

IDL: Manipulating Scientific Data Format Files and Data Visualization

Jules Kouatchou and Bruce Van Aartsen

`Jules.Kouatchou@nasa.gov`, `bruce.vanaartsen@nasa.gov`



Goddard Space Flight Center
ASTG
Code 606

2017

- 1 Introduction
- 2 Overview on Arrays and Structure Data Type
- 3 Manipulating Scientific Data Format Files
 - Manipulating netCDF Files
 - Manipulating HDF4 Files
 - Manipulating HDF5 Files
- 4 Plotting with IDL

Training Objectives

We want to introduce:

- Basic concepts of scientific data format files
- Manipulation of netCDF and HDF files
- Data Visualization

Create a File

For each file format, we will show examples on how to:

- Define dimensions
- Define variables
- Add variable and global attributes
- Write data

Read a File

For each file format, we will show examples on how to:

- Read a variable
- Read attributes
- Read a subset of a variable
- Read a file a list all the variables, their dimension, type, etc.

Plotting Data

We will show examples on how to read data from (netCDF, HDF) files and the do following types of plots:

- Contours
- Zonal mean heights
- Various time series

Journaling

```
IDL> journal, 'my_journal.pro'  
IDL> x = -1.0 + 0.1*FINDGEN(21)  
IDL> y = exp(x)  
IDL> plot, x, y  
IDL> journal
```

```
IDL> @my_journal.pro
```

Arrays

General Principles on Arrays

- Any type of variable may be put in an array
- Arrays may have up to 8 dimensions
- Arithmetic operations that are independent for each array element may be performed using a compact syntax instead of loops (faster and cleaner code)
- Arrays are initialized to zero
Example: **rdata = fltarr(360,180)** creates a 360×180 zero-valued floating point array

Array Subscripts

- Array elements are accessed with brackets [], to distinguish from function calls which use parentheses.
- The first element in each dimension is given an index of 0 (not 1)
- To access a range of elements, separate the indices by a colon:
Example: **print, x[3:6]**
- To access all elements in a given dimension, use an asterisk
Example: **print, x[0,*]**

Examples Arrays - I

```

1 a=intarr(100)           ; define a as integer array a[0],...,a[9]
2 a=dblarr(100)           ; double-precision float array = 0.0000
3 a=indgen(20)            ; same thing: a=[0,1,....,19] without a
4
5 b = intarr(3,5)
6 b = dblarr(3,5)
7 c = identity(3) ; identity matrix
8 d = diag_matrix([4,5,6]) ; diagonal matrix
9
10 f = lindgen(3,5) ; array of long integers
11 f = indgen(3,5)  ; array of integers
12 f = findgen(3,5) ; array of floating point numbers

```

Examples of Arrays - II

```
1 ar = [[1,2,3],[4,5,6]] ; integer [3,2] array
2 print,ar[0],ar[0,0] ; mind the virtual finger
3 print,ar[0,*] ; * = all values of this index
4 print,n_elements(ar) ; predict all these
5 print,total(ar) ; for large arrays set /double
6 print,shift(ar,-1)
7 print,transpose(ar)
8 print,reverse(ar)
```

Examples of Arrays - III

```

1 ar=indgen(3,4,5)+1 ; let s say 3x4 px frames in a 5-fr
2 ar3=ar(*,*,2)      ; third movie frame
3 print,total(ar)     ; sum all elements
4 print,total(ar,1)   ; (4,5) row sums=sum over other dimen
5 print,total(ar,2)   ; (3,5) column sums
6 print,total(ar,3)   ; (3,4) frame sums
7 sizear=size(ar)
8 print,sizear ; nr dims, dim1, dim2, dim3, type (integer
9 mean=total(ar,3)/sizear(3) ; temporal mean of this movie
10 xslice=ar[:,0,:]      ; distill (x,t) timeslice at y=0
11 xslice=reform(xslice) ; reform removes degenerate dimens

```

Operations on Arrays - 1

`n_elements()` - number of array elements

`size()` - array size and type info

`reform()` - reduces number of dimensions without changing the total number of elements

`reverse()` - reverses the order of one dimension

`rotate()` - rotates a 1D or 2D array by multiples of 90 degrees

`transpose()` - reflects array elements about a diagonal

`sort()` - returns indices of array elements in ascending order

Operations on Arrays - 2

`min()`, `max()` - minimum and maximum values (and optionally, index)

`mean()` - mean value of array

`variance()` - variance of array values

`stddev()` - standard deviation of array values

`moment()` - mean, variance, skew, kurtosis

`total()` - sum of array values

`median()` - median array value

`invert()` - inverts a square ($n \times n$) array

`round()` - rounds elements to nearest integer

`ceiling()` - smallest integer greater than or equal to its argument

`floor()` - largest integer less than or equal to its argument

Structure Data Type

Definition of Structures

- Structures are a special data type that allows variables of different types and sizes to be packaged into one entity.
- There are two kinds of structures:
 - anonymous structure:** a package of arbitrary variables
 - named structure:** a package of variables that conform to a template created by the user.

Structures are used when it makes sense to collect and store a group of related items (e.g., the name, identification number, and grade for each student in a class).

Anonymous Structure

Created by enclosing variable name/value pairs within curly brackets .

```
1 image = {name:'Test Image', $  
2         valid_range: [0.0, 100.0], $  
3         data:dist(256)}
```

```
IDL>help, image  
IDL>help, image, /structure  
IDL>print, image.name  
IDL>help, image.(1)
```

Scientific Data Format

Advantages of using a standard scientific data format:

- Portability
- Self-describing data (*metadata*)
- Mixed data types
- Widely available read/write software

When creating a scientific data format file, we need to make an extra effort to include enough metadata so that we can understand the file after it is created.

Standard Variables Attributes

Attribute Name	Definition
long_name	A text string that describes a variable in detail.
units	A text string that describes the units of a variable.
valid_range	A two-element array containing the valid minimum and maximum values for a variable (e.g., [-10.0, 250.0]).
scale_factor	A multiplier to be applied to a variable after it is read
add_offset	An offset to be added to a variable after it is read, and after scale_factor (if present) is applied.
_FillValue	A value indicating that no data were written.

Manipulating netCDF Files

Overview of netCDF

- The network Common Data Form (netCDF) was developed by the Unidata Program Center at the University Corporation for Atmospheric Research in Boulder, Colorado.
- The basic building blocks of netCDF files are variables, attributes, and dimensions:
 - 1 Variables are scalars or multidimensional arrays.
 - 2 Attributes contain supplementary information about a single variable (variable attribute) or an entire file (global attribute).
 - 3 Dimensions are *long* scalars that record the size of one or more variables.

Sample Structure of a netCDF File

Name:

```
image.nc
```

```
Dimensions.
```

```
xsize = 1200
```

```
ysize = 600
```

Variables:

```
byte image[xsize, ysize]
```

```
    long_name = 'Imager visible channel'
```

```
    units = 'Counts'
```

```
    valid_range = 0, 255
```

```
double time[ysize]
```

```
    long_name = 'Seconds since 0000 UTC, Jan 1 1970'
```

```
    units = 'seconds'
```

```
    valid_range = 0.0D+0, 10.0D+308
```

Global Attributes:

```
title = 'GOES Image'
```

```
history = 'Created Wed Jul 14 14.15.01 1993'
```

Routines for Writing netCDF Files

NCDF_CREATE - Create a new file that is put into define mode.

NCDF_DIMDEF - Create dimensions for the file

NCDF_VARDEF - Define the variables to be used in the file.

NCDF_ATTPUT - Optionally, use attribute to describe the data

NCDF_CONTROL, /ENDEF - Call NCDF_CONTROL and set the ENDEF keyword to leave define mode and enter data mode.

NCDF_VARPUT - Write the appropriate data to the netCDF file.

NCDF_CLOSE - Close the file.

Writing a Simple file

```
1 ny      = 12
2 nx      = 6
3 data = LINDGEN(nx, ny)
4
5 ncfid    = NCDF_CREATE('simple_xy.nc', /CLOBBER)
6 xid      = NCDF_DIMDEF(ncfid, 'x', nx)
7 yid      = NCDF_DIMDEF(ncfid, 'y', ny)
8 vid      = NCDF_VARDEF(ncfid, 'data', [xid, yid], /LONG)
9 NCDF_CONTROL, ncfid, /ENDEF
10
11 NCDF_VARPUT, ncfid, 'data', data
12
13 NCDF_CLOSE, ncfid
```

Routines for Reading netCDF Files

NCDF_OPEN - Create an existing netCDF file..

NCDF_INQUIRE - Find the format of the netCDF file.

NCDF_DIMINQ - Retrieve the names and sizes of dimensions in the file.

NCDF_VARINQ - Retrieve the names, types, and sizes of variables in the file.

NCDF_ATTNAME - Optionally, retrieve attribute names..

NCDF_ATTINQ - Optionally, retrieve the types and lengths of attributes.

NCDF_ATTGET - Optionally, retrieve the attributes.

NCDF_VARGET - Read the data from the variables.

NCDF_CLOSE - Close the file.

Reading a Simple file

```
1  
2 ncfid = NCDF_OPEN('simple_xy.nc') ; Open file  
3  
4 NCDF_VARGET, ncfid, 'data', data ; Read variable 'data'  
5  
6 NCDF_CLOSE, ncfid ; Close file
```

Reading an Attribute

```
1 varid = NCDF_VARID(ncfid, variable_name)
2 NCDF_ATTGET, ncfid, varid, var_attribute_name, var_attri
3 print, var_attribute_value
4
5 NCDF_ATTGET, ncfid, glob_attribute_name, glob_attribute_v
6 print, glob_attribute_value
```

Reading a Subset of a Variable

```
1 varid = NCDF_VARID(ncfid, variable_name)
2 NCDF_VARGET, ncfid, varid, data, $
3     OFFSET = [...], $
4     COUNT   = [...], $
5     STRIDE   = [...]
```

OFFSET: The first element in each dimension to be read (zero-based; default is [0,0,...,0])

COUNT: The number of elements to be read in each dimension (default is from the current OFFSET to the last element in each dimension)

STRIDE: The sampling interval along each dimension (default is [1,1,...,1], which samples every element)

Exercise on netCDF

Modify the file *NC_printVariablesInfo.pro* to:

- Not list the dimension variables from the list of variables
- List the data type of each variable (before dimensions)

Manipulating HDF4 Files

Overview of HDF4

- The Hierarchical Data Format (HDF) is a data file format designed by the National Center for Supercomputing Applications (NCSA).
- Some features of HDF:
 - 1 Supports a variety of data types.
 - 2 Makes it possible for programs to obtain information about the data from the data file itself, rather than from another source.
 - 3 Standardizes the format and descriptions of many types of commonly used data sets, such as raster images and scientific data..

Commonly used HDF4 Scientific Data Set Routines

Name	Purpose
<code>hdf_sd_start()</code>	Open a HDF file in SDS mode
<code>hdf_sd_end</code>	Close a HDF file in SDS mode
<code>hdf_sd_nametoindex()</code>	Return variable index
<code>hdf_sd_select()</code>	Return variable identifier
<code>hdf_sd_getdata</code>	Read variable data
<code>hdf_sd_endaccess</code>	End access to a variable
<code>hdf_sd_attrfind()</code>	Return variable/global attribute index
<code>hdf_sd_attrinfo</code>	Read variable/global attribute data
<code>hdf_sd_fileinfo</code>	Return file information
<code>hdf_sd_getinfo</code>	Return variable information
<code>hdf_sd_create()</code>	Create a variable
<code>hdf_sd_dimgetid()</code>	Create a dimension
<code>hdf_sd_dimset</code>	Set dimension information
<code>hdf_sd_adddata</code>	Write variable data
<code>hdf_sd_attrset</code>	Write attribute data

Writing a Simple HDF4 File

```
1 ny      = 12
2 nx      = 6
3 data = LINDGEN(nx, ny)
4
5 hdfid    = HDF_SD_START('simple_xy.hdf', /CREATE)
6
7 vid      = HDF_SD_CREATE(hdfid, 'data', [nx, ny], /LONG)
8
9 xid      = HDF_SD_DIMGETID(vid, 0)
10 HDF_SD_DIMSET, xid, NAME = 'x'
11
12 yid      = HDF_SD_DIMGETID(vid, 1)
13 HDF_SD_DIMSET, yid, NAME = 'y'
14
15 HDF_SD_ADDDATA, vid, data
16
17 HDF_SD_ENDACCESS, vid
18 HDF_SD_END, hdfid
```

Reading a Simple HDF4 File

```
1 hdfid = HDF_SD_START('simple_xy.hdf')
2
3 ; Find the index of the sds to read using its name
4 index = HDF_SD_NAMETOINDEX(hdfid, 'data')
5
6 ; Select it
7 vid = HDF_SD_SELECT(hdfid, index)
8
9 ; Get data set information including
10 ; dimension information
11 HDF_SD_GetInfo, vid, name = 'data', natts = num_attributes,
12                                     ndim = num_dims, di
13
14 ; Read the data
15 HDF_SD_GETDATA, vid, data
16
17 HDF_SD_END, hdfid
```

Reading a Subset of a Variable

```
1 index = HDF_SD_NAMETOINDEX(hdfid, variable_name)
2 vid = HDF_SD_SELECT(hdfid, index)
3 HDF_SD_GETDATA, vid, data, $
4     START = [...], $
5     COUNT = [...], $
6     STRIDE = [...]
7 HDF_SD_ENDACCESS, vid
```

START: The first element in each dimension to be read (zero-based; default is [0,0,...,0])

COUNT: The number of elements to be read in each dimension (default is from the current START to the last element in each dimension)

STRIDE: The sampling interval along each dimension (default is [1,1,...,1], which samples every element)

Reading an Attribute from a HDF4 File

```
1 index = HDF_SD_NAMETOINDEX(hdfid, variable_name)
2 vid = HDF_SD_SELECT(hdfid, index)
3 attindex = HDF_SD_ATTRFIND(vid, attribute_name)
4 HDF_SD_ATTRINFO, vid, attindex, data=attribute_value
5 HDF_SD_ENDACCESS, vid
6 print, attribute_value
```

Exercise on HDF4

Modify the file *HDF4_printVariablesInfo.pro* to:

- Not list the dimension variables from the list of variables
- List the data type of each variable (before dimensions)

Manipulating HDF5 Files

Overview of HDF5

- The Hierarchical Data Format (HDF) is a data file format designed by the National Center for Supercomputing Applications (NCSA).
- Some features of HDF5:
 - 1 Can represent very complex data objects and a wide variety of metadata.
 - 2 Portable file format.
 - 3 A rich set of integrated performance features that allow for access time and storage space optimizations.
 - 4 Tools and applications for managing, manipulating, viewing, and analyzing the data in the collection.

HDF5 Data Model

- **Groups** - provide structure among objects
- **Datasets** - where the primary data goes
 - Data arrays
 - Rich set of datatype options
 - Flexible, efficient storage and I/O
- **Attributes** - for metadata

Everything else is built essentially from these parts.

Routines for HDF5 Interfaces

Interface	Routine Class	Example
Attributes	H5A	H5A_CREATE, H5A_OPEN H5A_CLOSE, H5A_READ, H5A_WRITE
Datasets	H5D	H5D_READ
File	H5F	H5F_OPEN
Group	H5G	H5G_CREATE, H5G_OPEN, H5G_CLOSE
Reference	H5R	
Dataspace	H5S	H5S_CLOSE
Datatype	H5T	H5T_CREATE, H5T_CLOSE

Basic IDL HDF5 Functions

```
1 H5F_CREATE (H5F_OPEN)           ; create (open) File
2
3   H5S_CREATE                     ; create dataSpace
4
5       H5D_CREATE (H5D_OPEN)      ; create (open) Dataset
6
7           H5D_READ , H5D_WRITE   ; access Dataset
8
9       H5D_CLOSE                  ; close Dataset
10
11   H5S_CLOSE                     ; close dataSpace
12
13 H5F_CLOSE                       ; close File
```

Writing a Simple HDF5 File

```
1 h5fid      = H5F_CREATE('simple_xy.h5') ; Create the file
2
3 ; Get data type and space needed to create the dataset
4 datatype_id = H5T_IDL_CREATE(data)
5 dataspace_id = H5S_CREATE_SIMPLE(size(data,/DIMENSIONS))
6
7 ; create dataset in the output file
8 dataset_id = H5D_CREATE(h5fid, 'data', datatype_id, $
9                        dataspace_id)
10
11 ; write data to dataset
12 H5D_WRITE,dataset_id,data
13
14 ; close all open identifiers
15 H5D_CLOSE,dataset_id
16 H5S_CLOSE,dataspace_id
17 H5T_CLOSE,datatype_id
18 H5F_CLOSE,h5fid
```

Reading a Simple HDF5 File

```
1 h5fid = H5F_OPEN('simple_xy.h5')
2
3 ; Open the data set
4 vid = H5D_OPEN(h5fid, 'data')
5
6 ; Read the data
7 data = H5D_READ(vid)
8 help, data
9
10 ; Close all open identifiers
11 H5D_CLOSE, vid
12 H5F_CLOSE, h5fid
```

Reading a Subset of a Variable

```
1 h5fid = H5F_OPEN(file_name)
2 vid   = H5D_OPEN(h5fid, 'data')
3 vspace = H5D_GET_SPACE(vid)
4 H5S_SELECT_HYPERSLAB, vspace, START=[...], $
5         COUNT=[...], BLOCK=[...], STRIDE=[...], /RESET
6 rspace = H5S_CREATE_SIMPLE(count)
7 data = H5D_READ(vid, FILE_SPACE=vspace, $
8             MEMORY_SPACE=rspace)
```

START: The first element in each dimension to be read (zero-based; default is [0,0,...,0])

COUNT: The number of elements to be read in each dimension (default is from the current START to the last element in each dimension)

BLOCK: The size of each block, typically a single element at the time.

STRIDE: The sampling interval along each dimension (default is [1,1,...,1], which samples every element)

Reading a Subset of a Variable - II

Assume that we have a 3D array *arr* and we want to extract the entries:

```
arr[3:3:1, 5:49:2, 0:49:3]
```

We then have:

```
1 | START    = [3, 5, 0]
2 | COUNT    = [1, 23, 17]
3 | STRIDE   = [1, 2, 3]
4 | BLOCK    = [1, 1, 1]
```

Reading an Attribute from a HDF5 File

```
1 index = HDF_SD_NAMETOINDEX(hdfid, variable_name)
2 vid = HDF_SD_SELECT(hdfid, index)
3 attindex = HDF_SD_ATTRFIND(vid, attribute_name)
4 HDF_SD_ATTRINFO, vid, attindex, data=attribute_value
5 HDF_SD_ENDACCESS, vid
6 print, attribute_value
```


IDL Utility for HDF5 Tables

http://www.github.com/superchromix/wmb_lib

Plotting

Line Graph: Sine Plot

```
1 x=findgen(101)*(0.01 * 2.0 * !pi)
2 y=sin(x)
3 plot,x,y
```

Overplotting

```
1 x=findgen(101)*(0.01 * 2.0 * !pi)
2 plot,x,sin(x)
3 oplot, x, sin(-x)
4 oplot, x, sin(x)*cos(x)
```

Scatter Plot

```
1 n = 100
2 x=findgen(n)
3 y = x + (20.0 * randomu(-1L, n))
4 plot, x, y, psym=1
```

Specify a Plot Position

The keyword `textitposition` can be used to specify the location for a plot.

```
1 window, /free, xsize=640, ysize=512
2 x = findgen(200) * 0.1
3 plot, x, sin(x), position=[0.10, 0.10, 0.45, 0.90]
4 plot, x, cos(x), position=[0.55, 0.10, 0.90, 0.90], $
5     /noerase
```

```
1 window, /free
2 x = findgen(200) * 0.1
3 pos = getpos(1.0)
4 plot, x, sin(x), position=pos
```

Positioning Multiple Plots

```
1 !p.multi = [0, 2, 2, 0, 0]
2 x = findgen(200) * 0.1
3 plot, x, sin(x)
4 plot, x, sin(x) * x^2
5 plot, x, randomu(1, 200) * x, psym=1
6 plot, x, 4.0 * !pi * x * 0.1, /polar
```

!p.multi is:

- A *long* vector with five elements
- The second element specifies the number of plot columns
- The third element specifies the number of plot rows

To disable multiple plots, simply reset all elements of *!p.multi* to zero.

Plot Customization

```
1 x = findgen(200) * 0.1
2 plot, x, sin(x), $
3     title      = 'SIN(X) vs. X', $
4     subtitle   = 'A sample IDL plot' , $
5     charsize   = 1.25, $
6     font       = 1, $
7     linestyle  = 3, $
8     thick      = 2.0, $
9     psym       = -1
```


Axis Customization Example 1

```
1 x = findgen(200) * 0.1
2 y = sin(x)
3 plot, x, sin(x), xrange=[0,13.5]
4 plot, x, y, xrange=[0,13.5], xstyle=1
5 plot, x, y, xrange=[0,13.5], xstyle=1, $
6     yrange=[-2.5,2.5], ystyle=1
```

Axis Customization Example 2

```
1 t = findgen(11)      ; time
2 a = 9.8              ; acceleration due to gravity
3 v = a * t            ; velocity
4 x = 0.5 * a * t^2    ; distance
5
6 plot, t, x, /nodata, ystyle = 4, $
7     xmargin = [10, 10] , xtitle = 'Time (sec) '
8 axis, yaxis = 0, yrange = [0, 100], /save, $
9     ytitle = 'Velocity (meters/sec, solid line)'
10 oplot, t, v, linestyle = 0
11 axis, yaxis = 1, yrange = [0, 500], /save, $
12     ytitle = 'Distance (meters, dashed line)'
13 oplot, t, x, linestyle = 2
```

Logarithmic Axes

```
1 x = findgen(200) * 0.1 + 1.0  
2  
3 plot, x, x^3, /ylog
```

Titles and Symbols

```
1 x = findgen(100) * 0.1 - 5.0
2 y = 1.0 - exp(-(x^2))
3 title   = '!3C0!D2!N Spectral Absorption Feature!X'
4 xtitle  = '!3Wavenumber (cm!U-1!N)!X'
5 ytitle  = '!3Transmittance!X'
6 plot, x + 805.0, y, title = title, xtitle = xtitle, $
7         ytitle=ytitle
```

Error Plot

```
1 n = 10
2 x = findgen(n)
3 y = randomu(-1L, n) + 10
4 plot, x, y, yrange=[9.5, 11.5 ]
5 err = 0.1
6 err_plot, x, y - err , y+err
```

Contour Plot

```
1 n = 50
2 z = randomu(-100L, n, n)
3 for i = 0, 4 do z = smooth(z, 15, /edge)
4 z = (z - min(z)) * 15000.0 + 100.00 ; total ozone
5 x = findgen(n) - 100.0 ; longitude
6 y = findgen(n) + 10.0 ; latitude
7 levels = [150, 200, 250, 300, 350, 400, 450, 500]
8 c_labels = [0, 1, 0, 1, 0, 1, 0, 1]
9 contour, z, x, y, levels = levels , c_labels=c_labels
```

Filled Contour Plot

```
1 levels = [150, 200, 250, 300, 350, 400, 450, 500]
2 nlevels = n_elements(levels)
3 ncolors = nlevels + 1
4 bottom = 1
5 c_levels = [min(z), levels, max(z)]
6 c_labels = [0, replicate(1, nlevels), 0]
7 c_colors = indgen(ncolors) + bottom
8 loadct, 33, ncolors=ncolors, bottom=bottom
9 contour, z, x, y, $
10   levels=c_levels, c_colors=c_colors, /fill, $
11   xstyle=1, ystyle=1, title='Simulated Total Ozone', $
12   xtitle='longitude', ytitle='Latitude'
13 contour, z, x, y, $
14   levels = c_levels, c_labels=c_labels, /overplot
```

Saving a Plot in a png File

```
1 SET_PLOT, 'png'
2 DEVICE, /DECOMPOSED, /COLOR
3
4 v = findgen(41) * 0.5 - 10.0
5 x = rebin(v, 41, 41, /sample)
6 y = rebin(reform(v, 1, 41), 41, 41, /sample)
7 r = sqrt(x^2 + y^2) + 1.0e-6
8 z = sin(r) / r
9 shade_surf, z, x, y
10
11 WRITE_PNG, 'output.png', TVRD(TRUE = 1)
12 DEVICE, /CLOSE
```


References I



Kenneth P. Bowman.

An Introduction to Programming with IDL: Interactive Data Language.

Elsevier Inc, 2005.



David W. Fanning.

Traditional IDL Graphics.

Coyote Book Publishing, 2011.



Lilian Gumley.

Practical IDL Programming.

Morgan Kaufmann, 2001.