NCL Visualization Workshop

Part 1: Introduction to NCL

NCL

NCAR Command Language

November 28-29, 2013

Deutsches Klimarechenzentrum

Karin Meier-Fleischer, DKRZ and Mary Haley, NCAR

DKRZ DEUTSCHES KLIMARECHENZENTRUM

Max-Planck-Institut für Meteorologie

NCAR NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

CISL

NSF

NCAR is sponsored by the National Science Foundation

NCL team

Dennis Shea
Science guy
Data expert
Trainer

Dave Brown
NCL Tech Lead
Everything

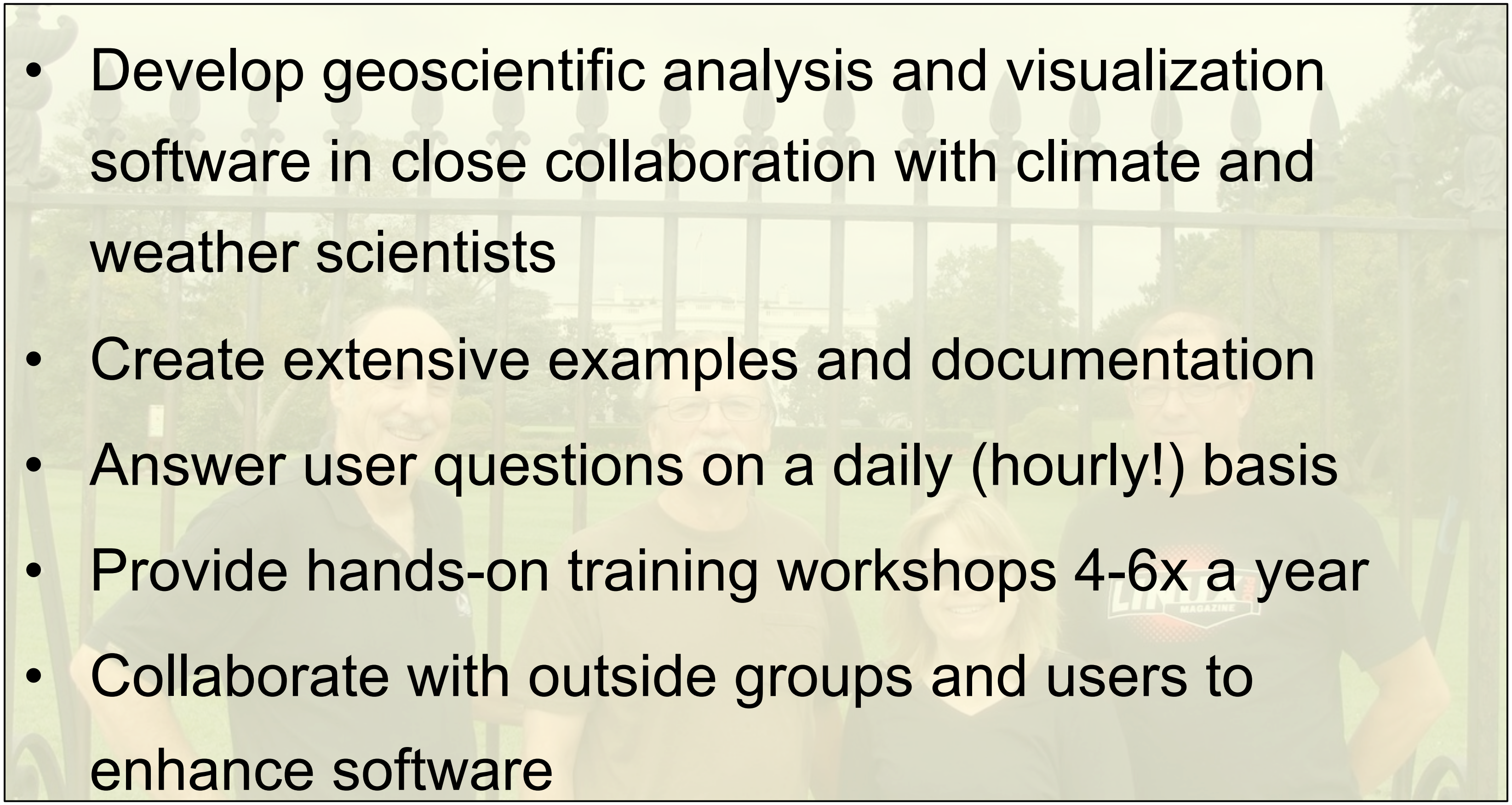
Mary Haley
Project lead
Trainer

Rick Brownrigg
Developer
Research

Wei Huang
Developer
Data Formats

Adam Phillips
Science guy
Graphical expert

NCL team



- Develop geoscientific analysis and visualization software in close collaboration with climate and weather scientists
- Create extensive examples and documentation
- Answer user questions on a daily (hourly!) basis
- Provide hands-on training workshops 4-6x a year
- Collaborate with outside groups and users to enhance software

Dennis Shea
Science guy
Data expert
Trainer

Dave Brown
NCL Tech Lead
Everything

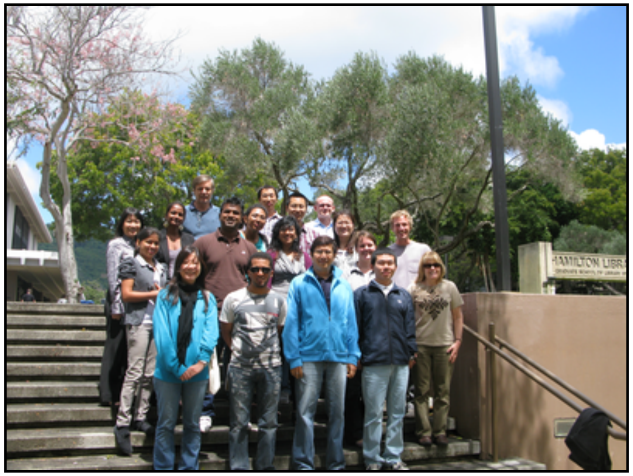
Mary Haley
Project lead
Trainer

Rick Brownrigg
Developer
Research

Wei Huang
Developer
Data Formats

Adam Phillips
Science guy
Graphical expert

14 NCL Workshops in last 2 years

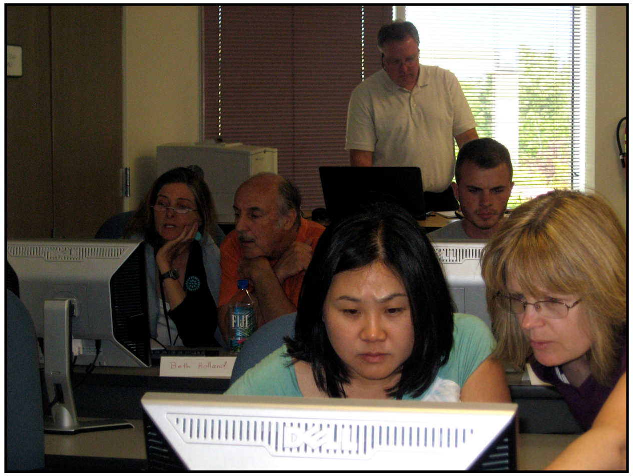


University of Hawaii @ Manoa

CERFACS, Toulouse, France

Jackson State, Mississippi

Yale School of Forestry and
Environmental Studies



NCAR, Boulder

University of Alaska @ Fairbanks

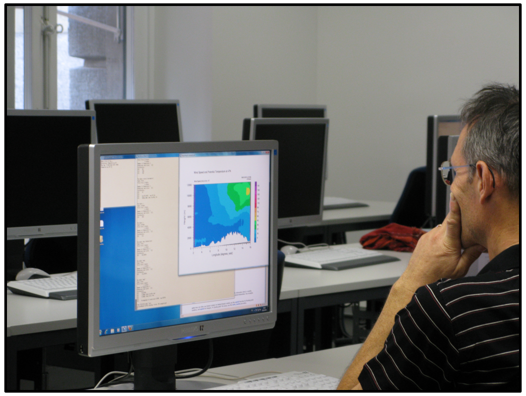
68 workshops since Feb 2000, 1041 students

4 NCL Workshops at MPI

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

International Workshops

MPI Hamburg 2008-2013



ETH Zürich 2010

BoM Melbourne 2011

UNSW Sydney 2011

CERFACS Toulouse 2012

UFRN Natal 2013

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

MAX-PLANCK-GESELLSCHAFT

CERFACS

DKRZ

Australian Government
Bureau of Meteorology

CRG

APCC
APEC CLIMATE CENTER

ETH Zürich

cnrs
dépasser les frontières

LEGOS

CPTEC

METEO FRANCE
Toujours un temps d'avance

Mercator Ocean
Ocean Forecasters

Important note

- I am hard-of-hearing in both ears and read lips.
- Questions welcome! Raise hand and get my attention first. You may need to gesture wildly.
- If I don’t understand, speak up slightly and more slowly. You shouldn’t have to yell. ☺

Thanks

- Karin Meier-Fleischer, DKRZ
  - NCL Tutorial is well-written, lots of examples!
- Michael Böttinger, DKRZ
- Niklas Röber, DKRZ
- Antje Weitz, Bjorn Stevens, MPI
- Wiebke Boehm, MPI

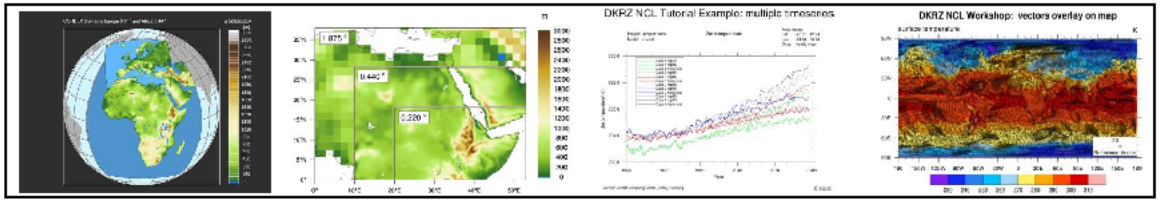
NCL

TUTORIAL

High Quality Graphics
with
NCL 6.1.2

Karin Meier-Fleischer
Michael Böttinger
DKRZ

Version: 1.0 2013/10/22


0.875°
0.440°
0.220°
DKRZ NCL Tutorial Example: multiple timeseries
DKRZ NCL Workshop: vectors overlay on map
surface temperature
K

Purpose of this workshop

- Introduce you to NCL and its language features
- Show you how to examine and read NetCDF files
- Show you how to create high-quality visualizations with NCL
- Lab exercises: focused on reading NetCDF files and creating two-dimensional visualizations and animations

Purpose of this lecture

Geared towards new users of NCL, but all users are welcome. Assumption that you have some knowledge of programming.

- Give you a quick overview of NCL
- Introduce you to NCL language basics
- Discuss importance of metadata
- Demonstrate looking at NetCDF files
- Demonstrate reading variables from NetCDF files
- Do a website tour (if there's time)

Topics

- Overview of NCL
- NCL language basics
  - How to run NCL
  - Language syntax
  - Variables (scalars and arrays)
- Metadata
- NetCDF files
- Website tour

Topics

- Overview of NCL
- NCL language basics
  - How to run NCL
  - Language syntax
  - Variables (scalars and arrays)
- Metadata
- NetCDF files
- Website tour

First . . . an informal survey

- What software do you use?
  - Fortran, C/C++
  - Scripting languages and tools: Matlab, IDL, Python, R, Ruby, Java, NCL, Climate data operators (CDO), NetCDF operators (NCO), GrADS, Ncview, Avizo, ParaView
- What types of data do you work with?
  NetCDF, HDF, HDF-EOS, GRIB, Shapefiles
- Tell us about yourself: name, where you work, what kind of data or models do you work with, what software do you currently use?

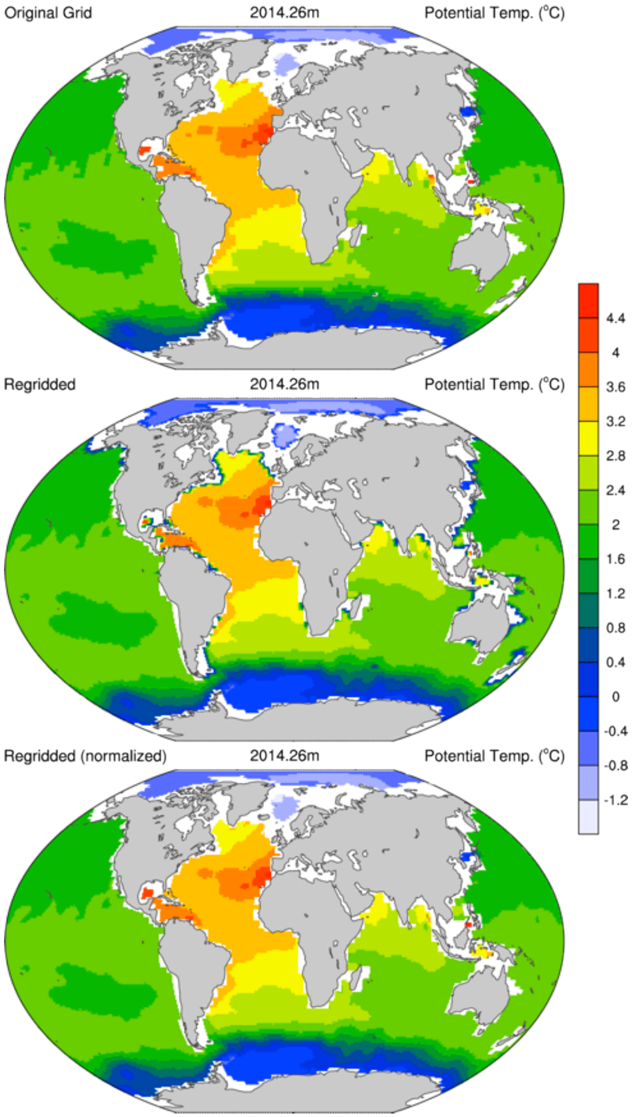
NCAR Command Language (NCL)

A scripting language developed at NCAR and tailored for the analysis and visualization of geoscientific data

1. Simple, robust file input and output
2. Hundreds of analysis (computational) functions. Can call your own Fortran/C code from NCL.
3. Visualizations (2D) are publication quality and highly customizable

- Users range from grad students doing individual research to programmers in scientific organizations working on large scale projects
- UNIX binaries & source available, free
- Extensive website, training workshops

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

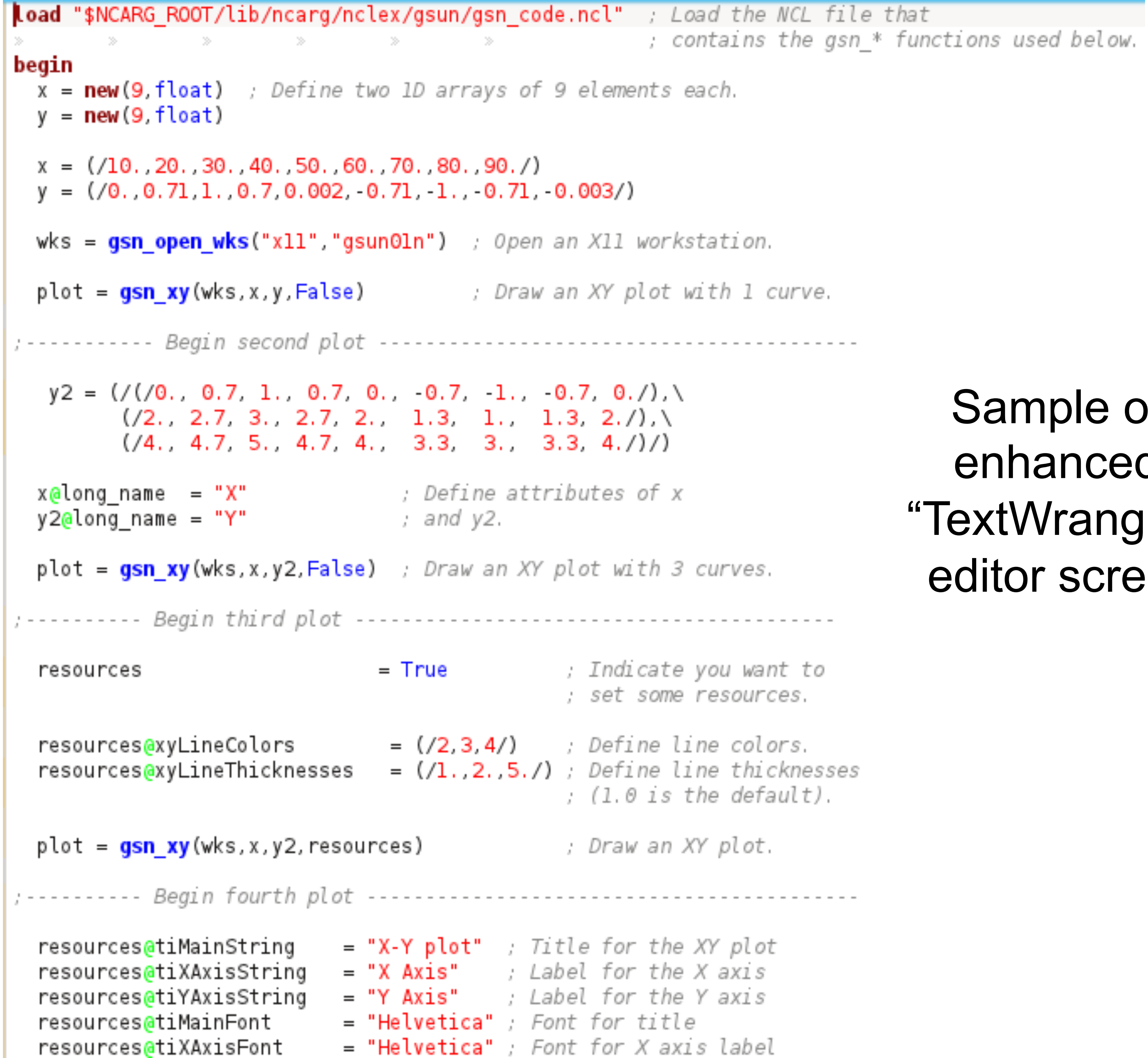


Topics

- Overview of NCL
- NCL language basics
  - How to run NCL
  - Language syntax
  - Variables (scalars and arrays)
- Metadata
- NetCDF files
- Website tour

NCL basics

- You can run NCL interactively or in batch mode
- We highly recommend batch mode!
- Interactive is useful for trying things out
- Batch mode: use a UNIX editor like “emacs”, “vi”, “nedit”, “TextWrangler”, “NetBeans”
- There are editor enhancements available http://www.ncl.ucar.edu/Applications/editor.shtml



```
load "$NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"  ; Load the NCL file that
                                                      ; contains the gsn_* functions used below.
begin
  x = new(9,float)  ; Define two 1D arrays of 9 elements each.
  y = new(9,float)

  x = (/10.,20.,30.,40.,50.,60.,70.,80.,90./)
  y = (/0.,0.71,1.,0.7,0.002,-0.71,-1.,-0.71,-0.003/)

  wks = gsn_open_wks("x11","gsun01n")  ; Open an X11 workstation.

  plot = gsn_xy(wks,x,y,False)         ; Draw an XY plot with 1 curve.

;---------- Begin second plot ------------------------------------

   y2 = (/(/0., 0.7, 1., 0.7, 0., -0.7, -1., -0.7, 0./),\
          (/2., 2.7, 3., 2.7, 2.,  1.3,  1.,  1.3, 2./),\
          (/4., 4.7, 5., 4.7, 4.,  3.3,  3.,  3.3, 4./)/)

  x@long_name  = "X"            ; Define attributes of x
  y2@long_name = "Y"            ; and y2.

  plot = gsn_xy(wks,x,y2,False) ; Draw an XY plot with 3 curves.

;---------- Begin third plot ------------------------------------

  resources                    = True          ; Indicate you want to
                                               ; set some resources.

  resources@xyLineColors       = (/2,3,4/)     ; Define line colors.
  resources@xyLineThicknesses  = (/1.,2.,5./)  ; Define line thicknesses
                                               ; (1.0 is the default).

  plot = gsn_xy(wks,x,y2,resources)            ; Draw an XY plot.

;---------- Begin fourth plot ------------------------------------

  resources@tiMainString    = "X-Y plot"  ; Title for the XY plot
  resources@tiXAxisString   = "X Axis"    ; Label for the X axis
  resources@tiYAxisString   = "Y Axis"    ; Label for the Y axis
  resources@tiMainFont      = "Helvetica" ; Font for title
  resources@tiXAxisFont     = "Helvetica" ; Font for X axis label
```

Sample of enhanced “TextWrangler” editor screen

Running NCL interactively

Open a UNIX terminal window and type:

```
ncl <return>
```

Several lines will be echoed. You will get a prompt where you can type commands:

```
Copyright (C) 1995-2013 - All Rights Reserved
University Corporation for Atmospheric Research
NCAR Command Language Version 6.1.2
The use of this software is governed by a License Agreement.
See http://www.ncl.ucar.edu/ for more details.
ncl 0> print("hello")
(0)     hello
ncl 1> quit
```

Command line options

To get the version:

```
% ncl –V
6.1.2
```

To get a list of other options:

```
% ncl –h
Usage: ncl -fhnpxV <args> <file.ncl>
       -f: Use New File Structure, and NetCDF4 features
       -n: don't enumerate values in print()
       -p: don't page output from the system() command
       -x: echo NCL commands
       -V: print NCL version and exit
       -h: print this message and exit
```

Running NCL in batch mode

- Create an NCL script using a UNIX editor
- Call it whatever you like. We recommend ending it with ".ncl", like "**plot_icon.ncl**"
- Run "ncl" on it on the UNIX command line:

```
      ncl plot_icon.ncl
```

- Printed output will appear on UNIX standard out
- Graphical output will go wherever you tell it (later)

NCL language basics

- Optionally start and end script with “begin” and “end”
- Comments start with “;” – they can be on line by themselves or at the end of a line.
- Code can start anywhere on a line
- Strings are always enclosed in double quotes (“Hello DKRZ”)
- A routine that returns a value is called a “function”
- A routine that doesn’t return a value is called a “procedure”
- Continuation character is a backwards slash (“\”)

NCL_basics_1.ncl

```
begin

;---Open a netCDF file and print contents
  f = addfile (“ECHAM5_OM_A1B_2001_0101-1001_2D.nc”,”r”)

;---Read “slp” off NetCDF file and print its info
  slp = f->slp                   ; (time,lat,lon)
  printVarSummary(slp)
  print(min(slp))    ; function inside a procedure
  print(max(slp))

;---Calculate running average across time
  slp_ts = runave (slp, 3, 0)
  print(min(slp_ts))
  print(max(slp_ts))

end
```

Output from “NCL_basics_1.ncl” script

```
Variable: slp
Type: float
Total Size: 2949120 bytes
            737280 values
Number of Dimensions: 3
Dimensions and sizes:     [time | 40] x [lat | 96] x [lon | 192]
Coordinates:
            time: [   0.. 234]
            lat: [88.57216851400727..-88.57216851400727]
            lon: [-180..178.125]
Number Of Attributes: 5
  long_name :mean sea level pressure
  units :     Pa
  code :      151
  table :     128
  grid_type :gaussian
(0)     94403.69
(0)     105950.7
(0)     94422.02
(0)     105835.5
```

print versus printVarSummary

Both procedures important for debugging!

```
x = (/1,2,3,-999,5/)
print(x)
printVarSummary(x)
```

“Look at your data!”

print

```
Variable: x
Type: integer
Total Size: 20 bytes
            5 values
Number of Dimensions: 1
Dimensions and sizes:   [5]
Coordinates:
(0)      1
(1)      2
(2)      3
(3)      -999
(4)      5
```

printVarSummary

```
Variable: x
Type: integer
Total Size: 20 bytes
            5 values
Number of Dimensions: 1
Dimensions and sizes:   [5]
```

NCL_basics_2.ncl

```
;---Note: no begin or end

;---Open a netCDF file and print contents
  filename = “ECHAM5_OM_A1B_2001_0101-1001_2D.nc”
  print(“Opening a NetCDF file called “ + filename)
  f = addfile (filename, “r“)

;---Read first time step and print info
  slp = f->slp(0,:,:)      ; (lat x lon)
  printVarSummary(slp)
  print(“min/max slp = “ + min(slp) + “/“ + max(slp))

;---Calculate average across all values
  slp_avg = avg (slp)     ; returns a single value
  printVarSummary(slp_avg)
  print(“Average of slp = “ + slp_avg)
```

```
Opening a NetCDF file called ECHAM5_OM_A1B_2001_0101-1001_2D.nc

Variable: slp
Type: float
Total Size: 73728 bytes
            18432 values
Number of Dimensions: 2
Dimensions and sizes:   [lat | 96] x [lon | 192]
Coordinates:
            lat: [88.57216851400727..-88.57216851400727]
            lon: [-180..178.125]
Number Of Attributes: 6
  time :             0
  long_name :   mean sea level pressure
  units :       Pa
  code :        151
  table :       128
  grid_type :   gaussian
min/max slp = 96715.2/105364

Variable: slp_avg
Type: float
Total Size: 4 bytes
            1 values
Number of Dimensions: 1
Dimensions and sizes:   [1]
Coordinates:
Average of slp = 100941
```

Output from “NCL_basics_2.ncl” script which was executed with command line option “-n”:

```
ncl –n NCL_basics_2.ncl
```

Scalar variables

```
;---Explicit scalar assignment
ndys = 30                    ; type integer
x_f  = 2983.599918           ; type float
long_name = “Water Vapor”    ; type string

;---Use “literals” to force a type
d = 3.14159265358979d        ; double
dim = 32676l                 ; long
short_val = 10h              ; short

;---Logicals have no quotes
done = True                  ; False

;---Variables are case-sensitive
Lat = 10.8   ; these are three
LAT = 90.    ; different variables
lat = -30
```

Mixing types: calculations and strings

```
;---Mixing types, “largest” type used
i = 7/10   ; integer (i=0)
x = 7/10.  ; float   (x=0.7)

y = (22./7)/2d  ; double (1.57142857368774)

z = (i+5) * x   ; float (z=3.5)

;---”atan” returns a float
rad2deg = 45/atan(1)     ; 57.29578

;---Use “+” for string concatenation
str = “x = “ + 2            ; “x = 2”

j = 2
s = “var_” + (j+1) + “_f“   ; s = “var_3_f“
```

Coercing values to other types

```
;
; Use conversion functions “toxxx” to convert a
; value to a “lower” type. Precision will be
; compromised.
;
dx = 12345.678901234d     ; dx is double
fx = tofloat(dx)          ; fx = 12345.68
ix = toint(dx)            ; ix = 12345
iy = totype(x,”integer”)  ; iy = 12345


;---Strings are handled differently
s = tostring(dx)          ; “12345.678901”
s = tostring(iy)          ; s = “12345”
s = “” + iy              ; s = “12345”

;---Use “typeof” function to print type
print(typeof(dx))      ; “double”
```

Changing variables to a “higher” type

```
ff = 1.5e20     ; float
ff = 1000       ; Assigning integer value to
                ; float variable is okay!
ff = 1d36       ; Assigning double value
                ; to float not okay.
                ; Error: “type mismatch”

;---Use delete or reassignment (:=) operator
delete(ff)
ff = 1d36       ; double

;---This will work too
ff = 1.5e20
ff := 1d36
```

NCL statements

- “if” statement
- “do” loops
- “load” to load other NCL scripts
- “function” to define your own function
- “procedure” to define your own procedure
- “exit” to exit NCL script at that point
- “quit” if you are running interactively

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

“if” and “do” statements

```
if(varname.eq."slp".and.units.eq."Pa") then
  do something
else                     ; No “else if”
  do something else
end if               ; There must be a space
                     ; between “end” and “if”
```

```
do i=0,ndims-1    ; do i=ndims-1,0,1 for reverse
  do something
end do           ; space between “end” and “do”
```

```
j     = 0
found = False
do while (j.lt.nvalues .and. .not.found)
  if(something) then
    do something
    found = True
  end if
  do something
  increment j in some fashion
end do
```

Trick to get around lack of “elseif”

```
if(x.lt.0) then
  do something
else if(x.gt.0) then
  do something else
else       ; x = 0
  do something else
end if ; Need one of these for every “if”
end if
```

Array basics

- Row major like C/C++ (*Fortran is column major*)
- Leftmost dimension varies the slowest, rightmost varies fastest
- Use “(/” and “/)” to create arrays
- Dimensions are numbered left to right (0,1,…)
- Indexes (subscripts) start at 0 (0 to n-1)
- Use parentheses to access elements
- Can do calculations across whole arrays without looping

Array basics

```
;---One-dimensional (1D) arrays, 3 elements
lat = (/-80,0.,80/)     ; float
LAT = (/-80,0,80/)      ; integer

;---1D string array, 4 elements
MM = (/"March", "April", "May", "June"/)

;---Create 3x2 two-dimensional (2D) double array
z = (/(/1,2d/),(/3,4/),(/9,8/)/)


;---Assume “x” is a one-dimensional array
dx = x(2) – x(1)  ; 3rd value minus 2nd value


;---Assume Y is three-dimensional(nx,ny,nz)
y1 = y(0,0,0)           ; y1 = first value of y
yn = y(nx-1,ny-1,nz-1)  ; yn = last value of y
```

Array subscripting

- Three kinds of array subscripting
  1. **Index (uses ':' and '::')**
  2. Coordinate (uses curly braces '**{**' and '**}**')
  3. Named dimensions (uses '**!**')
- You can mix subscripting types in one variable
- Be aware of dimension reduction
- Index subscripting is 0-based

  (Fortran by default is 1-based)

Array index subscripting, : and ::

```
;---Assume T is a 3D array (ntime x nlat x nlon)
  t = T              ; Copy entire array to new variable
  t = T(:,:,:)       ; Don't need to do this!

  t = (/T/)           ; Copy entire array, don't copy metadata
                      ; (_FillValue is retained)

;---The following examples create a 2D array "t" from "T"
  t = T(0,:,::5)        ; 1st time index, all lat, every 5th lon
                        ; (nlat x nlon/5)

  t = T(0,::-1,:50)     ; 1st time index, reverse lat,
                        ; first 51 lons (nlat x 51)

  t = T(:1,45,10:20)    ; 1st two time indices, 46th index of lat,
                        ; 11th-21st indices of lon (2 x 11)

;---To prevent dimension reduction, use n:n
  t = T(0:0,:,::5)              ; 1 x nlat x nlon/5
  t = T(:1,45:45,10:20)         ; 2 x 1 x 21
```

Calculations on arrays

- Don’t need to loop to do array calculations
- Arrays need to be same size, but scalars can be used anywhere; scalars are arrays with one dimension and one element
- Use “conform” function if you need to conform one array to size of another

Calculations on arrays

```
;---Assume “clat” and “clon” are lat/lon arrays in radians
rad2deg = 45./atan(1.)    ; radians to degrees
lat      = clat * rad2deg ; Convert to degrees
lon      = clon * rad2deg
lat@units = “degrees_north”   ; Good idea to do this!
lon@units = “degrees_east”


;---Can also do this
lat = (/clat * rad2deg/)   ; Special: don’t copy metadata

;---Be careful with ordering of syntax
zlev = (-7*log(lev/10^3))      ; evaluated as
                               ; (-7)*log(lev/(10^3))
;
; Use “conform” to promote an array to the size of another.
;                    0   1   2   3
; Assume “Twk” is (time,lat,lon,lev), and
; “ptp” is (time,lat,lon)
             0   1   2
ptropWk = conform(Twk, ptp, (/0,1,2/)) ; time,lat,lon,lev
```

Array efficiency

```
;---Inefficient
do i=0,ny-1
  do j=0,nx-1
    x(i,j) = y(i,j) * 0.01
  end do
end do

;---Efficient
x = y*0.01

;---Inefficient
do i=0,nlon-1
  if(lon(i).lt.0) then
    lon(i) = lon(i) + 360.
  end if
end do
;---Efficient
lon = where(lon.lt.0,lon+360,lon)
```

Array reorder, reshape, reverse

```
;---Reshaping an array
  t1D = ndtooned(T)               ; Convert to 1D array
  t2D = onedtond(t1D, (/N,M/) )   ; Convert to N x M array
```

```
;---Reordering an array, uses “named dimensions”
; Let T(time,lat,lon)
  t = T(lat|:,lon|:,time|:)   ; Can’t assign to same var
```

```
;---Reversing dimensions of an array

; Let T(lev,lat,lon)
  T = T(::-1,:,:)     ; Will reverse coordinate array too,
```

Special functions for arrays

```
;---Very useful “where” function
  q = where(z.gt.pi .and. z.lt.pi2, pi*z, 0.5*z)
```

```
; “num”, “any”, “all”

  npos = num (xTemp.gt.0.0)

  if (.not.any(string_array.eq.”hello world”)) then
    do something
  end if


  if (all(xTemp.lt.0)) then
    do something
  end if
```

```
; “ind” function, only on 1D arrays
  ii = ind(pr.lt.500. .and. pr.gt.60.)
```

Useful array functions

- "dimsizes" – get dimension sizes
- "any" or "all" – check array values
- "where" – perform operation on array based on conditional statements
- "conform" – make a smaller array conform to size of larger array
- "mask" – mask an array based on another array
- "ind" – get the indexes of a one-dimensional array where an array condition is True
- "reshape" – reshape an array to another dimension size
- "ndtooned" and "onedtond" – convert arrays from one-dimensional to multi-dimensional, and vice versa

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

DEMO

- Creating scalar and array variables
- Calling NCL functions
- Using "print" and "printVarSummary"

http://www.ncl.ucar.edu/Training/Workshops/interactive.shtml

Topics

- Overview of NCL
- NCL language basics
  - How to run NCL
  - Language syntax
  - Variables (scalars and arrays)
- Metadata
- NetCDF files
- Website tour

Metadata

- Metadata is information about a variable.
- Metadata can consist of:
  - Attributes (can describe files and variables)
  - Named dimensions (describes a variable’s dimensions)
  - Coordinate arrays (coordinates for data values)
- “_FillValue” attribute is special: indicates a variable’ s missing value
- When you do an “ncdump -h” or “ncl_filedump” on a “self-describing” data file, you see all the metadata
- NCL will use metadata in many cases,

Why is metadata important?

- Can give important information about a variable: units, description, location (lat/lon), date, how it was calculated, etc.
- Languages like NCL, GrADS, Ncview, CDO, NCO, depend on metadata to correctly interpret data for calculations and graphics
- When you share data files with someone else, metadata is like a document for your data.

NetCDF Climate and Forecast (CF) Metadata Conventions
http://cf-pcmdi.llnl.gov

Metadata assignment (attributes)

```
; Use the “@” symbol to assign attribute metadata.
; Useful for assigning units, long names, missing vals
; Assume “T” is 3 x 4 x 5 float array of temperature
; values in degrees celsius.

  T@_FillValue   = -999                ; Missing value
  T@units        = “deg C”
  T@long_name    = “temperature”
  T@wgts         = (/ 0.25, 0.5, 0.25 /)
  printVarSummary(T)    ; To see contents of T
```

printVarSummary(T) results

```
Variable: T
Type: float
Total Size: 240 bytes
            60 values
Number of Dimensions: 3
Dimensions and sizes:   [3] x [4] x [5]
Coordinates:
Number Of Attributes: 5
  wgts :          ( 0.25, 0.5, 0.25 )
  long_name :     temperature
  units :         deg C
  _FillValue :    -999
```

Metadata assignment (named dimensions)

```
; Named dimensions are useful for arrays.
; Use the “!” symbol to name dimensions.
; Assume “T” is same 3D array as before
  T!0 = “time” ; Leftmost dimension
  T!1 = “lat”  ; Middle dimension
  T!2 = “lon”  ; Rightmost dimension

  printVarSummary(T)  ; To see metadata of T
```

```
Variable: T
Type: float
Total Size: 240 bytes
            60 values
Number of Dimensions: 3
Dimensions and sizes:  [time | 3] x [lat | 4] x [lon | 5]
Coordinates:
Number Of Attributes: 5
  wgts :        ( 0.25, 0.5, 0.25 )
  long_name :   temperature
  units :       deg C
  _FillValue :  -999
```

Missing values ("_FillValue" attribute)

- "_FillValue" is a NetCDF *and* NCL reserved attribute
- Must be same as type of variable
- "missing_value" attribute has **no** special status to NCL.

  If "T" has "missing_value" attribute and no "_FillValue":

```
T@_FillValue = T@missing_value
```

- Best not to use zero as a _FillValue
- Default missing values for all NCL variable types:

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

Missing values in an NCL script

- Most NCL functions ignore _FillValue:

```
x            = (/1,2,3,-999,5/) ; no msg val yet
xavg         = avg(x)           ; = -197.6
x@_FillValue = -999             ; now has a msg val
xavg         = avg(x)           ;(1+2+3+5)/4 = 2.7
```

- Use “default_fillvalue” to set a missing value for a variable that doesn’t have one:

```
x@_FillValue = default_fillvalue(typeof(x))
```

Missing value functions

- Use any, all, and ismissing functions to query a variable for missing values:

```
if (.not.any(ismissing(T))) then
   do something
end if
if (all(ismissing(T))) then
   do something
end if
```

- Use num & ismissing to count missing values:

```
nmsg = num(ismissing(T))
```

Metadata assignment (coordinate arrays)

```
; Coordinate arrays are 1D arrays representing values
; for dimensions of an array. Use the “&” symbol to assign
; coordinate values; must name dimensions first.
  T!0    = “time”
  T!1    = “lat”
  T!2    = “lon”
  T&time = (/0,5,10/)           ; Coordinate arrays must
  T&lat  = fspan(-90,90,4)      ; be 1D and same length
  T&lon  = fspan(-180,180,5)    s; as dimension they represent
  T&lat@units = “degrees_north”
  T&lon@units = “degrees_east”
```

Important for graphics

```
Variable: T
Type: float
Total Size: 240 bytes
            60 values
Number of Dimensions: 3
Dimensions and sizes:   [time | 3] x [lat | 4] x [lon | 5]
Coordinates:
            time: [0..10]
            lat: [-90..90]
            lon: [-180..180]
```

DEMO

Looking at variables with metadata

http://www.ncl.ucar.edu/Training/Workshops/interactive.shtml

Array Subscripting

- Three kinds of array subscripting
  1. Index (uses ‘:’ and ‘::’) (already covered)
  2. **Coordinate (uses curly braces ‘{ ‘ and ‘}’)**
  3. **Named dimensions (uses ‘!’)**
- You can mix subscripting types in one variable
- Be aware of dimension reduction
- Index subscripting is 0-based (Fortran by default is 1-based)

http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclVariables.shtml#Subscripts

Array coordinate subscripting, {…}

```
; Consider T(ntime x nlat x nlon)

  t = T(:,{-30:30},:)        ; all time and lon, lat 30°S to 30°N

  t = T(0,{-20},{-180:35:3})  ; 1st time, lat nearest 20°S,
                              ; every 3rd lon from 180°W to 35°E
                              ; "t" will be one-dimensional
```

```
; Can mix index and coordinate subscripting

  t = T(:,{-30:30},1::2) ; all time, lat 30°S to 30°N,
                         ; every other lon starting with 2nd
```

NCL syntax characters

;	comment (on line by itself, or at end of line)
@	reference/create attributes
!	reference/create named dimensions
&	reference/create coordinate variables
{…}	coordinate subscripting
$...$	enclose strings when (im/ex)port variables via addfile
(/…/)	array construct characters
: or ::	array syntax
|	separator for named dimensions
\	continuation character
::	syntax for external shared objects (fortran/C)
->	use to (im/ex)port variables via addfile function

Topics

- Overview of NCL
- NCL language basics
  - How to run NCL
  - Language syntax
  - Variables (scalars and arrays)
- Metadata
- NetCDF files
- Website tour

Looking at NetCDF files

- Many ways to look at NetCDF files

  1. On the UNIX command line using “ncdump”
  2. On the UNIX command line using “ncl_fileump”
  3. With an NCL script using “addfile” function

DEMO

Looking at files with ncdump (NetCDF tool) and ncl_filedump (NCL tool)

Rectilinear grids – grids whose latitude and longitude arrays are "coordinate arrays" (one-dimensional arrays, and rightmost two dimensions are nlat x nlon)

Curvilinear grids – grids whose latitude and longitude arrays are two-dimensional arrays, and rightmost two dimensions are nlat x nlon)

Unstructured grids – one-dimensional arrays whose latitude and longitude arrays are also one-dimensional and all the same length

Writing NCL script to open NetCDF file

- “addfile” – open NetCDF, HDF4, HDF5, GRIB1, GRIB2, HDF-EOS2, HDF-EOS5, Shapefile
- “addfile” can also be used to write NetCDF or HDF4
- Variables read off these files contain everything, including metadata
- Use “->” syntax to read a variable off the file
- “addfiles” – read multiple files

DEMO

Reading and writing NetCDF files using “addfile”

Important URLS

- DKRZ NCL Tutorial Document

  http://mms.dkrz.de/pdf/vis/NCL_Tutorial_V1.1.pdf

- NCL Reference Manual

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

- Mini Reference Manual

  http://www.ncl.ucar.edu/Document/Manuals/language_man.pdf

- Frequently Asked Questions

  http://www.ncl.ucar.edu/FAQ/

- Metadata conventions

  http://www.unidata.ucar.edu/software/netcdf/examples/files.html

Topics

- Overview of NCL
- NCL language basics
  - How to run NCL
  - Language syntax
  - Variables (scalars and arrays)
- Metadata
- NetCDF files
- Website tour ?