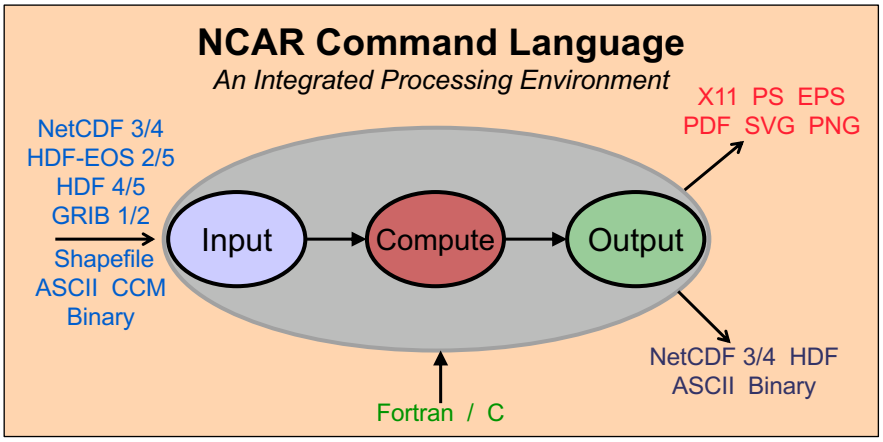
Introduction to NCL Data Analysis



Mary Haley
(with thanks to Dennis Shea)

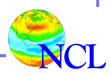
Sponsored by the National Science Foundation

NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Regridding functions
- Creating your own functions and procedures
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices

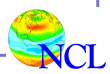
Goals

- Provide general overview
- Introduce most common functions
- Show how to write your own
- Provide **tips**, best practices, links
- Intersperse interactive demos



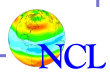
NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Regridding functions
- Creating your own functions and procedures
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices



Functions and procedures overview (1 of 2)

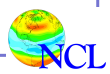
- Over 1,400 documented functions and procedures
  - "core"
  - file I/O
  - computational
  - graphics
- Developed with scientific community in mind
- Users have contributed code, bug fixes, suggestions
- Can develop your own - contribute back to NCL!



Functions and procedures overview (2 of 2)

- Functions return a value
- Procedures do not return a value
- Two types: built-in and NCL-based
- Organized by type and category
- Every single function documented here:
  http://www.ncl.ucar.edu/Document/Functions/

Sample categories		
General applied math	ESMF regridding	Meteorology
Bootstrap	Extreme values	Spherical harmonics
Date	Heat stress	Statistics
EOFs	Interpolation	WRF

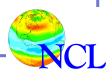


"Built-in" versus "NCL-based"

- Built-in functions are "compiled" into NCL
  - Written in C or Fortran
  - Cannot be easily modified

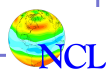
… WRAPIT is your friend…

- NCL-based functions are written in NCL and can be modified



NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Regridding functions
- Creating your own functions and procedures
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices

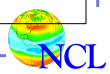


Essential "utility" functions

dimsizes
any • all • num
print
printMinMax • printVarSummary
write_matrix
where • mask
conform • conform_dims
dim_*xxxx* • dim_*xxxx*_Wrap
dim_*xxxx*_n • dim_*xxxx*_n_Wrap
ind
ismissing • default_fillvalue
ndtooned • onedtond • reshape
isatt • isvar
ispan • fspan
sprinti • sprintf
str_get_cols • str_get_field

copy_VarMeta • copy_VarAtts
qsort • sqsort
cd_calendar • cd_string
yyyymm_to_yfrac
typeof
system • systemfunc
get_cpu_time • unique_string
new • delete

Not an exhaustive list!



dimsizes

- Returns the dimension sizes of a variable
- For a 1D array, same as the length of the array:

Script

```
x    = (/1,2,3,4,5/)
xlen = dimsizes(x)
print("len = " + xlen)
```

Output

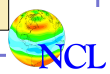
```
(0)      len = 5
```

Tip To get rid of "(0)" output:

```
ncl -n myscript.ncl
```

Output

```
len = 5
```

dimsizes – get rank of variable

Script

```
f = addfile("uv300.nc","r")
v = f->V  ; Read V into local variable v
dimv = dimsizes(v)
print(dimv)
rank = dimsizes(dimv)
print ("rank="+rank)
```

Output

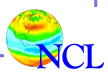
```
(0)   2
(1)   64
(2)   128
(0)   rank=3
```

Tip

```
print(str_join(""+dimv," x "))
```

Output

```
2 x 64 x 128
```

dimsizes / strlen

Script

```
strs = (/"January","February","March"/)
dims = dimsizes(strs)
print(dims)
print(dimsizes(strs(0)))  ; not what you think

;---Print length of string
print("len of first str = "+strlen(strs(0)))
```

Output

```
(0)   3
(0)   1
(0)   len of first str = 7
```


ispan / yyyymm_time

Returns a 1D array of integers, given start, finish, stride

Script

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

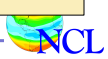
Output

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

Tip To create YYYYMM array:

```
ym = yyyymm_time(1850,2001,"integer")
print(ym)
```

```
Variable: ym
Type: integer
Dimensions and sizes: [time | 1824]
Coordinates:
            time: [185001..200112]
Number Of Attributes: 2
  long_name :   time
  units :       YYYYMM
(0)     185001
(1)     185002
(2)     185003
(3)     185004
. . .
```

ispan / sprinti

Tip If you want "zero-filled" two digit fields:

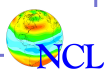
Instead of:

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

Use:

```
imonth = ispan(1,12,1)
smonth = sprinti("%0.2i",imonth)
print(smonth)
```

```
Variable: smonth
Type: string
(0)    01
(1)    02
(2)    03
(3)    04
 . . .
(8)    09
(9)    10
(10)   11
(11)   12
```

fspan / sprintf

Returns a 1D array of floats or doubles, given start, end, # points

Script

```
xflt = fspan(-89.125, 9.3, 100)
xdbl = fspan(-89.125, 9.3d, 100)

print(xflt)
print(xdbl)
```

Tip "Pretty-print"

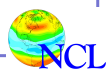
```
print(sprintf("%7.3f",xflt))
```

```
 (0)  -89.125
 (1)  -88.131
 (2)  -87.137
 (3)  -86.142
  . . .
(78)  -11.578
(79)  -10.584
(80)   -9.590
(81)   -8.595
  . . .
```

Output

```
Variable: xflt
Type: float
(0)   -89.125
(1)   -88.13081
(2)   -87.13662
. . .
(97)  7.311616
(98)  8.305808
(99)  9.3
```

```
Variable: xdbl
Type: double
(0)   -89.125
(1)   -88.13080808080808
(2)   -87.1366161616161
. . .
(97)  7.311616161616158
(98)  8.305808080808077
(99)  9.300000000000001
```

any / all / num

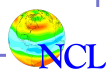
Testing and counting array values

Sniplets

```
If(any(x.lt.0)) then
  print("Error: all values must be > 0")
end if
```

```
maxlat = 30.5
if(all(lat.ge.maxlat)) then
  print("Warning: all your lat values are > " + maxlat)
end if
```

```
min_temp = 0
max_temp = 100
count = num(temp.ge.min_temp.and.temp.le.max_temp)
print("There are " + count + " values in the range " +  \
      min_temp + " to " + max_temp)
```

ismissing

Essential function for testing for missing values

Script

```
x = (/ 1, 2, -99, 4, -99, -99, 7 /)
print(ismissing(x))
```

Output

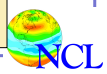
```
(0)     False
(1)     False
(2)     False
(3)     False
(4)     False
(5)     False
(6)     False
```

Script

```
x = (/ 1, 2, -99, 4, -99, -99, 7 /)
print(ismissing(x))

x@_FillValue = -99
print(ismissing(x))
```

Output

```
(0)     False
(1)     False
(2)     True
(3)     False
(4)     True
(5)     True
(6)     False
```

ismissing: use with any/all/num

```
If(any(ismissing(x)) then
  print("Warning: x contains one or more missing values.")
else
  . . .
end if
```

```
if(all(ismissing(lat)) then
  print("Error: all your lat values are missing!")
  exit
end if
```

```
num_msg   = num(ismissing(temp))
num_valid = num(.not.ismissing(temp))
print("temp contains " + num_msg + " missing values.")
print("temp contains " + num_valid + " valid values. ")
```

Tip

```
if(.not.isatt(x,"_FillValue")) then
  x@_FillValue = default_fillvalue(typeof(x))
end if
```


Do NOT test for missing values using actual values!

Tip

No:

```
if(any(x.eq.-999)) then
  print("Error: x cannot contain missing values.")
end if
```

Yes:

```
If(any(ismissing(x)) then
  print("Error: x cannot contain missing values.")
end if
```

No:

```
if(all(x.eq.x@_FillValue)) then
  print("Warning: all your x values are missing!")
end if
```

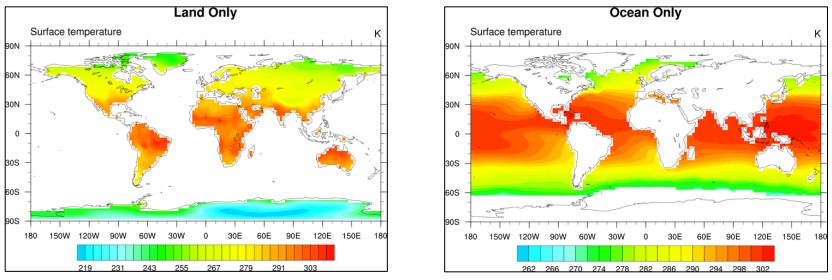
Yes!

```
if(all(ismissing(x))) then
  print("Warning: all your x values are missing!")
end if
```


mask

- Mask data array based on some condition
- First two arguments must be same dimensionality

```
in  = addfile("atmos.nc","r")
ts  = in->TS(0,:,:)
oro = in->ORO(0,:,:)      ; ocean=0, land=1, sea_ice=2
tsl = mask(ts,oro,1)      ; mask ocean
tso = mask(ts,oro,0)      ; mask land
```



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

where

Performs array assignments based upon a conditional array

```
;---Add 256 to all values in q < 0.0
q = where(q .lt. 0, q+256, q)
```

```
x  = where (T.ge.0 .and. ismissing(Z), a+25, 1.8*b)
ts = where (oro.eq.1, ts, ts@_FillValue)
salinity = where (sst.lt.5 .and. ice.gt.icemax,   \
                  salinity*0.9,  salinity)
```

Tip

No

```
y = where(y.eq.0, y@_FillValue, 1./y)
```

Yes

```
y = 1.0/where(y.eq.0, y@_FillValue, y)
```


Demos

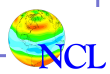
any, all, num, ismissing

mask, where

dim_xxx and dim_xxx_n

- Performs common operations on given dimensions
- dim_*xxx* operates across rightmost dimension ONLY
- dim_*xxx*_n allows you to specify dimension(s), uses much less memory, cleaner code!
- dim_*avg*_n (*stddev*, *sum*, *sort*, *median*, *rmsd*, …) (20+)
- Still okay to use original dim_*xxxx* functions if don't need to reorder!

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



dim_xxx and dim_xxx_n

Assume you have 3D array, z(time x lat x lon)

```
;---To average across longitude (zonal) no reordering required
zAvgZon  = dim_avg(z)      ; → zAvgZon(ntim,nlat)
```

This also works, but makes no difference in memory or timing

```
zAvgZon  = dim_avg_n(z,2)          ; → zAvgZon(ntim,nlat)
```

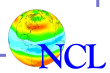
Works, but can be slow and memory intensive

```
;---To average across time, reordering required
zAvgTime = dim_avg(z(lat|:,lon|:,time|:))   ; → zAvgTime(nlat,nlon)
```

Faster, cleaner, and less memory intensive

```
zAvgTime = dim_avg_n(z,0)          ; → zAvgTime(nlat,nlon)
```

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



Demo

dim_avg, dim_avg_n

conform / conform_dims

- Conform smaller array to size of larger array
- Smaller array must be scalar or subset in size of larger array

```
;---Assume x(nlat,mlon), wgt(nlat)
wx = conform (x, wgt, 0)                ;  wx(nlat,mlon)
```

```
wx = conform_dims(dimsizes(x),wgt,0)   ; identical
```

Script

```
x = (/(/1,2,3/),(/4,5,6/)/)   ; x(2,3)
y = (/7,8,9/)                 ; y(3)
z = conform(x,y,1)

opt = True          Tip  ; print matrix
opt@title="x (2 x 3)"
write_matrix (x, "3i3", opt)
opt@title="z (2 x 3)"
write_matrix (z, "3i3", opt)
```

Output

```
x (2 x 3)

  1   2   3
  4   5   6

z (2 x 3)

  7   8   9
  7   8   9
```

ind

- Returns indexes where condition is True
- Operates on 1D array only

Script

```
;           0    1    2    3    4    5    6    7    8
year = (/1990,1986,2011,1776,1923,2018,1800,1953,1965/)
print(ind(year.ge.2000))
print(ind(year.ge.1900.and.year.le.1999))
print(ind(year.eq.0))
```

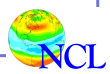
Output

```
(0)     2
(1)     5
```

```
(0)     0
(1)     1
(2)     4
(3)     7
(4)     8
```

```
(0)    -2147483647
```

(default missing value for an integer)



ind

Can use indexes in array operations

Script

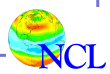
```
year = (/1990,1986,2011,1776,1923,2018,1800,1953,1965/)
xavg = (/ 50 , 61 , 32 , 40 , 55 , 42 , 65 , 18 ,  0 /)
ii = ind(year.ge.2000)
jj = ind(year.ge.1900.and.year.le.1999)
kk = ind(year.eq.0)
print(ii + " / " + year(ii) + " / " + xavg(ii))
print(jj + " / " + year(jj) + " / " + xavg(jj))
print(xavg(kk))                ; What will happen?
```

Output

```
(0) 2 / 2011 / 32
(1) 5 / 2018 / 42
```

```
(0) 0 / 1990 / 50
(1) 1 / 1986 / 61
(2) 4 / 1923 / 55
(3) 7 / 1953 / 18
(4) 8 / 1965 / 0
```

```
???
```

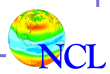
ind / ndtooned / onedtond

Can use on nD arrays, but need to convert to 1D

```
; 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))
```

Tip

```
x = where (q.gt.0.and.q.lt.5, sqrt(q), x)
x = where (q.gt.25, 72, x)
x = where (q.lt.-50, -x*3.14159, x)
```

Demos

ind

conform / conform_dims

Many date conversion functions

- cd_calendar / cd_inv_calendar
- cd_string / yyyymm_to_yyyyfrac

```
time  (/ 17522904, 17522928, 17522952 /)
time@units = "hours since 1-1-1 00:00:0.0"
date1 = cd_calendar(time, 0)
date2 = cd_calendar(time, -2)
print(date1)
print(date2)
```

date1

(0,0)	2000
(0,1)	1
(0,2)	1
(0,3)	0
(0,4)	0
(0,5)	0

(1,0)	2000
(1,1)	1
(1,2)	2
(1,3)	0
(1,4)	0
(1,5)	0

(2,0)	2000
(2,1)	1
(2,2)	3
(2,3)	0
(2,4)	0
(2,5)	0

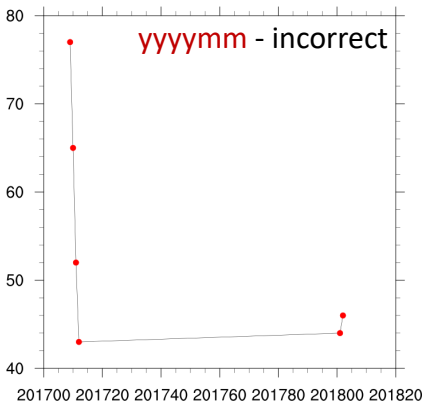
date2

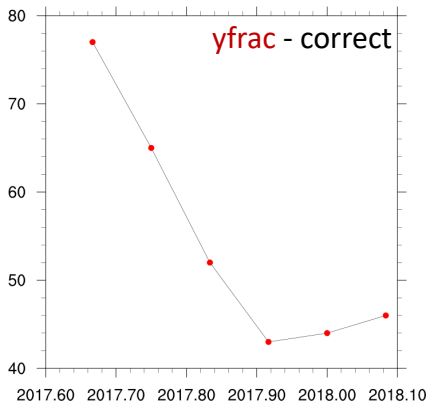
(0)	20000101
(1)	20000102
(2)	20000103

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

Date functions are crucial for plotting

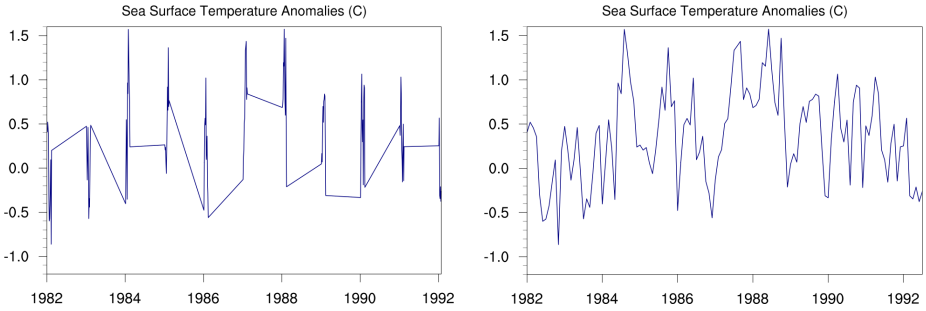
```
yyyymm = (/201709,201710,201711,201712,201801,201802/)
temp   = (/  77  ,  65  ,  52  ,  43  ,  44  ,  46  /)

wks = gsn_open_wks ("x11","xy")
plot  = gsn_csm_xy (wks,yyyymm,temp,False) ; Incorrect

yfrac = yyyymm_to_yyyyfrac(yyyymm,0.)
plot  = gsn_csm_xy (wks,yfrac,temp,False)  ; Correct
```





```
a       = addfile("sst8292a.nc","r")
sst     = a->SSTA     ; time (127) x lat x lon
yyyymm  = a->date     ; 198201, 198202, . . .
yfrac   = yyyymm_to_yyyyfrac(yyyymm,0) ; 1982, 1982.083

;---Bad plot
plot = gsn_csm_xy(wks, yyyymm, sst(:,{0},{0}), res)

;---Good plot
plot = gsn_csm_xy(wks, yfrac, sst(:,{0},{0}), res)
```

New Slide



String functions

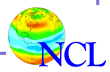
- Splitting data from a CSV file
- Parsing complicated strings
- Formatting numeric values to write to ASCII file
- Finding matching strings

str_get_cols
str_get_field
str_join
str_match
str_strip
str_split

str_split_csv
sprinti
sprintf
strlen
unique_string

Useful for reading ASCII or CSV files.

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



Parsing strings, converting to float/int

Script

```
x = (/"u_052134_C "," q_1234_C ", "temp_72.55_C"/)
```

```
f1  = str_get_field(x, 1, "_")
f2  = str_get_field(x, 2, "_")
print("f1='" + f1+ "'")
print("f2='" + f2+ "'")
```

```
col = str_get_cols(x, 2, 4)
print("col='" + col + "'")
```

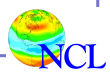
```
N =   toint(str_get_cols(x(0), 3, 7))
T = tofloat(str_get_cols(x(2), 5,9))
print("N = " + N)
print("T = " + T)
```

Output

```
(0) f1='u'
(1) f1=' q'
(2) f1='temp'
(0) f2='052134'
(1) f2='1234'
(2) f2='72.55'
```

```
(0) col='052'
(1) col='_12'
(2) col='mp_'
```

```
(0) N = 52134
(0) T = 72.55
```

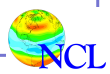



system / systemfunc

- Execute UNIX commands from NCL (bash shell)
- Useful for returning a list of files, removing a file, etc

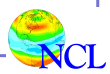
```
system ("convert foo.eps foo.png ; rm foo.eps")    ; multiple cmds
system ("ncrcat  -v T,Q foo*.nc FOO.nc")
system ("rm -f " + file_name)
```

```
files = systemfunc("ls wrfout*")                  ; list of files
files = systemfunc("cd /some/directory ; ls foo*nc")
UTC   = systemfunc("date")                        ; UNIX date
date  = systemfunc("date '+%a  %m%d%y %H%M' ")    ; single quote
City  = systemfunc("cut -c100-108 " + fname)
```

NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Regridding functions
- Creating your own functions and procedures
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices



Handling metadata

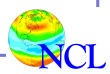
- Computations can cause loss of metadata

```
y = x          ; x is exact copy of y
T = T + 273    ; T retains its metadata
z = 5 * x      ; z will have no metadata
```

- Built-in functions also cause metadata loss

```
Tavg = dim_avg_n(T,0)
s    = sqrt(u^2 + v^2)
```

- Some exceptions: `vinth2p`



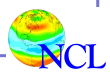
Ways to retain metadata (1 of 3)

- Use special "wrapper" functions

```
dim_avg_n_Wrap
dim_variance_n_Wrap
g2gsh_Wrap
int2p_Wrap
smth9_Wrap
wgt_runave_Wrap
. . .
```

- Takes same arguments as the non-Wrap versions

```
f = addfile("dummy.nc", "r")
x = f->X
xZon = dim_avg_n(x, 3)       ; xZon will NOT have metadata
xZon = dim_avg_n_Wrap(x, 3)  ; xZon will have metadata
```

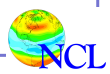



Ways to retain metadata (2 of 3)

Use `copy_XXXX` functions

```
copy_VarMeta    (attributes + coordinate arrays)
copy_VarAtts    (attributes only)
copy_VarCoords  (coordinate arrays only)
```

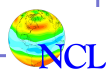
```
f = addfile("dummy.nc", "r")
x = f->X                    ; x(time,lev,lat,lon)

;---calculation
xZon = dim_avg_n(x, 2)      ; xZon

;---copy meta data
copy_VarMeta(x, xZon)       ; xZon(time,lat)
```

Ways to retain metadata (3 of 3)

- Use variable-to-variable copy
- Make sure to use variable with correct metadata!
- Might need to correct some metadata (long_name, units)

```
f = addfile("dummy.nc", "r")
x = f->X                    ; x(time,lev,lat,lon)

;---variable-to-variable copy & dimension reduction
xZon = x(:,:,:,0)       ; xZon(time,lev,lat)

;---calculation
xZon = dim_avg_n (x, 0)
xZon@op = "Zonal Avg: " + x@long_name
```

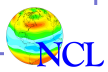



Demo

dim_avg_n_Wrap, metadata

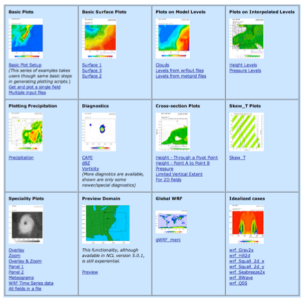
NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Regridding functions
- Creating your own functions and procedures
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices



WRF functions (`wrf_xxxx`)

- Weather and Research Forecast Model
- Computational and plotting functions developed by NCAR/MMM
- **`wrf_user_getvar`** gets diagnostics
- MMM has their own NCL/WRF web page

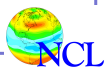


http://www.mmm.ucar.edu/wrf/OnLineTutorial/Graphics/NCL/

- Python version available! **wrf-python**

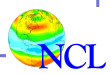
http://wrf-python.readthedocs.io/en/latest/

http: //www.ncl.ucar.edu/Applications/wrf.shtml
http://www.ncl.ucar.edu/Applications/wrfdebug.shtml



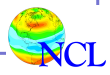
NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Regridding functions
- Creating your own functions and procedures
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices



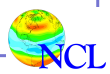
Regridding overview

- Many regridding / interpolation functions available
- Spatial (lat,lon) and vertical (pressure,height)
- In general, ESMF is best spatial regridding suite to use
  http://www.ncl.ucar.edu/Applications/ESMF.shtml
- Must know your data to know which is best
  - rectilinear
  - curvilinear
  - unstructured (mesh)
  - random points
- Documentation for all regridding and interpolation functions:
  http://www.ncl.ucar.edu/Document/Functions/regrid.shtml
  http://www.ncl.ucar.edu/Document/Functions/interp.shtml



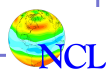
Common (spatial) grids and meshes

- Rectilinear – data with coordinate arrays
  - 1x2, 2x3, gaussian, Finite Volume (FV), Global Reanalysis
  - x(…,lat,lon) → lat(lat) and lon(lon)
- Curvilinear – data with 2D lat/lon arrays
  - WRF, POP, GODAS, RegCM, NARR, Satellite
  - x(…,nlat,mlon) → lat2d(nlat,mlon) and lon2d(nlat,mlon)
- Unstructured (mesh or random)
  - Spectral Element (SE), Finite Element (FE), MPAS, ICON
  - x(…,ncells) → lat(ncells) and lon(ncells)

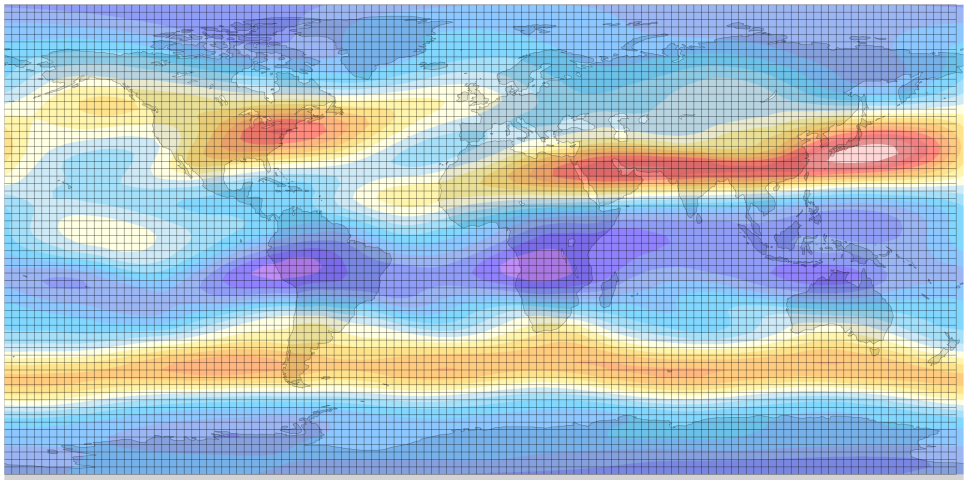


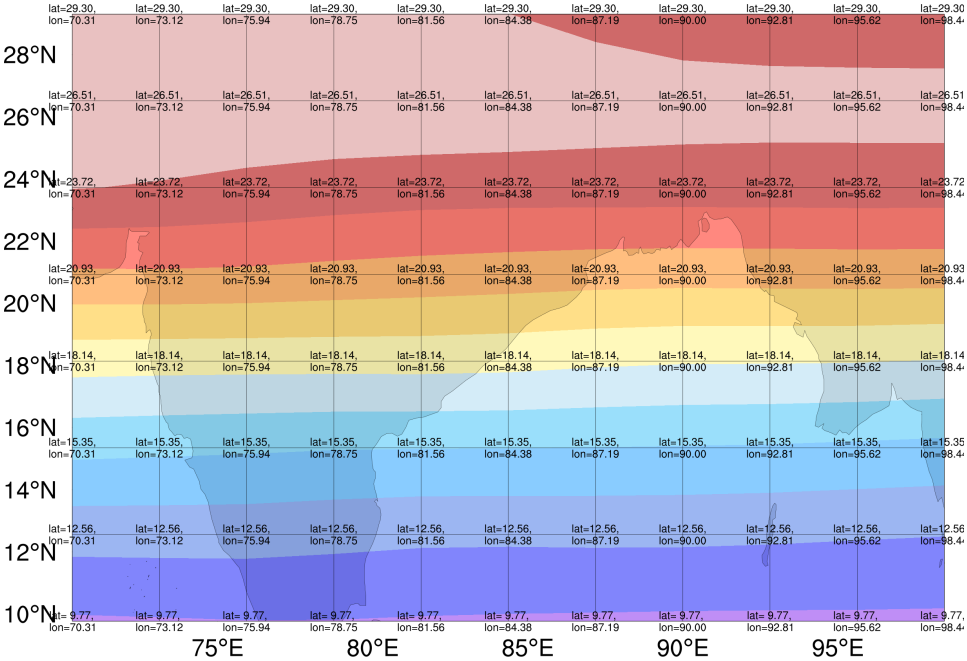
Why different grids?

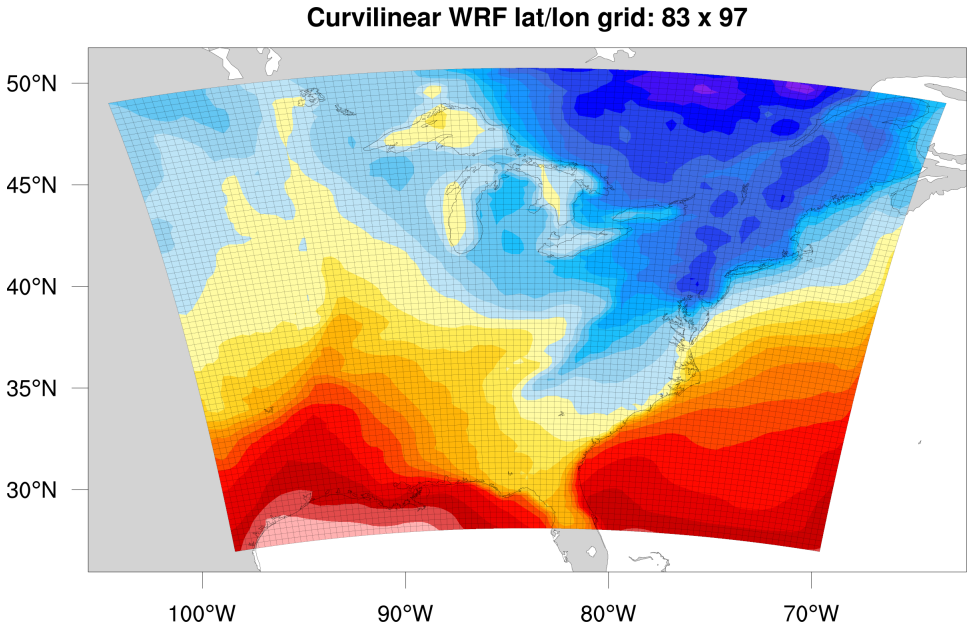
- Advances in computer architecture
- Computational efficiency
- Addresses pole singularities
- Better representation for physics and/or dynamical core

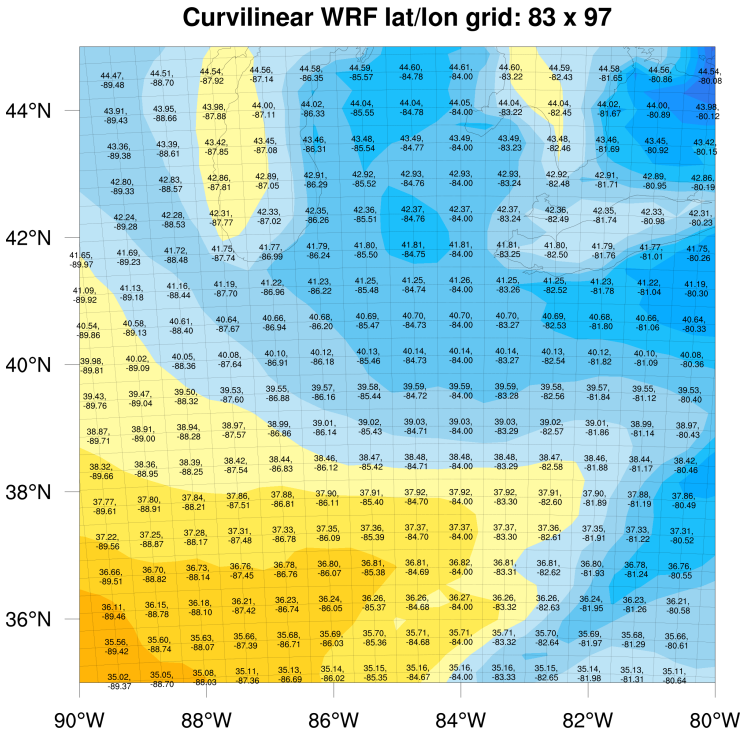


Rectilinear grid (64 x 128)

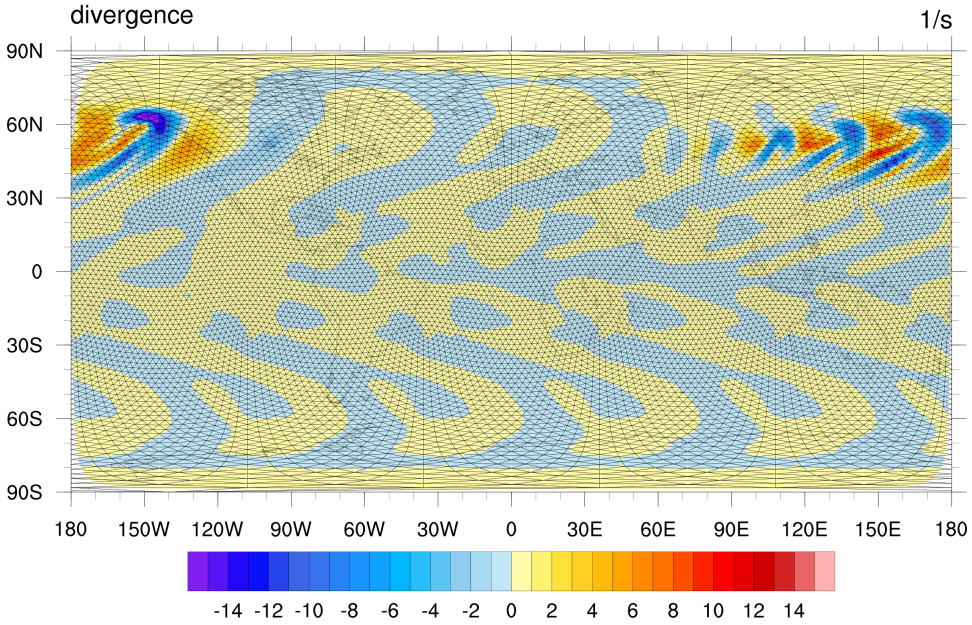




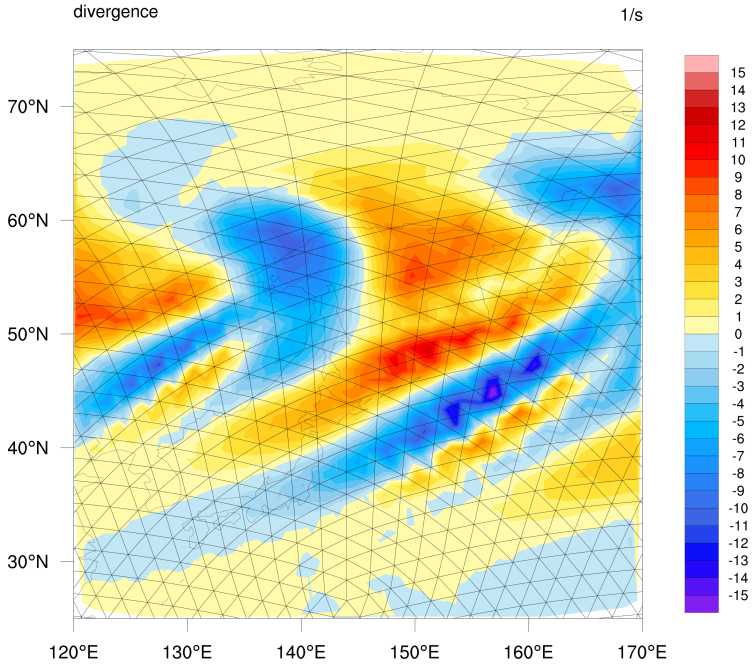

Curvilinear WRF lat/lon grid: 83 x 97
50°N
45°N
40°N
35°N
30°N
100°W
90°W
80°W
70°W

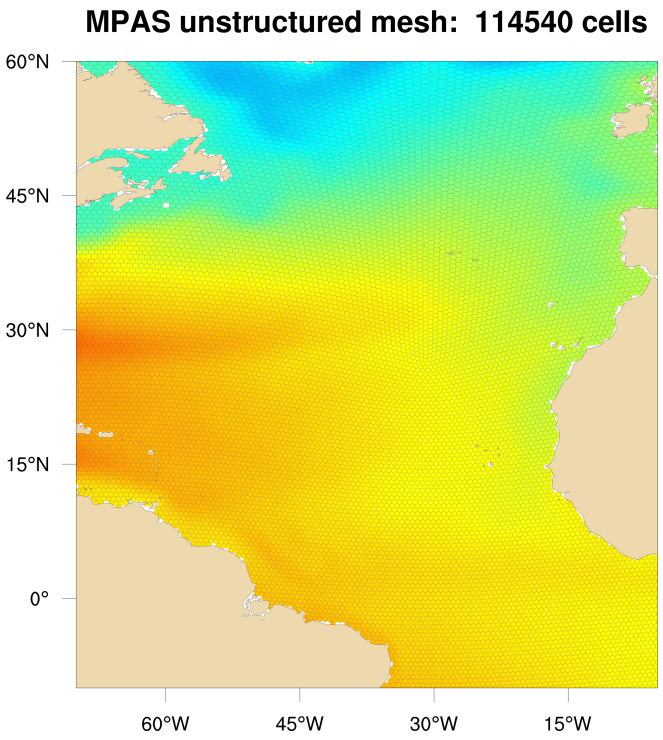

Curvilinear WRF lat/lon grid: 83 x 97
44°N
42°N
40°N
38°N
36°N
90°W
88°W
86°W
84°W
82°W
80°W

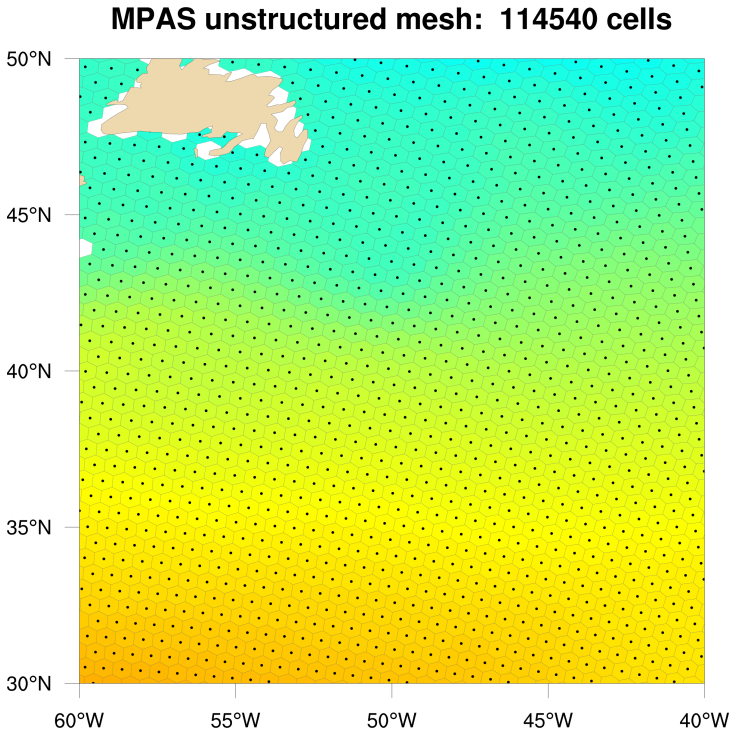
ICON model: 20480 triangles



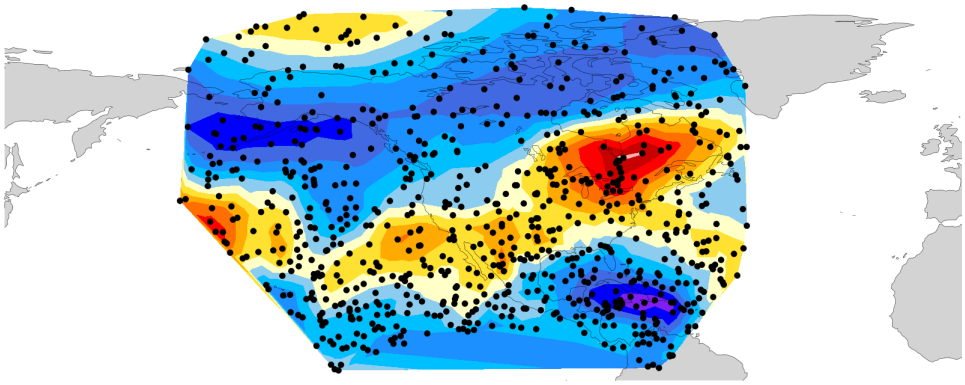
ICON model: 20480 triangles




MPAS unstructured mesh: 114540 cells
60°N
45°N
30°N
15°N
0°
60°W
45°W
30°W
15°W

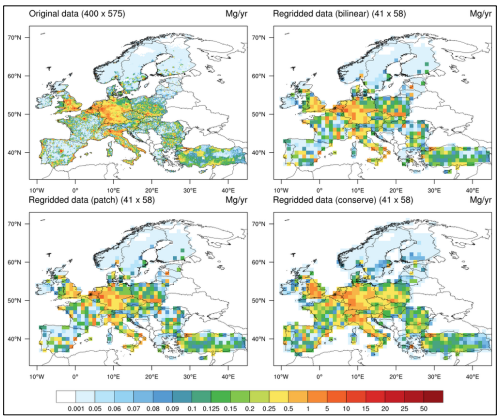

MPAS unstructured mesh: 114540 cells
50°N
45°N
40°N
35°N
30°N
60°W
55°W
50°W
45°W
40°W

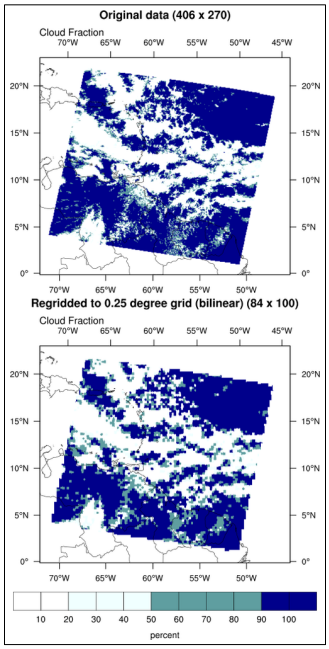
Random points



ESMF Regridding

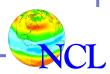
http://www.earthsystemmodeling.org/





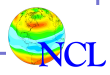
ESMF regridding comments (1 of 3)

- Integrated in conjunction with NOAA / CIRES
- Works with rectilinear, curvilinear, unstructured grids and meshes
- **CREATES A WEIGHTS FILE (NetCDF file)**
- Multiple interpolation methods available
  - Bilinear
  - Conservative
  - Patch
  - Nearest neighbor
- Can handle masked points
- Better treatment for values at poles
- Works on global/regional grids



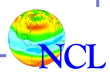
ESMF regridding comments (2 of 3)

- "Source grid" – the grid/mesh you are starting with
- "Destination grid" – the grid/mesh you want to regrid to
- For both source / destination lat/lon, must know:
  - Global or regional?
  - Rectilinear, curvilinear, unstructured, random?
  - Have missing values?
  - Is it a smooth field (temperature) or not (precipitation)?



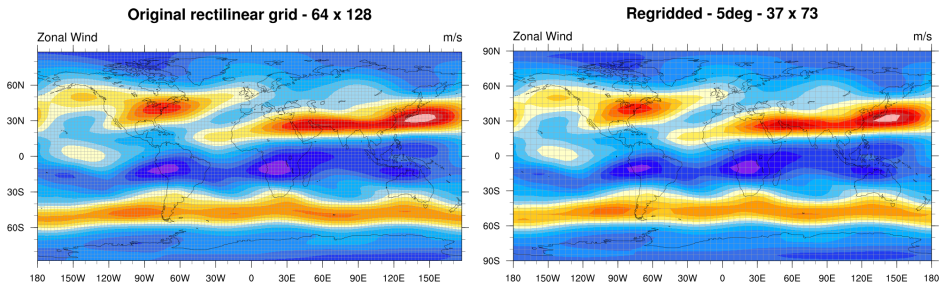
ESMF regridding comments (3 of 3)

- Must create a weights file for every grid / mesh that has different lat/lon values or different location of missing values
- Once you have weights file, you can use it to regrid; **MUCH FASTER!**
- Be careful with regridding U/V quantities
- ESMF examples page is extensive
  http://www.ncl.ucar.edu/Applications/ESMF.shtml



Regridding rectilinear variable to 5 degree

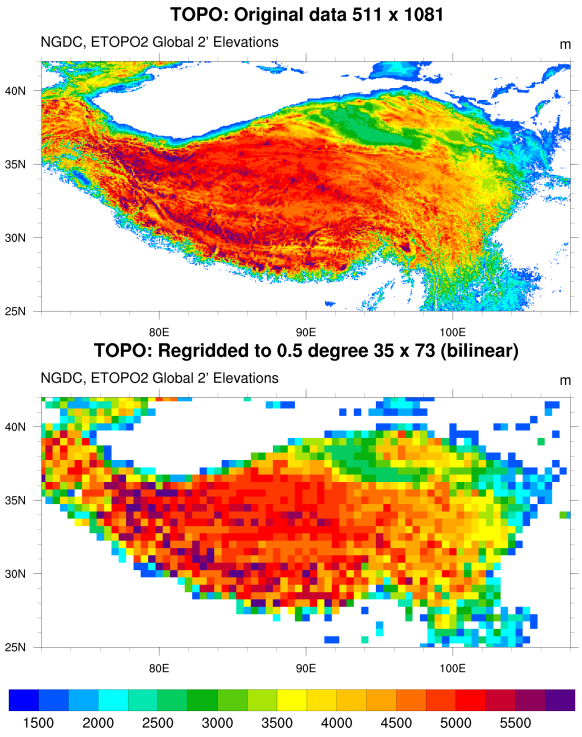
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/esmf/ESMF_regridding.ncl"

f = addfile("uv300.nc","r")     ; Open file
u = f->U(0,:,:)                 ; Read first time step

opt              = True         ; Set regridding options
opt@DstGridType  = "5deg"       ; Set destination grid
opt@WgtFileName  = "rect_to_5deg.nc"
u_regrid = ESMF_regrid(u,opt)   ; Regrid u (and create
                                ; weights file)
```

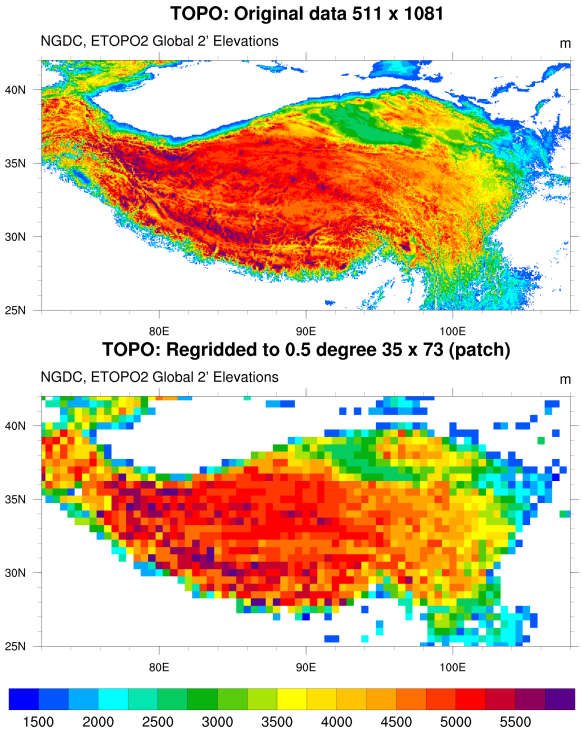


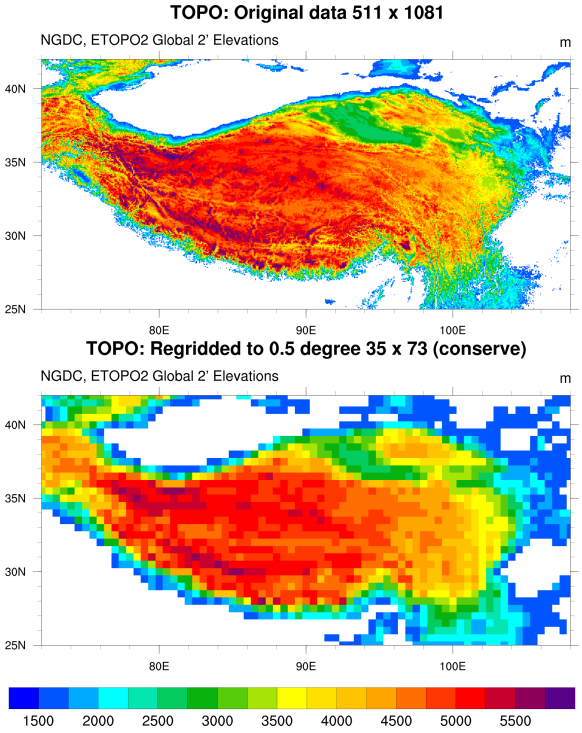
ESMF Regridding – using weights file

- Slow part is creating weights file
- Use **ESMF_regrid_with_weights** once you have file

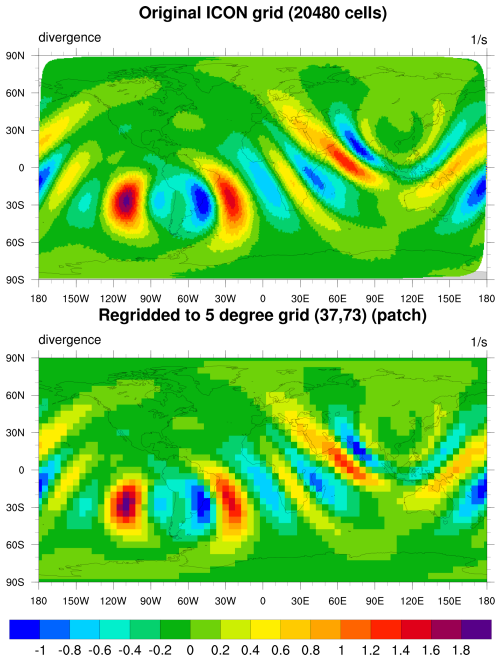
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/esmf/ESMF_regridding.ncl"

f = addfile("uv300.nc","r")     ; Open file
u = f->U(0,:,:)                 ; Read first time step

wgts_file = "rect_to_5deg.nc"
u_regrid = ESMF_regrid_with_weights(u,wgts_file,False)
```


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

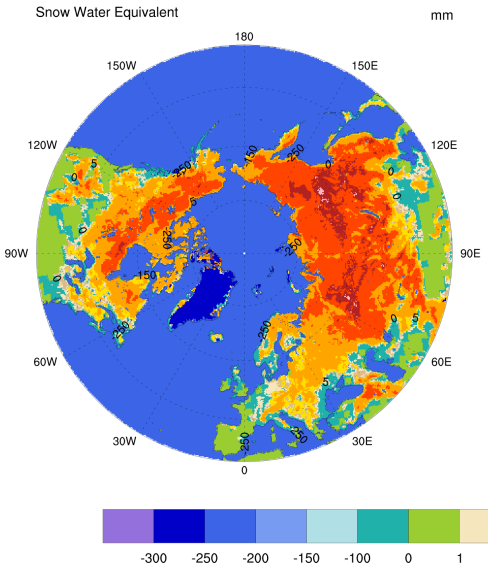

TOPO: Original data 511 x 1081
NGDC, ETOPO2 Global 2' Elevations
m
40N
35N
30N
25N
80E
90E
100E
TOPO: Regridded to 0.5 degree 35 x 73 (conserve)
NGDC, ETOPO2 Global 2' Elevations
m
40N
35N
30N
25N
80E
90E
100E
1500
2000
2500
3000
3500
4000
4500
5000
5500

ICON

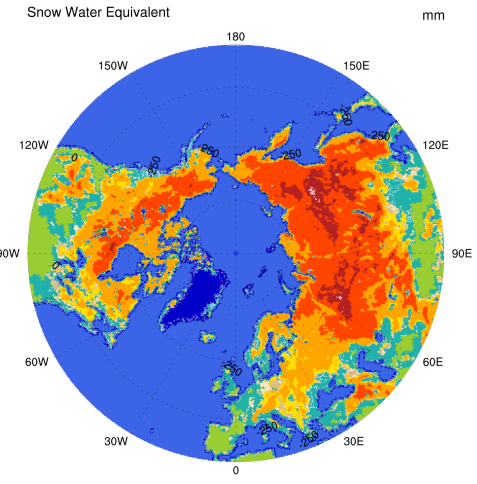


EASE

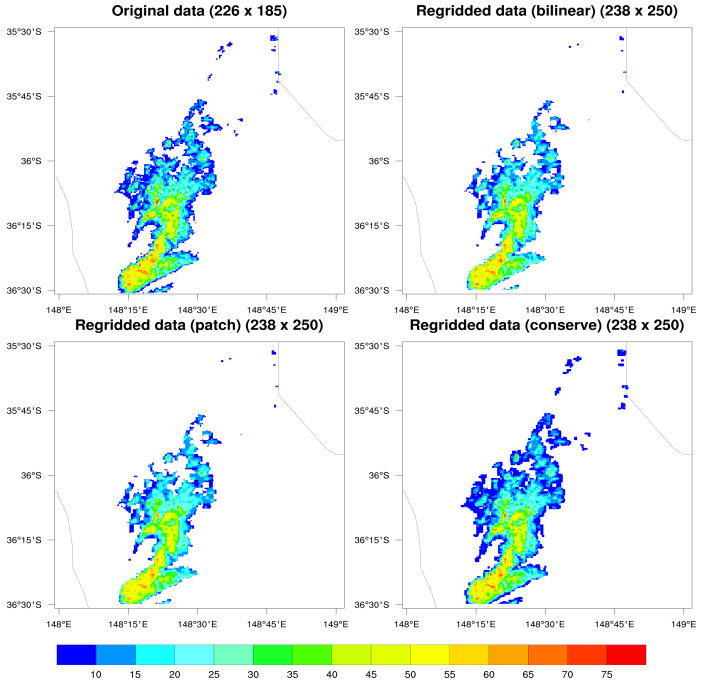
Original EASE grid (721,721)



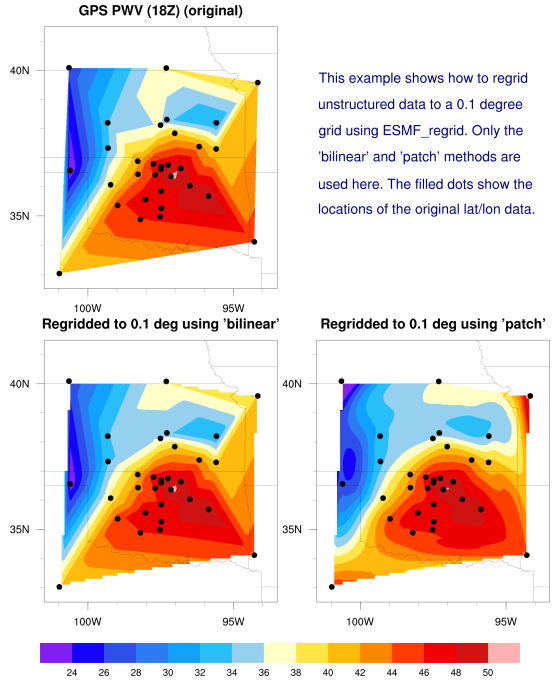
Regridded to 0.25 degree grid (359 x 1439)



Swath to WRF Grid: Australia Snow

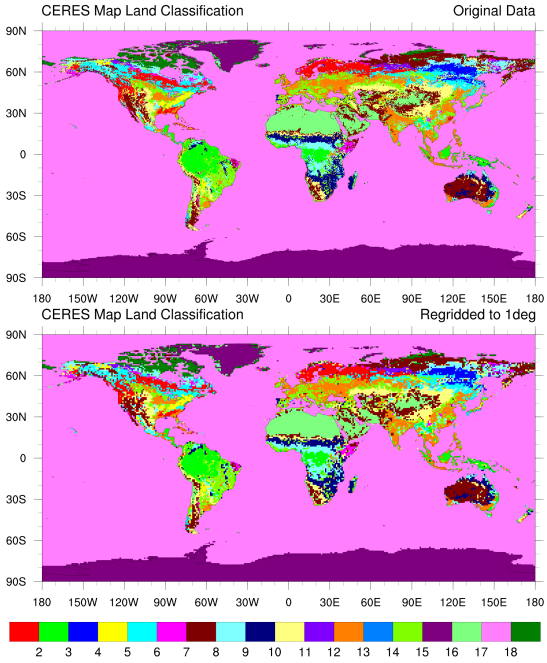


Random to Grid



This example shows how to regrid unstructured data to a 0.1 degree grid using ESMF_regrid. Only the 'bilinear' and 'patch' methods are used here. The filled dots show the locations of the original lat/lon data.

Categorical



Demo

ESMF regridding

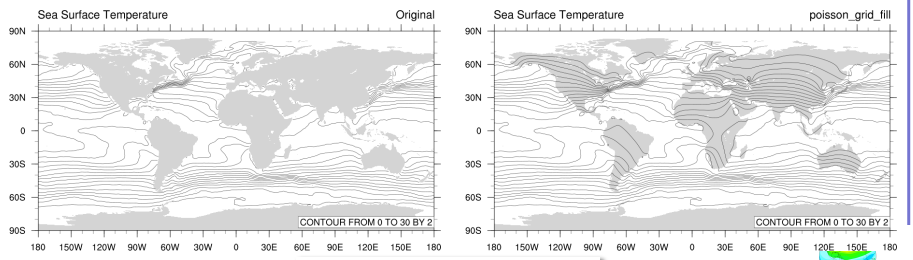
poisson_grid_fill

Tip

Replaces all _FillValue grid points

- Poisson's equation solved via relaxation
- Values at non-missing locations are used as boundary conditions
- 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)
```



NCL

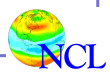
NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Regridding functions
- Creating your own functions and procedures
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices

NCL

Creating your own functions and procedures

- Crucial for repetitive tasks
- Customize existing ones for your needs
- Give them to other people
- Contribute back to NCL team!



NCL scripts

- Use "load" command, similar to Python's "import":

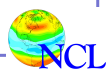
```
load "./myscript.ncl"
load "/glade/p/haley/myscript.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/crop.ncl"
```

- Examine contents of any script distributed with NCL

```
less $NCARG_ROOT/lib/ncarg/nclscripts/csm/crop.ncl
more $NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl
```

- Use any UNIX editor to extract code and modify for your needs

```
vi $NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl
```

Example: modifying **printMinMax** procedure

[in $NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl]

Original procedure

```
undef("printMinMax")
procedure printMinMax (x:numeric,optSpace:logical)
begin
   vLongName = (/"long_name", "description", "standard_name"/)
   vUnits    = get_valid_units()
   long_name = ""
   units     = ""
   do n=0,dimsizes(vLongName)-1
      if (isatt(x,vLongName(n))) then
          long_name = x@$vLongName(n)$
      end if
   end do

  if(long_name.ne."") then
     do n=0,dimsizes(vUnits)-1
        if (isatt(x,vUnits(n))) then
            units = x@$vUnits(n)$
        end if
     end do
   end if

   if (optSpace) then
       print (" ")
   end if

   if(units.ne."") then
     print (long_name+ " ("+units+")" + " : min="+min(x)+\
                                        "   max="+max(x))
   else
     print (long_name+ " : min="+min(x)+"   max="+max(x))
   end if
end
```

Modified procedure – "myprint.ncl"

```
undef("printMinMax")
procedure printMinMax (x:numeric)
begin
   vLongName = (/"long_name", "description", "standard_name"/)
   vUnits    = get_valid_units()
   long_name = ""
   units     = ""
   do n=0,dimsizes(vLongName)-1
      if (isatt(x,vLongName(n))) then
          long_name = x@$vLongName(n)$
      end if
   end do

  if(long_name.ne."") then
     do n=0,dimsizes(vUnits)-1
        if (isatt(x,vUnits(n))) then
            units = x@$vUnits(n)$
        end if
     end do
   end if

;;   if (optSpace) then
;;       print (" ")
;;   end if

  if(units.ne."") then
     print (long_name+ " ("+units+")" + " : min="+min(x)+\
                                        "   max="+max(x))
   else
     print (long_name+ " : min="+min(x)+"   max="+max(x))
   end if
end
```

```
load "./myprint.ncl"

f = addfile("uv300.nc","r")
u = f->U(0,:,:)
printMinMax(u)           ; No second argument!
```

General structure of NCL function/procedure

```
undef ("function_name")          ; optional, HIGHLY recommended
function function_name (declaration_list)
local local_variable_list        ; optional
begin
  …statements…
  return(…)
end
```

```
undef ("procedure_name")         ; optional, HIGHLY recommended
procedure procedure_name (declaration_list)
local local_variable_list        ; optional
begin
  …statements…
end
```

Example: `mult` function – one script

Script

```
undef ("mult")
function mult(x1,x2,x3,x4)
local sx1, foo
begin
  sx1           = sin(0.01745329*x1)
  foo           = sx1*x2*x3*sqrt(x4)
  foo@long_name = "result"
  foo@units     = "Unknown"
  return (foo)
end

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

Output

```
Variable: x
Type: float
Number of Dimensions: 1
Number Of Attributes: 2
  long_name : result
  units : Unknown
(0)    2233.906
```

Example: `mult` function – two scripts

myLib.ncl

```
undef ("mult")
function mult(x1,x2,x3,x4)
local sx1, foo
begin
  sx1           = sin(0.01745329*x1)
  foo            = sx1*x2*x3*sqrt(x4)
  foo@long_name = "result"
  foo@units     = "Unknown"
  return (foo)
end
```

myscript.ncl

```
load "myLIB.ncl"

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

NOTE: myLib.ncl can contain multiple scripts

Create your own printMinMax, printVarSummary with shorter names

my_ncl_lib.ncl

```
undef("printm")
procedure printm (x:numeric)
begin
  printMinMax(x,0)
end

undef("printv")
procedure printv (x)
begin
  printVarSummary(x)
end
```

```
load "./my_ncl_lib.ncl"

f = addfile("uv300.nc","r")
u = f->U(0,:,:)
printv(u)
printm(u)
```

Function / procedure prototyping

- Arguments passed by reference
- Can specify type and/or size, or neither

```
procedure ex(year[12]:integer, title:string)
```

```
function xy_interp(x1[*][*][*]:numeric, x2:numeric)
```

```
function set_color(cmap[*][3]:float, res[1]:logical)
```

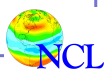
```
procedure whatever(a, b, c)
```

Sneaking in extra arguments

- Use attributes, which are just like variables
- Add extra argument to attach these attributes

```
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,"add")) then
    d = data + opt@add
 end if
 if (opt .and. isatt(opt,"wgts")) then
  . . .
 end if
. . .
end
```

```
optArg       = True
optArg@scale = 0.01
optArg@add   = 1000
optArg@wgts  = (/1,2,1/)
optArg@name  = "sample"

ex(x2D, "Example", optArg)
```

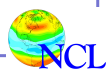



Demo

Creating your own functions

NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Creating your own functions and procedures
- Regridding functions
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices



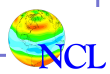
Calling Fortran routine from NCL

- Easier to use F77 code, but works with F90
- Isolate definition of input variables and wrap with special comment statements:

```
C NCLFORTSTART
. . .
C NCLEND
```

- Use a tool called **WRAPIT** to create a ***xxxx*.so** file
- Load **`xxxx.so`** file in NCL script with "external" statement
- Call Fortran function with special **`::`** syntax
- Must preallocate arrays! (using NCL's **`new`** statement)

http://www.ncl.ucar.edu/Document/Tools/WRAPIT.shtml



Sample Fortran 77 code

```
C NCLFORTSTART
      subroutine compute_tk(tk,pressure,theta,nx,ny,nz)
      implicit none
      integer nx, ny, nz
      real    tk(nx, ny, nz)
      real    pressure(nx, ny, nz), theta(nx, ny, nz)
C NCLEND
      integer i, j, k
      real    pi

      do k=1,nz
        do j=1,ny
          do i=1,nx
            pi = (pressure(i,j,k)/1000.)**(287./1004.)
            tk(i,j,k) = pi*theta(i,j,k)
          end do
        end do
      end do
      end
```

Create "myTK.so" file and use in script

From UNIX command line:

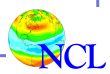
```
WRAPIT myTK.f
```

This will create a "**myTK.so**" file

```
external myTK "./myTK.so"

p = a->pressure
t = a->T
t = t + 300

;---Must preallocate space for output arrays
dim = dimsizes(t)
tk  = new( dimsizes(t), typeof(t) )

;---Remember, Fortran/NCL arrays are ordered differently
myTK :: compute_tk (tk,p,t,dim(2),dim(1),dim(0))
```

Calling Fortran 90 from NCL

- Can use simple Fortran 90 code
- Your F90 program cannot contain any of the following features:
  - pointers or structures as arguments
  - missing or optional arguments
  - keyword arguments
  - recursive procedures
- The input arguments must be reproduced in a separate F77-like "stub" file
- "WRAPIT" is a modifiable script



Sample Fortran 90 code with "stub file"

myTK.f90

```
subroutine compute_tk (tk, pres, theta, nx, ny, nz)
  implicit none
  integer :: nx,ny,nz
  real, dimension (nx,ny,nz) :: tk, pres, theta, pi

  pi = (pres/1000.)**(287./1004.)
  tk = pi * theta

end subroutine compute_tk
```

myTK.stub "stub file" should look like F77 code

```
C NCLFORTSTART
  subroutine compute_tk (tk, pres, theta, nx, ny, nz)
  implicit none
  integer nx,ny,nz
  real    tk(nx,ny,nz)
  real    pres(nx,ny,nz), theta(nx,ny,nz)
C NCLEND
```

Using WRAPIT with a stub file

From UNIX command line:

```
WRAPIT myTK.f mkTK.stub
```

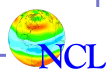
This will create a "**myTK.so**" file

NCL script will be exactly the same

```
external myTK "./myTK.so"

p = a->pressure
t = a->T
t = t + 300

;---Must preallocate space for output arrays
dim = dimsizes(t)
tk  = new( dimsizes(t), typeof(t) )

;---Remember, Fortran/NCL arrays are ordered differently
myTK :: compute_tk (tk,p,t,dim(2),dim(1),dim(0))
```

NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Creating your own functions and procedures
- Regridding functions
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices



Command Line Arguments (CLAs)

- Allows you to set variables *when you run NCL*
- Assume you have 3-line "cla.ncl" script:

```
print("title = '" + t + "'")
y = x * 0.01
print("y = "+y)
```

- Running "ncl cla.ncl" will give you errors:

```
fatal:Variable (t) is undefined
fatal:Variable (x) is undefined
fatal:Variable (y) is undefined
```

- Instead, run NCL with:

```
ncl x=5000 'title="CLAs"' cla.ncl
```

```
(0) title = 'CLAs'
(0) y = 50
```

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

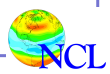
CLAs: more detailed example

```
if (.not. isvar("fNam") .and.(.not. isvar("var") ) then
  print("fNam not specified: exiting")
  exit
end if

f = addfile (fNam, "r") ; open file
x =f->$var$             ; read variable

if (.not. isvar("year")) then
  year = 1900               ; provide default
end if
if (.not. isvar("lev")) then
  lev = 500                 ; provide default
end if
```

```
ncl year=1930 'lev=(/250, 750/)' 'var="T"' 'fNam="foo.nc"' sample.ncl
```

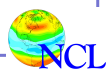
NCL Data Analysis Outline

- Goals
- Functions and procedures overview
- Essential utility functions
- Special _Wrap functions / metadata
- WRF functions
- Creating your own functions and procedures
- Regridding functions
- Calling Fortran code from NCL
- Command line arguments
- Programming tips / Clean coding / Best practices



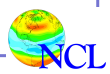
Things to think of when coding

- Am I writing this for me or someone else?
  - Reusability is important, even for yourself
- How generic does it need to be?
- Clean coding, even for small scripts
  - Find your coding style and get into the habit
- RTM
  - Make sure you are using the best function for the job
- Manage memory carefully
- Write the test before the code ☺



Clean coding practices

- Use indentation (and do it consistently)
- Comment code well
- Start with existing scripts
- Reuse code!
- Use meaningful variables names
- Use functions for repetitive tasks
- Use "\" character to break long lines
- Don't hard-code values
- Test as you go
- Use editor enhancements



Watch those do loops! (1 of 3)

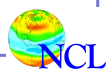
Code that doesn't change inside do loop should be moved outside do loop

No

```
colors = (/"blue","red","tan",…/)
do nv=0,nvars-1
  vname = "var" + nv
  x = a->time
  y = x->$vname$
  res = True
  res@gsnMaximize = True
  res@xyLineColor = colors(nv)
  plot = gsn_xy(wks,x,y,res)
end do
```

Yes

```
x = a->time
res = True
res@gsnMaximize = True
colors = (/"blue","red","tan",…/)
do nv=0,nvars-1
  vname = "var" + nv
  y = x->$vname$
  res@xyLineColor = colors(nv)
  plot = gsn_xy(wks,x,y,res)
end do
```

Watch those do loops! (2 of 3)

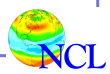
Take advantage of array arithmetic – cleaner and MUCH faster!

No

```
; nx = ny = nz = 100
tk = new((/nx,ny,nz/),float)
do k=0,nz-1
  do j=0,ny-1
    do i=0,nx-1
      pi = (p(i,j,k)*0.001)^(0.285)
      tk(i,j,k) = pi*theta(i,j,k)
    end do
  end do
end do
```

Yes – **80x** faster

```
; nx = ny = nz = 100
pi = (p(i,j,k)*0.001)^(0.285)
tk = pi*theta
```

Watch those do loops! (3 of 3)

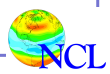
If can spare the memory, use arrays, especially with strings.

Okay, but…

```
do nv=0,nvars-1
  vname = "var" + nv
  y = x->$vname$
  . . .
end do
```

Maybe better

```
vname = "var" + ispan(0,nvars-1,1)
do nv=0,nvars-1
  y = x->$vname(nv)$
  . . .
end do
```

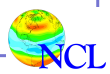



Memory management

- Functions help control memory issues
- Use **delete** to remove large, unneeded variables

```
delete(x)
delete([/x,data,opt/])
```

- Use (/.../) around an array to strip metadata
- Avoid reordering data if possible
  - Do it once and save to new variable
  - Make use of ***xxxx*_n** functions
- Don't use **new** unless you have to
- Strings are expensive!
- Do loops can be useful for large data arrays; do operations across **subset** of data



Debugging tips

- LOOK AT YOUR DATA!
  - Use **print** / **printVarSummary** / **printMinMax** liberally
- If script takes a long time and/or big arrays
  - Write intermediate data to NetCDF
  - Run script with smaller subset of data
  - Use "exit" to skip execution of rest of script
- Use "-x" command line option to echo every line
  - Must comment out the main "begin" and "end"
- Editor enhancements help!
- Read errors and warnings carefully

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