Tutorial 4: NCL part 1: programming

The NCAR Command Language (NCL) is a programming language

(The NCAR Command Language (NCL), a product of the Computational & Information Systems Laboratory at the National Center for Atmospheric Research (NCAR) and sponsored by the National Science Foundation, is a free interpreted language designed specifically for scientific data processing and visualization.

NCL has robust file input and output. It can read and write netCDF-3, netCDF-4 classic (as of version 4.3.1), HDF4, binary, and ASCII data, and read HDF-EOS2, GRIB1, GRIB2 (as of version 4.3.0), and OGR files (shapefile, MapInfo, GMT, TIGER) (as of version 5.1.1). The graphics are world class and highly customizable. Almost everything can be plotted (time series, maps, contours...).

Learning to use NCL

NCL has three distinct areas. The first is file I/O. NCL has unique syntax that supports directly referencing variables in data files. Not only can the whole variable be read or written to, but portions of data can be accessed as well using NCL's file variable subscripting rules. Unique syntax for accessing additional file variable information, called metadata, is also useful. This additional information is often grid coordinate information, units, missing values, etc.

Second, there are the data processing features of NCL. The difficulty in learning this area is dependent on the complexity of the user's data processing requirements. Differences between fields on identical grids and averages are fairly straightforward, and many examples exist. However, learning to write efficient NCL data processing code requires a fair amount of understanding of the language.

Finally, there is the graphics component. The syntax for the graphics commands are simple, but the use of them can be complex. NCL uses the NCAR Graphics High Level Utilities Library (HLUs) to configure graphical output.

More details (Help and example):

NCL website: http://www.ncl.ucar.edu/

General documentation: http://www.ncl.ucar.edu/Document/

Reference Manual (online): http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/

Language Manual (pdf): http://www.ncl.ucar.edu/Document/Manuals/language_man.pdf

Graphics Manual (pdf): http://www.ncl.ucar.edu/Document/Manuals/graphics_man.pdf

Most of the basic examples: http://www.ncl.ucar.edu/Applications/

Execute a script:

Type “**ncl script.ncl**” in a linux terminal

You can also type “**ncl script.ncl > output.txt**”

This way all printed informations on screen are stored in the text file “output.txt”

Generic script snapshot:

;;;;; is the comment character (after this whatever you write does not matter)

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/shea_util.ncl"

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"

;; Then you can load other functions if you defined them

begin ; the script need to start with “begin” and end by “end”

;;

;Defining parameters and constants

;;

;[Command and program instructions: reading / writing netcdf files, applying function, computation]

;;

;[Graphic instructions: creating outputs + defining the graphic resources]

;;

end

Print informations and exit:

print(var) ; This prints the variable var

printVarSummary(var) ; This prints informations (coordinates, dimension, attributes) about var

exit ; This stops the program here, really usefull to debug

Printing informations: http://www.ncl.ucar.edu/Document/Functions/printing.shtml

I/O: Read and write a netcdf file using the addfile function: an example

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/shea_util.ncl"

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Read Netcdf file and display informations

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

begin

;;;;;;;;;;;;;;;;;;;;;;;;;

; Read the file

;;;;;;;;;;;;;;;;;;;;;;;;;

; First define a string which defines the file location

fileinobs="/home/caminade/Obs/CMAP/pr_1m_197901_200707_CMAP.nc"

print("Reading "+fileinobs+"") ;This will print on screen: Reading
/home/caminade/Obs/CMAP/pr_1m_197901_200707_CMAP.nc

f=addfile(fileinobs,"r") ; use the addfile function to point out the file in read mode

fld = f->pr ; Read the variable pr in the netcdf file and assign it to the variable fld

; Then to print information about this variable:

printVarSummary(fld) ; VERY usefull function, this will print in the linux terminal various informations

;;;;;;;;;;;;;;;;;;;;;;;;;;

; Read dimensions

;;;;;;;;;;;;;;;;;;;;;;;;;;

dime=dimsizes(fld) ; Assigne the matrix dimension to fld

dtime=dime(0) ; first element of the dime array is the time dimension

dlat=dime(1) ; 2nd element of the dime array is the latitude dimension

dlon=dime(2) ; 3rd element of the dime array is the longitude dimension

delete (dime) ; delete the dime variable in memory

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

; **Manipulate the original array and write the output in a file**

;;;

fld_new=fld*30. ;convert mm/day to mm/month

fld_new@units="mm/month" ; modify the unit attribute to mm/month (attributes refered as to @ and their ;type is string)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; **Write the file**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

fileoutobs="/home/caminade/Dave/NCL_tutorial/test.nc"

system ("rm "+fileoutobs) ; system can call a linux function within NCL, like spawn in ferret, thus here remove ;the file first if it exists

write_file= addfile(fileoutobs,"c") ; use the addfile function to point out the file in create mode

write_file->pr=fld_new ; use fld_new to write the variable pr in the outfile

end

Example available at /home/caminade/Dave/NCL_tutorial/ read_write_file_example.ncl

Note that you can also read ASCII, GRIB etc, all functions and examples are available at: http://www.ncl.ucar.edu/Applications/list_io.shtml

Variable, ordering, coordinates, attributes, types and creation

Types

NCL works with types like character, string, short, integer, float and double. To check the type of a variable use the printVarSummary or the typeof function.

printVarSummary(var)

print(typeof(var))

Type can be converted from one to another using the type converter functions

Variable creation

Single constant values:

A=2. ; assign 2 to A, the point specifies the float format

A=”toto” ; assign the string “toto” to A

A=True ; define A as a Boolean (True)

Vector and Matrix:

A=(/1,2,3,4,5,6,7,8,9/) ; assign a 1D vector with the integer inputs

A=(/"JANUARY", "FEBRUARY", "MARCH"/) ; assign a 1D vector with the string inputs

dlat=64

dlon=128

A =new((/dlat,dlon/),"float") ; create a 64x128 float matrix, see the new function for further details

A= fspan(0, 100, 11) ; creates a float vector starting at 0 going to 100 with 11 elements

A=ispan(1, 10, 1) ; creates an integer vector starting at 0 going to 10 with a stride of 1

More details: Array creators and Array Manipulation

Basic Math manipulation / Algebraic

- Negation
* Multiplication
% Modulus
+ Addition
^ Exponentiation
/ Division
Matrix multiplication

A=2.

B=A^2 ; power of 2

C=sqrt(A) ; sqrt means root mean square

Basic mathematic functions like cos, sin, abs, tan... works just beware the radian / degree convention

D=1e6 ; basic scientific notation here one million

String manipulation

This can be usefull if you try to read several files etc

You can cat string variables by using +

Example

variable="pr"

date1=197901

date2=200707

freq="1m"

infile=variable+"_"+freq+"_"+date1+"_"+date2+"_CMAP.nc"

print(infile) ; this will print pr_1m_197901_200707_CMAP.nc

Variable sub setting and indices

BE CARREFULL NCL starts the indices at 0

A=ispan(1, 10, 1) ; define an integer 1D vector

print(A(0)) ; will print 1, first element

print(A(dimsizes(A)-1)) ; will print 10, the last element

Let's come back to the former example after printVarSummary(fld)

```
Variable: fld
Type: float
Total Size: 14224896 bytes
        3556224 values
Number of Dimensions: 3
Dimensions and sizes:   [time | 343] x [lat | 72] x [lon | 144]
Coordinates:
        time: [17338824..17588616]
        lat: [88.75..-88.75]
        lon: [1.25..358.75]
Number Of Attributes: 15
 long_name :   Average Monthly Rate of Precipitation
 valid_range : (  0, 70 )
 units :      mm/day
 add_offset :   0
 scale_factor :       1
 actual_range :       (  0, 56.54 )
 missing_value :      -9.96921e+36
 precision :   2
 least_significant_digit :    2
 var_desc :    Precipitation
 dataset :     CPC Merged Analysis of Precipitation Standard
 level_desc :  Surface
 statistic :   Mean
 parent_stat : Mean
 _FillValue :  -9.96921e+36
```

fld is a 343 time x 72 latitude x 144 longitude matrix, with the related time, lat and lon coordinates and several attributes.

toto=fld(0:11,:,:) ; will get the first 12 elements of the time dimension, namely the 1st year (monthly ;means)

toto= fld(time|0:11,lat|:,lon|:) ; same command but here you specify the dimensions

toto= fld(lat|:,lon|:, time|0:11) ; same as before, but here you reorder the matrix per lat x lon x time

toto=fld(:,{-30:30},:) ; subset the latitude between **the values** 30S and 30N

toto=fld(:,::-1,:) ; this changes the latitude ordering from descending to ascending order

toto=fld(:,:,::2) ; this will subset the initial array every 2 points in longitude (stride subset ::stride)

Variable and coordinate manipulations

latitude=fld**&**lat ; will assign the coordinate lat related to fld to the variable latitude

longitude=fld**&**lon ; will assign the coordinate lon related to fld to the variable longitude

time=fld**&**time ; will assign the coordinate lon related to fld to the variable longitude

fld**!0**=”T” ; this will rename the time coordinate dimension to T

fld**!1**=”latitude” ; ” ; this will rename the lat coordinate dimension to latitude

fld**!2**=”l;ongitude” ; ” ; this will rename the lon coordinate dimension to longitude

Attributes

You can create an attribute if it does not exist:

A=ispan(1, 10, 1) ; define an integer 1D vector

A@units=”mm/day” ; assign the string mm/day as an attribute

_FillValue:

Back to the former example: fld

The attribute _FillValue is VERY IMPORTANT. This defines the missing value in your field / matrix. If this is defined properly, NCL will not take this into account for complex calculation! Always be careful with this attribute.

This attribute is generally properly defined in the netcdf file. However sometimes a missing_value attribute is also possible. In this case assign the _FillValue attribute to the missing_value one:

toto@_FillValue=toto@missing_value

delete(toto@missing_value)

toto@_FillValue=-9999. ; assign the value -9999. As missing / flagged value, this way this is not considered in the calculation. This is usefull as you can define mask etc etc

Loops, if statements etc

Do loops:

“do loops" in NCL are the same as those in Fortran except that the optional stride must always be positive. The components of the loop include the familiar identifier *(do n...)*, a scalar start expression *(do n=0...)* , a scalar end expression *(do n = 0, nfiles-1)*, and, optionally, a stride *do n=0, nfiles-1,4)*. **Note: there is a space between the "end" and the "do" in NCL:** *(end do)*! Second note: remember that array indexing in NCL begins with a 0 instead of a 1.

NCL also accepts "do while" loops:

```
i = 0
do while(i.le.n)
   .
   .
   .
   i=i+1
end do
```

There are two ways of breaking out of a loop. *break* causes the loop to abort and proceed to the statement after the *end do* statement, and *continue* causes the loop to proceed to the next iteration.

Since NCL is an interpreted language, it is best to avoid do loops as much as possible. They can cause considerable slow downs. Small loops should not be a problem.

Logical Expressions:

NCL's logical expressions are similar to F77/F90

- *.le.* Less than or equal
- *.ge.* Greater than or equal
- *.ne.* Not equal
- *.and.* And
- *.or.* Or
- *.lt.* Less than
- *.gt.* Greater than
- *.eq.* Equal to
- *.xor.* Exclusive or
- *.not.* Not

If statements:

If (statement) then
else
end if

Example:
A= 2.
B= 1.
If (A.gt.B) then

```
        print("A is bigger than B")
else
        print("A is lower than B")
end if
```

where:

This performs array assignments based on a conditional array.

where(condt_expression, true_value, false_value)

example:

mask rainfall value below the 1mm threshold

fld_thresh=where(fld.gt.1., fld,fld@_FillValue)

ind:

Returns the indices where the input is True.

Example:

```
; Create sample array
  a = (/1,2,3,4,5,5,4,3,2,1,1,2,3,4,5/)
; Assign missing value
  a@_FillValue = 5
; Assigns 0 to locations that are missing values.
  a(ind(ismissing(a))) = 0;
; Performs an expression on non-zero values
  a(ind(a.ne.0)) = -a(ind(a.ne.0)) * 2 + 1
```

Example available at /home/caminade/Dave/NCL_tutorial/ algebraic_examples.ncl

The basic language syntax documentation:

http://www.ncl.ucar.edu/Document/Language/index.shtml

Few Usefull functions

There are legions of functions in NCL. They can be accessed by alphabetical order here or by categories. In the following find enclosed the most basic ones with few illustration examples:

dim_avg_Wrap: Computes the average of a variable's rightmost dimension at all other dimensions and retains metadata (associated coordinates and variables).

```
fldclim=dim_avg_Wrap(fld(lat|:,lon|:,time|:)) ; compute the mean
climatology over the time dimension
```

dim_stddev_Wrap: Computes the population standard deviation of a variable's rightmost dimension at all other dimensions and retains metadata.

```
fldstd=dim_stddev_Wrap(fld(lat|:,lon|:,time|:)) ; compute the standard
deviation over the time dimension
```

dim_max: Finds the maximum of a variable's rightmost dimension at all other dimensions.

```
Create a variable (q) of size (3,5,10) array. Then determine the maximum of
the rightmost dimension.

    q    = random_uniform(-20,100,(/3,5,10/))
    qMax = dim_max(q)    ;==>  qMax(3,5)

the same for dim_min and dim_median
```

lonFlip: Reorders the longitude coordinate variable. The rightmost dimension must be the longitude dimension, which must be global and without a cyclic point. Currently, the size of the longitude dimension must be an even number.

```
fld_flip=lonFlip(fld(time|:,lat|:,lon|:)); reorder the longitude between -
180 and 180 if it is originally defined to range between 0 and 360.
```

mask: Masks a multi-dimensional array against another given a single mask value.

Assume that *oro* is a two-dimensional array dimensioned *lat* x *lon* defining land and ocean points. When *mvalue*=1, the land is masked out leaving the ocean; similarly, when *mvalue*=0, the ocean is masked out leaving the land. Assume *x* is a two-dimensional array also dimensioned *lat* x *lon*. Then:

```
    fldLand  =  mask(fld, oro, 1)
    fldOcean =  mask(fld, oro, 0)
```

month_to_seasonN: Computes a user-specified list of three-month seasonal means (DJF, JFM, FMA, MAM, AMJ, MJJ, JJA, JAS, ASO, SON, OND, NDJ).

```
season=(/"DJF", "MAM","JJA","SON"/); typical seasons

fldseas= month_to_seasonN(fld,season)

fldseas will be a matrix: (Nseas,time/12,lat,lon) thus

Nseason x Nyear x nlat x nlon
```

Listing of NCL Functions by Category

Alphabetical listing | Function type listing | Browsable listing

Visualization: Graphics routines Color Object manipulation Workstation	**General NCL routines:** Array creation, manipulation, query String System Type conversion Variable query, manipulation
Earth Science: Climatology CCSM Date Lat/lon functions Metadata/missing values Meteorology Oceanography RIP functions WRF functions	**Math and statistics:** General applied math Cumulative distribution functions Empirical orthogonal functions Interpolation Random number generators Regridding Singular value decomposition Spherical harmonics
Input/output: File input/output Printing	