Class #5 Wednesday 2 February 2011

- 9:45-10:45, room rebooked also
- What did we discuss last time?
- Today (1.3.3 NCl Part 3)
- Continuing with NCL tutorials

1. Finish Data Processing
2. Wrapper examples

Next time NCL graphics and NCO operators

- Loose Ends
  - Functions versus Procedures

Go through these to learn NCL

<http://www.ncl.ucar.edu/Training/Workshops/lectures.shtml>

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 (“msrcp –n ‘mss:/SHEA/REANALYSIS/*’  /ptmp/shea”)
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)
```

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 loading functions more than once** (undef)

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

```
load "/fs/cgd/home0/shea/ncld/mult.ncl"
begin
  x = mult(4.7, 34, 567, 2)
end
```

3

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

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

2

User-Built Functions and Procedures(2 of 4)

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

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

```
undef ("procedure_name")                ; optional
procedure procedure_name (declaration_list)
local local_identifier_list             ; optional
begin
   statements
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"
begin
   f = addfile("dummy.nc", "r")
   x = f->X
   xZon = dim_avg_Wrap(x)   ; xZon will have meta data
end
```

Ex: compute PSI/CHI add meta data

```
load “$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl“
begin
    f       = addfile (“UV300.nc”, “r”)
    u       = f->U
    v       = f->V
; calculate psi and chi
    psi   = ilapsG (uv2vrG(u,v), 0)
    chi   = ilapsG (uv2dvG(u,v), 0)
; copy coordinate variables using function in contributed.ncl
   copy_VarCoords(u, psi)
   copy_VarCoords(u, chi)
; create unique attributes
    psi@long_name = "PSI"
    chi@long_name = "CHI"
    psi@units           = "m2/s"
    chi@units           = “m2/s“
; scale values for plotting
    scale = 1.e06                       ; incorporate this into units label
    psi     = psi/scale
    chi     = chi/scale
    ….. plot …..
end
```

regrid: bilinear interpolation linint2

- Cartesian, global or limited area grids
- Must be grids that can be representd by one-dim coords
- 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 )
```

Example: Arbitrary Transect

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

begin
                                        ; open file and read in data
   diri = “/fs/scd/home1/shea/ncldata_input/”
   fili  = “01-50.nc”
   f     = addfile(diri+fili ,  "r")
   T     = f->T                        ; T(time,lev,lat,lon)
                                       ; create arrays of lat and lon points
   lonx = (/ 295, 291.05, 287.4 , 284,   281, 274.5, 268, 265 /)
   laty  = (/ -30,     -25.2,   -20.3, -15, -10.3,     0.0,  9.9,   15 /)
                                       ; must have a “regular” grid

                                       ; interpolate data to given lat/lon points
   transect = linint2_points_Wrap (T&lon, T&lat, T, True,lonx,laty, 0)
                                       ; transect(time,lev, 8)
end

ncl < PR_ex04.ncl
```

Create/Use a Fortran Shared Object (1 of 2)

Step 1

- bracket f77 subroutine + argument declarations with **interface deliminators**
- only **codes actually called** from NCL need special interface deliminators
- no Fortran argument can be named data (bug)

```
C   NCLFORTSTART
        subroutine foo ( xin,xout, mlon, nlat, text)
        integer mlon, nlat
        real xin(mlon,nlat), xout(mlon,nlat)
        character*(*) text
C   NCLEND
      rest of fortran code; may include many subroutines
      or other declarations
```

Create/Use a Fortran Shared Object (2 of 2)

Step 2: create shared object using **WRAPIT** utility

- **WRAPIT** foo.f
- **WRAPIT** –specified_f90_compiler foo.stub foo.f90
  - **WRAPIT** –pg foo.stub foo.f90

Step 3: add external statement to NCL script

- external SO_NAME   "path_name"
  - SO_NAME is arbitrary (capital by convention)
  - external   DEMO   "./foo.so"   (".so" by convention)

Step 4: invoking the shared object in the script

- SO_NAME::fortran_name(arguments)
- DEMO::foo(x,y,mlon,nlat,label)

what WRAPIT does

- **automatically creates NCL-fortran interface(s)**

- **uses wrapit77 to create C interface [f77 syntax]**
  - wrapit77 < foo.f   >! foo_W.c

- **only uses code enclosed between delimeters**
  - input can be code fragment(s) or full subroutine(s)

- **compiles and creates object modules**
  - nhlcc -c foo_W.c ➔ foo_W.o
  - nhlf90 -c foo.f ➔ foo.o

- **creates dynamic shared object [.so] using ld**
  - SGI: ld -64 -o foo.so -shared foo_W.o foo.o -fortran
  - SUN: /usr/ccs/bin/ld -o foo.so foo_W.o foo.o -G -lf90 -L /opt/SUNWspro/lib -l sunperf

- **removes extraneous intermediate files**

WRAPIT –h <return> will show options and examples

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
```

15

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
```

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
```

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
```

```
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
```

Fortran vs. NCL: string - character

NCL (C) ←→ Fortran: string/character interchange problematical

```
C  NCLFORTSTART
         subroutine csdemo (string_in, string_out)
             character*(*)    string_in      ! passed FROM ncl
             character*8      string_out    ! passed   TO    ncl
C NCLEND
           print *, string_in
           string_out  = "F-to-NCL"
           return
        end
```

```
external DEMO "./csdemo.so"
begin
    cstring = new (8, character)
    DEMO::csdemo("NCL-F" , cstring)

    stringc = chartostring( cstring )
    print ( stringc )
end
```

Can **NOT** pass arrays of strings or characters

Linking Commercial IMSL (NAG,…) routines

```
            STEP 1: rcurvWrap.f
C  NCLFORTSTART
       subroutine rcurvwrap (n, x, y, nd, b, s, st, n1)
       integer  n, nd, n1
       real       x(n),  y(n),  st(10), b(n1), s(n1)
C  NCLEND
       call rcurv(n,x,y,nd,b,s,st)          ! IMSL
       return
       end
```

STEP 2: rcurvWrap.so

```
WRAPIT –l mp  –L /usr/local/lib64/r4i4  –l imsl_mp  rcurvWrap.f
```

```
external  IMSL "./rcurvWrap.so"
begin
   x = (/ 0,0,1,1,2,2,4,4,5,5,6,6,7,7 /)
   y = (/508.1, 498.4, 568.2, 577.3, 651.7, 657.0, 755.3 \
         758.9, 787.6, 792.1. 841.4, 831.8, 854.7, 871.4 /)

   nobs = dimsizes(y)
   nd     = 2
   n1     = nd+1
   b       = new ( n1, typeof(y))
   s       = new ( n1, typeof(y))
   st      = new (10,  typeof(y))

   IMSL::rcurvwrap(nobs, x, y, nd, b, s, st, n1)
end
```

Accessing LAPACK (1 of 2)

- **double precision** LAPACK (BLAS) => **distributed with NCL**
  - explicitly link LAPACK lib via fortran interface: **WRAPIT**
  - eg: subroutine dgels solves [over/under]determined real linear systems

```
C NCLFORTSTART
      SUBROUTINE DGELSI( M, N, NRHS, A, B, LWORK, WORK )
      IMPLICIT   NONE
      INTEGER    M, N, NRHS, LWORK
      DOUBLE PRECISION   A( M, N ), B( M, NRHS), WORK(LWORK)
C NCLEND
C                                    declare local variables
      INTEGER     INFO
      CHARACTER*1 TRANS
      TRANS = "N”
      CALL DGELS(TRANS, M,N,NRHS,A,LDA,B,LDB,WORK,LWORK,INFO)
      RETURN
      END
```

```
WRAPIT –L  $NCARG_ROOT/lib  -l  lapack_ncl   dgels_interface.f
```

Accessing LAPACK (2 of 2)

```
external DGELS "./dgels_interface.so”
; NAG example: http://www.nag.com/lapack-ex/node45.html
; These are transposed from the fortran example
 A = (/ (/ -0.57, -1.93, 2.30, -1.93, 0.15, -0.02 /), \  ; (4,6)
        (/ -1.28,  1.08, 0.24,  0.64, 0.30,  1.03 /), \
        (/ -0.39, -0.31, 0.40, -0.66, 0.15, -1.43 /), \
        (/  0.25, -2.14,-0.35,  0.08,-2.13,  0.50 /)  /)*1d0     ; must be double
   dimA  = dimsizes(A)
   N     = dimA(0)        ; 4
   M     = dimA(1)        ; 6
   B     = (/-2.67 ,-0.55 ,3.34, -0.77, 0.48, 4.10/)*1d0       ; must be double
                             ; LAPACK wants 2D
   nrhs  = 1
   B2    = conform_dims ( (/nrhs,M/), B, 1 )    ; (1,6)
   B2(0,:) = B


   lwork  = 500                                            ; allocate space
   work   = new ( lwork, "double", "No_FillValue")         ; must be double


   DGELS::dgelsiw(M, N, nrhs, A , B2, lwork, work )
   print(B2(0,0:N-1))
```

Combining NCL and Fortran in C-shell

```
#!/usr/bin/csh
# =========== NCL ============
cat   >!   main.ncl   <<  "END_NCL"
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“
external SUB  "./sub.so"
begin
    ...
end
"END_NCL"
# ===========FORTRAN ========
cat   >!   sub.f   << "END_SUBF"
C NCLFORTSTART
  ...
C NCLEND
"END_SUBF"
# =========== WRAPIT==========
WRAPIT  sub.f
# =========== EXECUTE ========
ncl  main.ncl  >&!  main.out
```

Global Variables and Scope [1 of 2]

- **Global Variable(s)**
  - **by definition:** can be accessed from any function or procedure
  - different from local variables
- **NCL does not have explicit “global variables”**
  - requires understanding of NCL’s variable scope [identical to Pascal]
  - **http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclStatements.shtml#Scoping**

```
load “dummy_1.ncl”      ; not aware of constants below
GRAVITY         = 9.8
RGAS            = 204
load “dummy_2.ncl”      ; can use GRAVITY and RGAS
REARTH          = 6000000
load “dummy_3.ncl”      ; can use GRAVITY, RGAS, REARTH
begin                   ; can use GRAVITY, RGAS, REARTH
    :
end
```

Global Variables and Scope [2 of 2]

- **knowledgeable user can simulate … one approach**
  - create a file **GLOBAL.ncl**
  - populate with desired constants
  - best to follow some user defined conventions [*e.g.* capital letters]

```
; contents of   GLOBAL.ncl
GRAVITY         = 9.8
RGAS            = 204
REARTH          = 6000000.
GRAVITY_d       = 9.8
```

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
load "/my/path/GLOBAL.ncl“
load "foo_1.ncl“
begin
end
```

NCL as a scripting tool

```
begin
   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
end
```

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