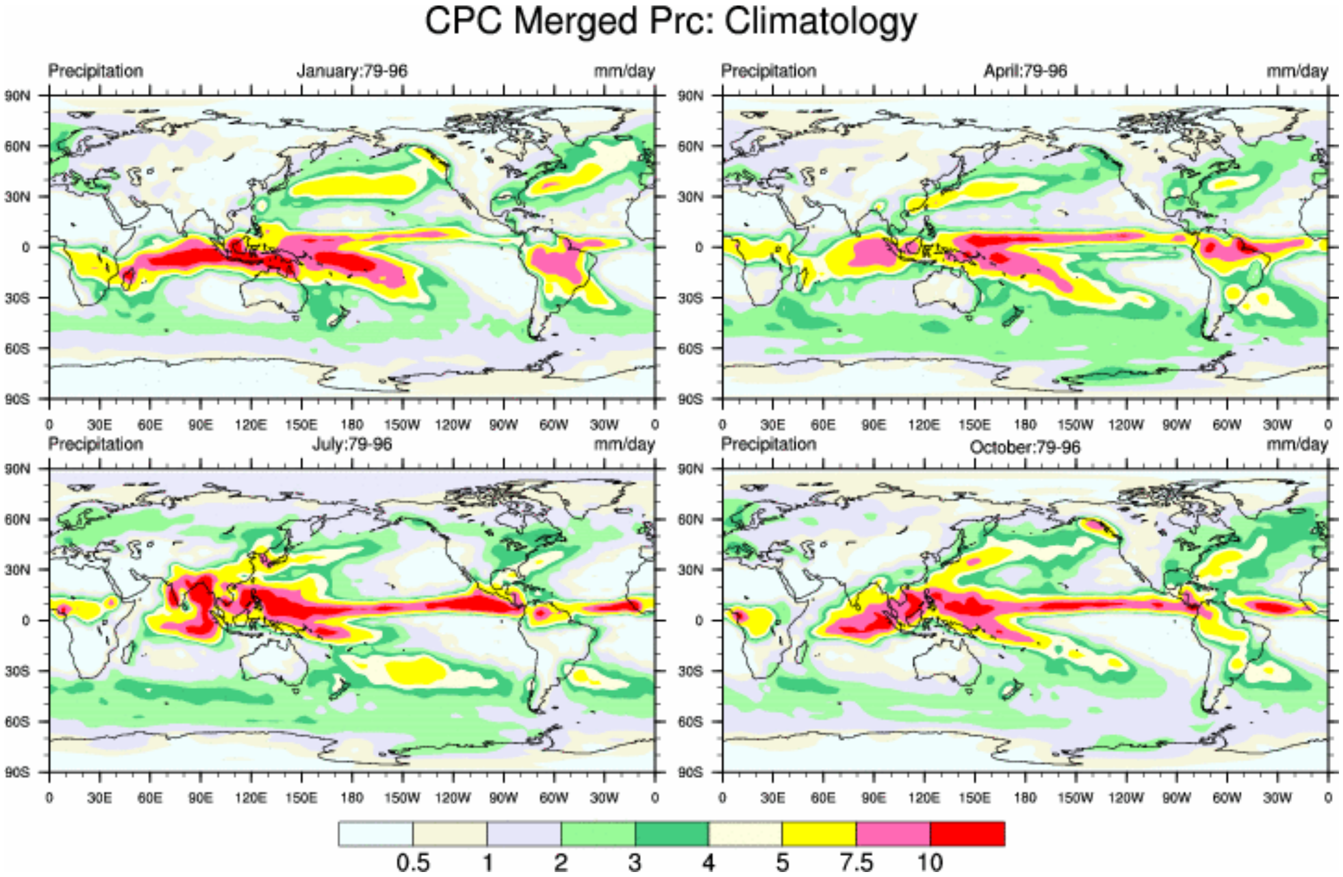
NCL Data Processing



Dennis Shea

National Center for Atmospheric Research

NCAR is sponsored by the National Science Foundation

Built-in Functions and Procedures

- NCL continually adds I/O, Graphics, Functions
- Objective is to meet evolving community needs

  internal (CGD, …)
  ncl-talk
  workshops

Built-in Functions and Procedures

- use whenever possible
- learn and use **utility** functions (any language)
  - all, any, conform, ind, ind_resolve, dimsizes, num
  - fspan, ispan, ndtooned, onedtond, reshape
  - mask, ismissing, str*
  - system, systemfunc
  - cd_calendar, cd_inv_calendar
  - to* (toint, tofloat, …); round, short2flt, ….

  - sort, sqsort, dim_pqsort_n, dim_sort_n
  - generate_sample_indices (6.3.0) [bootstrap]
  - get_cpu_time, wallClockElapseTime

Built-in Functions and Procedures

common computational functions

- **dim_*_n**, **where**
- avg, stddev, min, max, ….
- escorc, pattern_cor, esccr, esacr (correlation)
- rtest, ttest, ftest, kolsm2_n
- regression/trend: regline_stats, trend_manken_n (6.3.0)
- filtering: filwgts_lanczos, dim_bfband_n (6.3.0)
- eofunc, eofunc_ts, eof_varimax
- diagnostics: MJO, Space-Time, POP, kmeans (6.3.0)
- regridding: linint2, ESMF, …
- random number generators
- climatology & anomaly (hrly, daily, monthly,…)
- wgt_areaave, wgt_arearmse,…
- fft: ezfftf, ezfftb, fft2d, specx_anal, specxy_anal
- spherical harmonic: synthesis, analysis, div, vort, regrid,

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
```

ispan, sprinti

People want ‘zero filled’ two digit field

```
month = (/ ”01”,”02”, ”03”,”04”, ”05”,”06”  \
           , ”07”,”08”, ”09”,”10”, ”11”,”12”  /)

day     = (/ ”01”,”02”, ”03”,”04”, ”05”,”06”  \
           , ”07”,”08”, ”09”,”10”, ”11”,”12”  \
           , ….., “30”,”31”)
```

cleaner / nicer code:

```
month = sprinti("%0.2i", ispan(1,12,1) )
day     = sprinti("%0.2i", ispan(1,31,1) )

year    = “” + ispan(1900,2014,1)
```

fspan(start:numeric, finish:numeric, n:integer)

- 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)
```

```
d = fspan(-89.125,  9.3d0, 100)
print(d)  ; type double
```

```
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, num, all, any, .not.

- **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)

          = (/ False, False, True, False, True, True, False /)
```

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

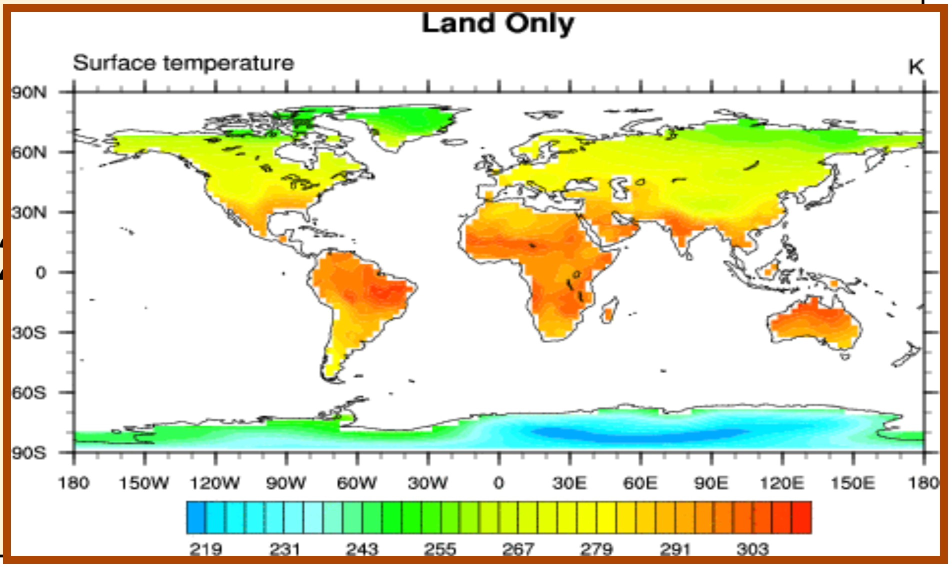
```
if (any( ismissing(xOrig) )) then
    ….
else
    ….
end if
```

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 exp.
- function **where(conditional_expression \**
- **, True_value(s) \**
- **, False_value(s))**
- **components evaluated separately via array operations**

```
; q is an array;    q<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(s)
  - **dim_avg_n** (**stddev, sum**, **sort**, **median**, **rmsd**,…)

- **dim_*_n** functions operate on a **user specified** dimension
  - use less memory, cleaner code than older **dim_***

- **dim_*** functions are **original** (old) interfaces; **deprecated**
  - operate on **rightmost** dimension only
  - may require dimension reordering
  - kept for backward compatibility

Recommendation: use **dim_*_n**

dim_*_n [dim_*]

Consider: x(ntim,nlat,mlon) => x(**0,1,2**)

- function **dim_avg_n(** x, **n**) => operate on dim **n**
  - xZon = **dim_avg_n**(x, **2**) => xZon(ntim,nlat)
  - xTim = **dim_avg_n**(x, **0**) => xTim(nlat,mlon)

Consider: x(ntim,nlat,mlon)

- function **dim_avg (x)** => operate on rightmost dim
  - 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**) on dimensions sizes (**dims**)
- **ndim:** scalar or array indicating which dimension(s)

  of **x** or **dims** match the dimensions of array

- array **r** is ‘broadcast’ (replicated) to array sizes of **x**

```
x(nlat,mlon), w(nlat)            ;   x(  0   ,  1)    ,   w( 0 )
wx   = conform (x,  w,  0)       ;  wx(nlat,mlon)
xwx = x*wx                       ;  xwx = x* conform (x,  w,  0)
xar  = sum(xwx)/sum(wx)          ; area avg (wgt_areaave,…)
```

conform, conform_dims

Let T be (30,30,64,128), P be (30).

```
         (  0,  1,  2,   3 )          <= dimension numbers
  theta  = T*(1000/conform(T,P,1))^0.286

  theta(30,30,64,128)
```

```
T(ntim, klev, nlat,mlon), dp(klev)
  (  0   ,  1    ,  2   ,   3    ),      (  0  )

dpT = conform (T,  dp,  1)              ;  dpT(ntim,klev,nlat,mlon)

T_wgtAve = dim_sum_n (T*dpT, 1)/dim_sum_n(dp, 0)

                T_wgtAve(ntim,nlat,mlon)
delete(dpT)                              ; not necessary
```

conform, conform_dims

```
function pot_temp_n (p:numeric, t:numeric, npr[*]:integer, opt:integer)
; Compute potential temperature; aby dimensionality
begin
  rankp  = dimsizes(dimsizes(p))
  rankt  = dimsizes(dimsizes(t))
  p0     = 100000.                              ; default [units = Pa]
  if (rankp.eq.rankt) then
     theta = t*(p0/p)^0.286                      ; conforming arrays
  else
     theta = t*(p0/conform(t,p,npr))^0.286  ; non-conforming
  end if
  theta@long_name = "potential temperature”   ; meta data
  theta@units     = "K”
return( theta )
end
```

ind

- **ind** operates on 1D array only
  - returns indices of elements that evaluate to **T**rue
  - 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))
```

- **Recommendation**: isolate above in user function

User function: ind, ndtooned, onedtond

```
function merge(q, x)  ; merge q and x
begin
    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))
    x@info = “x after merge with q”
    return(x)
end
```

date: cd_calendar, cd_inv_calendar

- **Date/time functions:**
  - http://www.ncl.ucar.edu/Document/Functions/date.shtml

```
time = (/ 17522904,  17522928, 17522952/)

time@units       = "hours since 1-1-1 00:00:0.0"

time@calendar = "gregorian"            ; default is 'gregorian'

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(YYYYMM.eq.ymStrt)           ; index of start time
  iLast      = ind(YYYYMM.eq.ymLasrt) ;                 last  time
  x          = f->X(iStrt:iLast,...)            ; read only specified time period; f[:]
  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], to*

- 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 [  ! –d  ‘ “+DIR+” ]  ;  then ;  mkdir  -p “+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)
```

preview: user written functions

- **http://www.cgd.ucar.edu/~shea/meteo.ncl**
- Pot. Temp; Static Stability; Pot. Vorticity (hybrid, isobaric)
- Advect Variable (q): u*(dq/dx+ v*dq/dy)

- **http://www.cgd.ucar.edu/~shea/reg_func.ncl**
- Multiple Linear regression: ANOVA
- Simple Linear Regression: ANOVA