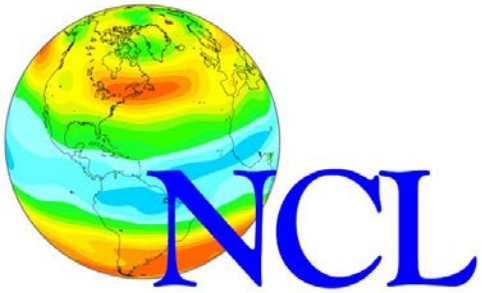
Introduction to the NCAR Command Language



Mattia Righi
DLR – Institut für Physik der Atmosphäre

Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

May 2015– Based on NCL version 6.3.0

Outline

1. NCL as a programming language (syntax, variables, arrays, functions, file input/output etc.)
2. NCL graphics (with examples)
3. Practical exercises

Why NCL?

- Easy to learn: programming syntax similar to other high-level languages (Fortran, C, IDL)
- Excellent on-line documentation (http://www.ncl.ucar.edu/), including manuals, examples and mailing-lists to submit specific problems
- Huge amount of built-in functions for statistics, advanced math and geo-scientific data analysis
- Support for many data formats (including NetCDF, HDF, GRIB, ASCII, binary...)
- Highly-flexible and intuitive routines for high-quality graphics
- Data analysis and data plotting in a single environment
- Free and open source

Download and install NCL

- Download the latest version at http://www.earthsystemgrid.org/
- Select the latest version (6.3.0) and the **precompiled binaries (!)** (much easier to install, works on most Linux distributions)
- To check the right binaries for your Linux system http://www.ncl.ucar.edu/Download/linux.shtml
- Follow the instructions at http://www.ncl.ucar.edu/Download/install.shtml
- It can also be installed on Windows (using the Cygwin emulator or the VirtualBox) and on MAC
  http://www.ncl.ucar.edu/Download/cygwin.shtml
  http://www.ncl.ucar.edu/Download/macosx.shtml

Getting started

Two possibilities:

- Interactive command line: every command is executed immediately as it is typed (for quick operations):

```
ncl
```

- Batch command: write a script using a text editor (e.g., emacs) and execute it:

```
ncl myscript.ncl
```

Arguments can be passed upon call

```
ncl x=5 myscript.ncl
```

Organize your script. Usually three things are needed:

- Read data: variables, coordinates, attributes
- Process data: regridding, average, unit conversions etc.
- Plot data: choose plot type, set plot options, draw the plot

The NetCDF format

- NetCDF (Network Common Data Form) is a machine-independent data format that supports the creation of array-oriented scientific data
- A key feature of the NetCDF (.nc) format is metadata: attributes, named dimensions, coordinate arrays associated with the data
- A special attribute is **`_FillValue`**, indicating a variable's missing value
- To see all the metadata from a NetCDF file, type:

```
ncdump -h filename.nc
ncl_filedump filename.nc
```

needs NetCDF lib installed

- To open a NetCDF file (graphic visualization), type:

```
ncview filename.nc
```

external software tool

- NCL variables are based on the NetCDF data structure!

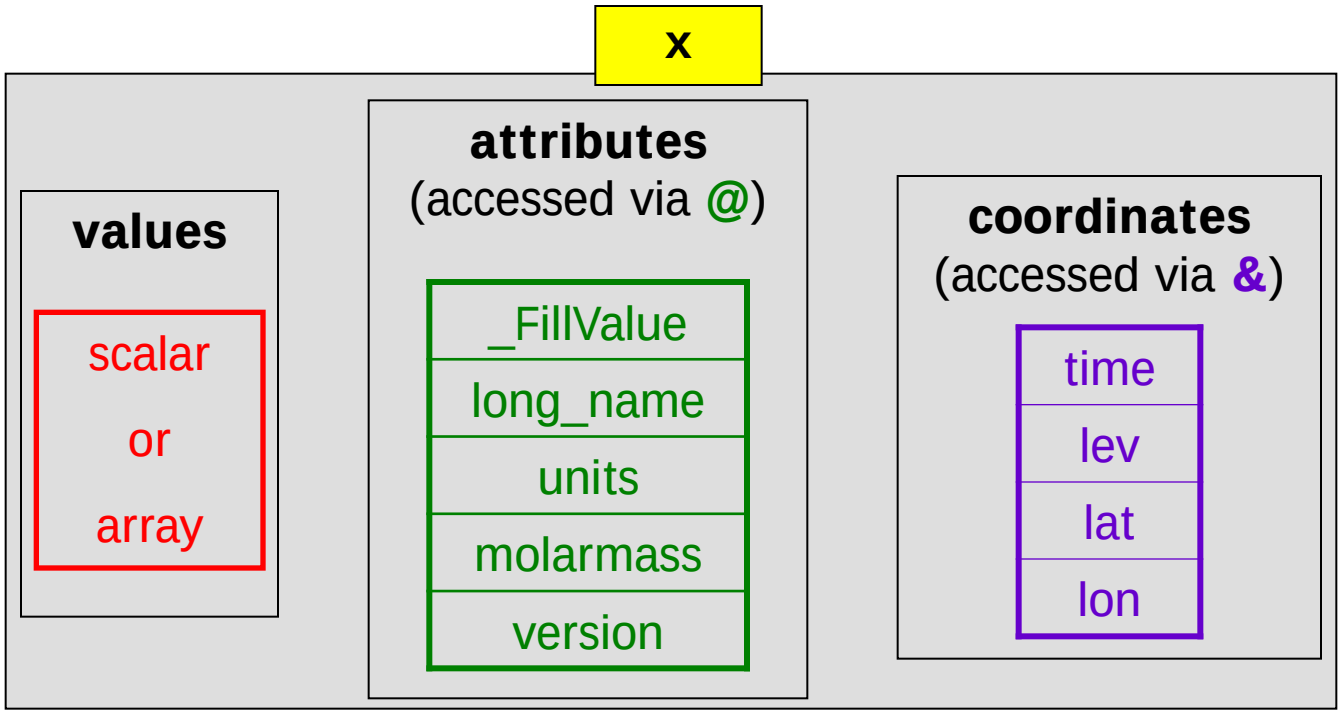
NetCDF/NCL variable model

- Reading a variable (**varname**) from a NetCDF file (**filename.nc**) is very easy:

```
f = addfile("filename.nc","r")
x = f->varname
printVarSummary(x)
```

"r" for reading

- NCL reads values, attributes and coordinates as a single object



Variable types

Type	Category	Size	Min	Max	_FillValue	Suffix
integer	numeric	32 bits	-2147483648	+2147483647	-2147483647	i
float	numeric	32 bits	+/-1.17549e-38	+/-1.70141e+38	9.96921e+36	-
double	numeric	64 bits	+/-2.2250e-308	+/-8.9884e+307	9.96921e+36	D or d
short	numeric	16 bits	-32768	32767	-32767	h
logical	-	-	False	True	_Missing	-
string	-	-	-	-	"missing"	-
graphic	-	-	-	-	-1	-
file	-	-	-	-	-1	-

Other types are available (long, ulong, uint, ushort, byte, character, etc.)

Arithmetic overflow/underflow are not always reported as error to the user.
The effect of such errors is unpredictable!

Basic arithmetic operators

Arithmetic operators in order of precedence:

-	Negative	Highest precedence! x = -3^2 = (-3)^2 = 9
^	Exponent	Imaginary numbers not supported yet!
*	Multiply	No restrictions
/	Divide	If both operands are integer, decimal truncated
%	Modulus	Integer remainder of integer division
#	Matrix multiplication	Dot product of 1-D or 2-D arrays
+	Plus	Also concatenates strings
-	Minus	No restrictions

Variable assignment: scalar

- If variables are mixed, the "highest" type is used:

```
i = 5/2 = 2          integer
x = 5/2. = 2.5       float
```

- Once defined, a variable cannot be changed to higher type:

```
x = 1.5          float
x = 1520         ok, still a float
x = 1.5d2        error (Assignment type mismatch)
```

- Iin this case use delete:

```
delete(x)
x=1.5d2
```

- or the reassignment operator (no need to delete):

```
x:=1.5d2
```

- There are many type-conversion functions available (http://www.ncl.ucar.edu/Document/Functions/type_convert.shtml):

```
fx = tofloat(x) = 1.5
ix = toint(x) = 1
```

Variable assignment: array

- Arrays in NCL are row-major (rightmost dimension varies fastest), like C and IDL
- Arrays can be defined using **(/…/)**:

```
y = (/-5., -2., 3./)                          float (3 elements)
months = (/"Jan","Feb","Mar","Apr"/)          string (4 elements)
z = (/(/1.5d,2.0/),(/3.,5./),(/9,2/)/)        double (3×2 elements)
```

- Or using the **new** statement:

```
y = new(dimension, type)
y = new(5, integer)
y = new((/3, 4, 1/), float)
```

- **_FillValue** is assigned by default but can be changed (or not assigned):

```
y = new((/3,4,1/), float, 1.e20)
y = new((/3,4,1/), float, "No_FillValue")     not recommended
```

- The function **dimsizes** gives the size of each dimension (from left to right):

```
print(dimsizes(y))     should give 3, 4, 1
```

- Use **printVarSummary** to check:

```
printVarSummary(y)
```

Missing values

- Missing values are defined in NetCDF/NCL using the special attribute **_FillValue**
- Most NCL built-in functions recognizes and ignores missing values
- For example, the **dim_avg** function computes the average of all elements in an array:

```
x = (/1., 5., 8., -999., 10./)
print(dim_avg(x))          this will give -195
x@_FillValue = -999.       now -999. denotes a missing value
print(dim_avg(x))          this gives 6: missing value is ignored!
```

- Better not to use zero as a missing value
- Use **printVarSummary** to check if a variable has a defined **_FillValue**
- Like any other attribute, **_FillValue** can be accessed using **@**
- There are many functions to deal with missing values. See: http://www.ncl.ucar.edu/Document/Functions/metadata.shtml

Metadata assignment: attributes @

- Attributes are accessed using @:

```
T = new((/4,8,3/), float)
T@_FillValue = -999.
T@units = "K"
T@long_name = "Temperature"
T@model = "EMAC"
```

- Test for an attribute:

```
isatt(T,"units")
```

- Retrieve all attributes from an NCL variable:

```
T_atts = getvaratts(T)
```

- Retrieve all attributes from an NCL variable on a file:

```
f = addfile("filename.nc","r")
T_atts = getfilevaratts(f,"T")
```

- Delete an attribute:

```
delete(T@model)
```

Metadata assignment: named dims !

- Dimesions of a variable can be named using **!**:

```
T = new((/4,8,3/), float)
T!0 = "time"
T!1 = "lat"
T!2 = "lon"
```

- Checking for dimension names:

```
isdimnamed(T,0)          left-most dimension
isdimnamed(T,(/1,2/))    two right-most dimensions
```

- Retrieve all dimensions from an NCL variable:

```
T_dims = getvardims(T)
```

- Retrieve all dimensions from an NCL variable on a file:

```
f = addfile("filename.nc","r")
T_dims = getfilevardims(f,"T")
```

- Delete a named dimension:

```
T!1=""
```

Metadata assignment: coordinates &

- Coordinate arrays are 1-D arrays representing the values for a given (named!) dimension
- Coordinates can be assigned using **&**:

```
T = new((/4,8,3/), float)
T!0 = "time"
T!1 = "lat"
T!2 = "lon"
T&time = (/0., 6., 12., 18./)
T&lat  = fspan(-90.,90.,8)
T&lon  = fspan(-180,180,3)
```

fspan creates a uniformly spaced array

- Attributes can be assigned to coordinate arrays too:

```
T&time@units = "hours"
T&lat@units  = "degrees North"
T&lon@units  = "degrees East"
```

- More functions to deal with attributes, dimensions and coordinates: http://www.ncl.ucar.edu/Document/Functions/metadata.shtml

String reference $

- Reference to an attribute, coordinate or named dimension can be obtained also using a string variable, by enclosing it in **$...$**:

```
str = "units"
T@$str$ = "temperature"

str = "time"
print(T&$str$)
```

- When reading a variable from a file, variable name can be replaced by a string variable:

```
str = "T"
f = addfile("filename.nc","r")
x = f->$str$
```

- It cannot be used by itself:

```
$str$ = x
```
error

Operations on arrays

- Like Fortran 90 and C: for arrays of the same sizes, arithmetics can be performed without looping:

```
; x1 is (time,lev,lat,lon)
; x2 is (time,lev,lat,lon)
x3 = 12. * x1
x3 = x1 * x2
x3 = 5.          set all elements to 5.
```

- Very useful **conform** function, to promote an array and perform computations:

```
; x1 is (time,lev,lat,lon)
; x2 is (time,lat,lon)
x3 = x1 * x2              error (Number of dimensions do not match)
x2_prom = conform(x1, x2, (/0,2,3/))
x3 = x1 * x2_prom
```

- Metadata are not copied during operations:

```
x3 = x1 * x2   metadata are not copied
x3 = x1        copy metadata first
x3 = x1 * x2
```

Operations on arrays

- Array reshaping, **ndtooned** and **onedtond** functions:

```
T = (/ (/4.,5.,3./), (/9.,10.,11./), (/0.,7.,8./) /)
T1D = ndtooned(T)
```
convert to 1D array

```
T = (/1.,2.,3.,4.,5.,6./)
T2D = onedtond(T, (/2,3/))
```
convert to 2×3 array

- Array reordering (named dimensions required, very expensive operation!):

```
; x1 is (time,lev,lat,lon)
x2 = x1(lat|:,lon|:,time|:,lev|:)
```

- Dimension reversing:

```
; x1 is (time,lev,lat,lon)
x1 = x1(::-1,:,:,:)
```
reverse time dimension (and coordinate too!)

- Other functions for array creation, manipulation and query. See:
  http://www.ncl.ucar.edu/Document/Functions/array_create.shtml
  http://www.ncl.ucar.edu/Document/Functions/array_manip.shtml
  http://www.ncl.ucar.edu/Document/Functions/array_query.shtml

Special array functions

- **where**: replace array values given a condition:

```
x = where(condition, value if condition True, value if condition False)
```

```
x = (/ -5., 1., 3., -7., 0., 11., -999./)
x@_FillValue = -999.
y = where(x.lt.0., 0., x)              replace negative values with zero
y = where(ismissing(x),0.,x)           replace missing values with zero
y = where(x.eq.0.,x@_FillValue,x)      replace zero with missing value
```

- **num**: number of elements for which the condition is true:

```
nn = num(x.lt.0.)                      should give 2
```

- **any**: gives True if any of the elements satisfies the condition:

```
ll = any(ismissing(x))                 should be True
ll = any(x.eq.50.)                     should be False
```

- **ind**: return the index (indexes) for which the condition is True (1D arrays):

```
nn = ind(x.lt.0.)                      should give 0 and 3
nn = ind(ismissing(x))                 should give 6
```

Operations and missing values

- If more than one term in an expression contains a missing value and the values are not equal, the missing value of the value of the left-most term in the expression containing a missing value is used in the output:

```
x1 = (/1, 2, -99/)          x1@_FillValue = -99
x2 = (/3, -999, 5/)         x2@_FillValue = -999
x3 = (/-9999, 7, 8/)        x3@_FillValue = -9999

out1 = x1 * x2 * x3         out1@_FillValue = -99
out2 = x2 * x1 * x3         out2@_FillValue = -999
out3 = x3 * x1              out3@_FillValue = -9999
```

- Use the function **ismissing** to check for missing values of a given variable
- Use the function **assignFillValue** to transfer the attribute **_FillValue** from one variable to another

Array subscripting

- Subscripting is used to access specific elements of an array
- In NCL there are 3 types of subscripting:

  index (like Fortran, C, IDL...): uses **:** and **::** and the *index* value
  coordinate: uses **{** and **}** and the *coordinate* value
  named dimensions: uses **|** and reordering

- Subscripting types can be mixed in the same array
- Index subscripting is 0-based (like C, IDL and python; while Fortran is 1-based). For example:

```
x = (/5., 7., 9., 12., 25./)
x(0) = 5.
x(4) = 25.
x(5)    error (Subscript out of range)
```

- Be aware of dimension reduction when subscripting

Index subscripting, : and ::

- Consider an array **x(time,lat,lon)**:

```
y = x(:,:,:)      copy entire array
y = x             equivalent to above (indexes not required)
y = (/x/)         copy entire array (without metadata!)
```

- 1st time, all lat, first 51 lon:

```
y = x(0,:,:50)    dimensions = (nlat, 51)
y = x(0,:,0:50)   equivalent to above
```

- 1st time, all lat, every 2nd lon:

```
y = x(0,:,::2)    dimensions = (nlat, nlon/2)
```

- Like above, but preventing dimension reduction:

```
y = x(0:0,:,::2)  dimensions = (1, nlat, nlon/2)
```

- Vectors of indexes can also be used. 1st, 2nd, 4th, 7th time:

```
y = x((/0,1,3,6/),:,:)   dimensions = (4, nlat, nlon)
```

Coordinate subscripting, { and }

- Coordinate subscripting works only with monothonically increasing or monothonically decreasing coordinate arrays
- If only one value is specified, the nearest coordinate is selected:

```
y = x(:,{20},:)      lat nearest to 20°N
y = x(:,:,{-80})     lon nearest to 80°W
```

- If a range is given, only values inside such range are considered:

```
y = x(:,{5:15},:)        every lat from 5°N to 15°N
y = x(:,:,{10:50:3})     every 3rd lon from 10°E to 50°E
```

- Be very careful with the longitude coordinate, since it could be given as [0,360] or as [-180,180]. Common mistake:

```
y = x(:,:,{-20:30})      if lon is [0,360] this is wrong!
```

- Subscripting types can be mixed:

```
y = x(0:5,{45},{20:30})
```

Named dimensions,

- Only use for dimensions reordering
- Dimension names must be used for every subscript
- Reorder **`x(time,lat,lon)`**:

```
y = x(lat|:,lon|:,time|:)
```

- Can be mixed with other subscripting:

```
y = x(lat|0,lon|::5,time|:10)
```
first lat, every 5th lon, first 11 time elements

```
y = x(time|:,{lon|2:9},lat|:)
```
all time, lon in the range [2,9], all lat

- Remember that reordering is a computationally expensive operation!
- The structure of climate data (like model output) is typically (time,lev,lat,lon): do not change it inside the script, unless absolutely necessary

Lists

- Lists can be defined using **[/…/]**
- Lists can contain a heterogeneous set of variables, with different types and sizes:

```
A = 12.                          float
B = "Hello"                      string
C = (/-31, 2, 14, 6/)            integer array
mylist = [/A, B, C/]             list
```

- List can also be initialized with the function **NewList** and new elements can be added using **ListPush**

```
mylist = NewList("fifo")   fifo=first-in, first-out / lifo=last-in, first-out
ListPush(mylist, A)
ListPush(mylist, B)
```

- Additional functions are available to handle list **ListPush**, **ListPop**, **ListCount**, **ListIndex**, **ListGetType**, **ListSetType**.

NCL syntax summary

;	comment
@	reference/create attributes
!	reference/create named dimensions
&	reference/create coordinate variables
:=	reassignment operator
:	array index subscripting
{...}	array coordinate subscripting
|	array named dimensions
$...$	string reference (to reference metadata or variable from file)
(/.../)	array construct character
[/.../]	list construct character
****	continuation character (statement to span multiple lines)
->	import/export variable via the **`addfile`** function
::	syntax for external shared objects (Fortran/C)
~	function code http://www.ncl.ucar.edu/Document/Graphics/function_code.shtml

NCL statements

- **if**, similar to Fortran, but no **else if** statement:

```
if (condition) then
  do something
else
   do somehing else
end if
```

- **do**, use **continue** to proceed to the next iteration and **break** to exit loop:

```
do ii=0,10
   do something
end do
```

- **do while**:

```
ii=0
ll = False
do while (.not.ll)
   do something / set ll=True
   ii = ii+1
end do
```

Logical operators

`.eq.`	equal
`.ne.`	not-equal
`.lt.`	less-than
`.le.`	less-than-or-equal
`.gt.`	greater-than
`.ge.`	greater-than-or-equal
`.and.`	True if both operands are True
`.or.`	True if either operand is True
`.xor.`	True if one of the operands is True and the other is False
`.not.`	True if the operand is False and vice versa

Beware of missing values in logical expressions!

Left	Oper	Right	Result
False	`.and.`	*Any*	False
True	`.and.`	False	False
True	`.and.`	True	True
True	`.and.`	Missing	Missing
Missing	`.and.`	*Any*	Missing
True	`.or.`	*Any*	True
False	`.or.`	True	True
False	`.or.`	False	False
False	`.or.`	Missing	Missing
Missing	`.or.`	Missing	Missing

File input and output

NetCDF

Read:

```
f = addfile("file.nc","r")
x  = f->varname
```

`r` for read

`varname` is a string

Write:

```
f = addfile("file.nc","c")
f = addfile("file.nc","w")
f->varname = x
```

`c` for create (new file)

`w` for write (existing file)

`varname` is a string

OTHER FORMATS

GRIB1
GRIB2
HDF4
HDF-EOS2
CCM
OPeNDAP
Binary

See the link below...

ASCII

Read:

```
x = asciiread(file, dimension, type)
x = asciiread("file.dat",(/10,5/),float)
```

Write:

```
asciiwrite(file, variable)
asciiwrite("file.dat",x)
```

STDOUT

Print variable‘s values:
`print(x)`

Print variable‘s information:
`printVarSummary(x)`

For 2D arrays:
`write_matrix(x)`

More functions for input/output: http://www.ncl.ucar.edu/Document/Functions/io.shtml

Functions and procedures

- Three kinds of functions/procedures:

  Built-in
  User-generated
  C and Fortran

- When using a function, the return value must be referenced:

```
x = (/12.2, 21.5, 0.5, -4.1, 8.2, 5.4/)
max(x)                  error (return value must be referenced)
y = max(x)              ok (assign to y)
print(max(x))           ok (print on screen)
```

- Arguments of functions/procedures are passed-by-reference: changes to a variable's value/metadata within the function/procedure are propagated back to the main code!
- Most of built-in functions ignore missing values
- Most of built-in functions do not retain metadata (unless **_Wrap** version is used)
- Useful **system** and **systemfunc** to execute shell commands within the script:

```
system("ls *.nc")
```

Built-in functions/procedures

http://www.ncl.ucar.edu/Document/Functions/

- General routines (variables, arrays, strings, type conversion, system...)
- Math and statistics (basic, distribution functions, spherical harmonics, random number generators...)
- Earth science (climatology, meteorology, oceanography, latitude/longitude, regridding, time/date...)
- Input and output (NetCDF, ascii, binary...)
- Graphics (plot types, colors...)

Remember!

function → requires a return value (e.g., **`dim_avg`** , **`ispan`**)

procedure → no return value (e.g., **`printVarSummary`**, **`delete`**)

Arguments are always passed-by-reference

User-defined functions/procedures

- Two possibilities:

  ➔Paste the function code at the beginning of the script
  ➔Save the function code in an external **.ncl** file and load it

```
load "./myfunc.ncl"
```

- How to create your own function:

```
undef("myfunc")
function myfunc(arg1,arg2,...,argn)
begin
  ...
  return(value)
end
```

- For procedures a return value is not required
- Optionally specify the expected argument type and/or size:

```
function myfunc(arg1:numeric,arg2[*]:integer)
procedure myproc(arg1[*][*]:string,arg2:logical)
```

Importing Fortran/C functions

- Write the Fortran code in **mycode.f**, including the special wrapper text:

```
C NCLFORTSTART
      subroutine mysub (arg1,arg2,arg3)
      real arg1,arg2,arg3
C NCLEND
          ...
      return
      end
```

- Compile using WRAPIT:

  WRAPIT mycode.f will create an object **mycode.so**

- Add the shared object at the beginning of the NCL script:

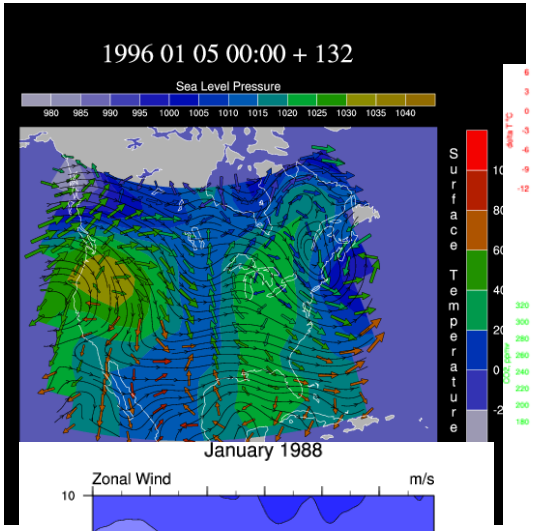
```
external EX01 "./mycode.so"
```

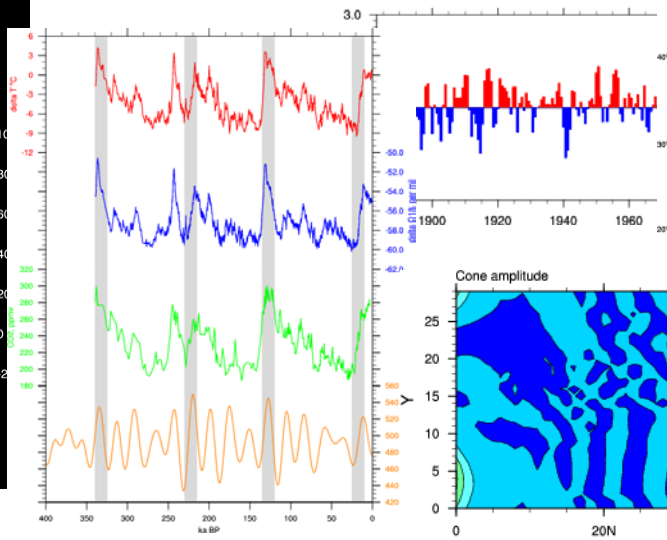
- Call the function inside the NCL script:

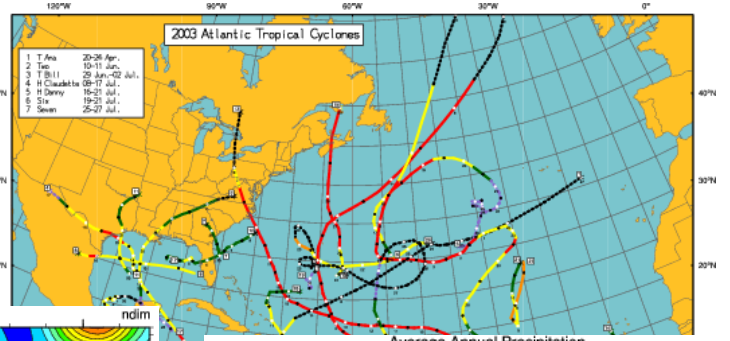
```
EX01::mysub(x1,x2,x3)
```

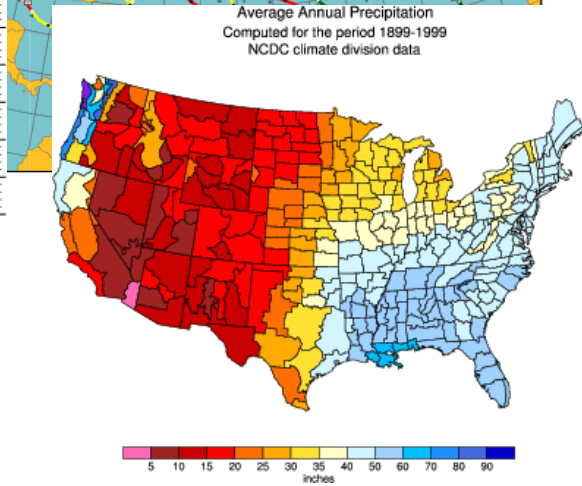
A similar method can also be applied to Fortran 90 and C codes.

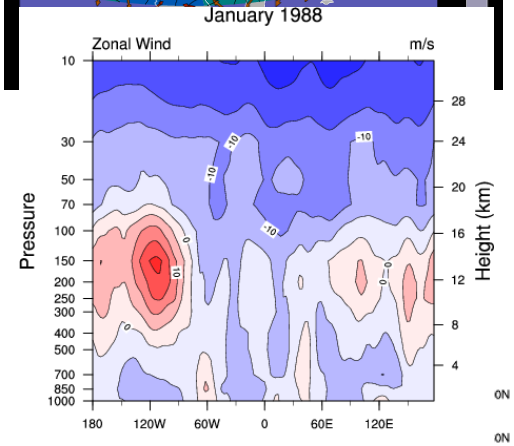
http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclExtend.shtml



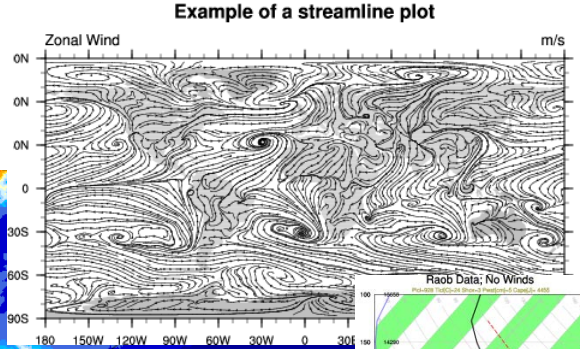


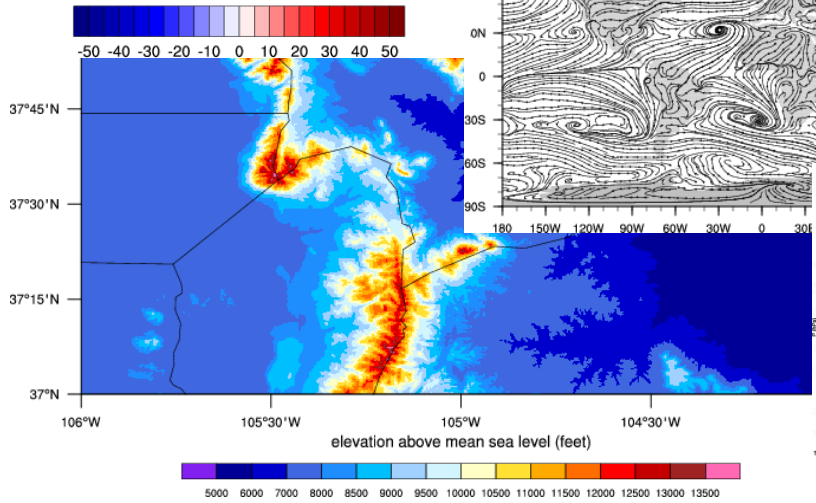


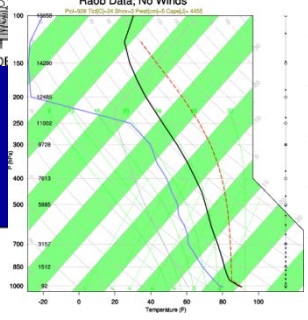


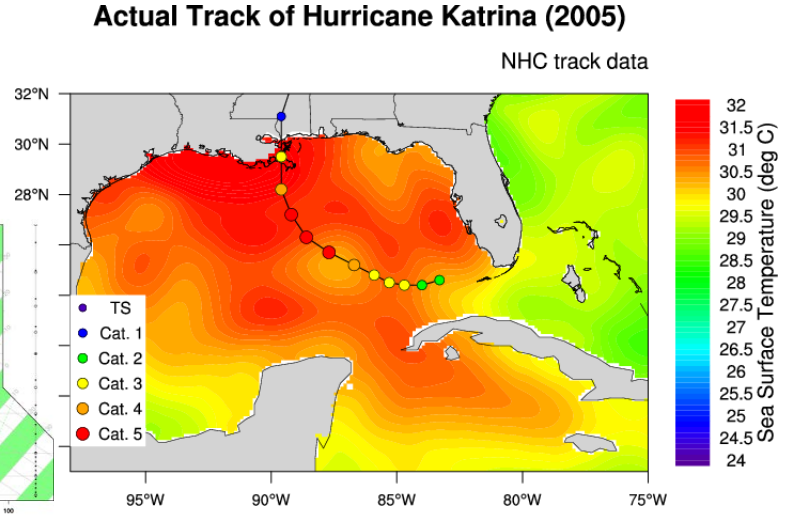


NCL graphics









Sample graphic script

1. Read (and process) data to be plotted

```
f = addfile("filename.nc","r")
x = f->varname
x_avg = dim_avg_n(x,0)
```

2. Open a workstation (ps, pdf or screen) and define an associated color table

```
wks = gsn_open_wks("ps","plotfile")
gsn_define_colormap(wks,"rainbow")
```

3. Set the plot resources (plot options, like tickmarks, levels, title, labels etc.)

```
res                          = True
res@cnLevelSelectionMode = "Explicit"
res@cnLevels                 = fspan(0.,100.,11)
...
```

4. Choose a plot type and draw the plot with the corresponding plot function

```
plot = gsn_csm_contour_map_ce(wks,x_avg,res)
```

Plot types

gsn generic interfaces
(functions or procedures to create basic plots)

```
gsn_xy
gsn_y
gsn_contour
gsn_contour_map
gsn_vector
gsn_vector_scalar
gsn_vector_map
gsn_vector_scalar_map
gsn_streamline
gsn_streamline_map
gsn_map
```

gsn_csm interfaces
(functions or procedures to create high-level plots)

```
gsn_csm_contour
gsn_csm_streamline
gsn_csm_vector
gsn_csm_pres_hgt
gsn_csm_lat_time
gsn_csm_xy
```

- Much more powerful
- Automatically recognize `_FillValue`
- Use variable attributes for plot titles, labels...
- Use variable coordinates for the axes

No need to write a plotting script from scratch!

Start from an existing script: choose a plot example from the NCL website, get the script and modify it.

http://www.ncl.ucar.edu/gallery.shtml

http://www.ncl.ucar.edu/Applications/list_ptypes.shtml

Workstation

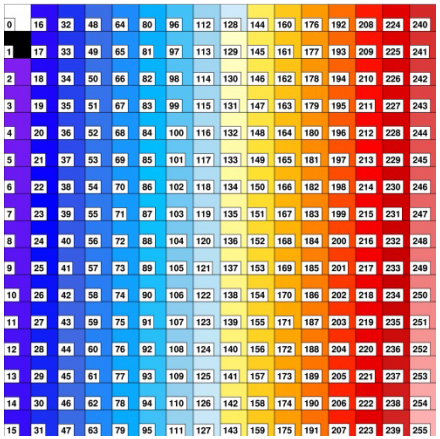
- Before drawing any plot you need to open a workstation: this can be either a file (like .eps) or the screen (x11)
- There are 6 types of workstation: ps, eps, epsi, png, pdf, ncgm, x11
- Specific resources can be associated to the workstation (but default is usually ok):

```
type = "ps"
type@wkOrientation = "landscape"
type@wkPaperSize = "A4"
wks = (type,"plotfile")
```

- An important element to be associated to a workstation is the colormap (see http://www.ncl.ucar.edu/Document/Graphics/color_table_gallery.shtml)

```
gsn_define_colormap(wks,"rainbow")
```

- If no color map is loaded, the defaul one will be used (256 colors)



Resources

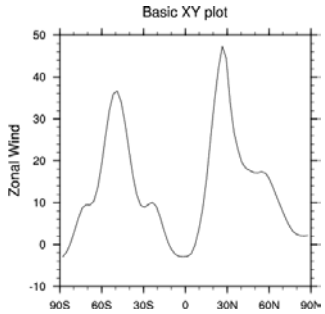
- Resources are the heart of a graphic NCL script
- They allow to customize the default NCL plots
- They can be strings, float, integers, logical... depending on the type
- More than 1400 available!
- Grouped by type: **cn** (contour), **gs** (graphic styles) **lb** (labelbar), **lg** (legend), **ti** (title), **tm** (tickmarks), **xy** (xy plots) etc.
- Written as the type (2 or 3 letters) and a full name describing it: **xyLineColor**, **cnFillColor**, **tiMainString**, **cnLevels**
- To set a resource: define a logical variable (whatever name, usually **res**) and attach the resource as an attribute (with @):

```
res                 = True          define a logical variable
res@tiMainString    = "My plot"     set the plot title
res@cnFillOn        = True          fill contours with color
res@xyLineColor     = "Yellow"      use a yellow line
res@tiMainAngleF    = 45            tilt the plot title of 45°
```

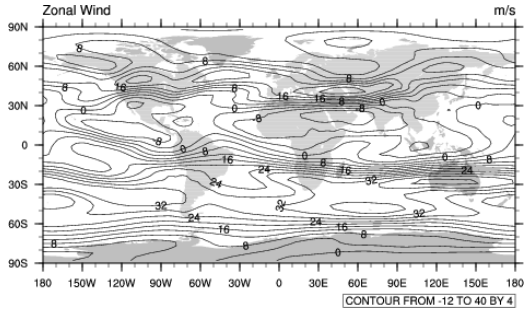
See: http://www.ncl.ucar.edu/Document/Graphics/Resources/list_alpha_res.shtml

Draw the plot

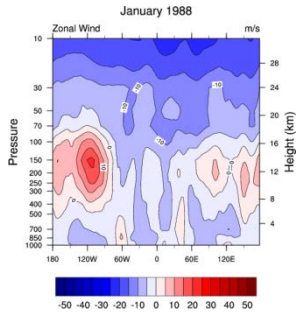
plot = plot_function(*workstation*, *data*, *resources*)




```
plot = gsn_csm_xy(wks,data_x,data_y,res)
```




```
plot = gsn_csm_contour_map_ce(wks,data,res)
```




```
plot = gsn_csm_pres_hgt(wks,data,res)
```


http://www.ncl.ucar.edu/gallery.shtml
http://www.ncl.ucar.edu/Applications/list_ptypes.shtml

Example 1: xy plot

We load the 4-D variable CO from the sample file, compute the time average and plot its value as a function of the vertical level (mlev) at a specified location (lat, lon):

```
begin
  f = addfile("NetCDF_sample.nc","r")
  x = f->CO
  printVarSummary(x)
```

The dimensions of **x** are (time,mlev,lat,lon), this is the CO mixing ratio and the units are mol/mol. We can proceed and compute the time average. We use the **dim_avg_n** function, which computes the average over a specified dimension. Since we need the time average, this would be dimension 0 (dimensions are ordered left to right: in this case time is 0, mlev is 1, lat is 2, lon is 3):

```
x_timavg = dim_avg_n(x,0)
printVarSummary(x_timavg)
```

Now we have a 3-D variable (mlev, lat, lon), the time dimension is gone since we averaged over it. But all metadata information disappeared! Use the **_Wrap** version of the function to retain metadata:

```
x_timavg = dim_avg_n_Wrap(x,0)
printVarSummary(x_timavg)
```

Example 1: xy plot

Now we have the time-averaged variable with all metadata. We can get rid of **x** (this is optional, but is a good practice when dealing with large scripts and lots of variables, to save memory):

```
delete(x)
```

Next, we need to extract a specific location, for example 30°N and 55°W. We use coordinate subscripting for selecting this position:

```
x_sel = x_timavg(:,{30.},{-55.})
```

This will give an error message! Check again the longitude coordinate:

```
printVarSummary(x_timavg)
lon: [   0..357.1875]
```

The range of longitude is [0,360], we have to convert 55°W to a [0,360] range:

```
x_sel = x_timavg(:,{30},{305.})
printVarSummary(x_sel)
delete(x_timavg)
```

Now we have a 1-D variable containing CO mixing ratios as a function of the mlev coordinate. Convert it from mol/mol to ppb:

```
x_sel = x_sel * 1.e9
```

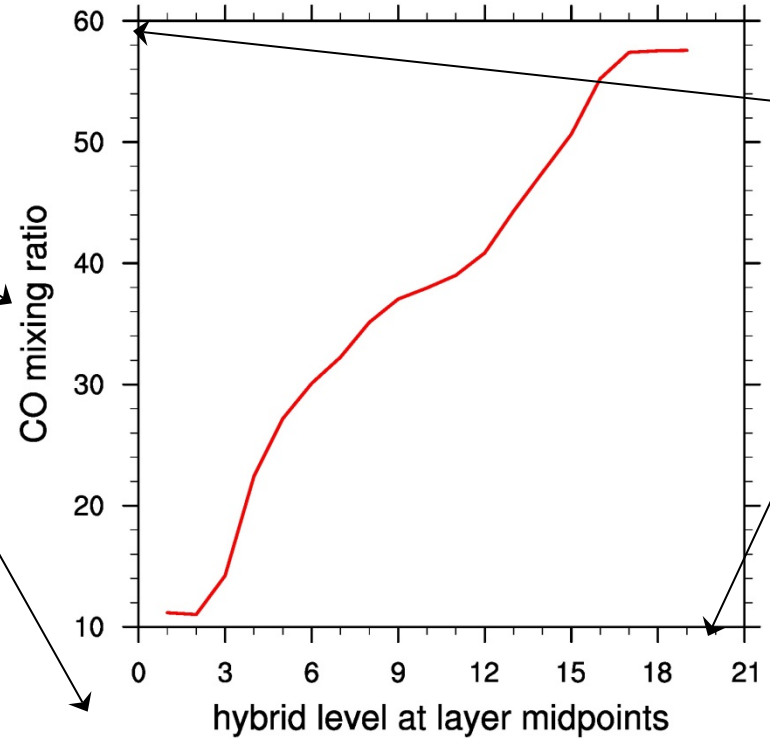
It‘s a good idea (not mandatory) to change the "units" attribute to keep track of this conversion:

```
x_sel@units = "ppbv"
```

Example 1: xy plot

We can now draw the plot: open a workstation, set some resources and choose the appropriate plot function:

```
wks = gsn_open_wks("eps","example1")
res = True
res@xyLineColor      = "red"
res@xyLineThicknessF = 3
plot = gsn_csm_xy(wks,x_sel&mlev,x_sel,res)
```



Axes titles from variable attributes

min/max values for the axes automatically set

We can change these settings acting on the corresponding resources

Example 1: xy plot

You can change the X- and Y-axis titles, for example including the units. These kind of resources are of the type "title" (**ti**) (http://www.ncl.ucar.edu/Document/Graphics/Resources/ti.shtml):

```
res@tiXAxisString = "Level"
res@tiYAxisString = "CO mixing ratio [ppb]"
```

We can also set the Y-axis title using the variable attributes. Use **+** to concatenate the strings:

```
res@tiYAxisString = x_sel@longname + " [" + x_sel@units + "]„
```

To add a title to the plot:

```
res@tiMainString = "Example 1"
```

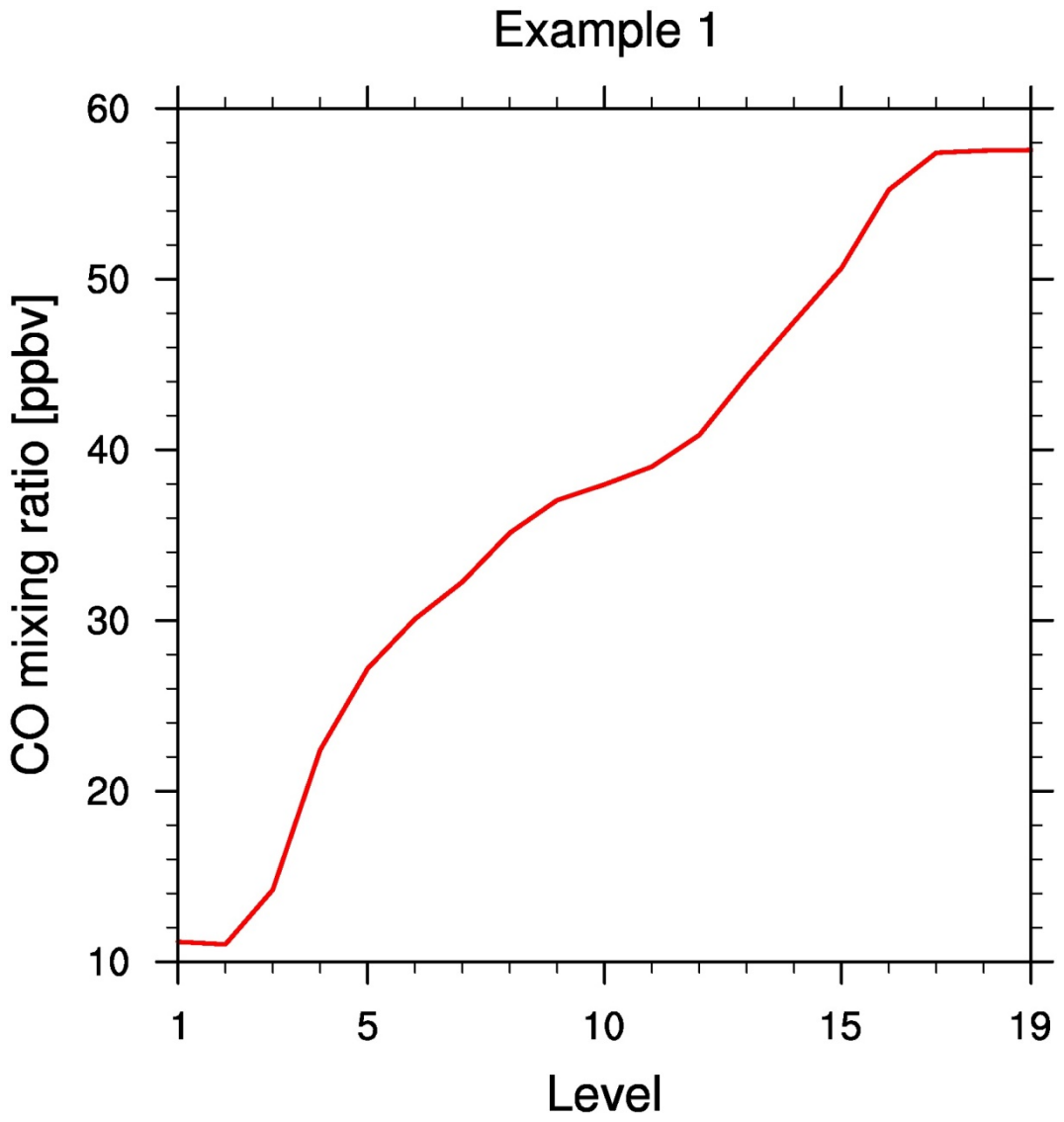
You can also change the min/max of the axes, this is a "transformation" resource (**tr**) (http://www.ncl.ucar.edu/Document/Graphics/Resources/tr.shtml):

```
res@trXMinF = 1.
res@trXMaxF = 19.
```

You can explicitly set the tickmark values, using the "tickmark" resources (**tm**):

```
res@tmXBMode = "Explicit"                  use user-defined tickmarks
res@tmXBValues = (/1,5,10,15,19/)          position of major tickmarks
res@tmXBMinorValues = ispan(1,19,1)        position of minor tickmarks
res@tmXBLabels = (/"1","5","10","15","19"/)  labels for the tickmarks
```

Example 1: xy plot



Example 2: contour plot

We load the 4-D variable O3 from the sample file, compute the time average and make a contour plot of the surface level (levels are ordered top-to-bottom in this file):

```
begin
  f = addfile("NetCDF_sample.nc","r")
  x = f->O3
  x_timavg = dim_avg_n_Wrap(x,0)
```

Since levels are ordered top-to-bottom, the surface level corresponds to the last element of the mlev coordinate. This can be found using the dimsizes function, remembering that arrays are 0-based:

```
x_sel = x_timavg(dimsizes(x_timavg&mlev)-1,:,:)
```

These commands can also be written in a single statement:

```
x_sel = dim_avg_n_Wrap(x(:,dimsizes(x&mlev)-1,:,:),0)
```

Unit conversion:

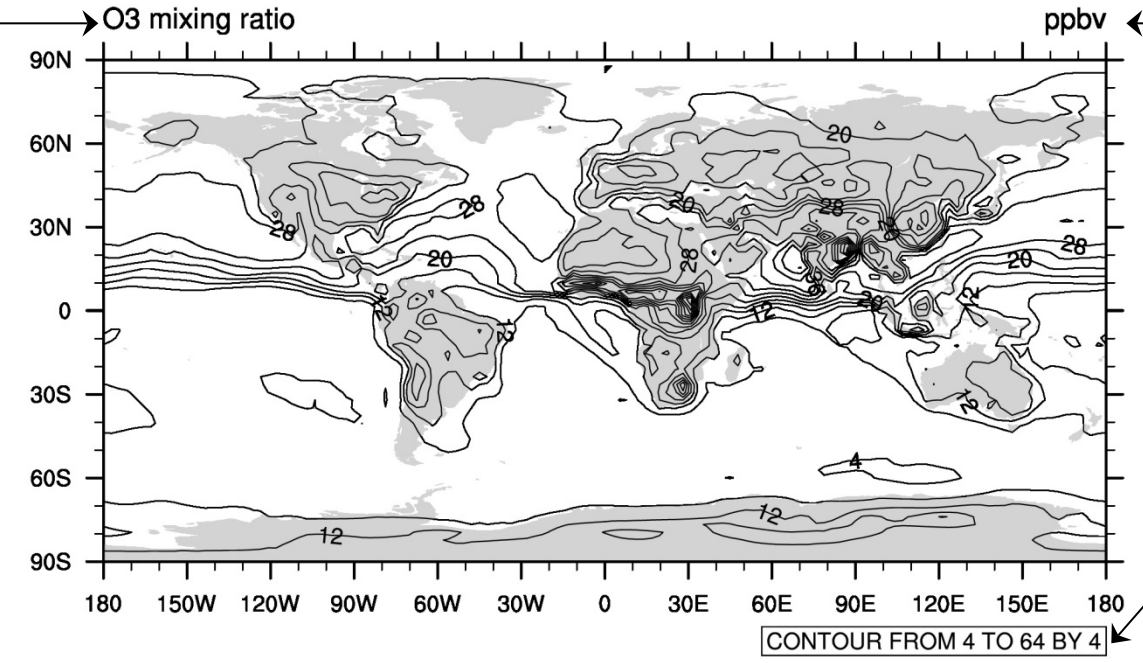
```
x_sel = x_sel * 1.e9
x_sel@units = "ppbv"
```

Example 2: contour plot

We have now a 2-D (lat,lon) variable, we can draw a contour plot over a map. There are many possibilities, depending on the map projection: cylindrical equidistant (**_ce**), polar (**_polar**), Lambert, satellite, etc. (http://www.ncl.ucar.edu/Applications/proj.shtml). Let's try with the cylindrical equidistant:

```
wks = gsn_open_wks("eps","example2")
res = True
plot = gsn_csm_contour_map_ce(wks,x_sel,res)
```

Read from @long_name



Read from @units

Automatically set

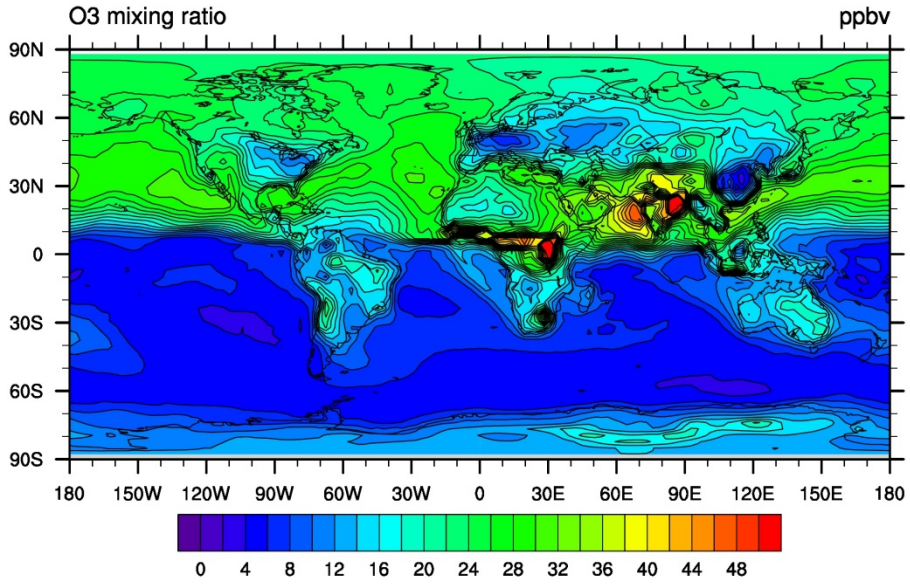
Example 2: contour plot

We can change contour levels acting on the "contour" resources (**cn**).

```
res@cnLevelSelectionMode = "ManualLevels"
res@cnMinLevelValF = 0.
res@cnMaxLevelValF = 50.
res@cnLevelSpacingF = 2.
```

We can also use colors, we need to load a colormap and turn on contour fill:

```
gsn_define_colormap(wks,"rainbow")
res@cnFillOn = True          turn on contour fill
```



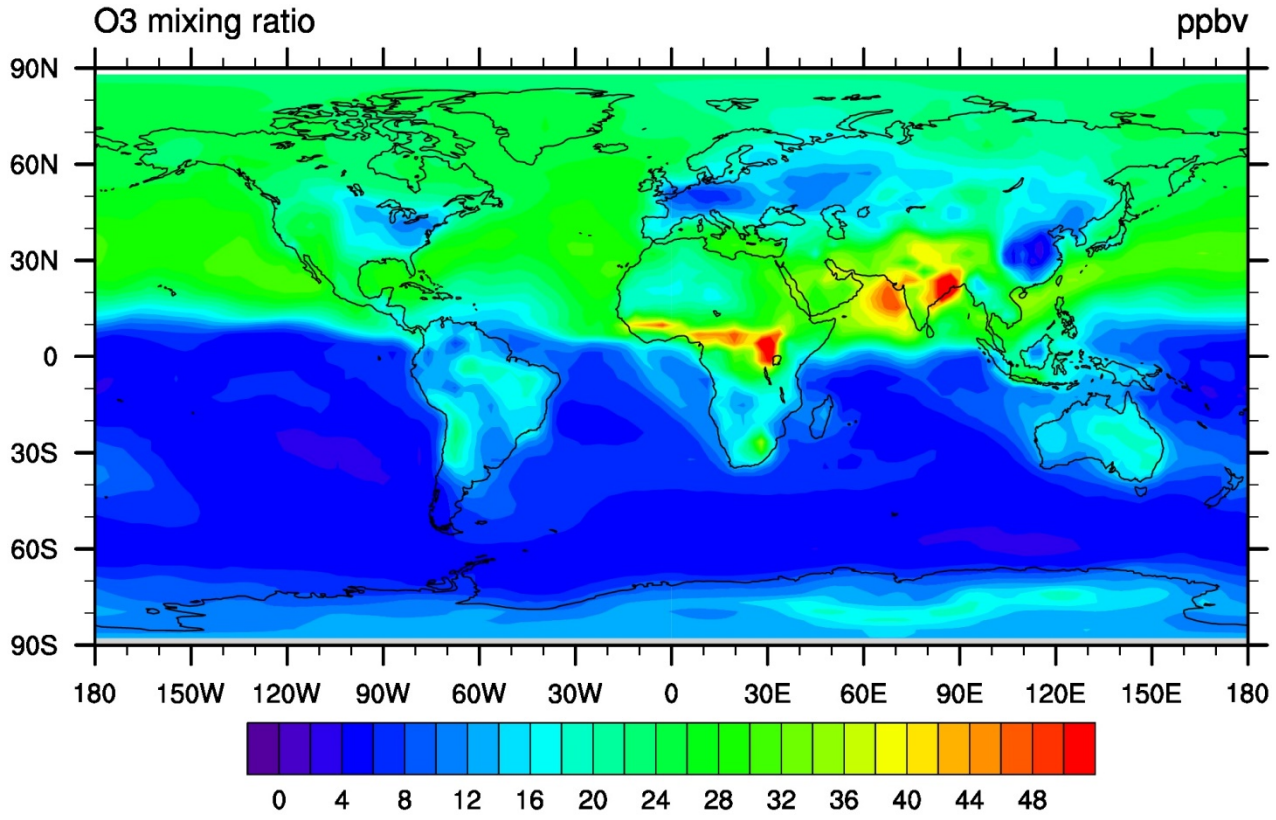
Still too crowded!

Turn off contour lines

Example 2: contour plot

Turn of contour lines:

```
res@cnLinesOn = False
```



Paneling

To panel multiple plots (say 3) in a single image, first create a graphic array of dimension 3:

```
plot = new(3, graphic)
```

When setting resources, remember to include the following:

```
res@gsnDraw = False
res@gsnFrame = False
```

This is because the high-level graphic interfaces (like **plot_gsn_xy**) automtically create and draw graphical objects and advance the frame (i.e. "turn the page"). When paneling, this behaviour must be turned off: different plots are saved in an array of graphical objects and drawn all together with the paneling function.

Returning to previous example, suppose we want to plot O3 mixing ratio at the three lowermost levels and panel the three plots:

```
nlev = dimsizes(x_sel&mlev)
plot(0) = gsn_csm_contour_map_ce(wks,x_sel(nlev-1,:,:),res)
plot(1) = gsn_csm_contour_map_ce(wks,x_sel(nlev-2,:,:),res)
plot(2) = gsn_csm_contour_map_ce(wks,x_sel(nlev-3,:,:),res)
```

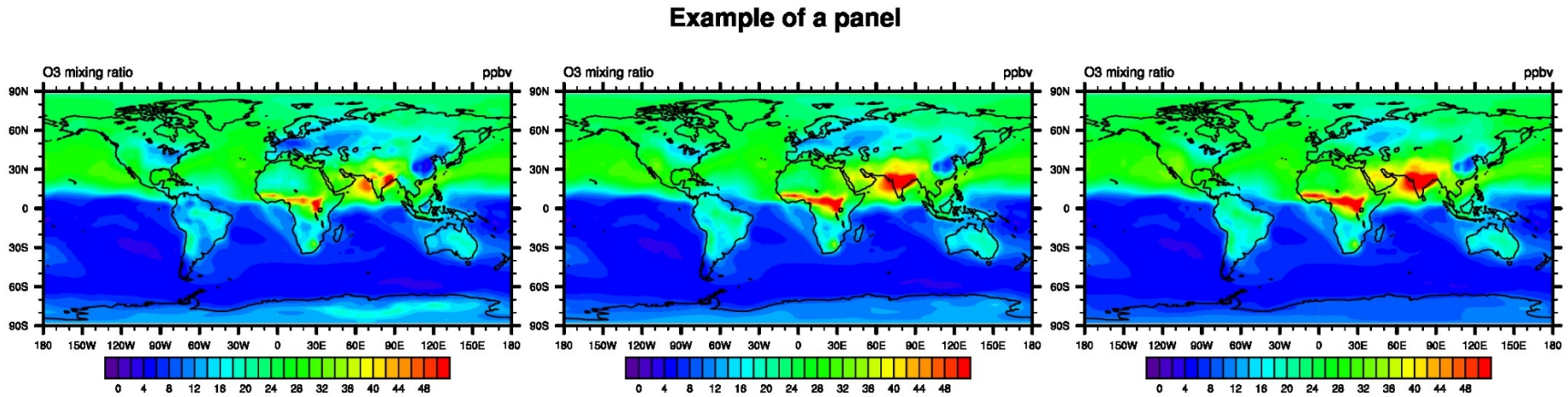
The 3 plots are stored in the graphic array **plot**, but they have not been drawn yet!

Paneling

Now we can call the panling procedure. This is equivalent to any other graphical interface and can have ist own specific resources:

```
resPan = True
resPan@txString = "Example of a panel"        set the title
resPan@txFontHeightF = 0.012                  set the title font size
resPan@txFont = 22                            set the title font type
gsn_panel(wks,plot,(/1,3/),resPan)
```

By setting **(/1,3/)** the 3 plots are drawn in 1 row and 3 column.



There are many font types to choose from (http://www.ncl.ucar.edu/Document/Graphics/font_tables.shtml)

Adding text, lines and markers

Two methods to add elements (text, lines, polygons etc.) to a plot:

➔**gsn_add_*** functions: use plot coordinates (must be referenced to a graphic variable)
➔**gsn_*_ndc** procedures: use normalize coordinates [0,1] on the workstation

To add a text string:

```
newtext = gsn_add_text(wks,plot,"Some text",xpos,ypos,resT)
gsn_text_ndc(wks,"Some text",xpos,ypos,resT)
```

To draw a polygon:

```
newpoly = gsn_add_polygon(wks,plot,xcoords,ycoords,resP)
gsn_polygon_ndc(wks,xcoords,ycoords,resP)
```

To draw a line:

```
newline = gsn_add_polyline(wks,plot,xcoords,ycoords,resL)
gsn_polyline_ndc(wks,xcoords,ycoords,resL)
```

To add a marker (symbols like http://www.ncl.ucar.edu/Document/Graphics/Images/markers.png):

```
newmark = gsn_add_polymarker(wks,plot,xpos,ypos,resM)
gsn_polymarker_ndc(wks,xpos,ypos,resM)
```

Remember to set **res@gsnDraw** and **res@gsnFrame** to False when adding these objects!
Resources must be associated to a different graphic variable than the one used for the plot.

Tips & tricks

- Start from an existing script, if possible
- Use indentation: it is not mandatory, but makes the script more readable
- Use comments (`;`) inside the script to include some descriptions
- Use **`printVarSummary`** to examine variables and **`ismissing`** to search for missing values
- Before writing a function/procedures check for the built-in ones
- Avoid unnecessary **`do`** loops, use array arithmetics if possible
- Avoid dimension reordering in arrays: this is an expensive operation
- Save memory: use **`delete`** to get rid of large arrays
- Configure the text exitor (e.g. emacs) with highlighting, see this page: http://www.ncl.ucar.edu/Applications/editor.shtml
- Use the NCL webpage: examples, scripts, manuals, FAQ, mailing-lists...

Useful links

NCL home	http://www.ncl.ucar.edu/index.shtml
Source code	http://www.ncl.ucar.edu/Download/
Reference manual	http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclExtend.shtml
Language manual	http://www.ncl.ucar.edu/Document/Manuals/language_man.pdf
Graphics manual	http://www.ncl.ucar.edu/Document/Manuals/graphics_man.pdf
Reference cards	http://www.ncl.ucar.edu/Document/Reference_Cards/
DKRZ Supplement	https://www.dkrz.de/Nutzerportal-en/doku/vis/sw/ncl/DKRZ_NCL_Supplements_Doc_layout.pdf/view
FAQ	http://www.ncl.ucar.edu/FAQ/
NCL/NCAR mailing list	http://www.ncl.ucar.edu/Support/email_lists.shtml
Lecture material	http://www.pa.op.dlr.de/~MattiaRighi/NCL/LECTURE/lecture_index.html
Contact	Mattia.Righi@dlr.de