seed, 6) * 10
time = timegen(n_elements(vector), $
       start=julday(7, 1, 2016, 0, 0, 0), $
       units='days')
void = label_date(date_format='%D %M %Y')
plot, time, vector, xtitle='Measurements', $
      ytitle='Signal Strength', title='Experiment 1A', $
      charsize=1.5, xcharsize=1.0, ycharsize=1.0, $
      xtickformat='label_date', xticks=5, xstyle=1, $
      color=cgcolor('black'), background=cgcolor('white'), $
      position=[0.10, 0.15, 0.90, 0.85]
```

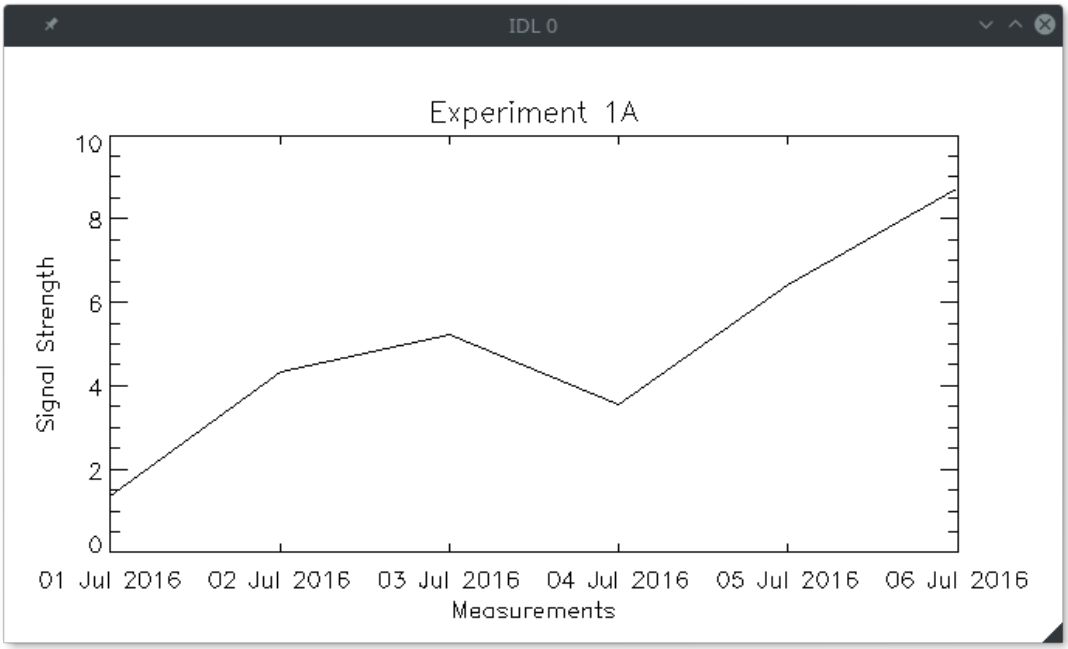

Figure 15: Calendar Labels

8.4 error bar plot

```
data = Congrid(cgDemoData(1), 15)
seed = -5L
time = cgScaleVector(Findgen(N_Elements(data)), 1, 9)
high_yerror = RandomU(seed, N_Elements(data)) * 5 > 0.5
low_yerror = RandomU(seed, N_Elements(data)) * 4 > 0.25
high_xerror = RandomU(seed, N_Elements(data)) * 0.75 > 0.1
low_xerror = RandomU(seed, N_Elements(data))  * 0.75 > 0.1

xtitle = 'Time'
ytitle = 'Signal Strength'
title = 'Error Bar Plot'
position = [0.125, 0.125, 0.9, 0.925]
thick = (!D.Name EQ 'PS') ? 3 : 1

cgDisplay, 600, 500, Title='Errorbar Plot'

```

```
cgPlot, time, data, Color='red5', PSym=-16, SymColor='olive', $
        SymSize=1.0, Thick=thick, Title=title, XTitle=xtitle, $
        YTitle=ytitle, Position=position, YRange=[-5, 35], $
        XRange=[0,10], YStyle=1, $
        ERR_XLow=low_xerror, ERR_XHigh=high_xerror, $
        ERR_YLow=low_yerror, ERR_YHigh=high_yerror, ERR_Color='blu5'
```

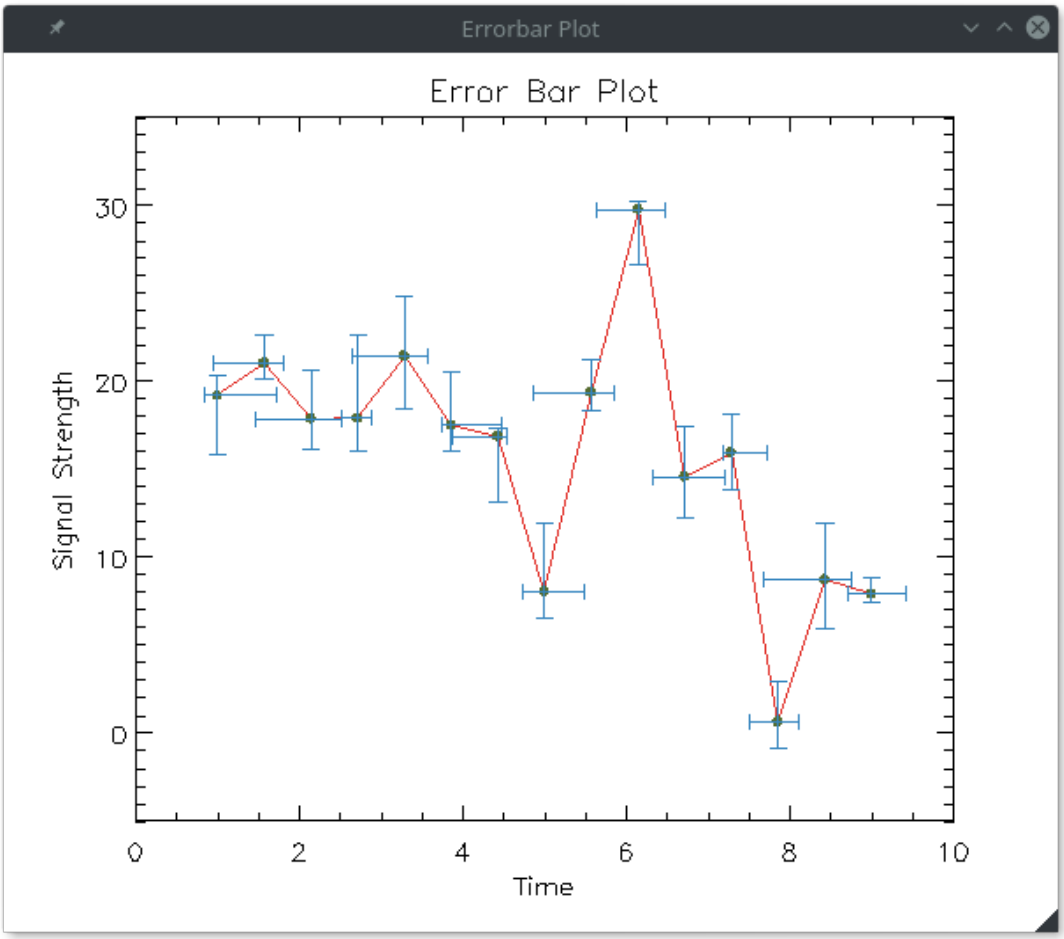

Figure 16: Error Bar Plot

8.5 histogram line plot

```
cgdisplay
image = cgdemodata(7)
h = histogram(image, binsize=5B, /nan, $
              omin=omin, omax=omax)
x = scale_vector(findgen(n_elements(h)), omin, omax)
plot, x, h, psym=10, max_value=15000, xstyle=1, $
      ytitle='Number of Pixels', xtitle='Pixel Value'
```

```
cghistoplot, cgdemodata(7), /fill, max_value=10000, $
             ytitle='Number of Pixels', xtitle='Pixel Value'
```

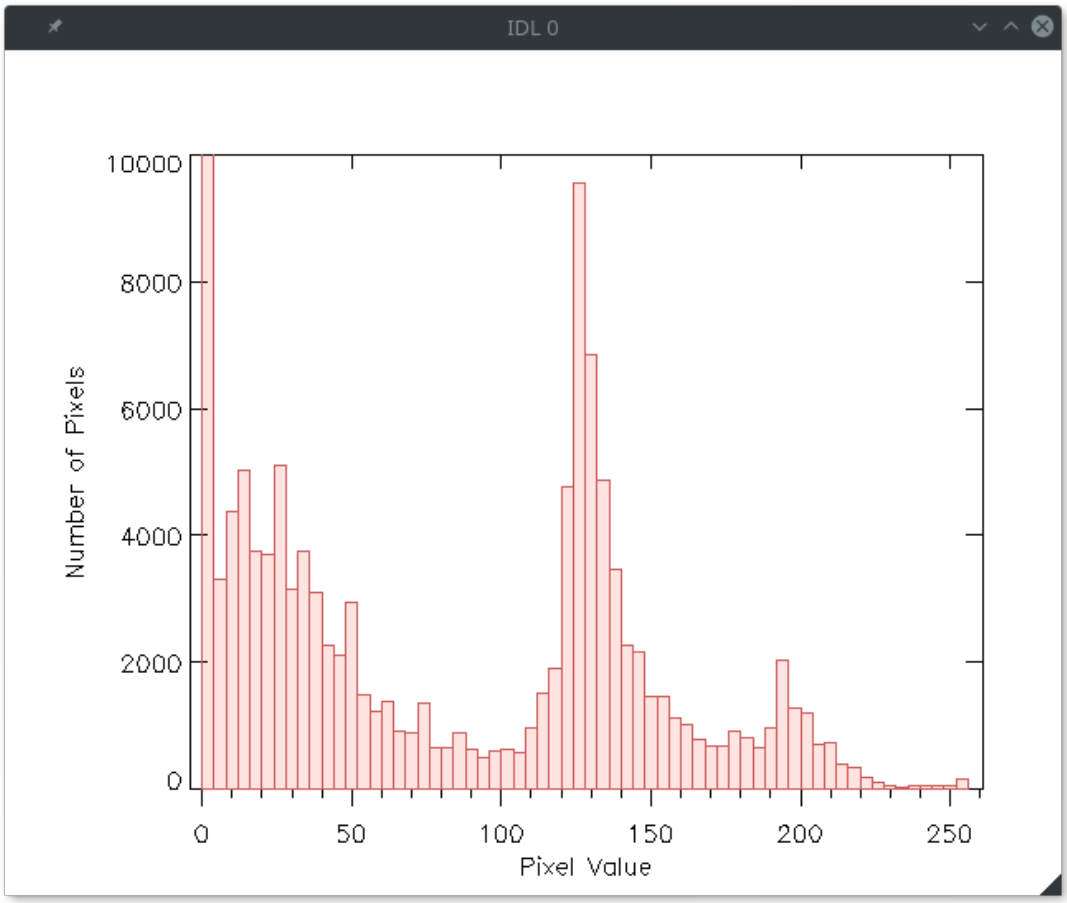

Figure 17: cgHistoplot

```
data = randomu(5L, 200) * 20
!p.multi = [0, 2, 2]
cghistoplot, data, binsize=1.0
cghistoplot, data, binsize=1.0, /fill, $
             polycolor=['charcoal', 'dodger blue']
cghistoplot, data, binsize=1.0, /line_fill, $
             polycolor=['charcoal', 'dodger blue'], $
             orientation=[45, -45]
cghistoplot, data, binsize=1.0, /fill, $
             polycolor='sky blue', mininput=0
moredata = randomu(3L, 80) * 20
cghistoplot, moredata, binsize=1.0, /fill, $
             polycolor='royal blue', /oplot, mininput=0
!p.multi = 0
```

```
cghistoplot, cgdemodata(7), /fill, $
             probcolorname='blu6', /oprobability
```

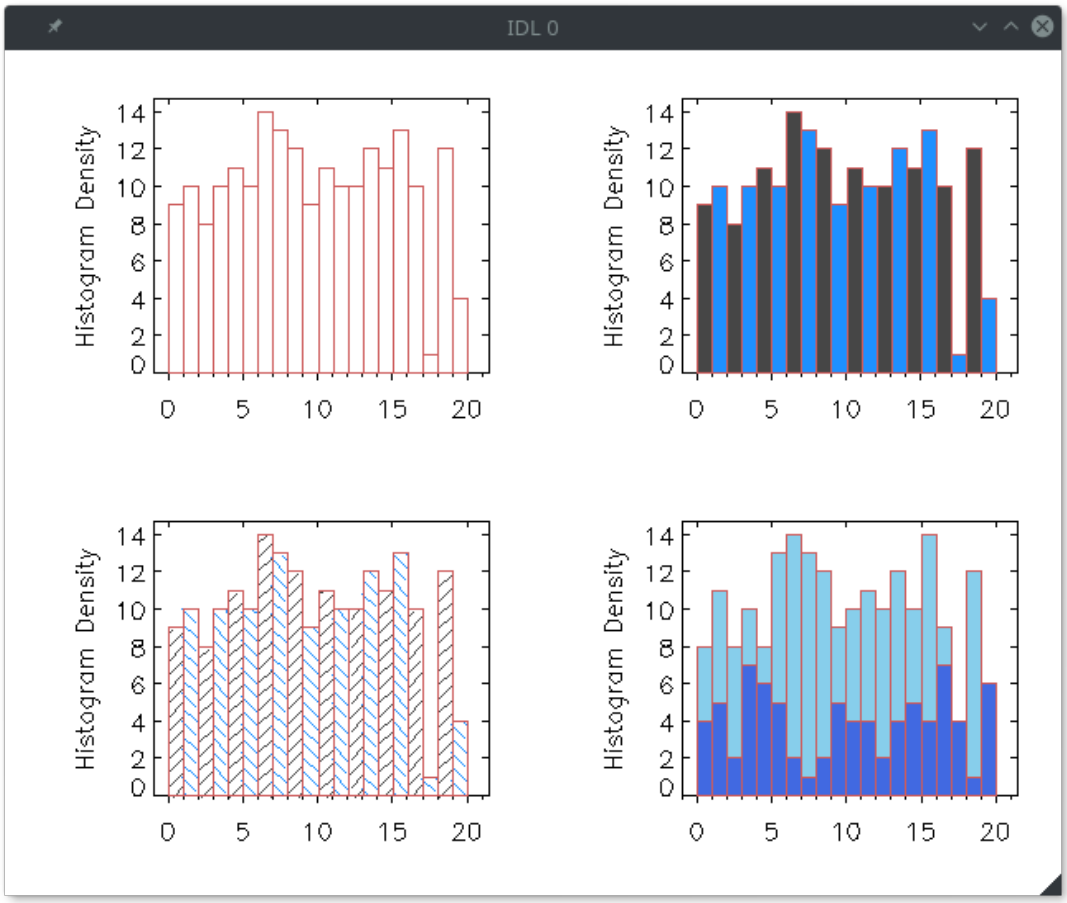

Figure 18: cgHistoplot Multiplot

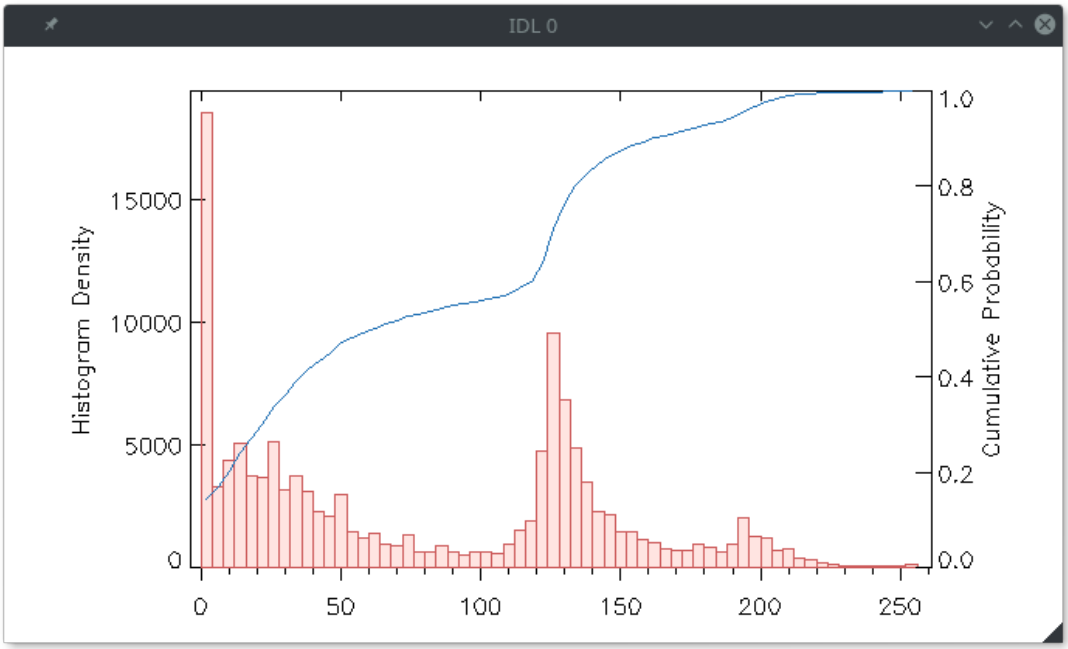

Figure 19: cgHistoplot Cumulative Probability

8.6 missing data

```
data = float(cgdemodata(1))
data[40:49] = !values.f_nan
plot, data
```

8.7 cgplot

```
cgdisplay, 600, 300
cgplot, cgdemodata(1), psym=-16, color='red', $
        symcolor='blue', axiscolor='dark green'
```

```
cgplot, cgdemodata(1), aspect=2/3.0, title='Aspect Plot'
```

```
data = cgdemodata(1)
time = scale_vector(findgen(101), 0, 6)
cgloadct, 33, /silent
cgplot, time, data, /nodata
cgplots, time, data, psym=-16, $
         color=bytscl(data), symcolor=bytscl(data), $
         symsize=scale_vector(data, 1.0, 2.5)
```

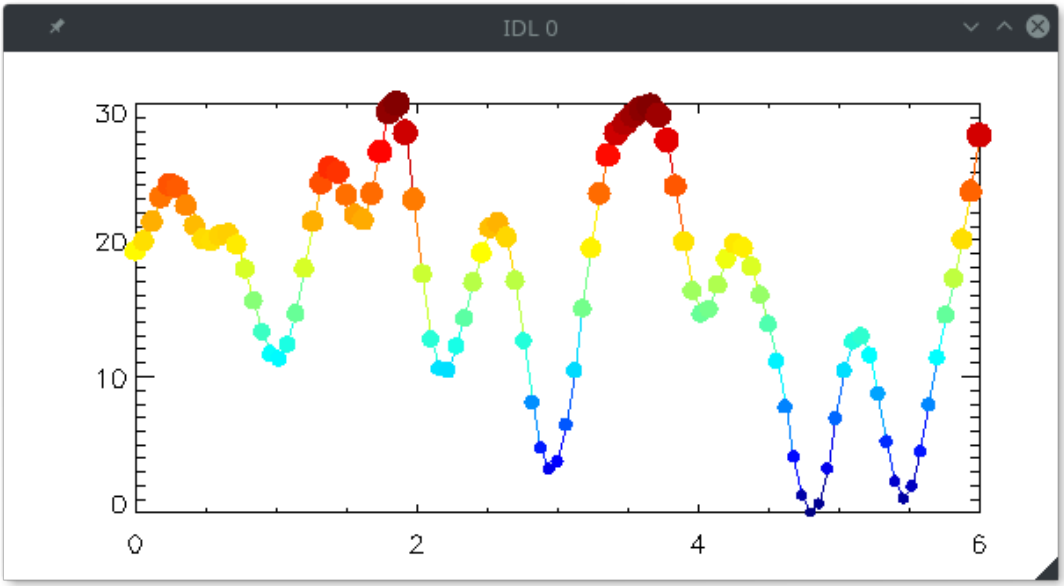

Figure 20: cgPlot

9 Contour

9.1 basic contour

```
peak = cgDemoData(2)
lat = FIndGen(41) * (24. / 40) + 24
lon = FindGen(41) * 50.0 / 40 - 122
Contour, peak, lon, lat, XTitle='Longitude', YTitle='Latitude', $
         xstyle=1, ystyle=1, /follow, color=cgcolor('black'), $
         background=cgcolor('white'), charsize=1.5, c_charsize=1.0
```

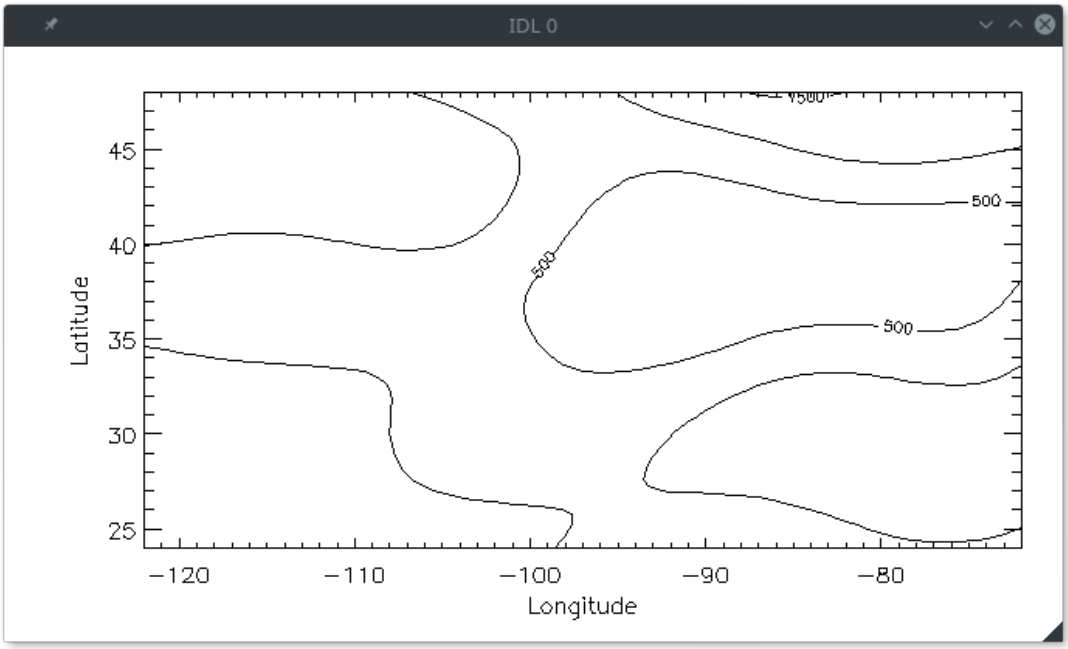

Figure 21: basic contour

9.2 levels

The *NLevels* keyword does not, generally, give you “N contouring levels” as claimed by the IDL documentation.

```
nlevels = 12
step = (max(peak) - min(peak)) / nlevels
vals = indgen(nlevels) * step + min(peak)
contour, peak, lon, lat, xstyle=1, ystyle=1, /follow, levels=vals, $
         color=cgcolor('black'), background=cgcolor('white'), $
         charsize=1.5, c_charsize=1.0
```

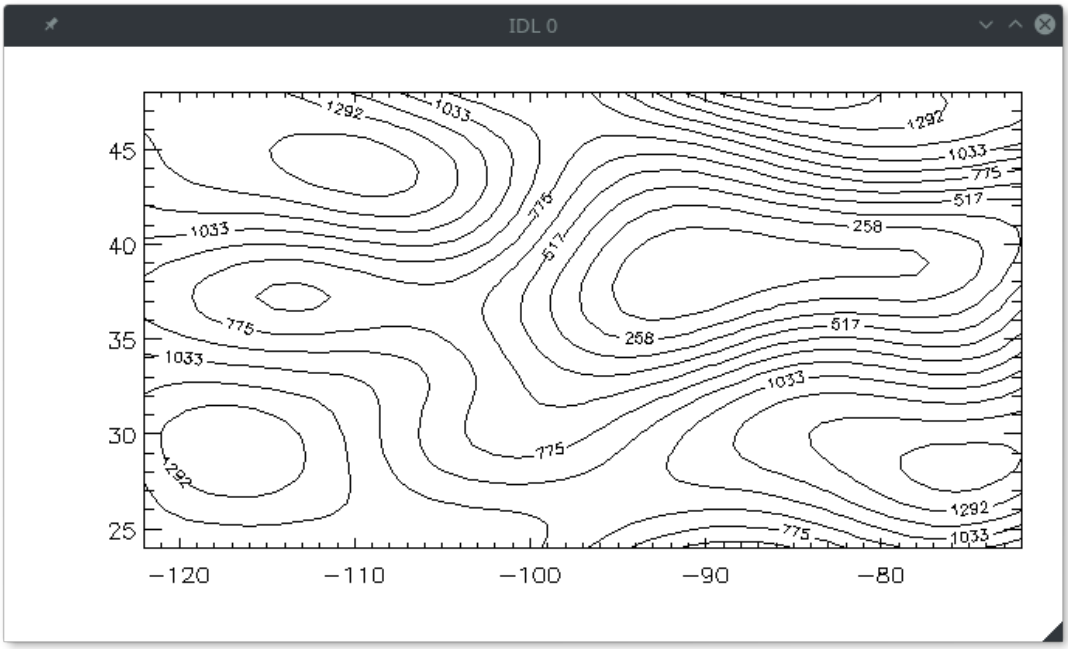

Figure 22: contour levels

9.3 labels

```
threeLevels = [250, 750, 1200]
annotations = ['Low', 'Medium', 'High']
contour, peak, lon, lat, xstyle=1, ystyle=1, /follow, $
         levels = threeLevels, C_annotation=annotations, $
         color=cgcolor('black'), background=cgcolor('white'), $
         charsize=1.5, c_charsize=1.5
```

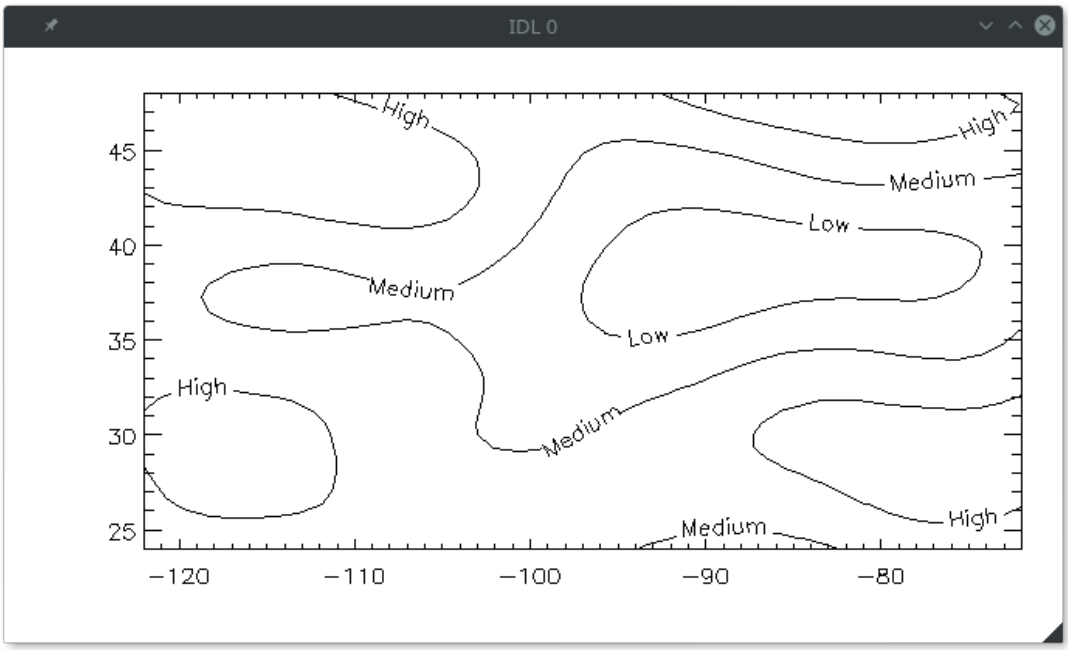

Figure 23: contour labels

9.4 modifying contour lines

9.4.1 *downhill*

We can put small tick marks in the downhill direction of the contour lines

```
nlevels = 12
step = (max(peak) - min(peak)) / nlevels
vals = indgen(nlevels) * step + min(peak)
contour, peak, lon, lat, xstyle=1, ystyle=1, /follow, $
         levels=vals, /downhill, color=cgcolor('black'), $
         background=cgcolor('white'), charsize=1.5, $
         c_charsize=1.0
```

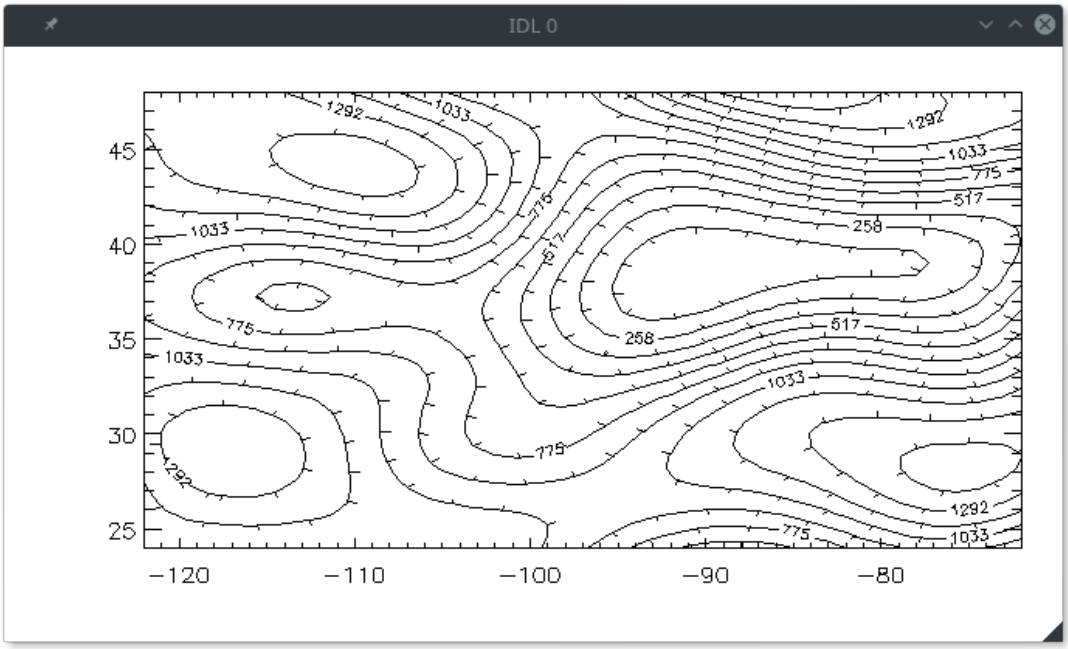

Figure 24: contour downhill

9.4.2 *c_linestyle*

```
contour, peak, lon, lat, xstyle=1, ystyle=1, /follow, $
         levels=vals, C_linestyle=[0,2], color=cgcolor('black'), $
         background=cgcolor('white'), charsize=1.5, c_charsize=1.0
```

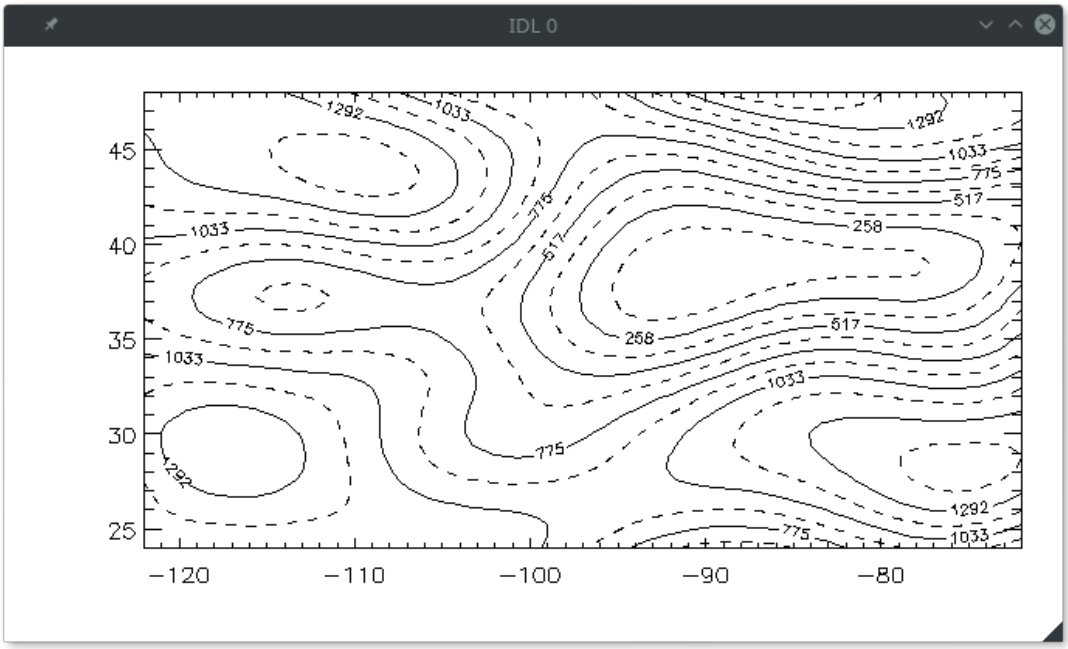

Figure 25: contour linestyle

9.4.3 *c_thick*

```
contour, peak, lon, lat, xstyle=1, ystyle=1, /follow, $
         levels=vals, C_thick=[1,1,2], color=cgcolor('black'), $
         background=cgcolor('white'), charsize=1.5, c_charsize=1.0
```

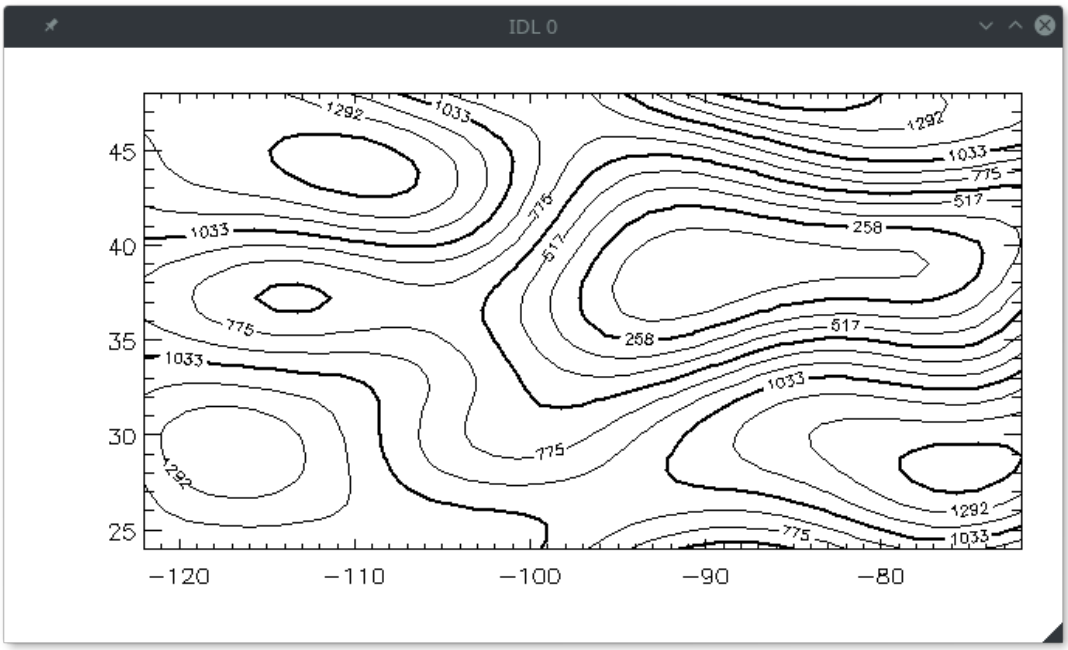

Figure 26: contour thickness

9.4.4 line color

```
contour, peak, lon, lat, xstyle=1, ystyle=1, levels=vals, $
         color=cgcolor('blue'), c_color=cgcolor('orange'), $
         background=cgcolor('white'), /follow, charsize=1.5, $
         c_charsize=1.0
```

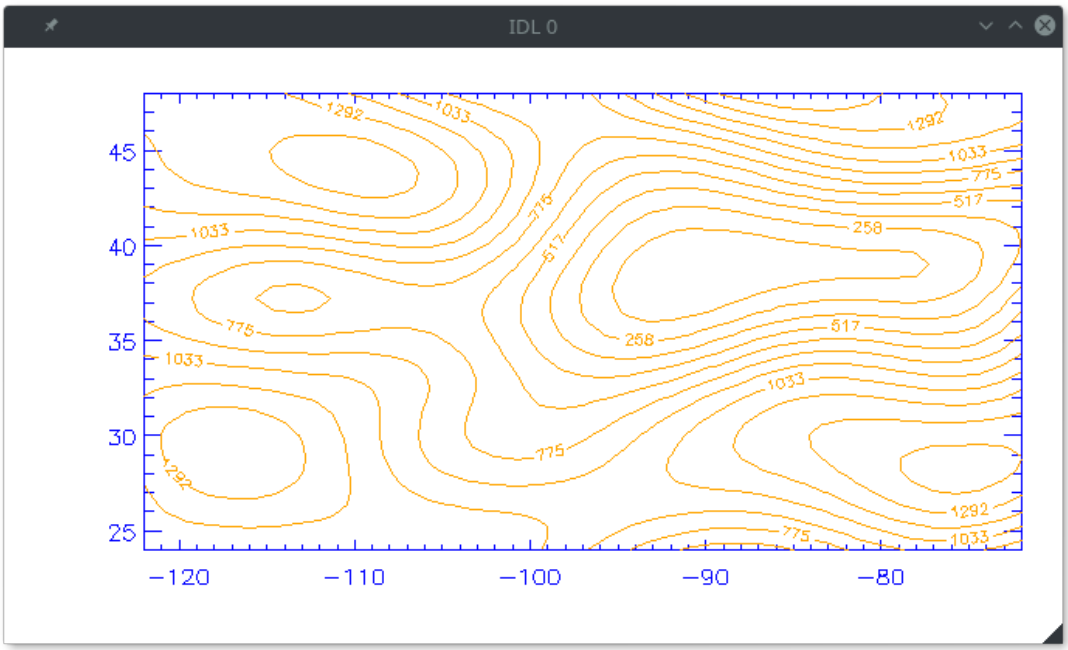

Figure 27: contour color

```
Device, Get_Decomposed=currentMode
Device, Decomposed=0
LoadCT, 33, NColors=12, Bottom=1
Contour, peak, lon, lat, xstyle=1, ystyle=1, Levels=vals, /NoData, $
         Background=cgColor('white'), Color=cgColor('black'), $
         charsize=1.5, c_charsize=1.0
Contour, peak, lon, lat, Levels=vals, /Overplot, $
         C_Colors=IndGen(12) + 1, /follow, charsize=1.5, $
         c_charsize=1.0
Device, Decomposed=currentMode
```

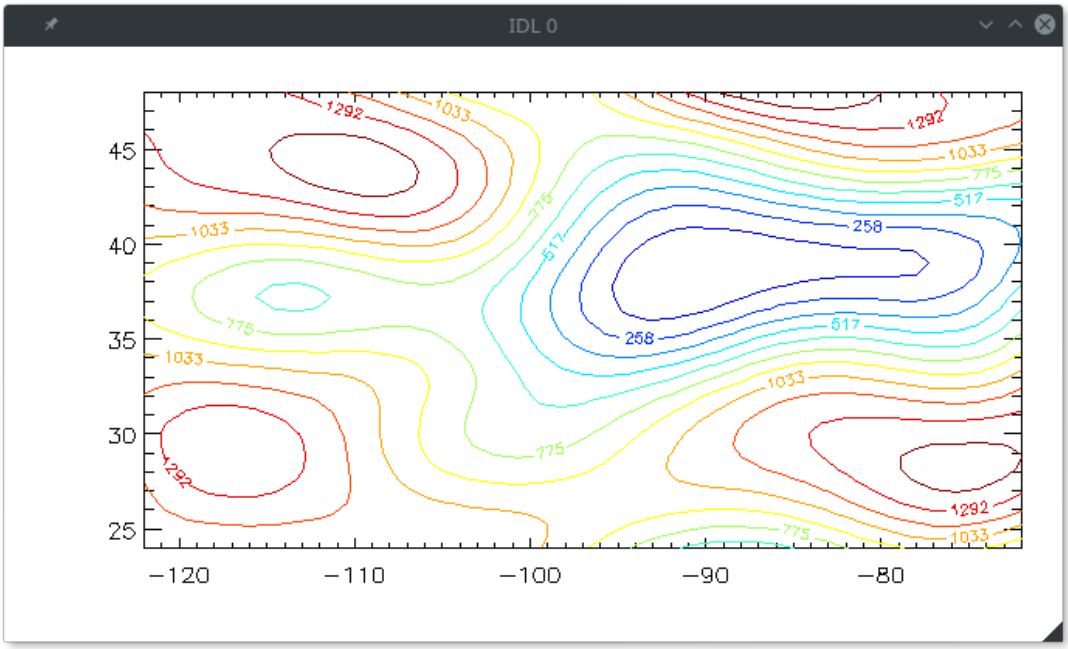

Figure 28: contour more color

9.5 filling color

```
Device, Get_Decomposed=currentMode
Device, Decomposed=0
LoadCT, 33, NColors=12, Bottom=1
Contour, peak, lon, lat, xstyle=1, ystyle=1, Levels=vals, /fill, $
         C_Colors=IndGen(12)+1, Background=cgColor('white'), $
         Color=cgColor('black')
Device, Decomposed=currentMode
```

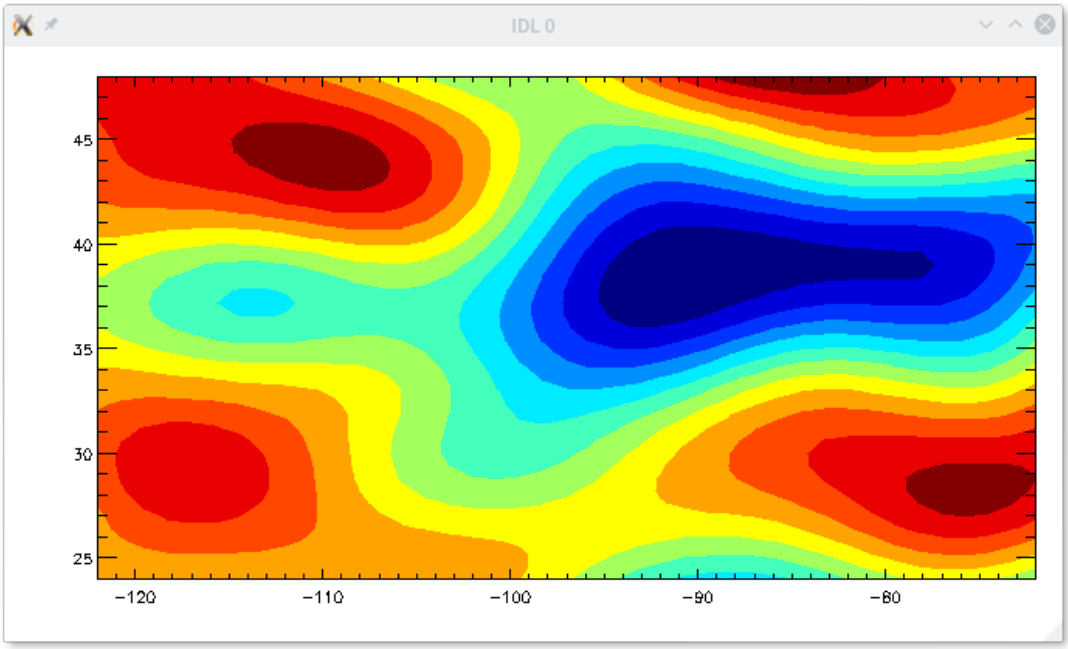

Figure 29: contour filling color

if you want to see contour lines on the color filled image, just use overplot

```
contour, peak, lon, lat, xstyle=1, ystyle=1, /follow, levels=vals, $
         /overplot, c_color=cgColor('white')
```

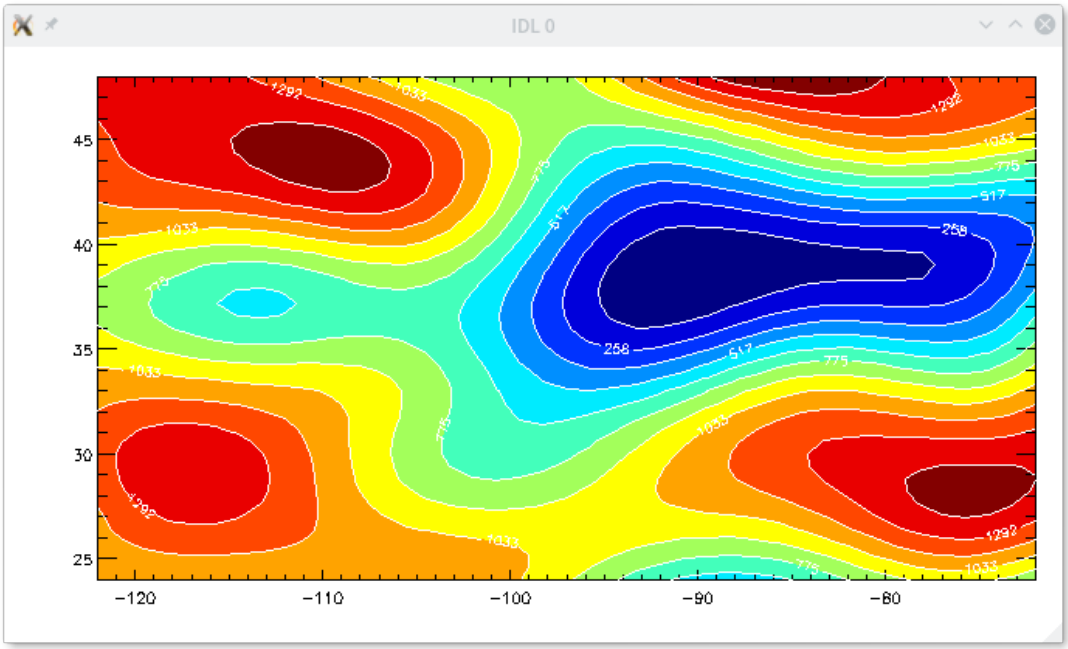

Figure 30: contour filling color overplot

Add a color bar

```
cgsetcolorstate, 0, CurrentState=currentState
LoadCT, 33, NColors=nlevels, Bottom=1
contour, peak, lon, lat, /cell_fill, levels=vals, $
         position=[0.125, 0.125, 0.95, 0.80], $
         background=cgColor('white'), color=cgColor('black'), $
         xstyle=1, ystyle=1, c_colors=indgen(nlevels)+1
contour, peak, lon, lat, /overplot, /follow, $
         color=cgColor('black'), levels=vals
cgsetcolorstate, currentState
cgColorbar, range=[min(peak), max(peak)], $
            divisions=nlevels, xticklen=1, xminor=0, $
            annotatecolor='black', ncolors=nlevels, bottom=1, $
            position=[0.125, 0.915, 0.955, 0.95], charsize=0.75
```

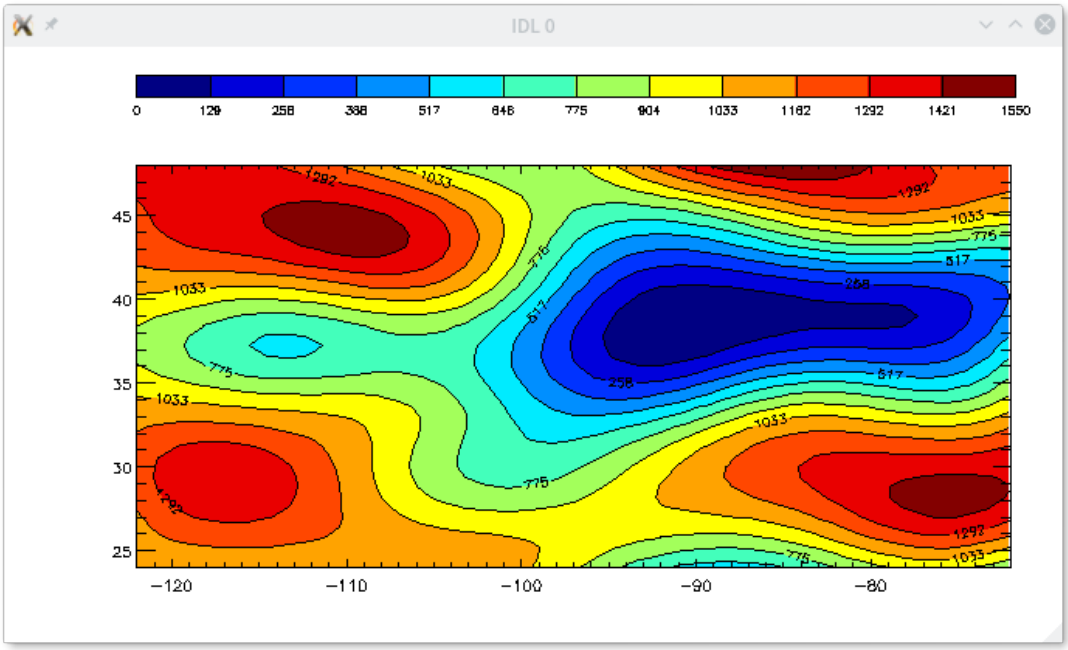

Figure 31: contour color bar

9.6 cgContour

```
data = cgdemodata(2)
loadct, 0, /silent
window
cgcontour, data, title='Normal Contour Plot', $
           xtitle='X Title', ytitle='Y Title'
```

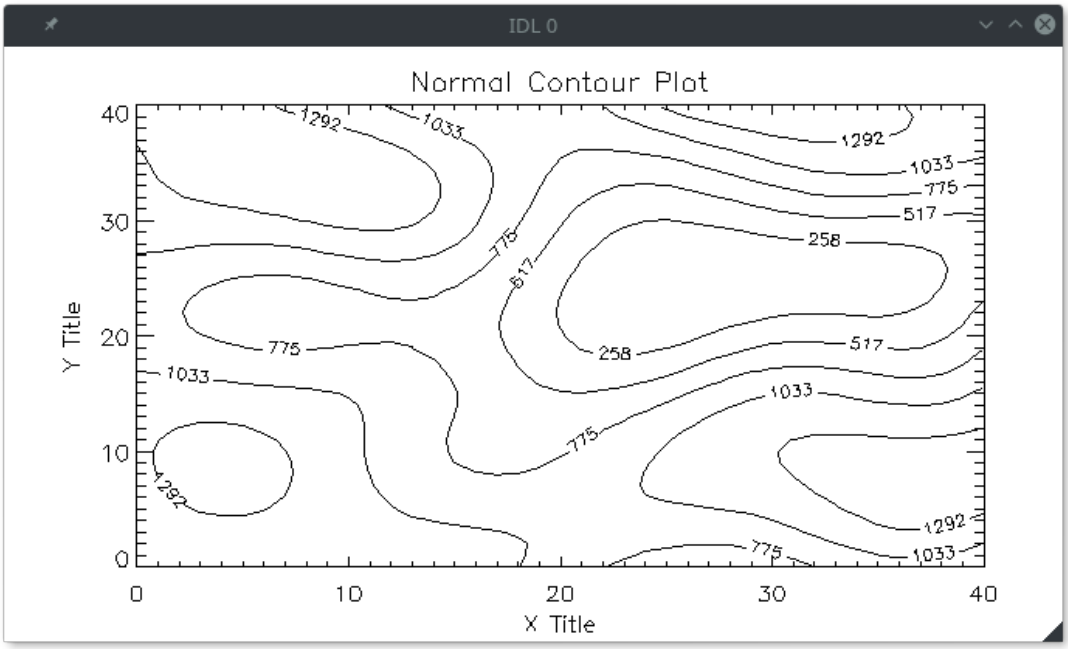

Figure 32: cgContour Normal Plot

```
cgloadct, 17, /brewer
cgloadct, 4, ncolors=10, bottom=1, /brewer, /reverse
c_colors = indgen(10) + 1
position = [0.1, 0.1, 0.9, 0.8]
cgcontour, data, nlevels=10, /fill, $
           position=position, c_colors=c_colors
cgcontour, data, nlevels=10, /overplot
cgcolorbar, divisions=10, ncolors=10, bottom=1, $
            range=[min(data), max(data)], ticklen=1.0, $
            position=[0.1, 0.90, 0.90, 0.94], charsize=0.75
```

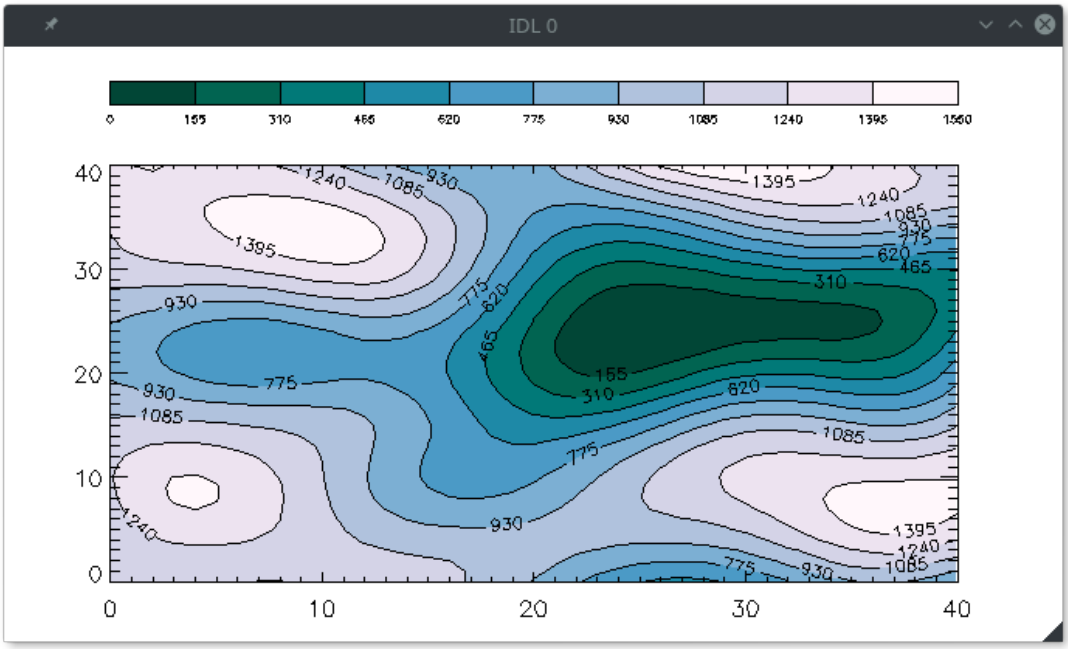

Figure 33: cgContour Filled Contour

Table 24: cgContour Labels

Label keyword	Meaning
1 (default)	labels every contour level
2	labels every other contour
3	labels every third contour
0	no contour labeling

```
window, 0, xsize=400, ysize=400
cgcontour, data, nlevels=10, axiscolor='blue', $
           color='red'
window, 1, xsize=400, ysize=400
cgcontour, data, nlevels=5, axiscolor='brown', $
           c_colors=['aquamarine', 'dark green', 'orange', $
                     'crimson', 'purple']
```

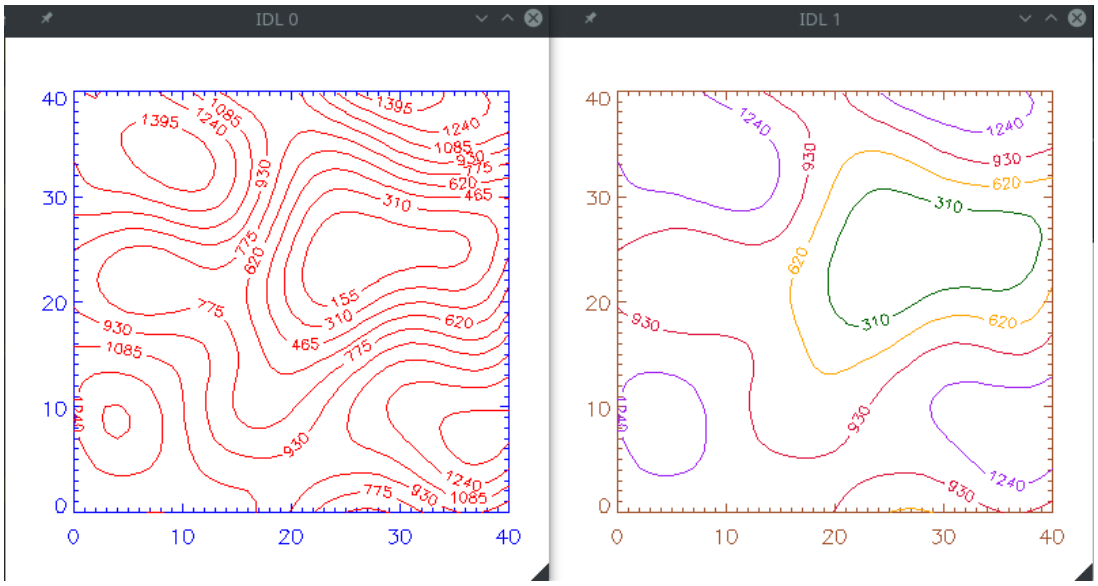

Figure 34: cgContour Colors

10 Surface

10.1 basic

Surface of 2D data can be easily made using 'surface' command

```
peak=cgdemodata(2)
lon=findgen(41)*50./41-122
lat=findgen(41)*(24./40)+24
surface,peak,lon,lat,xtitle='longitude',$
        ytitle='latitude',ztitle='elevation', charsize=2.5, $
        color=cgcolor('black'), background=cgcolor('white'), $
        xstyle=1, ystyle=1, zstyle=0
```

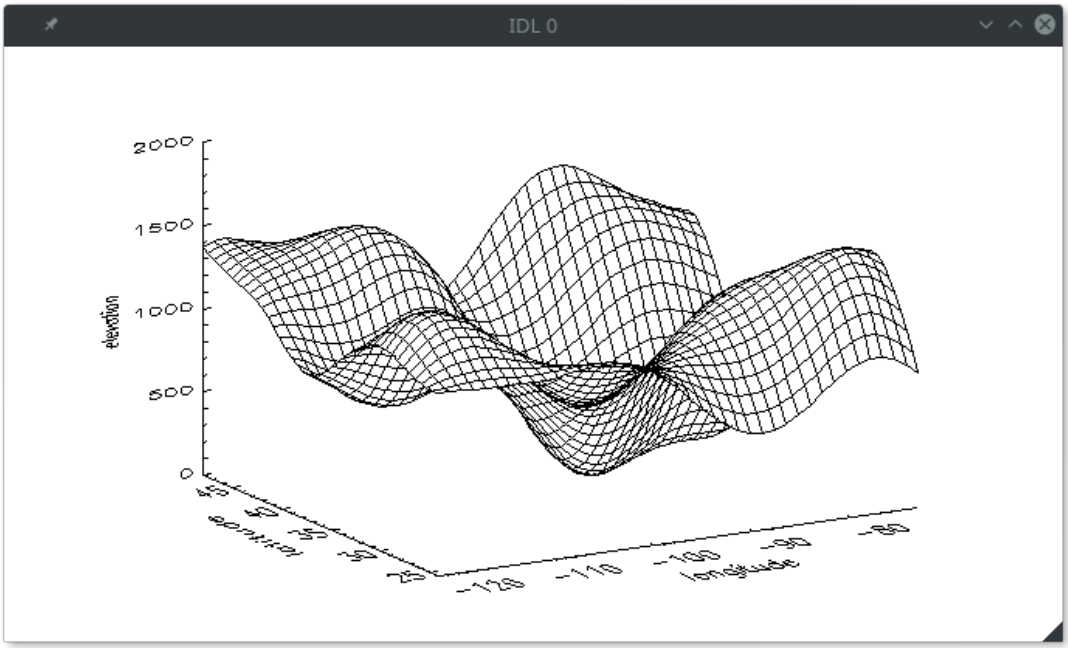

Figure 35: surface

```
cgsurface,peak,lon,lat,xtitle='longitude',$
          ytitle='latitude',ztitle='elevation', $
          xstyle=1, ystyle=1, zstyle=0
```

10.2 title

We usually use *xyouts* to make title of the surface

```
xyouts,0.5,0.90,/normal,size=2.0,'Surface',align=0.5, $
       color=cgcolor('black')
```

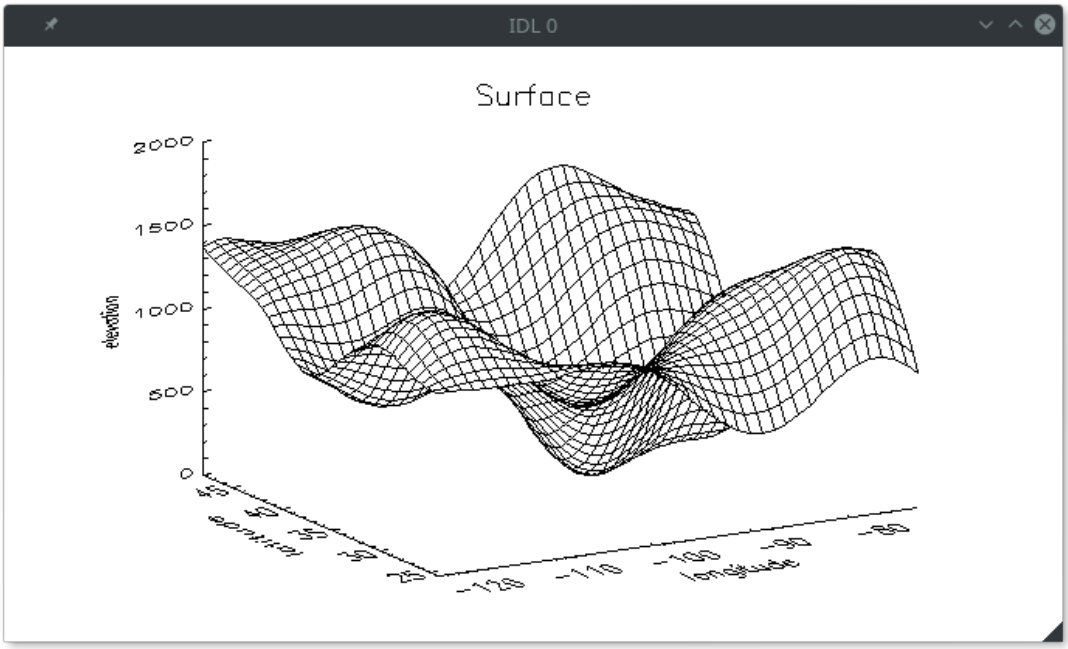

Figure 36: xyouts

10.3 rotating

The default value of *A[xz]* is 30

```
surface,peak,lon,lat,Ax=60,Az=60,charsize=2.5, $
        color=cgcolor('black'), background=cgcolor('white')
```

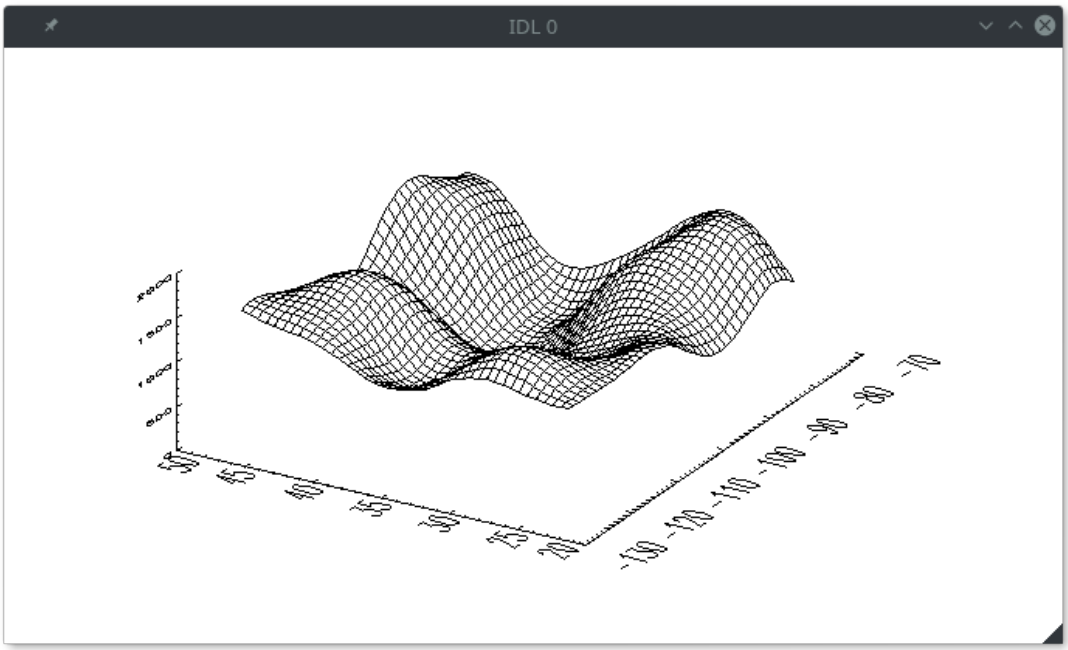

Figure 37: surface rotating

10.4 with contour

```
Surface, peak, lon, lat, charsize=2.5, $
         Color=cgColor('black'), $
         Background=cgColor('white'), $
         XStyle=5, YStyle=5, ZStyle=0, $
         AX=45, AZ=60, /Save
Contour, peak, lon, lat, /T3D, XStyle=1, charsize=2.5, $
         YStyle=1, XTitle=xtitle, YTitle=ytitle, $
         /NoErase, Color=cgColor('black'), /NoData
Contour, peak, lon, lat, /T3D, /Overplot, charsize=2.5, $
         NLevels=12, C_Label=Replicate(1,12), /NoClip, $
         ZValue=0.0, /Downhill, Color=cgColor('blu6')
```

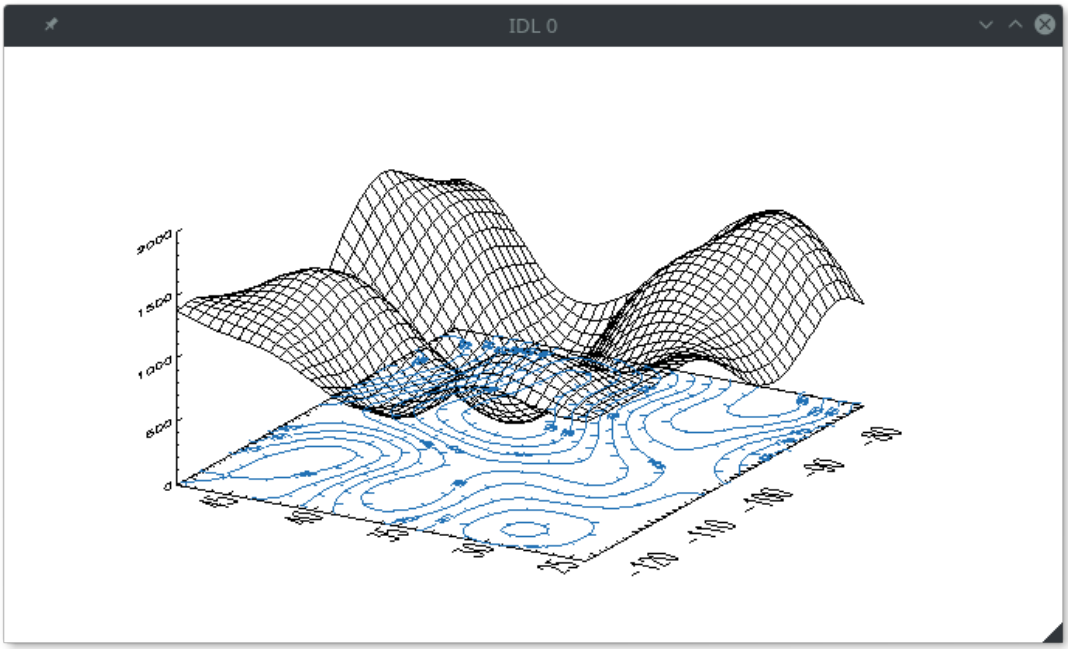

Figure 38: Surface with Contour

10.5 skirt

```
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('black'), $
         background=cgcolor('white'), $
         xstyle=1, ystyle=1, zstyle=0, $
         ax=30, az=60, skirt=0
```

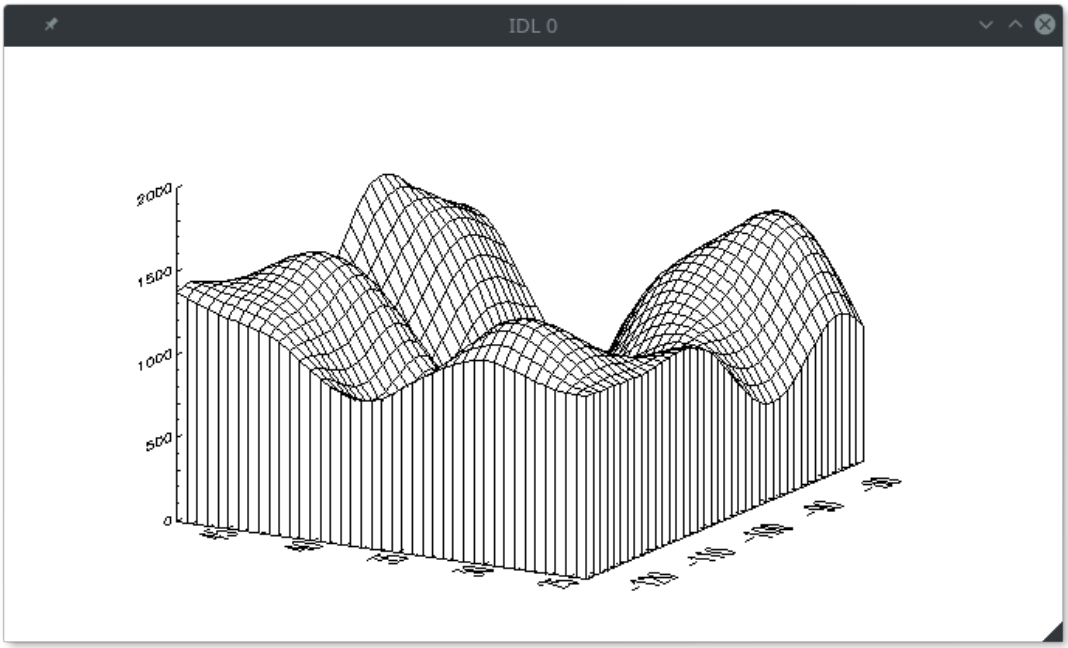

Figure 39: Surface Skirt

10.6 color

```
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('black'), $
         background=cgcolor('white'), $
         xstyle=1, ystyle=1, zstyle=0, /nodata
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('grn5'), /noerase, $
         xstyle=5, ystyle=5, zstyle=4
```

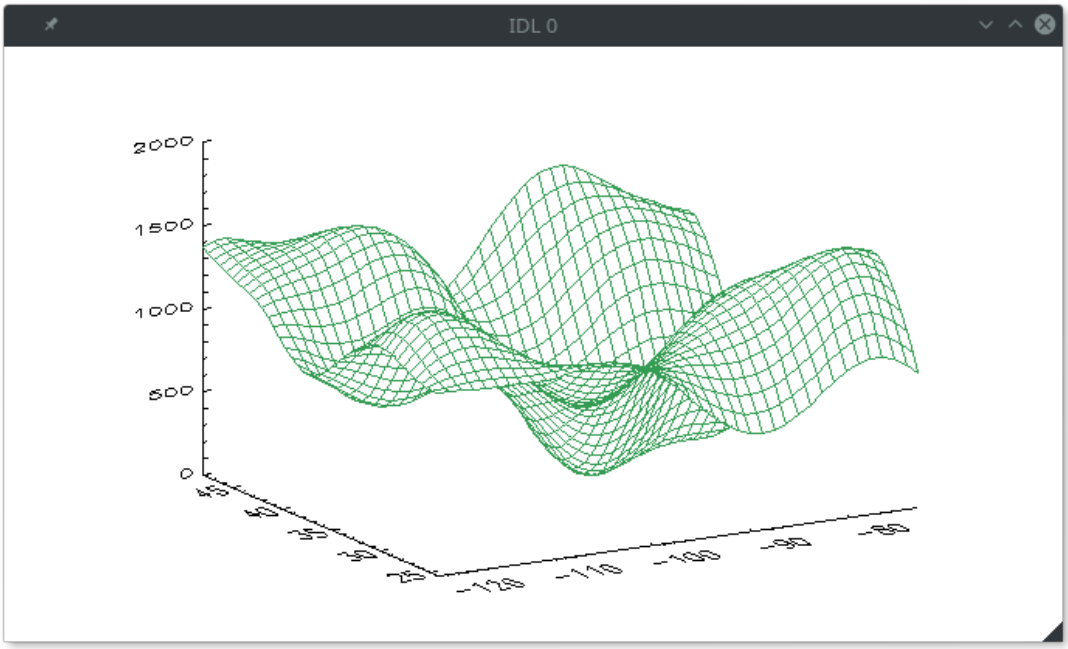

Figure 40: surface color

```
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('black'), $
         background=cgcolor('white'), $
         xstyle=1, ystyle=1, zstyle=0
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('grn5'), /noerase, $
         xstyle=5, ystyle=5, zstyle=4, $
         bottom=cgcolor('red5')
```

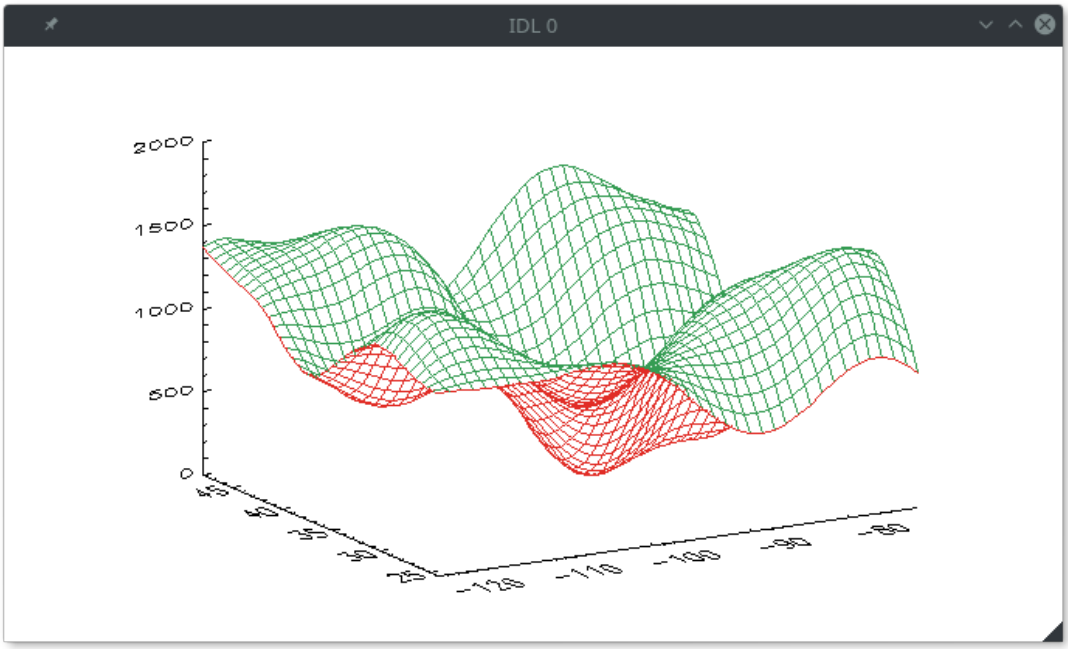

Figure 41: Surface Bottom Color

```
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('black'), $
         background=cgcolor('white'), $
         xstyle=1, ystyle=1, zstyle=0
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('red5'), /noerase, $
         xstyle=5, ystyle=5, zstyle=4, /skirt
surface, peak, lon, lat, charsize=2.5, $
         color=cgcolor('grn5'), /noerase, $
         xstyle=5, ystyle=5, zstyle=4
```

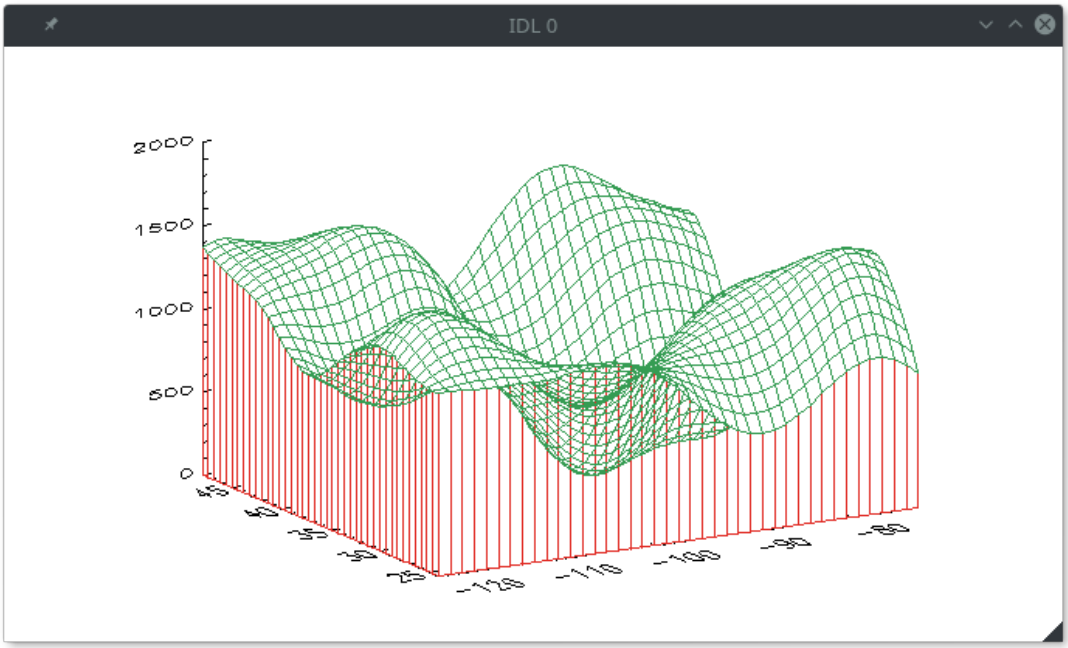

Figure 42: Surface Skirt Color

```
snowdepth = cgdemodata(3)
cgsetcolorstate, 0, currentstate=currentstate
set_shading, values=[0, 249]
loadct, 33
surface, peak, lon, lat, charsize=2.0, $
         color=cgcolor('black', 250), $
         background=cgcolor('white', 251), $
         xstyle=1, ystyle=1, zstyle=0, $
         shades=bytscl(snowdepth, top=249)
cgsetcolorstate, currentstate
cgcolorbar, ncolors=250, divisions=8, $
            range=[min(peak), max(peak)], $
            xminor=0, format='(I0)', $
            annotatecolor='black', xtitle='', $
            position=[0.2, 0.92, 0.8, 0.95], $
            title='Pressure Units'
```

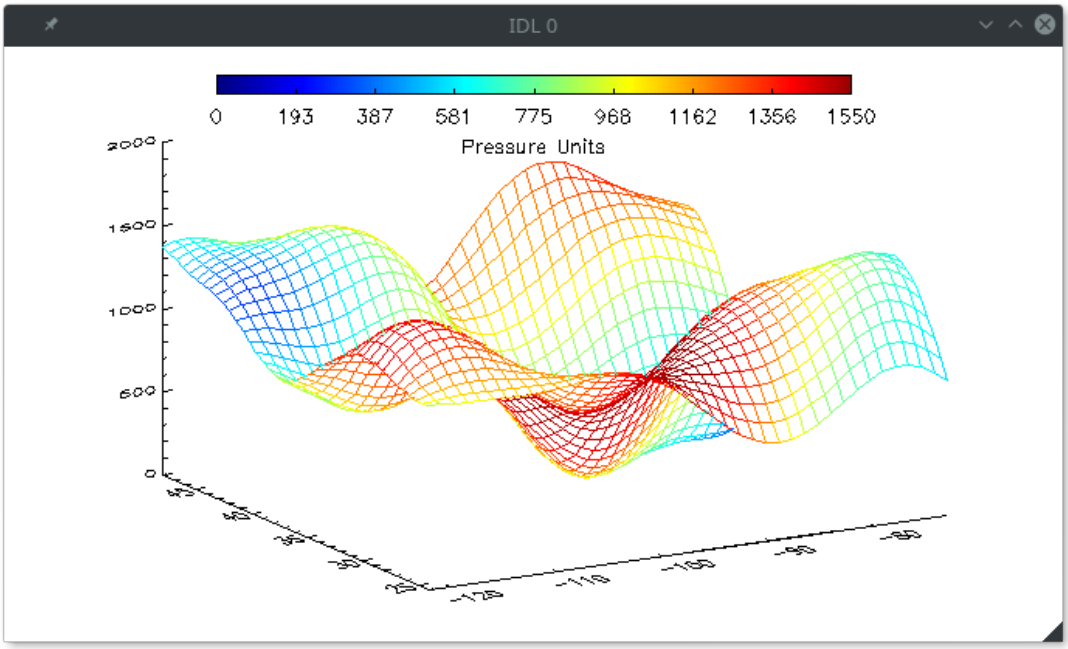

Figure 43: Surface Shading with a Second Data Set

10.7 shaded surface

```
shade_surf,peak,lon,lat,Az=45, charsize=2.0, $
           color=cgcolor('black'), $
           background=cgcolor('white')
```

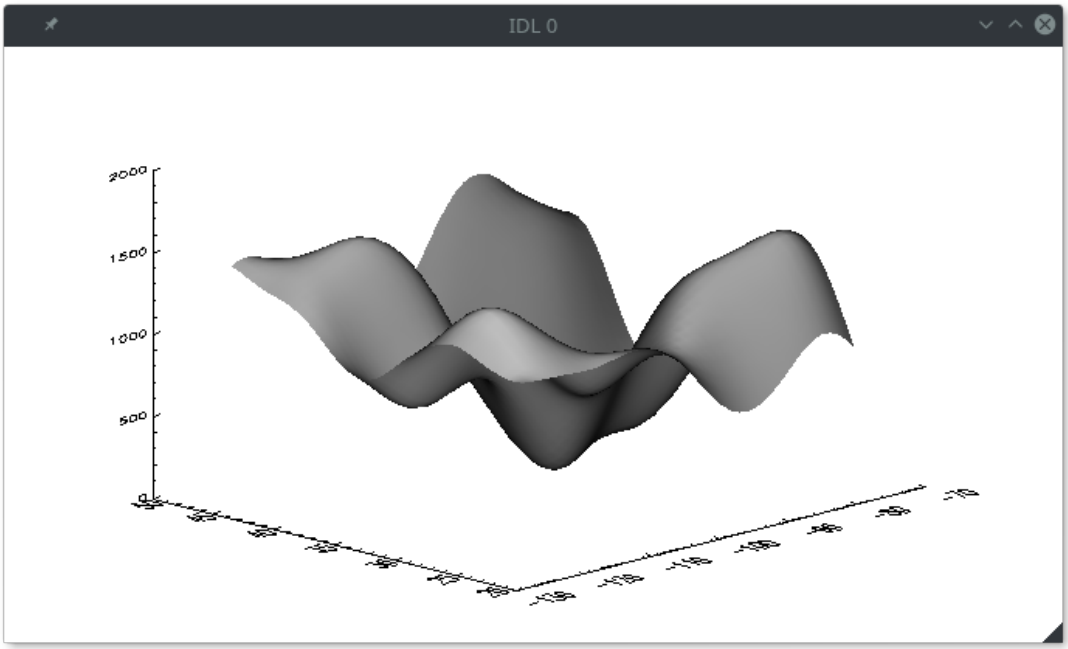

Figure 44: shaded surface

```
cgsetcolorstate, 0, currentstate=currentstate
set_shading, values=[0, 249], light=[0.0, 1.0, 0.5]
```

```
cgloadct, 5, /brewer, /reverse, ncolors=250
shade_surf, peak, lon, lat, charsize=2.0, $
            color=cgcolor('black', 250), $
            background=cgcolor('white', 251), $
            xstyle=1, ystyle=1, zstyle=0
cgsetcolorstate, currentstate
```

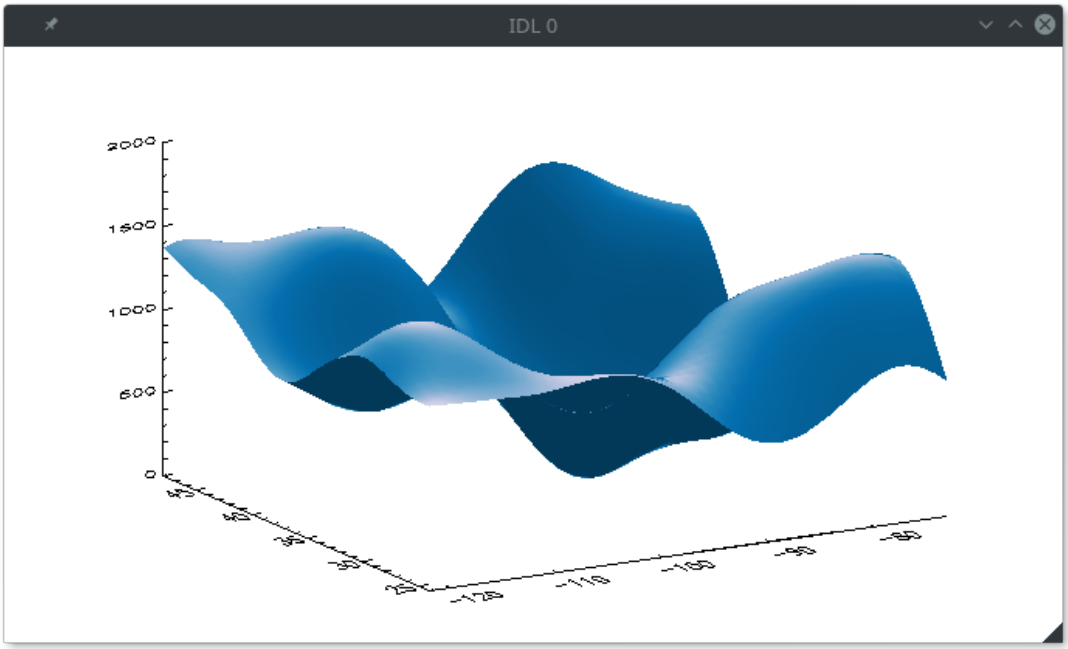

Figure 45: shaded surface with color and different light position

```
cgsetcolorstate, 0, currentstate=currentstate
cgloadct, 5, /brewer, /reverse, ncolors=250
shade_surf, peak, lon, lat, charsize=2.0, $
            color=cgcolor('black', 250), $
            background=cgcolor('white', 251), $
            xstyle=1, ystyle=1, zstyle=0, $
            shades=bytscl(peak, top=249)
cgloadct, 33, ncolors=250
surface, peak, lon, lat, charsize=2.0, $
         color=cgcolor('black', 250), $
         background=cgcolor('white', 251), $
         xstyle=5, ystyle=5, zstyle=4, /noerase, $
         shades=bytscl(snowdepth, top=249)
cgsetcolorstate, currentstate
```

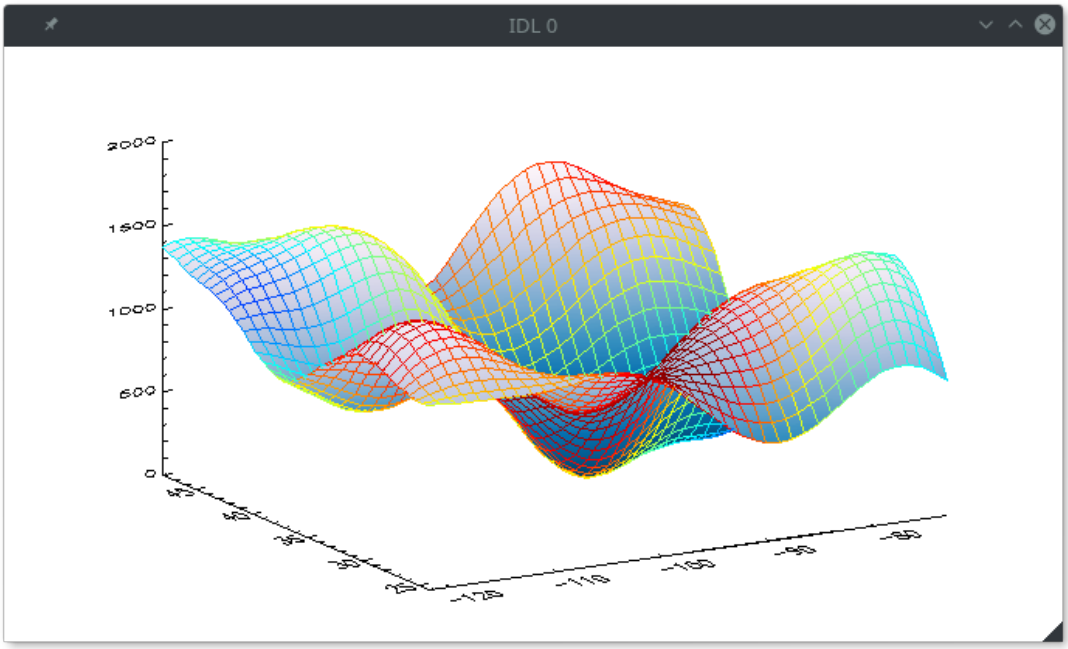

Figure 46: shaded surface with a wire mesh surface

10.8 3d scatter plot

```
;; random data
data2d = cgdemodata(2)
s = size(data2d, /dimensions)
lon = (indgen(s[0]) * 25000L - 500000L) / 1000
lat = (indgen(s[1]) * 25000L - 500000L) / 1000
lat2d = rebin(reform(lat, 1, 41), 41, 41)
lon2d = rebin(lon, 41, 41)
pts = round(randomu(-3L, 200) * 41 * 41)
datairr = data2d[pts]
lonirr = lon2d[pts] + random(5L, 200) * 50 - 25
latirr = lat2d[pts] + random(8L, 200) * 50 - 25
surface, dist(100), xstyle=1, ystyle=1, /nodata, $
         xrange=[-550, 550], yrange=[-550, 550], $
         zrange=[0, 1500], color=cgcolor('black'), $
         background=cgcolor('white'), charsize=2.0, $
         position=[0.075, 0.075, 0.925, 0.750], /save
;; add axes
axis, xaxis=1, /t3d, xstyle=1, charsize=2.0, $
      color=cgcolor('black')
axis, yaxis=1, /t3d, ystyle=1, charsize=2.0, $
      color=cgcolor('black')
axis, zaxis=0, /t3d, 550, 550, charsize=2.0, $
      color=cgcolor('black')
axis, zaxis=0, /t3d, 550, -550, charsize=2.0, $
      color=cgcolor('black')
;; load color table
symcolors = fix(bytscl(datairr))
loadct, 33, /silent
;; draw light gray lines
num = n_elements(datairr)
for j=0, num-1 do plots, [lonirr[j], lonirr[j]], $
```

```
                          [latirr[j], latirr[j]], $
                          [datairr[j], 0], $
                          color=cgcolor('light gray'), /t3d
;; draw a dot at the bottom of the light gray lines
for j=0, num-1 do plots, lonirr[j], latirr[j], 0, $
                          /t3d, psym=3, symsize=2.0, $
                          color=cgcolor('black')
;; draw a filled circle symbol in color at the top of the
;; light gray lines
for j=0, num-1 do plots, lonirr[j], latirr[j], $
                          datairr[j], /t3d, psym=symcat(16), $
                          symsize=4.0, $
                          color=cgcolor(strtrim(symcolors[j], 2))
;; color bar
loadct, 33
cgcolorbar, divisions=6, format='(I0)', $
            annotatecolor='black', charsize=1.0, $
            position=[0.2, 0.88, 0.8, 0.92]
```

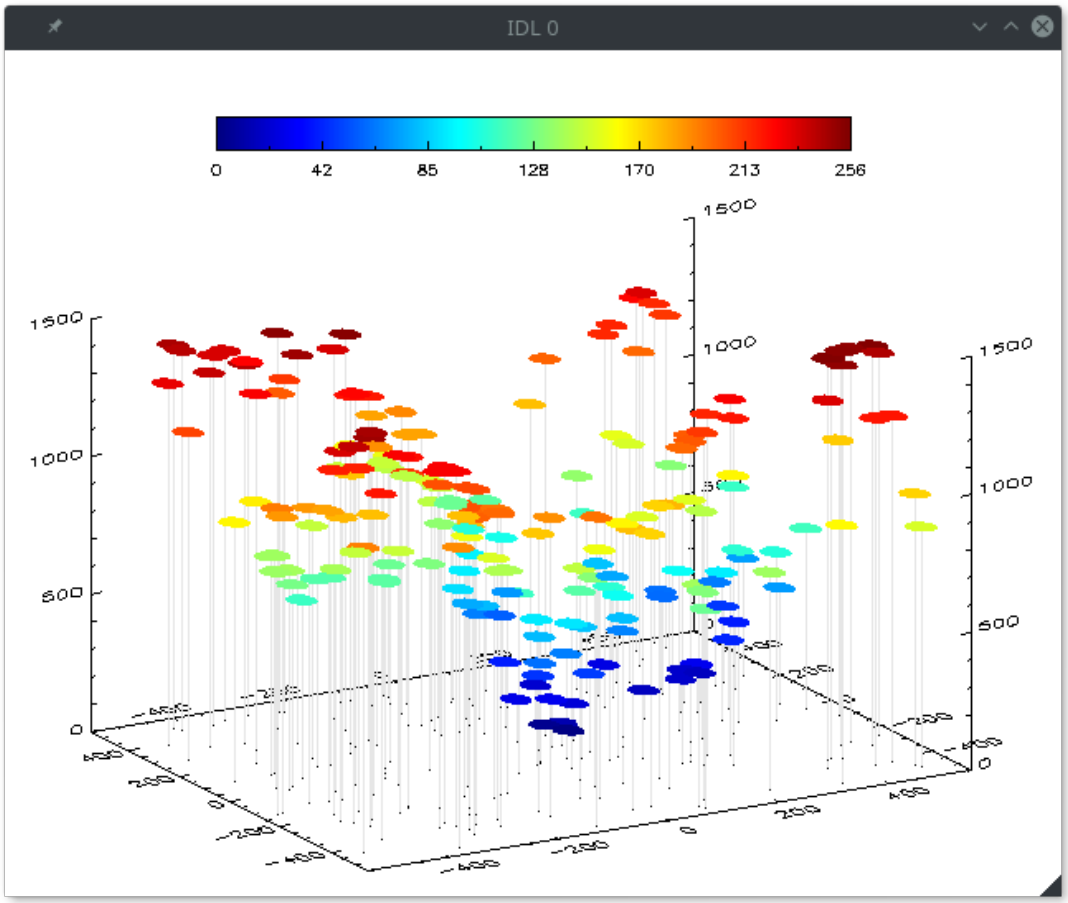

Figure 47: 3D Scatter Plot

10.9 cgSurf

```
cgsurf, cgdemodata(3), color='red', bottom='blue', $
        title='Colorful Surface', tsize=2.0
```

```
cgloadct, 5, /brewer, /reverse
cgsurf, data2d, lon, lat, charsize=2.0, $
        xstyle=1, ystyle=1, zstyle=0, $
        shades=bytscl(data2d), /shaded
cgloadct, 33
cgsurf, data2d, lon, lat, charsize=2.0, $
        xstyle=5, ystyle=5, zstyle=4, /noerase, $
        shades=bytscl(snowdepth)
```

```
cgloadct, 5, rgb_table=palette
cgsurf, cgdemodata(2), palette=palette, /elevation, $
        charsize=2.0
```

```
cgwindow, wxsize=800, wysize=400
cgloadct, 33, /window
cgsurf, data2d, /elevation, layout=[2, 1, 1], $
        /shaded, /addcmd
cgcontour, data2d, nlevels=12, /fill, $
           layout=[2, 1, 2], /addcmd
cgcontour, data2d, nlevels=12, color='charcoal', $
           layout=[2, 1, 2], /addcmd
```

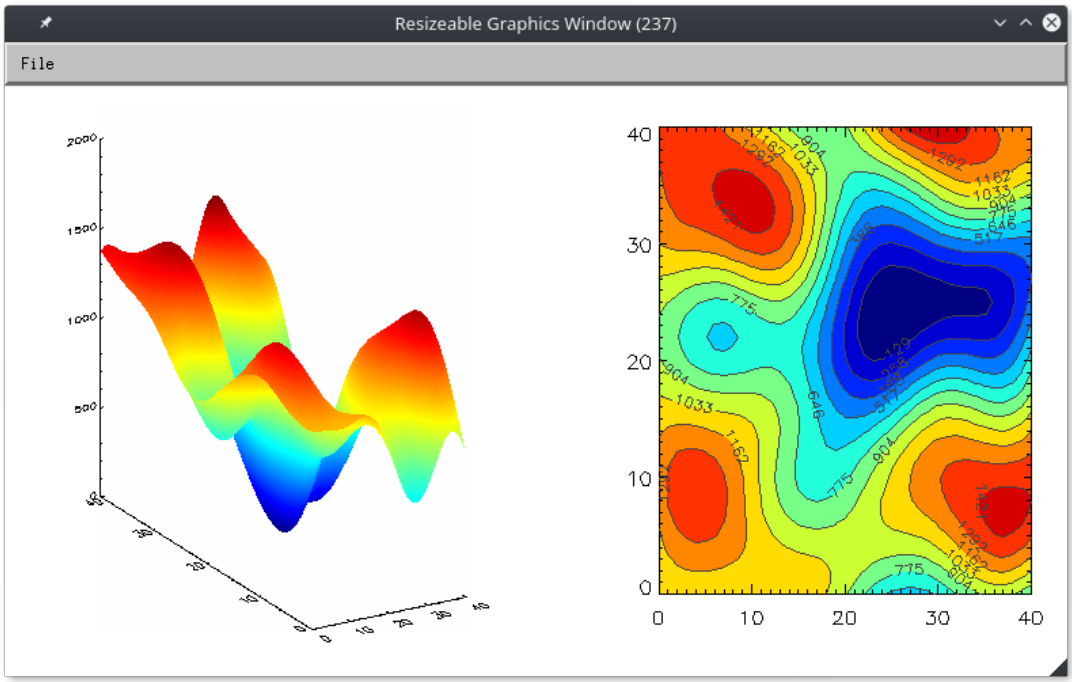

Figure 48: cgSurf Multiplot

10.10 cgSurface

```
cgsurface, data2d, lon, lat, $
           xtitle='Longitude', ytitle='Latitude', $
           ztitle='Atmosphere Pressure'
cgsurface, data2d, lon, lat, $
           xtitle='Longitude', ytitle='Latitude', $
           ztitle='Atmosphere Pressure', /shaded
```

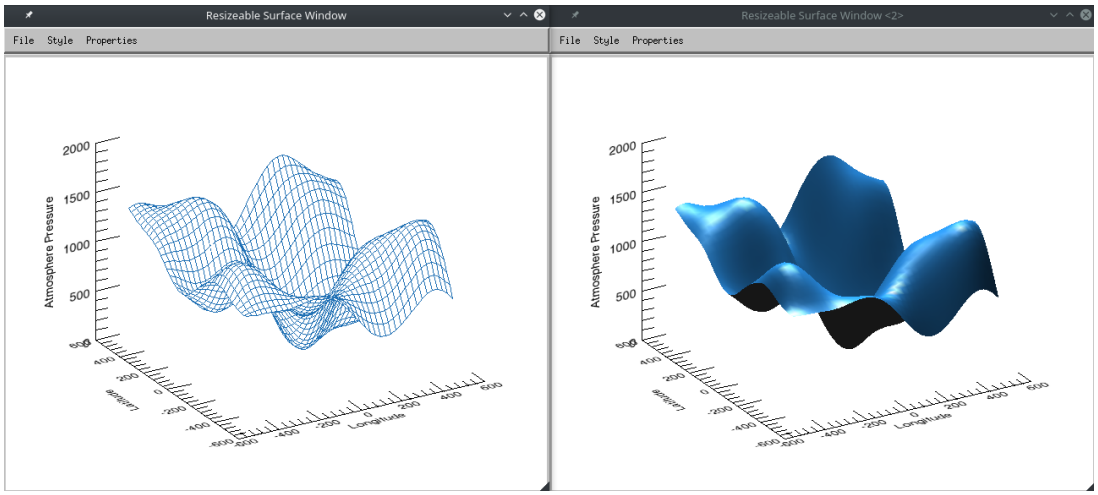

Figure 49: cgSurface Basic

```
earth = cgdemodata(7)
cgsurface, data2d, lon, lat, zscale=0.5, $
           xtitle='Longitude', ytitle='Latitude', $
           ztitle='Atmosphere Pressure', $
           texture_image=earth, ctable=4, /brewer, $
           title='2D Image as Texture'
rose = cgdemodata(16)
cgsurface, data2d, lon, lat, zscale=0.5, $
           xtitle='Longitude', ytitle='Latitude', $
           ztitle='Atmosphere Pressure', $
           texture_image=rose, $
           title='True-Color Image as Texture'
```

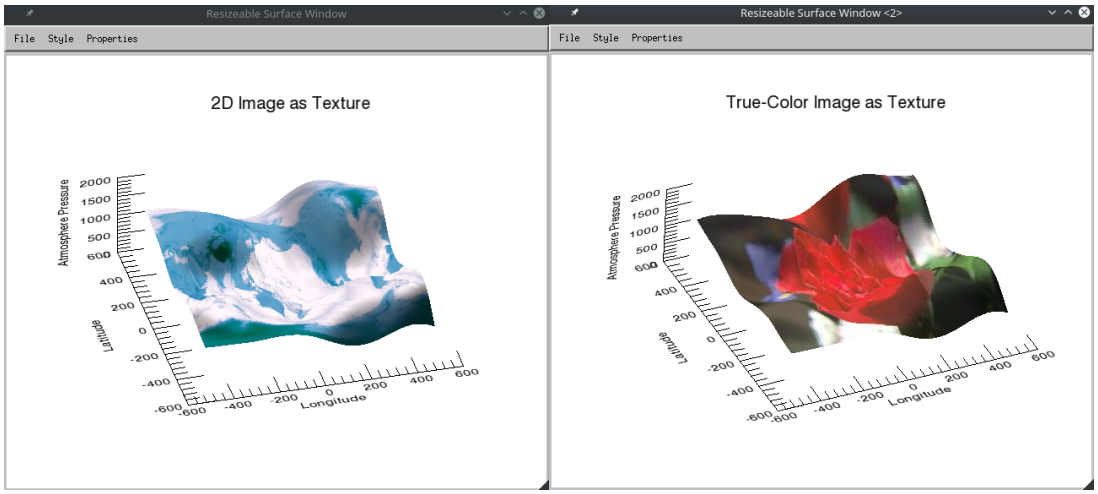

Figure 50: cgSurface Image as Texture

```
;; read the image
file = filepath(subdir=['examples', 'data'], $
                'elev_t.jpg')
read_jpeg, file, image
;; read the 64x64 dem byte data
file = filepath(subdir=['examples', 'data'], $
```

```
                'elevbin.dat')
dem = bytarr(64, 64)
openr, lun, file, /get_lun
readu, lun, dem
free_lun, lun
;; place the image onto the dem as a texture map
cgsurface, dem, texture_image=image, zscale=0.5
```

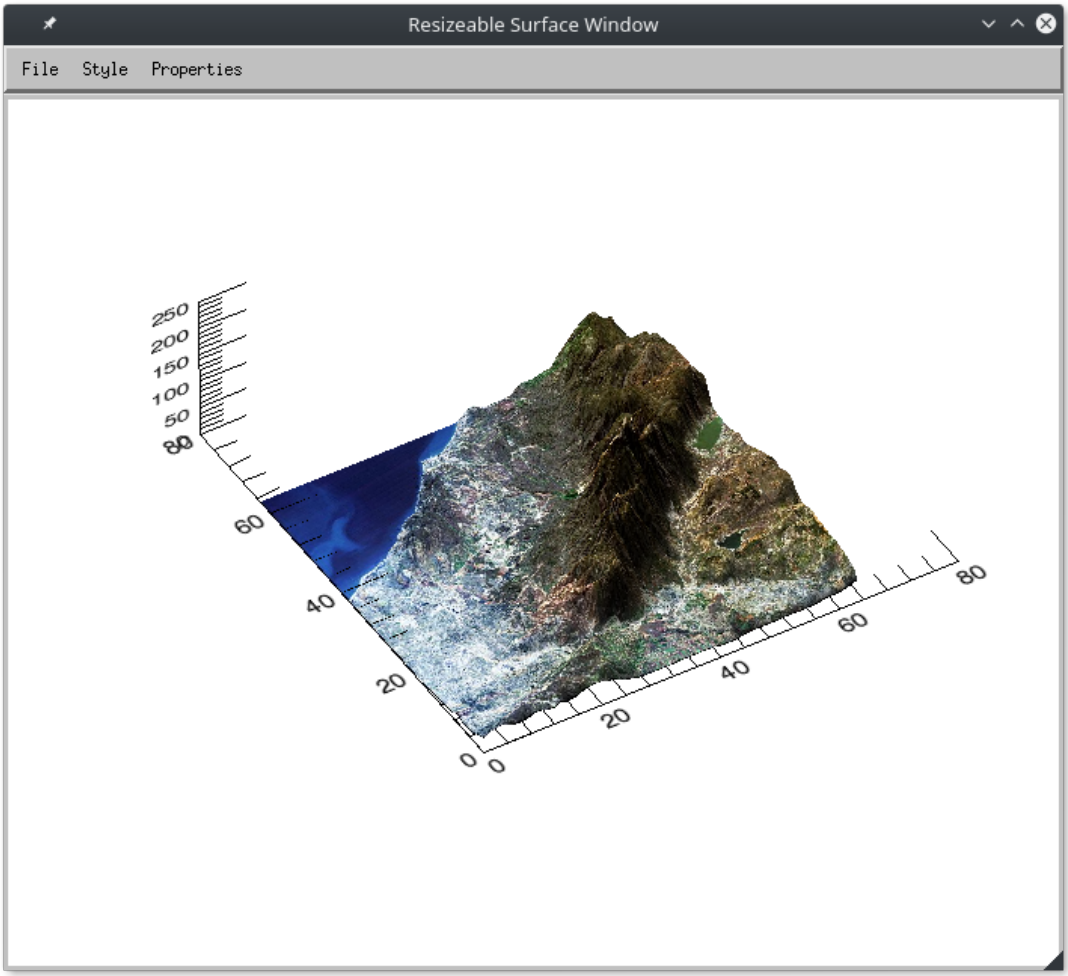

Figure 51: cgSurface 3D Map

11 Image position

11.1 margin

Margins are controlled by *!x.margin* and *!y.margin*, also by keywords of plot command *xmargin* and *ymargin*. The default values are [10,3] and [3,3], which means N times of character width of margin each side.

11.2 relative position

Relative position is controlled by *!p.position* or keyword *position* of plot command,which means the relative position of the left bottom corner and the top right corner.The value

is between 0 and 1.

```
contour, peak, position=[0.1, 0.1, 0.5, 0.9]
```

11.3 region

Similar to relative position, position of region is controlled by *!p.region*, but no plot keyword available.Don't forget to reset the system variable to 0 after use.

11.4 multiple image

!P.Multi is used to create multi-image in a window or PostScript page.

Table 25: !P.Multi

item	meaning
0	number of images rest you want to draw
1	number of columns
2	number of rows
3	number of floors in z axis
4	by row–0, or by column–1

Don't forget to reset it to 0 after use.
Sometimes we have to add a title, hence the out margin needs to be modified.

```
!p.Multi = [0,2,2,0,1]
!y.Omargin = [2,4]
Plot, cgDemoData(1), color=cgcolor('black'), $
      background=cgcolor('white')
Contour, cgDemoData(2), xstyle=1, ystyle=1, nlevels=10, $
         color=cgcolor('black')
Surface, cgDemoData(2), color=cgcolor('black')
Shade_Surf, cgDemoData(2), color=cgcolor('black'), $
            background=cgcolor('white')
XYOuts, 0.5, 0.9, /Normal, 'Four Graphics Plots', align=0.5, $
        charsize=2.5, color=cgcolor('black')
!p.Multi = 0
```

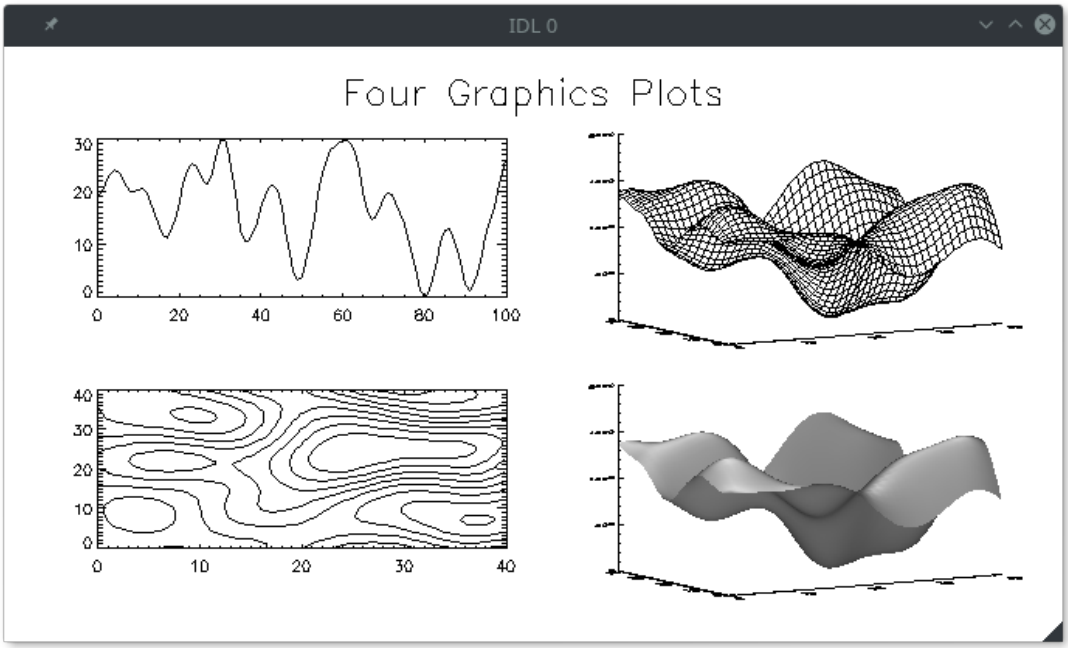

Figure 52: multi image

Another way is to use *Coyote library*,

```
cgdisplay
cgloadct, 33, rgb_table=palette
cgplot, cgdemodata(1), layout=[2, 2, 3], color='red'
cgcontour, cgdemodata(2), nlevels=12, $
           layout=[2, 2, 1], color='dodger blue'
cgsurf, cgdemodata(2), /elevation, layout=[2, 2, 4], $
        palette=palette
cgimage, cgdemodata(19), multimargin=4, /axes, $
         layout=[2, 2, 2]
```

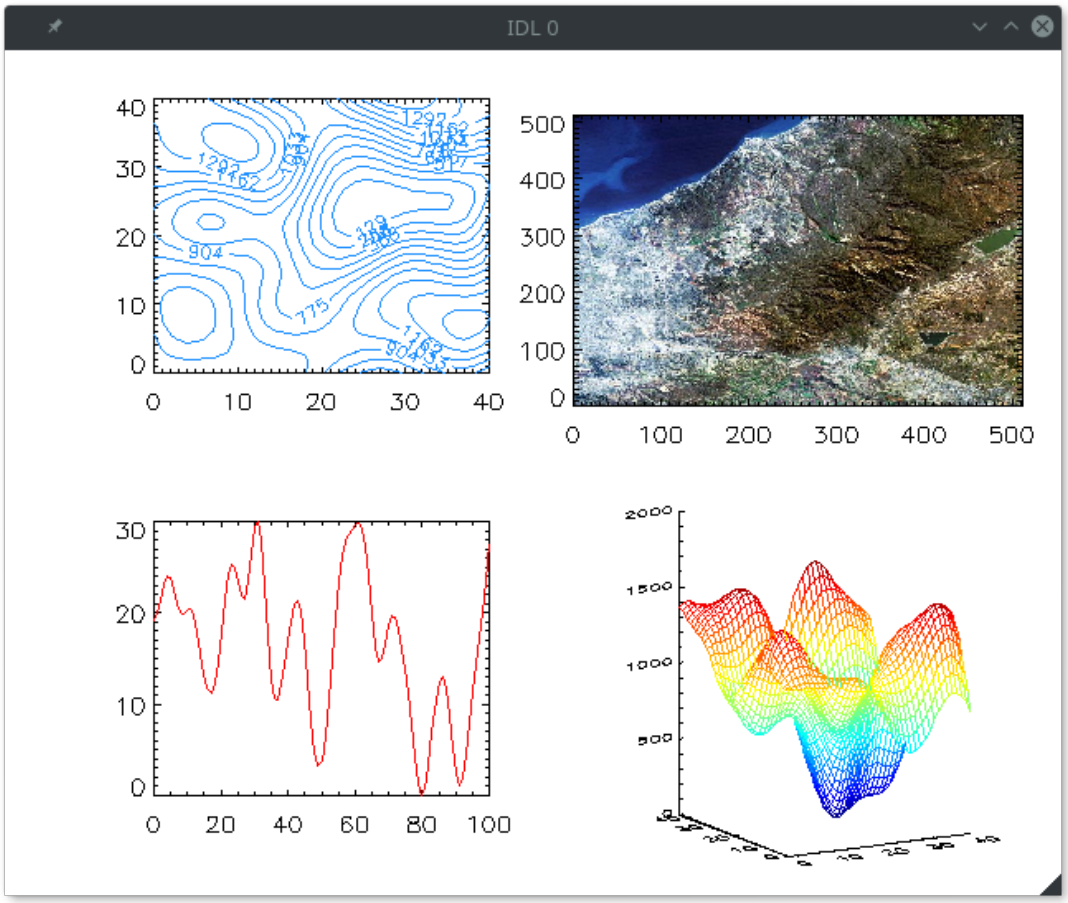

Figure 53: multi image

12 Add text to image

12.1 font type

Font is determined by *!P.Font* or *Plot* keyword *Font*.

Table 26: !P.Font

!P.Font	Font Type
-1	Vector font (aka soft font or Hershey font)
0	hard font
1	TrueType font

Set default *TrueType* font to *Courier*,

```
1 device, set_font='Courier', /tt_font
```

The same as *Times*,/Helvetica/ and *Symbol*.

12.2 XYOutS

12.2.1 *position*

The first parameter means X, the second parameter means Y by data coord by default. We can add *Normal* keyword to use normalized position

```
time = findgen(100)*6.0/100.0
plot, time, cgDemoData(1), Position=[0.15, 0.15, 0.95, 0.85]
XYoutS, 0.2, 0.92, 'Results: Experiment 35F3a', Size=2.0, /Normal
```

12.2.2 vector font

Table 27: Hershey font and index in IDL

value	font	value	font
!3	Simplex Roman	!12	Simplex Script
!4	Simplex Greek	!13	Complex Script
!5	Duplex Roman	!14	Gothic Italian
!6	Complex Roman	!15	Gothic German
!7	Complex Greek	!16	Cyrillic
!8	Complex Italian	!17	Triplex Roman
!9	Math Font	!18	Triplex Italian
!10	Special Characters	!20	Miscellaneous
!11	Gothic English	!X	(Back to default)

we can use *showfont* command to view each font

```
window, /free, xsize=600, ysize=600
showfont, 9, ''
```

use different vector font,

```
time = findgen(100)*6.0/100.0
plot, time, cgDemoData(1), Position=[0.15, 0.15, 0.95, 0.85], $
      xtitle='Time', ytitle='Signal', color=cgcolor('black'), $
      background=cgcolor('white'), charsize=1.5
XYoutS, 0.2, 0.92, '!12Results: Experiment 35F3a!X', Size=2.0, $
        /Normal, color=cgcolor('black')
```

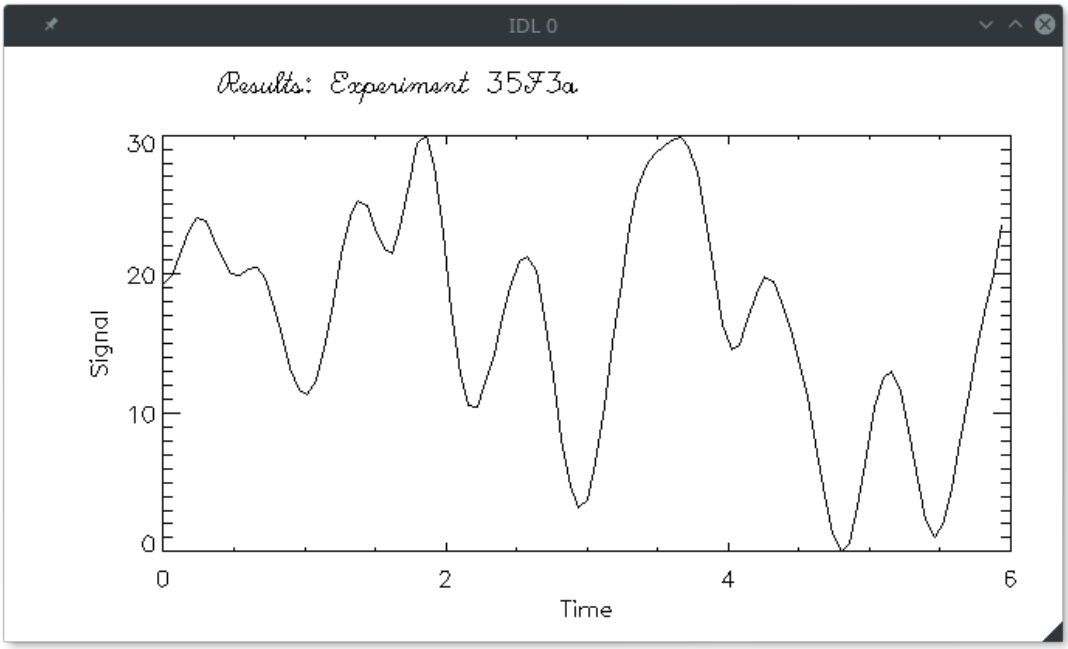

Figure 54: another vector font

12.2.3 *alignment*

Table 28: Alignment

Value	Alignment
0.0	left
0.5	center
1.0	right

```
plot, time, cgDemoData(1), Position=[0.15, 0.15, 0.95, 0.85], $
      xtitle='Time', ytitle='Signal', color=cgcolor('black'), $
      background=cgcolor('white'), charsize=1.5
XYoutS, 0.5, 0.92, '!12Results: Experiment 35F3a!X', Size=2.0, $
        /Normal, align=0.5, color=cgcolor('black')
```

will align center.

12.2.4 delete

we can delete text by repeat the previous *XYOutS* command followed by *Color=!P.Background* or other background color used.

12.2.5 orientation

Using the keyword *orientation* (between -180 and 180).

13 Image I/O of common formats

13.1 formats

Table 29: image formats

Format	Read	Write	Query
BMP	read_bmp	write_bmp	query_bmp
GIF	read_gif	write_gif	query_gif
JPEG	read_jpeg	write_jpeg	query_jpeg
PICT	read_pict	write_pict	query_pict
PBM/PPM	read_ppm	write_ppm	query_ppm
PNG	read_png	write_png	query_png

13.2 reading image data

```
startPath = File_DirName(Filepath(Root_Dir=!Dir, $
                                  Subdirectory=['examples', 'data'], '*'))
file = Dialog_Pickfile(File='muscle.jpg', $
                       Filter='*,jpg', Path=startPath)
Read_JPEG, file, image
```

or you can use *read_image*

```
image = read_image(file)
```

for *png* and *tiff* format image

```
image = read_png(png_file)
image = read_image(png_file)
image = read_tiff(file, r, g, b, GeoTIFF=geo)
image = read_image(file, r, g, b, GeoTIFF=geo)
```

for *gif* file

```
read_gif, file, image, r, g, b
```

13.3 information about image files

```
validFile = query_image(file, info)
help, info, /structure
print, info.dimensions
```

There is a program in the *Coyote Library* named *ImageSelect* that takes advantage of *Dialog_Pickfile*, and the *Read_** and *Query_** routines to allow you to select common image files to read.

```
image = imageselect(filter='*.jpg', /demo)
```

13.4 writing image files

let's take a snapshot and write the result to the image file

```
cghistoplot, cgdemodata(7), /fill
snapshot = tvrd(true=1)
help, snapshot
```

the resulting image can be sent to any appropriate output file type.

```
write_jpeg, 'test.jpg', snapshot, true=1, quality=75
write_png, 'test.png', snapshot
write_tiff, 'test.tiff', reverse(snapshot)
write_image, 'test.jpg', 'JPEG', snapshot
```

Or you can simply use *cgSnapshot*

```
void = cgsnapshot(filename='test', /jpeg)
void = cgsnapshot(filename='test', /png)
void = cgsnapshot(filename='test', /tiff)
void = cgsnapshot(filename='test', /jpeg, /nodialog)
```

14 Display image

14.1 traditional image display commands

14.1.1 TVScl

It is not recommended to use this command. You'd better use *BytScl* to byte scale an image data and then display the image

14.1.2 TV

Not recommended either :-(

If you want to use the *TV* command to display a 2D image, then you need to use the indexed color model when you display the image.

display true color image

```
rose = cgDemoData(16)
tv, rose, true=1
```

Unfortunately The *TV* command wouldn't display an alpha channel image properly in any case

> Basically, it takes on the order of about 25 lines of code to display an image properly with the *TV* command. And that is only if you want to get the colors right. If you want the image to show up in the right place in a PostScript file, or if you want to do other smart things, and believe me you do, you can multiply this number by a factor of 10 or so to get the number of commands you really need. – *David W. Fanning*

14.2 alternative image display commands

14.2.1 *ImDisp* (*by Liam Gumley*)

http://www.gumley.com/PIP/Programs/imdisp.pro

14.2.2 *cgImage* (*Coyote Library*)

14.3 positioning

```
position = [0.15,0.15,0.9,0.9]
stern = (cgdemodata(10))[*, *, 1]
cgimage, stern, position=position
```

Figure 55: cgImage

```
position = [0.15,0.15,0.9,0.9]
stern = (cgdemodata(10))[*,*,1]
cgimage, stern, position=position, /keep_aspect
```

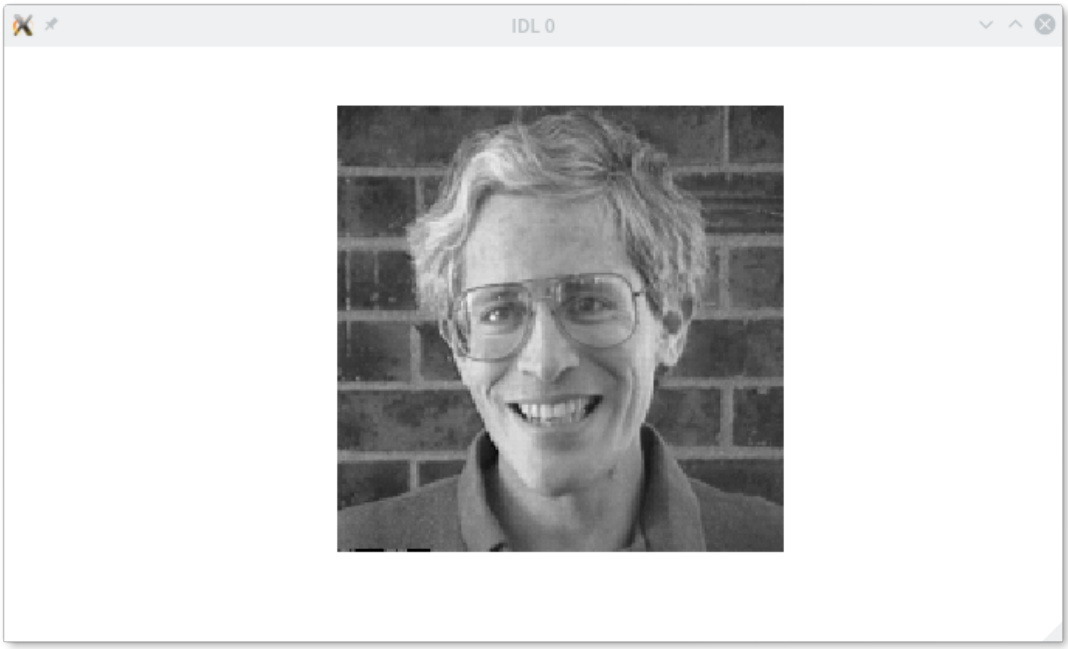

Figure 56: cgImage keep aspect

```
cgImage, stern, Margin=0.1, /Keep_Aspect, /Axes, $
         XRange=[-10,10], YRange=[0,50], Color='navy', $
         AxKeywords={XTicklen:-0.035, YTickLen:-0.035}
```

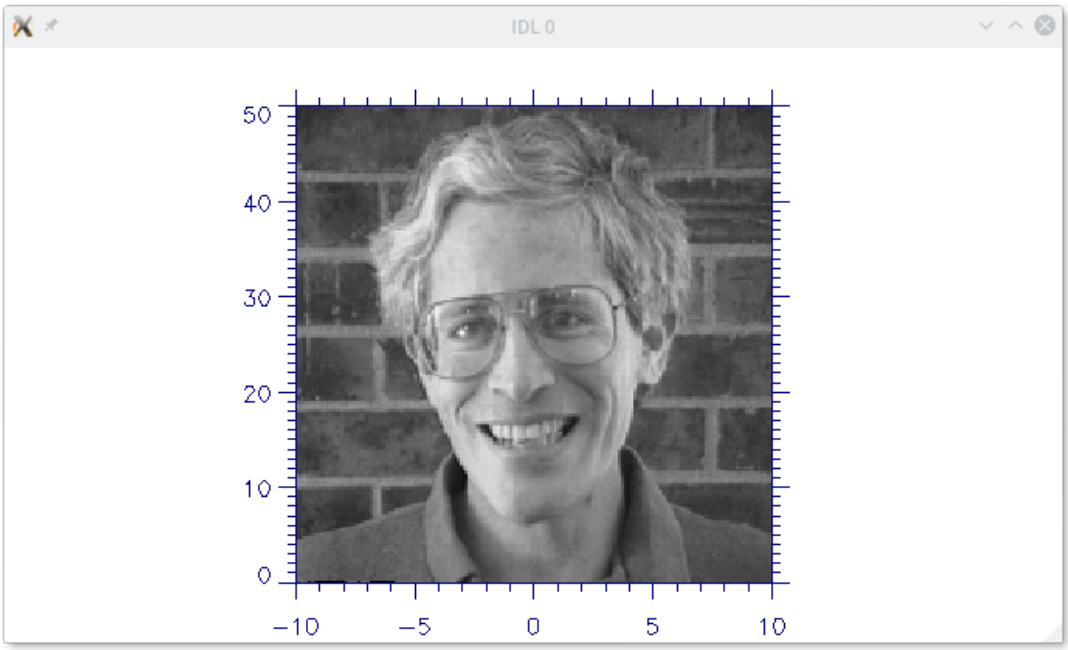

Figure 57: cgImage coordinate

14.4 erase graphics window before display

```
cgErase, 'white'
cgImage, stern, margin=0.1
```

14.5 image backgrounds

```
cgImage, stern, background='wheat', margin=0.15, /axes, $
         color='opposite'
```

by setting the axes color to *opposite*,/cgColor/ will calculate a color that contrast with the background color

14.6 display images with colors

```
cgLoadCT, 13, /brewer, /reverse, rgb_table=palette
cgImage, stern, background='wheat', margin=0.15, $
         /axes, color='opposite', palette=palette
```

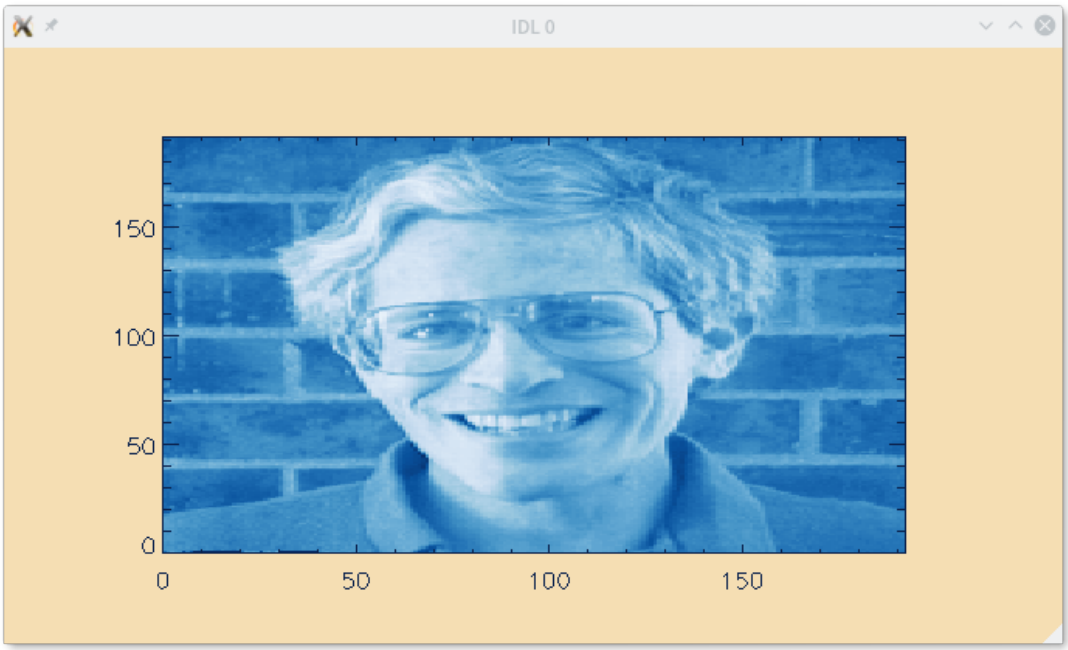

Figure 58: cgImage color

14.7 combining images with other graphical display

```
image = cgDemoData(18)
cgLoadCT, 4, /brewer, /reverse
cgImage, image, /save, /axes, /keep_aspect
cgContour, image, nlevels=10, label=2, /overplot
```

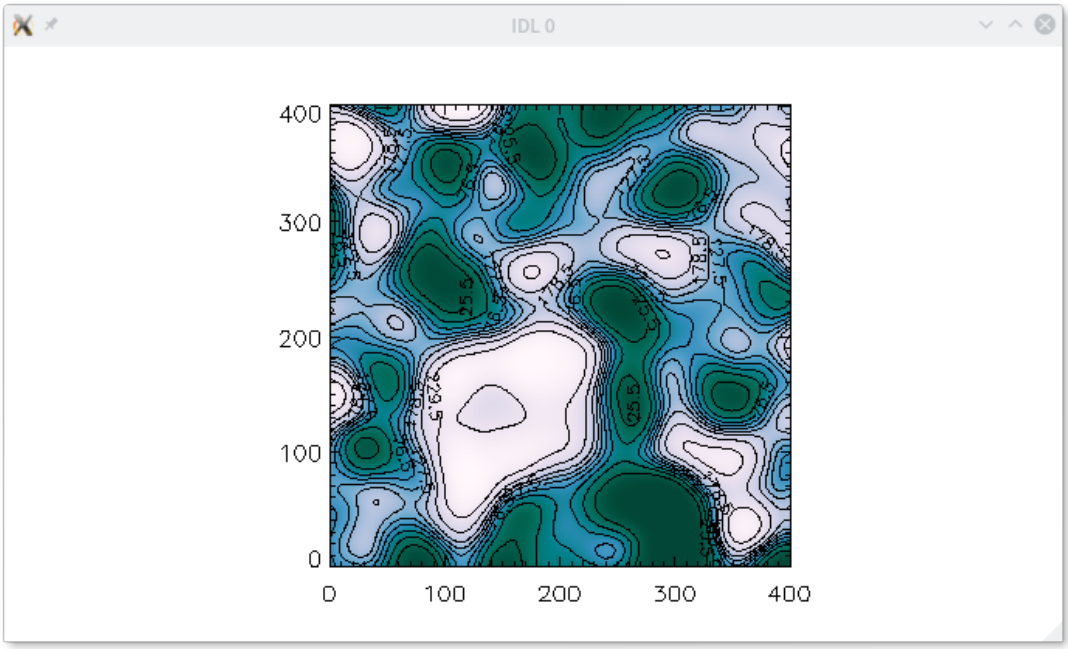

Figure 59: cgImage color

14.8 interacting with images

use *cgimageinfo* to identify the location of the cursor in the display and returns the value in that place, interactively

```
cgimageinfo,image
Click cursor in image window (window 0) to get image values.
Use LEFT mouse buton to inquire.
Use RIGHT mouse button to exit program.

Expecting cursor clicks in image window...
Value at  (85,258) is 7
Value at  (143,141) is 213
Value at  (176,169) is 255
Value at  (303,327) is 8
Value at  (175,384) is 25

Good-bye. cgImageInfo has returned control to IDL.
```

14.9 identify locations in the image

```
Cursor, xloc, yloc, /Down, /Device
normalcoords = convert_coord(xloc, yloc, /device, /to_normal)
xn = normalcoords[0]
yn = normalcoords[1]
p = position
dims=image_dimensions(image, xsize=xsize, ysize=ysize)
xvec = scale_vector(findgen(xsize + 1), p[0], p[2])
yvec = scale_vector(findgen(xsize + 1), p[1], p[3])
xpixel = value_locate(xvec, xn)
ypixel = value_locate(yvec, yn)
print, image[xpixel, ypixel]
```

14.10 zooming images

14.10.1 for non-true-color images

```
minimg = min(stern, max=maximg)
subregion = stern[50:99, 50:99]
window, xsize=400, ysize=400
cgimage, bytscl(subregion, min=minimg, max=maximg)
```

14.10.2 for true-color-images

```
rose = cgdemodata(16)
subregion = rose[*, 50:100, 50:100]
cgimage, subregion
```

14.10.3 *FSC_ZImage*

an interactive zoom image program from *Coyote Library*

14.11 multiple images

```
head = cgdemodata(8)
head = reverse(head, 2)
s=size(head, /dimensions)
cgloadct, 0
cgdisplay, 9*s[0], 6*s[1]
!p.multi = [0, 9, 6]
!x.omargin = [0, 25]
!y.omargin = [0, 8]
for j=0, 53 do cgimage, head[*, *, j], $
                        background='white', multimargin=0.25
!p.multi = 0
cgColorbar, /vertical, range=[0, 10], $
            position=[0.94, 0.15, 0.97, 0.80], $
            format='(F0.2)'
cgtext, 0.45, 0.95, 'MRI Study of the Head', $
        /normal, Alignment=0.5
!x.omargin = 0
!y.omargin = 0
```

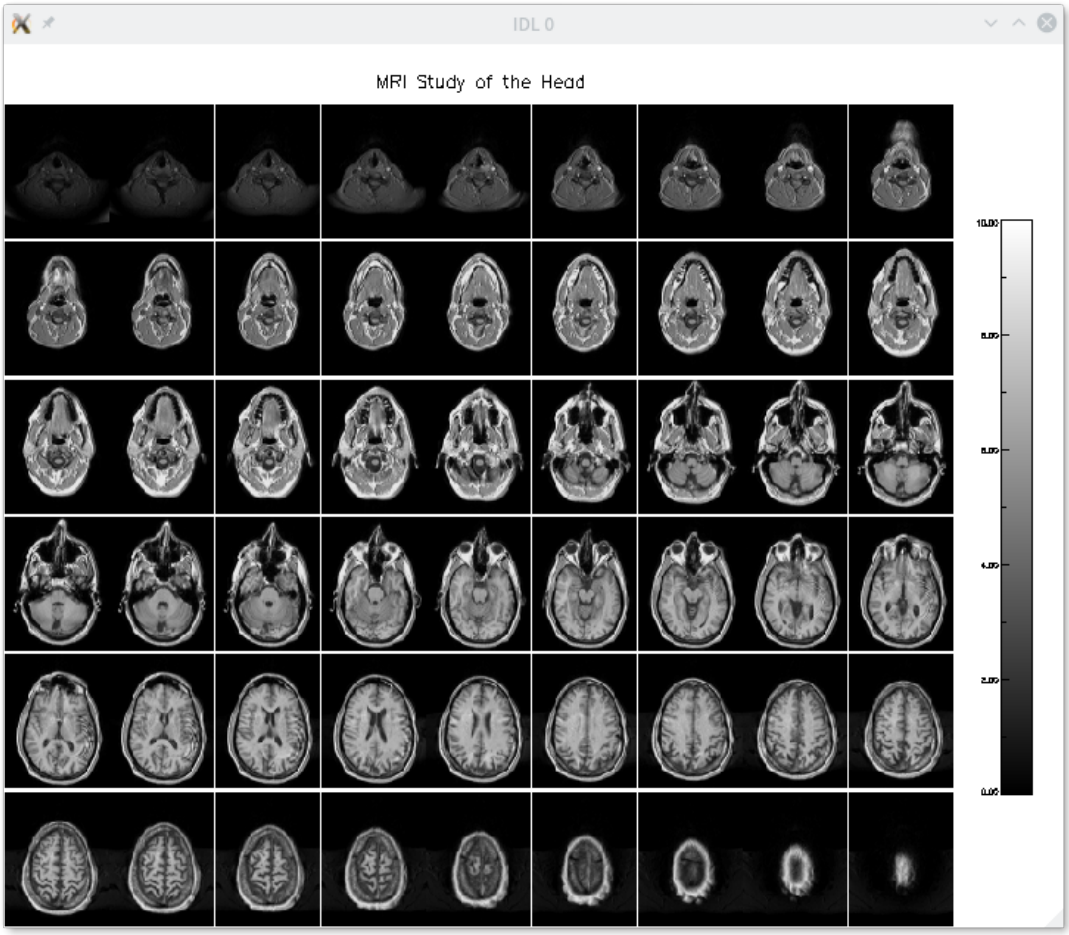

Figure 60: cgImage Multi-display

The MultiMargin keyword is a four-element array that specifies margins about the bottom, left, top, and right of the image, respectively.

```
loadct, 33, /silent
!p.multi = [0, 2, 2]
margin = [4, 4, 2, 0]
cgimage, cgdemodata(18), /keep_aspect, /axes, $
         multimargin=margin
cgimage, cgdemodata(18), /keep_aspect, /axes, $
         multimargin=margin
cgimage, cgdemodata(18), /keep_aspect, /axes, $
         multimargin=margin
cgimage, cgdemodata(18), /keep_aspect, /axes, $
         multimargin=margin
!p.multi = 0
```

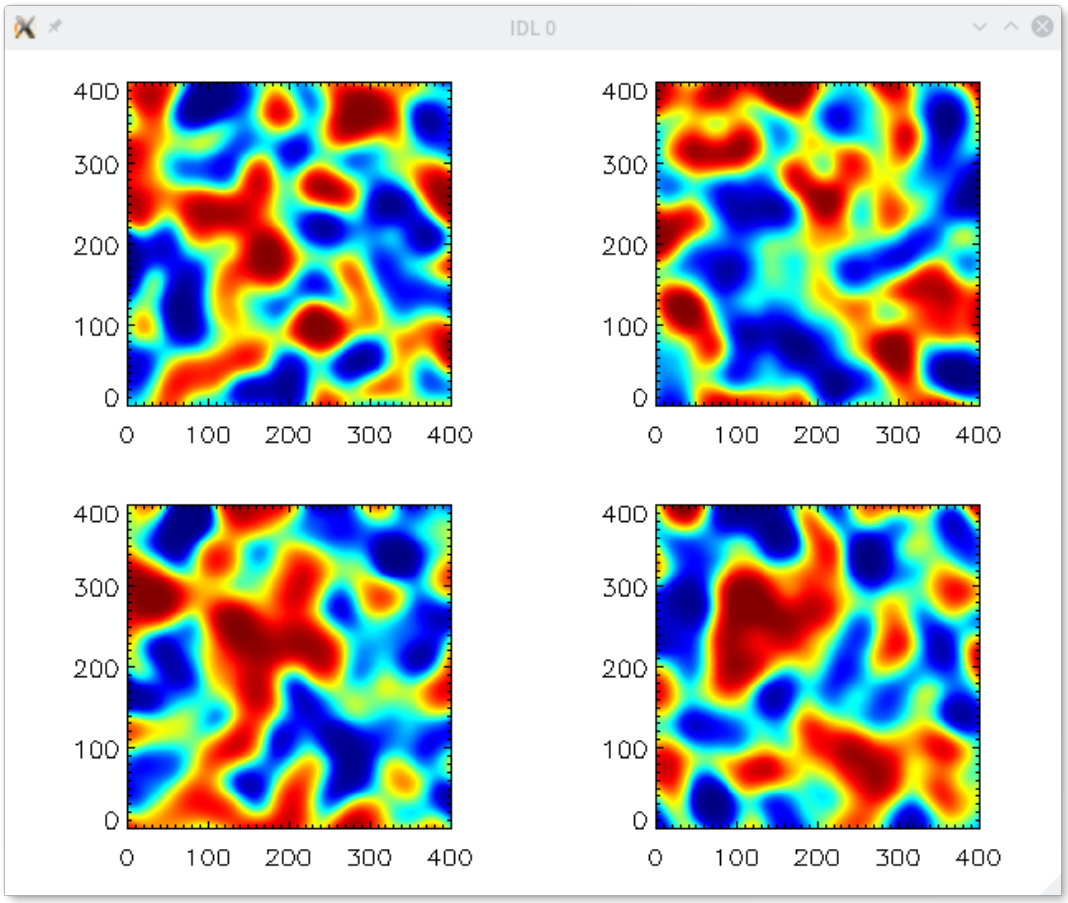

Figure 61: cgImage MultiMargin

14.12 transparent images

Transparent images have an alpha channel that indicates how the transparent image is to be “blended” with the information already on the display.Alpha channels are generally coded as values from 0 to 1 (or, in images, from 0 to 255), with 0 indicating total transparency (i.e., display the background pixel) and 1 indicating total opacity (i.e., display the foreground pixel).

14.12.1 image with alpha channel

download a png file from `http://www.idlcoyote.com/books/tg/data/toucan.png`

```
toucan = read_png('toucan.png')
!p.multi = [0, 2, 1]
window, xsize=162*2, ysize=150
cgimage, toucan[0:2, *, *]
cgimage, reform(toucan[3, *, *])
!p.multi = 0
```

Figure 62: cgImage separate alpha

```
window, xsize=400, ysize=400
cgerase, color='bisque'
cgimage, toucan, /keep_aspect, /noerase, margin=0.2
```

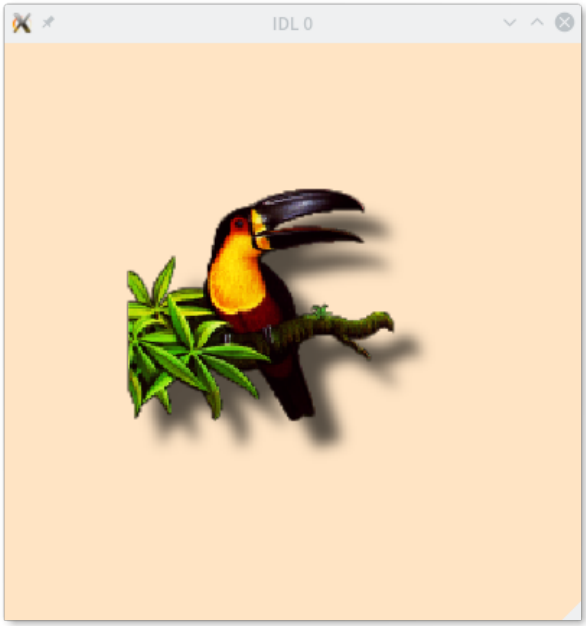

Figure 63: cgImage alpha channel image

```
window, 1, xsize=400, ysize=400
marsfile = Filepath(Subdir=['examples',  'data'], $
                    'marsglobe.jpg')
read_jpeg, marsfile, mars
cgimage, mars
cgimage, toucan, alphafgpos=[0.3, 0.3, 0.7, 0.7]
```

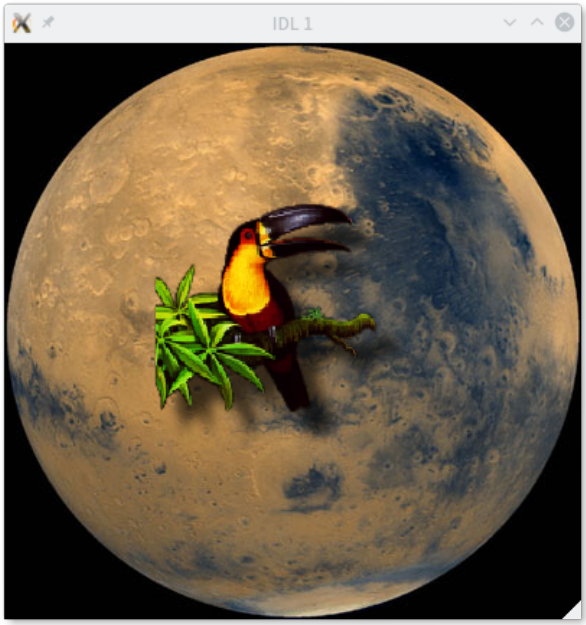

Figure 64: cgImage alpha channel image with background image

14.12.2 creating blended images

```
scanfile = Filepath(Subdir=['examples','data'], $
                    'md1107g8a.jpg')
Read_JPEG, scanfile, scan
marsfile = Filepath(Subdir=['examples', 'data'],$
                    'marsglobe.jpg')
read_jpeg,marsfile,mars
window,xsize=400,ysize=400
LoadCT, 0
cgImage, scan, /NoInterp
snapshot = cgSnapshot()
alpha = 0.6
blended = (snapshot * alpha) + (mars * (1-alpha))
Window, XSize=400*3, YSize=400
!P.Multi = [0,3,1]
cgImage, mars
cgImage, scan
cgImage, blended
!P.Multi = 0
```

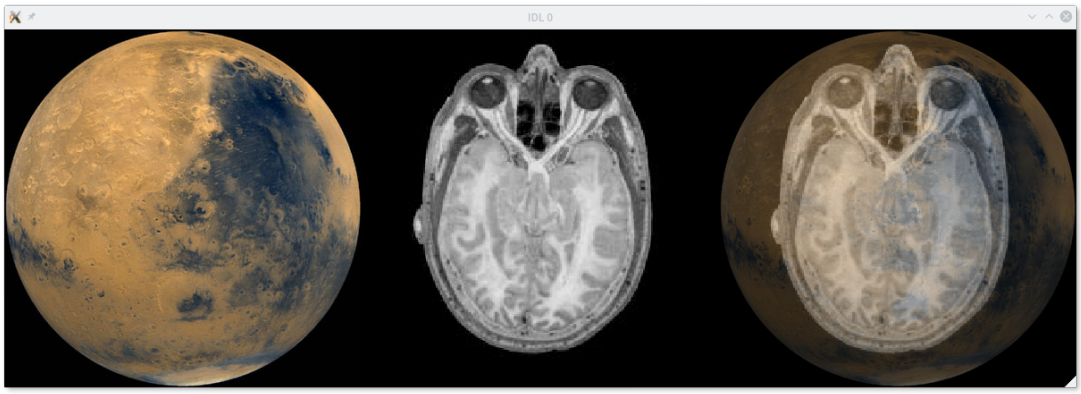

Figure 65: cgImage blended image

or you can use *cgBlendImage* program from *Coyote Library*

```
window,xsize=400,ysize=400
cgBlendImage,snapshot,mars,alpha=0.6
```

14.12.3 creating transparent images

```
logofile = Filepath(Subdir=['examples','data'], $
                    'examples.tif')
logo = Read_TIFF(logofile, r, g, b)
logo = Reverse(logo, 2)
TVLCT, r, g, b
Window, XSize=375, YSize=150
cgImage, logo, /NoInterp
logo24 = cgSnapshot()
alpha = bytarr(375,150)+255B
logo24 = Transpose(logo24, [1, 2, 0])
red = logo24[*, *, 0]
grn = logo24[*, *, 1]
blu = logo24[*, *, 2]
blackIndices = Where( (red EQ 0) AND (grn EQ 0) $
                      AND (blu EQ 0), count)
IF count GT 0 THEN alpha[blackIndices] = 0B
logo32 = [ [[red]], [[grn]], [[blu]], [[alpha]] ]
logo32 = Transpose(logo32, [2,0,1])
Write_PNG, 'idl_logo.png', logo32
image = Make_Transparent_Image(Color='black', $
                               Filename='idl_logo.png', $
                               /Save_PNG)
Window, XSize=400, YSize=400
cgImage, mars
cgImage, logo32, alphafgpos=[0.30, 0.05, 0.70, 0.25]
```

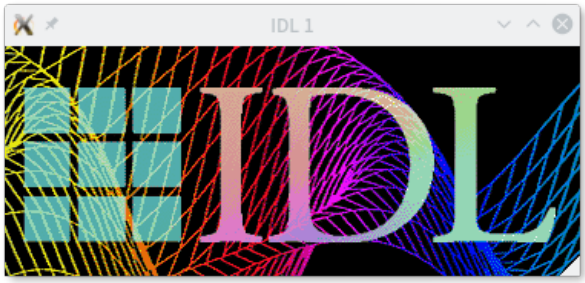

Figure 66: idl logo

Figure 67: cgImage transparent logo

14.12.4 *transparent*

```
cgDisplay, 500, 500
cgImage, cgDemoData(7), CTIndex=0
cgImage, cgDemoData(5), CTIndex=33, Transparent=30, $
         Missing_Value=0, AlphaFGPos=[0.5, 0.5, 1.0, 1.0]
```

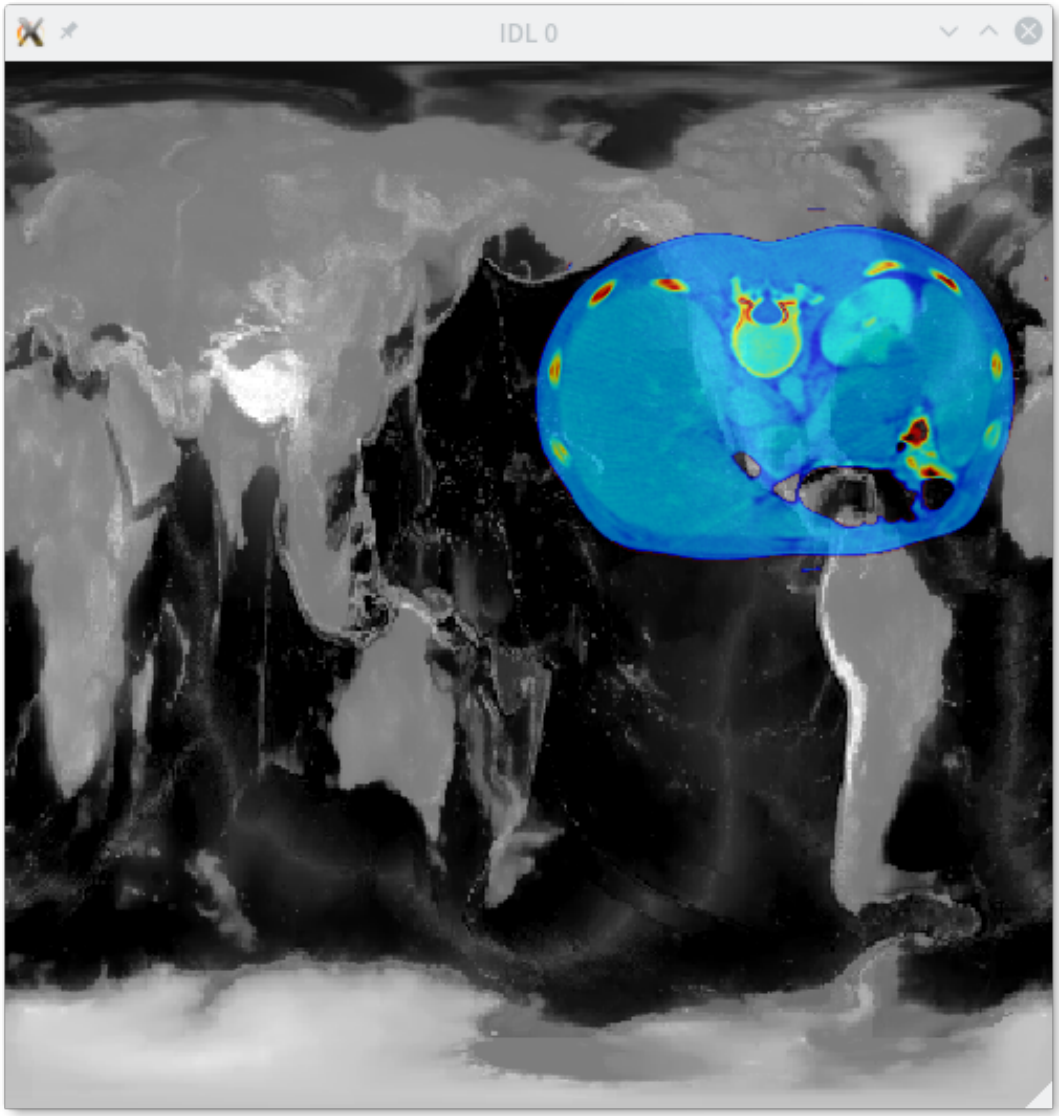

Figure 68: cgImage transparent display

14.13 image in resizable graphics windows

14.13.1 *ImDisp*

```
image = cgdemodata(19)
cgwindow, 'imdisp', image, margin=0.2, /erase
```

14.13.2 *cgImage*

```
cgimage, image, margin=0.2, /window
```

when combined with other graphical display

```
cgplot, cgdemodata(1), yrange=[0, 80], /window
cgimage, image, position=[0.7, 0.5, 0.9, 0.9], $
         /keep_aspect, /addcmd
```

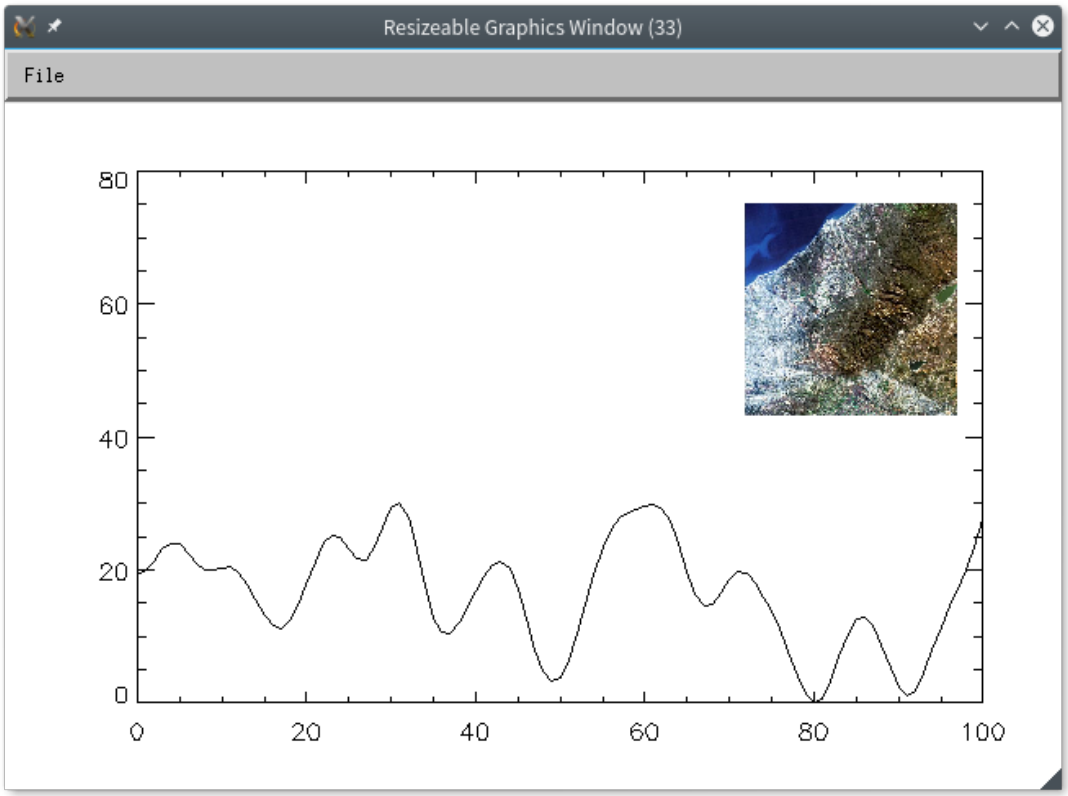

Figure 69: cgImage transparent display

15 Postscript output

15.1 how does it work

```
thisDevice = !d.name
set_plot, 'ps'
device, filename='plot_example.ps'
x = findgen(200) * 0.1
plot, x, sin(x)
device, /close_file
set_plot, thisDevice
```

The *!d.name* system variable stores the name of the current graphics device. This is *WIN* on a Windows operating system, and is *X* on an Unix operating system. Once you have changed your graphics device, any graphical output will be sent to the new graphics device, e.g., when *!d.name* is set to *PS*, graphical output will be written into a PostScript file. By default, the PostScript file that is created is named *idl.ps*, which can be modified with the *filename* keyword to the *Device* command.

15.2 change the configuration

You can get the information of the current state of the PostScript device by

```
set_plot, 'ps'
help, /device
```

The default page orientation is *Portrait* with color output disabled. If you switch to *Landscape* orientation and check the configuration

```
device, /landscape
help, /device
```

Table 30: Commonly used *device* keywords for the *PS* device

Keyword	Purpose
filename	Set the output file name (default: idl.ps)
/portrait	Select portrait orientation (default)
/landscape	Select landscape orientation
/color	Select color output (default if grayscale)
bits_per_pixel	Set number of bits per pixel (default is 4)
xsize	Set width of drawable area
ysize	Set height of drawable area
xoffset	Set offset of drawable area origin from left of page
yoffset	Set offset of drawable area origin from bottom of page
/inches	Set units of inches for size and offset keywords in this call to device only (default is centimeters)
font_size	Set font size (points, default is 12)
/close_file	Complete and close the output file

15.3 producing color (or grayscale) output

15.3.1 24-bit color

```
thisdevice = !d.name
set_plot, 'ps'
device, /color, bits_per_pixel=8
set_plot, thisdevice
```

Setting the *color* keyword automatically copies the current color table vectors into the PostScript file, although colors can be changed at any time in PostScript files by loading colors and color tables. You can also copy the current color table vectors into the PostScript file at the time you set the PostScript device by using the *copy* keyword to *set_plot*.

```
set_plot, 'ps', /copy
```

But the input color mode now is *index* by default, which makes the device act like an 8-bit index color graphics device. To put the PostScript device into a 24-bit color mode, you set the *decomposed* keyword to 1.

```
device, decomposed=1
```

or use the routines from the *Coyote Library*

```
cgsetcolorstate, 1, currentstate=currentstate
```

15.3.2 color background

On a display monitor, which is a raster device,

```
plot, cgdemodata(1), background=cgcolor('wheat'), $
      color=cgcolor('opposite')
```

However, this same command would result in a black on a white background if sent to a PostScript file, you have to 'fill' the PostScript file first.

```
thisdevice = !d.name
set_plot, 'ps'
device, filename='plot_fill_back.ps'
device, /color, bits_per_pixel=8
cgsetcolorstate, 1, currentstate=currentstate
polyfill, [0,1,1,0,0], [0,0,1,1,0], /normal, $
          color=cgcolor('wheat')
plot, cgdemodata(1), color=cgcolor('opposite'), $
      /noerase
device, /close_file
set_plot, thisdevice
```

or, simply use *cgplot*

```
thisdevice = !d.name
set_plot, 'ps'
device, filename='plot_fill_back.ps'
device, /color, bits_per_pixel=8
cgsetcolorstate, 1, currentstate=currentstate
cgplot, cgdemodata(1), background='wheat'
device, /close_file
set_plot, thisdevice
```

15.4 window aspect ratio

The *Coyote Library* contains a function, named *pswindow*, that can "match" the aspect ratio of the current graphics window with a centered PostScript window having the same aspect ratio.

```
window, xsize=600, ysize=400
output = pswindow(/metric)
help, output, /structure
```

then simply add the *output* as extra arguments

```
set_plot, 'ps'
device, _extra=output
```

We can also use *pswindow* command to set the window aspect ratio rather than matching the window.

```
output = pswindow(aspectratio=2./3)
```

15.5 setting size and offset manually

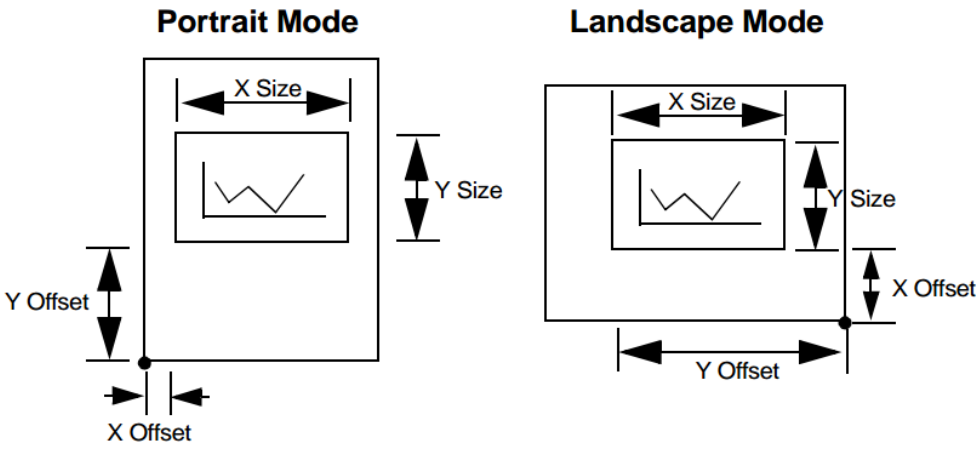

Figure 70: Portrait and Landscape orientation

Now if you'd like to reproduce the plot in Portrait orientation, on a 8.5- inch × 11-inch page.

```
;; Set size and offset
page_width = 8.5
page_hight = 11.0
xsize = 6.5
ysize = 5.5
xoffset = (page_width - xsize) * 0.5
yoffset = (page_hight - ysize) * 0.5
;; Config the PostScript device and plot
this_device = !d.name
set_plot, 'ps'
device, filename='plot_port.ps', /portrait
device, xsize=xsize, ysize=ysize, $
        xoffset=xoffset, yoffset=yoffset, /inches
x = findgen(200) * 0.1
plot, x, sin(x)
device, /close_file
set_plot, this_device
```

To create the same plot in Landscape orientation with the same page size, the offset of the drawable area must be recomputed using a method that takes account of the new orientation

```
;; Set offset for landscape orientation
xoffset = (page_width - ysize) * 0.5
yoffset = (page_hight - xsize) * 0.5 + xsize
;; Config and plot
this_device = !d.name
set_plot, 'ps'
device, filename='plot_landscape.ps', /landscape
device, xsize=xsize, ysize=ysize, $
        xoffset=xoffset, yoffset=yoffset, /inches
x = findgen(200) * 0.1
```

```
plot, x, sin(x)
device, /close_file
set_plot, this_device
```

However, the Landscape file produced is **UPSIDE DOWN** ! You can simply use the *Cgfixps* routine from the *Coyote Library* to fix the bug

```
cgfixps, 'plot_landscape.ps'
```

15.6 setting size and offset automatically

```
PRO PSON, FILENAME=FILENAME, PAPER=PAPER, MARGIN=MARGIN, $
          PAGE_SIZE=PAGE_SIZE, INCHES=INCHES, ASPECT=ASPECT, $
          LANDSCAPE=LANDSCAPE, QUIET=QUIET

  ;; Check arguments
  if (n_elements(filename) eq 0) then filename = 'idl.ps'
  if (n_elements(paper) eq 0) then paper = 'LETTER'
  if (n_elements(margin) eq 0) then begin
     margin = 2.5
  endif else begin
     if keyword_set(inches) then margin = margin * 2.54
  endelse

  ;; Check if PostScript mode is active
  if !d.name eq 'PS' then begin
     message, 'POSTSCRIPT output is already active', /continue
     return
  endif

  ;; Get ratio of character width/height to screen width/height
  xratio = float(!d.x_ch_size) / float(!d.x_vsize)
  yratio = float(!d.y_ch_size) / float(!d.y_vsize)

  ;; Save current device information in common block
  common pson_infomation, info
  info = {device:!d.name, window:!d.window, font:!p.font, $
          filename:filename, xratio:xratio, yratio:yratio}

  ;; Get size of page (centimeters)
  widths = [[8.5, 8.5, 11.0, 7.25] * 2.54, 21.0, 29.7]
  heights = [[11.0, 14.0, 17.0, 10.50] * 2.54, 29.7, 42.0]
  names = ['LETTER', 'LEGAL', 'TABLOID', 'EXECUTIVE', $
           'A4', 'A3']
  index = where(strupcase(paper) eq names, count)
  if (count ne 1) then begin
     message, 'PAPER selection not supported', /continue
     return
  endif
  page_width = widths[index[0]]
  page_height = heights[index[0]]

  ;; If page size was supplied, use it
  if (n_elements(page_size) eq 2) then begin
     page_width = page_size[0]
     page_height = page_size[1]
     if keyword_set(inches) then begin
```

```
      page_width = page_width * 2.54
      page_height = page_height * 2.54
   endif
endif

;; Compute aspect ratio of page when margins are subtracted
page_aspect = float(page_height - 2.0 * margin) / $
              float(page_width - 2.0 * margin)

;; Get aspect ratio of current graphics window
if (!d.window ge 0) then begin
   win_aspect = float(!d.y_vsize) / float(!d.x_vsize)
endif else begin
   win_aspect = 512.0 / 640.0
endelse

;; If aspect ratio was supplied, use it
if (n_elements(aspect) eq 1) then $
   win_aspect = float(aspect)

;; Compute size of drawable area
case keyword_set(landscape) of
   0 : begin
      if (win_aspect ge page_aspect) then begin
         ysize = page_height - 2.0 * margin
         xsize = ysize / win_aspect
      endif else begin
         xsize = page_width - 2.0 * margin
         ysize = xsize * win_aspect
      endelse
   end
   1 : begin
      if (win_aspect ge (1.0 / page_aspect)) then begin
         ysize = page_width - 2.0 * margin
         xsize = ysize / win_aspect
      endif else begin
         xsize = page_height - 2.0 * margin
         ysize = xsize * win_aspect
      endelse
   end
endcase

;; Compute offset of drawable area from page edges
if (keyword_set(landscape) eq 0) then begin
   xoffset = (page_width - xsize) * 0.5
   yoffset = (page_height - ysize) * 0.5
endif else begin
   xoffset = (page_width - ysize) * 0.5
   yoffset = (page_height - xsize) * 0.5 + xsize
endelse

;; Switch to PostScript device
set_plot, 'PS'
device, landscape=keyword_set(landscape), scale_factor=1.0
device, xsize=xsize, ysize=ysize, $
        xoffset=xoffset, yoffset=yoffset
device, filename=filename, /color, bits_per_pixel=8

```

```
  ;; Set character size
  xcharsize = round(info.xratio * !d.x_vsize)
  ycharsize = round(info.yratio * !d.y_vsize)
  device, set_character_size=[xcharsize, ycharsize]

  ;; Report to user
  if (keyword_set(quiet) eq 0) then $
     print, filename, $
            format='("Started POSTSCRIPT output to ", a)'

END
```

```
PRO PSOFF, QUIET=QUIET

  ;; Check that PostScript output is active
  if (!d.name ne 'PS') then begin
     message, 'POSTSCRIPT output not active: ' + $
              'nothing done', /continue
     return
  endif

  ;; Get entry device information from common block
  common pson_infomation, info
  if (n_elements(info) eq 0) then begin
     message, 'PSON was not called prior to PSOFF: ' + $
              'nothing done', /continue
     return
  end

  ;; Close PostScript device
  device, /close_file
  set_plot, info.device

  ;; Restore window and font
  if (info.window ge 0) then wset, info.window
  !p.font = info.font

  ;; Report to user
  if (keyword_set(quiet) eq 0) then $
     print, info.filename, $
            format='("Ended POSTSCRIPT output to ", a)'

END
```

```
pson, file='surface.ps', margin=1.0
device, color=0, bits_per_pixel=4
z = dist(64)
z = shift(z, 32, 32)
z = exp(-(z * 0.1) ^ 2)
shade_surf, z, xstyle=1, ystyle=1, charsize=2, $
pixels=1000, xmargin=[4, 1], ymargin=[1, 0]
psoff
```

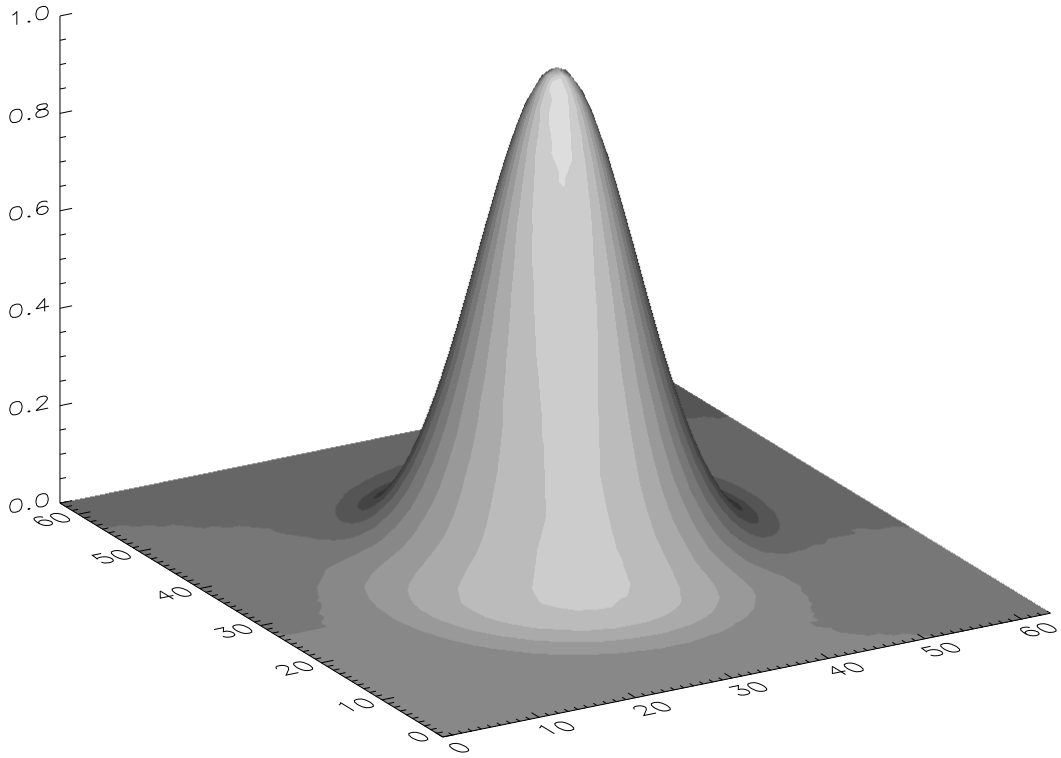

Figure 71: Surface

15.7 encapsulated postscript output

To produce PostScript output that can be inserted directly into other Post- Script documents, such as journal articles or books, you must select the encapsulated option before you send graphics output to the PostScript file.

```
device, /encapsulated
```

If you would like to be able to see a representation of the graphic in the document into which you imported the file, you must specify a value for the *preview* keyword. A Preview keyword value of 1 will cause the Post- Script driver to include a bit-mapped image of the graphic along with its PostScript description. If you would prefer to use a TIFF preview image, instead of a bit-mapped image, set the *preview* keyword to 2.

```
device, /encapsulated, preview=2
```

15.8 setting the language level

```
device, language_level=2
```

15.9 using PostScript fonts

15.9.1 font system

The font type is controlled by *!p.font* variable. A value of -1 selects Hershey fonts, a value of 0 selects hardware fonts, and a value of 1 selects TrueType fonts.

```
this_device, !d.name
set_plot, 'ps'
device, _extra=pswindow(aspectratio=1.0/3), font_size=12, $
        /encapsulated, filename='three_font_system.ps', $
        /helvetica, /landscape
!p.multi=[0, 3, 1]
data = cgdemodata(1)
cgplot, data, font=-1, title='Hershey Fonts'
cgplot, data, font=0, title='Hardware Fonts'
cgplot, data, font=1, title='TrueType Fonts'
device, /close_file
!p.multi=0
set_plot, this_device
```

15.9.2 font mapping

Table 31: Default mapping of fonts

Number	Hershey Font	PostScript Font
!3	Simplex Roman	Helvetica
!4	Simplex Greek	Helvetica-Bold
!5	Duplex Roman	Helvetica-Narrow
!6	Complex Roman	Helvetica-Narrow-BoldOblique
!7	Complex Greek	Times-Roman
!8	Complex Italian	Times-BoldItalic
!9	Math Font	Symbol
!10	Special Characters	ZapfDingbats
!11	Gothic English	Courier
!12	Simplex Script	Courier-Oblique
!13	Complex Script	Palatino-Roman
!14	Gothic Italian	Palatino-Italic
!15	Gothic German	Palatino-BoldItalic
!16	Cyrillic	AvantGrande-Book
!17	Triplex Roman	NewCenturySchlbk-Roman
!18	Triplex Italian	NewCenturySchlbk-Bold
!20	Miscellaneous	Undefined
!X	(Back to default)	(Back to default)

```
thisdevice = !d.name
set_plot, 'ps'
device, /zapfchancery, font_size=9, /encapsulated, filename='zaph.ps'
cgplot, cgdemodata(1), font=0, title='ZapfChancery Font', $
        xtitle='Time', ytitle='Signal Strength'
device, /close_file
set_plot, thisdevice
```

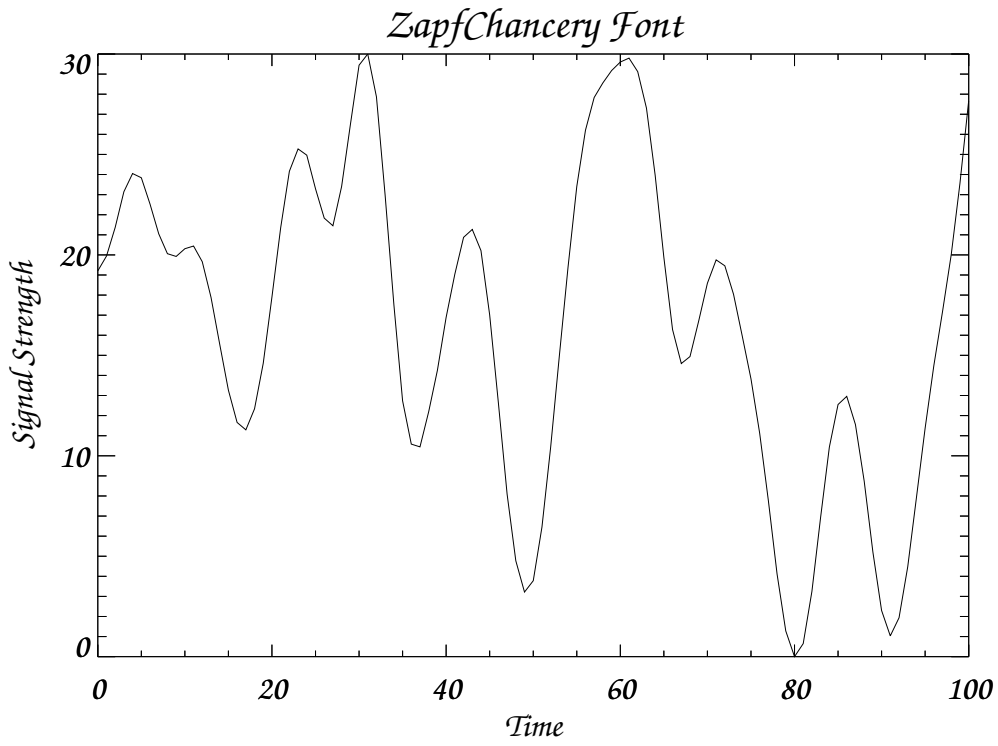

Figure 72: ZapfChancery Font

15.9.3 TrueType Fonts

```
set_plot, 'ps'
device, set_font='Times', /tt_font
!p.font = 1
```

for Bold and Italic fonts

```
device, set_font='Times Bold', /tt_font
device, set_font='Times Italic', /tt_font
device, set_font='Times Bold Italic', /tt_font
```

15.9.4 rotating fonts

```
set_plot, 'x'
cgplot, cgdemodata(1), font=0, xtitle='Good Fonts Here', $
        ytitle='Bad Fonts Here'
```

```
set_plot, 'ps'
device, _extra=pswindow(aspectratio=1.0/2)
!p.multi = [0, 2, 1]
cgplot, cgdemodata(1), font=0, $
        xtitle='Good Fonts', ytitle='And Good Fonts'
cgsurf, cgdemodata(2), xtitle='X Fonts', $
        ytitle='Y Fonts', ztitle='Z Fonts', font=0
!p.multi=0
device, /close_file
cgfixps, 'idl.ps'
set_plot, 'x'
```

```
set_plot, 'ps'
device, _extra=pswindow(aspectratio=1.0/2)
!p.multi = [0, 2, 1, 0, 1]
cgsurf, cgdemodata(2), xtitle='X Fonts', $
        ytitle='Y Fonts', ztitle='Z Fonts', $
        Font=-1, title='Hershey Fonts'
cgsurf, cgdemodata(2), xtitle='X Fonts', $
        ytitle='Y Fonts', ztitle='Z Fonts', $
        Font=1, title='TrueType Fonts'
!p.multi = 0
device, /close_file
cgfixps, 'idl.ps'
set_plot, 'x'
```

15.9.5 Greek characters

1. using embedded font

```
window, /free, xsize=600, ysize=600
showfont, 9, 'Symbol'
```

```
thisdevice = !d.name
set_plot, 'ps'
mu = '!9' + String("155B) + '!X'
cgplot, cgdemodata(1), font=1, $
        xtitle='Wavelength (' + mu + 'm)', charsize=1.0
device, /close_file
set_plot, thisdevice
```

The *Coyote Library* contains a program named *cgGreek* makes it easier for us to use Greek font. You can select lowercase Greek letters by specifying the Greek letter wanted in lowercase letters. You can select uppercase Greek letters either by setting the Capital keyword or by making the first letter of the Greek letter name a capital letter.

```
void = cggreek(/example)
```

2. using *textoidl*

You need to install the TeXtoIDL program.

```
showtex
```

```
cgps_open, 'textoidl.ps'
cgdisplay, 500, 100
astring = textoidl('\Sigma_{n=1}^{\infty} 1/n^2 = \pi^2/6')
xyouts, 0.5, 0.5, align=0.5, charsize=3.0, astring
cgps_close, /png, /width=500
```

$$\Sigma_{n=1}^{\infty} 1/n^2 = \pi^2/6$$

Figure 73: TeXtoIDL

3. using *latexify*

   See Slug's guide.

```
pson, file='latexify.eps', margin=1.0
cgplot, findgen(10), xtitle='xt', charsize=1.5, font=0
psoff
latexify, 'latexify.eps', 'xt', 'Mass [$M_{\odot}$]'
```

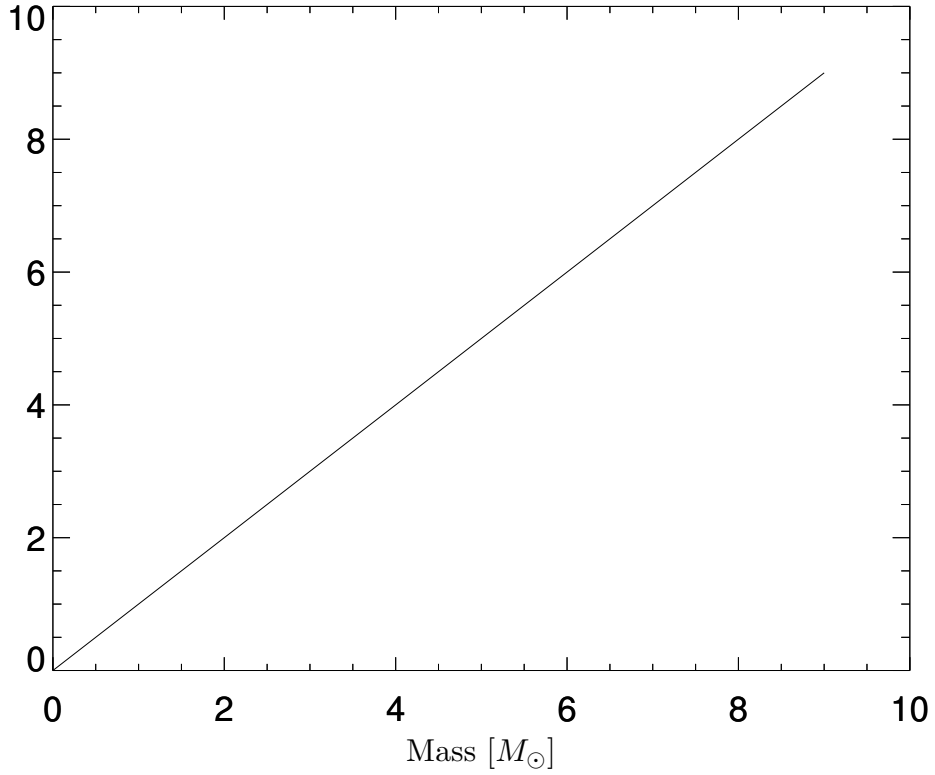

Figure 74: latexify

15.9.6 configuring the PostScript device interactively

```
keywords = psconfig()
thisdevice = !d.name
set_plot, 'ps'
device, _extra=keywords
```

If *psconfig* is called with the *match* keyword set, the initial window configuration will match the aspect ratio of the current graphics window.

```
cgdisplay, 600, 300
keywords = psconfig(/match)
help, keywords, /structure
```

For font information

```
keywords = psconfig(/fontinfo)
```

```
thisdevice = !d.name
thisfont = !p.font
keywords = psconfig(cancel=cancel, /fontinfo, fonttype=1)
if cancel then return
!p.font = keywords.fonttype
set_plot, 'ps'
;; graphics commands here
device, /close_file
set_plot, thisdevice
!p.font = thisfont
```

If you prefer not to use the graphical user interface to *psconfig*,

```
set_plot, 'ps'
device, _extra=psconfig(filename='coyote.ps', /match, /nogui)
```

15.9.7 display images in PostScript

```
thisdevice = !d.name
set_plot, 'ps'
image = congrid(cgdemodata(7), 400, 400)
device, xsize=5, ysize=5, /inches, decomposed=1
tv, image, 0.5, 0.5, xsize=4, ysize=4, /inches
cgplot, findgen(100), /nodata, /noerase, $
        position=[0.1, 0.1, 0.9, 0.9], font=0, charsize=1
device, /close_file
set_plot, thisdevice
```

```
rose = cgdemodata(16)
set_plot, 'ps'
image2d = color_quan(rose, 1, r, g, b)
tvlct, r, g, b
tv, image2d
device, /close_file
```