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

June 20, 2017

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

Materials

The presentation slides are in the *Slides/* directory and the scripts for:

Beginners are in the directories:

```
BasicSyntax/     If_Loops/
Array/           Structures/
Proc_Function/   File_IO/       Plots/
```

This training are in the directories:

```
netCDF/   HDF4/    HDF5/
```

Data Files

In case you downloaded the "LARGE" file containing the data files, make sure that you move:

- The two netCDF files into the directory: *netCDF/ncFiles/*
- The two HDF4 files into the directory: *HDF4/hdFiles/*

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

```
a=intarr(100)      ; define a as integer array a[0],...,a[9
a=dblarr(100)      ; double-precision float array = 0.0000
a=indgen(20)       ; same thing: a=[0,1,....,19] without a

b = intarr(3,5)
b = dblarr(3,5)
c = identity(3) ; identity matrix
d = diag_matrix([4,5,6]) ; diagonal matrix

f = lindgen(3,5) ; array of long integers
f = indgen(3,5)  ; array of integers
f = findgen(3,5) ; array of floating point numbers
```

Examples of Arrays - II

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

Examples of Arrays - III

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

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 .

```
image = {name:'Test Image', $
         valid_range: [0.0, 100.0],  $
         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 sca1e_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

```
ny    = 12
nx    =  6
data = LINDGEN(nx, ny)

ncfid    = NCDF_CREATE('simple_xy.nc', /CLOBBER)
xid  = NCDF_DIMDEF(ncfid, 'x', nx)
yid  = NCDF_DIMDEF(ncfid, 'y', ny)
vid  = NCDF_VARDEF(ncfid, 'data', [xid, yid], /LONG)
NCDF_CONTROL, ncfid, /ENDEF

NCDF_VARPUT, ncfid, 'data', data

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

```

ncfid = NCDF_OPEN('simple_xy.nc')   ; Open file

NCDF_VARGET, ncfid, 'data', data    ; Read variable 'data

NCDF_CLOSE, ncfid                   ; Close file
```

Reading an Attribute

```
varid = NCDF_VARID(ncfid, variable_name)
NCDF_ATTGET, ncfid, varid, var_attribute_name, var_attri
print, var_attribute_value

NCDF_ATTGET, ncfid, glob_attribute_name, glob_attibute_va
print, glob_attibute_value
```

Reading a Subset of a Variable

```
varid = NCDF_VARID(ncfid, variable_name)
NCDF_VARGET, ncfid, varid, data, $
        OFFSET = [...], $
        COUNT  = [...], $
        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
hdf_sd_start()	Open a HDF file in SDS mode
hdf_sd_end	Close a HDF file in SDS mode
hdf_sd_nametoindex()	Return variable index
hdf_sd_select()	Return variable identifier
hdf_sd_getdata	Read variable data
hdf_sd_endaccess	End access to a variable
hdf_sd_attrfind()	Return variable/global attribute index
hdf_sd_attrinfo	Read variable/global attribute data
hdf_sd_fileinfo	Return file information
hdf_sd_getinfo	Return variable information
hdf_sd_create()	Create a variable
hdf_sd_dimgetid()	Create a dimension
hdf_sd_dimset	Set dimension information
hdf_sd_adddata	Write variable data
hdf_sd_attrset	Write attribute data

Writing a Simple HDF4 File

```
ny   =  12
nx   =   6
data = LINDGEN(nx, ny)

hdfid    = HDF_SD_START('simple_xy.hdf', /CREATE)

vid  = HDF_SD_CREATE(hdfid, 'data', [nx, ny], /LONG)

xid  = HDF_SD_DIMGETID(vid, 0)
HDF_SD_DIMSET, xid, NAME = 'x'

yid  = HDF_SD_DIMGETID(vid, 1)
HDF_SD_DIMSET, yid, NAME = 'y'

HDF_SD_ADDDATA, vid, data

HDF_SD_ENDACCESS, vid
HDF_SD_END, hdfid
```

Reading a Simple HDF4 File

```
hdfid = HDF_SD_START('simple_xy.hdf')

; Find the index of the sds to read using its name
index = HDF_SD_NAMETOINDEX(hdfid, 'data')

; Select it
vid = HDF_SD_SELECT(hdfid, index)

; Get data set information including
; dimension information
HDF_SD_GetInfo, vid, name = 'data', natts = num_attribut
                                     ndim = num_dims, di

; Read the data
HDF_SD_GETDATA, vid, data

HDF_SD_END, hdfid
```

Reading a Subset of a Variable

```
index = HDF_SD_NAMETOINDEX(hdfid, variable_name)
vid = HDF_SD_SELECT(hdfid, index)
HDF_SD_GETDATA, vid, data, $
        START  = [...], $
        COUNT  = [...], $
        STRIDE = [...]
 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

```
index = HDF_SD_NAMETOINDEX(hdfid, variable_name)
vid = HDF_SD_SELECT(hdfid, index)
attindex = HDF_SD_ATTRFIND(vid, attribute_name)
HDF_SD_ATTRINFO, vid, attindex, data=attribute_value
HDF_SD_ENDACCESS, vid
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

```
H5F_CREATE (H5F_OPEN)           ; create (open) File

  H5S_CREATE                    ; create dataSpace

     H5D_CREATE (H5D_OPEN)      ; create (open) Dataset

        H5D_READ, H5D_WRITE     ; access Dataset

     H5D_CLOSE                  ; close Dataset

  H5S_CLOSE                     ; close dataSpace

H5F_CLOSE                       ; close File
```

Writing a Simple HDF5 File

```
h5fid    = H5F_CREATE('simple_xy.h5') ; Create the file

; Get data type and space needed to create the dataset
datatype_id  = H5T_IDL_CREATE(data)
dataspace_id = H5S_CREATE_SIMPLE(size(data,/DIMENSIONS))

; create dataset in the output file
dataset_id = H5D_CREATE(h5fid, 'data', datatype_id, $
                        dataspace_id)

; write data to dataset
H5D_WRITE,dataset_id,data

; close all open identifiers
H5D_CLOSE,dataset_id
H5S_CLOSE,dataspace_id
H5T_CLOSE,datatype_id
H5F_CLOSE,h5fid
```

Reading a Simple HDF5 File

```
h5fid = H5F_OPEN('simple_xy.h5')

; Open the data set
vid = H5D_OPEN(h5fid, 'data')

; Read the data
data = H5D_READ(vid)
help, data

; Close all open identifiers
H5D_CLOSE, vid
H5F_CLOSE, h5fid
```

Reading a Subset of a Variable

```
h5fid = H5F_OPEN(file_name)
vid   = H5D_OPEN(h5fid, 'data')
vspace = H5D_GET_SPACE(vid)
H5S_SELECT_HYPERSLAB, vspace, START=[...], $
        COUNT=[...], BLOCK=[...], STRIDE=[...], /RESET
 rspace = H5S_CREATE_SIMPLE(count)
 data = H5D_READ(vid, FILE_SPACE=vspace, $
                 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:

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

Reading an Attribute from a HDF5 File

```
index = HDF_SD_NAMETOINDEX(hdfid, variable_name)
vid = HDF_SD_SELECT(hdfid, index)
attindex = HDF_SD_ATTRFIND(vid, attribute_name)
HDF_SD_ATTRINFO, vid, attindex, data=attribute_value
HDF_SD_ENDACCESS, vid
print,  attribute_value
```

IDL Utility for HDF5 Tables

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

Plotting

Line Graph: Sine Plot

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

Overplotting

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

Scatter Plot

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

Specify a Plot Position

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

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

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

Positioning Multiple Plots

```
!p.multi = [0, 2, 2, 0, 0]
x = findgen(200) * 0.1
plot, x, sin(x)
plot, x, sin(x) * x^2
plot, x, randomu(1, 200) * x, psym=1
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

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

Axis Customization Example 1

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

Axis Customization Example 2

```
t = findgen(11)      ; time
a = 9.8              ; acceleration due to gravity
v = a * t            ; velocity
x = 0.5 * a * t^2    ; distance

plot, t, x, /nodata, ystyle = 4, $
      xmargin = [10, 10] , xtitle = 'Time (sec) '
axis, yaxis = 0, yrange = [0, 100], /save, $
      ytitle = 'Velocity (meters/sec, solid line)'
oplot, t, v, linestyle = 0
axis, yaxis = 1, yrange = [0, 500], /save, $
      ytitle = 'Distance (meters, dashed line)'
oplot, t, x, linestyle = 2
```

Logarithmic Axes

```
x = findgen(200) * 0.1 + 1.0

plot, x, x^3, /ylog
```

Titles and Symbols

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

Error Plot

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

Contour Plot

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

Filled Contour Plot

```
levels = [150, 200, 250, 300, 350, 400, 450, 500]
nlevels = n_elements(levels)
ncolors = nlevels + 1
bottom = 1
c_levels = [min(z), levels, max(z)]
c_labels = [0, replicate(1, nlevels), 0]
c_colors = indgen(ncolors) + bottom
loadct, 33, ncolors=ncolors, bottom=bottom
contour, z, x, y, $
  levels=c_levels, c_colors=c_colors, /fill, $
  xstyle=1, ystyle=1, title='Simulated Total Ozone', $
  xtitle='longitude', ytitle='Latitude'
contour, z, x, y, $
    levels = c_levels, c_labels=c_labels, /overplot
```

Plots to be Presented

We will open netCDF files and read data to have the following types of plots:

- Contour
- Colorbar
- Map projections
- Selecting a region
- Zonal mean heights
- Wind Vectors
- Animation

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.