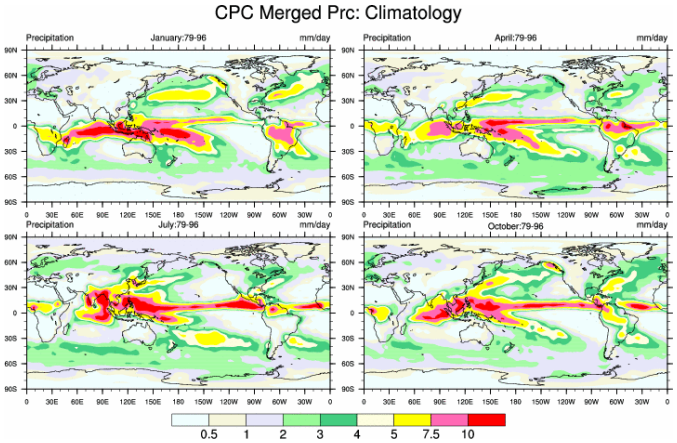
NCL Data Processing



Dennis Shea

National Center for Atmospheric Research

NCAR is sponsored by the National Science Foundation

Data Processing Outline

- Algebraic/logical expression operators
- Manual and automatic array creation
- **if** statements , **do** loops
- Built-in and Contributed functions
- User developed NCL functions/procedures
- User developed external procedures
- Sample processing
- Command Line Arguments [CLAs]
- Fortran external subroutines
- NCL as a scripting tool [time permitting]
- Global Variables [time permitting]

Algebraic Operators

Algebraic expression operators

-	Negation	^	Exponentiation
*	Multiply	/	Divide
%	Modulus [integers only]	#	Matrix Multiply
+	Plus	-	Minus
>	Greater than selection	<	Less than selection

- Use (…) to circumvent precedence rules
- All support scalar and array operations [like f90]
- + is overloaded operator
  - algebraic operator:
    - 5.3 + 7.95 ➔ 13.25
  - concatenate string
    - "alpha_" + (5.3 + 7) ➔ "alpha_12.3"

Logical Expressions

Logical expressions formed by relational operators

.le.	(less-than-or-equal)
.lt.	(less-than)
.ge.	(greater-than-or-equal)
.gt.	(greater-than)
.ne.	(not-equal)
.eq.	(equal)
.and.	(and)
.xor.	(exclusive-or)
.or.	(or)
.not.	(not)

Manual Array Creation

- **array constructor characters (/…/)**
  - a_integer = (/1,2,3/)
  - a_float = (/1.0, 2.0, 3.0/) , a_double = (/1., 2, 3.2**d** /)
  - a_string = (/"abc",”12345",”hello, world"/)
  - a_logical = (/True, False,True/)
  - a_2darray = (/ (/1,2,3/), (/4,5,6/), (/7,8,9/) /)

- **new** function [Fortran dimension, allocate ; C malloc]
  - x = **new** (array_size/shape, type, **_FillValue**)
    - **_FillValue** is optional [assigned default if not user specified]
    - **“No_FillValue”** means no missing value assigned
  - a = **new**(3, float)
  - b = **new**(10, double, **1d20**)
  - c = **new**((/5, 6, 7/), integer)
  - d = **new**(**dimsizes**(U), string)
  - e = **new**(**dimsizes**(**ndtooned**(U)), logical)

- **new** and (/…/) can appear anywhere in script
  - new is not used that often

Automatic Array Creation

- **data importation via supported format**
  - u = f->U
  - same for subset of data: u = f->U(:, 3:9:2, :, 10:20)
    - meta data (coordinate array will reflect subset)

- **variable to variable assignment**
  - **y = x** **y => same size, type as x plus meta data**
  - **no need** to pre-allocate space for **y**

- **functions**
  - return array: **no need** to pre-allocate space
  - grido = **f2fsh** (gridi, (/ 64,128/))
  - gridi(10,30,73,144) ➔ grido(10,30,64,128)
    - grido = **f2fsh_Wrap** (gridi, (/ 64,128/)) ; contributed.ncl

Array Dimension Rank Reduction

- **subtle point: singleton dimensions eliminated**
- **let T(12,64,128)**
  - Tjan = T(0, :, :) ➔ Tjan(64,128)
  - Tjan automatically becomes 2D: Tjan(64,128)
  - array rank reduced; 'degenerate' dimension
  - all applicable meta data copied

- **can override dimension rank reduction**
  - Tjan = T(0:0,:,:) ➔ Tjan(1,64,128)
  - TJAN = **new**(**(/**1,64,128**/)**, **typeof**(T), T@_FillValue)
    - TJAN(0,:,:) = T(0,:,:)

- **Dimension Reduction is a "feature" [really ☺]**

Array Syntax/Operators

- **similar to f90/f95, Matlab, IDL**
- **arrays must be same size and shape: conform**
- **let A and B be (10,30,64,128)**
  - C = A+B
  - D = A-B
  - E = A*B
  - C, D, E automatically created if they did not previously exist
- **let T and P be (10,30,64,128)**
  - theta = T*(1000/P)^0.286 ➔ theta(10,30,64,128)
- **let SST be (100,72,144) and SICE = -1.8 (scalar)**
  - SST = SST **>** SICE [f90: where (sst.lt.sice) sst = sice]
  - the operation performed by **<** and **>** is (sometimes) called *clipping*
- **use built-in functions whenever possible**
  - Let T be (30,**30**,64,128) and P be (**30**) then
  - theta = T*(1000/**conform**(T,P,**1**))^0.286
- **all array operations automatically ignore _FillValue**

if blocks (1)

- **if-then-end if** (note: end if has space)

```
if ( all(a.gt.0.) ) then
        …statements
end if
```

- **if-then-else-end if**

```
if ( any(ismissing(a)) ) then
        …statements
else
        …statements
end if
```

- lazy expression evaluation [left-to-right]

```
if ( any(b.lt.0.)  .and.  all(a.gt.0.) ) then
  …statements
end if
```

loops

- **do loop** (traditional structure; **end do** has space)
  - **do** i=*scalar_start_exp, scalar_end_exp [, scalar_skip_exp]*

```
do n = nStrt, nLast [,stride]
    … statements
end do                    ; 'end do' has a space
```

  - if start > end
    - identifier 'n' is decremented by a positive stride
    - stride must always be present when start>end

- **do while loop**

```
do while (x .gt. 100)
    … statements
end do
```

- **break**: loop to abort [f90: exit]
- **continue**: proceed to next iteration [f90: cycle]

- **minimize loop usage in any interpreted language**
  - use array syntax, built-in functions, procedures
  - use Fortran/C codes when efficient looping is required

Built-in Functions and Procedures (1 of 2)

- **use whenever possible**
- **learn and use utility functions**
  - all, any, conform, ind, ind_resolve, dimsizes
  - fspan, ispan, ndtooned, onedtond,
  - mask, ismissing, where
  - system, systemfunc [use local system]
- **functions may require dimension reordering**
  - *must* use named dimensions to reorder

```
; compute zonal and  time averages of variable
; T(time,lev,lat,lon) ➔ T(0,1,2,3)
;          (zonal average requires rectilinear grid)

; no meta data transfered
    Tzon = dim_avg_n( T, 3)                ; Tzon(ntim,klev,nlat)
    Tavg = dim_avg_n( T, 0)                ; Tavg(klev,nlat,mlon)
```

dimsizes(x)

- returns the dimension sizes of a variable
- will return 1D array of integers if the array queried is multi-dimensional.

```
fin    = addfile("in.nc","r")
t      = fin->T
dimt = dimsizes(t)
print(dimt)

rank = dimsizes(dimt)
print ("rank="+rank)
```

```
Variable: dimt
Type: integer
Total Size: 16 bytes
4 values
Number of dimensions: 1
Dimensions and sizes:(4)
(0)  12
(1)  25
(2) 116
(3) 100
(0) rank=4
```

ispan(start:integer, finish:integer, stride:integer)

- returns a 1D array of integers
  - beginning with **start** and ending with **finish**.

```
time = ispan(1990,2001,2)
print(time)
```

```
Variable: time
Type: integer
Number of Dimensions: 1
Dimensions and sizes:(6)
(0) 1990
(1) 1992
(2) 1994
(3) 1996
(4) 1998
(5) 2000
```

fspan(start, finish, npts)

- returns a 1D array of evenly spaced float/double values
- **npts** is the integer number of points including **start** and **finish** values

```
b = fspan(-89.125,  9.3, 100)
print(b)
```

```
Variable b:
Type: float
Number of Dimensions: 1
Dimensions and sizes:(100)
(0) -89.125
(1) -88.13081
(2) -87.13662
(…) ….
(97) 7.311615
(98) 8.305809
(99) 9.3
```

ismissing

- **must** be used to check for **_FillValue** attribute
  - if (x.eq.x@_FillValue) will **not** work

```
x = (/ 1,2, -99, 4, -99, -99, 7/)     ; x@_FillValue = -99
xmsg = ismissing(x)    ; print(xmsg)
        (/False, False, True, False, True, True, False /)
```

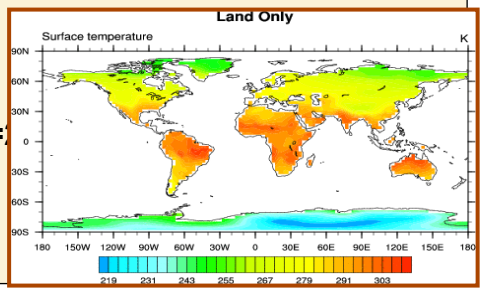
- often used in combination with array functions
  - if (**any**(**ismissing**(x))) then … [else …] end if
  - nFill = **num**(**ismissing**(x))
  - nVal = **num**(.not. **ismissing**(x))

```
if (all( ismissing(xOrig) )) then
   ....
else
   ....
end if
```

mask

- sets values to _FillValue that **DO NOT** equal mask array

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

   in   = addfile("atmos.nc","r")
   ts   = in->TS(0,:,:)
   oro = in->ORO(0,:,:)
; mask ocean
;  [ocean=0, land=1, sea_ice=
   ts    = mask(ts,oro,1)
```



- NCL has 1 degree land-sea mask available [landsea_mask]
  - load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/shea_util.ncl"
  - flags for ocean, land, lake, small island, ice shelf

where

- performs array assignments based upon a conditional array
- function **where(conditional_expression **
- **, true_value(s) **
- **, false_value(s))**
- **similar to f90 "where" statement**
- **components evaluated separately via array operations**

```
; q is an array;    q<0 => q=q+256
; f90: where(q.lt.0)  q=q+256
  q = where (q.lt.0,  q+256, q)
```

```
x = where (T.ge.0 .and. ismissing(Z) ,   a+25 ,   1.8*b)
```

```
salinity  = where (sst.lt.5 .and. ice.gt.icemax \
                  , salinity*0.9,  salinity)
```

can **not** do: y = **where**(y.eq.0, y@_FillValue, 1./y)

instead use: y = 1.0/**where**(y.eq.0, y@_FillValue, y)

dim_*_n [dim_*]

- perform common operations on an array **dim**ension
  - avg, var, sum, sort, median, rmsd, …..
- **dim_*_n** functions are **newer** (added to v5.1.1)
  - operate on a **user specified** dimension
  - use less memory
- **dim_*** functions are **original** interfaces
  - operate on **rightmost** dimension only
  - may require dimension reordering

consider: x(time,lat,lon) => x(**0,1,2**)

- function **dim_avg_n(x, n)**
  - xZon = **dim_avg_n**(x, **2**) => xZon(ntim,nlat)
  - xTim = **dim_avg_n**(x, **0**) => xTim(nlat,mlon)
- function **dim_avg (x)**
  - xZon = **dim_avg**(x) => xZon(ntim,nlat)
  - xTim = **dim_avg**(x(lat|:,lon|:,time|:)) => xTim(nlat,mlon)

conform, conform_dims

Array operations require that arrays conform

- function conform(x, r, ndim)
- function conform_dims(dims, r, ndim)
- expand array (r) to match (x) or dimensions sizes (dims)
- ndim: scalar or array indicating which dimension(s) of x or dims match the dimensions of r
- array r is 'broadcast' to array sizes of x

```
x(ntim,klev,nlat,mlon), w(nlat)        ;   x( 0 , 1 , 2 , 3 )
wx = conform (x, w, 2)                 ; wx(ntim,klev,nlat,mlon)
q = x*wx                               ; q = x* conform (x, w, 2)
wx = conform_dims ( (/ntim,klev,nlat,mlon/) , w, 2)
wx = conform_dims ( dimsizes(x), w, 2)
```

```
T(ntim, klev, nlat,mlon), dp(klev)   ;   T( 0 , 1 , 2 , 3 )
dpT = conform (T, dp, 1)             ; dpT(ntim,klev,nlat,mlon)
T_wgtAve = dim_sum_n (T*dpT, 1)/dim_sum_n(dpT, 1)
```

ind

- ind operates on 1D array only
  - returns indices of elements that evaluate to True
  - generically similar to IDL "where" and Matlab "find" [returns indices]

```
; let x[*], y[*], z[*]    [z@_FillValue]
; create triplet with only 'good' values
   iGood    = ind (.not. ismissing(z) )
   xGood    = x(iGood)
   yGood    = y(iGood)
   zGood    = z(iGood)
```

```
; let a[*],   return subscripts can be on lhs
   ii    =  ind (a.gt.500 )
   a(ii) =  3*a(ii) +2
```

- Should check the returned subscript to see if it is missing
  - if (any(ismissing(ii))) then …. end if

ind, ndtooned, onedtond

- **ind** operates on 1D array only
  - if **nD** … use with **ndtooned;** reconstruct with **onedtond, dimsizes**

```
; let q and x be nD arrays
  q1D     = ndtooned (q)
  x1D     = ndtooned (x)
  ii      = ind(q1D.gt.0. .and. q1D.lt.5)
  jj      = ind(q1D.gt.25)
  kk      = ind(q1D.lt. -50)
  x1D(ii) = sqrt( q1D(ii) )
  x1D(jj) = 72
  x1D(kk) = -x1D(kk)*3.14159
  x       = onedtond(x1D, dimsizes(x))
```

date: cd_calendar, cd_inv_calendar

- **Date/time functions:**
  - http://www.ncl.ucar.edu/Document/Functions/date.shtml
  - **cd_calendar, cd_inv_calendar**
    - deprecated**: ut_calendar, ut_inv_calendar**

```
time = (/ 17522904,  17522928, 17522952/)
time@units = "hours since 1-1-1 00:00:0.0"
date = cd_calendar(time,0)
print(date)
```

```
Variable: date
Type: float
Total Size: 72 bytes       18 values
Number of Dimensions: 2
Dimensions and sizes:   [3] x [6]
(0,0:5) 2000  1   1  0  0  0
(1,0:5) 2000  1   2  0  0  0
(2,0:5) 2000  1   3  0  0  0
```

```
date = cd_calendar(time,-2)
print(date)
```

```
Variable: date
Type: integer
Total Size: 12 bytes       3 values
Number of Dimensions: 1
Dimensions and sizes:   [3]
(0) 20000101
(1) 20000102
(2) 20000103
```

```
TIME = cd_inv_calendar (iyr, imo, iday, ihr, imin, sec  \
                        ,"hours since 1-1-1 00:00:0.0" ,0)
```

cd_calendar, ind

```
f          = addfile("...", "r")          ; f = addfiles(fils, "r")
                                          ; ALL times on file
TIME       = f->time                      ; TIME = f[:]->time
YYYYMM = cd_calendar(TIME, -1)            ; convert
ymStrt     = 190801                       ; year-month start
ymLast     = 200712                       ; year-month last
iStrt      = ind(ymStrt.eq.YYYYMM)        ; index of start time
iLast      = ind(ymStrt.eq.YYYYMM)        ;         last  time
x          = f->X(iStrt:iLast,...)        ; read only specified time period
xAvg       = dim_avg_n (x, 0)             ; dim_avg_n_Wrap
;===== specify and read selected dates; compositing
ymSelect   = (/187703, 190512, 194307, ..., 201107 /)
iSelect    = get1Dindex(TIME, ymSelect)   ; contributed.ncl
xSelect    =  f->X(iSelect,...)           ; read selected times only
xSelectAvg = = dim_avg_n (xSelect, 0)     ; dim_avg_n_Wrap
```

str_* [string functions]

- many new **str_*** functions
  - http://www.ncl.ucar.edu/Document/Functions/string.shtml
  - greatly enhance ability to handle strings
  - can be used to unpack 'complicated' string arrays

```
x = (/ "u_052134_C",  "q_1234_C",  "temp_72.55_C"/)
var_x  = str_get_field( x, 1, "_")
result:   var_x = (/"u", "q", "temp"/)          ; strings
; --------
col_x  = str_get_cols( x, 2, 4)
result: col_x  = (/"052", "123", "mp_" /)   ; strings
;---------
N = toint( str_get_cols( x(0), 3, 7) )      ; N=52134  (integer)
T = tofloat( str_get_cols( x(2), 5,9 ) )    ; T=72.55   (float)
```

system, systemfunc (1 of 2)

- **system** passes **to** the shell a command to perform an action
- NCL executes the Bourne shell (can be changed)

- create a directory if it does not exist (Bourne shell syntax)

```
DIR = "/ptmp/shea/SAMPLE"
system ("if ! test –d "+DIR+" ; then mkdir "+DIR+" ; fi")
```

- same but force the C-shell (csh) to be used

the single quotes (') prevent the Bourne shell from interpreting csh syntax

```
system ( "csh –c ' if (! –d "+DIR+") then ; mkdir "+DIR+" ; endif ' " )
```

- execute some local command

```
system ("convert foo.eps foo.png ; /bin/rm foo.eps ")
system ("ncrcat -v T,Q foo*.nc FOO.nc ")
system ("/bin/rm –f " + file_name)
```

system, systemfunc (1 of 2)

- **systemfunc** returns to NCL information **from** the system
- NCL executes the Bourne shell (can be changed)

```
UTC = systemfunc("date")                                    ; *nix date
Date = systemfunc("date '+%a %m%d%y %H%M' ")    ; single quote
fils = systemfunc ("cd /some/directory ; ls foo*nc")    ; multiple cmds
city = systemfunc ("cut -c100-108 " + fname)
```

User-built Functions and Procedures (1 of 4)

- **two ways to load existing files w functions/proc**
  - **load** "/path/my_script.ncl"
  - use environment variable: NCL_DEFAULT_SCRIPTS_DIR
- **must be loaded prior to use**
  - unlike in compiled language
- **avoid function conflict** (**undef**)

```
undef ("mult")
function mult(x1,x2,x3,x4)
begin
    return ( x1*x2*x3*x4)
end
```

```
load "/some/path/mult.ncl"
begin
  x = mult(4.7, 34, 567, 2)
end
```

```
undef ("mult")
function mult(x1,x2,x3,x4)
begin
    return ( x1*x2*x3*x4)
end

begin
  x = mult(4.7, 34, 567, 2)
end
```

User-Built Functions and Procedures (2 of 4)

- **Development process similar to Fortran/C/IDL**
- **General Structure:**

```
undef ("function_name")          ; optional
function function_name (declaration_list)
local local_identifier_list      ; optional
begin                            ; required
   statements
   return (return_value)         ; required
end                              ; required
```

```
undef ("procedure_name")         ; optional
procedure procedure_name (declaration_list)
local local_identifier_list      ; optional
begin                            ; required
  statements
end                              ; required
```

User-Built Functions and Procedures (3 of 4)

- **arguments are passed by reference [fortran]**
- **constrained argument specification:**
  - require specific type, dimensions, and size
  - procedure ex(data[*][*]:float,res:logical,text:string)
- **generic specification:**
  - type only
  - function xy_interp(x1:numeric, x2:numeric)
- **no type, no dimension specification:**
  - procedure whatever (a, b, c)
- **combination**
  - function ex (d[*]:float, x:numeric, wks:graphic, y[2], a)
- **function prototyping**
  - built-in functions are prototyped

User-Built Functions and Procedures (4 of 4)

- **additional ('optional') arguments possible**
- **attributes associated with one or more arguments**
  - often implemented as a separate argument (not required)
  - procedure ex(data[*][*]:float, text:string,**optArg:logical**)

```
optArg          = True
optArg@scale  = 0.01
optArg@add    = 1000
optArg@wgts   = (/1,2,1/)
optArg@name = "sample"
optArg@array  = array_3D
ex(x2D, "Example", optArg)
```

```
procedure ex(data, text, opt:logical)
begin
      :
  if (opt .and. isatt(opt,"scale")) then
     d = data*opt@scale
  end if
  if (opt .and. isatt(opt,"wgts")) then
      :
  end if
  if (opt .and. isatt(opt,"array")) then
     xloc3D = opt@array_3D  ; nD arrays
  end if                    ; must be local before use
end
```

Computations and Meta Data

- **computations can cause loss of meta data**
  - y = x ; variable to variable transfer; all meta copied
  - T = T+273 ; T retains all meta data
    - T@units = "K" ; user responsibility to update meta
  - z = 5*x ; z will have no meta data

- **built-in functions cause loss of meta data**
  - Tavg = **dim_avg_n**(T, 0)
  - s = **sqrt**(u^2 + v^2)

- **vinth2p is the exception**
  - retains coordinate variables
  - http://www.cgd.ucar.edu/csm/support/Data_P/vert_interp.shtml
    - hybrid to pressure (sigma to pressure) + other examples

Ways to Retain Meta Data (1 of 3)

- use copy functions in **contributed.ncl**
  - **copy_VarMeta** (coords + attributes)
  - **copy_VarCoords**
  - **copy_VarAtts**

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
  f = addfile("dummy.nc", "r")
  x = f->X                          ; (ntim,nlat,mlon)
 ; ---------------- calculations--------------------------
   xZon = dim_avg _n(x, 2)          ; xZon(ntim,nlat)
; ----------------copy meta data-------------------------
  copy_VarMeta(x, xZon)             ; xZon(time,lat)
end
```

Ways to Retain Meta Data (2 of 3)

- **use wrapper functions** (eg:)
  - dim_avg_n_Wrap
  - dim_variance_n_Wrap
  - dim_stddev_n_Wrap
  - dim_sum_n_Wrap
  - dim_rmsd_n_Wrap
  - smth9_Wrap
  - g2gsh_Wrap
  - g2fsh_Wrap
  - f2gsh_Wrap
  - f2fsh_Wrap
  - natgrid_Wrap
  - f2fosh_Wrap
  - g2gshv_Wrap
  - g2fshv_Wrap
  - f2gshv_Wrap
  - f2fshv_Wrap
  - f2foshv_Wrap
  - linint1_Wrap
  - linint2_Wrap
  - linint2_points_Wrap
  - eof_cov_Wrap
  - eof_cov_ts_Wrap
  - zonal_mpsi_Wrap
  - etc

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

  f = addfile("dummy.nc", "r")
  x = f->X                                ; time,lev,lat,lon  (0,1,2,3)
  xZon = dim_avg_n_Wrap(x, 3)    ; xZon will have meta data
```

Ways to Retain Meta Data (3 of 3)

- use **variable to variable transfer** + **dimension reduction** to prefine array before calculation
  - requires that user know a priori the array structure

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

  f = addfile("dummy.nc", "r")
  x = f->X                                ; x(time,lev,lat,lon)
; -------- var-to-var transfer + dim reduction--------
  xZon  = x(:,:,:,0)                    ; xZon(time,lev,lat)
; --------------------calculations-----------------------
   xZon = dim_avg_n (x, 0)
   xZon@op = "Zonal Avg: "+x@long_name
```

  - xZon will have all appropriate meta data of x
  - NCL will add an attribute [here: xZon@lon = lon(0)]

regrid: bilinear interpolation
linint2

- Cartesian, global or limited area **rectilinear** grids only
- wrapper versions preserve attributes
  and create coordinate variables
- missing data allowed

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

  f      = addfile ("/fs/cgd/data0/shea/CLASS/T2m.nc", "r")
 T       = f->T
 TLI     = linint2_Wrap(T&lon, T&lat, T, True,  LON, LAT, 0 )
```

regrid: areal conservative interpolation
area_conserve_remap,
area_conserve_remap_Wrap

- **global rectilinear grids only**
- **_Wrap** preserves attributes; creates coordinate variables
- missing data (_FillValue) *NOT* allowed

In particular, use for (say) flux or precipitation interpolation

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

f   = addfile (“GPCP.nc", "r")
p  = f->PRC
P =  area_conserve_remap_Wrap (p&lon, p&lat, p          \
                                ,newlon, newlat, False)
```

regrid: areal average interpolation
area_hi2lores, area_hi2lores_Wrap

- rectilinear grids; can be limited area
- _Wrap preserves attributes; creates coordinate variables
- missing data allowed
- designed for TRMM data

NOT strictly 'conservative' … but close 50S to 50N

Use **area_hi2lores** for highly structured fields => lower res
Use **linint2** for low resolution=> high resolution

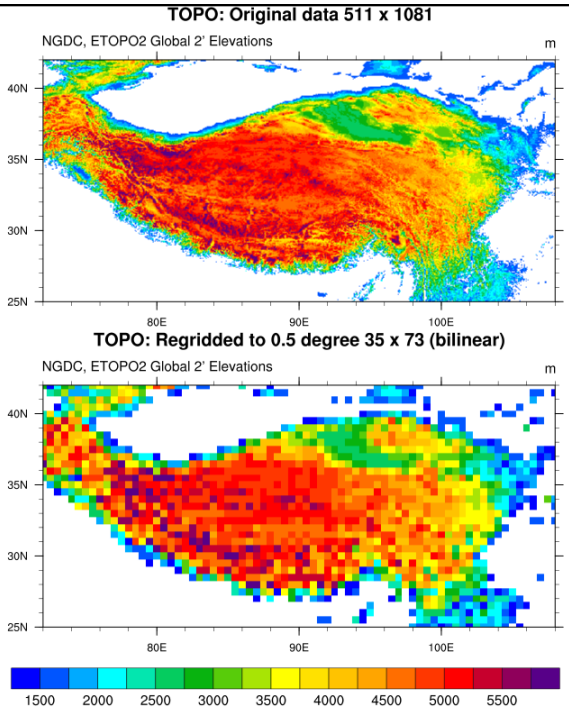
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"

f   = addfile (trmm.nc", "r")
p  = f->PRC
P =  area_hi2lores_Wrap (p&lon, p&lat, p, True, LON, LAT, 0 )
```

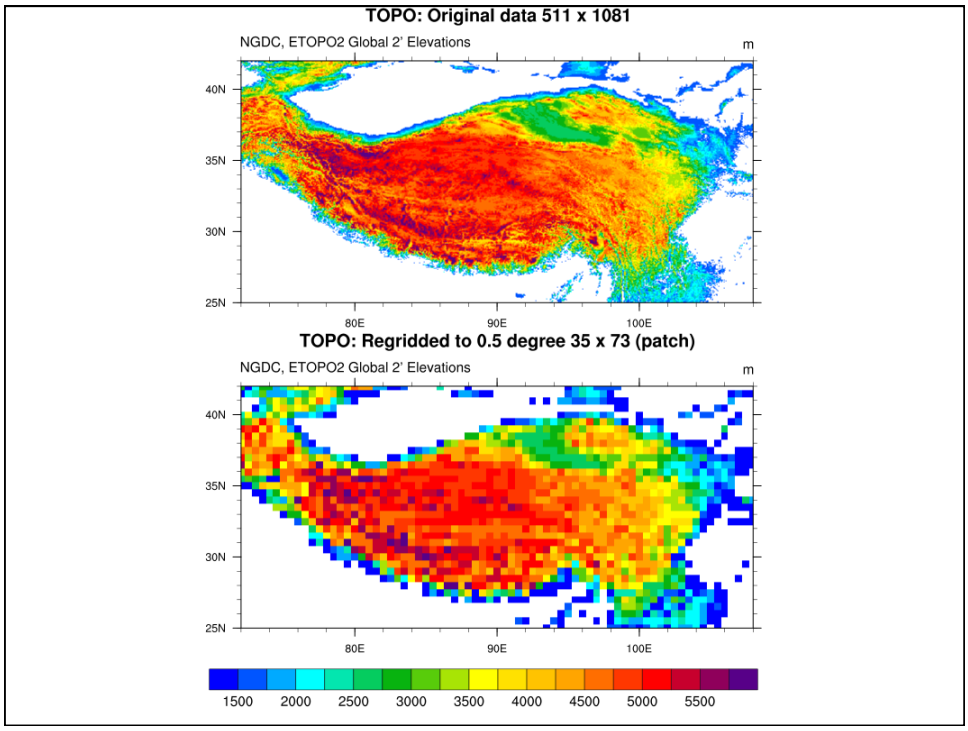
NCL-ESMF regridding

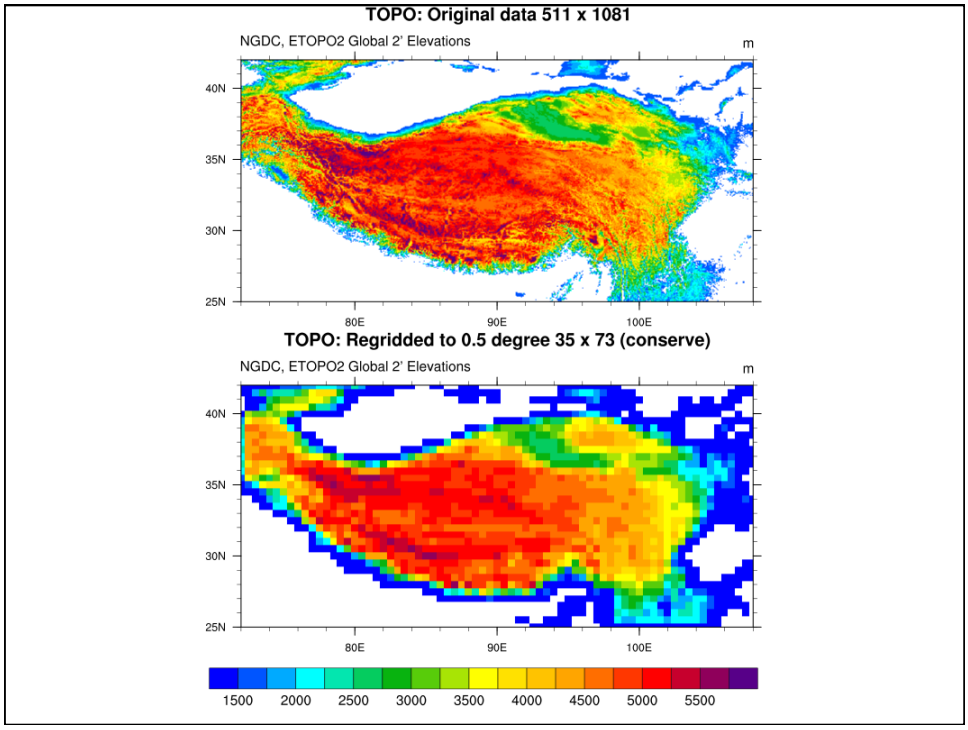
- Integrated in conjunction with NOAA Cooperative Institute for Research in Environmental Sciences
- Available in NCL V6.1.0-beta (May 2012)
- Works with **structured and unstructured** grids
- Multiple interpolation methods available
  - Bilinear
  - Conservative
  - Patch
- Can handle masked points
- Better treatment for values at poles
- Works on global or regional grids
- Can run in parallel or single-threaded mode

ESMF versus other regridding software

- Is ESMF regridding better than SCRIP?
  - ESMF takes extra care at the pole; does interpolation in 3D cartesian space.
  - ESMF's 'patch' method has no equivalent in SCRIP. Weight generation is slow but the results can be used to estimate derivatives (i.e. gradients, ocean curl)
  - The “conserve” method is fast even on one processor
- NCL can be used to regrid data on GRIB{1/2} and HDF{4/5} files, and written to netCDF




TOPO: Original data 511 x 1081
NGDC, ETOPO2 Global 2' Elevations
m
40N
35N
30N
25N
80E
90E
100E
TOPO: Regridded to 0.5 degree 35 x 73 (patch)
NGDC, ETOPO2 Global 2' Elevations
m
40N
35N
30N
25N
80E
90E
100E
1500
2000
2500
3000
3500
4000
4500
5000
5500


TOPO: Original data 511 x 1081
NGDC, ETOPO2 Global 2' Elevations
m
40N
35N
30N
25N
80E
90E
100E
TOPO: Regridded to 0.5 degree 35 x 73 (conserve)
NGDC, ETOPO2 Global 2' Elevations
m
40N
35N
30N
25N
80E
90E
100E
1500
2000
2500
3000
3500
4000
4500
5000
5500

Interpolation methods

- "bilinear" - the algorithm used by this application to generate the bilinear weights is the standard one found in many textbooks. Each destination point is mapped to a location in the source mesh, the position of the destination point relative to the source points surrounding it is used to calculate the interpolation weights.
- "patch" - this method is the ESMF version of a technique called "patch recovery" commonly used in finite element modeling***. It typically results in better approximations to values and derivatives when compared to bilinear interpolation.***
- "conserve" - this method will typically have a larger interpolation error than the previous two methods, but will do a much better job of preserving the value of the integral of data between the source and destination grid.

ESMF Interpolation Steps

The basic steps of NCL/ESMF regridding involve:

1. Reading or generating the "source" grid.
2. Reading or generating the "destination" grid.
3. Creating special NetCDF files that describe these two grids.
4. *Generating a NetCDF file that contains the weights.
5. Applying the weights to data on the source grid, to interpolate the data to the destination grid.
6. Copying over any metadata to the newly regridded data.

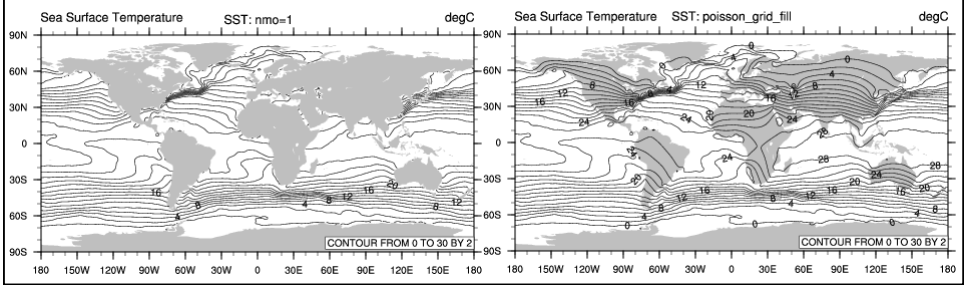
*This is the most important step. Once you have a weights file, you can skip steps #1-4 if you are interpolating data on the same grids

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

poisson_grid_fill

- replaces all _FillValue grid ponts
  - Poisson's equation solved via relaxation
  - values at non-missing locations are used as boundary cond.
  - works on **any** grid with spatial dimensions [*][*]

```
in   = addfile  (Ocean.nc","r")
sst  = in->SST
poisson_grid_fill (sst, True, 1, 1500, 0.02, 0.6, 0)
```



Example: Compositing

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

  t1  = (/ 15, 37, 95, 88,90 /)          ; cd_calendar, ind, get1Dindex
  t2  = (/  1, 22, 31, 97, 100, 120/)

  f   = addfile("01-50.nc",  "r")
 T1 = f->T(t1,:,:,:)                     ; T(time,lev,lat,lon)
 T2 = f->T(t2,:,:,:)
                                         ; composite averages
 T1avg = dim_avg_n_Wrap(T1, 0) ; (lev,lat,lon)
 T2avg = dim_avg_n_Wrap(T2, 0)

  Tdiff  = T2avg                         ; trick to transfer meta data
  Tdiff  = T2avg - T1avg                 ; difference
  Tdiff@long_name = T2@long_name + ": composite difference"
------
Also use coordinate subscripting: let "time" have units yyyymm
  t1    = (/ 190401, 191301, 192001, ……, 200301/)
 T1    = f->T({t1},:,:,:))
```

Empirical Orthogonal Functions (EOFs)

- **principal components, eigenvector analysis**
- **provide efficient representation of variance**
  - May/may not have dynamical information
- **successive eigenvalues should be distinct**
  - if not, the eigenvalues and associated patterns are noise
  - 1 from 2, 2 from 1 and 3, 3 from 2 and 4, etc
  - North et. al (*MWR*, July 1982: eq 24-26) provide formula
  - Quadrelli et. Al (*JClimate*, Sept, 2005) more information
- **geophysical variables: spatial/temporal correlated**
  - no need sample every grid point
    - no extra information gained
    - oversampling increases size of covar matrix + compute time
- **patterns are domain dependent**

Calculating EOFS, writing a NetCDF file (next page)

```
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"
                                                    ; rectilinear
f  = addfile("erai_1989-2009.mon.msl_psl.nc","r")   ; open file
p  = f->SLP(::12,{0:90},:)                          ; (20,61,240)

w  = sqrt(cos(0.01745329*p&latitude) )              ; weights(61)
wp = p*conform(p, w, 1)                             ; wp(20,61,240)
copy_VarCoords(p, wp)

x  = wp(latitude|:,longitude|:,time|:)              ; reorder data
neof   = 3
eof     = eofunc_Wrap(x, neof, False)
eof_ts = eofunc_ts_Wrap (x, eof, False)

printVarSummary( eof )                  ; examine EOF variables
printVarSummary( eof_ts )
```

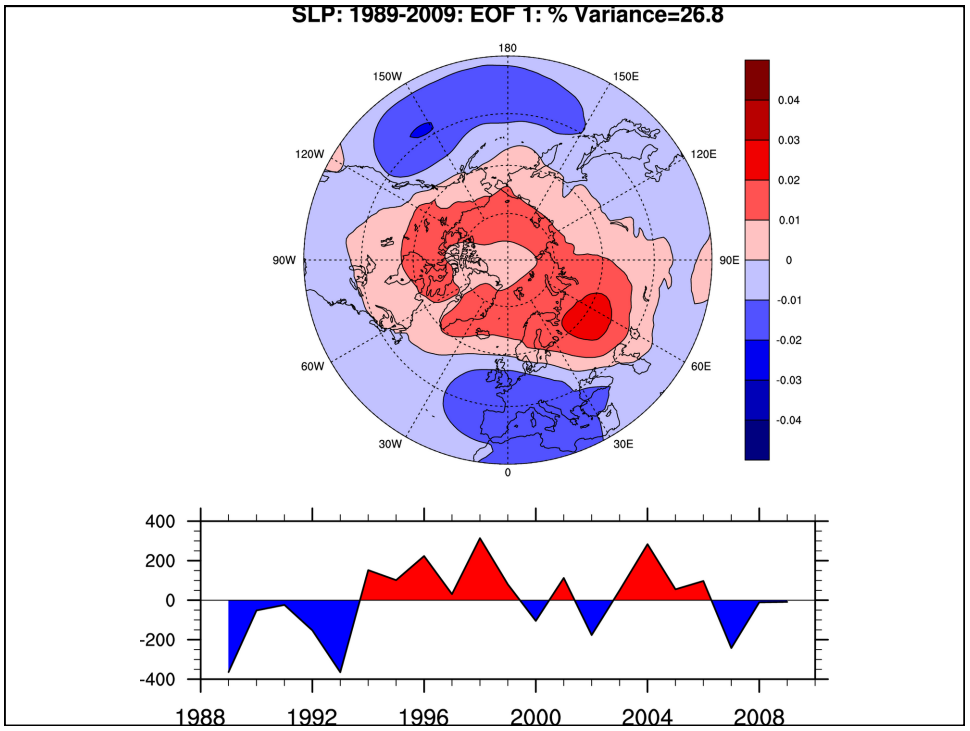
"printVarSummary" output

```
Variable: eof
Type: float
Total Size: 175680 bytes
            43920 values
Number of Dimensions: 3
Dimensions and sizes:        [evn | 3] x [latitude | 61] x [longitude | 240]
Coordinates:
            evn: [1..3]
            latitude: [ 0..90]
            longitude: [ 0..358.5]
Number Of Attributes: 6
  eval_transpose :  ( 47.2223, 32.42917, 21.44406 )
  eval :  ( 34519.5, 23705.72, 15675.61 )
  pcvar :          ( 26.83549, 18.42885, 12.18624 )
  matrix :         covariance
  method :         transpose
  _FillValue :     1e+20


Variable: eof_ts
Type: float
Total Size: 252 bytes
            63 values
Number of Dimensions: 2
Dimensions and sizes:        [evn | 3] x [time | 21]
Coordinates:
            evn: [1..3]
            time: [780168..955488]
Number Of Attributes: 3
  ts_mean :        ( 3548.64, 18262.12, 20889.75 )
  matrix :         covariance
  _FillValue :     1e+20
```

writing a NetCDF file

```
; Create netCDF: no define mode [simple approach]
system("/bin/rm -f EOF.nc")             ; rm any pre-existing file
fout         = addfile("EOF.nc", "c")   ; new netCDF file
fout@title   = "EOFs of SLP 1989-2009"
fout->EOF    = eof
fout->EOF_TS = eof_ts
```

Graphics: http://www.ncl.ucar.edu/Applications/Scripts/eof_2.ncl



External Fortran-C Codes

external codes, Fortran, C, or local commerical libraries may be executed from within NCL

- **generic process**
  - develop a **wrapper** (interface) to transmit arguments
  - compile the external code (eg, f77, f90, cc)
  - link the external code to create a **shared object**

  process simplified by operator called WRAPIT

- **specifying where shared objects are located**
  - **external statement**
    - **external "/dir/code.so"**
  - system environment variable:
    - LD_LIBRARY_PATH
  - NCL environment variable:
    - NCL_DEF_LIB_DIR
    - external functions need not have **::** before use

NCL/Fortran Argument Passing

- **arrays: NO reordering required**
  - x(time,lev,lat,lon) <=map=> x(lon,lat,lev,time)

- **ncl: x(N,M) => value <= x(M,N) :fortran** [M=3, N=2]
  - x(0,0) => 7.23 <= x(1,1)
  - x(0,1) => -12.5 <= x(2,1)
  - x(0,2) => 0.3 <= x(3,1)
  - x(1,0) => 323.1 <= x(1,2)
  - x(1,1) => -234.6 <= x(2,2)
  - x(1,2) => 200.1 <= x(3,2)

- **numeric types must match**
  - integer <==> integer
  - double <==> double
  - float <==> real

- **Character-Strings: a nuisance [C,Fortran]**

Example: Linking to Fortran 77

STEP 1: quad.f

```
C  NCLFORTSTART
      subroutine cquad(a,b,c,nq,x,quad)
      dimension x(nq), quad(nq)
C  NCLEND
        do i=1,nq
           quad(i) = a*x(i)**2 + b*x(i) + c
        end do
        return
        end

C  NCLFORTSTART
      subroutine prntq (x, q, nq)
      integer nq
      real x(nq), q(nq)
C  NCLEND
      do i=1,nq
         write (*,"(i5, 2f10.3)") i, x(i), q(i)
      end do
      return
      end
```

STEP 2: quad.so

```
WRAPIT quad.f
```

STEPS 3-4

```
external QUPR   "./quad.so"
begin
  a  = 2.5
  b  = -.5
  c   = 100.
  nx = 10
  x = fspan(1., 10., 10)
  q = new (nx, float)
  QUPR::cquad(a,b,c, nx, x,q)
  QUPR::prntq  (x, q, nx)
end
```

Example: Linking F90 routines

STEP 1: quad90.stub

```
C  NCLFORTSTART
      subroutine cquad (a,b,c,nq,x,quad)
      dimension x(nq), quad(nq)  ! ftn default
C  NCLEND
C  NCLFORTSTART
      subroutine prntq (x, q, nq)
      integer nq
      real x(nq), q(nq)
C  NCLEND
```

prntq_I.f90

```
module prntq_I
interface
subroutine prntq (x,q,nq)
real, dimension(nq) :: x, q
integer, intent(in)    :: nq
end subroutine      end interface
end module
```

cquad_I.f90

```
module cquad_I
interface
subroutine cquad (a,b,c,nq,x,quad)
real, intent(in)      :: a,b,c
integer, intent(in) :: nq
real,dimension(nq), intent(in) :: x
real,dimension(nq),intent(out) :: quad
end subroutine       end interface
end module
```

quad.f90

```
subroutine cquad(a, b, c, nq, x, quad)
     implicit none
     integer , intent(in) :: nq
     real , intent(in)      :: a, b, c, x(nq)
     real , intent(out)    :: quad(nq)
     integer                 :: i       ! local
     quad = a*x**2 + b*x + c     ! array
     return
end subroutine cquad

subroutine prntq(x, q, nq)
     implicit none
     integer , intent(in) :: nq
     real , intent(in)      :: x(nq),  q(nq)
     integer                 :: i          ! local
     do i = 1, nq
       write (*, '(I5, 2F10.3)') i, x(i), q(i)
     end do
    return
end
```

STEP 2: quad90.so

```
WRAPIT –pg quad90.stub printq_I.f90 \
          cquad_I.f90 quad.f90
```

STEP 3-4: same as previous

```
ncl  <  PR_quad90.ncl
```

NCL as a scripting tool

```
mssi  = getenv ("MSSOCNHIST")  ; get environment variable
diri    = "/ptmp/user/"                ; dir containing input files
fili     = "b20.007.pop.h.0"          ; prefix of input files
diro   = "/ptmp/user/out/"          ; dir containing output files
filo    = "b20.TEMP."                 ; prefix of output files

nyrStrt = 300                          ; 1st year
nyrLast= 999                           ; last year
do nyear=nyrStrt,nyrLast
    print ("---- "+nyear+" ----")
                                       ; read 12 months for nyear
    msscmd = "msrcp –n 'mss:" +mssi+ fili+nyear+ "-[0-1][0-9].nc' "+diri+"."
    print ("msscmd="+msscmd)
    system (msscmd)
                                       ; strip off the TEMP variable
    ncocmd = "ncrcat –v TEMP " +diri+fili+"*.nc "+ diro+filo+nyear+".nc"
    print ("ncocmd="+ncocmd)
    system (ncocmd)
                                      ; remove the 12 monthly files
    rmcmd = "'/bin/rm' "+diri+fili+nyear+ ".nc"
    print ("rmcmd="+rmcmd)
    system (rmcmd)
end do
```

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