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

**1 **Introduction......................................................................................................................1  
1.1 Documents from NCAR .................................................................................................1  
1.2 Example Scripts and Data ..............................................................................................1  
1.3 Support for NCL ..............................................................................................................2  
1.4 Installation ......................................................................................................................2  
1.5 NCARG_ROOT and PATH ............................................................................................2  
1.6 .hluresfile ......................................................................................................................2  
1.7 UNIX Editor Settings ......................................................................................................3  

**2 **Overview........................................................................................................................4  
2.1 Interactive Mode ..............................................................................................................4  
2.2 Batch Mode .....................................................................................................................5  
2.3 Input, Export and Graphics Output File Formats ............................................................6  
2.3.1 NetCDF ....................................................................................................................6  
2.3.2 NetCDF Metadata .....................................................................................................6  
2.4 Data Operators: CDOs and NCOs .................................................................................7  
2.5 User Guide Data Sets .....................................................................................................7  

**3 **Language Basics..............................................................................................................8  
3.1 Syntax Characters ..........................................................................................................8  
3.2 Expressions .....................................................................................................................8  
3.2.1 Algebraic Operators ..................................................................................................8  
3.2.2 Logical Operators ......................................................................................................9  
3.2.3 Array Expressions .....................................................................................................9  
3.2.4 Functions ................................................................................................................9  
3.3 Data Types .......................................................................................................................10  
3.4 Variables .........................................................................................................................10  
3.4.1 Metadata and Attributes .........................................................................................10  
3.4.2 Named Dimensions .................................................................................................11  
3.4.3 Coordinate Variables ...............................................................................................11  
3.4.4 String References ....................................................................................................12  
3.4.5 List Variables (Container Variables) .......................................................................12  
3.4.6 Variable Assignment ...............................................................................................14  
3.5 Arrays .............................................................................................................................21  
3.5.1 Standard Subscripts .................................................................................................22  
3.5.2 Named Subscripts .....................................................................................................22  
3.5.3 Coordinate Subscripts .............................................................................................23  
3.6 Statements .......................................................................................................................23  
3.6.1 Block .......................................................................................................................23  


3.6.2  If - Statement ........................................................................................................24
3.6.3  Loops .....................................................................................................................24
3.6.4  Assignment / Reassignment .................................................................................25
3.7  Print Data and Variable Information .......................................................................25
3.8  Reserved Keywords ..................................................................................................26
4  File I/O .......................................................................................................................27
  4.1  addfile ....................................................................................................................27
  4.2  addfiles...................................................................................................................28
  4.3  Assign a Variable from a File Variable .................................................................29
  4.4  Read ASCII File .....................................................................................................30
  4.5  Read Excel CSV Data Files ....................................................................................33
  4.6  Read Binary File .....................................................................................................35
  4.7  Write ASCII File .....................................................................................................37
  4.8  Write CSV File .......................................................................................................42
  4.9  Write Binary File .....................................................................................................42
  4.10 Write NetCDF File ................................................................................................44
5  Tools ..........................................................................................................................48
6  Data Processing ..........................................................................................................49
  6.1  NCL - Compute yearly means from monthly data ..................................................49
  6.2  NCL - Compute the time average at each grid point ..............................................50
  6.3  NCL - Compute the standard deviation of a dimension using \texttt{dim_stddev_n} ...50
  6.4  NCL - Compute the weighted area average .............................................................50
  6.5  NCL – Compute Linear Regression ........................................................................51
  6.6  NCL – Compute Running Mean .............................................................................52
  6.7  CDO - Compute annual means from monthly data .................................................54
  6.8  CDO - Compute the time average at each grid point ..............................................54
  6.9  CDO - Compute the temporal standard deviation at each grid point using \texttt{timstd} 
      54
  6.10 CDO - Compute the area average ........................................................................54
  6.11 CDO - Compute Linear Regression ......................................................................54
  6.12 CDO - Compute Running Mean ............................................................................55
  6.13 CDO – Select Variables .........................................................................................55
  6.14 CDO – Piping commands .......................................................................................55
7  Advanced NCL features ............................................................................................56
  7.1  Masking ....................................................................................................................56
  7.2  Date Conversion .....................................................................................................57
  7.3  String Operations ...................................................................................................58
  7.4  System Calls ...........................................................................................................59
  7.5  User-defined Functions and Procedures ...............................................................60
    7.5.1 Procedures .........................................................................................................60
8.18.1 Title Strings and Function Codes .......................................................... 149
8.18.2 Adding Text To The Plot .................................................................. 151
8.18.3 Function Codes for Creating Special Characters ........................... 154
8.18.4 Axis Annotations .............................................................................. 156
8.18.5 Contour Lines and Label Settings .................................................. 157
8.18.6 Colorizing Land, Ocean and Lakes ................................................ 158
8.18.7 Colorize Countries by Values ......................................................... 160
8.18.8 Labelbar Settings ............................................................................. 162
8.18.9 Legend Settings ............................................................................... 165
8.18.10 Tickmark Settings .......................................................................... 168
8.18.11 Date Format .................................................................................... 170
8.18.12 Insert a Logo .................................................................................. 172

9 Regridding ................................................................................................. 175
  9.1 ESMF Regridding .................................................................................. 175
    9.1.1 Curvilinear Grid to Rectilinear Grid .............................................. 176
    9.1.2 Curvilinear Grid to Rectilinear Grid from a given File ................. 178
    9.1.3 Unstructured Grid to Rectilinear Grid ........................................... 179
    9.1.4 Unstructured Grid to Rectilinear Grid from a Given File ............. 183
    9.1.5 Rectilinear Grid to Curvilinear Grid from a Given File ................ 186
    9.1.6 CMIP5 Grid to 1x1 degrees Grid .................................................... 190
  9.2 CDO Regridding ................................................................................... 192
    9.2.1 Curvilinear Grid to Gaussian N32 Grid ......................................... 192
    9.2.2 Curvilinear Grid to Rectilinear Grid (Korn-Shell Script) ............. 193

10 Using External Fortran or C Code ............................................................. 195
  10.1 Fortran ............................................................................................... 195
  10.2 C Code .............................................................................................. 197

11 Creating Images for PowerPoint, Keynote, Web .................................... 201

12 Customizing the NCL Graphics Environment ........................................ 202

13 Tips ........................................................................................................... 203
  13.1 Reverse Latitudes in Data File ......................................................... 203
  13.2 Convert NaNs (not a number) to _FillValue .................................. 203

14 PyNGL and PyNIO ................................................................................... 207
  14.1 XY-Plot ............................................................................................. 207
  14.2 Contour Plot – Rectilinear Gridded Data ........................................ 208
  14.3 Vector Plot – Rectilinear Gridded Data ............................................ 210
  14.4 Slice Plot – Rectilinear Gridded Data ................................................. 212
  14.5 Contour Plot – Curvilinear Gridded Data ........................................ 214
  14.6 Contour Plot – Unstructured Data .................................................. 215
  14.7 Triangles Plot – ICON Data .............................................................. 218

15 Common Error Messages ........................................................................ 222
1 Introduction

NCL (NCAR Command Language) is an open source interpreted language, designed specifically for scientific data processing and visualization. It is a powerful language for reading, writing, manipulating, and visualizing scientific data. It uses an internal netCDF variable model. Further, it supports a variety of input file formats: netCDF3, netCDF4, GRIB1, GRIB2, HDF-SDS, HDF-EOS, HDF5, Fortran/C binary, shapefiles and ASCII.

This tutorial will give an introduction on the use of interactive NCL for testing purposes, or in batch mode with NCL scripts to process and visualize data.

NCL Home page: http://www.ncl.ucar.edu/


1.1 Documents from NCAR

Two printable NCL PDF manuals, the so-called called “mini” documents are available:


Manuals web page: http://www.ncl.ucar.edu/Document/Manuals/
Getting Started: http://www.ncl.ucar.edu/Document/Manuals/Getting_Started/
Examples: http://www.ncl.ucar.edu/Document/Manuals/Getting_Started/examples.shtml

Further information can be found online on the documentation page:

Online Documentation: http://www.ncl.ucar.edu/Document/
NCL Frequently Asked Questions (FAQ): http://www.ncl.ucar.edu/FAQ/

1.2 Example Scripts and Data

All example scripts used in the NCL User Guide are distributed within the NCL software since version 6.4.0. That applies to many example data sets used in the NCL User Guide and the bigger data sets are provided on the NCL web page only.

Example scripts: $NCARG_ROOT/lib/ncarg/nclex/nug/
or

Example data sets: $NCARG_ROOT/lib/ncarg/data/nug/
or
1.3 Support for NCL

The NCL group at NCAR offers two email lists: one for installation issues and the second for general questions, information exchange and bug reports.

Email support information:

http://www.ncl.ucar.edu/Support/email_lists.shtml

Before you send a request to the email lists, you have to subscribe to them. Please read the posting guidelines first:

http://www.ncl.ucar.edu/Support/posting_guidelines.shtml

1.4 Installation

All information on how to download and install NCL on a UNIX-based operating system (Linux, MacOSX and Cygwin/X) can be found on the NCL installation web page:

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

1.5 NCARG_ROOT and PATH

If you have installed NCL on your local computer, you will have to do the following steps:

Set the NCARG_ROOT environment variable to the root directory of where the NCL software is installed, e.g. /opt/local/ncl-6.2.1. Then add the bin directory of the NCARG_ROOT path to your PATH environment variable.

For sh:

```bash
NCARG_ROOT="/opt/local/ncl-6.2.1"
export NCARG_ROOT
PATH=${NCARG_ROOT}/bin:$PATH
```

For bash or ksh:

```bash
export NCARG_ROOT="/opt/local/ncl-6.2.1"
export PATH=${NCARG_ROOT}/bin:$PATH
```

For csh and tcsh:

```csh
setenv NCARG_ROOT "/opt/local/ncl-6.2.1"
setenv PATH "$NCARG_ROOT/bin:$PATH"
```

1.6 .hluresfile

NCL has a default graphical environment that most users prefer to alter. This is accomplished through the .hluresfile. Upon execution, NCL looks for this file in the user's home directory.

The following lists the most common usage of this file:

```
! White background/black foreground (the default)
*wkForegroundColor : ([0.,0.,0.])
*wkBackgroundColor : ([1.,1.,1.])
! Color map (default is ncl_default)
*wkColorMap : rainbow
```
Placing this file in your home directory would result in a large X11 window size, larger PNG images, a common font, different default color map, and plots that have white as the background color and black as the foreground color.

You can download a sample .hluresfile at:  

1.7 UNIX Editor Settings

It is very helpful when editors can do syntax specific highlighting, which means that language specific names, resource names, built-in functions, and procedures of your NCL script are colored (highlighted) in your editor window. Then e.g. you can easily see if a resource or function name you typed is misspelled or correct (colored) and it is more convenient reading the script text when script parts are colored, too.

Many different editor plugins or settings are available to support syntax specific highlighting for NCL scripts, like Emacs, NEdit, VIM, JED, TextMate, gedit, Aquamacs, NetBeans, TextWrangler, Kate, Sublime Text, and Notepad++.

The page http://www.ncl.ucar.edu/Applications/editor.shtml contains the installation instructions of the editor enhancements and some handy scripts for customizing various editors to do special highlighting of NCL syntax. They were documented and contributed by other users.
2 Overview

NCL was designed for the analysis and visualization of scientific data, specifically in the area of atmospheric modelling. It combines many features of modern programming languages with a huge number of analysis and visualization functions and examples. For testing purposes, NCL can be run in an interactive mode, meaning that each command is interpreted as it is entered in your workstation. For more complex and repeated work, it is recommended to first write the NCL commands into script files. These files can later be used in batch mode where NCL works as an interpreter of the complete script.

2.1 Interactive Mode

For a short example on how to use NCL in interactive mode, enter the following at a UNIX prompt and hit <return>:

```
ncl
```

NCL will prompt:
```
ncl 0>
```

Next, enter:
```
val=102
a=val/4.
print(a)
```

You should get the following lines on the standard output:
```
Variable: a
Type: float
Total Size: 4 bytes
  1 values
Number of Dimensions: 1
Dimensions and sizes:   [1]
Coordinates:
  (0)     25.5
```

To exit NCL, enter:
```
quit
```

There are several command line options you can include when running NCL. For example, to get the NCL version:
```
ncl -V
```

To get information on all the NCL command line options, type:
```
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
-o:   retain former behavior for certain backwards-incompatible changes
-x:   echo NCL commands
-V:   print NCL version and exit
-h:   print this message and exit
```

To prevent printing the NCL copyright information on stdout at the beginning you can use the undocumented option '-Q':
```
ncl -Q
```
To record an interactive session:

```
 ncl 0> record "my_script.ncl"
 ncl 1> statements
 ...
 ncl 8> stop record
 ncl 9> quit
```

### 2.2 Batch Mode

NCL can be used interactively by entering the commands and settings directly in the command line. This might be useful for testing purposes or development work. However, especially for repeated actions, it is suggested that NCL scripts be used in order to save time. Due to the fact that NCL offers a variety of graphical resources, you won’t want to enter lengthy commands again and again. Instead, open a UNIX editor and create an NCL script which can be easily changed and executed.

The commands from the interactive section can be written into a file using the `record` command in NCL:

```
 ncl <return>
 ncl 0> record "my_script.ncl"
 ncl 1> val=102
 ncl 2> a=val/4.
 ncl 3> print(a)
 ncl 4> stop record
 ncl 5> quit
```

The `my_script.ncl` file will contain all commands after the `record` statement and before the `stop record` statement. The saved script will include any errors you have made so you may have to edit `my_script.ncl` prior to execution. Now you are able to run your first NCL script file `my_script.ncl` in batch mode:

```
 ncl my_script.ncl
```

You should get the following text returned:

```
Variable: a
Type: float
Total Size: 4 bytes
  1 values
Number of Dimensions: 1
Dimensions and sizes:   [1]
Coordinates:
  (0) 25.5
```

The `-n` option prevents the enumeration of the `print()` command. To prevent the long output of the variable description, use a Unix-pipe and the “`tail`” command to only print the last line of the output:

```
 ncl -n my_script.ncl | tail -1
```

Will return:

```
 25.5
```
2.3 Input, Export and Graphics Output File Formats

NCL can import the following data file formats:

- netCDF3, netCDF4
- HDF4-SDS, HDF2-EOS, HDF5, HDF5-EOS
- GRIB 1, GRIB 2
- CCM History Tape
- shapefiles
- binary
- ASCII

NCL can export (write) the following data file formats:

- netCDF, netCDF4
- HDF4, HDF5
- Binary (flat or fortran sequential; big or little endian)
- ASCII

NCL can write the following graphics output file formats:

- PS, EPS, EPSI
- PDF
- PNG
- SVG (good for web usage)
- NCGM (an older format that is generally not recommended)
- X11 (graphics output only to a X11 window)

2.3.1 NetCDF

NetCDF (Network Common Data Form) is a self-describing and machine independent data format from UNIDATA:

http://www.unidata.ucar.edu/software/netcdf/

NetCDF is a “container format”; different data structures and different types of content are supported. Along with the data itself, metadata needed for the interpretation of the data can be stored in the files.

2.3.2 NetCDF Metadata

The netCDF metadata section allows you to store a description that should enable users to correctly interpret or use the data: the underlying grid, its dimensions and coordinates (lat, lon, depth, time, ..), the variable names, the units used for the variables, global information and so forth.

One of the most commonly used conventions in climate modelling is the NetCDF Climate and Forecast (CF) Metadata Convention (NetCDF-CF):

http://cf-pcmdi.llnl.gov/

CF-Metadata: "The conventions define metadata and provide a definitive description of what the data in each variable represents and the spatial and temporal properties of the data".
2.4 Data Operators: CDOs and NCOs

Within the atmospheric, oceanographic and land model communities, general purpose tools which can be executed from the command line are commonly used to process data sets. These tools are very efficient and simple to use. No ‘programming’ is involved. Each operator typically performs one task. The output from the tools are files (typically, netCDF) containing the computational results. For example, it might be advantageous to extract one or more selected variables and/or time steps from a large data set spanning multiple files. Subsequently, the derived file may be used as input to a tool such as NCL, Matlab or Python. This and many other tasks can be done using the Climate Data Operators (CDO) developed by the Max-Planck-Institute for Meteorology. The CDO is a collection of command line operators that manipulate and facilitate analysis of climate data sets in netCDF and GRIB formats:

https://code.zmaw.de/projects/cdo/embedded/1.6.4/cdo.pdf

A feature is that the output of each CDO operator can be used as input to the next operator. This operation is called pipe lining.

cdo [options] operator_1 [operator_2 [operator_n]]

Manipulation and some calculations on data within netCDF and some HDF files can be done with the NetCDF Operators (NCO):

http://nco.sourceforge.net/

2.5 User Guide Data Sets

The datasets used in this document are available for download:


or in the directory

$NCARG_ROOT/lib/ncarg/data/nug/

The most important rule in data processing is:

Look At Your Data

You can avoid many errors and warnings if you are familiar with your data. To get an overview of a file's contents, type either of the following at the command line:

For a netCDF-3/4, GRIB-1/2 or HDF-4/5 enter:

```
ncl Filedump <the_filename>
```

or for netCDF-3/4

```
ncdump -h <the_filename>
```
3 Language Basics

Similar to fortran, C, Matlab, IDL, etc, NCL has many features of modern programming languages like variables, data types, constants, functions, procedures, operators (arithmetic, relational and logical), expressions, conditional statements, loops, functions and procedures. To get more comfortable with the work in this document, simple example scripts which may readily be executed, can be found in subsequent chapters.

3.1 Syntax Characters

Commonly used syntax characters include:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>assignment syntax</td>
</tr>
<tr>
<td>:=</td>
<td>reassignment operator</td>
</tr>
<tr>
<td>;</td>
<td>starts a comment</td>
</tr>
<tr>
<td>/;...;/</td>
<td>starts a block comment</td>
</tr>
<tr>
<td>@</td>
<td>create or reference an attribute</td>
</tr>
<tr>
<td>!</td>
<td>create or reference a named dimension</td>
</tr>
<tr>
<td>&amp;</td>
<td>create or reference a coordinate variable</td>
</tr>
<tr>
<td>$...$</td>
<td>enclose strings when importing or exporting variables via addfile</td>
</tr>
<tr>
<td>{...}</td>
<td>subscript arraya using coordinate values</td>
</tr>
<tr>
<td>[...]</td>
<td>subscript variables of type list</td>
</tr>
<tr>
<td>(...)</td>
<td>array constructor</td>
</tr>
<tr>
<td>[....]</td>
<td>list constructor</td>
</tr>
<tr>
<td>:</td>
<td>array syntax delimiter</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>continuation character for wrapping long lines</td>
</tr>
<tr>
<td>::</td>
<td>separator when calling external codes</td>
</tr>
<tr>
<td>-&gt;</td>
<td>used for inputting/outputting supported data formats</td>
</tr>
</tbody>
</table>

3.2 Expressions

After the execution of an expression, a value will be returned. There are three different kinds of expressions:

- Algebraic expressions
- Logical expressions
- Array expressions
- Functions

3.2.1 Algebraic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition, string concatenation</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction / Negation</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Modulus (integers only)</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>^</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>#</td>
<td>Matrix multiplication</td>
</tr>
</tbody>
</table>
Note:

• Use parentheses to circumvent precedence rules:

  - $x = (2 + 3) \times 3 \rightarrow 15$
  - `print(-(3+2)^2)` \rightarrow 25
  - `print(-((3+2)^2))` \rightarrow -25

• The + sign is an overloaded operator, which means that it has two different applications:

  - Addition: \( x = 2.3 + 5.8 \) \rightarrow \( x = 8.1 \)
  - String concatenation: "Value: "+12.7 \rightarrow "Value: 12.7"

• The – sign can also be used in two different ways:

  o negation has the highest precedence:
    - \( x = -3^2 \) is equal to \( x = (-3)^2 \) \rightarrow 9
  
  o treated as a minus:
    - \( x = 8 - 3^2 \) \rightarrow -1

3.2.2 Logical Operators

| .lt.  | Less than |
| .le.  | Less than or equal |
| .eq.  | Equal |
| .ne.  | Not equal |
| .ge.  | Greater than or equal |
| .gt.  | Greater than |
| .and. | AND |
| .or.  | OR |
| .xor. | Exclusive OR |
| .not. | NOT |

Logical expressions are guaranteed to be evaluated left to right. Hence, for efficiency, put the logical expression which will most likely fail on the left side:

```
if ( x .gt. 3 .and. x .lt. 7) then
  [statement(s)]
end if
```

3.2.3 Array Expressions

NCL’s arithmetic operators (add, multiply, divide, compare and so forth) can be applied for both scalars and arrays. See Arrays in chapter 3.5.

3.2.4 Functions

A function is an expression, too, because it returns a value. Functions include statements and are called by their name and an argument list. See Functions in chapter 7.5.
3.3 Data Types

Numeric data types:
- double (64 bit)
- float (32 bit)
- long (32 bit or 64 bit; signed +/-)
- integer (32 bit; signed +/-)
- short (16 bit; signed +/-)
- byte (8 bit; signed +/-)
- complex NOT supported

Numeric data types:
- int64 (64 bit; signed +/-)
- uint64 (64 bit; unsigned)
- uint (32 bit; unsigned)
- ulong (32 bit or 64 bit; unsigned)
- ushort (16 bit; unsigned)
- ubyte (8 bit; unsigned)

Non-numeric data types:
- string
- character
- graphic
- file
- logical
- list

3.4 Variables

Variable names are case sensitive, which means "T2M" is different from "t2m".

To assign a variable:

```plaintext
x  = 1           ; integer
y  = 2.6         ; float
z  = 20.d        ; double
title = "This is the title string" ; string
a  = True        ; logical

a  = (/1, 3, 2, 4/)  ; integer array
b  = (/1, 2.0, 3.0, 4./) ; float array
c  = (/1., 2, 3., 4.0/) * 1d5 ; double array
d  = (/"green", "red")/ ; string array
e  = (/True, False, False, True/) ; logical array
f  = (/ (/1,2/), (/3,6/), (/4,2/) /) ; 2 dimensional array
```

NCL variables may have associated information called metadata (like netCDF metadata). There are three types of variable metadata: attributes, named dimensions, and coordinate variables.

3.4.1 Metadata and Attributes

Metadata is textual or numeric information associated with a variable or file. Typical variable attributes include `units, long_name, standard_name, coordinates, scale_factor, add_offset,`
valid_min, valid_max and axis. However, there may be many other attributes, especially associated with satellite data. Here are some examples:

```
var@units = "degK"
lon@units = "degrees_east"
t@long_name = "Near-Surface Air Temperature"
time@units = "days since 1949-12-01 00:00:00"
temp@FillValue = 1e20
temp@missing_value = 1e20
title = var@long_name
```

To get all attributes of a variable "slp" from a file named "file.nc" you can use the built-in function `getfilevaratts`:

```
fin = addfile("file.nc", "r")
file_atts = getfilevaratts(fin, "slp")
```

To verify whether a specific attribute of a variable exists, use the function `isatt`:

```
if(isatt(slp,"units")) then
  print(slp@units)
end if
```

If missing_value is set, the attribute _FillValue must be the same type and value.

### 3.4.2 Named Dimensions

The number of dimensions (shape) and the number of elements for each dimension (size) are integer values, and a single-dimension variable with one value is called a scalar variable.

By convention, the dimensions are numbered from 0 to n-1 (like in C), where n is the number of dimensions of the referenced variable, and the leftmost dimension is numbered 0. This also applies to arrays.

If a variable has four dimensions, the following statements will attach the names to the dimensions using the ! syntax character:

```
tas!0 = "time"
tas!1 = "height"
tas!2 = "latitude"
tas!3 = "longitude"
```

### 3.4.3 Coordinate Variables

By netCDF definition, "a coordinate variable" is a one-dimensional variable with the same name as a dimension, which names the coordinate values of the dimension. It must not have any missing data (for example, no _FillValue or missing_value attributes) and must be strictly monotonic (values increasing or decreasing).

This is best illustrated via an example. Consider the two-dimensional variable ‘temp(4,5)’, The following two statements name the dimensions "lat" and "lon":

```
temp!0 = "lat" ; left dimension
temp!1 = "lon" ; right dimension
```

Now, the coordinate values can be defined, here, "lon_pts" and "lat_pts":

```
lon_pts = (/ 0., 15., 30., 45., 60. /) ; size 5
lat_pts = (/ 30., 40., 50., 60. /) ; size 4
```
Also, it is suggested that a units attributes be assigned using the @ syntax:

\[
\begin{align*}
\text{lon}_\text{pts}@\text{units} & = \text{"degrees\_east"} \\
\text{lat}_\text{pts}@\text{units} & = \text{"degrees\_north"}
\end{align*}
\]

Lastly, assign the "lon_pts" and "lat_pts" arrays to the named dimensions "lon" and "lat" of the variable temp using the & character. The coordinate variable is now the construct "temp&lon" or "temp&lat", respectively, which points to the arrays with the coordinate values.

\[
\begin{align*}
temp&\text{lon} & = \text{lon}_\text{pts} \\
temp&\text{lat} & = \text{lat}_\text{pts}
\end{align*}
\]

Rules:
- coordinate arrays associated with a coordinate variable must have the same size as the named dimension with which the coordinate variable is associated
- a coordinate variable must have the same name as its corresponding named dimension: eg, lat(lat), p(p), time(time)
- the elements in a coordinate array must be monotonically increasing or decreasing
- any of the numeric data types may be used for values in the coordinate arrays associated with a coordinate variable

### 3.4.4 String References

Sometimes it is impossible to know the names of the attributes and coordinates before writing a script, or these names may vary from variable to variable. To solve this problem, string variables can be used to reference attributes and coordinates by enclosing the variable reference within dollar signs '$'. The following are examples of this:

\[
\begin{align*}
\text{dimnames} & = (/\text{"frtime"}, \text{"lat"}, \text{"lon"}/) \\
\text{attnames} & = (/\text{"_FillValue"}, \text{"long\_name"}/)
\end{align*}
\]

Direct access to the attribute

\[
\text{att0} = \text{temperature}\$\text{attnames}(0)\$
\]

Example of referencing a coordinate variable without knowing the dimension name using the built-in function \textit{iscoord}:

\[
\begin{align*}
\text{if}(\text{iscoord(dimnames(0)))} \\
\quad \text{coord0} = \text{temperature}\&\text{temperature!0} \\
\text{end if}
\end{align*}
\]

### 3.4.5 List Variables (Container Variables)

Variables of type list may be used to contain a heterogeneous suite of NCL variables. Specifically, the variables within a list may be different types, sizes and shapes. An additional feature is that list variables can be treated like stacks or queues.

There are two ways to create a list variable. The first is use the \['/.../'\] syntax as shown in the following example:

\[
\begin{align*}
i & = (/ (1,2,3/), (4,5,6/), (7,8,9/) /) \quad \text{2-dimensional integer array} \\
x & = 5.0 \quad \text{scalar of type float} \\
d & = (/100000.d, 283457.23d/) \quad \text{1-dimensional double array} \\
s & = \text{"abcde"} \quad \text{string} \\
c & = \text{stringtochar("abcde")} \quad \text{character}
\end{align*}
\]


The second is to treat the list variable as a stack. This feature allows variables to be dynamically added to an existing list variable. The following example illustrates the basic approach. The NewList function creates a list and ListPush can be used to dynamically add variables to the list.

```
x = (/1, 2, 3, 4/)
x@attr = "integer array"
y = (/6., 7., 8., 9./)
y@attr = "float array"
s = ("one", "two", "three")
s@attr = "string array"
my_list = NewList("lifo")
ListPush(my_list, x)
ListPush(my_list, y)
ListPush(my_list, s)
```

Note: When using ListPush to add elements to the list, the newly added elements are always at the the head (top) of the list.

The ListCount counts the total elements in a list, and a function named ListIndex can be used to check if an element is in the list.

```
cnt = ListCount(my_list)
print(cnt)
idx = ListIndex(my_list, x)
print(idx)
```

There are two ways to access elements in a list. The first way is to access an element at a certain position via numeric indexing using square brackets. This does not change the element (number) in the list.

```
e = my_list[1]
print(e)
idx = ListIndex(my_list, x)
print(idx)
nx = my_list[idx]
print("ori x = " + x)
print("new x = " + nx)
```

The second is to access the element with the ListPop function. For those familiar with stacks and queues, there is an alias ListDequeue.

```
a = ListPop(my_list)
```

Note: When using ListPop to access an element in the list, the element itself is removed from the list. Also, there is a choice to Pop/Dequeue from head/tail of the list depending on the type of the list. There are two types of list: 1. FIFO (First-In, First-Out, which functions like a queue in computer science); and, 2. LIFO (Last-In, First-Out, which is like a stack).
The list type can be checked using `ListGetType`, and changed with `ListSetType`.

```ncl
lt = ListGetType(my_list)
ListSetType(my_list, "lifo")
ListSetType(my_list, "fifo")
```

### 3.4.6 Variable Assignment

It is important to understand what happens when a variable is used in an assignment statement in NCL. The assignment statement functions differently depending on whether the variable being assigned to is currently undefined or defined. The assignment statement also functions differently depending on whether a variable or a value occupies the right side of the assignment.

When a variable appearing on the left side of an assignment has not be defined or was previously deleted, the assignment statement causes the variable to become defined and the data type and dimensionality of the variable is determined by the right side.

When a variable appearing on the left side is already defined, then the right side must have the same type, or be coercible to the type on the left, and the right side must have the same dimensionality.

#### 3.4.6.1 Value-only assignment

Value-only assignments to variables are fairly straightforward. In essence, value-only assignments mean that the right side of the assignment is not a variable, it is the result of an expression, a value. In this case, if the left side variable reference was not defined prior to the assignment statement, the variable on the left side becomes defined and references the value of the right side. No dimension names, coordinate variables or attributes other than `_FillValue` are assigned. If the right side of the expression does not contain any missing values, then `_FillValue` is not assigned either.

If the left side variable was defined prior to the assignment statement, then the value on the left side is assigned the value of the right side. If the left side is a subscripted reference to a variable, then the right side elements are mapped to the appropriate location in the left side variable. If the left side has meta data, they are left unchanged since only a value is being assigned to the left side variable. When the left side is defined, then the type of the right side and the dimensionality must match. However, there is one exception to the requirement that the dimension sizes of the left side and the right side match, a single scalar value can be assigned to more than one location. Consider the following example:

```ncl
a = (/1,2,3,4,5,6,7,8,9,10/)
a(0:3) = -1
print(a)
```

Variable: a  
Type: integer
Total Size: 40 bytes
10 values
Number of Dimensions: 1
Dimensions and sizes: [10]
Coordinates:
(0) -1
(1) -1
(2) -1
(3) -1
This example demonstrates the value of -1 being assigned to the first four elements of the variable 'a'.

3.4.6.2 Variable-to-variable assignments

During variable-to-variable assignment attributes, coordinate variables and dimension names, in addition to actual multi-dimensional values, are assigned. Before discussing this type of assignment, it is important to note that the array designator characters '(/' and '/)' can be used when assigning one variable to another to force only the right side's value to be assigned to the left side and the right side's attributes, dimensions, and coordinates are ignored. Essentially using the array designator characters forces value-to-variables assignment. The following shows how the array designator characters can be used to do this:

Example of array designator use to force "Value Only" assignment

\[
\text{variable1} = (/ \text{variable2} /)
\]

Variable-to-variable assignment occurs when both the left side and the right side are variables. In this situation, the assignment statement also tries to assign attributes, dimension names, and coordinates of the right side to the left side.

The two simplest cases are:

- The left side is undefined prior to the assignment
- The variable on the left side is not subscripted, meaning the entire variable is being referenced

In both these situations, all of the right side's attributes, coordinates, and dimension names are assigned to the left side. If the left side has the same dimension and coordinate names, then only the coordinate variable is overwritten with the value and attributes of the right side's coordinate variables. However, if the names of the dimension names do not match, a warning message is generated and the names and coordinate variables of the left side are overwritten. As far as attributes go, if the left side has attributes, then the left side's attribute list is merged with that of the right side. If the same attribute name appears on both the left and right sides, the right side's attribute overwrites the left side's. If the types of the attribute values do not match, you could have a type mismatch error.

The following are examples of some variable-to-variable assignment situations:

This first example shows assignment to an undefined variable and then shows the use of the array designator characters '(/' and '/)' to perform a value-only assignment.

Create variable be with values, dimension names, coordinate variables and attributes

\[
\begin{align*}
&b = (/ (/1.0,2.0,3.0/), (/4.0,5.0,6.0/), (/7.0,8.0,9.0/) /) \\
b!0 = "\text{dim0}" \\
b!1 = "\text{dim1}" \\
b@units = "\text{none}"
\end{align*}
\]
b&dim0 = (/0.1,0.2,0.3/)
b&dim1 = (/10,100,1000/)

Variable-to-variable assignment with left side undefined

a = b

Use of array designator characters to assign "Value Only" to undefined left side

c = (/b/)

This print shows that all of the dimension names, attributes, and coordinate variables have been assigned to a.

print(a)
Variable: a
Type: float
Total Size: 36 bytes
9 values
Number of Dimensions: 2
Dimensions and sizes: [dim0 | 3] x [dim1 | 3]
Coordinates:
  dim0: [0.1..0.3]
  dim1: [10..1000]
Number Of Attributes: 1
  units :       none
    (0,0)   1
    (0,1)   2
    (0,2)   3
    (1,0)   4
    (1,1)   5
    (1,2)   6
    (2,0)   7
    (2,1)   8
    (2,2)   9

This print shows that only the values of b were assigned to c.

print(c)
Variable: c
Type: float
Total Size: 36 bytes
9 values
Number of Dimensions: 2
Coordinates:
  (0,0)   1
  (0,1)   2
  (0,2)   3
  (1,0)   4
  (1,1)   5
  (1,2)   6
  (2,0)   7
  (2,1)   8
  (2,2)   9

This second example demonstrates a defined variable being assigned to a defined variable. Note the changes resulting from assignment to the dimension names, attribute values, and coordinate variables in variable a. These assignments that change the left side’s coordinates
and dimension names generate errors. When left and right dimension names are different, NCL considers this an error that the user should be warned about. To avoid these errors you can either make sure before assignment that the left and right sides have the same dimension names, or if you only want to assign a value and don't care about attributes, dimensions, and coordinate variables, you can enclose the right side using '(/' and '/)', which forces NCL to use only the value of the right side.

Define variable a with value, dimension names and attributes. No coordinate variables assigned.

```ncl
a = (/ (/1.1,1.2,1.3/), (/2.1,2.2,2.3/), (/3.1,3.2,3.3/) /)
a!0 = "test0"
a!1 = "test1"
a@units = "Degrees"
a@long_name = "A"
```

Define variable b with value, dimension names, attributes, and coordinate variables.

```ncl
b = (/ (/1.0,2.0,3.0/), (/4.0,5.0,6.0/), (/7.0,8.0,9.0/) /)
b!0 = "dim0"
b!1 = "dim1"
b@units = "none"
b&dim0 = (/10,100,1000/)  
b&dim1 = (/1.0,1.2,1.3/)  
```

Here is the "Variable-to-variable" assignment. The dimension names of a change, and the coordinate variables of b are assigned to a. In addition, the attribute lists are merged.

```ncl
a = b
print(a)
```

Variable: a  
Type: float  
Total Size: 36 bytes   
9 values  
Number of Dimensions: 2  
Dimensions and sizes:   [dim0 | 3] x [dim1 | 3]  
Coordinates:   
dim0: [0.1..0.3]  
dim1: [10..1000]  
Number Of Attributes: 2  
units :   none  
long_name :   A  
(0,0)  1  
(0,1)  2  
(0,2)  3  
(1,0)  4  
(1,1)  5  
(1,2)  6  
(2,0)  7  
(2,1)  8  
(2,2)  9  
```

The remaining case is that when the left side is subscripted, only a portion of the target variable is being assigned to. The simplest case here is when the left-side dimension names are the same and both the left side and right side have coordinate variables for the same dimensions. In this case, assignment occurs for each coordinate variable. The subscripted left-side coordinate variable is assigned the subscripted right side coordinate. The attributes lists for the right side is merged with that of the left side and assigned to the left side variable. The following demonstrates this kind of variable-to-variable assignment.
Define variable a with values, dimension names, attributes and coordinate variables.

\[
a = (/ (1.1,1.2,1.3/), (2.1,2.2,2.3/), (3.1,3.2,3.3/) /)
\]
\[
a!0 = "dim0"
a!1 = "dim1"
a&dim0 = (1.1,2.2,3.3)
a&dim1 = (1.1,0.01,0.001)
a@units = "Degrees"
a@long_name = "A"
\]

Define b with same dimension names, and assign different coordinate variables for dim1.

\[
b = (/ (1.0,2.0,3.0/), (4.0,5.0,6.0/), (7.0,8.0,9.0/) /)
\]
\[
b!0 = "dim0"
b!1 = "dim1"
b@units = "none"
b&dim0 = (1.0,2.0,3.0)
b&dim1 = (10.0,100.0,1000.0)
\]

Here is the example of "Variable to Variable" assignment where the left side is already defined. The coordinate variable for "dim1" is overwritten.

\[
b(0,:) = a(0,:)
\]
\[
\text{print(b)}
\]

Variable: b  
Type: float  
Total Size: 36 bytes  
9 values  
Number of Dimensions: 2  
Dimensions and sizes: [dim0 | 3] x [dim1 | 3]  
Coordinates:  
  dim0: [0.1..0.3]  
  dim1: [0.1..0.001]  
Number Of Attributes: 2  
  units : Degrees  
  long_name : A  
(0,0) 1.1  
(0,1) 1.2  
(0,2) 1.3  
(1,0) 4  
(1,1) 5  
(1,2) 6  
(2,0) 7  
(2,1) 8  
(2,2) 9  

If the left side variable does not have a coordinate variable and the right side does, a coordinate variable is created and assigned. If the left side is subscripted, then the created coordinate variable only has values assigned for the subscripted range, and the rest of the coordinate variable is filled with missing values. The following example illustrates this feature:

Define b with no coordinate variables.

\[
b = (/ 1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0/)
\]
\[
b!0 = "dim0"
\]

Define a with coordinate variables.
Assignment of a to b. Selection of dim0 selects only every other element.

\[ b(:,:2) = a(:,) \]

Print of the coordinate variable "dim0" demonstrates filling of missing value for non-selected element.

```
print(b&dim0)
```

Variable: dim0 (coordinate)
Type: float
Total Size: 36 bytes
  9 values
Number of Dimensions: 1
Dimensions and sizes:   [dim0 | 9]
Coordinates:
Number Of Attributes: 1
  _FillValue : -999
(0)     0.1
(1)     -999
(2)     0.2
(3)     -999
(4)     0.3
(5)     -999
(6)     0.4
(7)     -999
(8)     0.5

The final situation that must be considered when assigning one variable to another is when the dimension names of the left side and the right side do not match. In this case, the assignment overrides the left side's dimension names and coordinate variables, and a warning message is generated. If this is not the desired effect, then the array designator characters '/(' and '/)' can be used to make the assignment a "value-only" assignment.

### 3.4.6.3 Variable reassignment

Reassignment is reusing a variable that has been previously defined (if it is not defined, the reassignment will be simple assignment). With reassignment, a variable appearing on the left side of a reassignment can be defined or not defined, the reassignment statement causes the variable to become defined (or redefined) and the data type and dimensionality of the variable is determined by the right side.

When a variable appearing on the left side is already defined, then it is actually deleted first, and then redefined to the data type and dimensionality of the variable is determined by the right side. The reassignment operator is commonly used in loops where array sizes may change with each iteration.

The reassignment is available in version 6.1.1 or later.

#### 3.4.6.3.1 Value-only reassignment
Value-only reassignments to variables are similar to value-only assignment, except that the left hand side variable could have been in different type and shape.

Consider the following example:

```plaintext
a = (/1,2,3,4,5,6,7,8,9,10/)
; use a for something
a := (/("I", "am"/), (/"string", "now"/)/)
print(a)
```

Variable: a  
Type: string  
Total Size: 32 bytes  
4 values  
Number of Dimensions: 2  
Coordinates:  
(0,0) I  
(0,1) am  
(1,0) string  
(1,1) now  

This example demonstrates the type of a changed from integer to string, and the dimensionality changed as well.

### 3.4.6.3.2 Variable-to-variable reassignments

During variable-to-variable reassignment the left hand side variable is deleted (if it is defined) and then variable-to-variable assignment is performed. The following shows how it works in NCI:

```plaintext
a = new((/20, 20/), string)
; Create variable be with values, dimension names, coordinate variables and attributes
b = (/ (/1.0, 2.0, 3.0/), (/4.0, 5.0, 6.0/), (/7.0, 8.0, 9.0/) /)
b!0 = "dim0"
b!1 = "dim1"
b@units = "none"
b&dim0 = (/1, 2, 3/)  
b&dim1 = (/10, 100, 1000/)  

; Variable-to-variable reassignment with left side defined
a := b  

; Variable-to-variable reassignment with left side undefined

c := b  

print(a)
print(c)
```

Variable: a  
Type: float  
Total Size: 36 bytes  
9 values  
Number of Dimensions: 2  
Dimensions and sizes: [dim0 | 3] x [dim1 | 3]
Arrays

The array processing capabilities of NCL are similar to those of Fortran 90, Matlab, IDL, etc. The arithmetic operators (add, multiply, divide, compare, etc.) apply to arrays as well as scalars. Array operations require that all arrays conform to each other. This means that the arrays must have the same size and shape!

But remember, the subscription of an array or dimension starts, like in C, with the index 0.

```
a = (/ 4, 2, 1, 3 /)        4 elements; index 0-3
b = (/ 0, 1, 1, 0 /)        4 elements; index 0-3

c = a + b    \rightarrow    c = (/ 4, 3, 2, 3 /)
c = a - b    \rightarrow    c = (/ 4, 1, 0, 3 /)
c = a * b    \rightarrow    c = (/ 0, 2, 1, 0 /)
c = a / (b+0.1) \rightarrow  c = (/40,1.818182,0.909090,30 /
```

The leftmost dimension of a multi-dimensional array varies slowest and the rightmost dimension varies fastest (row major). Similar to C, the arrays are stored in the "row x column" format.

```
T(12, 5, 4)       left   dimension index 0 with 12 elements (varying slowest)
middle dimension index 1 with 5 elements
```
To assign a new array you can use the NCL statement `new`:

```
new (array_size, type, [_FillValue])
```

- `n = new(4, integer)` → integer array of size 4
- `m = new(12, float)` → float array of size 12
- `q = new((/2,3,5/), float)` → float array of size 2 x 3 x 5
- `k = new(100, float, 1e20)` → float array of size 100 with `_FillValue=1e20`
- `l = new(100, float, "No_FillValue")` → same as `k` but no `_FillValue`
- `p = new(dimsize(pr), double)` → new array of shape of `pr`, type double
- `cities = new(20, string)` → string array of size 20

The `new` statement will automatically fill all values of the array with the default missing value for that type, unless a missing value is specified, or if the special “No_FillValue” option is set.

### 3.5.1 Standard Subscripts

The indices used in standard subscripting are integers and the general form of a standard subscript is:

```
m:n:i
```

range `m` to `n` in strides of `i`

The following NCL statements illustrate the possibilities for standard subscripts:

#### Example array

```
a = (/ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, /)
```

- `a1 = a` → `new_1` contains 0,1,2,3,4,5,6,7,8,9
- `a2 = a(3)` → `new_2` contains 3
- `a3 = a(1:4)` → `new_3` contains 1,2,3,4
- `a4 = a(0:9:3)` → `new_4` contains 0,3,6,9
- `a5 = a(:5)` → `new_5` contains 0,1,2,3,4,5
- `a6 = a(:7)` → `new_6` contains 7,8,9
- `a7 = a(1:6:-1)` → `new_7` contains 6,5,4,3,2,1
- `a8 = a(8:4)` → `new_8` contains 8,7,6,5,4
  - note: no need to set the stride to -1
- `a9 = a(::-1)` → `new_9` contains 9,8,7,6,5,4,3,2,1,0
  - this is equal to `new_9 = a(9:0)`

#### Example 3-dimensional array

```
T = (12, 100, 120)
```

```
T1 = T(0:11:3, :19, :)
```

defines a 4 x 20 x 120 array

### 3.5.2 Named Subscripts

Named subscripting allows you to reorder arrays, but is only allowed when all dimensions of the array are named dimensions.

Let us use a variable pressure that has two dimensions named "lon" and "lat". The dimension "lat" is of size 21 and the dimension "lon" is of size 40:

```
pres(21,40) → pres(lat, lon)
```
Re-order the dimensions:

\[ p_{\text{reord1}} = \text{pres}(\text{lon}|:\, \text{lat}|:) \rightarrow p_{\text{reord1}}(40,21) \sim p_{\text{reord1}}(\text{lon},\text{lat}) \]

\[ p_{\text{reord2}} = \text{pres}(\text{lon}|19:39, \text{lat}|0:9) \rightarrow \text{define an array 20x10} \]

\[ p_{\text{reord2}}(20,10) \]

The vertical bar | after the dimension name and before the subscript range is required.

### 3.5.3 Coordinate Subscripts

For coordinate subscripting, all of the rules for the standard subscripting apply except for curly brackets {}, which are used to distinguish coordinate subscripts from standard subscripts.

**Example array**

\[ m = (/ -5.0, 10.0, 15.0, 20.0, 25.0, 30.0 /) \]

\[ m!0 = \text{"lat"} \rightarrow \text{name the dimension} \]

\[ m&\text{lat} = m \rightarrow \text{associate the array} \]

\[ mw = m(\{-5.0 : 25.0 : 2\}) \rightarrow \text{contains the values} -5.0, 15.0, 25.0 \]

---

### 3.6 Statements

The fundamental elements of NCL are statements, just like in other scripting or programming languages. Statements are blocks (a group of statements), conditional expressions (if-then, if-then-else), loops (do, do-while), assignments, reassignments, procedures, functions, or graphic statements.

#### 3.6.1 Block
Blocks can be used to bundle many statements into a group of statements as used in functions and procedures. The statements between the `begin` and `end` statement will be executed.

```plaintext
begin
  statement 1
  statement 2
  ....
end
```

The begin and the end statement in the main part of a script are not necessary but sometimes a good practice.

### 3.6.2 If - Statement

```plaintext
if (scalar_logical_expression) then
  [statement(s)]
else
  [statement(s)]
end if
```

There is no "else if", but you can use a trick to get the same effect. Combine the "if" and "else" on one line, as long as you end with an "end if" for each one:

```plaintext
if (scalar_logical_expression_A) then
  [statement(s)]
else if (scalar_logical_expression_B) then
  [statement(s)]
else if (scalar_logical_expression_C) then
  [statement(s)]
else
  [statement(s)]
end if ; C (includes the "else")
end if ; B
end if ; A
```

For example:

```plaintext
x = 7
if ( x .eq. -5 ) then
  print("if-statement 1")
else if ( x .gt. 0 .and. x .lt. 5 ) then
  print("if-statement 2")
else if ( x .lt. 0 ) then
  print("if-statement 3")
else
  print("if-statement 1 else")
end if
end if
end if
```

### 3.6.3 Loops

Loops are useful but may not be efficient and should be minimally used in any interpreted language. To be more efficient, use array arithmetic and/or built-in functions if available. It may be better to write a Fortran or C function and use a wrapper to load it into your NCL script (see chapter 0).
Loop over n-times:

```
   do n=start,end[,stride]
     [statement(s)]
   end do
```

The stride value is not optional if the end value is less than the start value.

Loop while a logical expression is True:

```
   do while (scalar_logical_expression)
     [statement(s)]
   end do
```

Special statements: 
- `break`  exit a loop
- `continue`  skip to next loop iteration

### 3.6.4 Assignment / Reassignment

To assign values to variables, arrays, or attributes, or to assign string values to a variable of type character or named dimensions, the assignment statement, already introduced in chapter 3.4 and 3.5, is used.

Once a variable or array has already been defined and values assigned by using the `=` syntax, it can only be overwritten with the values of the same type and shape. In order to reuse a variable with values of a different data type, size or shape, the variable can be deleted before re-defining it:

```
   var = "This is a string"     ;-- var of type string
   ...
   delete(var)
   var = (/1.0,10.0,15.0/)     ;-- var of type float
```

Or since version 6.1.1, NCL provides the new reassignment syntax `:=` which can change the size and/or shape of a variable. The `:=` operator eliminates the need to delete a variable before reassigning and makes the code a little bit cleaner. Now, the example above can be written as

```
   var = "This is a string"     ;-- var of type string
   var := (/1.0,10.0,15.0/)     ;-- var of type float
```

The reassignment operator is particularly useful in loops when the size and shape of arrays may change with each iteration.

### 3.7 Print Data and Variable Information

NCL provides print procedures to return information to the standard output (stdout).

```
print(variable_or_expression)
   prints the value of a variable or an expression

printVarSummary(data_variable)
   prints summary of a variable’s information

print_table(list)
```
formatted print of all elements from a list

\texttt{printMinMax(data\_variable,0)}
prints the minimum and maximum value of a variable

\texttt{printFileVarSummary(file,varname)}
prints a summary of a file variable’s information

The \texttt{printVarSummary} procedure should be used frequently when debugging code. If questions are sent to \texttt{ncl-talk@ucar.edu}, providing the output from \texttt{printVarSummary} is very useful.

### 3.8 Reserved Keywords

The following keywords can not be used for user defined variables, arrays, lists, functions, or procedures:

<table>
<thead>
<tr>
<th>begin</th>
<th>break</th>
<th>byte</th>
<th>character</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue</td>
<td>create</td>
<td>defaultapp</td>
<td>do</td>
</tr>
<tr>
<td>double</td>
<td>else</td>
<td>end</td>
<td>external</td>
</tr>
<tr>
<td>False</td>
<td>file</td>
<td>float</td>
<td>function</td>
</tr>
<tr>
<td>getvalues</td>
<td>graphic</td>
<td>if</td>
<td>integer</td>
</tr>
<tr>
<td>load</td>
<td>local</td>
<td>logical</td>
<td>long</td>
</tr>
<tr>
<td>new</td>
<td>noparent</td>
<td>numeric</td>
<td>procedure</td>
</tr>
<tr>
<td>quit</td>
<td>quit</td>
<td>QUIT</td>
<td>record</td>
</tr>
<tr>
<td>return</td>
<td>setvalues</td>
<td>short</td>
<td>string</td>
</tr>
<tr>
<td>then</td>
<td>True</td>
<td>undef</td>
<td>while</td>
</tr>
</tbody>
</table>

All built-in function and procedure names are also reserved keywords.
4 File I/O

NCL provides two functions `addfile` and `addfiles` to open and access files for reading, writing, and creating. An important feature of these functions is that when variables are imported, they are placed into a consistent variable model regardless of the original source file format (netCDF-3/4, HDF-4/5, GRIB-1/2). This common structure facilitates data processing and the creation of generic functions. Specifically, the variable model is based on the netCDF variable model. It is a data structure containing the array values and all associated meta data. Many NCL functions are 'meta data aware'.

4.1 addfile

The function `addfile` can open existing data files written in supported file formats or create new data files in one of the supported data file formats.

```plaintext
f = addfile(filename, status)
```

- `f`: reference to the data file (type 'file')
- `filename`: file name with full or relative path
- `status`: read-only `r`, read-write `w` (overwrite), create `c`

If you use the status "c" to open a file, it will be created if it doesn’t already exist. If the file already exists, an error message is returned to stdout. Previously created files should be removed before using `addfile` to create new data files with the same name, e.g.

```plaintext
system("rm -f new_data.nc")
```

```plaintext
f = addfile("data.nc", "r")
```

opens the file data.nc in the current directory for reading.
g = addfile("dataT.nc", "w")
    opens the file dataT.nc in the current directory for reading and writing

filename = "/tmp/dataT_new.nc"
system("rm -f " + filename)
g = addfile(filename, "c")
    creates a new file dataT_new.nc in the /tmp directory

Once a file is opened, you can get more information about it using the following functions:

- getvaratts: return a list of global attributes on the file
- getfiledimsizes: return a list of dimension sizes
- getfilevaratts: return a list of attributes associated with a given variable on the file
- getfilevardims: return a list of dimension names associated with a given variable on the file
- getfilevardimsizes: return a list of dimension sizes associated with a given variable on the file
- getfilevarnames: return a list of variable names on the file
- getfilevartypes: return a list of variable types for the given variable(s) on the file

To read a variable from any supported file format with all metadata information included:

    fin = addfile("data.nc", "r") ; .grb, .hdf, .h5, .hdfeos, .shp
    t   = fin->T

To strip off the metadata, enclose the file variable reference with ‘(/…/)’. Only, the special _FillValue attribute will be carried over.

    fin = addfile("data.nc", "r")
    t   = (/ fin->T /)

Keep in mind that every calculation made on the data will strip of the meta data because NCL is not able to know which meta data is still correct.

### 4.2 addfiles

For the comparison of different simulations or the joint analysis of ensemble simulations, it is very useful to access multiple files at once. The function addfiles can open multiple existing data files or create multiple new data files using one of the supported file formats.

    file_list = addfiles(list_of_files, status)

- file_list: list of references to the multiple data files
- list_of_files: a 1D array of strings containing the full or relative path of the data files
- status: r = read-only
          w = read-write
          c = create

However, there are some differences from the function addfile:

1. addfiles returns a variable of type list. The returned variable contains references pointing to each file. This yields a special type of list: the file list type:

    files = systemfunc("ls *.nc") ; NetCDF file names
f = addfiles(files, "r") ; data type 'list'

2. To import a variable into memory, two options are provided via the ListSetType procedure: "cat" and "join". The "cat" option is the default. It will concatenate a file variable which spans multiple files. Accessing elements of a list variable requires use of the [...] syntax. Consider three files containing the variable 'TEMP' with sizes TEMP(1,10,20,30), TEMP(22,10,20,30) and TEMP(4,10,20,30), respectively. Then the following will import the variable with all meta data:

\[
t = f[:]->TEMP ; \text{default is 'cat'}
\]
\[
\text{printVarSummary}(t) ; t(27,10,20,30)
\]

Use of the "join" option requires that the leftmost dimension be the same across all files. Consider X(12,72,144) on each of three files, then

\[
\text{ListSetType}(f, \text{"join")}
\]
\[
x = f[:]->X ; x(3,12,72,144)
\]

The "join" option results in an additional dimension being added.

3. Access to the opened files can be done more specific. For example, you can read data from every second file of an input file list. First, you have to specify from which files the variable should be read. For example, to access the variable T (first time step, first level) in all files opened, T must exist in all files of the file list \( f \) and have the same shape:

\[
T_{all} = f[:]->T(0,0,:,:)
\]

To get every second file of a list of 12 files:

\[
T_{sec} = f[0:12:2]->T(0,0,:,:)
\]

To get the variable names off the list of files, use first file in list:

\[
\text{varnames} = \text{getfilevarnames}(f[0])
\]

4. It is only possible to write to an individual file of the file list \( f \):

\[
f = \text{addfiles}(files,"w")
\]

4.3 Assign a Variable from a File Variable

\[
f = \text{addfile}(\text{"dataT.nc"}, \text{"r")}
\]
\[
T = f->T \rightarrow \text{set T to file variable T}
\]

or

\[
T = f->T(0,:,:) \rightarrow \text{set T to file variable T, only first time step}
\]

If the name of a file variable, attribute, or coordinate variable includes hyphens or blanks, NCL will exit with a fatal error. To avoid this, the name can first be stored in a string variable. If this is enclosed by '$' characters, it can then be referenced.

\[
tnam = \text{"RCP85 MPI-ESM tas"}
\]
\[
\text{var} = f->\$\text{tnam}\$
\]
\[
\text{lon} = x&\text{"lon-1"}
\]
4.4 Read ASCII File

An ASCII file contains integers or floating point data values in ASCII format. Sometimes it can also contain a header of type string. NCL provide the function asciiread to do this for the user. More examples can be found at http://ncl.ucar.edu/Applications/read_ascii.shtml

Here is a short example how to read an ASCII file containing 14 lines with a single value each line.

Example data asc1.txt:

<table>
<thead>
<tr>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1965</td>
</tr>
<tr>
<td>1970</td>
</tr>
<tr>
<td>1971</td>
</tr>
<tr>
<td>1973</td>
</tr>
<tr>
<td>1978</td>
</tr>
<tr>
<td>1980</td>
</tr>
<tr>
<td>1982</td>
</tr>
<tr>
<td>1985</td>
</tr>
<tr>
<td>1990</td>
</tr>
<tr>
<td>2000</td>
</tr>
<tr>
<td>2001</td>
</tr>
<tr>
<td>2002</td>
</tr>
<tr>
<td>2005</td>
</tr>
<tr>
<td>2008</td>
</tr>
</tbody>
</table>

The NCL script NUG_read_ASCII_1.ncl reads the data in a 1D array:

```ncl
begin
; Read data into a one-dimensional int array of length 14:

  data = asciiread("asc1.txt",14,"integer")

  npts = dimsizes(data)   ; should be 14
  print("Number of values: "+npts)
  print(data)             ; print the values

end
```

If you don't know how many data values you have, you can use the special "-1" value for the dimension size. When you use -1, data values will be read from left-to-right, top-to-bottom, into a 1D array, until there are no values left.

```ncl
begin
; Read data into a one-dimensional array of unknown length:

  data = asciiread("asc1.txt",-1,"integer")

  npts = dimsizes(data)   ; should be 14
  print("Number of values: "+npts)
  print(data)             ; print the values

end
```

Both versions will return the following lines:

```
(0) Number of values: 14
Variable: data
Type: integer
Total Size: 56 bytes
```
ASCII files containing a header line followed by multiple columns of integer and floating point data are more common and the next example will show how to handle it.

Example data asc2.txt:

<table>
<thead>
<tr>
<th>Year</th>
<th>Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>310</td>
</tr>
<tr>
<td>1750</td>
<td>791</td>
</tr>
<tr>
<td>1800</td>
<td>978</td>
</tr>
<tr>
<td>1850</td>
<td>1262</td>
</tr>
<tr>
<td>1900</td>
<td>1650</td>
</tr>
<tr>
<td>1950</td>
<td>2518.6</td>
</tr>
<tr>
<td>1955</td>
<td>2755.8</td>
</tr>
</tbody>
</table>

In this case, the header line will be ignored because it doesn't contain any numerical data.

NUG_read_ASCII_2.ncl:

```
begin
  ; To read this data into a 2D array dimensioned 17 x 2
  ; (17 rows by 2 columns), use:
  data = asciiread("asc2.txt",(/17,2/),"float")
  print(data) ; Print the values
end
```

Output to the terminal:

<table>
<thead>
<tr>
<th>Variable: data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: float</td>
</tr>
<tr>
<td>Total Size: 136 bytes</td>
</tr>
<tr>
<td>Number of Dimensions: 2</td>
</tr>
<tr>
<td>Dimensions and sizes: [17] x [2]</td>
</tr>
<tr>
<td>Coordinates:</td>
</tr>
<tr>
<td>Number Of Attributes: 1</td>
</tr>
<tr>
<td>_FillValue : 9.96921e+36</td>
</tr>
<tr>
<td>(0,0) 1000</td>
</tr>
</tbody>
</table>
Another case is a file containing several columns of integer, float and string data. See example data file asc3.txt:

```
200306130209 0.38 25.28 10088 233.95 6 92 9.99 99999.0 0.0 -9.99 167.9 p p p 1782 BOS ATL 3
200306130209 0.38 25.28 10088 233.95 6 92 9.99 99999.0 0.0 -9.99 167.9 p p p 1782 ORD ATL 3
200306132341 -45.10 168.70 914 279.35 4 272 9.99 9.99 1.1 0.0 0.01 -99.9 p p 4552 DTW MSP 3
200306122341 -45.10 168.70 914 279.35 4 272 9.99 9.99 1.1 0.0 0.01 -99.9 p p 4552 SDF MHR 3
```

The first column contains date values, which we want to parse into separate year, month, day, hour, and minute arrays. We also want to read the third-from-the-last-column, which are the station names.

NUG_read_ASCII_3.ncl:

```
begin
    fname  = "asc3.txt"
data   =  asciiread(fname,-1,"string")
year   =  tofloat(str_get_cols(data, 1,4))
month  =  tofloat(str_get_cols(data,5,6))
day    =  tofloat(str_get_cols(data,7,8))
hour   =  tofloat(str_get_cols(data,9,10))
minute =  tofloat(str_get_cols(data,11,12))

    ;-- read the station name (field 18)
    sta = str_get_cols(data,99,101);-- you must know the digit column numbers
    sta = str_get_field(data,18," ");-- you must know the field number

    print("Year: "+year+"  month: "+month+"  day: "+day+"  hour: "+hour+" 
    "+minute);
    print("Data:");
    print(""+sta)
end
```

Output to the terminal:

```
(0) Year: 30 month: 61 day: 30 hour: 20 minute: 9
(1) Year: 30 month: 61 day: 30 hour: 20 minute: 9
(2) Year: 30 month: 61 day: 22 hour: 34 minute: 1
(3) Year: 30 month: 61 day: 22 hour: 34 minute: 1
(0) Data:
(0) BOS
(1) ORD
(2) DTW
(3) SDF
```
4.5 Read Excel CSV Data Files

An Excel sheet can be exported as a CSV text file with delimiters. These CSV files can be read and plotted with NCL using the built-in functions asciiread, str_fields_count and str_get_field.

Given example data file Test_6h.csv contains a value for 0h,6h,12h,18h per line:

<table>
<thead>
<tr>
<th>Time (h)</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.00</td>
<td>2.40</td>
<td>2.60</td>
<td>3.00</td>
</tr>
<tr>
<td>6</td>
<td>3.50</td>
<td>3.10</td>
<td>3.70</td>
<td>4.10</td>
</tr>
<tr>
<td>12</td>
<td>5.10</td>
<td>4.80</td>
<td>5.30</td>
<td>6.05</td>
</tr>
<tr>
<td>18</td>
<td>8.20</td>
<td>8.90</td>
<td>10.10</td>
<td>10.50</td>
</tr>
</tbody>
</table>

The NCL script to read and plot the data from each line with different colors on a XY-plot:

NUG_read_CSV_1.ncl

```ncl
load "&NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_code.ncl"
load "&NCARG_ROOT/lib/ncarg/nclex/gsun/gsn_csm.ncl"
begin
  diri = "./"
  fili = "Test_6h.csv"

  ;-- read in file as array of strings so we can parse each line
  delim = ";"
  data   = asciiread(diri+fili, -1, "string")
  scount = str_fields_count(data(0),delim)

  ;-- read 6h values
  nl    = dimsizes(data)
  lines = new(nl, "string")
  cols  = new(scount, "string")
  val   = new((/nl,scount/), float)
  do i=0,nl-1
    do j=1,scount
      value = tofloat(str_get_field(data(i),j,delim))
      val(i,j-1) = value
    end do
  end do
  print("Val: " + val)

  ;-- 4 timesteps, interval 6h
  x = ispan(0,18,6)

  wks = gsn_open_wks("png","plot_read_CSV_1")

  res@gsnDraw = False
  res@gsnFrame = False
  res@tiMainString = "NCL Doc Examples: Read CSV data (delimiter = ;)"
  res@xyLineThicknessF = 5
  res@trYMinF = 0.0
  res@trYMaxF = 12.0
  res@trXMinF = 0
  res@trXMaxF = 18
  res@xyDashPattern = 0          ;-- make lines all solid
  res@xyLineColor = (/"blue", "red", "green", "black", "orange")
```
The next example will show you how to read a more complex CSV data file.

Example data file: multiple_columns.csv

```
19747,01/20/2014, 9:31:38, 43.194, 27.971, 75, 1011, 64.23, 2.291, 0.969, 22.0,
0, 395.20, 369.01, 361.25, 734.49, NaN, NaN, NaN, NaN, NaN,
1.0710, 1.0215, 0.4918, 0.6072, 0.308, 0.278, 0.265, 0.208, 0.144, NaN

19747,01/20/2014, 10:18:05, 43.188, 28.062, 75, 1008, 63.28, 2.216, 0.969, 21.2,
0, 414.54, 389.68, 377.82, 770.75, 1216.23, NaN, NaN, NaN, NaN, NaN,
1.0638, 1.0314, 0.4902, 0.6337, 0.313, 0.272, 0.259, 0.195, 0.147, NaN

19747,01/20/2014, 10:32:54, 43.208, 7.012, 75, 1008, 65.32, 2.384, 0.969, 19.8,
0, 388.66, 364.70, 355.26, 731.68, 1171.65, NaN, NaN, NaN, NaN, NaN,
1.0657, 1.0266, 0.4855, 0.6245, 0.287, 0.263, 0.257, 0.200, 0.152, NaN

19747,01/20/2014, 10:52:49, 43.236, 28.219, 75, 1007, 63.83, 2.258, 0.969, 19.0,
0, 413.64, 385.64, 375.76, 751.42, 1184.56, NaN, NaN, NaN, NaN, NaN,
1.0726, 1.0263, 0.5001, 0.6343, 0.300, 0.267, 0.254, 0.201, 0.156, NaN
```

Example script to read the CSV file and plot it: NUG_read_CSV_2.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/contrib/time_axis_labels.ncl"

begin
  ;-- input file name
  fname = "multiple_columns.csv"

  ;-- used delimiter in file for columns or strings (date)
  delim1 = "/"
  delim2 = ":" 
  delim3 = ":" 

  ;-- read in file as array of strings so we can parse each line
  data  = asciiread(fname, -1, "string")
  scount = str_fields_count(data(0),delim1) ;-- get number of columns

  ;-- read variable AOD (500)
  var = tofloat(str_get_field(data,29,delim1))

  ;-- read date and split it to year, month and day
  dat  = str_get_field(data,2,delim1)
  year = toint(str_get_field(dat,3,delim2))
  month = toint(str_get_field(dat,1,delim2))
```

```
plot = gsn_csm_xy(wks, x, val, res)
draw(plot)
frame(wks)
end
```
day = toint(str_get_field(dat,2,delim2))

;-- read time and split it to hour, minutes and seconds
tim = str_get_field(data,3,delim1)
hour = toint(str_get_field(tim,1,delim3))
minute = toint(str_get_field(tim,2,delim3))
second = toint(str_get_field(tim,3,delim3))

;-- convert the UT-referenced time to a mixed Julian/Gregorian time
units = "hours since 2000-01-01 00:00:00"
time2 = cd_inv_calendar(year,month,day,hour,minute,second,units, 0)

;-- define the workstation (plot type and name)
wks = gsn_open_wks("png","plot_NUG_read_CSV_2")

;-- set resources
res = True
res@gsnMaximize = True ;-- maximize the plot
res@xyMarkLineModes = "Markers" ;-- use markers instead of lines
res@xyMarkers = 5 ;-- type of marker (cross)
res@xyMarkerColor = "red" ;-- marker color
res@xyMarkerSizeF = 0.007 ;-- marker size
res@vpXF = 0.25 ;-- viewport x-position
res@vpYF = 0.6 ;-- viewport y-position
res@vpWidthF = 0.7 ;-- viewport width
res@vpHeightF = 0.37 ;-- viewport height
res@tiMainString = "AOD (500nm)" ;-- title string
res@tiYAxisString = "AOD" ;-- y-axis string
res@trYMinF = 0.0 ;-- y-axis minimum value
res@trYMaxF = 0.6 ;-- y-axis maximum value
res@tmXLabelFontHeightF = 0.01 ;-- x-axis label font size
res@tmXLabelJust = "CenterRight" ;-- x-axis label justification
res@tmXLabelDeltaF = 1.0 ;-- moves x-axis labels downward
res@tmXLabelAngleF = 50. ;-- rotate x-axis labels
res@tmYRon = False ;-- no tick marks on right y-axis
res@tmXRon = False ;-- no tick marks on top x-axis

;-- set the time format for res
restime = True
restime@tmFormat = "%d %C %h %m"
time_axis_labels(time2,res,restime)

;-- create the plot
plot = gsn_csm_xy(wks, time2, var, res)
end

4.6 Read Binary File

A binary file is a file whose contents are to be interpreted as a sequence of bits, rather than characters. There are different flavours of binary files. A 'flat' binary file is a sequence of bits with no ancillary information about the file contents. All records are the same size. This type
is created and read by C programs. Fortran creates and reads flat binary files only when in direct-access mode. By default, Fortran creates another type of binary file which can contain variable-length records. This is called a sequential-access binary file. In a sequential-access binary file, record length is embedded prior to each record.

Each type of binary data has its own read function. You must know how your data was written.

**Direct Access:**
```
data = fbindirread (path,rec,dim,type)
```

**Sequential Access:**
```
data = fbinrecread (path,rec,dim,type)
```

**Cray (C block IO write):**
```
data = cbinread (path,dim,type)
```

**Cray sequential:**
```
data = craybinrecread (path,rec,dim,type)
```

Binary files created on one machine may not be directly portable to another machine. The terms used to describe the way numbers are stored are big-endian and little-endian. A big-endian representation means the most significant byte is on the left while a little-endian representation means the most significant byte is on the right.

NCL allows users to read files created using, say, big-endian machines on little-endian machines and vice versa via the `setfileoption` procedure. This procedure also allows the data to be written according to a specific byte order.

```
setfileoption("bin","ReadByteOrder","LittleEndian")
v = cbinread("data.littleEndian.bin",-1,"float")
```

```
setfileoption("bin","WriteByteOrder","BigEndian")
cbinwrite("data.bigEndian.bin",v)
```

To read the example big-endian binary file topo.bin: NUG_read_Binary_1.ncl

```
begin

;-- path and file name
diri = "./"
fname = "topo.bin"

;-- set byte order
setfileoption("bin","ReadByteOrder","BigEndian")

;-- read binary file
topo = fbindirread(diri+fname,0,(/293,343/),"float")

;-- set some attributes
topo@units = "m"
topo@long_name = "topography"

;-- print information
printVarSummary(topo)
print("Minimum value: "+min(topo))
print("Maximum value: "+max(topo))

end
```

To read a GrADS binary data file you need the information from the descriptor file (.ctl), like the dimensions of time (TDEF), latitude (YDEF) and longitude(XDEF).

```
ps_grads_model.ctl:
DSET   ^ps_grads_model.dat
```
OPTIONS little_endian
UNDEF -2.56E33
TITLE 5 Days of Sample Model Output
XDEF 73 LINEAR 0.0 5.0
YDEF 46 LINEAR -90.0 4.0
ZDEF 1 linear 1 1
TDEF 5 LINEAR 02JAN1987 1DY
VARS 1
PS 0 99 Surface Pressure
ENDVARS

Example script to read the ps_grads_model.dat binary file: NUG_read_Binary_GrADS.ncl

```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"
begin
    diri = "/"
    fili = "ps_grads_model.dat"
    read data
    setfileoption("bin", "ReadByteOrder", "LittleEndian")
    ps = fbindirread (diri+fili, 0, (/ 5, 46, 73 /), "float")
    print ("-- read GrADS binary data -- done")
    ps@long_name = "Surface Pressure"
    ps@units = "Pa"
    printVarSummary(ps)
    print ("min(ps)="+min(ps))
    print ("max(ps)="+max(ps))
end
```

4.7 Write ASCII File

NCL has five main functions for writing data to an ASCII file:

- **write_table**: writes formatted, mixed-type data with a single format statement.
- **write_matrix**: writes nicely-formatted 2D arrays of integer, float, or double precision data.
- **asciiwrite**: an older and rather limited function that writes one value per line. This is useful for outputting a one-dimensional time series.
- **sprintf**: converts floats or doubles into formatted strings.
- **printfi**: converts integers into formatted strings.

The first example shows how to write the data values one value per line to the output file.
NUG_write_ASCII_1.ncl:

```ncl
begin
    ;-- Generate a dummy 2x3x4 array
    data = random_uniform(-5,5,(/2,3,4/))
    ;-- Write it to a file
    asciiwrite("file1.txt",data)
end
```

The first lines of file1.txt should look like this:

```
-1.762895
-1.75608
-0.06612359
-2.112701
-1.469934
-3.460391
0.6621115
```

The second example shows how to write a mix of data (string, float, and integer) using `write_table`.

NUG_write_ASCII_2.ncl:

```ncl
load "$/NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
    ;---Generate some dummy integer and float data.
    npts = 100
    i    = ispan(1,npts,1)
    j    = generate_unique_indices(npts)
    k    = generate_unique_indices(npts)
    x    = random_uniform(-10,10,npts)
    y    = random_uniform(0,1000.,npts)
    write_table("file2.txt","w","/[j,x,i,y,k/], \n        "string_%03i %8.2f %4.0i %8.1f     string_%03i")
end
```

The first lines of file2.txt should look like:

```
string_031  -0.11  1  269.1  string_040
string_074  5.17  2  798.3  string_018
string_015  8.73  3  408.6  string_082
string_037  2.14  4  546.1  string_054
string_016  -1.39  5  653.1  string_017
```

The third example writes the same ASCII file as the previous example, except it uses a combination of `sprintf` and `sprinti` to format the data, and then `asciiwrite` to write the strings to a file.

This method can be "very" slow if you have a lot of data to format.

NUG_write_ASCII_3.ncl:
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"

begin
    -- Generate some dummy integer and float data.
    npts = 100
    i = ispan(1,npts,1)
    j = generate_unique_indices(npts)
    k = generate_unique_indices(npts)
    x = random_uniform(-10,10,npts)
    y = random_uniform(0,1000.,npts)

    lines = "string_" + sprinti("%03i", j) + " " + sprintf("%8.2f",x) + " " + sprinti("%4.0i", i) + " " + sprintf("%8.1f",y) + " " + "string_" + sprinti("%03i", k)

    ;--Write to a file
    asciiwrite("file3.txt",lines)
end

The first lines of file3.txt should look like:

| string_031 | -0.11 | 1 | 269.1 | string_040 |
| string_074 | 5.17  | 2 | 798.3 | string_018 |
| string_015 | 8.73  | 3 | 408.6 | string_082 |
| string_037 | 2.14  | 4 | 546.1 | string_054 |
| string_016 | -1.39 | 5 | 653.1 | string_017 |

The fourth example shows how to write a 3D array to a file, by writing 2D blocks of data at a time. In general, we do not recommend writing large arrays to an ASCII file as this is very inefficient.

We have to use write_table because it is the only NCL function that allows you to append data to an existing ASCII file. This function requires the data be in a list object, so we push each column of a 2D subsection of data to the list object, and then write out that 2D list of data to the file using the "append" option.

This method gets a little kludgy because you have to use a unique variable for every "push" onto the list object. We do this by using unique_string to generate a unique string, and attributes.

NUG_write_ASCII_4.ncl:

begin
    ;-- Generate a dummy 3D array
    nx = 200 ; # of blocks
    ny = 100 ; # of rows
    nz = 10 ; # of columns
    data = random_uniform(-5,5,(/nx,ny,nz/))

    ; Use this to create "nice" numbers for debug purposes.
    ; This makes it easier to see how the data file is being written.

    ; data = reshape(conform_dims(/(/200,ny*nz/),ispan(1,ny*nz,1),1),/(/nx,ny,nz/)) + conform_dims(/(/nx,ny,nz/),ispan(1,nx,1),0)

    /1000.
```ncl
;-- Remove file just in case
filename = "file4.txt"
system("rm -f " + filename)

;-- Write a header to the file
header = "This ASCII file contains " + nx + " blocks of " + ny + " x " + nz + " arrays"
write_table(filename, "w", ["header"]) ; Use "w" to create file

;-- Create row format string. It will have "%7.3f" repeated nz times
fmt_str = "%s" + str_concat(conform_dims(nz,"%8.3f",-1))

;-- Loop through each column of each block and write the column of data to a List object. We can then use write_table to append a whole block of formatted data to an ASCII file.
row_labels = "Row " + sprinti("%3i",ispan(1,ny,1))
dtmp = True ; Variable to hold temporary attributes
do i=0,nx-1
   ;-- Write out the block number. Use "a" to append to existing file.
   slist = ["Block " + (i+1) + " of " + nx/]
   write_table(filename, "a", slist, "%s")

   ;-- Create a new List object for this block of data
   dlist = NewList("lifo")

   ;-- Loop in reverse order so items are written in correct order
   do j=nz-1,0,1
      ListPush(dlist,(/data(i,:,j)/))
   end do

   ;-- Push array of row headers onto list object
   str = unique_string("test")
dtmp@$str$ = row_labels
ListPush(dlist,dtmp@$str$)

   ;-- Append this List of data to file.
   write_table(filename, "a", dlist, fmt_str)
end do
end

The first lines of file4.txt should look like:

```

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.763</td>
<td>1.756</td>
<td>0.066</td>
<td>2.113</td>
<td>1.470</td>
<td>3.460</td>
<td>0.662</td>
<td>3.207</td>
<td>-1.745</td>
<td>-1.599</td>
</tr>
<tr>
<td>3.952</td>
<td>1.634</td>
<td>2.150</td>
<td>0.034</td>
<td>2.735</td>
<td>-4.788</td>
<td>-4.630</td>
<td>-2.094</td>
<td>-4.139</td>
<td>2.476</td>
</tr>
<tr>
<td>1.406</td>
<td>2.919</td>
<td>4.202</td>
<td>3.521</td>
<td>2.082</td>
<td>-1.046</td>
<td>2.644</td>
<td>1.459</td>
<td>-0.269</td>
<td>3.595</td>
</tr>
<tr>
<td>0.025</td>
<td>4.976</td>
<td>4.233</td>
<td>2.525</td>
<td>-1.127</td>
<td>-0.600</td>
<td>3.011</td>
<td>-4.735</td>
<td>-3.468</td>
<td>-4.152</td>
</tr>
</tbody>
</table>
```

The fifth example shows how to write nicely-formatted two-dimensional arrays of numeric data to an ASCII file using write_matrix. This script creates three ASCII files, one with float data, one with integer data, and one with double data.

NUG_write_ASCII_5.ncl:

```ncl
begin
```
;-- create random data
nrows = 5
ncols = 7
ave  = 0.0
std  = 5.0
xf   = random_normal (ave, std, (/nrows,ncols/)) ; float
xi   = round (xf, 3) ; integer
xd   = todouble(xf)

;-- set 2 missing values
xf@_FillValue = 1e36
xf(1,1) = xf@_FillValue
xf(3,3) = xf@_FillValue

;-- set resources for write_matrix
option   = True
option@row = False
option@tspace = 0

;-- output type float
option@fout = "file5.f.txt"
option@title = "floating point data with two missing values"
write_matrix (xf, "7f7.2", option)

;-- output type integer
option@fout = "file5.i.txt"
option@title = "integer data with no missing values"
write_matrix (xi, "7i7", option)

;-- output type double precision
option@fout = "file5.d.txt"
option@title = "double precision data with no missing values"
write_matrix (xd, "7f7.2", option)
end

The file5.f.txt should look like (containing 2 missing values):

floating point data with two missing values
4.35  4.36  9.73  4.91  1.77 -0.63 -4.29
4.39***** -5.84  4.59  3.68 -14.12  0.07
0.27  3.77  0.89 -3.09  5.08 -2.51  5.85
-3.35 -1.66  8.46*****  0.14  1.76  0.87
-6.90  4.06 10.39  4.56 -5.63 -1.43  0.65

The file5.i.txt should look like (no missing values):

integer data with no missing values
 4  4  10  5  2  -1  -4
 4  5  -6  5  4  -14  0
 0  4  1  -3  5  -3  6
-3  -2  8  8  0  2  1
-7  4  10  5  -6  -1  9

The file5.d.txt should look like (no missing values):

double precision data with no missing values
4.35  4.36  9.73  4.91  1.77 -0.63 -4.29
4.39  4.66 -5.84  4.59  3.68 -14.12  0.07
0.27  3.77  0.89 -3.09  5.08 -2.51  5.85
-3.35 -1.66  8.46  7.55  0.14  1.76  0.87
### 4.8 Write CSV File

The next example will show you how to write a 2-dimensional array printed as comma-separated-values (CSV) to an ASCII output file.

The example 2 of the write_table function manual page on [http://ncl.ucar.edu/Document/Functions/Built-in/write_table.shtml](http://ncl.ucar.edu/Document/Functions/Built-in/write_table.shtml) is one approach. Another approach is to do your own line formatting.

```ncl
outfile = "out.txt"
x = (/ (/ 4.35, 4.36, 9.73, 4.91 /),
     (/ 4.39, 4.66, -5.84, 4.59 /),
     (/ 0.27, 3.77, 0.89, -3.09 /) /)
dimx  = dimsizes (x)
nrows = dimx(0)                        ;-- ncols = dimx(1)
lines = new (nrows, string)
do i = 0, nrows-1
   lines(i) = str_concat (sprintf "%7.2f,", x(i,:))
end do
asciiwrite (outfile, lines)
```

To conserve space, you can remove all spaces between numbers by changing the format string to "%0.2f,". This is standard CSV format as used by spreadsheet software. For single spaces between numbers, use " %0.2f,".

### 4.9 Write Binary File

NCL has four main functions for writing data to binary files:

- `fbinrecwrite`: multiple unformatted sequential records
- `fbindirwrite`: specified direct record
- `fbinwrite`: single unformatted sequential record
- `cbinwrite`: mimics a C block IO write

This example shows how to read data from a NetCDF file and write it to a Fortran unformatted binary file using `fbinrecwrite`.

**NUG_write_Binary_1.ncl**:

```
begin

   ;-- set path and file name
diri = "/"
fill = "rectilinear_grid_2D.nc"
file_out = "example.bin"

   ;-- read NetCDF file and variable
fi = addfile(diri+fill,"r")
t = fi->tsurf

   if (isfilepresent(file_out)) then
```

---

<table>
<thead>
<tr>
<th>-6.90</th>
<th>4.06</th>
<th>10.39</th>
<th>4.56</th>
<th>-5.63</th>
<th>-1.43</th>
<th>8.65</th>
</tr>
</thead>
</table>
The next example shows how to use fbindirwrite to write three variables to the same file, and then fbindirread to read them back in. NUG_write_Binary_2.ncl:

begin

  ;-- Create some dummy arrays.
  nlev = 10
  nlat = 64
  nlon = 128
  t1 = random_uniform(0,100,(/nlev,nlat,nlon/))
  t2 = random_uniform(0,100,(/nlev,nlat,nlon/))
  t3 = random_uniform(0,100,(/nlev,nlat,nlon/))

  ;-- Remove file just in case
  filename = "dummy_file.bin"
  system("rm -f " + filename)

  ;-- Write first variable to file
  fbindirwrite(filename,t1)
  system("ls -l " + filename)

  ;-- Append second variable to file
  fbindirwrite(filename,t2)
  system("ls -l " + filename)

  ;-- Append third variable to file
  fbindirwrite(filename,t3)
  system("ls -l " + filename)

  ;-- Read data back in and compare the diffs. Should be equal to 0.
  t1r = fbindirread(filename,0,(/nlev,nlat,nlon/),"float")
  t2r = fbindirread(filename,1,(/nlev,nlat,nlon/),"float")
  t3r = fbindirread(filename,2,(/nlev,nlat,nlon/),"float")

  print("")
  print("If the result below is 0/0 then everything is fine!")
  print(min(t1r-t1) + "/" + max(t1r-t1))
  print(min(t2r-t2) + "/" + max(t2r-t2))
  print(min(t3r-t3) + "/" + max(t3r-t3))
end
Output to terminal:

```
-rw-r--r-- 1 k204045 staff 327680 9 Jan 18:23 dummy_file.bin
-rw-r--r-- 1 k204045 staff 655360 9 Jan 18:23 dummy_file.bin
-rw-r--r-- 1 k204045 staff 983040 9 Jan 18:23 dummy_file.bin
(0) (0) If the result below is 0/0 then everything is fine!
(0) 0/0
(0) 0/0
(0) 0/0
```

## 4.10 Write NetCDF File

NetCDF is a self-describing, machine-independent data format which is very common in climate science. There are four types of NetCDF files currently supported by NCL: classic, 64-bit offset, netCDF-4 classic, and netCDF-4.

If you have a newer version of NetCDF installed on your system, you can use "ncdump -k" to determine the type of NetCDF file you have. It will output either "classic", "64-bit offset", or "netCDF-4" for the various types of NetCDF files.

Some of the examples below make use of the setfileoption procedure for setting options before you write to a file. The first example shows the simple but inefficient way and the second is the more efficient and faster way.

### 1st Method – NUG_write_netCDF_1.ncl:

```ncl
begin
  diri = "/
  fili = "rectilinear_grid_2D.nc"
  outfile = "t_in_Celsius_1.nc"

  if (isfilepresent(outfile)) then
    system("rm -rf "+outfile)    ;-- make sure that file does not exist
  end if

  fin  = addfile(diri+fili,"r")    ;-- open data file
  fout = addfile(outfile,"c")     ;-- create new file (netcdf 3)

  filedimdef(fout,"time",-1,True) ;-- make time and UNLIMITED dimension

  tK    = fin->tsurf     ;-- get variable
  tC    = tK            ;-- copy variable and its related
                         ;dimensions and attributes
  tC    = tK - 273.15   ;-- convert from Kelvin to Celsius
  tC@units = "degC"     ;-- define new units

  fout->tC = tC         ;-- write variable to new netCDF file
end
```

Terminal output of ‘ncdump -h t_in_Celsius_1.nc’:

```
netcdf t_in_Celsius_1 {
    dimensions:
      time = UNLIMITED ; // (40 currently)
      lat = 96 ;
      lon = 192 ;
```
variables:

float tC(time, lat, lon);
    tC:grid_type = "gaussian";
    tC:table = 128;
    tC:code = 169;
    tC:units = "degC";
    tC:long_name = "surface temperature";

double time(time);
    time:calendar = "standard";
    time:units = "hours since 2001-01-01 00:00:00";
    time:standard_name = "time";

double lat(lat);
    lat:axis = "y";
    lat:units = "degrees_north";
    lat:long_name = "latitude";
    lat:standard_name = "latitude";

double lon(lon);
    lon:axis = "x";
    lon:units = "degrees_east";
    lon:long_name = "longitude";
    lon:standard_name = "longitude";

2nd Method – NUG_write_netCDF_2.ncl:

begin
diri = "/"
fili = "rectilinear_grid_2D.nc"
outfile = "t_in_Celsius_2.nc"

if (isfilepresent(outfile)) then
    system("rm -rf "+outfile) ;-- make sure that file does not exist
end if

;-- open data file
    fin = addfile(diri+filii,"r") ;-- open data file

;-- get variable t and its dimensions and dimension sizes
    tK = fin->tsurf ;-- get variable
    time = fin->time ;-- get dimension time
    lat = fin->lat ;-- get dimension lat
    lon = fin->lon ;-- get dimension lon

    ntim = dimsizes(time) ;-- get dimension sizes of time
    nlat = dimsizes(lat) ;-- get dimension sizes of lat
    nlon = dimsizes(lon) ;-- get dimension sizes of lon

    printVarSummary (tK) ;-- print variable information

;-- convert variable t: Kelvin to Celsius
    tC = tK ;-- copy variable and its dimensions
        ;and attributes
    tC = tK - 273.15 ;-- convert from Kelvin to Celsius
    tC@units = "degC" ;-- define new units

;-- create new netCDF file
    fout = addfile(outfile,"c")

;-- begin output file settings
    setfileoption(fout,"DefineMode",True) ;-- explicitly declare file
; definition mode

fAtt = True ; assign file attributes
fAtt@title = "NCL Efficient Approach to netCDF Creation"
fAtt@source_file = fili
fAtt@Conventions = "CF"
fAtt@creation_date = systemfunc ("date")
fAtt@history = "NCL script: NUG_write_netCDF_2.ncl"
fAtt@comment = "Convert variable tsurf from degrees Kelvin to degrees Celsius"

fileattd(fout,fAtt) ; copy file attributes

; predefine the coordinate variables and their dimensionality
dimNames = ("time", "lat", "lon")
dimSizes = (-1, nlat, nlon)
dimUnlim = (True, False, False)
filedimdef(fout,dimNames,dimSizes,dimUnlim)

; predefine the dimensionality of the variables to be written out
filevardef(fout, "time" , typeof(time) , getvardims(time))
filevardef(fout, "lat" , typeof(lat) , getvardims(lat))
filevardef(fout, "lon" , typeof(lon) , getvardims(lon))
filevardef(fout, "tC" , typeof(tK) , getvardims(tK))

; copy attributes associated with each variable to the file
filevarattd(fout,"time",time) ; copy time attributes
filevarattd(fout,"lat",lat) ; copy lat attributes
filevarattd(fout,"lon",lon) ; copy lon attributes
filevarattd(fout,"tC", tC) ; copy tC attributes

; explicitly exit file definition mode (not required)
setfileoption(fout,"DefineMode",False)

; output only the data values since the dimensionality and such
; have been predefined. The "(/.../)") syntax tells NCL to only
; output the data values to the predefined locations on the file.
fout->time = (/time/) ; write time to new netCDF file
fout->lat = (/lat/) ; write lat to new netCDF file
fout->lon = (/lon/) ; write lon to new netCDF file
fout->tC = (/tC/) ; write variable to new netCDF file
end

Terminal output of 'ncdump -h t_in_Celsius_2.nc':

```
netcdf t_in_Celsius_2 {
  dimensions:
    time = UNLIMITED ; // (40 currently)
    lat = 96 ;
    lon = 192 ;
  variables:
    double time(time) ;
      time:standard_name = "time" ;
      time:units = "hours since 2001-01-01 00:00:00" ;
      time:calendar = "standard" ;
    double lat(lat) ;
      lat:standard_name = "latitude" ;
      lat:long_name = "latitude" ;
      lat:units = "degrees_north" ;
      lat:axis = "Y" ;
    double lon(lon) ;
      lon:standard_name = "longitude" ;
```
lon:long_name = "longitude";
lon:units = "degrees_east";
lon:axis = "X";
float tC(time, lat, lon);
tC:grid_type = "gaussian";
tC:table = 128;
tC:code = 169;
tC:units = "degC";
tC:long_name = "surface temperature";

// global attributes:
:comment = "Convert variable tsurf from degrees Kelvin to degrees Celsius";
:history = "NCL script: NUG_write_netCDF_2.ncl";
:creation_date = "Fr 9 Jan 2015 17:38:36 CET";
:Conventions = "CF";
:source_file = "rectilinear_grid_2D.nc";
:title = "NCL Efficient Approach to netCDF Creation";
5 Tools

The following shell commands are included in the NCL software distribution:

- **ncl_filedump** generates an ASCII representation of supported files (netCDF, HDF, GRIB1, GRIB2, shapefile) on the standard output. It is similar to the netCDF program called `ncdump -h`.

- **ncl_convert2nc** converts GRIB1, GRIB2, shapefile, HDF, or HDF-EOS files to NetCDF files

- **ng4ex** script for generating hundreds of available C, Fortran, and NCL object-oriented examples

- **WRAPIT** wraps Fortran 77 or 90 code so you can call it directly from NCL
6 Data Processing

Prior to visualization, it is quite often necessary to apply additional processing steps to the original data. For processing NetCDF data, you can use separate tools such as CDO or NCO. These tools are designed to efficiently perform specific tasks and, depending on the task, can be more efficient than NCL. Another approach is to use NCL’s internal processing capabilities directly.

Here, we will only show a few examples of data processing with NCL and CDO:

1) Compute yearly means from monthly data
2) Compute the time average at each grid point
3) Compute the standard deviation of a dimension
4) Compute the area average
5) Compute Linear regression
6) Compute Running mean

The most important rule in data processing is to

look at your data!

You can avoid many errors and warnings if you are familiar with your data. Keep that in mind.

To get an overview of the contents of your file, type the following on the shell command line:

```
ncfFiledump <your_filename>
```

or

```
ncdump -h <your_filename>
```

Assume an input file is of format netCDF and contains the variable ‘tas’ on a lat/lon grid with 120 timesteps (10 years monthly data):

```
tas(120,194,201)
```

6.1 NCL - Compute yearly means from monthly data

To calculate the yearly (annual) mean, use the function `month_to_annual` which can be found in the contributed.ncl library:

```
ret_array = month_to_annual(array_mon, option)
```

array_mon: an array containing monthly data
option: option=0 compute the unweighted sum of 12 values
option=1 divide the unweighted sum by 12 to get the annual mean value(s)
ret_array: returned array with "time" dimension is decimated by a factor of 12

Example code:

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

tas_ym = month_to_annual(tas,1); -- tas_ym(10,194,201)

6.2 NCL - Compute the time average at each grid point

To compute the area mean without weights of the variable tas, use the functions \texttt{dim\_avg\_n} or \texttt{dim\_avg\_n\_Wrap}:

Example code: assume \texttt{tas(time,lat,lon)}

Without transferring the metadata -

tasAvg = dim\_avg\_n(dim\_avg\_n(tas,2),1)

\begin{itemize}
\item Results 1D array of area mean values
\item First average over \texttt{lon} (= index 2) and then \texttt{lat} (= index 1)
\end{itemize}

To preserve the metadata of \texttt{tas}, use the \texttt{\_Wrap} version of this function

\begin{verbatim}
tasAvg = dim\_avg\_n\_Wrap(dim\_avg\_n\_Wrap(tas,2),1);
\end{verbatim}

\begin{verbatim}
\textend{verbatim}

6.3 NCL - Compute the standard deviation of a dimension using \texttt{dim\_stddev\_n}

\begin{verbatim}
ret_var = dim\_stddev\_n(var)
\end{verbatim}

\begin{verbatim}
\textend{verbatim}

\begin{verbatim}
var: variable of numerical type and any dimension
ret_var: variable of same type as \texttt{var}, dimension rank is reduced by one
\end{verbatim}

If you want to retain the metadata, use \texttt{dim\_stddev\_Wrap}. Use the function \texttt{dim\_stddev\_n\_Wrap} if no reordering is needed and the wanted dimension can be specified directly.

To calculate the temporal standard deviation at each grid point:

Example code:

\begin{verbatim}
tasStdT = dim\_stddev\_n(tas,0); -- tasStdLon(194,201),
\end{verbatim}

\begin{verbatim}
\textend{verbatim}

\begin{verbatim}
tasStdT = dim\_stddev\_Wrap(tas(lat|:,lon|:,time|:)); -- tasStdLon(194,201),
\end{verbatim}

\begin{verbatim}
\textend{verbatim}

\begin{verbatim}
tasStdT = dim\_stddev\_n\_Wrap(tas,0); -- tasStdLon(194,201),
\end{verbatim}

\begin{verbatim}
\textend{verbatim}

6.4 NCL - Compute the weighted area average

\begin{verbatim}
rad = get_d2r(lat); -- float if "lat" is float,
\end{verbatim}

\begin{verbatim}
\textend{verbatim}
weights = cos(lat*rad) ;-- cosine weights
area_avg = wgt_areaave(var,weights,1.0,1)

6.5 NCL – Compute Linear Regression

Linear regression is an approach for modeling the relationship between a dependent variable \( y \) and one or more explanatory or independent variables denoted \( x \).

The NCL built-in function `regline` computes the information needed to construct a regression line: regression coefficient (trend, slope,...) and the average of the \( x \) and \( y \) values. `regline` is designed to work with one-dimensional \( x \) and \( y \) arrays. Missing data are allowed. If the regression coefficients for multi-dimensional arrays are needed, use `regCoef`.

Note: Version 6.2.0 has an improved version named `regline_stats` that also returns an ANOVA table.

Example script: NUG_statistics_linear_regression.ncl

```ncl
begin
  diri = "/"
  fili = "tas_mod1_rcp85_rectilin_grid_2D.nc"

  f = addfile(diri+fili,"r")
  var = f->tas(:,0,:,:)
  time = f->time ;-- get time values

  ;-- convert a mixed Julian/Gregorian date to a UT-referenced date
  utc_date = cd_calendar(time, 0)
  year = tointeger(utc_date(:,0))
  month = tointeger(utc_date(:,1))
  day = tointeger(utc_date(:,2))
  date_str = sprinti("%0.4i-%0.2i-%0.2i", year, month, day)

  ;------------------------------------------------------------
  ;-- y = mx+b
  ;-- m is the slope: rc returned from regline
  ;-- b is the y intercept: rc@yave attribute of rc returned from regline
  ;------------------------------------------------------------
  x = time ;-- get time values
  y = dim_avg_n_Wrap(dim_avg_n_Wrap(var,1),1) ;-- timeserie

  rc = regline(x, y) ;-- calculate the linear regression
  y_stat2 = rc*(x-rc@xave) + rc@yave ;-- rc@yave = y intercept; rc = slope

  ;-- open workstation
  wks = gsn_open_wks("png","plot_stat_linear_regression")

  ;-- set resources
  res = True
  res@gsnDraw = False ;-- don't draw plot yet
  res@gsnFrame = False ;-- don't advance frame
  res@tiMainString = "NCL Doc Example:  Linear Regression"

  res@vpHeightF = 0.4 ;-- viewport height
  res@vpWidthF = 0.8 ;-- viewport width
  res@vpXF = 0.125 ;-- viewport x start pos
```

51
res@trXMinF = min(x) ;-- x-axis min value
res@trXMaxF = max(x) ;-- x-axis max value
res@trYMinF = min(y) ;-- y-axis min value
res@trYMaxF = max(y) ;-- y-axis max value

res@tmXBMode = "Explicit" ;-- use explicit x-axis values
res@tmXBValues = time(::4) ;-- x-axis values (every 4th)
res@tmXBLabels = date_str(::4) ;-- x-axis labels (every 4th)
res@tmXBLabelJust = "CenterRight" ;-- x-axis label justification
res@tmXBLableDeltaF = 0.15 ;-- move x-axis labels down
res@tmXBLabelAngleF = 45. ;-- rotate x-axis labels
res@tmXBLabelFontHeightF = 0.012 ;-- decrease x-axis label font size

res@xyMarkLineModes = "Lines" ;-- line mode

res@xyLineColor = "black" ;-- line color black
plot1 = gsn_csm_xy(wks, x, y, res)

res@xyLineColor = "red" ;-- line color red
plot2 = gsn_csm_xy(wks, x, y_stat2, res)
overlay(plot1,plot2) ;-- overlay plot2 on plot1

;-- draw the plot
draw(plot1)
frame(wks)
end

6.6 NCL – Compute Running Mean

In statistics, a running mean (moving average) is a series of calculated averages of different subsets of the full data set to analyse data points.

Example script: NUG_statistics_running_mean.ncl

begin

  diri = "./
  fili = "tas_mod1_rcp85_rectilin_grid_2D.nc"
  f = addfile(diri+fili,"r")
  var = f->tas(:,0,:,:)
  time = f->time ;-- get time values

  end
;-- convert a mixed Julian/Gregorian date to a UT-referenced date
utc_date = cd_calendar(time, 0)
year     = tointeger(utc_date(:,0))
month    = tointeger(utc_date(:,1))
day      = tointeger(utc_date(:,2))
date_str = sprinti("%0.4i-", year)+sprinti("%0.2i-", month)+ \
           sprinti("%0.2i", day)

x         = time           ;-- get time values
y         = dim_avg_n.Wrap(dim_avg_n.Wrap(var,1),1) ;-- timeserie

;-- calculate the running mean
y_rave    = runave_n.Wrap(y,10,0) ;-- 10 time steps are included in
                                    ;-- running average

;-- open workstation
wks = gsn_open_wks("png","plot_stat_running_mean")

;-- set resources
res = True
res@gsnDraw = False ;-- don't draw plot yet
res@gsnFrame = False ;-- don't advance frame
res@tiMainString = "NCL Doc Example:  Running Mean"
res@vpHeightF = 0.4      ;-- viewport height
res@vpWidthF  = 0.78     ;-- viewport width
res@vpXF      = 0.14     ;-- viewport x start pos
res@trXMinF   = min(x)   ;-- x-axis min value
res@trXMaxF   = max(x)   ;-- x-axis max value
res@trYMinF   = min(y)   ;-- y-axis min value
res@trYMaxF   = max(y)   ;-- y-axis max value
res@tmXBMode  = "Explicit" ;-- use explicit x-axis values
res@tmXBValues = time(:,4) ;-- x-axis values (every 4th)
res@tmXLabels = date_str(:,4) ;-- x-axis labels (every 4th)
res@tmXLabelJust = "CenterRight" ;-- x-axis label justification
res@tmXLabelDeltaF = 0.15 ;-- move x-axis labels down
res@tmXLabelAngleF = 45. ;-- rotate x-axis labels
res@tmXLabelFontHeightF = 0.012 ;-- decrease font size

;-- create timeserie plot
res@xyLineColor = "black" ;-- line color black
plot1 = gsn_csm_xy(wks, x, y, res)

;-- create linear regression plot
res@xyLineColor = "blue" ;-- line color blue
plot2 = gsn_csm_xy(wks, x, y_rave, res)
overlay(plot1,plot2) ;-- overlay plot2 on plot1

;-- draw the plot
draw(plot1)
frame(wks)
end
6.7 CDO - Compute annual means from monthly data

The CDOs must be run on the command line or within a shell and in most cases need an input file (stream) and output file (stream). In the next few cases we will write out a netCDF file (‘-f nc’) with a relative time axis (‘-r’).

To calculate the yearly mean use the operator yearmean:

\[ \text{cdo} \quad -r \quad -f \text{ nc yearmean } \ <\text{input file}> \ <\text{output file}> \]

6.8 CDO - Compute the time average at each grid point

To compute the time average of all time steps included in the NetCDF file use the operator timmean:

\[ \text{cdo} \quad -r \quad \text{timmean } \ <\text{input file}> \ <\text{output file}> \]

6.9 CDO - Compute the temporal standard deviation at each grid point using timstd

\[ \text{cdo} \quad -r \quad \text{timstd } \ <\text{input file}> \ <\text{output file}> \]

6.10 CDO - Compute the area average

To calculate the area average for each time step use fldmean:

\[ \text{cdo} \quad \text{fldmean } \ <\text{input file}> \ <\text{output file}> \]

6.11 CDO - Compute Linear Regression

To calculate a timeserie use the fldmean operator and compute the linear regression by piping the output stream to the detrend operator. The output file contains the values, which have to be subtracted from the original timeseries values to be plotted:
cdo -sub <input file> -detrend -fldmean <input file> \ 
<output file>

6.12 CDO - Compute Running Mean

To calculate the running mean (moving average) values over 10 timesteps, use the runmean operator:

```
cdo runmean,10 <input file> <output file>
```

6.13 CDO – Select Variables

For large data sets it would be more efficient to do the computations first, e.g. selecting the wanted variable, do the computation, and save the file in the appropriate file format.

To select a variable use the selvar parameter with a single variable name or list of variables separated by a colon:

select 1 variable
```
cdo -r -f nc selvar,temp <input file> <output file>
```

select 3 variables
```
cdo -r -f nc selvar,temp,u10,v10 <input file> <output file>
```

6.14 CDO – Piping commands

The operations can be piped within cdo, e.g. select variable temp, only the first 10 time steps and do an area average:

```
cdo -r fldmean -seltimestep,1/10 -selvar,temp <input file> \ 
<output file>
```

Note that the second and third option starts with a dash!

The computations will be done right to left. At first, select the variable temp and pipe the output stream to seltime step operator which selects the first 10 time steps, and then further pipe the output stream to the fldmean operator to compute the field mean, which will be stored in the output file outfile.

See also: [https://code.zmaw.de/projects/cdo/embedded/1.6.4/cdo.pdf](https://code.zmaw.de/projects/cdo/embedded/1.6.4/cdo.pdf)
7 Advanced NCL features

Now, let us start going deeper into some features of NCL.

7.1 Masking

To mask data elements means that you mark parts of the data, i.e. specific grid points, which should not be drawn. With NCL’s mask function you can mark data elements with _FillValue (out-dated: missing_value) in order to exclude them from being plotted. Furthermore, you can use so-called “graphical resources” to control the masking or change the order of the plotting elements.

See also the applications “mask” page, http://www.ncl.ucar.edu/Application/mask.shtml

Assume var is a two dimensional data array and lsm is a two dimensional array which contains the land sea mask array; the value 1 represents land and the value 0 represents water.

- Using the mask function (see also NCL example mask_1.ncl)
  
  - xLand = mask(var, lsm, 1) → equal to var where lsm=1
  - xOcean = mask(var, lsm, 0) → equal to var where lsm=0

- Using the where function and the _FillValue (old netCDF notation is missing_value) from the data
  
  - xLand = where(lsm .eq. 1, T, T@_FillValue) → if lsm equal 1 then plot T else set to _FillValue
  - xOcean = where(lsm .eq. 0, T, T@_FillValue) → if lsm equal 0 then plot T else set to _FillValue

- Graphical resources (see also NCL example mask_4.ncl)
  
  - change the order of the plotting elements
7.2 Date Conversion

The dimension data of time can be stored in different ways: relative and absolute values. To convert the time values, NCL provides a set of calendar functions, e.g. `cd_calendar`, `cd_convert`, `cd_inv_calendar` and `cd_string`.

More ‘Date Routines’ : [http://www.ncl.ucar.edu/Document/Functions/date.shtml](http://www.ncl.ucar.edu/Document/Functions/date.shtml)

- `cd_calendar` converts a mixed Julian/Gregorian date to a UT-referenced date
- `cd_string` converts time values into nicely formatted strings
- `cd_convert` converts a variable from one set of units to another
- `cd_inv_calendar` converts a mixed Julian/Gregorian date to a UT-referenced date

NUG_date_format.ncl:

```
...
    time = var&time
    timax = dimsizes(time) - 1
    ...
    ;-- convert the time proleptic_gregorian calendar to UTC date
    utc_date = cd_calendar(time, 0)
    ;-- set date variable names
    year = toint(utc_date(:,0))
    month = toint(utc_date(:,1))
    day = toint(utc_date(:,2))
    hour = toint(utc_date(:,3))
    minute = toint(utc_date(:,4))
    second = utc_date(:,5)
    ;-- write date as string (DD.MM.YYYY)
    date_str_i = sprinti("%0.2i",day) + "." + sprinti("%0.2i",month) + "." + sprinti("%0.4i",year)
    ...
    ;-- create the time strings, plot every second axis annotation
    incr = 2
    labels = (/ date_str_i(0::incr) /)
    ...
    ;-- set the resources
    res@trXMinF = time(0) ;- time minimum on axis
    res@trXMaxF = time(timax) ;- time maximum on axis
    res@tmXBMode = "Explicit" ;- explicit time setting
    res@tmXBValues = var&time(:,::incr) ;- axis ticks position
    res@tmXBLables = labels ;- labels on axis ticks
    ...
    plot = gsn_csm_xy(wks, var&time, var, res)
    ...
```

NCL Doc Example: date format
7.3 String Operations

Sometimes you may need to manipulate a string in order to get rid of leading blanks, convert from lower case to upper case, or just select parts of a text line. NCL offers a variety of string manipulation functions.

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

- Convert upper to lower case, lower to upper case and capitalize a text string using the functions `str_lower`, `str_upper` and `str_capitalize`

  ```
  str = "HELLO WORLD"
  strlower = str_lower(string) → "hello world"
  
  str = "good morning"
  strupper = str_upper(string) → "GOOD MORNING"
  
  str = "good morning to everybody"
  strcapital = str_capitalize(string) → "Good Morning To Everybody"
  ```

- Strip off blanks: leading, ending, all, or replace multiple blanks or TABs with a single blank using the functions: `str_left_strip`, `str_right_strip`, `str_strip`, `str_squeeze`

  ```
  str = "This is the title"
  strnew = str_left_strip(str) → "This is the title"
  strnew = str_right_strip(str) → "This is the title"
  strnew = str_strip(str) → "This is the title"
  strnew = str_squeeze(str) → "This is the title"
  ```

- Count and select fields of a text string: `str_fields_count` and `str_get_field`

  ```
  str = "This is a string"
  nf = str_fields_count(str, " ") → nf = 4
  str = "tas_domain_model_ensemble_version_starttime-endtime.nc"
  delim = ".-".
  nf = str_fields_count(str, delim) → nf = 8
  str = "20130101000000 53.33 10.0 278.32 t2m"
  field_1 = str_get_field(str,1," ") → field_1 = "20130101000000"
  field_5 = str_get_field(str,5," ") → field_1 = "t2m"
  ```

- Split strings with a given delimiter or split a CSV string using `str_split` or `str_split_csv`

  ```
  str = "Using NCL makes a lot of fun"
  strlist = str_split(str, " ")
  qc = str_get_dq()
  print(qc + strlist + qc); the quote character
  (0) "Using"
  (1) "NCL"
  (2) "makes"
  ```
(3) "a"
(4) "lot"
(5) "of"
(6) "fun"

CSV (Comma-separated values) is an output format, e.g. used by EXCEL to export the data of a table to an ASCII file. It can contain consecutive delimiters, because there are no values available, the missing value will be inserted.

```plaintext
str = "20130101,000000,53.33,10.0,278.32,t2m,,,
     Near-Surface Air Temperature"

str_new = str_split_csv(str, ",", 0)
print(str_new)
```

- **Variable**: str_new
- **Type**: string
- **Total Size**: 72 bytes
- **9 values**
- **Number of Dimensions**: 2
- **Dimensions and sizes**: [1] x [9]
- **Coordinates**:
  - _FillValue_: missing
  - (0,0) 20130101
  - (0,1) 000000
  - (0,2) 53.33
  - (0,3) 10.0
  - (0,4) 278.32
  - (0,5) t2m
  - (0,6) missing
  - (0,7) missing
  - (0,8) Near-Surface Air Temperature

Note that the returned values of the str_* functions are of type string. To use the string content as a numeric value, it must be converted using `tofloat`, `todouble`, `toint`, `toshort` or `tolong`:

```plaintext
str = "20130101000000 53.33 10.0 278.32 t2m"
val = tofloat(str_get_field(str,4," "))
     → val = 278.32 of type float

idate = toint(str_get_field(str,1," "))
     → idate = 20130101000000 of type integer
```

### 7.4 System Calls

The `system` procedure and `systemfunc` function are used to interact with the underlying operating system. The difference is that `system` is used to pass a command to the system to perform an action while `systemfunc` returns information to the NCL environment. Other system calls include: `status_exit`, `getenv`, `sleep` and `get_cpu_time`. More details are at:

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

- **To execute a shell command**:

  ```plaintext
  system("rm -f tmp.asc")
  system("export NCARG_COLORMAPS=$HOME/NCL/Colors")
  ```
• To execute a shell command and return the output of the call:

```plaintext
file_list = systemfunc("ls t2m_*.nc")
date_string = systemfunc("date")
```

• Exit the NCL script returning an integer value as status code:

```plaintext
fin = addfile("tas.nc", "r")
if(ismissing(fin)) then
    status_exit(99)
end if
```

• Get the content of a shell environment variable:

```plaintext
ret = getenv("SHELL")
```

This will return a string with the full path to the shell interpreter.

### 7.5 User-defined Functions and Procedures

Generally, functions return values and procedures perform tasks. The structure of a procedure or function in NCL is similar to Fortran and C. Procedures and functions can be written directly as the uppermost part of your script code, or they can separately be saved in an external file, which can be loaded by the "load" or "loadscript" command. It might even be useful to collect multiple functions and procedures often needed and save them as your personal external library files, which can be made available within other NCL scripts with the "load" command.

For example:

```plaintext
;-- load pre-defined functions and procedures
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"  # GSN code lib
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"  # GSN csm lib
load "$HOME/NCL/mylib/funs_and_procs.ncl"               # User defined lib
```

### 7.5.1 Procedures

Generally, procedures are used to perform a task (e.g., draw a plot).

**General structure:**

```plaintext
undef ("procedure_name")       ;-- optional
procedure procedure_name(declaration_list)
local local_variables          ;-- optional
begin
    statements
end
```
You can, for example, save the user-defined procedure to a new file `my_library.ncl` in the directory `/your_home/NCL/lib` and load it into your NCL scripts.

```
/your_home/NCL/lib/my_library.ncl:

undef("wallClockElapsedTime_german")
procedure wallClockElapsedTime_german(tstart:string, title:string)
begin
    tend = systemfunc("date +%s")
    tend_i = toint(tend)
    tstart_i = toint(tstart)
    elapsed_time = tend_i - tstart_i
    NL = str_get_nl()
    print (NL + "------> Wall clock elapsed time - " + title + ": " + elapsed_time + " s" + NL)
end

To use this new procedure, load the file to your NCL script:

load "$HOME/NCL/lib/my_library.ncl"
```

Other example procedures:

```
;-- convK2C: convert data from Kelvin to Celsius
undef("convK2C")
procedure convK2C(var)
begin
    var = var - 273.15
    var@units = "C"
end

;-- convK2F: convert data from Kelvin to Fahrenheit
undef("convK2F")
procedure convK2F(var)
begin
    var = ((var-273.15)*9/5)+32
    var@units = "F"
end
```

### 7.5.2 Functions

Functions are used to perform one or more tasks and return values to the parent NCL script.

**General structure:**

```
undef ("function_name") ;-- optional
procedure function_name(declaration_list) ;-- optional
local local_variables
begin
    statements
    return(return_value)
end
```

To compute the value of \( \pi \), you can write a short function:

```
undef("my_pi")
function my_pi()
local lpi
```
begin
  lpi = 4*atan(1) ;-- 4d*atan(1) for double precision
  return(lpi)
end
...
x=my_pi() ;-- Note: NCL has a "get_pi" function
print(x)

→
  Variable: x
  Type: float
  Total Size: 4 bytes
    1 values
  Number of Dimensions: 1
  Dimensions and sizes: [1]
  Coordinates:
    (0) 3.141593

NCL can return multiple variables contained within a variable of type list.

If a function is to return multiple variables (eg: ni,nj and nk) they can be returned as a variable of type list. The can be created using the [/ .../] syntax For example:

    undef ret_mulvar(val1,val2)
    function ret_mulvar(val1,val2)
      local ni,nj,nk
      begin
        ni = val1 + val2
        nj = val1 - val2
        nk = val1 * val2
        return([/ni,nj,nk/]) ;-- return values as list variable
      end
    end

Use the function as customary. For example:

    comp = ret_mulvar(5,2)
    vadd = comp(0) ;-- store first list element to vadd
    vsub = comp(1) ;-- store second list element to vsub
    vmul = comp(2) ;-- store third list element to vmul
    delete(comp) ;-- not needed any longer

7.6 Handling Metadata

Metadata is defined as "data describing data". When a data file is opened with addfile, variables and the corresponding metadata included in the file are accessed. As an example, information on the variables such as their dimensions, the grid used, the variable names, the units, the file history, etc. are typical metadata for the files we work with.

Some processing steps can cause the loss of metadata or invalidate it, so it is important that the user take care of it. Some NCL functions have a corresponding "_Wrap" function, like "dim_avg_n_Wrap", which means that the metadata will be copied (retained) to the new computed variable.

Sometimes it will be necessary to set, overwrite, or delete variable attributes.

- Set an attribute
\[ m = \sqrt{u^2 + v^2} \]

To add new metadata information to the variable:

```python
m@standard_name = "magnitude of wind velocity"
m@long_name = "magnitude of wind velocity"
m@units = "m s^{-1}"
```

- **Overwrite an attribute**

```python
m@long_name = "magnitude (\sqrt{u^2+v^2})"
```

- **Delete an attribute**

```python
delete(m@long_name)
delete(m@units)
```
Example:

```fortran
if(isatt(var,"original_name")) then
  delete(var@original_name)
end if
```

A newly-assigned variable has no named dimensions or attributes unless the user defines them manually:

```fortran
latitudes = u&lat ; allocate latitudes array
longitudes = u&lon ; allocate longitudes array

m = sqrt(u^2+v^2) ; assign and compute m
m!0 = "lat" ; define named dimension lat
m!1 = "lon" ; define named dimension lon
m&lat = latitudes ; allocate latitude values
m&lon = longitudes ; allocate longitude values
m&lat@units = "degrees_north" ; set lat units
m&lon@units = "degrees_east" ; set lon units
m@standard_name = "magnitude_of_wind_velocity"
m@long_name = "magnitude (sqrt(u^2+v^2)"
m@units = "m s-1"
```

A shortcut is assigning the new variable by copying a variable u with the same dimensions first. The new variable m has the same dimensions and attributes as u:

```fortran
m = u ; assign: copy u to m
m = sqrt(u^2+v^2) ; compute m

m@standard_name = "magnitude_of_wind_velocity"
m@long_name = "magnitude (sqrt(u^2+v^2)"
```
8 Introduction to NCL Graphics

In this chapter we will present an overview of the NCL’s graphics capabilities. An introduction of the following plot types is given: simple XY-Plots, multiple time series, contour plots, displaying lines or colored fields, paneled plots, overlays, map projections, and the use of shapefiles. The complete variety of NCL’s plotting features can hardly be described here, but we’ve highlighted some of the most common and interesting ones.

An NCL script for plotting data commonly can be sectioned into different parts:

0) Load and/or define functions and procedures
1) Open data file/files
2) Define the variable/variables
3) Open the plot output
4) Define the plot resources
5) Plot

NCL provides a huge number of plot resources to control and manage the layout of the plot, such as annotations, projection, plot type, colors, labelbars, multiple plots on one page, plot output format and name, and further more. For most of the settings reasonable defaults are set, such as a default colortable.

CAUTION: starting with NCL 6.1.0 the default color table had been changed. This may cause differently colored results for the examples compared with older versions of NCL.


Example scripts and data are available on the NCL examples page:

[http://www.ncl.ucar.edu/Applications/](http://www.ncl.ucar.edu/Applications/)

If you don’t know how to write the plotting script, but know exactly what it should look like, it would be very useful to visit the following NCL web pages:

[http://www.ncl.ucar.edu/Training/Workshops/Scripts/](http://www.ncl.ucar.edu/Training/Workshops/Scripts/)
[http://www.ncl.ucar.edu/Applications/Templates/](http://www.ncl.ucar.edu/Applications/Templates/)

On these pages there are many plot examples supplied with their appropriate scripts for downloading.

We recommend writing and saving the scripts to a file, rather than typing commands and resources again and again.

8.1 NCL Graphics – in 5 steps

To create a very simple plot with default settings, only 5 steps are necessary. Here is a contour plot example to show how easy NCL scripting can be.

A short template file is provided in the directory NCL_Doc/scripts: NUG_template.ncl.

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

begin
end
```
The first two lines tell NCL to load the graphic libraries `gsn_code.ncl` and `gsn_csm.ncl`. These library files contain high level procedures and functions for contours, vectors, legends, labelbars and so on, which are not yet included directly in NCL. Additional library files like `contributed.ncl` or `shea_util.ncl` which are also used in the examples, contain procedures and functions for averaging, converting and other helpful tasks. You may take a look into these files to copy and modify procedures and functions for your own purpose, but we strongly recommend using your own function or procedure names.

Since NCL release (6.2.0) these libraries (`gsn_code.ncl`, `gsn_csm.ncl` and `contributed.ncl`) are not longer required because they will be loaded automatically. But for backward compatibility it is a good idea to load the libraries in your scripts.

To get your own simple plot, copy the template file and edit it as shown in the box below. Don’t forget to save it in a directory where you have write permission, e.g. `$HOME/my_simple_plot.ncl`.

1. Open a data file
2. Set variable references (e.g. first time step)
3. Open the plot output (X11 → output to screen)
4. Set plot resources (Detailed list of all available resources: http://www.ncl.ucar.edu/Document/Graphics/Resources/list_alpha_res.shtml)
5. Plot

NUG_plot_in_5_steps.ncl:

```
load "\$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "\$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
  f= addfile("\$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc", "r")
  var = f->tsurf(0,:,:)
  wks = gsn_open_wks("x11","plot_in_5_steps")
  res = True
  res@tiMainString = "NCL plot in 5 steps"
  plot = gsn_csm_contour_map(wks, var, res)
end
```

![NCL plot in 5 steps](image_url)
8.2 The Viewport

The viewport of NCL in which the plot appears uses the NDC (Normalized Device Coordinates) with its values range from 0.0 to 1.0. The plots will be placed in the viewport automatically with the best aspect ratio. You can resize, move, overwrite text and many things more by using the appropriate resources.

The graphic below shows the NDC grid and the vp-resources (Viewport) used to define the start x- and y-position for plotting the plot size in NDC coordinates.

8.3 Maps

NCL supports many different map types and projections. By default, the continents are color filled using light grey; this behaviour can easily be changed by setting the `mpLandFillColor` resource to the desired color. You can also turn off the color fill by setting `mpFillOn` to False.

The default tickmark settings for a map can be changed by some `@gsn` and `@tm` resources, Not every resource can be used for map projections other than 'Cylindrical Equidistant', but on the NCL examples web page there are some examples for work arounds.

See also [http://ncl.ucar.edu/Applications/mptick.shtml](http://ncl.ucar.edu/Applications/mptick.shtml)

8.3.1 Default Map

To create a simple map of the Earth open a workstation and call the function `gsn_csm_map`. 
8.3.2 Map Grid and Tickmark Settings

To control the grid lines of the latitude and longitude axis as well as their major and minor tickmarks you have to use different resource groups like @gsn, @tm, and @mp.

NUG_map_grid_and_tickmark_settings.ncl:

```
begin
  wks = gsn_open_wks("png","plot_map_grid_and_tickmark_settings") ;-- define the workstation
  res = True
  res@mpGridAndLimbOn = True ;-- draw grid lines on the plot
  res@mpGridLatSpacingF = 20 ;-- grid line latitude spacing
  res@mpGridLonSpacingF = 45 ;-- grid line longitude spacing
  res@mpGridLineColor = "gray" ;-- grid line color
  res@mpGridLineThicknessF = 2 ;-- grid line thickness
  res@mpGridLineDashPattern = 2 ;-- grid line dash pattern
                           ;-- (2: dotted)
  res@gsnMajorLatSpacing = 10 ;-- change major lat tickmark spacing
  res@gsnMinorLatSpacing = 2.5 ;-- change major lat tickmark spacing
  res@tmYLabelStride = 3 ;-- write only every 3rd label
  res@tmYLabelFontHeightF = 0.016 ;-- change major lat tickmark spacing
  res@tmYMajorLengthF = 0.02 ;-- change the tickmark length
  res@tmYMinorLengthF = 0.01 ;-- change the tickmark length
  res@tmYMajorLineColor = "blue" ;-- change major tickmarks color
  res@tmYMinorLineColor = "grey20" ;-- change major tickmarks color
end
```
res@tmYLLLabelFontColor = "blue" ;-- change label color

;-- longitude settings
res@gsnMajorLonSpacing = 10 ;-- change major lon tickmark spacing
res@gsnMinorLonSpacing = 2.5 ;-- change major lon tickmark spacing
res@tmXBLLabelStride = 4 ;-- write only every 4th label
res@tmXBLLabelFontHeightF = 0.014 ;-- change major lat tickmark spacing
res@tmXBMajorLengthF = 0.02 ;-- change the tickmark length
res@tmXBMinorLengthF = 0.01 ;-- change the tickmark length
res@tmXBMajorLineColor = "red" ;-- change major tickmarks color
res@tmXBMinorLineColor = "grey20" ;-- change major tickmarks color
res@tmXBLabelFontColor = "red" ;-- change label color

;-- draw the map
map = gsn_csm_map(wks, res)
end

8.3.3 Map Content Settings

To change the color of land, inland water and ocean areas NCL provides the resources mpOceanFillColor, mpInlandWaterFillColor, and mpLandFillColor. Also, the line color and thickness of the continents can be controlled by resources.

NUG_map_land_ocean_settings.ncl:

begin
;-- define the workstation (plot type and name)
wks = gsn_open_wks("png","plot_map_land_ocean_settings")

;-- set resources
res = True ;-- use land fill (default: True)
res@mpOutlineOn = True ;-- outline land (default: False)
res@mpOceanFillColor = "lightblue" ;-- color to fill ocean
res@mpInlandWaterFillColor = "lightblue" ;-- color to fill inland water
res@mpLandFillColor = "navajowhite1" ;-- color to fill land
res@mpGeophysicalLineColor = "blue" ;-- outline color
res@mpGeophysicalLineThicknessF = 1.2 ;-- thickness of continental
Using the MediumRes map database allows you to add country outlines in addition to the default map outlines. (Note: http://www.gadm.org/ provides shapefiles containing better outlines. See also chapter 8.12)

NUG_map_countries.ncl:

```plaintext
begin
  ;-- define the workstation (plot type and name)
  wks = gsn_open_wks("png","plot_map_countries")

  ;-- set resources
  res  = True
  res@mpFillOn             = True       ;-- use land fill (default: True)
  res@mpOutlineOn          = True       ;-- outline land  (default: False)
  res@mpOutlineBoundarySets = "National" ;-- turn on country boundaries
  res@mpOceanFillColor     = "lightblue" ;-- color to fill ocean
  res@mpInlandWaterFillColor = "lightblue" ;-- color to fill inland water
  res@mpLandFillColor      = "navajowhite1" ;-- color to fill land
  res@mpGeophysicalLineColor = "blue"    ;-- outline color
  res@mpGeophysicalLineThicknessF = 1.2 ;-- thickness of continental
                                        ;-- outlines
  res@mpDataBaseVersion    = "MediumRes" ;-- map resolution

  ;-- draw the map
  map = gsn_csm_map(wks, res)
end
```
The map dataset "Earth..4" provides the provinces of Canada, the states of Mexico, and the states and counties of the conterminus United States, it also provides the state/province outlines of Australia, Brazil, China, and India. The ice shelves of Antarctica are included as separate entities, that can be made to appear or not, as desired.

The next example will show how to use this map dataset to plot Europe and color and/or outline some named countries.

NUG_map_selected_countries.ncl:

```ncl
begin
fill_colors = ("red", "green", "white", "gray", "blue", 
   "violet", "orange", "purple") ;-- fill colors
fill_areas = ("Spain", "Italy", "Switzerland", "Germany", "Latvia", 
   "Ireland", "Norway", "Greece");-- countries to be colored
outline_areas = ("Belgium", "Croatia");-- only outline countries
outline_areas := array_append_record (outline_areas, fill_areas, 0) ;-- concatenate arrays
outline_colors = ("red", "green", "white", "gray", "blue", "hotpink", 
   "orange", "purple") ;-- define the workstation (plot type and name)
wks = gsn_open_wks("png", "plot_map_select_countries")

;-- set resources
res = True
res@gsnMaximize = True
res@mpOutlineOn = True ;-- outline land (default: False)
res@mpOutlineSpecifiers = outline_areas ;-- which country to be outlined
res@mpFillOn = True ;-- use land fill (default: True)
res@mpFillAreaSpecifiers = fill_areas ;-- which country to be colored
res@mpSpecifiedFillColors = outline_colors ;-- set colors to be used
res@mpOceanFillColor = "lightblue" ;-- color to fill ocean
res@mpInlandWaterFillColor = "lightblue" ;-- color to fill inland water
res@mpLandFillColor = "navajowhite1" ;-- color to fill land
res@mpGeophysicalLineColor = "blue" ;-- outline color
res@mpGeophysicalLineThicknessF = 1.2 ;-- thickness of continental outlines
```

71
8.3.4 Change Map Projection

How to create a simple filled contour plot using the Mollweide projection can be seen in the next example.

NUG_projections_mollweide.ncl:

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
    ;-- read the data and define
diri = ";./"
    fili = "MITgcm_rectilinear_grid_3D.nc"
    file1 = addfile(diri+fili,"r")
    var = file1->SSS(0,0,:,:)

    ;-- define the workstation (plot type and name)
    wks = gsn_open_wks("png","NUG_mollweide")

    ;-- set resources
    res = True
    res@gsnMaximize = True
    res@lbLabelStride = 2
end
```
To change the map projection from "Mollweide" to "Robinson", just change the `res@mpProjection` setting to "Robinson".

The default projection is "CylindricalEquidistant". To change back to the default projection, uncomment the `res@mpProjection` line or set it to "CylindricalEquidistant".
8.3.5 Regional Map

Sometimes just a specific region of the data is of interest. To define the extent of a map region, insert the following resource settings in the script NUG_map_settings.ncl:

```
res@mpMinLonF = -20.0 ;-- min longitude
res@mpMaxLonF =  60.0 ;-- max longitude
res@mpMinLatF = -37.0 ;-- min latitude
res@mpMaxLatF =  40.0 ;-- max latitude
res@mpDataBaseVersion = "MediumRes" ;-- better map resolution
```

NUG_map_settings.ncl:
8.3.6 Polar Plot

To create a polar plot of the Northern Hemisphere, the `gsn_csm_contour_map_polar` function of NCL can easily be used. In this example, some additional settings are made to show the capabilities of polar plot resources.

Plot Northern Hemisphere:  
```
res@gsnPolar = "NH"
```
Label distance to the map:  
```
res@gsnPolarLabelDistance = 1.1
```
Label font size:  
```
res@gsnPolarLabelFontHeightF = 0.015
```
Label font:  
```
res@gsnPolarLabelFont = "helvetica-bold"
```

A procedure `polar_map_circle` is included to draw a thicker line around the map.

Use the acronym "SH" to plot the Southern Hemisphere instead of "NH" for the Northern Hemisphere in the resource `res@gsnPolar`.

Simple polar plot for the Northern Hemisphere: `NUG_polar_NH.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"

procedure polar_map_circle(wks,plot:graphic,wsize:integer,col:string,offset:numeric)
local degrad,degrees,xcos,xsin,xcenter,ycenter,radius,xc,yc
begin
    getvalues plot ;-- get viewport coordinates
    "vpXF"    : x
    "vpYF"    : y
    "vpWidthF" : w
    "vpHeightF" : h
end getvalues

degrad  = 0.017453292519943
degrees = ispan(0,360,1)
xcos    = cos(degrad * degrees)
xsin    = sin(degrad * degrees)
xcenter = w/2 + x
ycenter = h/2 + (y-h)
radius  = w/2 + offset
xc      = xcenter + (radius * xcos)
yc      = ycenter + (radius * xsin)

;-- set resources for circle and plot
lnres    = True
lnres@gsLineColor = col
lnres@gsLineThicknessF = wsize
gsn_polyline_ndc(wks,xc,yc,lnres)
end
```

```
fill = "rectilinear_grid_2D.nc"
f = addfile(diri+fill, "r")
u = f->tsurf(0,:,:)

;-- define the workstation (plot type and name)
wks = gsn_open_wks("png","plot_polar_settings")
gsn_define_colormap(wks,"ncl_default")

res = True
res@gsnDraw = False ;-- don't draw the plot
res@gsnFrame = False ;-- don't advance the frame
res@gsnPolar = "NH" ;-- show Northern Hemisphere
res@gsnPolarLabelSpacing = 15 ;-- grid spacing
res@gsnPolarLabelDistance = 1.1 ;-- default is 1.04
res@gsnPolarLabelFontHeightF = 0.015
res@gsnPolarLabelFont = "helvetica-bold"
res@gsnSpreadColorStart = 14 ;-- color index start
res@gsnSpreadColorEnd = -8 ;-- color index end
res@cnFillOn = True
res@tiMainString = "NCL Doc Example: Polar Plot (NH)"

;-- draw the plot
plot = gsn_csm_contour_map_polar(wks,u,res);-- from contributed.ncl lib
draw(plot);-- draw plot

;-- draw a circle around the map of plot: wsize=10, col="black", offset=0
;-- use function polar_map_circle from above
polar_map_circle(wks, plot, 7, "black", 0)

;-- advance the frame
frame(wks)
end
8.3.7 Map Resolutions

Maps can be drawn by NCL using one of the three different map resolution databases. The "LowRes" and "MediumRes" databases come with the NCL distribution, and the "HighRes" database has to be installed separately (a very simple installation). At DKRZ the high resolution database is already installed.


Low resolution: \[ \text{res@mpDataBaseVersion } = "\text{LowRes}" ;-- \text{the default} \]

Medium resolution: \[ \text{res@mpDataBaseVersion } = "\text{MediumRes}" \]

Using the HighRes database differs from the other settings, because you can specify a resolution, too.

High resolution: \[ \begin{align*}
\text{res@mpDataBaseVersion } &= "\text{HighRes}" \\
\text{res@mpDataResolution } &= "<\text{resolution}>" 
\end{align*} \]

\[
\text{resolution } = \begin{cases} 
\text{Unspecified} & \text{(default)} \\
\text{Coarsest} \\
\text{Coarse} \\
\text{Medium} \\
\text{Fine} \\
\text{Finest} 
\end{cases} 
\]

If "Unspecified" (default), the resolution is automatically set depending on the scale and size of the chosen region. A coarse resolution would be chosen for a map with a smaller scale, and a fine resolution for a map with a large scale. It is not recommended to use the "HighRes" database when plotting data over the whole globe, as it will take a long time and likely will not produce the correct plot.

You can also turn off the NCL map outlines and draw your own map outlines read in from a shapefile. Shapefiles can be downloaded for free, and are provided in many different resolutions. See section 9.13.

Map resolution example: NUG_map_resolutions.ncl
8.4 XY-Plots

An XY-plot is a plot containing curves made up of X/Y coordinate pairs. It also may contain tick marks, titles and/or a legend, and the look of the curves can be changed to different patterns or colors.

Simple xy-plot example: NUG_xy_plot.ncl

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

begin
  x = (/1,2,3,4,5,6,7,8,9,10/)
  y = (/3,7,4,9,2,1,8,8,4,2/)

  wks = gsn_open_wks("png","xy_plot")
  res = True
  res@tiMainString = "NCL Doc Example: xy-plot"

  plot = gsn_csm_xy(wks, x, y, res)
end
```

Insert the following lines below ‘res = True’ and see what happens:

```ncl
res@xyLineColor = "blue"
res@xyLineThicknessF = 5
```
To change the x-axis and y-axis strings some resource settings can be changed. The @ti in the resource name indicates that the 'Title' resources are used. Some title strings are set by default by NCL but if you want to change them you are able to overwrite them. Good examples are the sub-title strings on top of the plot which are set to the long_name of the variable (gsnLeftString) and units of the variable (gsnRightString). The gsnCenterString in the middle has no default setting.

8.4.1 Tickmark Settings

NCL draws major and minor tickmarks on the x- and y-axis by default. In the next part the modification of tickmarks and their spacing values and labels are demonstrated for xy-plots.
8.4.2 Time-series

A time-series plot is a little bit tricky because the time data type is commonly an integer or floating point value representing the values of "seconds since..." or "days since ..." for instance. To convert these numeric values to a normal date format, NCL provides a bunch of "calendar" functions. See also section 7.2 and the special procedure "time_axis_labels".

Simple contour example: NUG_xy_plot_timeseries.ncl

```plaintext
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/contrib/time_axis_labels.ncl"
begin
```
diri = "./"
fili = "rectilinear_grid_2D.nc"
f = addfile(diri+filii, "r")
var = f->tsurf
time = var&time

;-- compute the area mean without weighting
fldmean = wgt_areaave_Wrap(var,1.0,1.0,1)

wks = gsn_open_wks("png","xy_plot_timeseries")

;-- set resources
res                      = True
res@tiMainString         = "NCL Doc Example: xy-plot timeseries"
restime = True ;-- set time tickmark resources
restime@ttmFormat        = "%d %c %y" ;-- time tickmark format

time_axis_labels(time,res,restime) ; sets the correct time labels

res@tmXBLabelFontHeightF = 0.01
res@tmXBLabelJust       = "CenterRight"
res@tmXBLabelDeltaF     = 1.0
res@tmXBLabelAngleF     = 50.
res@tmLabelAutoStride   = True

plot = gsn_csm_xy(wks, time, fldmean, res)

end
8.5 Contours

Simple contour example: NUG_contour_map.ncl

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

begin
    ;----- read the data and define variable reference var
    diri = "./"
    fili = "rectilinear_grid_2D.nc"
    f  =  addfile(diri+fili, "r")
    var = f->tsurf(0,:,:)

    ;----- define the workstation (plot output type and name)
    wks = gsn_open_wks("png","plot_contour_map")

    ;----- set resources
    res = True
    res@gsnMaximize = True
    res@tiMainString = "NCL Doc Example: contour map" ;-- title string
    res@tiMainFontHeightF = 0.02

    ;----- draw the contour map
```
plot = gsn_csm_contour_map(wks, var, res)
end

The default type of contours lines are solid black lines. To change to a different color or dash pattern, use the cn resources:

Insert

res@cnLineDashPattern = 1 ; use dash pattern 1
or
res@cnMonoLineDashPattern = False ; use different dash pattern for each contour
res@cnLineColor = "NavyBlue"

A table of all available dash Patterns can be found in the Appendix C - Dash Pattern.
### 8.5.1 Filled Contours

**Simple filled contour example: NUG_contour_filled_map.ncl**

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

begin
  
  ;-- read the data and define
  
diri = "./"
  fill = "rectilinear_grid_2D.nc"
  f = addfile(diri+fill, "r")
  var = f->tsurf(0,:,:)

  ;-- define the workstation (plot type and name)
  wks = gsn_open_wks("png","plot_contour_filled_map")

  ;-- set resources
  res             =  True
  res@gsnMaximize =  True
  res@gsnMaximize =  True
  res@cnFillOn =  True ;-- turn on contour fill
  res@cnFillPalette = "BlueWhiteOrangeRed" ;-- choose color map
  res@cnLineOn =  False ;-- turn off contour lines
  res@cnLineLabelsOn =  False ;-- turn off line labels
  res@cnLevelSelectionMode = "ManualLevels" ;-- set contour levels manually
  res@cnMinLevelValF = 250. ;-- minimum contour level
  res@cnMaxLevelValF = 310. ;-- maximum contour level
  res@cnLevelSpacingF = 4. ;-- contour level spacing
  res@lbBoxMinorExtentF = 0.15 ;-- decrease the height of the labelbar
  res@tiMainString = "NCL Doc Example: filled contour map" ;-- title string
  res@tiMainFontHeightF = 0.02

  ;-- draw the contour map
  plot = gsn_csm_contour_map(wks, var, res)
end
```

![NCL Doc Example: filled contour map](image)
By default, NCL will calculate an equally-spaced array of 10 to 16 “nice” contour levels based on the minimum and maximum of your data values. You can change the level spacing that NCL chooses by simply setting `res@cnLevelSpacingF` to the desired spacing. To further control the contour levels as the above example does, you can set:

```
res@cnLevelSelectionMode  = "ManualLevels"
res@cnMinLevelValF        = 250.            ;-- minimum contour level
res@cnMaxLevelValF        = 310.            ;-- maximum contour level
res@cnLevelSpacingF       = 1               ;-- contour level spacing
```

To set an array of unequally-spaced contour levels, set:

```
res@cnLevelSelectionMode  = "ExplicitLevels"
res@cnLevels              = (/250,255,270,275,280,300,310/)
```
8.5.2 Filled and Dash Pattern Contour

A table of all available fill patterns can be found in the Appendix D - Fill Pattern.

Simple fill pattern contour example: NUG_contour_fillpattern.ncl

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

begin
    ;----- read the data and define variable reference var
    diri = "./"
    fili = "rectilinear_grid_2D.nc"
    file1 = addfile(diri+fili,"r")
    var = file1->tsurf(0,:,:)

    ;----- define the workstation (plot output type and name)
    wks = gsn_open_wks("png","plot_contour_fillpattern")

    ;----- set resources
    res = True
    res@gsnMaximize = True
    res@tiMainString = "NCL Doc Example: contour fill pattern"
    res@tiMainFontHeightF = 0.02
    res@cnLevelSelectionMode = "ManualLevels"
    res@cnMinLevelValF = 250.
    res@cnMaxLevelValF = 310.
    res@cnLevelSpacingF = 5.
    res@cnMonoFillPattern = False
    res@cnMonoFillScale = False
    res@cnFillOn = True
    res@cnFillColors = (/"blue4","blue","darkgreen","green","cyan3","
                        "gold","orange","darkorange","red","red4","
                        "violet","purple","mediumorchid4","purple4"/)
    res@cnFillPatterns = (/0,1,2,3,4,5,6,7,8,17,10,11,12,16/)
    res@cnFillDotSizeF = 0.003
    res@cnFillScales = (/1.,.4,.5,.3,.5,.5,.5,.5,1.,.5,.5,.5/.4/)

    ;----- draw the contour map
    plot = gsn_csm_contour_map(wks, var, res)
end
```

NCL Doc Example: contour fill pattern
8.6 Vector Plots

Plotting vector data as arrows is as simple as was seen earlier for contour lines. NCL provides a function that does all the work for the user. The next example uses only the NCL default settings for vector plotting.

Vector field example: NUG_vector_default.ncl

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

begin
  diri = "./"
  fili = "rectilinear_grid_2D.nc"
  f    = addfile(diri+fili, "r")
  u    = f->u10(0,:,:); -- first time step
  v    = f->v10(0,:,:); -- first time step

  ;-- define the workstation (graphic will be written to a file)
  wks = gsn_open_wks("png","plot_vector_default")

  ;-- draw the vectors
  plot = gsn_csm_vector_map(wks,u,v,False)

end
```
A very nice way of displaying a vector field is **CurlyVector**, which plots short streamline segments with curved arrows instead of straight arrows. This example also sets some useful resources to control the length and density of the vectors.

**Vector field example: NUG_vector_curly.ncl**

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

begin
  diri = "./"
  fili = "rectilinear_grid_2D.nc"
  f = addfile(diri+fili, "r")
  u = f->u10(0,:,:)
  v = f->v10(0,:,:)
  ;-- first time step

  wks = gsn_open_wks("png","plot_vector_curly")

  vres = True
  vres@gsnMaximize        = True
  vres@tiMainString       = "NCL Doc Example: vector curly"
  vres@vcMinFracLengthF   = 1.0 ;-- length of min vector as fraction of reference vector
  vres@vcRefMagnitudeF    = 3.0 ;-- make vectors larger
  vres@vcRefLengthF       = 0.045 ;-- ref vec length
  vres@vcGlyphStyle       = "CurlyVector" ;-- turn on curly vectors
  vres@vcMinDistanceF     = 0.01 ;-- thin out vectors

  ;-- draw the vectors
  plot = gsn_csm_vector_map(wks,u,v,vres)
end
```
Vector field colorized by surface temperature: NUG_vector_plot_colored.ncl

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

begin
  diri = "./"
  fili = "rectilinear_grid_2D.nc"
  uname = "u10"
  vname = "v10"
  tname = "tsurf"

  -- read the data
  f = addfile(diri+fili,"r") ;-- open file with read
  access
  u = f->$uname$(0,:,:) ;-- first time step
  v = f->$vname$(0,:,:) ;-- first time step
  t = f->$tname$(0,:,:) ;-- first time step

  -- define the workstation (graphic will be written to a file)
  wks = gsn_open_wks("png","plot_vector_colorized")

  -- set plot resources
  res = True
  res@gsnMaximize = True ;-- maximize plot in frame
  res@vcMinFracLengthF = 1.0 ;-- length of min vector as
                             -- fraction of reference vector
  res@vcRefMagnitudeF = 3.0 ;-- make vectors larger
  res@vcRefLengthF = 0.045 ;-- ref vec length
  res@vcGlyphStyle = "CurlyVector" ;-- turn on curly vectors
  res@vcMinDistanceF = 0.01 ;-- thin out vectors
  res@vcLevelPalette = "ncl_default" ;-- choose color map

  res@tiMainString = "NCL Doc Example: vectors colorized by temperature" ;-- title

  -- draw the vectors
  plot = gsn_csm_vector_scalar_map(wks,u,v,t,res)
end
```
This example uses the "overlay" procedure to overlay vectors on a contour/map plot. See Section 9.10 for more information about overlays.

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
  diri = "./"
  fili = "rectilinear_grid_2D.nc"
  uname = "u10"
  vname = "v10"
  tname = "tsurf"

  ;;-- read the data
  f   =  addfile(diri+fili,"r") ;;-- open file with read access
  u   =  f->u10(0,:,:)
  v   =  f->v10(0,:,:)
  t   =  f->tsurf(0,:,:)

  ;;-- define the workstation (graphic will be written to a file)
  wks =  gsn_open_wks("png","plot_vector_overlay")

  ;;-- set plot resources
  cnres                     = True
  cnres@gsnDraw             = False ;;-- don't draw
  cnres@gsnFrame            = False ;;-- don't advance frame
  cnres@cnFillOn            = True ;;-- turn on color
  cnres@cnLinesOn           = False ;;-- no contour lines
  cnres@ncFillPalette       = "ncl_default" ;;-- choose color map
  cnres@mpFillOn            = False ;;-- no map fill
  cnres@gsnLeftString       = "surface temperature" ; change left string
  cnres@gsnRightString      = t@units ;;-- assign right string
  cnres@tiMainString        = "NCL Doc Example: vectors overlay on map"

  vcres                     = True ;;-- vector only resources
  vcres@gsnDraw             = False ;;-- don't draw
  vcres@gsnFrame            = False ;;-- don't advance frame
  vcres@vcGlyphStyle        = "CurlyVector" ;;-- curly vectors
  vcres@vcRefMagnitudeF     = 20 ;;-- define vector ref mag
  vcres@vcRefLengthF        = 0.045 ;;-- define length of vec ref
  vcres@vcRefAnnoOrthogonalPosF = -.535 ;;-- move ref vector into plot
  vcres@gsnRightString      = " " ;;-- turn off right string
```

This example uses the "overlay" procedure to overlay vectors on a contour/map plot. See Section 9.10 for more information about overlays.
Vector field on filled contour map example: NUG_vector_plot_overlay.ncl
8.7 Slice Plots

3-dimensional structures in the data can be examined by means of 2D visualization methods if different slices through the data are jointly analyzed. The example here shows a vertical slice through a 3D data volume at latitude 40N, longitudes ranging from 0 to 60E across all levels in hPa units.

Slice plot example: NUG_slice_plot.ncl

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

begin
diri = "/"
fili = "rectilinear_grid_3D.nc"
f = addfile(diri+fili, "r")
var = f->t(0,:,{40},{0:60}) ; -- first time step, latitude=40N, longitude=0-60E.
lon_t = f->lon({0:60}) ; -- longitude=0-60E
lev_t = f->lev ; -- currently 17 levels

;-- define workstation
wks = gsn_open_wks("png","plot_slices")

;-- set resources
res = True
res@tiMainString = "NCL Doc Example: Slice plot at 40N" ;-- title string
res@cnFillOn = True ;-- turn on color fill
res@cnLineLabelsOn = False ;-- turns off contour line labels
res@cnInfoLabelOn = False ;-- turns off contour info label
res@cnFillPalette = "ncl_default" ;-- choose color map
res@lbOrientation = "vertical" ;-- vertical label bar
res@tiYAxisString = var@long_name+" [hPa]"
res@sfXArray = lon_t ;-- uses lon_t as plot x-axis
res@sfYArray = lev_t/100 ;-- uses lev_t in hPa as plot y-axis
res@trYReverse = True ;-- reverses y-axis
res@gsnYAxisIrregular2Log = True ;-- converts y-axis irregular to linear depth

;-- generate the plot
plot = gsn_csm_contour(wks,var,res)
end
```

![Slice plot example](attachment:image.png)
8.8 Bar Charts

Bar chart plots are more or less simple XY-plots with bars for the xy-points. Graphically, there is only a small difference between histograms and bar charts, but histograms are for binning data and have their own resources, which are described in the last example of the bar chart section.

Bar chart example: NUG_bar_chart.ncl

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

low = 0.0
high = 1.0

n = 12
x = fspan(1.0, 12.0, n)
y = random_uniform(low, high, n)

wks = gsn_open_wks("png","plot_bar_chart")

res = True
res@gsnXYBarChart = True
res@gsnXYBarChartBarWidth = 0.3
res@gsnXYBarChartColors = "blue"
res@trXMinF = 0.0               ;-- x-axis min value
res@trXMaxF = 13.0              ;-- x-axis max value
res@trYMinF = 0.0               ;-- y-axis min value
res@trYMaxF = 1.0               ;-- y-axis max value
res@tmXBMode = "Explicit"       ;-- explicit labels
res@tmXBValues = ispan(1,12,1)
res@tmXBLabels =
("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep",
 "Oct","Nov","Dec")
res@tmXBLabelFontHeightF = 0.015
res@tiMainString = "NCL Doc Example: bar chart"

plot = gsn_csm_xy(wks, x, y, res)
end
```
Bar chart example displaying 3 different data sets - a method which can be used to visualize 3 different realizations of an ensemble simulation: NUG_bar_chart_multi.ncl

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

begin
low = 0.0
high = 1.0
n = 12

x = fspan(0.5, 11.5, n)
y1 = random_uniform(low, high, n)
y2 = random_uniform(low, high, n)
y3 = random_uniform(low, high, n)

wks = gsn_open_wks("png","plot_bar_chart_multi")

res = True
res@gsnDraw = True
res@gsnFrame = False
res@gsnXYBarChart = True
res@gsnXYBarChartBarWidth = 0.25

res@trXMinF = 0.0 ;-- x-axis min value
res@trXMaxF = 12.5 ;-- x-axis max value
res@trYMinF = 0.0 ;-- y-axis min value
res@trYMaxF = 1.0 ;-- y-axis max value

res@tmXBMode = "Explicit" ;-- explicit labels
res@tmXBValues = ispan(1,12,1)-0.25 ;-- center labels
res@tmXBLabels = ("Jan","Feb","Mar","Apr","May","Jun", "Jul","Aug","Sep", "Oct","Nov","Dec")

res@tmXLabelFontHeightF = 0.015
res@tiMainString = "NCL Doc Example: bar chart of multi data sets"
res@gsnXYBarChartColors = "blue"
plots1 = gsn_csm_xy(wks, x, y1, res)

res@gsnXYBarChartColors = "gold"
plots2 = gsn_csm_xy(wks, x+0.25, y2, res)

res@gsnXYBarChartColors = "green"
plots3 = gsn_csm_xy(wks, x+0.5, y3, res)

frame(wks)
end
```
Bar chart example displaying values above or below a reference value with different colors:

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

begin
    low  = -1.0
    high =  1.0
    n    = 24
    x = fspan(1.0, 12.0, n)
    y = random_uniform(low, high, n)

    wks = gsn_open_wks("png","plot_bar_chart_col_above_below")

    res = True
    res@gsnDraw = True
    res@gsnFrame = False
    res@gsnXYBarChart = True
    res@gsnXYBarChartBarWidth = 0.25
    res@trXMinF = 0.0 ;-- x-axis min value
    res@trXMaxF = 13.0 ;-- x-axis max value
    res@trYMinF = -1.0 ;-- y-axis min value
    res@trYMaxF = 1.0 ;-- y-axis max value
    res@tmXBMode = "Explicit" ;-- explicit labels
    res@tmXBValues = fspan(1,12,n)
    res@tmXBLables = ("Jan","","Feb","","Mar","","Apr","","May","","Jun","","Jul","","Aug","","Sep","","Oct","","Nov","","Dec","")
    res@tmXBLabelFontHeightF = 0.015
    res@tiMainString = "NCL Doc Example: bar chart coloring above/below"
    res@gsnYRefLine = 0. ; reference line
    res@gsnXYBarChart = True ; create bar chart
    res@gsnAboveYRefLineColor = "red" ; above ref line fill red
    res@gsnBelowYRefLineColor = "blue" ; below ref line fill blue

    plot = gsn_csm_xy(wks, x, y, res)

    frame(wks)
end
```
Histogram example: NUG_histograms.ncl

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

begin

;-- generate a 2D data set (gsn_histogram will go into compare mode)
data = new((/2,1000/),float)
data(0,:) = random_uniform(0,500.,1000)
data(1,:) = random_uniform(0,500.,1000)
xint = ispan(0,500,25)

;-- open workstation
wks = gsn_open_wks("png","plot_histograms") ;-- open workstation
gsn_define_colormap(wks,"rainbow") ;-- choose color map
res                                   = True
res@gsnHistogramBarWidthPercent       = 70.
res@gsnHistogramClassIntervals        = xint
res@tmXBLabelAngleF                   = 325. ;-- change label angle
res@tmLabelAutoStride                 = True ;-- prevent label overlap
res@tiMainString                      = "NCL Doc Example: Histograms"
plot = gsn_histogram(wks,data,res)    ;-- create histogram
end
```

**NCL Doc Example: Histograms**

![Histogram Example](image)
To change the filled histogram bars to a more transparent mode, the gsFillOpacityF resource can be used:

```
res@gsFillOpacityF = 0.4
```
8.9 Overlay Plots

One great feature of NCL is its support for overlaying graphical elements on top of other elements. Take a look at the NUG_vector_plot_colorized.ncl example in section 8.6, where a filled contour plot is overlaid by a vector plot. The powerful “overlay” procedure overlays one plot on a base plot, such that the base plot now contains both plots. NCL examines the data space of both plots to correctly transform the overlay plot to the base plot. If both plots you want to overlay are maps, then only the base plot can be a map, and the overlay plot must just be a contour, vector, or other type of plot.

Furthermore, NCL supports transparency in combination with overlays. This is, as an example, quite useful for highlighting a region of interest while the area outside of this region is still visible. The transparency/opaque example script shows how to emphasize a region of interest:

NUG_transparent_filled_contour.ncl

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

begin

;-- read data
diri = "./"
fill = "uv300.nc"

a = addfile(diri+fill,"r")
u = a->U(1,:,:)

;-- open a workstation
wks = gsn_open_wks("png","plot_transparency")

;-- set resources
res = True
res@gsnFrame = False
res@gsnDraw = False
res@cnFillOn = True
res@cnFillPalette = "BlueYellowRed"
res@cnLinesOn = False
res@cnLineLabelsOn = False
res@cnInfoLabelOn = False
res@cnLevelSelectionMode = "ExplicitLevels"
res@cnLevels = ispan(-12,40,2)

;-- set resources only for main plot displaying the contours
bres = res
bres@gsnMaximize = True ;-- maximize main plot
bres@mpFillOn = False
bres@tiMainString = "NCL Doc Example: transparency I"
bres@cnFillOpacityF = 0.5 ;-- 50% opaque

main_plot = gsn_csm_contour_map(wks,u,bres)

;-- set resources for the overlaid plot
ores = res
ores@cnFillOpacityF = 1.0 ;-- 100% opaque
ores@gsnRightString = ""
ores@gsnLeftString = ""
ores@lbLabelBarOn = False

overlay_plot = gsn_csm_contour(wks,u({-30:30},{-120:120}),ores)
```

overlay(main_plot, overlay_plot)

draw(main_plot)
frame(wks)

;-- create a new plot with 100% opacity
bres@tiMainString = "NCL Doc Example: transparency II"
bres@cnFillOpacityF = 1.0 ;-- 100% opaque
plot = gsn_csm_contour_map(wks, u, bres)

;---Set resources for a partially transparent polygon.
gnres = True
gnres@gsFillOpacityF = 0.5 ;-- 50% opaque
gnres@gsFillColor = "white"

lat_box = (-30, -30, 30, 30, -30/)
lon_box = (-120, 120, 120, -120, -120/)

gsid = gsn_add_polygon(wks, plot, lon_box, lat_box, gnres)

;-- draw the second plot to a new page
draw(plot)
frame(wks)
end

The first image shows a more 'highlighted' section:
The second image of this example, a semi transparent white rectangle is displayed on top of the opaque contours plot.

Another good application of the overlay technique is drawing multiple time series on one plot to compare different model data or variables. The lines of the data/variable are plotted with different colors, dash pattern, and a user-defined legend which will be replaced inside the plot frame.

This example is saved in NUG_multi_timeseries.ncl.

```ncl
begin
diri = "./"
fil1 = "tas_mod1_hist_rectillin_grid_2D.nc"
fil2 = "tas_mod1_rcp45_rectillin_grid_2D.nc"
fil3 = "tas_mod1_rcp85_rectillin_grid_2D.nc"
fil4 = "tas_mod2_hist_rectillin_grid_2D.nc"
fil5 = "tas_mod2_rcp45_rectillin_grid_2D.nc"
fil6 = "tas_mod2_rcp85_rectillin_grid_2D.nc"
fil7 = "tas_mod3_hist_rectillin_grid_2D.nc"
fil8 = "tas_mod3_rcp45_rectillin_grid_2D.nc"
fil9 = "tas_mod3_rcp85_rectillin_grid_2D.nc"
fil10 = "tas_mod4_hist_rectillin_grid_2D.nc"
fil11 = "tas_mod4_rcp45_rectillin_grid_2D.nc"
fil12 = "tas_mod4_rcp85_rectillin_grid_2D.nc"

f1 = addfile(diri+fil1, "r")
f2 = addfile(diri+fil2, "r")
f3 = addfile(diri+fil3, "r")
f4 = addfile(diri+fil4, "r")
f5 = addfile(diri+fil5, "r")
f6 = addfile(diri+fil6, "r")
end
```
f7 = addfile(diri+fil7, "r")
f8 = addfile(diri+fil8, "r")
f9 = addfile(diri+fil9, "r")
f10 = addfile(diri+fil10, "r")
f11 = addfile(diri+fil11, "r")
f12 = addfile(diri+fil12, "r")
;
---------------------------------------------------------------------------
temp1 = f1->tas(:,0,0,0)
temp1@long_name = "2m temperature"
temp2 = f2->tas(:,0,0,0)
temp2@long_name = "2m temperature"
temp3 = f3->tas(:,0,0,0)
temp3@long_name = "2m temperature"
---------------------------------------------------------------------------
temp4 = f4->tas(:,0,0,0)
temp4@long_name = "2m temperature"
temp5 = f5->tas(:,0,0,0)
temp5@long_name = "2m temperature"
temp6 = f6->tas(:,0,0,0)
temp6@long_name = "2m temperature"
---------------------------------------------------------------------------
temp7 = f7->tas(:,0,0,0)
temp7@long_name = "2m temperature"
temp8 = f8->tas(:,0,0,0)
temp8@long_name = "2m temperature"
temp9 = f9->tas(:,0,0,0)
temp9@long_name = "2m temperature"
---------------------------------------------------------------------------
temp10 = f10->tas(:,0,0,0)
temp10@long_name = "2m temperature"
temp11 = f11->tas(:,0,0,0)
temp11@long_name = "2m temperature"
temp12 = f12->tas(:,0,0,0)
temp12@long_name = "2m temperature"

---------------------------------------------------------------------------
time = ispan(1950,2098,1)
time@long_name = "Time"
---------------------------------------------------------------------------
; to plot multiple lines, you must put them into a multidimensional array
---------------------------------------------------------------------------
data = new((/12,149/),float)
data(0,0:55) = temp1
data(1,56:148) = temp2
data(2,56:148) = temp3
data(3,0:55) = temp4
data(4,56:148) = temp5
data(5,56:148) = temp6
data(6,0:55) = temp7
data(7,56:148) = temp8
data(8,56:148) = temp9
data(9,0:55) = temp10
data(10,56:148) = temp11
data(11,56:148) = temp12

;-- open a workstation
wks = gsn_open_wks("png", "plot_multi_timeseries")
;-- set resources
res = True
res@xyExplicitLegendLabels = (/" Data 1 historical"," Data 1 rcp85", \
" Data 2 historical"," Data 2 rcp85", \
" Data 3 historical"," Data 3 rcp85", \
" Data 4 historical"," Data 4 rcp85"/)
res@xyLineColors           = (/ "gray55", "gray55", "gray55", \"blue", "blue", "blue", \"red", "red", "red", "green", \"green", "green"/)
res@xyDashPatterns       = (/0, 5, 2, 0, 5, 2, 0, 5, 2, 0, 5, 2/) 
res@xyLineThicknessF       = 3
res@tiYAxisString          = temp1@long_name+"[ K ]"
res@tiYAxisFont            = 21
res@tiYAxisFontAspectF     = 1.3
res@tiYAxisFontHeightF     = 0.012
res@tiXAxisString          = "Year"
res@tiXAxisFont            = 21
res@tiXAxisFontAspectF     = 1.3
res@tiXAxisFontHeightF     = 0.012
res@tiXAxisOffsetYF        = 0.0
res@tiMainOffsetYF         = 0.11
res@tiMainString         = "NCL Doc Example: multiple timeseries"
res@tmXBLabelFontAspectF   = 1.3
res@tmXBLabelFontHeightF   = 0.012
res@trYMinF                = 292.0
res@trYMaxF                = 300.0
res@gsnMaximize            = True
res@gsnDraw                = False
res@gsnFrame               = False
res@vpXF                   = 0.25 ;-- set viewport resources
res@vpYF                   = 0.6
res@vpWidthF               = 0.7
res@vpHeightF              = 0.4
res@pmLegendDisplayMode    = "Always"
res@lgLabelFontHeightF     = 0.01
res@pmLegendWidthF         = 0.12 ;-- set legend width
res@pmLegendHeightF        = 0.19 ;-- set legend height
res@pmLegendOrthogonalPosF = -1.14 ;-- move legend up
res@pmLegendParallelPosF   = 0.18 ;-- move legend right

;-- create plot
plot = gsn_csm_xy(wks,time,data,res)

;-- set text resources
txres = True
txres@txFontHeightF = 0.010

;-- text bottom
TimeStamp           = systemfunc( "date +%F" )
txres@txJust        = "BottomLeft"
gsn_text_ndc(wks,"German Climate Computing Center (DKRZ), Hamburg",0.03,0.1, txres)
txres@txJust        = "BottomRight"
gsn_text_ndc(wks,TimeStamp,0.97, 0.1, txres)

;-- text top middle
txres@txFontHeightF = 0.012
A very pretty example demonstrating the benefit of overlays is plotting three data sets with different grid resolutions on one map.

Compare grid resolution example script: NUG_grid_resolution_comparation.ncl
```plaintext
diri = "./"
fil1 = "orog_mod1_rectilinear_grid_2D.nc"
fil2 = "orog_mod2_rectilinear_grid_2D.nc"
fil3 = "orog_mod3_rectilinear_grid_2D.nc"
msk1 = "sftlf_mod1_rectilinear_grid_2D.nc"
msk2 = "sftlf_mod2_rectilinear_grid_2D.nc"
msk3 = "sftlf_mod3_rectilinear_grid_2D.nc"

file_1 = addfile(diri+fil1, "r")
variable1 = file_1 ->orog(:,:,)

mask_1 = addfile(diri+msk1,"r")
lsml = mask_1 ->sftlf(:,:,)
lsml = lsml/100 ;-- do this cause data in percent
lsml = where(lsml.gt.0.5,-9999,lsml)

;;-- mask data over ocean
land_only1 = variable1
land_only1 = mask(variable1,lsml,-9999)

;;-- second file
file_2 = addfile(diri+fil2, "r")
variable2 = file_2 ->orog(0,[lat0:lat1+1],[lon0+1:lon1])
variable2&rlat@units = "degrees_north"
variable2&rlon@units = "degrees_east"

mask_2 = addfile(diri+msk2,"r")
lsml = mask_2 ->sftlf(0,[lat0:lat1+1],[lon0+1:lon1])
lsml = where(lsml.gt.0.5,-9999,lsml)

;;-- mask data over ocean
land_only2 = variable2
land_only2 = mask(variable2,lsml,-9999)

;;-- third file
file_3 = addfile(diri+fil3, "r")
variable3 = file_3 ->orog(0,[lat0:lat1+1],[lon0+1:lon1])
variable3&rlat@units = "degrees_north"
variable3&rlon@units = "degrees_east"

mask_3 = addfile(diri+msk3, "r")
lsml = mask_3 ->sftlf(0,[lat0:lat1+1],[lon0+1:lon1])
lsml = where(lsml.gt.0.5,-9999,lsml)

;;-- mask data over ocean
land_only3 = variable3
land_only3 = mask(variable3,lsml,-9999)

;;-- open workstation
wks = gsn_open_wks("png","plot_grid_resolution_comparison")
gsn_define_colormap(wks,"OceanLakeLandSnow")

cnres = True ; plot mods desired
cnres@gsnDraw = False
cnres@gsnFrame = False
cnres@gsnMaximize = True ; Maximize plot in frame
cnres@gsnAddCyclic = False ; Don't add a cyclic point
cnres@gsnLeftString = ""
cnres@gsnCenterString = ""
cnres@gsnRightStringFontHeightF = -0.02
cnres@gsnRightStringParallelPosF = 1.11 ; move the RightString slightly right

cnres@cnInfoLabelOn = False
cnres@cnLinesOn = False
cnres@cnLineLabelsOn = False
cnres@cnLevelSelectionMode = "ManualLevels" ; Set contour levels manually
cnres@cnMinLevelValF = 0. ; Minimum contour level
```

105
cnres@cnMaxLevelValF        =  3000.          ; Maximum contour level
cnres@cnLevelSpacingF       =  20             ; Contour level spacing
cnres@cnFillOn              =  True           ; Turn on contour fill
cnres@cnFillMode            = "RasterFill"
cnres@gsnSpreadColors       =  True
cnres@gsnSpreadColorStart   =  2
cnres@gsnSpreadColorEnd     = -26

cnres@lbLabelBarOn          =  True
cnres@lbLabelStride         =  10
cnres@lbOrientation         = "Vertical"
cnres@tiMainOn              = False

;-- resources first plot (map)
res = cnres
res@mpProjection            = "CylindricalEquidistant"
res@mpLimitMode             = "Corners"
res@mpLeftCornerLonF        = lon0
res@mpRightCornerLonF       = lon1
res@mpLeftCornerLatF        = lat0
res@mpRightCornerLatF       = lat1
res@mpGridAndLimbOn         = False               ; turn on grid lines
res@mpDataBaseVersion       = "MediumRes"
res@pmTickMarkDisplayMode   = "Always"            ; turn on tickmarks
res@lbLabelBarOn            = True
res@lbBoxLinesOn            = False               ; turn off labelbar box lines
res@gsnAddCyclic            = True

map = gsn_csm_contour_map(wks, land_only1, res)

;-- second plot
res2 = cnres
res2@lbLabelBarOn           = False
res2@gsnMaximize            = False
res2@cnLinesOn              = False
res2@cnInfoLabelOn          = False
res2@gsnRightString         = ""

plot2 = gsn_csm_contour(wks, land_only2, res2)

;-- third plot
res3 = cnres
res3@lbLabelBarOn           = False
res3@gsnMaximize            = False
res3@cnLinesOn              = False
res3@cnInfoLabelOn          = False
res3@gsnRightString         = ""

plot3 = gsn_csm_contour(wks, land_only3, res3)

;-- overlay the two data sets
overlay(map,plot2)

;-- draw a shaded box around plot2 (means draw two boxes with different line width)
shres2                      = True
shres2@gsLineThicknessF     =  7.0
shres2@gsLineColor          = "gray65"

dx = 0.15
dy = 0.25

shypts2=(/lat0+dy,lat1-border1+dy, lat1-border1+dy, lat0+dy/)
shxpts2=(/lon0+border1-dx,lon0+border1-dx,lon1-dx,lon0+border1-dx,lon0+border1-dx/)

shdum2 = new(4,graphic)
do i = 0 , 3
  shdum2(i)=gsn_add_polyline(wks,map,shxpts2(i:i+1),shypts2(i:i+1),shres2)
end do
;-- draw a box around plot2
lnres2 = True
lnres2@gSLineThicknessF = 1.0
lnres2@gsLineColor = "Black"

ypts2= (/ lat0, lat0, lat1-border1, lat1-border1, lat0/)
xpts2= (/ lon0+border1, lon1, lon1, lon0+border1, lon0+border1/)

dum2 = new(4, graphic)
do i = 0 , 3
   dum2(i)=gsn_add_polyline(wks,map,xpts2(i:i+1),ypts2(i:i+1),lnres2)
end do

;-- overlay plot3
overlay(map,plot3)

;-- draw a shaded box around plot3 ( means draw two boxes with different line width
shres3 = True
shres3@gSLineThicknessF = 7.0
shres3@gsLineColor = "gray65"
dx = 0.15
dy = 0.25

shypts3= (/lat0+dy, lat1-border2+dy, lat1-border2+dy, lat0+dy/)
shxpts3= (/lon0+border2-dx,lon0+border2-dx,lon1-dx,lon0+border2-dx,lon0+border2-dx/)

shdum3 = new(4, graphic)
do i = 0 , 3
   shdum3(i)=gsn_add_polyline(wks,map,shxpts3(i:i+1),shypts3(i:i+1),shres3)
end do

;-- draw a box around plot3
lnres3 = True
lnres3@gSLineThicknessF = 1.0
lnres3@gsLineColor = "Black"

ypts3 = (/ lat0, lat0, lat1-border2, lat1-border2, lat0/)
xpts3 = (/ lon0+border2, lon1, lon1, lon0+border2, lon0+border2/)

dum3 = new(4, graphic)
do i = 0 , 3
   dum3(i)=gsn_add_polyline(wks,map,xpts3(i:i+1),ypts3(i:i+1),lnres3)
end do

;-- drawing the map will draw contours and vectors too.
draw(map)

gsn_text_ndc(wks,"1.875 ~S~o~N~ GCM", 0.200, 0.730, txres)
gsn_text_ndc(wks,"0.440 ~S~o~N~ RCM", 0.330, 0.600, txres)
gsn_text_ndc(wks,"0.220 ~S~o~N~ RCM", 0.470, 0.460, txres)
frame(wks)
end
8.10 Panel Plots

NCL allows the drawing of multiple plots on a single page (frame) using the procedure `gsn_panel`. The plots will be drawn from the left to right side of the page and from the top down to the bottom. A common labelbar or title for all plots can be included.

Panel plot example: NUG_panel_plot.ncl

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
  -- read data and set variable references
  diri = "./"
  fil1 = "orog_mod1_rectilinear_grid_2D.nc"
  fil2 = "sftlf_mod1_rectilinear_grid_2D.nc"
  fil3 = "tas_rectilinear_grid_2D.nc"
  fil4 = "uas_rectilinear_grid_2D.nc"
  fil5 = "vas_rectilinear_grid_2D.nc"
  f1   = addfile(diri+fil1, "r")
  f2   = addfile(diri+fil2, "r")
  f3   = addfile(diri+fil3, "r")
  f4   = addfile(diri+fil4, "r")
  f5   = addfile(diri+fil5, "r")
  orog = f1->orog
  sftlf = f2->sftlf
  t    = f3->tas
  u    = f4->uas
  v    = f5->vas

  -- open a PNG file
  wks = gsn_open_wks("png","plot_panel_plot")

  -- create plot array
  plot = new(3,graphic)

  -- set resources for contour plots
```
res@gsnDraw = True
res@gsnFrame = False
res@gsnAddCyclic = True
res@gsnMaximize = True
res@tiMainString = "NCL Doc Example: panel plot"
res@cnInfoLabelOn = False

plot(0) = gsn_csm_contour_map(wks,u(0,:,:,:),res)
res@tiMainString = ""
plot(1) = gsn_csm_contour_map(wks,v(0,:,:,:),res)

;-- set resources for vector plot
vres = True
vres@gsnDraw = False
vres@gsnFrame = False
vres@gsnAddCyclic = True
vres@gsnMaximize = True
vres@gsnLeftString = "Vector Wind"
vres@vcRefAnnoOrthogonalPosF = -1.0
vres@vcRefMagnitudeF = 10.0
vres@vcRefLengthF = 0.045
vres@vcGlyphStyle = "CurlyVector"
vres@vcMinDistanceF = 0.017

plot(2) = gsn_csm_vector_map(wks,u(0,:,:4,:,:4),v(0,:,:4,:,:4),vres)

;-- create panel plot
gsn_panel(wks,plot,(/3,1/),False)
end
Panel plot example 3 rows x 2 columns: NUG_panel_plot_3x2.ncl

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

begin
  diri = "./"
  fil1 = "orog_mod1_rectilinear_grid_2D.nc"
  fil2 = "sftlf_mod1_rectilinear_grid_2D.nc"
  fil3 = "tas_rectilinear_grid_2D.nc"
  fil4 = "uas_rectilinear_grid_2D.nc"
  fil5 = "vas_rectilinear_grid_2D.nc"

  f1   =  addfile(diri+fil1, "r")
  f2   =  addfile(diri+fil2, "r")
  f3   =  addfile(diri+fil3, "r")
  f4   =  addfile(diri+fil4, "r")
  f5   =  addfile(diri+fil5, "r")

  orog  = f1->orog
  sftlf = f2->sftlf
  t     = f3->tas
  u     = f4->uas
  v     = f5->vas

  land_only = orog
  land_only = where(sftlf .ge.10, orog, orog@_FillValue)

;-- open a PNG file
  wks = gsn_open_wks("png","plot_panel_3x2_plot")

;-- create plot array (3 rows and 2 columns  3*2=6)
  plot = new(6,graphic)

;-- set resources for contour plots
  res = True
  res@gsnDraw = False
  res@gsnFrame = False
  res@gsnAddCyclic = True
  res@tiMainString = ""
  res@cnInfoLabelOn = False
  res@cnFillOn = True
  res@cnFillMode = "RasterFill"
  res@cnRasterSmoothingOn = True
  res@cnFillPalette = "BlueRed"

;-- upper left plot
  plot(0) = gsn_csm_contour_map(wks,u(0,:,:),res)

;-- upper right plot
;-- set the viewport to the same size as plot(0)
  getvalues plot(0)
    "vpWidthF"  : vpw
    "vpHeightF" : vph
  end getvalues

res@mpShapeMode = "FreeAspect"
res@vpWidthF = vpw
res@vpHeightF = vph
res@mpLimitMode = "Corners"
res@mpLeftCornerLonF = 60.0
res@mpRightCornerLonF = 120.0
res@mpLeftCornerLatF = 8.0
```

res@mpRightCornerLatF = 43.0
res@mpRightCornerLonF = 60.0
res@mpLeftCornerLatF = 34.0
res@mpLeftCornerLonF = -10.0

res@mpShapeMode = "FreeAspect"
res@vpWidthF = vpw
res@vpHeightF = vph
res@mpLinesOn = False
res@mpFillPalette = "ncl_default"
res@mpLimitMode = "Corners"
res@mpLeftCornerLonF = -10.0
res@mpRightCornerLonF = 60.0

res@cnFillPalette = "OceanLakeLandSnow"
res@cnLevelSelectionMode = "ManualLevels" ;-- manually levels
res@cnMinLevelValF = 0. ;-- minimum level
res@cnMaxLevelValF = 5000. ;-- maximum level
res@cnLevelSpacingF = 250. ;-- level spacing

plot(1) = gsn_csm_contour_map(wks, land_only, res)

;-- delete some resources
delete([/res@mpLimitMode, res@mpLeftCornerLonF, res@mpRightCornerLonF, \
       res@mpLeftCornerLatF, res@mpRightCornerLatF, \
       res@cnLevelSelectionMode, res@cnMinLevelValF, \
       res@cnMaxLevelValF, res@cnLevelSpacingF/])

;-- middle left plot
res@tiMainString = ""
res@cnFillPalette = "BlueRed"
res@cnLevelSelectionMode = "AutomaticLevels" ;-- automat. levels

plot(2) = gsn_csm_contour_map(wks, v(0,:,:), res)

;-- middle right plot
res@cnFillPalette = "WhiteGreen"

plot(3) = gsn_csm_contour_map(wks, sftlf, res)

;-- set resources for vector plot
vres = True
vres@gsnDraw = False
vres@gsnFrame = False
vres@gsnAddCyclic = True

vres@gsnLeftString = "Vector Wind"
vres@vcRefAnnoOrthogonalPosF = -1.0
vres@vcRefMagnitudeF = 10.0
vres@vcRefLengthF = 0.045
vres@vcGlyphStyle = "CurlyVector"
vres@vcMinDistanceF = 0.017

;-- lower left plot
plot(4) = gsn_csm_vector_map(wks, u(0,:,:4,:,:4), v(0,:,:4,:,:4), vres)

;-- lower right plot
;-- set the viewport to the same size as plot(4)
getvalues plot(4)
"vpWidthF" : vpw
"vpHeightF" : vph
end getvalues
res@mpRightCornerLatF    = 73.0

plot(5) = gsn_csm_contour_map(wks,t(0,:,:),res)

;-- plot one title on top of the plot
pnlres                = True
pnlres@txString      = "NCL Doc Example: panel 3x2"
;-- FYI: from version 6.4.0 gsnPanelMainString replace the txString here

;-- create panel plot
  gsn_panel(wks,plot(/3,2/),pnlres)
end

---

NCL Doc Example: panel 3x2

---

8.10.1  Control Panel Plots

There are just a few resources that can be used to control a panel plot. The next example shows a panel plot containing plots of different sizes. The "gsnPanelScalePlotIndex" allows you to indicate the index number of which plot to base the scaling on. If your plots are different sizes, then you want to use the index of the plot that has the largest width or height.

NUG_panel_control.ncl:

begin
;-- read data and set variable references
  diri = "./"
  fili = "tas_rectilinear_grid_2D.nc"
  f    = addfile(diri+fili, "r")
  t    = f->tas
;-- open a PNG file
wks = gsn_open_wks("png","plot_panel_control")

;-- set resources for contour plots
res = True
res@gsnDraw = False
res@gsnFrame = False
res@gsnAddCyclic = True
res@gsnLeftStringOrthogonalPosF = 0.03
res@gsnRightStringOrthogonalPosF = 0.03
res1 = res
res2 = res
res3 = res
res4 = res
res5 = res

;-- global
plot_1 = gsn_csm_contour_map(wks,t(0,:,:),res1)

;-- North America
res2@mpMinLatF = 10.0
res2@mpMaxLatF = 80.0
res2@mpMinLonF = -175.0
res2@mpMaxLonF = -50.0
plot_2 = gsn_csm_contour_map(wks,t(0,:,:),res2)

;-- Africa
res3@mpMinLatF = -40.0
res3@mpMaxLatF = 40.0
res3@mpMinLonF = -20.0
res3@mpMaxLonF = 50.0
plot_3 = gsn_csm_contour_map(wks,t(0,:,:),res3)

;-- South America
res4@mpMinLatF = -60.0
res4@mpMaxLatF = 15.0
res4@mpMinLonF = -100.0
res4@mpMaxLonF = -30.0
plot_4 = gsn_csm_contour_map(wks,t(0,:,:),res4)

;-- Europe
res5@mpMinLatF = 35.0
res5@mpMaxLatF = 80.0
res5@mpMinLonF = -20.0
res5@mpMaxLonF = 50.0
plot_5 = gsn_csm_contour_map(wks,t(0,:,:),res5)

;-- create the panel plot
pnlres = True
pnlres@gsnMaximize = True
pnlres@gsnPanelScalePlotIndex = 3
pnlres@gsnPanelTop = 0.94
pnlres@gsnPanelBottom = 0.001
pnlres@gsnPanelXWhiteSpacePercent = 0
pnlres@gsnPanelXWhiteSpacePercent = 30
pnlres@txFontHeightF = 0.020 ;-- text font size
pnlres@txString = "NCL Doc Example: panel control"

;-- FYI: from version 6.4.0 gsnPanelMainString replace the txString here
Sometimes you will get unexpected results trying to plot plots of different size using gsn_panel. An alternative way to place the plots is the use of @vp resources.

NUG_panel_vp.ncl:

```ncl
begin
;-- read data and set variable references
  diri = "/"
  fili = "tas_rectilinear_grid_2D.nc"
  f = addfile(diri+fili, "r")
  t = f->tas

;-- open a PNG file
  wks = gsn_open_wks("png","plot_panel_vp")

;-- set resources for contour plots
  res@gsnDraw = True
  res@gsnFrame = False
  res@gsnAddCyclic = True
  res@gsnLeftStringOrthogonalPosF = 0.03
  res@gsnRightStringOrthogonalPosF = 0.03
  res@vpWidthF = 0.4
  res@vpHeightF = 0.27

;-- global
  res@vpXF = 0.08
  res@vpYF = 0.99

  plot_1 = gsn_csm_contour_map(wks,t(0,:,:),res)
draw(plot_1)
end
```
;-- North America
res@vpXF = 0.55
res@vpYF = 0.99
res@mpMinLatF = 10.0
res@mpMaxLatF = 80.0
res@mpMinLonF = -175.0
res@mpMaxLonF = -50.0

plot_2 = gsn_csm_contour_map(wks,t(0,:,:),res)
draw(plot_2)

;-- Africa
res@vpXF = 0.1
res@vpYF = 0.67
res@mpMinLatF = -40.0
res@mpMaxLatF = 40.0
res@mpMinLonF = -20.0
res@mpMaxLonF = 50.0

plot_3 = gsn_csm_contour_map(wks,t(0,:,:),res)
draw(plot_3)

;-- South America
res@vpXF = 0.55
res@vpYF = 0.67
res@mpMinLatF = -60.0
res@mpMaxLatF = 15.0
res@mpMinLonF = -100.0
res@mpMaxLonF = -30.0

plot_4 = gsn_csm_contour_map(wks,t(0,:,:),res)
draw(plot_4)

;-- Europe
res@vpXF = 0.35
res@vpYF = 0.33
res@mpMinLatF = 35.0
res@mpMaxLatF = 80.0
res@mpMinLonF = -20.0
res@mpMaxLonF = 50.0

plot_5 = gsn_csm_contour_map(wks,t(0,:,:),res)
draw(plot_5)

;-- advance the frame
frame(wks)
8.11 Polylines, Polygons, Polymarkers, Text (primitives)

To draw lines, polygons, markers, or text strings on a plot or the NCL frame, NCL provides the gsn functions:

- `gsn_polyline(wks, plot, x, y, resource)`
- `gsn_polyline_ndc(wks, x, y, resource)`
- `gsn_add_polyline(wks, plot, x, y, resource)`
- `gsn_polygon(wks, plot, x, y, resource)`
- `gsn_polygon_ndc(wks, x, y, resource)`
- `gsn_add_polygon(wks, plot, x, y, resource)`
- `gsn_polymarker(wks, plot, x, y, resource)`
- `gsn_polymarker_ndc(wks, x, y, resource)`
- `gsn_add_polymarker(wks, plot, x, y, resource)`
- `gsn_add_text(wks, plot, text_string, x, y, resource)`
- `gsn_text(wks, plot, text_string, x, y, resource)`
- `gsn_text_ndc(wks, text_string, x, y, resource)`

The “ndc” functions draw primitives on the NCL canvas using NDC coordinates (values that go from 0 to 1). All the other functions draw primitives on the given plot, using that plot’s data space. The gsn_add_xxx functions are identical to the gsn_xxx procedures, except the “add” functions actually attach the primitives to the given plot. The “add” functions are especially useful if you plan to resize the plot later, say in a call to gsn_panel, because the primitives will also be automatically resized.

See also: [http://www.ncl.ucar.edu/Applications/polyg.shtml](http://www.ncl.ucar.edu/Applications/polyg.shtml)

The next example shows how to plot polylines, polygons and polymarkers on a map plot and add a polygons via a procedure.

**NUG_polyline_polygon_polymarker.ncl:**

```ncl
undef("add_poly")
procedure add_poly(wks,map)
local xx, yy, pres
begin
    xx = (/ -75., -10., -10., -75., -75./) ;-- define polygon x-array
    yy = (/ 55., 55., 87., 87., 57./) ;-- define polygon y-array

    pres = True
    pres@gsFillColor = "orange" ;-- fill color
    pres@gsFillOpacityF = 0.2 ;-- set opacity

    ;-- add polygon to map
    map@polygon = gsn_add_polygon(wks, map, xx, yy, pres)
end

;----------
; MAIN
;----------
begin
    ;-- open a workstation and define color map
```
wks = gsn_open_wks("png","plot_polystuff")

;-- set resources
res = True
res@gsnDraw = False ;-- don't draw the plot yet
res@gsnFrame = False ;-- don't advance the frame yet
res@vpXF = 0.08 ;-- x-position
res@vpYF = 0.92 ;-- y-position
res@vpWidthF = 0.88 ;-- width
res@vpHeightF = 0.65 ;-- height
res@mpFillOn = True
map = gsn_csm_map(wks,res) ;-- create the map, but don't draw it yet

;-- write strings at the bottom of the plot
txres = True
txres@txFontHeightF = 0.014 ;-- default size is HUGE!
txres@txFontColor = "blue"
txres@txJust = "CenterLeft" ;-- puts text on top of bars
dty = 0.3
gsn_text_ndc(wks,"Marker", 0.1, dty, txres)
txres@txFontColor = "red"
gsn_text_ndc(wks,"Polyline", 0.2, dty, txres)
txres@txFontColor = "green"
gsn_text_ndc(wks,"Polygon transparent", 0.3, dty, txres)

;-- polyline
x = (/ 6., 15., 15., 6., 6./)
y = (/47.5, 47.5, 54.5, 54.5, 47.5/)

;-- trace: polyline resources
plres = True
plres@gsLineThicknessF = 2.0 ;-- set line thickness
plres@gsLineColor = "red" ;-- set line color
box_1 = gsn_add_polyline(wks, map, x, y, plres) ;-- add polyline to map

;-- define polygon x- and y-arrays
x = (/110., 160., 160., 110., 110./)
y = (/45., -45., -45., -45., 45./)

;-- trace: polygon resources
pgres = True
pgres@gsFillColor = "green" ;-- fill color
pgres@gsFillOpacityF = 0.3 ;-- set opacity of polygon
gon_1 = gsn_add_polygon(wks, map, x, y, pgres) ;-- add filled polygon to map

;-- polymarker
pmres = True
pmres@gsMarkerColor = "blue" ;-- marker color
pmres@gsMarkerIndex = 1 ;-- use marker 1
pmres@gsMarkerSizeF = 0.03 ;-- set size of marker
pmres@gsLineThicknessF = 3. ;-- marker line thickness

;-- unique identifier name for polymarker drawing, here marker_1
marker_1 = gsn_add_polymarker(wks, map, -100., 30., pmres)

;-- draw all 16 marker on plot using unique identifier name and
;-- additional map attribute settings
x = -160. ;-- x-position of first marker
\[ y = -80. \quad ; \quad \text{-- y-position of first marker} \]

\[
\text{do i = 0,15} \quad ; \quad \text{-- 16 different marker}
\]

\[
\text{pmres@gsMarkerIndex} = i+1 \quad ; \quad \text{-- result is poly0-poly15}
\]

\[
\text{str = unique_string("poly")} \quad ; \quad \text{-- result is poly0-poly15}
\]

\[
\text{map@str$ = gsn_add_polymarker(wks, map, x+(i*20.), y+(i*10.), pmres)} \quad ; \quad \text{-- add marker to map}
\]

\[
\text{end do}
\]

\[
\text{add polygon calling the procedure add_poly()}
\]

\[
\text{add_poly(wks,map)}
\]

\[
\text{print map contents to see how unique_string works with map@str$}
\]

\[
\text{print(map)}
\]

\[
\text{create the plot and advance the frame}
\]

\[
\text{draw(map)}
\]

\[
\text{frame(wks)}
\]

\[
\text{end}
\]

**Tip:** Using polylines, polygons or polymarkers within a user defined function or procedure can cause problems. If plotting information of the poly*** routines are getting lost you can save them by assigning an additional attribute to the graphics plot object.

**Example user defined function 'add_poly':**

```
undef("add_poly")
procedure add_poly(wks,plot)
local xx, yy, pres
begin
    xx = (/ -75., -10., -10., -75., -75. /)
    yy = (/ 55., 55., 87., 87., 57. /)

    pres = True
    pres@gsFillColor = "orange"
    pres@gsFillOpacityF = 0.2

    plot@polygon = gsn_add_polygon(wks, plot, xx, yy, pres)
end
```
8.12 Shapefile Plots

Shapefiles are a popular geospatial vector data format for GIS software. They contain points, lines and polygons representing entities like rivers, lakes, countries, counties, population of cities, locations of popular landmarks, and so on.

Some useful data sets can be found at:
http://www.gadm.org/
http://www.nws.noaa.gov/geodata/
http://www.geodatenzentrum.de/geodaten/gdz_rahmen.gdz_div

To attach shapefile points, lines, or polygons to a plot, NCL provides the gsn functions

- `gsn_add_shapefile_polymarkers(wks, plot, filename, resource)`
- `gsn_add_shapefile_polylines(wks, plot, filename, resource)`
- `gsn_add_shapefile_polygons(wks, plot, filename, resource)`

The next example plots outlines from a Germany shapefile downloaded from gadm.org. This example also uses the "HighRes" map database to get high resolution coastal outlines for other areas. See the "Map Resolutions" section for information on how to download this database.

Simple shapefile example: NUG_shapefile_plot.ncl

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
  diri = "/",
  fili = "DEU_adm/DEU_adm1.shp"
  f    = addfile(diri+fili, "r")

  -- read data from shapefile
  segments = f->segments
  geometry = f->geometry
  segsDims = dimsizes(segments)
  geomDims = dimsizes(geometry)

  -- get global attributes
  geom_segIndex = f@geom_segIndex
  geom_numSegs = f@geom_numSegs
  segs_xxyzIndex = f@segs_xxyzIndex
  segs_numPnts = f@segs_numPnts
  numFeatures   = geomDims(0)

  -- open workstation
  wks = gsn_open_wks("png","plot_shapefile_plot")

  -- set resources for the map
  res = True
  res@gsnDraw  = False        ;-- don't draw the plot
  res@gsnFrame = False        ;-- don't advance frame yet
  res@gsnMaximize= True       ;-- maximize plot in frame
  res@mpDataBaseVersion = "HighRes"
  res@mpDataResolution  = "Medium"
  res@mpProjection    = "Mercator"        ;-- change projection

  -- select coordinates for Germany
```
res@mpLimitMode = "Corners"
res@mpLeftCornerLatF = 47.
res@mpRightCornerLatF = 55.
res@mpLeftCornerLonF = 5.
res@mpRightCornerLonF = 16.
res@tiMainString = "NCL Doc Example: Shapefile plot"
res@tiMainFontHeightF = 0.015

`;-- generate map, but don't draw it
plot = gsn_csm_map(wks,res) ;-- draw map, but don't advance frame

`;-- add polylines from the shape files to the plot
lines = new(segsDims(0),graphic) ;-- array to hold shapefile polylines

`;-- set resources for the polylines
plres = True
plres@gsLineColor = "red"

`;-- generate polylines, but don't draw it
lon = f->x
lat = f->y
segNum = 0
do i=0, numFeatures-1
   startSegment = geometry(i, geom_segIndex)
   numSegments = geometry(i, geom_numSegs)
   do seg=startSegment, startSegment+numSegments-1
      startPT = segments(seg, segs_xyzIndex)
      endPT = startPT + segments(seg, segs_numPnts) - 1
      lines(segNum) = gsn_add_polyline(wks, plot, lon(startPT:endPT),
         lat(startPT:endPT), plres)
      segNum = segNum + 1
   end do
end do

`;-- draw the plot
draw(plot)
frame(wks)
end
The next example will show how to combine a contour fill plot of your data and the content of a shapefile related to the latitudes and longitudes.

**Simple shapefile example: NUG_shapefile_plot_data.ncl**

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

begin
    diri  = "./"
    shpname = "DEU_adm/DEU_adm1.shp"

    ;-- read data to display
    fili = "rectilinear_grid_2D.nc"
    g    = addfile(diri+fili,"r")
    var  = g->tsurf(0,:,:)

    ;-- open workstation
    wks = gsn_open_wks("png","plot_shapefile_plus_data")

    ;-- set resources for the map
    res                   =  True
    res@gsnDraw           =  False ;-- don't draw the plot
    res@gsnFrame          =  False ;-- don't advance frame yet
    res@gsnMaximize       =  True  ;-- maximize plot in frame
    res@gsnSpreadColors   =  True  ;-- full color map
    res@gsnSpreadColorStart = 18   ;-- start at color 14
    res@gsnSpreadColorEnd = -3

    res@cnFillOn          =  True
    res@cnLinesOn         =  False
    res@cnSmoothingOn     =  True
    res@cnLevelSelectionMode = "ManualLevels"
    res@cnMinLevelValF    =  270.0
    res@cnMaxLevelValF    =  285.0
    res@cnLevelSpacingF   =    1.0

    ;-- select coordinates for Germany
    res@mpFillOn          =  False
    res@mpLimitMode       = "Corners"
    res@mpLeftCornerLatF  =  47.
    res@mpRightCornerLatF =  55.
    res@mpLeftCornerLonF  =  5.
    res@mpRightCornerLonF =  16.
    res@mpDataBaseVersion = "HighRes"
    res@mpDataResolution  = "Medium"
    res@mpProjection      = "Mercator"

    res@tiMainString = "NCL Doc Example: Using Shapefile for borderlines"
    res@tiMainFontHeightF =  0.015

    ;-- generate map, but don't draw it
    plot = gsn_csm_contour_map(wks,var,res)

    ;-- set resources for the polylines
    plres                   =  True
    plres@gsLineColor       = "black"

    poly = gsn_add_shapefile_polylines(wks,plot,shpname,plres)

    ;-- draw the plot
```
draw(plot)
frame(wks)
end

NCL Doc Example: Using Shapefile for borderlines

surface temperature

270 272 274 276 278 280 282 284

K
8.13 Color Maps

Many different color tables, also called color maps, are available. Alternatively, you can define your own color maps or convert color tables from other graphic packages, such as GrADS, into your own RGB (= Red Green Blue) or RGBA (= Red Green Blue and Alpha, the transparency channel) color map in order to use it within NCL.

At this point, only a short description of how to handle different color maps in NCL is given. A more detailed description can be found on the web page


The next example uses both gsn_define_colormap and cnFillPalette to set the color map for the filled contours. Using cnFillPalette is the preferred method.

Simple color map example: NUG_colormaps.ncl

```plaintext
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
  diri  = "/"
  fili  = "rectilinear_grid_2D.nc"
  file1 =  addfile(diri+fili, "r")
  var   =  file1->tsurf(0,:,:);

  -- set resources
  res                       =  True
  res@gsnMaximize           =  True
  res@cnFillOn              =  True       ;-- turn on contour fill
  res@tiMainString         = "NCL Doc Example: Color maps" ;-- title
  res@tiMainFontHeightF     =  0.02

  ;-- 1: set color map to "ncl_default" and draw the contour map
  gsn_define_colormap(wks,"ncl_default")
  plot = gsn_csm_contour_map(wks, var, res)

  ;-- 2: change the color map to "rainbow" and draw the contour map
  gsn_define_colormap(wks,"rainbow")
  plot = gsn_csm_contour_map(wks, var, res)

  ;-- 3: change the color map to "BlueRed" and draw the contour map
  gsn_define_colormap(wks,"BlueRed")
  plot = gsn_csm_contour_map(wks, var, res)

  ;-- 4: color map set by resource cnFillPalette instead of
  ;--  gsn_define_colormap
  res@cnFillPalette       = "OceanLakeLandSnow"
  plot = gsn_csm_contour_map(wks, var, res)

  ;-- 5: draw just the color map - no data. !! Uses the last
  ;--  gsn_define_colormap setting to wks
  gsn_define_colormap(wks,"BlGrYeOrReVi200")
  gsn_draw_colormap(wks)
end
```
If you don’t want to use the complete color map it can be read into an array and a subset of the array can be set as color map.

Old fashioned way:

```plaintext
gsn_define_colormap(wks,"ncl_default")
res@gsnSpreadColorStart = 14 ;-- color index start
res@gsnSpreadColorEnd = -8 ;-- color index end
```

Recommended way:

```plaintext
cmap = read_colormap_file("ncl_default")
res@cnFillPalette = cmap(14,247,;)
```

### 8.13.1 Converting a GrADS color table

This shell script converts a GrADS color table to an NCL color map:

`grads2ncl_coltab.ksh`

```ksh
#!/bin/ksh
#
#---------------------------------------------------
#-- convert GrADS color table to NCL color map (RGB)
#-- Usage: grads2ncl_coltab.ksh <GrADS color table file>
#---------------------------------------------------

in=$1
gscoltab=${in##*/}
coltab=NCL_${gscoltab%.*}.rgb

ncols=$(cat ${in} | grep -v "\*" | grep -v "\#" | wc -l)
ncols=$(expr ${ncols} + 2)

cat << EOF > ${coltab}
ncolors=${ncols}
# r   g   b
0 0 0
1 1 1
EOF
cat ${in} | grep -v "\*" | grep -v "\#" | sed -e "s/\'/g" | \awk '{print $4 "$5" "$6"}'' >> ${coltab}
exit
```

### 8.13.2 Converting a GMT color table

Since NCL release (6.2.0) many GMT color tables are imported to NCL.

This shell script can be used to convert a private GMT color table to an NCL color map:

`gmt2coltab.ksh`

```ksh
#!/usr/bin/ksh
#
#-----------------------------------------------
#-- KSH - NCL Doc Example script:
#--
```
# convert GMT color tables to NCL color tables
# used by  gsn_define_colormap(wks,"NCL_GMT-BYR-03")
# or cnFillPalette
# Example GMT color table file:  GMT-BYR-03.cpt
# Usage:  gmt2coltab.ksh GMT-BYR-03.cpt
#  ---> creates new file NCL_GMT-BYR-03.rgb
# KMF 03.05.13
#---------------------------------------------------------------
in=$1
gmt_coltab=${in##*/}
coltab=NCL_${gmt_coltab%.*}.rgb

# count number of colors plus foreground
# and background color.
# BUT, delete the last 3 color entries from the GMT color table
# they're not needed -- number of colors + 2 - 3 = number of colors -1
ncols=$(cat ${in} | grep -v "\#" | wc -l)
ncols=$(expr $ncols - 1)

# insert background (1) and foreground (0) colors
# NCL starts with color index 2
cat << EOF > tmp_col.rgb
ncolors=${ncols}
# r   g   b
0 0 0
1 1 1
EOF

# read and write the color values
cat ${in} | grep -v "\#" | sed -e "s/\//g" | awk '{print $2" "$3" "$4}' >> tmp_col.rgb
head -n -3 tmp_col.rgb > ${coltab}
\rm -rf tmp_col.rgb
exit

8.14 Curvilinear Grids

Curvilinear grids are those represented by two-dimensional latitude/longitude arrays. There
are two methods plotting curvilinear grids. One method is to set the special sfXArray and
sfYArray resources to the two-dimensional longitude and latitude arrays, respectively. A
second method is to add attributes "lat2d" and "lon2d" to your data variable, and set them to
the two-dimensional latitude and longitude arrays. If your data is regional, then you will also
need to set gsnAddCyclic to False. Generally, the second method is preferred, because the
first method doesn’t recognize the gsnAddCyclic resource.

The example script below will show you how to get a quick view of your curvilinear data file.

NUG_curvilinear_basic.ncl:

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
  diri = "./"
```
```ncl
fill = "tos_ocean_bipolar_grid.nc"

f = addfile(diri+fili, "r")
var = f->tos(0,:,:)
var@lat2d = f->lat
var@lon2d = f->lon

wks = gsn_open_wks("png","plot_curvilinear_basic")
res = True ;-- set resources
res@gsnAddCyclic = False ;-- lon < 360 degrees
res@cnFillOn = True ;-- turn on contour fill
res@cnFillPalette = "BlueWhiteOrangeRed" ;-- change color map

res@tiMainString = "Curvilinear grid:  MPI-ESM (2D lat/lon arrays)"

plot = gsn_csm_contour_map(wks,var,res) ;-- create the plot
end
```

**Curvilinear grid example: NUG_curvilinear_grid.ncl**

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

begin
  diri = "./"
  fili = "CNT ASN_1m_200103_grid_T_curvilinear_grid.nc"
  f = addfile(diri+fili,"r")
  var = f->votemper(0,0,:,:)
  var@lat2d = f->nav_lat ;-- 2D latitudes
  var@lon2d = f->nav_lon ;-- 2D longitudes

  ;-- define the workstation (plot type and name)
```
```ncl
wks = gsn_open_wks("png","plot_curvilinear_grid")

;-- set resources
res = True
res@gsnAddCyclic = False ;-- don't add lon cyclic point
res@gsnMaximize = True
res@cnFillOn = True ;-- turn on contour fill
res@cnMinLevelValF = 5.
res@cnLevelSpacingF = 0.5
res@cnLinesOn = False
res@tiMainString = "NCL Doc Example: Curvilinear grid (NEMO)" ;-- title
res@tiMainFontHeightF = 0.02

;-- Zoom in on map
res@mpMinLatF = min(var@lat2d)
res@mpMaxLatF = max(var@lat2d)
res@mpMinLonF = min(var@lon2d)
res@mpMaxLonF = max(var@lon2d)

res@pmTickMarkDisplayMode = "Always" ;-- nicer tickmarks

;-- draw the contour map
plot = gsn_csm_contour_map(wks,var,res)
end
```

![NCL Doc Example: Curvilinear grid (NEMO)](image_url)
8.14.1 MPI-ESM-LR

MPI-ESM bipolar grid example (MPI-OM TP04): NUG_bipolar_grid_MPI-ESM.ncl

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

begin
  diri = "/"
  fili = "tos_ocean_bipolar_grid.nc"
  f   = addfile(diri+fili, "r")
  tos = f->tos
  tos@lat2d = f->lat
  tos@lon2d = f->lon
  var   = tos(0,:,:)

  ;-- define the workstation (plot type and name)
  wks = gsn_open_wks("png","plot_bipolar_grid_MPI-ESM")

  ;-- set resources
  res       = True
  res@gsnMaximize = True
  res@gsnAddCyclic = True
  res@cnFillOn     = True ;-- turn on contour fill
  res@cnFillPalette = "ncl_default"
  res@cnFillMode    = "CellFill"
  res@cnLinesOn    = False ;-- Turn lines off
  res@cnLineLabelsOn = False ;-- Turn labels off
  res@cnCellFillEdgeColor = 1
  res@cnCellFillMissingValEdgeColor = "black"

  res@tiMainString = "NCL Doc Example: Bipolar grid MPI-ESM" ;-- title
  res@tiMainFontHeightF = 0.02

  ;-- draw the contour map
  plot = gsn_csm_contour_map(wks,var,res)
end
```

NCL Doc Example: Bipolar grid MPI-ESM

Sea Surface Temperature

![Sea Surface Temperature](image)

129
Plot a sub-region of the MPI-ESM bipolar grid example (MPI-OM TP04): NUG_bipolar_grid_MPI-ESM_subregion.ncl

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

begin
    -- read the data and define
    diri = "/"
    fill = "tos_ocean_bipolar_grid.nc"
    f = addfile(diri+fill,"r")

    tos = f->tos
    tos@lat2d = f->lat
    tos@lon2d = f->lon
    var = tos(0,:,:);  // select first time step

    -- define the workstation (plot type and name)
    wks = gsn_open_wks("png","bipolar_grid_MPI-ESM_subregion")

    -- set resources
    res = True
    res@gsnMaximize = True     // maximize plot output
    res@gsnAddCyclic = True    // add cyclic point
    res@cnFillOn = True        // turn on contour fill
    res@cnFillMode = "CellFill"  // fill mode
    res@cnFillPalette = "ncl_default"  // choose a color map
    res@cnLinesOn = False      // turn lines off
    res@cnLineLabelsOn = False // turn labels off
    res@cnCellFillEdgeColor = 1 // _FillValue color
    res@mpLimitMode = "Corners"
    res@mpLeftCornerLonF = -95. // min longitude
    res@mpRightCornerLonF = 35. // max longitude
    res@mpLeftCornerLatF = 20. // min latitude
    res@mpRightCornerLatF = 90. // max latitude
    res@mpDataBaseVersion = "MediumRes" // map data base
    res@mpFillOn = False        // turn off map fill
    res@tiMainString = "NCL Doc Example: Bipolar grid MPI-ESM"
    res@tiMainFontHeightF = 0.02

    -- draw the contour map
    plot = gsn_csm_contour_map(wks,var,res)
end
```

NCL Doc Example: Bipolar grid MPI-ESM subregion

Sea Surface Temperature

[Graph showing sea surface temperature with longitude and latitude ranges]
8.14.2  STORM

STORM: MPI-OM TP6M tripolar grid example: NUG_tripolar_grid_STORM.ncl

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

begin
    diri = "/
    fili = "tripolar_grid_STORM.nc"

    f     = addfile(diri+fili, "r")
    var   = f->sst(0,0,:,:)
    var@lat2d = f->lat
    var@lon2d = f->lon

    ;-- define the workstation (plot type and name)
    wks = gsn_open_wks("png","plot_tripolar_grid_STORM")

    plot = new(2,graphic)

    ;-- set resources
    res   = True
    res@gsnDraw = False
    res@gsnFrame = False
    res@gsnMaximize = True

    res@mpProjection = "CylindricalEquidistant" ;-- choose
    ;-- projection
    res@mpDataBaseVersion = "MediumRes"
    res@mpPerimOn = False ;-- turn off box around plot
    res@mpFillOn = False
    res@mpMinLonF = 2.0
    res@mpMaxLonF = 25.0
    res@mpMinLatF = 52.0
    res@mpMaxLatF = 65.0

    res@cnFillOn = True ;-- turn on contour fill
    res@cnFillMode = "CellFill"
    res@cnLinesOn = False ;-- Turn lines off
    res@cnLineLabelsOn = False ;-- Turn labels off
    res@cnCellFillEdgeColor = -1
    res@cnCellFillMissingValEdgeColor = -1

    res@tiMainString    = "NCL Doc Example: Tripolar grid STORM" ;-- title
    res@tiMainFontHeightF = 0.02

    ;-- plot sub-region
    plot(0) = gsn_csm_contour_map(wks,var,res)

    ;-- plot all -180-180 deg.
    res@mpMinLonF = -180.0
    res@mpMaxLonF = 180.0
    res@mpMinLatF = -90.0
    res@mpMaxLatF = 90.0

    delete(res@cnCellFillEdgeColor)
    delete(res@cnCellFillMissingValEdgeColor)
    delete(res@tiMainString)

    res@cnCellFillEdgeColor = -1
    res@cnCellFillMissingValEdgeColor = -1
```

plot(1) = gsn_csm_contour_map(wks, var, res)

;-- create panel plot
  gsn_panel(wks, plot, (/2, 1/), False)
end
8.15 Unstructured Grids

Unstructured grids are typically defined as points or cells, and consist of one-dimensional arrays of values, latitude points, and longitude points. To plot this data correctly you need to set the special sfXArray and sfYArray resources to the one-dimensional longitude and latitude arrays, respectively.

Unstructured grid example: NUG_unstructured_grid.ncl

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

begin
diri  = "./"
fil  = "camse_unstructured_grid.nc"
f     = addfile(diri+fil, "r")
var   = f->T850
lat1d = f->lat
lon1d = f->lon

;-- define the workstation (plot type and name)
wks = gsn_open_wks("png","plot_unstructured_grid_camse")

;-- set resources
res                      =  True
res@gsnMaximize           =  True
res@cnFillOn             =  True
res@cnFillPalette        =  "BlueWhiteOrangeRed"
res@tiMainString         =  "NCL Doc Example: Unstructured grid (CAM-SE)";-- title
res@tiMainFontHeightF    =  0.02

;-- Lat/lon arrays of curvilinear grid for overlaying on map
res@sfXArray              =  lon1d
res@sfYArray              =  lat1d

;-- draw the contour map
plot = gsn_csm_contour_map(wks,var,res)
end
```

![NCL Doc Example: Unstructured grid (CAM-SE)](image)

Temperature at 850 mbar pressure surface
8.15.1 ICON

ICON icosahedral grid example: NUG_triangular_grid_ICON.ncl

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/sheu_util.ncl"

begin

        start_code_time = get_cpu_time()

        diri = "./"
        fili = "triangular_grid_ICON.nc"
        f = addfile(diri+fili, "r")
        var = f->S(0,2,:)

        var@_FillValue = getVarFillValue(var) ;-- retrieve missing value of var
        var_min = 34.5 ;-- minimum value to be displayed
        var_max = 36.1 ;-- maximum value to be displayed
        var_inc = 0.1 ;-- increment
        lon_min = -85.0 ;-- minimum longitude
        lon_max = -30.0 ;-- maximum longitude
        lat_min = 15.0 ;-- minimum latitude
        lat_max = 70.0 ;-- maximum latitude

        rad2deg = get_r2d("float") ;-- radians to degrees
        x = f->clon * rad2deg ;-- cell center, lon
        y = f->clat * rad2deg ;-- cell center, lat

        ;-- if variable wet_c exist: create missing values with it:
        if (isfilevar(f, "wet_c")) then
            wet = f->wet_c(2,:) ;-- time=0, depth=0; cells
            var = where ( wet .ge. 0.01, var, -999.) ;-- missing value
            var@_FillValue = -999. ;-- missing value
        else
            print("WARNING: variable wet_c doesn't exist -- no masking")
        end if

        ;-- calculate the latitude and longitude values
        vlon = f->clon_vertices * rad2deg
        vlon = where(vlon.lt.0, vlon + 360, vlon) ;-- lon: 0 - 360
        vlat = f->clat_vertices * rad2deg

        wks = gsn_open_wks("png", "plot_ICON_data") ;-- open a workstation

        res@gsnDraw = True ;-- don't draw the plot yet
        res@gsnFrame = False ;-- don't advance the frame
        res@gsnLeftString = "" ;-- don't add variable name to plot
        res@gsnRightString = "" ;-- don't add units to plot
        res@gsnMaximize = True ;-- maximize plot output
        res@tiMainString = "NCL Doc Example: ICON - Salinity [psu]" ;-- title
        res@tiMainFontHeightF = 0.02
        res@pmTitleZone = 2
        res@mpFillOn = True ;-- fill map grey
        res@mpFillDrawOrder = "PostDraw" ;-- draw map outline at last
        res@mpDataBaseVersion = "MediumRes" ;-- map resolution
        res@mpMinLonF = lon_min ;-- sub-region minimum longitude
        res@mpMaxLonF = lon_max ;-- sub-region maximum longitude
        res@mpMinLatF = lat_min ;-- sub-region minimum latitude
        res@mpMaxLatF = lat_max ;-- sub-region maximum latitude
res@mpGreatCircleLinesOn = True ;-- important
res@sfXArray = x ;-- longitude grid cell center
res@sfYArray = y ;-- latitude grid cell center
res@cnFillOn = True ;-- contour fill
res@cnFillPalette = "BlueWhiteOrangeRed" ;-- choose color map
res@cnLinesOn = False ;-- don't draw contour lines
res@cnFillMode = "RasterFill" ;-- contour fill mode
res@cnLevelSelectionMode = "ManualLevels" ;-- set manual contour levels
res@cnMinLevelValF = var_min ;-- set min contour level
res@cnMaxLevelValF = var_max ;-- set max contour level
res@cnLevelSpacingF = var_inc ;-- set increment
res@pmTickMarkDisplayMode = "Always" ;-- nicer tickmarks
plot = gsn_csm_contour_map(wks,var,res) ;-- create the plot, but don't ;-- draw it

;-- retrieve contour level and color informations
getvalues plot@contour
  "cnLevels" : levels ;-- # 26 (n)
  "cnFillColors" : colors ;-- # 27 (n+1)
end getvalues

plot = setColorContourClear(plot,min(var),max(var)) ;-- clear plot, but ;-- keep all the information

;-- create color array for triangles
ntri = dimsizes(y) ;-- Number of triangles
gscolors = new(ntri,integer)
gscolors = -1 ;-- Initialize to transparent

;-- set resources for the triangles (polygons)
pres = True
pres@gsEdgesOn = True ;-- turn on edges
pres@gsFillIndex = 0 ;-- solid fill

;-- set color for data less than given minimum value var_min
vlow = ind(var .lt. levels(0)) ;-- get the indices of values less levels(0)
gscolors(vlow) = colors(0) ;-- choose color
ntri_calc = dimsizes(vlow) ;-- number of triangles

;-- set colors for all cells in between var_min and var_max
do i = 1, dimsizes(levels) - 1
  vind := ind(var .ge. levels(i-1) .and. var .lt. levels(i)) ;-- get the ;-- indices of 'middle' values
  gscolors(vind) = colors(i) ;-- choose the colors
  ntri_calc = ntri_calc + dimsizes(vind) ;-- number of triangles
end do

;-- set color for data greater than given maximum var_max
nc = dimsizes(colors)-1 ;-- get the number of colors minus one
nl = dimsizes(levels)-1 ;-- get the number of levels minus one
vhgh := ind(var .gt. levels(nl)) ;-- get indices of values greater ;-- levels(nl)
gscolors(vhgh) = colors(nc) ;-- choose color
ntri_calc = ntri_calc + dimsizes(vhgh) ;-- number of triangles

print("--- triangles calculated: "+ ntri_calc)

;-- Attach all the triangles using the list of colors
pres@gsColors = gscolors
pres@gsSegments = ispan(0, dimsizes(var) * 3, 3) ; -- assign segments array
polygon = gsn_add_polygon(wks, plot, ndtooned(vlon), ndtooned(vlat), pres)
               ; -- draw all triangles
draw(plot) ; -- draw plot and attached filled triangles
frame(wks) ; -- advance the frame
end_code_time = get_cpu_time()
print("---> Elapsed time in CPU seconds: "+(end_code_time-
start_code_time))
end
8.16 Rotated Grids

Some models compute their data on a rotated latitude-longitude grid, which means that the poles of the grid are shifted. The information about the rotation is sometimes stored in the netCDF or GRIB file, for example, as attributes called "Longitude_of_southern_pole" and "Latitude_of_southern_pole" or "grid_north_pole_longitude" and "grid_north_pole_latitude".

The output of the 'ncdump –h' command of a rotated grid file, here CORDEX EUR-11:

```bash
> ncdump -h tas_rotated_grid_EUR11.nc
netcdf tas_rotated_grid_EUR11 {
  dimensions:
    rlon = 424 ;
    rlat = 412 ;
    height = 1 ;
    time = UNLIMITED ; // (60 currently)
    tbnds = 2 ;
  variables:
    double rlon(rlon) ;
      rlon:standard_name = "grid_longitude" ;
      rlon:long_name = "rotated longitude" ;
      rlon:units = "degrees" ;
      rlon:axis = "X" ;
    double rlat(rlat) ;
      rlat:standard_name = "grid_latitude" ;
      rlat:long_name = "rotated latitude" ;
      rlat:units = "degrees" ;
      rlat:axis = "Y" ;
    char rotated_pole ;
      rotated_pole:grid_mapping_name = "rotated_latitude_longitude" ;
      rotated_pole:grid_north_pole_latitude = 39.25 ;
      rotated_pole:grid_north_pole_longitude = -162. ;
    double height(height) ;
      height:standard_name = "height" ;
      height:long_name = "height" ;
      height:units = "m" ;
      height:positive = "up" ;
      height:axis = "Z" ;
    double time(time) ;
      time:standard_name = "time" ;
      time:bounds = "time_bnds" ;
      time:units = "days since 1949-12-01 00:00:00" ;
      time:calendar = "proleptic_gregorian" ;
      time:long_name = "time" ;
    double time_bnds(time, tbnds) ;
      time_bnds:units = "days since 1949-12-01 00:00:00" ;
      time_bnds:long_name = "time bounds" ;
    float tas(time, height, rlat, rlon) ;
      tas:standard_name = "air_temperature" ;
      tas:long_name = "Near-Surface Air Temperature" ;
      tas:units = "K" ;
      tas:grid_mapping = "rotated_pole" ;
      tas:original_name = "T_2M" ;
      tas:cell_methods = "time: mean" ;
      tas:_FillValue = 1.e+20f ;
      tas:missing_value = 1.e+20f ;
}
```

If you try to plot this data using the given rlat/rlon grid it will incorrectly show you that the map sub-region is west of Africa and not in the correct part of Europe.
8.16.1 Plotting on the native grid

One of the keys to plotting rotated grids correctly on their native map projection is that you must know the correct center longitude and latitude of the data. This information may appear as metadata in your NetCDF or GRIB file, for example, as attributes "grid_north_pole_latitude" and "grid_north_pole_longitude" attached to a variable called "rotated_pole". To get the center lat/lon rotation correct in NCL, the formula for this grid is:

\[
\text{res@mpCenterLatF} = 90 - \text{rotated_pole@grid_north_pole_latitude} \\
\text{res@mpCenterLonF} = 180 + \text{rotated_pole@grid_north_pole_longitude}
\]

The first frame of the example below sets the \texttt{tfDoNDCOverlay} resource to True to tell NCL we are plotting the data natively. No data transformation will take place.

The second frame shows how to plot the data using the 2D lat/lon coordinates provided on the file. This data is plotted over a polar stereographic projection over the North Pole.

\begin{verbatim}
;****************************************************
; rotatedltln_2.ncl
; Concepts illustrated:
; - Drawing filled contours over a rotated lat-lon grid
; - Drawing a map using the medium resolution map outlines
; - Overlaying contours on a map without having lat,lon coordinates
; - Overlaying contours on a map using lat,lon coordinates
;****************************************************
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
  albedo_filename = "albedo_a.nc"
  albedo_grid_filename = "albedo_a_curvilinear.nc"

  f1 = addfile(albedo_filename,"r")
  f2 = addfile(albedo_grid_filename,"r")
\end{verbatim}
Variable to plot
sbt = f1->var242

Needed for getting the projection parameters in plot.
lat2d = f2->lat
lon2d = f2->lon
north_pole_lat = f1->rotated_pole@grid_north_pole_latitude ; 6.55
north_pole_lon = f1->rotated_pole@grid_north_pole_longitude ; 0.0

nlat = dimsizes(lat2d(:,0))
nlon = dimsizes(lon2d(0,:))

; Start the graphics section

wks = gsn_open_wks("png", "rotatedltln")

Set resources common to both types of plots we plan to create
res = True ; plot mods desired
res@gsnMaximize = True ; maximize plot in frame
res@cnFillOn = True ; turn on color
res@cnLinesOn = False ; no contour lines
res@cnLineLabelsOn = False ; no contour labels
res@cnFillPalette = "BlGrYeOrReVi200"
res@lbOrientation = "Vertical" ; vertical labelbar
res@pmLabelBarOrthogonalPosF = 0.18 ; move lbar away from plot
res@mpDataBaseVersion = "MediumRes" ; use finer database
res@gsnAddCyclic = False

First frame: use native projection information to plot
; the data.

res_native = res ; Copy over common resources.

Setting tfDoNDCOverlay to True means you have specified the
; exact projection that your data is on, and thus no data
; transformation takes place when the contours are overlaid
; on the map.

res_native@tfDoNDCOverlay = True
res_native@mpLimitMode = "Corners"
res_native@mpLeftCornerLatF = lat2d(0,nlon-1)
res_native@mpLeftCornerLonF = lon2d(0,nlon-1)
res_native@mpRightCornerLatF = lat2d(nlat-1,0)
res_native@mpRightCornerLonF = lon2d(nlat-1,0)
res_native@tiMainString = "Native projection"
res_native@pmTickMarkDisplayMode = "always"
res_native@mpCenterLatF = 90 - north_pole_lat ; north_pole_lat=6.55
res_native@mpCenterLonF = 180 + north_pole_lon ; north_pole_lon=0

plot_native = gsn_csm_contour_map (wks,sbt(0,0,:,:),res_native)
; Second frame: use lat2d/lon2d coordinates to plot
; the data.

res_nonnative = res   ; Copy over common resources
res_nonnative@sfXArray                 = lon2d    ; needed for non-native
res_nonnative@sfYArray                 = lat2d    ; contouring
res_nonnative@gsnPolar                 = "NH"
res_nonnative@mpMinLatF                = min(lat2d)
res_nonnative@pmLabelBarOrthogonalPosF = 0.05
res_nonnative@tiMainString             = "Non-native projection"

plot_nonnative = gsn_csm_contour_map (wks,sbt(0,0,:,:),res_nonnative)
end

8.16.2    Transform rotated to unrotated lat-lon grid

Another way to plot the data on an unrotated grid is to compute the transformation of the rotated to the unrotated grid. In the next example the input data file contains one-dimensional latitude and longitude arrays, which will be used to create the two-dimensional arrays for the unrotated latitudes and longitudes using the rotated pole information.

The upper figure zooms into the map using a Cylindrical Equidistant projection and the lower figure shows the data with an Orthographic projection.

NUG_plot_rotated_grid.ncl:

;;-------------------------------------------------------------------
;; set global constants
;;-------------------------------------------------------------------

deg2rad  = get_d2r("float")
rad2deg  = get_r2d("float")
fillval  = -99999.9

;;-------------------------------------------------------------------
;; Function:      unrot_lon(rotlat,rotlon,pollat,pollon)
;; Description:   transform rotated longitude to longitude
;;-------------------------------------------------------------------

function unrot_lon( rotlat:numeric, rotlon:numeric, pollat[1]:numeric,
pollon[1]:numeric

begin
    lon = fillval
    lon@_FillValue = fillval
    nrlat      = dimsizes(rotlat)
    nrlon      = dimsizes(rotlon)
    nrlat_rank = dimsizes(nrlat)
    nrlon_rank = dimsizes(nrlon)
    if (any(nrlat.ne.nrlon).and.(nrlat_rank.ne.1.or.
    nrlon_rank.ne.1)) then
        print("Function unrot_lon: unrot_lon: rotlat and rotlon dimensions do not match")
        return(lon)
    end if
    if (nrlat_rank.eq.1 .and. nrlon_rank.eq.1) then
        rla = conformDims(/nrlat,nrlon/),rotlat,0)
        rlo = conformDims(/nrlat,nrlon/),rotlon,1)
    else
        rla = rotlat
        rlo = rotlon
    end if
    rla = rla*deg2rad                      ;-- convert from degree to radians
    rlo = rlo*deg2rad                      ;-- convert from degree to radians
    lon := (/rlo/)                         ;-- reassign lon
    lon@_FillValue=fillval
    s1   = sin(pollat*deg2rad)
    c1   = cos(pollat*deg2rad)
    s2   = sin(pollon*deg2rad)
    c2   = cos(pollon*deg2rad)
    tmp1 = s2*(-s1*cos(rlo)*cos(rla)+c1*sin(rla))-
    c2*sin(rlo)*cos(rla)
    tmp2 = c2*(-s1*cos(rlo)*cos(rla)+c1*sin(rla))+s2*sin(rlo)*cos(rla)
    lon  = atan(tmp1/tmp2)*rad2deg        ;-- convert from degree to radians
    lon@Units = "degrees_east"
    print("Function unrot_lon: min/max     "+sprintf("%.4f", 
    min(lon(0,:),+)
    " "+sprintf("%.4f", max(lon(0,:))))
    delete([/rlo,rlo,c1,s1,c2,s2,tmp1,tmp2/])
    return(lon)
end

------------------------------------------------------------
;-- Function:      unrot_lat(rotlat,rotlon,pollat,pollon)
;-- Description:   transform rotated latitude to latitude
;--------------------------------------------------------------
undef("unrot_lat")
function unrot_lat( rotlat:numeric, rotlon:numeric, pollat[1]:numeric, 
    pollon[1]:numeric )
    local rotlat,rotlon,nrlat,nrlon,nrlat_rank,nrlon_rank,pollon,pollat, 
        lat, s1, c1, rlo, rla, i
begin
    lat = fillval
lat@_FillValue = fillval

nrlat = dimsizes(rotlat)
nrlon = dimsizes(rotlon)
nrlat_rank = dimsizes(nrlat)
nrlon_rank = dimsizes(nrlon)

if (any(nrlat.ne.nrlon).and.(nrlat_rank.ne.1 .or. nrlon_rank.ne.1)) then
  print("Function unrot_lat: rotlat and rotlon dimensions do not match")
  return(lat)
end if

if (nrlat_rank.eq.1 .and. nrlon_rank.eq.1) then
  rla = conform_dims((/nrlat,nrlon/),rotlat,0)    ; -- create 2D lat array
  rlo = conform_dims((/nrlat,nrlon/),rotlon,1)    ; -- create 2D lon array
else
  rla = rotlat
  rlo = rotlon
end if

rla = rla*deg2rad                      ; -- convert from degree to radians
rlo = rlo*deg2rad                      ; -- convert from degree to radians

lat := (/rla/)                         ; -- reassign lat
lat@_FillValue=fillval
s1  = sin(pollat*deg2rad)
c1  = cos(pollat*deg2rad)
lat = s1*sin(rla)+c1*cos(rla)*cos(rlo)
lat = asin(lat)*rad2deg
lat@units = "degrees_north"
print("Function unrot_lat: min/max     "+sprintf("%8.4f",\n  min(lat(:,0)))+\" "+sprintf("%8.4f", max(lat(:,0))))
delete([/rlo,rla,c1,s1/])
return(lat)
end

;----------------
;-- MAIN
;----------------
begin
;-- open file and read variables
  diri = "../data/"
  fili = "tas_rotated_grid_EUR11.nc"
  f = addfile(diri+fili,"r")
  var = f->tas
  rlat = f->rlat
  rlon = f->rlon
  rotpole = f->rotated_pole
  pollat = rotpole@grid_north_pole_latitude
  pollon = rotpole@grid_north_pole_longitude

;-- unrotate the grid and set 2D lat/lons
  var@lon2d = unrot_lon(rlat, rlon, pollat, pollon)
  var@lat2d = unrot_lat(rlat, rlon, pollat, pollon)
;-- calculate the min and max lat/lons for the map plot
minlat = min(var@lat2d) ;-- retrieve minimum latitude value
minlon = min(var@lon2d) ;-- retrieve minimum longitude value
maxlat = max(var@lat2d) ;-- retrieve minimum longitude value
maxlon = max(var@lon2d) ;-- retrieve maximum longitude value

;-- open a workstation
wks_type           = "png"
wks_type@wkWidth   = 1024
wks_type@wkHeight  = 1024
wks = gsn_open_wks(wks_type,"plot_rotated_grid")

;-- set resources
res                       = True
res@gsnFrame              = False ;-- don't advance frame
res@gsnAddCyclic          = False ;-- don't add lon cyclic point
res@pmTickMarkDisplayMode = "Always" ;-- draw nicer tickmarks
res@mpDataBaseVersion     = "MediumRes" ;-- choose map database
res@mpMinLatF             = minlat - 1. ;-- set min lat
res@mpMaxLatF             = maxlat + 1. ;-- set max lat
res@mpMinLonF             = minlon - 1. ;-- set min lon
res@mpMaxLonF             = maxlon + 1. ;-- set max lon
res@mpGridAndLimbOn       = True ;-- turn on grid lines
res@cnFillOn              = True ;-- turn on contour fill
res@cnLinesOn             = False ;-- don't draw contour lines
res@cnFillPalette         = "BlueYellowRed" ;-- choose color map
res@lbLabelBarOn          = True ;-- turn on labelbar
res@tiMainString          = "NCL Doc: rotated grid" ;-- title
res@tiMainOffsetYF        = -0.025 ;-- move title downward
res@vpWidthF              = 0.6 ;-- width of viewport
res@vpHeightF             = 0.48 ;-- height of viewport

;-- create the first plot
res@vpXF                  = 0.12 ;-- start x-position
res@vpYF                  = 1.02 ;-- start y-position
plot1 = gsn_csm_contour_map(wks,var(0,0,:,:),res) ;-- use default projection (CE)

;-- create the second plot
delete(res@tiMainString)   ;-- we don't need the title twice
res@vpXF                  = 0.15 ;-- start x-position
res@vpYF                  = 0.493 ;-- start y-position
res@mpProjection          = "Orthographic" ;-- change projection
res@mpCenterLatF          = minlat+(maxlat-minlat)/2 ;-- center point of view latitude
res@mpCenterLonF          = minlon+(maxlon-minlon)/2 ;-- center point of view longitude
res@mpLimitMode           = "LatLon" ;-- map limits mode
res@mpMinLatF             = minlat - 1. ;-- set min lat
res@mpMaxLatF             = maxlat + 1. ;-- set max lat
res@mpMinLonF             = minlon - 1. ;-- set min lon
res@mpMaxLonF             = maxlon + 1. ;-- set max lon
res@mpPerimOn             = False ;-- don't draw the box around the plot
res@lbOrientation = "vertical" ;-- vertical label bar
res@lbLabelStride = 2
res@lbLabelPosition = "Left" ;-- labelbar labels on left side
res@pmLabelBarOrthogonalPosF = -1.37 ;-- labelbar on the left side
res@tmXTLabelDeltaF = -0.5 ;-- decrease space between ticks
res@tmXBLabelDeltaF = -0.5 ;-- decrease space between ticks
res@tmYLLabelDeltaF = -0.5 ;-- decrease space between ticks
res@tmYRLabelDeltaF = -0.5 ;-- decrease space between ticks

plot2 = gsn_csm_contour_map(wks,var(0,0,:,:),res) ;-- draw second plot

;-- draw text
 txres = True
 txres@txFontHeightF = 0.016
 txres@txJust = "CenterLeft"
 gsn_text_ndc(wks,"Projection:", 0.74, 0.91, txres) ;-- next to first plot
 gsn_text_ndc(wks,"Cylindrical Equidistant", 0.74, 0.89, txres)
 gsn_text_ndc(wks,"Projection:", 0.77, 0.43, txres) ;-- next to second plot
 gsn_text_ndc(wks,"Orthographic", 0.77, 0.41, txres)

;-- advance the frame
 frame(wks)
 end
8.17 Globe with different grid resolutions

A very special kind of plot is the following example, which shows two regional model domains with different grid resolutions mapped onto one globe. The background color of the plotting frame is changed to dark grey and the foreground color to white. One single color map is used to visualize the orography in both model domains, while two different shades of blue are used for the water areas of the two data sets in order to make the grids more distinguishable. This is a good demonstration of the capabilities of NCL graphics.

Globe with different grid resolutions example: NUG_globe_orography_grid_resolution.ncl

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

begin
  dir1 = "/"
  fil1 = "HSURF_regional_model_0.11deg.nc"
  fil2 = "FR-LAND_regional_model_0.11deg.nc"
  fil3 = "HSURF_regional_model_0.44deg.nc"
  fil4 = "FR-LAND_regional_model_0.44deg.nc"

  f1 = addfile(dir1+fil1,"r")
  var1 = f1->HSURF(0,:,:)
  mask1 = addfile(dir1+fil2,"r")
  lsm1 = mask1->FR_LAND(0,:,:)
  lsm1 = where(lsm1.gt.0.5,-9999,lsm1)
  land_only1 = var1
  land_only1 = mask(var1,lsm1,-9999)

  f2 = addfile(dir1+fil3,"r")
  var2 = f2->HSURF(0,:,:)
  mask2 = addfile(dir1+fil4,"r")
  lsm2 = mask2->FR_LAND(0,:,:)
  lsm2 = where(lsm2.gt.0.5,-9999,lsm2)
  land_only2 = var2
  land_only2 = mask(var2,lsm2,-9999)

  lat2d = f1->lat
  lon2d = f1->lon
  nlat = dimsizes(lat2d(:,0))
  nlon = dimsizes(lon2d(0,:))

  -- open workstation
  wks_type = "png"
  wks_type@wkBackgroundColor = "grey18"
  wks_type@wkForegroundColor = "white"
  wks = gsn_open_wks(wks_type, "plot_globe_orography_grid_resolution")

  -- global resources
  res = True
  res@gsnDraw = False
  res@gsnFrame = False

  -- map resources
  mpres = res
  mpres@mpProjection = "Orthographic"
  mpres@mpLabelsOn = False
  mpres@mpPerimOn = True
  mpres@mpGridLineColor = "grey40"
  mpres@mpGridAndLimbOn = True
  mpres@mpFillOn = True
```

mpres@mpOutlineOn = True
mpres@mpOutlineDrawOrder = "PostDraw"
mpres@mpFillDrawOrder = "PreDraw"
mpres@mpOceanFillColor = (/ 0.824, 0.961, 1.0 /)
mpres@mpLandFillColor = (/ 0.7, 0.7, 0.7 /)
mpres@mpCenterLatF = 15.
mpres@mpCenterLonF = 15.
map = gsn_csm_map(wks,mpres)

--;-- AFR-44
cnres2 = res
cnres2@cnFillOn = True
(cnres2@cnMissingValFillColor = "steelblue3"
cnres2@cnLinesOn = False
(cnres2@cnLineLabelsOn = False
(cnres2@cnLevelSelectionMode = "ManualLevels"
cnres2@cnMinLevelValF = 0.0
(cnres2@cnMaxLevelValF = 3000.
cnres2@cnLevelSpacingF = 50.
cnres2@cnFillPalette = "OceanLakeLandSnow"
cnres2@cnFillDrawOrder = "PostDraw"
cnres2@gsnRightString = ""
cnres2@gsnLeftString = ""
cnres2@lbOrientation = "vertical"
cnres2@lbLabelFontHeightF = 0.013
cnres2@tiXAxisString = ""
cnres2@tiYAxisString = ""

plot2 = gsn_csm_contour(wks,land_only2,cnres2)

--;-- overlay Africa
overlay(map,plot2)

--;-- polyline resources
resl = True
(resl@gsLineThicknessF = 2.0
resl@gsLineColor = "black"

--;-- plot the box around the data field
xbox = (/ -29.04, 64.68, 64.68, -29.04, -29.04 /)
ybox = (/ -50.16, -50.16, 46.64, 46.64, -50.16 /)
dum1 = gsn_add_polyline(wks, plot2, xbox, ybox, resl)

--;-- delete unnecessary things
delete([/resl,cnres2,var2,mask2,lsm2,land_only2/])

--;-- EUR-11
cnres = res
cnres@cnFillOn = True ; turn on color
cnres@cnMissingValFillColor = "steelblue1"
cnres@cnLinesOn = False ; no contour lines
cnres@cnLineLabelsOn = False ; no contour labels
cnres@cnLevelSelectionMode = "ManualLevels" ; set manually levels
cnres@cnMinLevelValF = 0.0 ; minimum contour level
cnres@cnMaxLevelValF = 3000. ; maximum contour level
cnres@cnLevelSpacingF = 50. ; contour level spacing
cnres@cnFillPalette = "OceanLakeLandSnow"
cnres@cnFillDrawOrder = "PostDraw"
cnres@gsnRightString = "["+var1@units+""
cnres@gsnRightStringFontHeightF = 0.013
cnres@gsnRightStringParallelPosF = 1.19
cnres@gsnRightStringOrthogonalPosF = -0.007

-- overlay Europe
plot1 = gsn_csm_contour(wks,land_only1,cnres)

-- polyline resources
resl = True
resl@gsLineThicknessF = 3.0
resl@gsLineColor = "black"

-- define edges
lon_val_upper = lon2d(nlat-1,:)
lat_val_upper = lat2d(nlat-1,:)
lon_val_lower = lon2d(0,:)
lat_val_lower = lat2d(0,:)
lon_val_left = lon2d(:,0)
lat_val_left = lat2d(:,0)
lon_val_right = lon2d(:,nlon-1)
lat_val_right = lat2d(:,nlon-1)

-- draw edges
upper = gsn_add_polyline(wks, plot1, lon_val_upper, lat_val_upper, resl)
lower = gsn_add_polyline(wks, plot1, lon_val_lower, lat_val_lower, resl)
left = gsn_add_polyline(wks, plot1, lon_val_left, lat_val_left, resl)
right = gsn_add_polyline(wks, plot1, lon_val_right, lat_val_right, resl)
overlay(map,plot1)

-- draw title strings
str = "Domains Europe 0.11~S~o~N~ and Africa 0.44~S~o~N~"
txres = True
txres@txFontHeightF = 0.013
txres@txJust = "BottomLeft"
gsn_text_ndc(wks, str, 0.20, 0.82, txres)

txres@txJust = "BottomRight"
txres@txFontHeightF = 0.011
str = "(c) DKRZ/CCLM"
gsn_text_ndc(wks,str, 0.92, 0.82, txres)

-- draw the frame
draw(map)
frame(wks)
end
8.18 Helpful Resources

Many NCL resources were used to modify the NCL graphics to get the desired result for the plot layout.

For example:

```
res = True ; create a resource object list
res@cnFillOn = True
res@tiMainFontHeightF = 0.02
res@mpProjection = "Mollweide"
```

The first two lowercase letters are the abbreviation of the resource type.

Resource types:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>am</td>
<td>Annotation Manager (AnnoManager)</td>
</tr>
<tr>
<td>app</td>
<td>App (App)</td>
</tr>
<tr>
<td>ca</td>
<td>Coordinate Array (CoordArrays)</td>
</tr>
<tr>
<td>cn</td>
<td>Contour (ContourPlot)</td>
</tr>
<tr>
<td>ct</td>
<td>Coordinate Array Table (CoordArrTable)</td>
</tr>
<tr>
<td>dc</td>
<td>Data Comm (DataComm)</td>
</tr>
<tr>
<td>err</td>
<td>Error</td>
</tr>
<tr>
<td>gs</td>
<td>Graphic Style (GraphicStyle)</td>
</tr>
<tr>
<td>gsn</td>
<td>GSN High-level Interfaces (GSN)</td>
</tr>
<tr>
<td>lb</td>
<td>Label Bar (LabelBar)</td>
</tr>
<tr>
<td>lg</td>
<td>Legends (Legend)</td>
</tr>
<tr>
<td>mp</td>
<td>Maps (MapPlot and MapTransformation)</td>
</tr>
<tr>
<td>pm</td>
<td>Plot Manager (PlotManager)</td>
</tr>
<tr>
<td>pr</td>
<td>Primitives (Primitive)</td>
</tr>
<tr>
<td>sf</td>
<td>Scalar Field (ScalarField and MeshScalarField)</td>
</tr>
<tr>
<td>st</td>
<td>Streamline (StreamlinePlot)</td>
</tr>
<tr>
<td>tf</td>
<td>Transform</td>
</tr>
<tr>
<td>ti</td>
<td>Title</td>
</tr>
<tr>
<td>tm</td>
<td>Tickmark (TickMark)</td>
</tr>
<tr>
<td>tr</td>
<td>Transformation (Transformation, IrregularTransformation, LogLinTransformation)</td>
</tr>
<tr>
<td>tx</td>
<td>Text (TextItem)</td>
</tr>
<tr>
<td>vc</td>
<td>Vectors (VectorPlot)</td>
</tr>
<tr>
<td>vf</td>
<td>Vector Field (VectorField)</td>
</tr>
<tr>
<td>vp</td>
<td>View Port (View)</td>
</tr>
<tr>
<td>wk</td>
<td>Workstation (Workstation, DokumentWorkstation, ImageWorkstation, NcgmWorkstation, PDFWorkstation, PSWorkstation, XWorkstation)</td>
</tr>
<tr>
<td>ws</td>
<td>Workspace</td>
</tr>
<tr>
<td>xy</td>
<td>XY-Plots (XyPlot)</td>
</tr>
</tbody>
</table>

### 8.18.1 Title Strings and Function Codes

The “~” character in NCL has a special meaning for all text-based resources. It’s a “function code” that tells NCL you want to do something special to the string, like create a super/subscript, insert a carriage return, or change the font:

Text/title examples: [http://www.ncl.ucar.edu/Applications/text.shtml](http://www.ncl.ucar.edu/Applications/text.shtml)

Function code examples: [http://www.ncl.ucar.edu/Applications/fcodes.shtml](http://www.ncl.ucar.edu/Applications/fcodes.shtml)

Insert a carriage return: ~C~

Change font size:
- ~Z80~ decrease font size to 80%
- ~Z50~ decrease font size to 50%

Superscript: ~S~ o ~N~ degrees sign °

Subscript: ~B~ 2 ~N~ subscript 2 like ₂
If you want a different character to be the function code, then you can either set a resource, i.e. restiMainFuncCode = ":", or you can change the default for all resources by setting the special TextFuncCode value in your ".hluresfile". See section 1.6.

More than one title string on top of the plot: NUG_title_strings.ncl

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

begin
  diri = "./"
  fili = "rectilinear_grid_2D.nc"

  file1 = addfile(diri+fili, "r")
  var = file1->tsurf(0,:,:)

  ;---- define the workstation (plot output type and name)
  wks = gsn_open_wks("png","plot_title_strings")

  ;---- set resources
  res = True
  res@gsnMaximize = True

  ;-- set the gsn title strings
  res@gsnLeftString = "Left String"
  res@gsnCenterString = "Center String"
  res@gsnRightString = "Right String"
```

8.18.2 Adding Text To The Plot

Sometimes the settings described before do not fit your requirements and it is necessary to add more text to the plot. This can be done by the text function `gsn_add_text` which adds text to an existing plot (only in the plot area) or `gsn_text_ndc` which writes text at any place on the frame using NDC coordinates to position the text.
The right figure shows the location of the justification point for a string which is defined by resources that end in the word "Just".

```
value = 3.83927489235
str = "formatted value \"+sprintf("%3.4f", value)\" in text"
  txres@txFontColor      = "violet"
  txres@txFontHeightF    = 1.0
  id = gsn_add_text(wks, plot, str, 1, -3.5, txres)
```

```
value = 3.83927489235
str = "formatted value \"+sprintf("%3.4f", value)\" in text"
  txres@txFontColor      = "violet"
  txres@txFontHeightF    = 1.0
  id = gsn_add_text(wks, plot, str, 1, -3.5, txres)
```
Just to show you more about NCL's capabilities to add text to a plot.

```plaintext
ndcres@txFontHeightF = 0.03
ndcres@txJust = "BottomLeft"
gsn_text_ndc(wks, "Text line 1", 0.15, 0.95, ndcres)
ndcres@txFontHeightF = 0.015
ndcres@txFontColor = "blue"
gsn_text_ndc(wks, "x-axis title string", 0.09, 0.5, ndcres)
ndcres@txFontHeightF = 0.01
ndcres@txJust = "CenterCenter"
gsn_text_ndc(wks, "Text line 2", 0.15, 0.90, ndcres)
ndcres@txFontHeightF = 0.018
ndcres@txFontColor = "red"
gsn_text_ndc(wks, "y-axis title string", 0.09, 0.5, ndcres)
```

```plaintext
Auml = "A~H-15V6F35~H-FV-6H3~"
auml = "a~H-13V2F35~H-FV-2H3~"
Ouml = "O~H-16V6F35~H-FV-6H3~"
ouml = "o~H-14V2F35~H-FV-2H3~"
Uuml = "U~H-15V6F35~H-FV-6H3~"
uuml = "u~H-13V2F35~H-FV-2H3~"
string = "Umlaute: "+Auml+ "+auml+ "+Ouml+ "+ouml+ "+Uuml+ "+uuml
ndcres@txFontColor = "black"
```

```plaintext
ndcres@txFontHeightF = 0.025
gsn_text_ndc(wks, string, 0.05, 0.06, ndcres)
```
Function Codes for Creating Special Characters

Different text settings such as Umlaut, superscript and subscript: NUG_text_settings.ncl

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
begin
;
define german "Umlaute"
Auml = "A~H-15V6F35-H-FV-6H3-"
auml = "a~H-13V2F35-H-FV-2H3-"
Ouml = "O~H-16V6F35-H-FV-6H3-"
ouml = "o~H-14V2F35-H-FV-2H3-"
Uuml = "U~H-15V6F35-H-FV-6H3-"
uuml = "u~H-13V2F35-H-FV-2H3-"

define super- and subscript variable
super = "m~S-3-N~ [m s~S~-1-N~] [kg m~S~-2-N~] 30~S-o-N-C"
sub = "Schwefels+auml+"ure:   H~B~2~N~SO~B~4~N~"
data = (/ "Data-1", "Data-2", "Data-3", "Data-4", "Data-5")
diff = (/ 16.25, -0.93, 0.43, 3.5, 0.0/)
var = (/ 0.06, 0.02, 0.04, 0.05, 0.03/)
ratio = (/ 2, 2.4, 1.1, 0.9, 0.0/)

ntext = dimsizes(data)

open workstation
wks = gsn_open_wks("png","plot_text_settings")

x = 0.1
y = 0.95
inc = 0.07

txres=false

open workstation
wks = gsn_open_wks("png","plot_text_settings")

x = 0.1
y = 0.95
inc = 0.07

-- text resources
str = "NCL Doc Example: Text settings"
gsn_text_ndc(wks,str,0.5,y,txres)
txres@txFontHeightF = 0.03
txres@txJust = "CenterCenter"
str = "NCL Doc Example: Text settings"
gsn_text_ndc(wks,str,0.5,y,txres)
txres@txJust = "CenterLeft"
str1 = "Umlaute:"
gsn_text_ndc(wks,str1,x,y-2*inc,txres)
str2 = Auml+" "+auml+" "+Ouml+" "+ouml+" "+Uuml+" "+uuml
gsn_text_ndc(wks,str2,x+0.3,y-2*inc,txres)
str1 = "Superscript:"
gsn_text_ndc(wks,str1,x,y-3*inc,txres)
str2 = super
gsn_text_ndc(wks,str2,x+0.3,y-3*inc,txres)
str1 = "Subscript:"
gsn_text_ndc(wks,str1,x,y-4*inc,txres)
str2 = sub
gsn_text_ndc(wks,str2,x+0.3,y-4*inc,txres)

-- nice formated text output using sprintf
str = "Format:"
gsn_text_ndc(wks,str,x,y-5*inc,txres)
xpos = 0.4
do i=0,ntext-1
  ypos = y-5*inc+i*0.05
gsn_text_ndc(wks,data(i),xpos,ypos,txres)
end do
txres@txJust = "CenterRight"
do i=0,ntext-1
    xpos = 0.65
    ypos = y-5*inc+i*0.05
    if(diff(i).ne.0.0) then
        str = sprintf("%6.2f",diff(i))
        gsn_text_ndc(wks,str,xpos,ypos,txres)
    else
        str = "-"
        gsn_text_ndc(wks,str,xpos,ypos,txres)
    end if
    xpos = xpos + 0.12
    if(var(i).ne.0.0) then
        str = sprintf("%5.2f",var(i))
        gsn_text_ndc(wks,str,xpos,ypos,txres)
    end if
    xpos = xpos + 0.12
    if(ratio(i).ne.0.0) then
        str = sprintf("%3.1f",ratio(i))
        gsn_text_ndc(wks,str,xpos,ypos,txres)
    else
        str = "-"
        gsn_text_ndc(wks,str,xpos,ypos,txres)
    end if
end do

;-- greek characters
    xpos = 0.3
    ypos = 0.3
    str1 = "Greek font:"
    gsn_text_ndc(wks,str1,xpos,ypos,txres)
    str2 = "alpha = \(\alpha\)"
    gsn_text_ndc(wks,str2,xpos+0.27,ypos,txres)
    str2 = "beta   = \(\beta\)"
    gsn_text_ndc(wks,str2,xpos+0.27,ypos-0.05,txres)
    str2 = "sigma = \(\sigma\)"
    gsn_text_ndc(wks,str2,xpos+0.28,ypos-0.10,txres)

;-- decrease the font
    str1 = "Font size 100%"
    gsn_text_ndc(wks,str1,xpos+0.08,ypos-3*inc,txres)
    str2 = "\(\text{Font size 70\%}\)"
    gsn_text_ndc(wks,str2,xpos+0.3,ypos-3*inc,txres)
    str3 = "\(\text{Font size 40\%}\)"
    gsn_text_ndc(wks,str3,xpos+0.45,ypos-3*inc,txres)
frame(wks)
end
8.18.4 Axis Annotations

Adding units and axis labels to the plot: NUG_axis_annotations.ncl

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

begin

;---- read the data and define variable reference var

diri  = "/"
fili  = "rectilinear_grid_3D.nc"
f     = addfile(diri+fil, "r")
var   = f->t(0,70000,55,0:60)
lon_t = f->lon(0:60) ;-- longitude=0-60E

;---- define the workstation (plot output type and name)
wks = gsn_open_wks("png","plot_axis_annotations")

;---- set resources
res = True
res@gsnMaximize = True

;-- set the title string. ~C~ insert a carriage return (no \ allowed).
res@tiMainString = "NCL Doc Example: Axis Annotations"
res@tiMainFontHeightF = 0.02
res@tiXAxisSide = "Bottom" ;-- X-Axis title on bottom
res@tiXAxisFontHeightF = 0.015 ;-- X-Axis title font size
res@tiYAxisFontHeightF = 0.015 ;-- Y-Axis title font size
res@tiXAxisString = lon_t@long_name
res@tiYAxisString = var@long_name + " [" + var@units + "]"
res@tmLabelAutoStride = True
res@tmXBTickSpacingF = 5 ;-- label X-Axis every 10 deg
res@xyLineThicknessF = 2.0

plot = gsn_csm_xy(wks,lon_t,var,res)

end
```
The next example shows how to change the strings for the contour lines and their labels:

NUG_contour_labels.ncl

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

begin
  diri  = "./"
  fili  = "rectilinear_grid_2D.nc"
  file1 = addfile(diri+fili,"r")
  var   = file1->tsurf(0,:,:)

;---- define the workstation (plot output type and name)
  wks = gsn_open_wks("png","plot_contour_lines_labels")

;---- set resources
  res                       = True
  res@gsnDraw               = False
  res@gsnFrame              = False
  res@gsnMaximize           = True
  res@tiMainFontHeightF     = 0.02
  res@cnLevelSelectionMode  = "ManualLevels"
  res@cnMinLevelValF        = 250.
  res@cnMaxLevelValF        = 310.
  res@cnLevelSpacingF       = 2.
  res@mpLimitMode           = "Corners"
  res@mpLeftCornerLatF      = 35.
  res@mpRightCornerLatF     = 50.
  res@mpLeftCornerLonF      = -12.
  res@mpRightCornerLonF     = 10.

;-- create plots
  plot = new(3,graphic)
  plot(0) = gsn_csm_contour_map(wks, var, res)
  plot(1) = gsn_csm_contour_map(wks, var, res)
  plot(2) = gsn_csm_contour_map(wks, var, res)

;-- set to default: randomized
  res@cnLineColorBackground = "white"
  res@cnSmoothingOn        = True
  res@cnSmoothingTensionF  = 1.
  res@cnSmoothingDistanceF = 0.005
  res@cnLevelSpacingF      = 1.
  plot(2) = gsn_csm_contour_map(wks, var, res)

;-- draw the panel plot
  pres = True
  pres@gsnPanelFigureStrings = (/"a")","b")","c")/)
```

8.18.5 Contour Lines and Label Settings
8.18.6 Colorizing Land, Ocean and Lakes

Colorize land and ocean areas differently: NUG_color_Land_Ocean.ncl

begin
diri  = "/
fili  = "CNTASN_1m_200103_grid_T_curvilinear_grid.nc"
f     = addfile(diri+fili,"");
var   = f->votemper(0,0,:,:)
lon2d = f->nav_lon
lat2d = f->nav_lat

;-- define the workstation (plot type and name)
wks = gsn_open_wks("png","plot_color_Land_Ocean")

;-- set resources
res   = True
res@gsnAddCyclic = False  ;-- don't add lon cyclic point
res@gsnMaximize = True
res@cnFillOn = True ;-- turn on contour fill
res@cnFillPalette = "rainbow" ;-- choose color map
res@tiMainString = "NCL Doc Example: color land and ocean" ;-- title
res@tiMainFontHeightF = 0.02

;-- color land and ocean (looks nicer)
res@mpOceanFillColor = "lightblue"
res@mpInlandWaterFillColor = "lightblue"
res@mpLandFillColor = "tan"

;--- Zoom in on map
res@mpMinLatF = min(lat2d)
res@mpMaxLatF = max(lat2d)
res@mpMinLonF = min(lon2d)
res@mpMaxLonF = max(lon2d)

;--- Lat/lon arrays of curvilinear grid for overlaying on map
res@sfXArray = lon2d
res@sfYArray = lat2d

;-- draw the contour map
plot = gsn_csm_contour_map(wks,var,res)
end
Colorize Countries by Values

NCL allows the user to colorize the countries by given values. Let us say you have an ASCII file containing the country name and the number of users for something, separated by a delimiter. You can store the names of the countries into an array (e.g. "states") after reading and generate an array containing the desired colors (e.g. "icol") depending on the values of the countries. With the \@mp resources

\begin{verbatim}
res@mpFillAreaSpecifiers = states ;-- fill listed states
res@mpSpecifiedFillColors = icols ;-- use generated color array
\end{verbatim}

NCL will plot the country states(i) colored by the color icol(i).

NUG_color_country_user.ncl:

\begin{verbatim}
begin

diri = "./data/
fill = "data_country_user.txt"
data = asciiread(diri+fill, -1,"string ");-- read all lines
delim = ";:" ;-- set delimiter
nfields = str_fields_count(data(0),delim) ;-- count number of columns
states = str_get_field(data,1,delim) ;-- get 1st column
ivalues = toint(str_get_field(data,2,delim)) ;-- get 2nd column
nvalues = dimsizes(ivalues)
levels = (/1,2,5,10,50,100,200,500,1000,2000/) ;-- value levels
labels = ("1", "2", "5", ">10", ">50", ">100", ">500", ">1000", ">2000") ;-- labelbar labels
rgb_colors = (/ (/0.997785, 0.999139, 0.846059/), 
(0.910127, 0.964937, 0.695640/), 
(0.769320, 0.909419, 0.706959/), 
(0.521292, 0.812964, 0.731073/), 
(0.304483, 0.732118, 0.761430/), 
(0.141961, 0.597647, 0.756078/), 
(0.122107, 0.483137, 0.717717/), 
(0.131949, 0.382745, 0.665467/), 
(0.138408, 0.297578, 0.624990/), 
(0.031373, 0.113725, 0.345098/) )
nlevels = dimsizes(levels) ;-- number of levels
colors = new((/nvalues+1,3/),typeof(rgb_colors))

;-- compute the data color array
do i = 0,dimsizes(ivalues)-1
  if(ivalues(i).eq.levels(0)) then
    colors(i,:) = (/1.,1.,1./) ;-- white
  end if
  if(ivalues(i).gt.levels(nlevels-1)) then
    colors(i,:) = rgb_colors(nlevels-1,:)
  end if
  do j = 0,nlevels-2
    if(ivalues(i).gt.levels(j).and.ivalues(i).le.levels(j+1)) then
      colors(i,:) = rgb_colors(j,:)
    end if
  end do
  print("State: "+sprinti("%2.2i",i)+" Count: "+ 
        sprintf("%4.2i",ivalues(i))+" "+states(i))
end do
\end{verbatim}
;-- open a workstation
;-- set workstation resources
  wks_type                 = "png"         ;-- plot output type
  wks_type@wkWidth         = 1600         ;-- wk width
  wks_type@wkHeight        = 1600         ;-- wk height
  wks = gsn_open_wks(wks_type,"plot_color_country_user")

;-- set resources
  res = True
  res@gsnMaximize           = True         ;-- maximize plot
  res@gsnFrame              = False        ;-- don't advance the frame yet
  res@pmTickMarkDisplayMode = "Always"     ;-- turn on map tickmarks
  res@mpDataSetName         = "Earth..4"    ;-- new database
  res@mpDataBaseVersion     = "MediumRes"   ;-- Medium resolution database
  res@mpOutlineOn           = True         ;-- turn on map outlines
  res@mpFillOn              = True         ;-- turn on map fill
  res@mpOutlineBoundarySets = "National"   ;-- draw only national bounds
  res@mpLandFillColor       = "white"      ;-- set map land fill to white
  res@mpMinLatF             = -60          ;-- don't plot Antarctica

;-- set colors and states
  res@mpFillAreaSpecifiers  = states       ;-- fill listed states
  res@mpSpecifiedFillColors = colors       ;-- use generated color array
  res@tiMainString          = "User Count" ;-- title string
  res@tiMainFont            = "helvetica"  ;-- title string font
  res@tiMainFontHeightF     = 0.025        ;-- set title string font size

  map = gsn_csm_map(wks,res) ;-- create the map

;-- add custom label bar to the plot
  lbres = True
  lbres@lbPerimOn           = False        ;-- no label bar box outline
  lbres@lbOrientation      = "Horizontal" ;-- labelbar orientation
  lbres@vpXF                = 0.01         ;-- labelbar x-position
  lbres@vpYF                = 0.26         ;-- labelbar y-position
  lbres@vpWidthF            = 0.98         ;-- labelbar width
  lbres@vpHeightF           = 0.08         ;-- labelbar height
  lbres@lbLabelFontHeightF  = 0.012        ;-- label font height
  lbres@lbMonoFillPattern   = True         ;-- fill sold
  lbres@lbAutoManage        = False        ;-- make settings by yourself
  lbres@lbLabelAlignment    = "BoxCenters" ;-- where to draw the labelbar
                             ; labels
  lbres@lbFillColors        = rgb_colors   ;-- use colors

  gsn_labelbar_ndc(wks,nlevels,labels,0.13,0.28,lbres) ;-- draw labelbar
  frame(wks)                ;-- advance the frame
end
8.18.8  Labelbar Settings

Labelbars are often needed to show the mapping between physical values and colors. If a contour plot with fill mode turned on is chosen NCL will add a labelbar to the plot. The labelbar is centered below the plot, having nice values below the color boxes. By default no title string is included with the labelbar.

The user can change the labelbar width, height, position, color and font of the labels, exclude outer color boxes, add minimum and maximum labels to the outer boxes, add a title to the labelbar, etc.
The next example shows how to achieve different labelbar styles and annotation formats: `NUG_labelbars.ncl`

```ncl
begin
  diri = "/"
  fili = "T2M_ERAINT_rectilinear_grid_2D.nc"
  f   = addfile(diri+fili, "r")
  var = f->T2M(0,:,:);

  -- define the workstation (graphic will be written to a file)
  wks = gsn_open_wks("png","plot_labelbars")

  -- set plot resources
  res = True
  res@gsnDraw  = False
  res@gsnFrame = False
  res@gsnMaximize = True

  res@cnFillOn       = True
  res@cnLinesOn      = False
  res@cnLineLabelsOn = False
  res@cnInfoLabelOn  = False
  res@cnLevelSelectionMode = "ManualLevels"
  res@cnMinLevelValF = 250.
  res@cnMaxLevelValF = 310.
  res@cnLevelSpacingF = 5.

  plot = new(6,graphic)
  plot@conf@cnLabelBarEndStyle = "ExcludeOuterBoxes";-- exclude the outer color boxes
  plot@conf@lbTitleOn             = True;-- write title (default: "labelbar")
  plot@conf@lbTitleFont           = "courier-bold";-- set title font
  plot@conf@lbTitleFontColor      = "blue";-- set title color
  plot@conf@lbTitleFontHeightF    = 0.015;-- decrease the font size (default: 0.025)
  plot@conf@lbTitlePosition       = "Bottom";-- labelbar title position (default: "Top")
  plot@conf@lbTitleString         = t@units;-- define labelbar title string
  plot@conf@lbTitleOffsetF        = -0.3;-- move the labelbar title upwards
  plot@conf@lbBoxMinorExtentF     = 0.2;-- decrease height of labelbar boxes and vp
  plot@conf@lbBoxLinesOn          = False;-- don't draw lines around labelbar boxes
  plot@conf@lbLabelFontColor      = "blue";-- label color
  plot@conf@lbLabelFontColor      = "blue";-- label color
  plot@conf@lbLabelFontHeightF    = 0.015;-- label font height
  plot@conf@lbLabelFont           = "helvetica-bold";-- label font
  plot@conf@lbLabelOffsetF        = 0.07;-- move the labelbar labels downwards
  plot@conf@pmLabelBarWidthF      = 0.8;-- labelbar width; default is shorter
  plot@conf@pmLabelBarHeightF     = 0.1;-- labelbar height; default is taller
  plot@conf@pmLabelBarOrthogonalPosF = 0.07;-- y-position (positive: downward; def: 0.02)
  plot@conf@pmLabelBarParallelPosF = 0.5;-- x-position (CenterCenter); default: 0.5
end
```
res@tiMainString = "Labelbar: horizontal (default)"
plot(0) = gsn_csm_contour_map(wks,var,res)

res@tiMainString = "Labelbar: vertical"
res@lbOrientation = "vertical"
plot(1) = gsn_csm_contour_map(wks,var,res)

res@tiMainString = "Labelbar: exclude outer boxes"
res@lbOrientation = "horizontal"
res@cnLabelBarEndStyle = "ExcludeOuterBoxes"
plot(2) = gsn_csm_contour_map(wks,var,res)

res@tiMainString = "Labelbar: rotate labels and set labelbar title"
res@lbTitleOn = True
res@lbTitleString = "degK"
res@lbTitlePosition = "Right"
res@lbTitleOffsetF = -0.03
res@lbTitleFontHeightF = 0.015
res@lbLabelFontHeightF = 0.015
res@lbLabelAngleF = 30
res@pmLabelBarOrthogonalPosF = 0.10
delete(res@cnLabelBarEndStyle)
plot(3) = gsn_csm_contour_map(wks,var,res)

res@tiMainString = "Labelbar: reverse colors"
res@gsnSpreadColorStart = -1
res@gsnSpreadColorEnd = 2
delete(res@lbLabelAngleF)
plot(4) = gsn_csm_contour_map(wks,var,res)

res@lbLabelStrings = sprintf("%.2f",levels)
delete([/res@gsnSpreadColorStart, res@gsnSpreadColorEnd/])
res@tiMainString = "Labelbar: format labels"
plot(5) = gsn_csm_contour_map(wks,var,res)

-- draw the panel plot
gsn_panel(wks,plot,(/3,2/),False)
end
8.18.9 Legend Settings

For line plots with different line types or colors, legends are recommended to annotate the respective meanings of the different lines.

To add a legend to a XY-plot, set:

```
res@pmLegendDisplayMode = "Always"
```

This will add a legend below the plot with a size scaled to the plot width. The default legend will likely need to be customized, which you can do with pmLegendXXX and IgXXX resources.

Legends example: NUG_legends.ncl
begin

;----- read the data to be plotted
diri  = "/..\data/"
fili = "rectilinear_grid_3D.nc"
f     = addfile(diri+fili, "r")
temp  = f->t(0,:,{55},{0:60})

;-- set the desired levels to extract
levels = (/100000,85000,70000,50000/)

;-- define the colors of the lines and their labels
colors = (/"red", "green", "blue", "orange"/)
labels = " " + temp&lev@long_name + sprinti("%9i",levels) + \
         " [" + temp&lev@units + "]"

;----- define the workstation (plot output type and name)
wks = gsn_open_wks("png","plot_legends")

;----- set resources
res                        =  True
res@gsnMaximize            =  True
res@trYMinF                =  230
res@trYMaxF                =  max(temp)

;-- set the title string
res@tiMainString           = "NCL Doc Example: Legends"
res@tiMainFontHeightF      =  0.02
res@tiXAxisString          =  temp&lon@long_name
res@tiYAxisString          =  temp@long_name
res@tiXAxisSide            = "Bottom"     ;-- X-Axis title on bottom
res@tiXAxisFontHeightF     =  0.015       ;-- X-Axis title font size
res@tiYAxisFontHeightF     =  0.015       ;-- Y-Axis title font size
res@tmXBTickSpacingF       =  10          ;-- label X-Axis every 10 deg
res@xyLineColors           =  colors
res@xyExplicitLabels       =  labels
res@xyDashPatterns         =  (/0,0,0,0/) ;-- all dash pattern are solid
res@xyLineThicknessF       =  2.0
res@lgJustification        = "TopRight"
res@lgLabelJust            = "CenterRight"
res@lgBoxMinorExtentF      =  0.16       ;-- make the legend lines shorter
res@pmLegendDisplayStyle   = "Always"    ;-- set legend width
res@pmLegendWidthF         =  0.15       ;-- set legend height
res@pmLegendHeightF        =  0.1        ;-- move legend up
res@pmLegendOrthogonalPosF = -0.34       ;-- move legend right

plot = gsn_csm_xy(wks,temp&lon,temp({levels},:),res)
end
To change the legend size, its position, label font, the background color or reverse the order of the legend labels and lines the following resources are needed.

Let's assume the following default legend of a plot:
And here are some variations with their resource settings:

8.18.10  Tickmark Settings

NCL draws major tickmarks at the x- and y-axis by default. In the next part the modifications of tickmarks, their spacing, and their labels are demonstrated for xy- and map plots.
8.18.10.1 XY-plot

See also [http://ncl.ucar.edu/Applications/tickmarks.shtml](http://ncl.ucar.edu/Applications/tickmarks.shtml)

```ncl
;-- set minimum and maximum values of the axis
res@trYMinF = -1.0 ;-- y-axis minimum value
res@trYMaxF = 1.0 ;-- y-axis maximum value
res@trXMinF = min(x) ;-- x-axis minimum value
res@trXMaxF = max(x) ;-- x-axis maximum value

;-- set y-axis major and minor tickmarks and tickmarks values
res@tmYLMode = "Manual" ;-- set tickmarks resources manually
res@tmYLTickSpacingF = 0.2 ;-- label tickmarks at 0.2 intervals
res@tmYLMinorPerMajor = 4 ;-- draw 4 minor tickmarks between
;-- the labeled major tickmarks

;-- set x-axis major and minor tickmarks and tickmarks values
res@tmXBMode = "Manual" ;-- set tickmarks resources manually
res@tmXBTickSpacingF = 10.0 ;-- label tickmarks at 10. intervals
res@tmXBMinorPerMajor = 8 ;-- draw 8 minor tickmarks between
;-- the labeled major tickmarks
```

8.18.10.2 Map-plot

The default tickmark setting for a map can be changed by some @gsn and @tm resources. Not every resource can be used for other map projections than 'Cylindrical Equidistant' but on the NCL examples web page there are some examples for work arounds.

See also [http://ncl.ucar.edu/Applications/mptick.shtml](http://ncl.ucar.edu/Applications/mptick.shtml)
8.18.11 Date Format

The time steps of data in a netCDF file are commonly stored like

```plaintext
double time(time);
time:standard_name = "time";
time:units = "hours since 2001-01-01 00:00:00";
time:calendar = "standard";
```

Here in the example it is saved as "hours since the reference date" in the netCDF file, but we want to annotate something like day.month.year (e.g. 15.01.2000). We don’t want to use the numeric time for annotations directly, so we have to convert it into a more readable format. The function `cd_calendar` will do most of the work for us.
Date Format example: NUG_date_format.ncl
This example requires "cdo", which can be downloaded from
https://code.zmaw.de/projects/cdo/files

```ncl
undef ("getDate")
function getDate(time)
;----------------------
begin
 ;-- convert the time proleptic_gregorian calendar in UTC date
 utc_date = cd_calendar(time, 0)
 ;-- set date variable names (just helpful)
 year   = toint(utc_date(:,0))
 month  = toint(utc_date(:,1))
 day    = toint(utc_date(:,2))
 hour   = toint(utc_date(:,3))
 minute = toint(utc_date(:,4))
 second = utc_date(:,5)
 ;-- write date as string (DD.MM.YYYY)
 date_str_i = sprinti("%0.2i",day) + "." + sprinti("%0.2i",month) + "
" + sprinti("%0.4i",year)
 return(date_str_i)
end
;---------------------------------------------------
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"

begin
diri = "./"
fil1 = "tas_mon_1961-1990_rectilinear_grid_2D.nc"

system("cdo -s -r -f nc yearmean "+diri+fil1+" tser_tmp.nc")

f1  = addfile("tser_tmp.nc","r")
var = f1->tas(:,0,0,0)
time = var&time
timax = dimsizes(time) - 1

f2  = addfile(diri+fil1,"r")
var2 = f2->tas(:,0,:,:)
var2_avg = dim_avg_n_Wrap(month_to_annual(var2,1),2)
tas_avg = var2_avg(:,0)

;-- create the time strings, plot every second axis annotation
incr = 2
date_str_i = getDate(time)
labels = (/ date_str_i(0::incr) /)
;-- define the workstation (plot type and name)
wks  = gsn_open_wks("png", "plot_date_format")

;-- set resources
res = True
res@gsnDraw = False
res@gsnFrame = False
res@xyLineColor = "blue"
res@xyLineThicknessF = 2
res@xyDashPattern = 0
res@vpWidthF = 0.7
res@vpHeightF = 0.37
```

171
res@tiMainString = "NCL Doc Example: date format"
res@tiXAxisString = "Time"
res@tiYAxisString = "Temperature"
res@trYMinF = 279.2
res@trYMaxF = 280.6
res@trXMinF = time(0)
res@trXMaxF = time(timax)
res@tmXBMode = "Explicit"
res@tmXBValues = var&time(:::incr)
res@tmXBLables = labels
res@tmXBLabelFontHeightF = 0.01
res@tmXBLabelJust = "CenterRight"
res@tmXBLabelDeltaF = 1.0
res@tmXBLabelAngleF = 50.
res@tmYROn = False
res@tmXTOn = False

;-- draw the contour map
plot = gsn_csm_xy(wks, var&time, var, res)
res@xyLineColor = "red"
plot2 = gsn_csm_xy(wks, var&time, tas_avg, res)

;-- merge contents from plot2 to plot
overlay(plot,plot2)

;-- draw the merged plot
draw(plot)
frame(wks)

;-- delete temporary file
system("rm -f tser_tmp.nc")
end

8.18.12     Insert a Logo

It is not possible to overlay a file containing a picture of your logo onto the NCL plot, but we can use ImageMagick's 'composite' program to overlay a PNG or JPEG file. The NCL output file type (workstation type) can be PNG, PS or PDF. The call of 'composite' can be done
interactive as a command line call or in a NCL script file. The system call in the NCL script must be done after finishing the plot using the NCL procedure 'system'.

NUG_insert_logo.ncl:

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

;-- generate dummy data
x = ispan(0,100,1)
y = cos(0.0628*ispan(0,100,1)) ;-- generate a curve with 101 points.

;-- output file
dummy = "dum_tmp.png"
output = "plot_with_logo.png"

;-- open a workstation
wks = gsn_open_wks("png", dummy)

;-- set plot resources
res = True
res@gsnDraw = False ;-- don't draw plot yet
res@gsnFrame = False ;-- don't advance frame
res@gsnYRefLine = 0.0 ;-- create a reference line
res@gsnYRefLineThicknessF = 4.0 ;-- create a reference line
res@xyLineThicknessF = 4.0 ;-- line thickness
res@tiMainString = "Cosinus function-C-y = cos(0.0628 * ispan(0,100,1))" ;-- main title string
res@tiXAxisString = "x-axis" ;-- set x-axis title string
res@tiYAxisString = "y-axis" ;-- set y-axis title string
res@tiMainOffsetYF = 0.05 ;-- move title string
res@tiMainFont = "times-roman" ;-- title string font size
res@tmXBLabelFontHeightF = 0.025 ;--larger x-label font size
res@tmYLLabelFontHeightF = 0.025 ;--larger y-label font size

;-- create the plot
plot = gsn_csm_xy(wks, x, y, res) ;-- create the default plot

;-- additional text on plot using plot coordinate

;-- draw red lines
```

```ncl
plres = True
plres@gsLineColor = "red"
plres@gsLineThicknessF = 4.0
```
plid1 = gsn_add_polyline(wks, plot, (/ 5,40/), (/0.99,0.99/), plres)
plid2 = gsn_add_polyline(wks, plot, (/60,95/), (/0.99,0.99/), plres)
plid3 = gsn_add_polyline(wks, plot, (/20,45/), (/0.99,-0.99/), plres)

;-- additional text on plot using page coordinate (NDC)
;---------------------------------------
ndcres = True ;-- text resources copyright string
ndcres@txFontColor = "green" ;-- change to white
ndcres@txFontHeightF = 0.02 ;-- make font size smaller
ndcres@txJust = "CenterCenter" ;-- text justification

gsn_text_ndc(wks,"x-axis", 0.5, 0.83, ndcres) ;-- draw right axis label
ndcres@txAngleF = -90. ;-- rotate the text

gsn_text_ndc(wks,"y-axis", 0.84, 0.5, ndcres) ;-- draw right axis label

;-- draw the plot
;---------------------------------------
draw(plot)
frame(wks)
delete(wks)

;-- add a logo to the finished plot (upper left corner)
;-- (this could be done only for PNG plot output)
;---------------------------------------
logo = "./NCLLogoWeb.jpg"
cmd = "composite -geometry 80x80+20+20 "+logo+" "+dummy+" "+output
system(cmd)
system("rm -rf "+dummy)
end
9 Regridding

For data analysis and visualization in climate modeling, data sets from different sources have to regularly be compared or jointly analysed on the same grid. NCL supports displaying visualizations of different data sets on different grids within one single plot or by using a panel plot. But beyond that, it is often necessary to apply mathematical operations on the data before visualizing it. As an example, a model’s bias is typically visualized as the difference between a model data set and reference data such as observational data. The difference between two fields is built gridpoint by gridpoint, and therefore, as a prerequisite for this type of processing step, all data concerned need to be defined on one single common grid.

To regrid a model data set from one grid to another, the CDOs or the ESMF regridding functions can be used. The CDOs are command line based and can be called from within NCL with the systemfunc function in order to generate new, regiridde data files (see also 2.4). The ESMF functions can be directly called within your NCL script to generate a new data file or keep the data in the memory. The user has to load the contributed.ncl and ESMF_regridding.ncl library files first:

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

There are different methods for regridding the data:

- bilinear (CDOs, ESMF)
- patch (ESMF)
- conservative (CDOs, ESMF)
- bicubic (CDOs)
- distance weighted (CDOs)
- nearest neighbor (CDOs, ESMF)
- largest area fraction (CDOs)

These examples will only demonstrate the bilinear interpolation with a short NCL script and for CDOs with a short Korn-Shell script.

More information can be found on the following web pages:

ESMF: [http://www.ncl.ucar.edu/Applications/ESMF.shtml](http://www.ncl.ucar.edu/Applications/ESMF.shtml)


9.1 ESMF Regridding

The Earth System Modeling Framework (ESMF) is "software for building and coupling weather, climate, and related models". The ESMF "ESMF_RegridWeightGen" tool has been incorporated into NCL for generating weights for interpolating (regridding) data from a one grid to another.

The basic steps of NCL/ESMF regridding involve:

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

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

### 9.1.1 Curvilinear Grid to Rectilinear Grid

Regrid source data on a curvilinear grid to a destination rectilinear grid with grid distance 1° for latitude and longitude. Interpolation method is bilinear.

NUG_regrid_curvilinear_to_rectilinear_bilinear_weights_ESMF.ncl:

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

begin
  start_time = get_cpu_time() ;-- get cpu time
  diri = "./"
  fili = "thetao_curvilinear_ocean.nc"

  ;-- read data
  sfile = addfile(diri+fili,"r")
  thetao = sfile->thetao(0,0,:,:)
  thetao@lat2d = sfile->lat
  thetao@lon2d = sfile->lon
  printVarSummary(thetao)

  ;-- set resources
  Opt = True
  Opt@InterpMethod = "bilinear" ;-- interpolation method
  Opt@SrcFileName = "CMIP5_SCRIP_bilinear.nc" ;-- source file name
  Opt@DstFileName = "World1deg_SCRIP_bilinear.nc" ;-- destination file name
  Opt@WgtFileName = "CMIP5toWORLD_1x1_bilinear.nc" ;-- name of weights file, which will be generated
  Opt@ForceOverwrite = True ;-- force overwrite
  Opt@SrcMask2D = where(.not.ismissing(thetao),1,0) ;-- what to mask
  Opt@DstGridType = "1x1" ;-- Destination grid
  Opt@DstTitle = "World Grid 1x1-degree Resolution bilinear" ;-- destination title
  Opt@DstLLCorner = (/ -89.75d, 0.00d /) ;-- destination lower left corner
  Opt@DstURCorner = (/ 89.75d, 359.75d /) ;-- destination upper right corner
  print("-----------------------------------")
  print("Generating interpolation weights from CMIP5 to")
  print("World 1x1 degree grid.")
  print("")
  print("Method: bilinear")
  print("-----------------------------------")

  ;-- call ESMF_regrid
  thetao_regrid = ESMF_regrid(thetao,Opt)
  printVarSummary(thetao_regrid)
  nlon = dimsizes(thetao_regrid&lon)
```

```ncl
```
nlat = dimsizes(thetao_regrid&lat)

`;-- assign a output netcdf file for the new regridded data
`;-- (npoints = 180x360)
system("rm -rf regridded_bilinear_CMIP5_thetao_ESMF.nc")
fout = addfile("regridded_rectilinear_bilinear_ocean_thetao_ESMF.nc","c")

`;-- start to define output file settings
setfileoption(fout,"DefineMode",True)
`;-- explicitly declare file definition mode

`;-- create global attributes of the file
fAtt = True ;-- assign file attributes
fAtt@Conventions = "CF-1.4"
fAtt@comment = "Regrid curvilinear to 1x1 rectilinear grid using ESMF"
fAtt@title = "Regrid to 1x1 deg rectilinear grid"
fAtt@project_id = "NCL User Guide"
fAtt@source_file = fili
fAtt@creation_date = systemfunc("date")
fAtt@history = "NUG_regrid_CMIP5_bilinear_with_weights.ncl: "+fili+" to 1x1 deg rectilinear grid"
fileattdef(fout,fAtt) ;-- copy file attributes

`;-- predefine the coordinate variables and their dimensionality
dimNames = ("lat", "lon")
dimSizes = (/nlat, nlon/)
dimUnlim = (/False, False/)
filedimdef(fout,dimNames,dimSizes,dimUnlim)

`;-- predefine the the dimensionality of the variables to be written out
filevardef(fout,"lat",typeof(thetao_regrid&lat),
           getvardims(thetao_regrid&lat))
filevardef(fout,"lon",typeof(thetao_regrid&lon),
           getvardims(thetao_regrid&lon))
filevardef(fout,"thetao",typeof(thetao_regrid),
           getvardims(thetao_regrid))

`;-- copy attributes associated with each variable to the file
filevarattdef(fout,"lat",thetao_regrid&lat) ;-- copy lat attributes
filevarattdef(fout,"lon",thetao_regrid&lon) ;-- copy lon attributes
filevarattdef(fout,"thetao",thetao_regrid) ;-- copy thetao_regrid attributes

`;-- explicitly exit file definition mode (not required)
setfileoption(fout,"DefineMode",False)

`;-- output only the data values since the dimensionality and such
`;-- have been predefined; the "(/, ")" syntax tells NCL to only
`;-- output the data values to the predefined locations on the file
fout->lat = (/thetao_regrid&lat/) ;-- write lat to new netCDF file
fout->lon = (/thetao_regrid&lon/) ;-- write lon to new netCDF file
fout->thetao = (/thetao_regrid/) ;-- write variable to new netCDF file

`;-- get the resulting CPU time
end_time = get_cpu_time()
cpu_time = end_time - start_time
print("Elapsed time: "+ cpu_time + "s")
end
9.1.2 Curvilinear Grid to Rectilinear Grid from a given File

Comparing datasets from different models that are on different grids, ESMF can also be used to regrid the data onto a given destination grid. In this example a curvilinear dataset (MPIOM) will be regridded to a 192x96 rectilinear grid (ECHAM5).

NUG_regrid_curvilinear_to_rectilinear_bilinear_wgts_destgrid_ESMF.ncl:

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

begin
    start_time = get_cpu_time() ;-- get cpu time

diri = "/"  
fili = "thetao_curvilinear_ocean.nc"  
grid = "tas_rectilinear_grid_2D.nc"

;-- read destination grid data
g = addfile(diri+grid,"r")
dst_lat = g->lat

dst_lon = g->lon

;-- read data
sfile = addfile(diri+filii,"r")
thetao = sfile->thetao(0,0,:,:)
thetao@lat2d = sfile->lat
thetao@lon2d = sfile->lon
printVarSummary(thetao)

;-- set resources
Opt = True
Opt@InterpMethod = "bilinear" ;-- interpolation method
Opt@SrcFileName = "CMIP5_SCRIP_bilinear.nc" ;-- source file name
Opt@DstFileName = "World1deg_SCRIP_bilinear.nc" ;-- destination file name
Opt@WgtFileName = "CMIP5toWORLD_192x96_bilinear.nc" ;-- name of weights file, which will be generated
Opt@ForceOverwrite = True ;-- force overwrite
Opt@SrcMask2D = where(.not.ismissing(thetao),1,0) ;-- what to mask
Opt@DstGridType = "rectilinear" ;-- Destination grid
Opt@DstTitle = "World Grid 192x96 Resolution bilinear" ;-- destination title
Opt@DstGridLon = dst_lon
Opt@DstGridLat = dst_lat

print("--------------------------------------------------")
print("Generating interpolation weights from CMIP5 to")
print("World destination 192x96 grid.")
print("")
print("Method: bilinear")
print("--------------------------------------------------")

;-- call ESMF_regrid
thetao_regrid = ESMF_regrid(thetao,Opt)
printVarSummary(thetao_regrid)

nlon = dimsizes(thetao_regrid&lon)
nlat = dimsizes(thetao_regrid&lat)

;-- assign a output netcdf file for the new regridded data
;-- (npoints = 192x96)
```

178
system("rm -rf regridded_rectilinear_bilinear_ocean_thetao_destgrid_ESMF.nc")
fout = addfile("regridded_rectilinear_bilinear_ocean_thetao_destgrid_ESMF.nc", "c")

;;-- start to define output file settings
setfileoption(fout,"DefineMode",True) ;;-- explicitly declare file
;;-- definition mode

;;-- create global attributes of the file
fAtt = True ;;-- assign file attributes
fAtt@Conventions = "CF-1.4"
fAtt@comment = "Regrid curvilinear grid to 192x96 rectilinear grid"
fAtt@title = "Regrid to 192x96 deg rectilinear grid"
fAtt@project_id = "NCL User Guide"
fAtt@source_file = fill
fAtt@creation_date = systemfunc("date")
fAtt@history = "NUG_regrid_CMIP5_bilinear_with_weights.ncl:
"+fill+" to 1x1 deg rectilinear grid"
fileattdef(fout,fAtt) ;;-- copy file attributes

;;-- predefine the coordinate variables and their dimensionality
dimNames = ("lat", "lon")
dimSizes = (/nlat, nlon/)
dimUnlim = (/False, False/)
filedimdef(fout,dimNames,dimSizes,dimUnlim)

;;-- predefine the the dimensionality of the variables to be written out
filevardef(fout, "lat", typeof(thetao_regrid&lat), \
  getvardims(thetao_regrid&lat))
filevardef(fout, "lon", typeof(thetao_regrid&lon), \
  getvardims(thetao_regrid&lon))
filevardef(fout, "thetao", typeof(thetao_regrid), \
  getvardims(thetao_regrid))

;;-- copy attributes associated with each variable to the file
filevarattdef(fout,"lat", thetao_regrid&lat) ;;-- copy lat attributes
filevarattdef(fout,"lon", thetao_regrid&lon) ;;-- copy lon attributes
filevarattdef(fout,"thetao",thetao_regrid) ;;-- copy thetao_regrid

;;-- explicitly exit file definition mode (not required)
setfileoption(fout,"DefineMode",False)

;;-- output only the data values since the dimensionality and such
;;-- have been predefined. The "("/", ")" syntax tells NCL to only output
;;-- the data values to the predefined locations on the file.
fout->lat = (/thetao_regrid&lat/) ;;-- write lat to new netCDF file
fout->lon = (/thetao_regrid&lon/) ;;-- write lon to new netCDF file
fout->thetao = (/thetao_regrid/) ;;-- write variable to new netCDF file

;;-- get the resulting CPU time
end_time = get_cpu_time()
cpu_time = end_time - start_time
print("Elapsed time: "+ cpu_time + "s")

end

9.1.3 Unstructured Grid to Rectilinear Grid

In this section we will show you how to regrid the data from an unstructured triangular mesh with 20480 cells to a 1-degree rectangular grid, write the regridded data to a new netCDF file, and plot the original and regridded data in a panel.
load "${NCARG_ROOT}/lib/ncarg/nclscripts/esmf/ESMF_regridding.ncl"

begin
  start_time = get_cpu_time();
  -- get cpu time

  rad2deg = get_r2d("float");
  -- radians to degrees

  diri = "/home/NCL/general/scripts/new/NCL_Doc_Example_scripts_and_data/"
  fili = "triangular_grid_ICON.nc"

  -- read data
  f = addfile(diri+fili,"r")
  var = f->S(time|0,depth|0,ncells|:);
    -- set variable with dims:
    (time,depth,ncells)
  printVarSummary(var)

  x = f->clon * rad2deg;
    -- cell center, lon
  y = f->clat * rad2deg;
    -- cell center, lat
  x0 = "lon";
    -- set named dimension lon
  y0 = "lat";
    -- set named dimension lat
  x@units = "degrees_east"
    -- set lon units
  y@units = "degrees_north"
    -- set lat units

  vlon = f->clon_vertices * rad2deg;
    -- cell longitude vertices
  vlon = where(vlon.lt.0, vlon + 360, vlon);
    -- longitude: 0-360
  vlat = f->clat_vertices * rad2deg;
    -- cell latitude vertices
  nv = dimsizes(vlon(0,:));
    -- number of points in polygon

  -- set resources
  Opt = True
  Opt@InterpMethod = "bilinear";
    -- interpolation method
  Opt@ForceOverwrite = True;
    -- force overwrite
  Opt@Debug = True;
    -- print debug information
  Opt@PrintTimings = True

  Opt@SrcFileName = "CMIP5_SCRIP_bilinear.nc";
    -- source file name
  Opt@SrcInputFileName = diri+fili;
    -- optional, but good idea
  Opt@SrcRegional = False
  Opt@SrcGridLat = y
  Opt@SrcGridLon = x
  Opt@WgtFileName = "ICONtoWORLD_1x1_bilinear.nc";
    -- name of weights file, which will be generated
  Opt@DstFileName = "World1deg_SCRIP_bilinear.nc";
    -- dest. file name
  Opt@DstGridType = "rectilinear";
    -- destination grid
  Opt@DstTitle = "World Grid 1x1-degree Resolution bilinear"
    -- destination title
  Opt@DstRegional = False
  Opt@DstGridLon = fspan(-180.,180.,360)
  Opt@DstGridLat = fspan(-90.,90.,180)

  print("---------------------------------------------")
  print("Generating interpolation weights from ICON to")
  print("World 1x1 degree grid.")
  print("")
  print("Method: bilinear")
  print("---------------------------------------------")

  -- call ESMF_regrid
  var_regrid = ESMF_regrid(var,Opt)
    ;-- do the regridding
  printVarSummary(var_regrid)
nlon = dimsizes(var_regrid&lon) ;-- dim size new lon
nlat = dimsizes(var_regrid&lat) ;-- dim size new lat

;-- assign a output netcdf file for the new regridded data
;-- (npoints = 180x360)
system("rm -rf regridded_rectilinear_bilinear_ICON_S_ESMF.nc")
fout = addfile("regridded_rectilinear_bilinear_ICON_S_ESMF.nc", "c")

;-- start to define output file settings
setfileoption(fout,"DefineMode",True) ;-- explicitly declare file
;-- definition mode

;-- create global attributes of the file
fAtt = True ;-- assign file attributes
fAtt@Conventions = "CF-1.4"
fAtt@comment = "Regrid unstructured grid to 1x1 rectilinear grid - ESMF"
fAtt@title = "Regrid to 1x1 deg rectilinear grid"
fAtt@project_id = "NCL User Guide"
fAtt@source_file = filli
fAtt@creation_date = systemfunc("date")
fAtt@history = "NUG_regrid_ICON_bilinear_with_weights.ncl: "+filli+
" to 1x1 deg rectilinear grid"
fileattdef(fout,fAtt) ;-- copy file attributes

;-- predefine the coordinate variables and their dimensionality
dimNames = ("lat", "lon")
dimSizes = (/nlat, nlon/)
dimUnlim = (/False, False/)
filedimdef(fout,dimNames,dimSizes,dimUnlim)

;-- predefine the dimensionality of the variables to be written out
filevardef(fout, "lat", typeof(var_regrid&lat),getvardims(var_regrid&lat))
filevardef(fout, "lon", typeof(var_regrid&lon),getvardims(var_regrid&lon))
filevardef(fout, "S", typeof(var_regrid), getvardims(var_regrid))

;-- copy attributes associated with each variable to the file
filevarattdef(fout,"lat", var_regrid&lat) ;-- copy lat attributes
filevarattdef(fout,"lon", var_regrid&lon) ;-- copy lon attributes
filevarattdef(fout,"S", var_regrid) ;-- copy var_regrid attributes

;-- explicitly exit file definition mode (not required)
setfileoption(fout,"DefineMode",False)

;-- output only the data values since the dimensionality and such
;-- have been predefined. The "(/", ")" syntax tells NCL to only
;-- output the data values to the predefined locations on the file.
fout->lat = (var_regrid&lat/) ;-- write lat to new netCDF file
fout->lon = (var_regrid&lon/) ;-- write lon to new netCDF file
fout->S   = (var_regrid/) ;-- write variable to new netCDF file

;-- get the resulting CPU time
end_time = get_cpu_time()
cpu_time = end_time - start_time
print("Elapsed time: "+ cpu_time + ",s")

;-- open a PNG file
wks_type = "png"
wks_type@wkWidth = 1024
wks_type@wkHeight = 1024
wks = gsn_open_wks(wks_type,"plot_regrid_unstructured_to_rectilinear")

;-- set resources for contour plots
res = True
res@gsnDraw = False ;-- don't draw plot yet
res@gsnFrame = False ;-- don't advance the frame
res@gsnCenterString = "unstructured"
res@gsnAddCyclic = False
res@lbLabelBarOn = False ;-- no single label bar
res@cnFillOn = True ;-- turn color fill on
res@cnFillPalette = "BlueWhiteOrangeRed" ;-- choose color map
res@cnLinesOn = False ;-- turn lines off
res@cnLineLabelsOn = False ;-- turn labels off
res@cnLevelSelectionMode = "ManualLevels" ;-- use manual contour levels
res@cnMinLevelValF = 20. ;-- contour min. value
res@cnMaxLevelValF = 38. ;-- contour max. value
res@cnLevelSpacingF = 0.5 ;-- contour interval
res2 = res
res2@gsnCenterString = "rectilinear"
res@cnFillMode = "CellFill" ;-- set fill mode
res@sfXArray = x ;-- transform x to mesh scalar field
res@sfYArray = y ;-- transform y to mesh scalar field
res@sfXCellBounds = vlon ;-- needed if set "CellFill"
res@sfYCellBounds = vlat ;-- needed if set "CellFill"

;-- create the plots
plot0 = gsn_csm_contour_map(wks, var, res) ;-- original data
plot1 = gsn_csm_contour_map(wks, var_regrid, res2) ;-- regridded data

;-- create the panel plot
pres = True
pres@txString = "Regridding" ;-- panel title string
pres@gsnPanelLabelBar = True ;-- turn on a common
;-- labelbar for the entire panel plot

gsn_panel(wks,(/plot0,plot1/),(/2,1/),pres)
end
9.1.4 Unstructured Grid to Rectilinear Grid from a Given File

Regridding data on an unstructured grid onto a given destination grid read off a netCDF file is shown in the next example. The script does the regridding, writes the regridded data to a new netCDF file and plot the original and the regridded data on a panel.

NUG_regrid_unstructured_to_rectilinear_bilinear_wgts_destgrid_ESMF.ncl:

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/esmf/ESMF_regridding.ncl"
begin
  start_time = get_cpu_time() ;-- get cpu time
  outputfile = "regridded_rectilinear_bilinear_ICON_S_ESMF_destgrid.nc"
  rad2deg = get_r2d("float") ;-- radians to degrees
  diri = "$HOME/NCL/general/scripts/new/NCL_Doc_Example_scripts_and_data/
  filli = "triangular_grid_ICON.nc"
  grid = "tas_rectilinear_grid_2D.nc" ;-- use grid from file

  ;-- read destination grid data
  g
    = addfile(diri+grid,"r")
    dst_lat = g->lat
    dst_lon = g->lon

  ;-- read data
  f
    = addfile(diri+filli,"r")
  var
    = f->S(time|0,depth|0,ncells|:);-- set variable with dims
  printVarSummary(var)
  x
    = f->clon * rad2deg ;-- cell center, lon
  y
    = f->clat * rad2deg ;-- cell center, lat
```

![Diagram showing unstructured and rectilinear grids with salinity and pU values]

Salinity:

<table>
<thead>
<tr>
<th>Salinity Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Blue</td>
</tr>
<tr>
<td>1</td>
<td>Green</td>
</tr>
<tr>
<td>2</td>
<td>Yellow</td>
</tr>
<tr>
<td>3</td>
<td>Orange</td>
</tr>
<tr>
<td>4</td>
<td>Red</td>
</tr>
</tbody>
</table>

Polarization Units (pU):

<table>
<thead>
<tr>
<th>Polarization Unit</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Blue</td>
</tr>
<tr>
<td>1</td>
<td>Green</td>
</tr>
<tr>
<td>2</td>
<td>Yellow</td>
</tr>
<tr>
<td>3</td>
<td>Orange</td>
</tr>
<tr>
<td>4</td>
<td>Red</td>
</tr>
</tbody>
</table>
x!0 = "lon" ;-- set named dimension lon
y!0 = "lat" ;-- set named dimension lat
x@units = "degrees_east" ;-- set lon units
y@units = "degrees_north" ;-- set lat units

vlon = f->clon_vertices * rad2deg ;-- cell longitude vertices
vlon = where(vlon.lt.0, vlon + 360, vlon) ;-- longitude: 0-360
vlat = f->clat_vertices * rad2deg ;-- cell latitude vertices
nv = dimsizes(vlon(0,:)) ;-- number of points in polygon

;-- set resources
Opt = True
Opt@InterpMethod = "bilinear" ;-- interpolation method
Opt@ForceOverwrite = True ;-- force overwrite
Opt@Debug = True
Opt@PrintTimings = True

Opt@SrcFileName = "CMIP5_SCRIP_bilinear.nc" ;-- source file name
Opt@SrcInputFileName = diri+fili ;-- optional, but good idea
Opt@SrcRegional = False
Opt@SrcGridLat = y
Opt@SrcGridLon = x
Opt@WgtFileName = "ICONtoWORLD_bilinear_192x96.nc" ;-- name of weights file, which will be generated
Opt@DstFileName = "World1deg_SCRIP_bilinear.nc" ;-- destination file name
Opt@DstGridType = "rectilinear" ;-- destination grid
Opt@DstTitle = "World Grid 1x1-degree Resolution bilinear" ;-- destination title
Opt@DstRegional = False
Opt@DstGridLon = dst_lon
Opt@DstGridLat = dst_lat

print("----------------------------------------------")
print("Generating interpolation weights from ICON to")
print("World destination 192x96 degree grid.")
print(""
print("Method: bilinear")
print("----------------------------------------------")

;-- call ESMF_regrid
var_regrid = ESMF_regrid(var,Opt) ;-- do the regridding
printVarSummary(var_regrid)
nlon = dimsizes(var_regrid&lon) ;-- dims new lon
nlat = dimsizes(var_regrid&lat) ;-- dims new lat

;-- assign a output netcdf file for the new regridded data
;npoints = 180x360
system("rm -rf "+outputfile)
fout = addfile(outputfile, "c")

;-- start to define output file settings
setfileoption(fout,"DefineMode",True) ;-- explicitly declare file
;-- definition mode

;-- create global attributes of the file
fAtt = True ;-- assign file attributes
fAtt@Conventions = "CF-1.4"
fAtt@comment = "Regrid unstructured grid to 192x96 rectilin. grid - ESMF"
fAtt@title = "Regrid to 192x96 rectilinear grid"
fAtt@project id = "NCL User Guide"
fAtt@source file = filli
fAtt@creation_date = systemfunc("date")
fAtt@history = "NUG_regrid_ICON_bilinear_with_weights.ncl: "+fili+" to 1x1 deg rectilinear grid"

fileattdef(fout,fAtt) ;-- copy file attributes

;-- predefine the coordinate variables and their dimensionality
dimNames = ("/lat", "lon")
dimSizes = (/nlat, nlon/)
dimUnlim = (/False, False/)
filedimdef(fout,dimNames,dimSizes,dimUnlim)

;-- predefine the the dimensionality of the variables to be written out
filevardef(fout, "lat", typeof(var_regrid&lat), getvardims(var_regrid&lat))
filevardef(fout, "lon", typeof(var_regrid&lon), getvardims(var_regrid&lon))
filevardef(fout, "S", typeof(var_regrid), getvardims(var_regrid))

;-- copy attributes associated with each variable to the file
filevarattdef(fout,"lat", var_regrid&lat) ;-- copy lat attributes
filevarattdef(fout,"lon", var_regrid&lon) ;-- copy lon attributes
filevarattdef(fout,"S", var_regrid) ;-- copy var_regrid attributes

;-- explicitly exit file definition mode (not required)
setfileoption(fout,"DefineMode",False)

;-- output only the data values since the dimensionality and such
;-- have been predefined. The "(/", "/)" syntax tells NCL to only output
;-- the data values to the predefined locations on the file.
fout->lat = ("/var_regrid&lat") ;-- write lat to new netCDF file
fout->lon = ("/var_regrid&lon") ;-- write lon to new netCDF file
fout->S = ("/var_regrid") ;-- write variable to new netCDF file

;-- get the resulting CPU time
end_time = get_cpu_time()
cpu_time = end_time - start_time
print("Elapsed time: "+ cpu_time + "s")

;----------------------------------------------------------
;-- control the netCDF output file; open file and read variable
;----------------------------------------------------------
p = addfile(outputfile,"r")
new_var = p->S

;-- open a PNG file
wks_type = "png"
wks_type@wkWidth = 1024
wks_type@wkHeight = 1024
wks = gsn_open_wks(wks_type,"plot_regrid_unstruc_to_recti_destgrid")

;-- set resources for contour plots
res = True
res@gsnDraw = False
res@gsnFrame = False
res@gsnCenterString = "unstructured"
res@gsnAddCyclic = False
res@lbLabelBarOn = False ;-- no single label bar
res@cnFillOn = True
res@cnFillPalette = "BlueWhiteOrangeRed" ;-- choose a color map
res@cnLinesOn = False ;-- turn lines off
res@cnLineLabelsOn = False ;-- turn labels off
res@cnLevelSelectionMode = "ManualLevels" ;-- manual contour levels
res@cnMinLevelValF = 20. ;-- minimum level
res@cnMaxLevelValF = 38. ;-- maximum level
res@cnLevelSpacingF = 0.5 ;-- contour spacing
res2 = res
res@cnFillMode = "CellFill" ;-- set fill mode
res@sfXArray = x ;-- transform x to mesh scalar field
res@sfYArray = y ;-- transform y to mesh scalar field
res@sfXCellBounds = vlon ;-- needed if set "CellFill"
res@sfYCellBounds = vlat ;-- needed if set "CellFill"

;-- create the plot of the original data
plot0 = gsn_csm_contour_map(wks, var, res)

;-- create the plot of the regridded data
res2@gsnCenterString = "rectilinear"
res2@gsnAddCyclic = True
plot1 = gsn_csm_contour_map(wks, new_var, res2)

;-- create the panel plot
pres = True
pres@txString = "Regridding" ;-- panel title string
pres@gsnPanelLabelBar = True ;-- turn on a common labelbar
gsn_panel(wks,(/plot0,plot1/),(/2,1/),pres)
end

9.1.5 Rectilinear Grid to Curvilinear Grid from a Given File
Sometimes you need to regrid a rectilinear grid to a curvilinear grid, both read from given netCDF files. The declaration of the netCDF output file is now a little bit different, because the output latitude and longitude arrays are 2-dimensional.

NUG_regrid_rectilinear_to_curvilinear_bilinear_wgts_destgrid_ESMF.ncl:

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

begin
    start_time = get_cpu_time()               ; -- get cpu time

    ;-- read data
    diri = ".:/data/"
    fili = "tas_rectilinear_grid_2D.nc"
    f   = addfile(diri+fili,"r")
    var = f->tas(0,:,:)

    ;-- name of output file
    outputfile = "regridded_rectilin_to_curvilin_bilin wgts_destgrid_ESMF.nc"

    ;-- read the destination grid lat/lon arrays
    dstfili = "thetao_curvilinear_ocean.nc" ; -- get dest. grid from file
    d       = addfile(diri+dstfili,"r")
    dst_lat = d->lat
    dst_lon = d->lon
    dvar    = d->thetao(0,0,:,:)
    dims    = dimsizes(dst_lat)
    nlat    = dims(0)
    nlon    = dims(1)

    printVarSummary(dvar)
    print("---->

    print(""
    print("""
    print("----> Destination dims:  lat "+nlat+"  lon "+nlon)
    print(""

    ;-- set resources
    Opt               = True
    Opt@InterpMethod  = "bilinear"         ; -- interpolation method
    Opt@SrcFileName   = "ECHAM5_SCRIP_bilinear.nc" ; -- source file name
    Opt@DstFileName   = "WorldCurvilinear_SCRIP_bilinear.nc" ; -- dest. file
    Opt@WgtFileName   = "ECHAM5toWorldCurvilinear_bilinear.nc" ; -- name of
    ; -- weights file, which will be generated
    Opt@ForceOverwrite = True               ; -- force overwrite
    Opt@DstMask2D     = where(ismissing(dvar),0,1) ; -- create mask from dest.
    Opt@DstGridType   = "curvilinear"       ; -- Destination grid
    Opt@DstTitle      = "World Grid Curvilinear Resolution bilinear"
                       ; -- destination title
    Opt@DstGridLon    = dst_lon
    Opt@DstGridLat    = dst_lat

    print("""
    print("Generating interpolation weights from ECHAM5 to")
    print("World destination curvilinear grid.")
    print(""
    print("Method: bilinear")
    print("""

    ;-- call ESMF_regrid
    var_regrid = ESMF_regrid(var,Opt)
    var_regrid!0 = "y"                        ; -- named coordinate
    var_regrid!1 = "x"                        ; -- named coordinate

    printVarSummary(var_regrid)
```
delete(var_regrid@lat2d) ;-- delete attribute array lat2d
delete(var_regrid@lon2d) ;-- delete attribute array lon2d

;-- assign a output netcdf file for the new regridded data
system("rm -rf "+outputfile)
fout = addfile(outputfile, "c")

;-- start to define output file settings
setfileoption(fout,"DefineMode",True) ;-- explicitly declare file
                                    ;-- definition mode

;-- create global attributes of output file
fAtt = True ;-- assign file attributes
fAtt@Conventions = "CF-1.4"
fAtt@comment = "Regrid rectilinear grid to curvilinear grid - ESMF"
fAtt@title = "Regrid to curvilinear grid"
fAtt@project_id = "NCL User Guide"
fAtt@source_file = fill
fAtt@creation_date = systemfunc("date")
fAtt@history = "NUG_regrid_rectilinear_to_curvilinear_bilinear_wgts_destgrid_ESMF.ncl: "+
                fill+" to curvilinear grid"
fileattdef(fout,fAtt) ;-- copy file attributes

;-- predefine the coordinate variables and their dimensionality
dimNames = ("y", "x") ;-- curvilinear grid: dimensions not lat/lon
dimSizes = (/nlat, nlon/) ;-- dimension size of destination y/x
dimUnlim = (/False, False/)  
filedimdef(fout,dimNames,dimSizes,dimUnlim)

;-- predefine the the dimensionality of the variables to be written out
filevardef(fout,"lat",typeof(dst_lat),getvardims(dst_lat)) ;-- variable
                ;-- lat not dimension
filevardef(fout,"lon",typeof(dst_lon),getvardims(dst_lon)) ;-- variable
                ;-- lon not dimension
filevardef(fout,"var",typeof(var_regrid),getvardims(var_regrid))

;-- copy attributes associated with each variable to output file
filevarattdef(fout,"lat",dst_lat) ;-- copy attributes from dest. lat
filevarattdef(fout,"lon",dst_lon) ;-- copy attributes from dest. lon
filevarattdef(fout,"var",var_regrid) ;-- copy var_regrid attributes

;-- explicitly exit file definition mode (not required)
setfileoption(fout,"DefineMode",False)

;-- output only the data values since the dimensionality and such
;-- have been predefined; the "(/", "/)" syntax tells NCL to only output
;-- the data values to the predefined locations on the file.
fout->lat = (/dst_lat/) ;-- write lat to new netCDF file
fout->lon = (/dst_lon/) ;-- write lon to new netCDF file
fout->var = (/var_regrid/) ;-- write variable to new netCDF file

;-- get the resulting CPU time
end_time = get_cpu_time()
cpu_time = end_time - start_time
print("Elapsed time: "+ cpu_time + "s")

;-- control the netCDF output file; open file and read variable
----------------------------------------
----------------------------------------
p = addfile(outputfile,"r")
new_var = p->var
new_var@lat2d = p->lat
new_var@lon2d = p->lon

;-- open a workstation
wks_type = "png"
wks_type@wkWidth = 1024
wks_type@wkHeight = 1024
wks = gsn_open_wks(wks_type, "plot_regrid_rectilin_to_curvilin_destgrid")

;-- set resources for contour plots
res = True
res@gsnDraw = False
res@gsnFrame = False
res@gsnLeftString = "tas (original)"
res@gsnCenterString = "rectilinear"
res@gsnAddCyclic = True
res@cnFillOn = True
res@cnLevelSelectionMode = "ManualLevels" ;-- manual contour levels
res@cnMinLevelValF = 230. ;-- minimum level
res@cnMaxLevelValF = 310. ;-- maximum level
res@cnLevelSpacingF = 5. ;-- contour spacing
res@lbLabelBarOn = False ;-- no single label bar

;-- create the plot with original data on rectilinear grid
plot0 = gsn_csm_contour_map(wks, var, res)

;-- create the plot with regridded data on curvilinear grid
res2 = res
res2@gsnLeftString = "tas (regridded)"
res2@gsnCenterString = "curvilinear (MPIOM)"
res2@gsnAddCyclic = True

plot1 = gsn_csm_contour_map(wks, new_var, res2); -- from output netCDF file

;-- create the panel plot
pres = True
pres@txString = "Regridding" ;-- panel title string
pres@gsnPanelLabelBar = True ;-- turn on a common labelbar

;-- for the entire panel plot

gsn_panel(wks, (/plot0, plot1/), (/2, 1/), pres)
end
9.1.6 CMIP5 Grid to 1x1 degrees Grid

This example demonstrates the use of the ESMF regridding function to interpolate a global CMIP5 grid into a regular global 1x1° grid.

NUG_regrid_bilinear_CMIP5_grid_to_1x1deg_grid.ncl:

```ncl
load "$NCARG_ROOT/lib/ncarg/nclscripts/esmf/ESMF_regridding.ncl"
begin
    start_time = get_cpu_time()               ;-- get cpu time

    -- name of output file
    outfile = "regridded_CMIP5_to_rectilinear_bilinear_wgts_ESMF.nc"

    -- read data
    diri = "../data/
    fili = "thetao_curvilinear_ocean.nc"
    sfile = addfile(diri+fili,"r")
    thetao = sfile->thetao(0,0,:,:)
    thetao@lat2d = sfile->lat
    thetao@lon2d = sfile->lon

    -- set resources to generate the weights and grid files
    Opt = True
    Opt@InterpMethod = "bilinear"
    Opt@SrcFileName = "CMIP5_SCRIP_bilinear.nc" ;-- source grid file name
    Opt@DstFileName = "World1deg_SCRIP_bilinear.nc" ;-- dest. grid file name
    Opt@WgtFileName = "CMIP5toWORLD_1x1_bilinear.nc";-- name of weights file,
    Opt@ForceOverwrite = True
```
Opt@SrcMask2D = where(.not.ismissing(thetao),1,0)
Opt@DstGridType = "1x1" ;-- dest. grid type
Opt@DstTitle = "World Grid 1x1-degree Resolution bilinear"
Opt@DstLLCorner = (/ -89.75d, 0.00d /)
Opt@DstURCorner = (/ 89.75d, 359.75d /)

;-- interpolate data from CMIP5 to World 1x1 degree grid using ESMF
print("Generating interpolation weights from CMIP5 to World 1x1deg. grid")
print("Method: bilinear")
thetao_regrid = ESMF_regrid(thetao,Opt)
printVarSummary(thetao_regrid)

;-- write regridded data to file
system("rm -rf "+outfile)
fout = addfile(outfile, "c")
fout->thetao = thetao_regrid

;-- get the resulting CPU time
end_time = get_cpu_time()
cpu_time = end_time - start_time
print("Elapsed time: "+ cpu_time + "s")

;-- open a workstation
wks_type = "png"
wks_type@wkWidth = 1024
wks_type@wkHeight = 1024
wks = gsn_open_wks(wks_type,"plot_regrid_CMIP5_to_rectilinear_ESMF")

;-- set resources for contour plots
res = True
res@gsnDraw = False
res@gsnFrame = False
res@gsnLeftString = "thetao (original)"
res@gsnCenterString = "rectilinear"
res@gsnAddCyclic = True
res@cnFillOn = True
res@cnFillPalette = "BlueWhiteOrangeRed" ;-- choose color map
res@cnLineLabelsOn = False
res@cnLevelSelectionMode = "ManualLevels" ;-- manual contour levels
res@cnMinLevelValF = 230. ;-- minimum level
res@cnMaxLevelValF = 310. ;-- maximum level
res@cnLevelSpacingF = 5. ;-- contour spacing
res@lbLabelBarOn = False ;-- no single label bar

;-- create the plot with original data on rectilinear grid
plot0 = gsn_csm_contour_map(wks, thetao, res)

;-- create the plot with regridded data on curvilinear grid
res2 = res
res2@gsnLeftString = "thetao (regridded)"
res2@gsnCenterString = "curvilinear (MPIOM)"
res2@gsnAddCyclic = True

plot1 = gsn_csm_contour_map(wks, thetao_regrid, res2) ;-- from output

;-- create the panel plot
The script creates a new data file named **plot_regrid_CMIP5_to_rectilinear_ESMF**.

Elapsed time: 6s

9.2 CDO Regridding

CDO (Climate Data Operators) is a collection of **Command Line Operators** to manipulate and analyse Climate and NWP model Data. Supported data formats are GRIB 1/2, netCDF 3/4, SERVICE, EXTRA and IEG. There are more than 600 operators available.

The built-in operator module REMAPGRID contains operators to remap (=regrid) all input fields to a new horizontal grid.

Remapping operators:

- **remapbil**  Bilinear interpolation
- **remapbic**  Bicubic interpolation
- **remapdis**  Distance-weighted remapping
- **remapnn**   Nearest neighbor remapping
- **remapcon**  First order conservative remapping
- **remapbil**  Second order conservative remapping
- **remapbil**  Largest area fraction remapping

See also [https://code.zmaw.de/projects/cdo section REMAPGRID](https://code.zmaw.de/projects/cdo) in the documentation.

9.2.1 Curvilinear Grid to Gaussian N32 Grid
Let us assume that the data file \textit{ifile} contains fields on a quadrilateral curvilinear grid. To remap all fields bilinear to a Gaussian N32 grid, type in a terminal window:

```
cdo remapbil,n32 ifile ofile
```

To remap the input data to a 1x1° rectilinear grid:

```
cdo remapbil,r360x180 ifile ofile
```

To remap all fields bilinear to a Gaussian N32 grid using generated interpolation weights type in a terminal window:

```
cdo genbil,n32 ifile remapweights.nc
cdo remap,n32,remapweights.nc ifile ofile
```

### 9.2.2 Curvilinear Grid to Rectilinear Grid (Korn-Shell Script)

This example demonstrates the use of the CDO remapping function to interpolate a global CMIP5 grid into a regular world grid 1x1° equivalent to the ESMF regridding. The commands

```
cdo genbil,r360x180 data.nc weights_bil.nc
cdo remap,r360x180,weights_bil.nc data.nc regridded_data.nc
```

will generate a bilinear weight file \textit{weights\_bil.nc}, which will be used to interpolate the data.

The next command line will generate a bilinear weight file, but only in memory, not saved on the disk:

```
cdo remapbil,r360x180 ${fin} ${fout}
```

The script creates a new netCDF file named \textit{remap\_bilinear\_CMIP5\_thetao.nc}. Elapsed time: 48s
10 Using External Fortran or C Code

NCL supports calling subroutines in external code (e.g. Fortran or C). In this section we will show how to call external Fortran routines using the NCL WRAPIT tool. See also:

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

To use WRAPIT, you must follow the next four steps:

1. Write special wrapper text file
2. Run WRAPIT
3. Load the generated shared object file
4. Call the subroutine/function

10.1 Fortran

In this example, the korn shell script NUG_use_Fortran_subroutines.ksh does it all: it first creates a Fortran code and pass it to WRAPIT in order to generate a shared object file. The script also creates the ncl script ncl_script, which finally is executed and invokes the wrapped Fortran executable.

Fortran Wrapper example: NUG_use_Fortran_subroutines.ksh

```
#!/usr/bin/ksh
#----------------------------------------------------------
#-- NCL Doc Example script: NUG_use_Fortran_subroutines.ksh
#--
#-- Write a short Fortran subroutine --> ex01.f
#-- Run the wrapper --> ex01.so
#-- Write NCL script --> NUG_use_Fortran_subroutines.ncl
#-- Run NCL script --> write results to stdout
#----------------------------------------------------------
example=${0%.*}
ncl_script=${example}.ncl
#------------
#-- write the NCL script
#--
#----------------------------------------------------------
cat << EOF > ${ncl_script}
external EX01 "./ex01.so"
begin
    print(""")
    ;-- Calculate three values of a quadratic equation
    nump = 3
    x = (/ -1., 0.0, 1.0 /)
    qval = new(nump,float)
    ;-- Call the NCL version of your Fortran subroutine.
    EX01::cquad(-1., 2., 3., nump, x, qval)
    print("Polynomial value = " + qval) ;-- should be (/0,3,4/)
    ;-- Calculate an arc length.
    xc = (/ 0., 1., 2. /)
    yc = (/ 0., 1., 0. /)
    ;-- Call the NCL version of your Fortran function.
    arclen = EX01::arcln(nump,xc,yc)
    print("Arc length = " + arclen) ;-- should be 2.82843
print(""")
end
```
```fortran
C NCLFORTSTART
subroutine cquad (a, b, c, nq, x, quad)
   real x(nq), quad(nq)
C NCLEND
C
C  Calculate quadratic polynomial values.
C
   do 10 i=1,nq
      quad(i) = a*x(i)**2 + b*x(i) + c
   10 continue
   return
end
C NCLFORTSTART
function arcln (numpnt, pointx, pointy)
dimension pointx(numpnt),pointy(numpnt)
C NCLEND
C
C  Calculate arc lengths.
C
   if (numpnt .lt. 2) then
      print *, 'arcln: number of points must be at least 2'
      stop
   endif
   arcln = 0.
   do 10 i=2,numpnt
      pdist = sqrt((pointx(i)-pointx(i-1))**2 +
                    (pointy(i)-pointy(i-1))**2)
      arcln = arcln + pdist
   10 continue
   return
end
EOF
```

---

run the NCL wrapper. Generates the ex01.so shared object file.

WRAPIT ex01.f

---

run ncl

ncl -n ${ncl_script}

exit

---

Result on stdout:

```
WRAPIT Version: 120209
COMPILING ex01.f
LINKING
END WRAPIT

Copyright (C) 1995-2017 - All Rights Reserved
```
10.2 C Code

The same example can also be set up for C code, but since NCLs WRAPIT is not able to work directly with C, a little more effort is necessary and we need to apply some tricks.

1. Write the C code
2. Create a Fortran stub file with the calling sequences and types
3. Run "wrapit77" on the Fortran stub file to create the C wrapper
4. Make some changes in the C wrapper file
5. Run "WRAPIT" with the '-d' option to get compiling informations
6. Create a Makefile to compile the C code and create the shared library
7. Use the external subroutine and functions within NCL

All steps are done by the following Korn-Sheel script.

C Wrapper example: NUG_use_C_subroutines.ksh

```bash
#!/bin/ksh
#------------------------------------------
#-- NCL Doc Example script: NUG_use_C_subroutines.ksh
#--
#-- original: http://www.ncl.ucar.edu/Document/Tools/WRAPIT.shtml#Example_6
#--
#-- Write a short C subroutine --> ex01C.c
#-- Write a Fortran stub file --> ex01C.stub
#-- Run the wrapper for ex01.stub --> ex01CW.c
#-- Modify ex01W.c --> ex01CW.c
#-- Run the wrapper with option '-d' --> returns compilation information
#-- Compile the files --> ex01C.so
#-- Write NCL script --> NUG_use_C_subroutines.ncl
#-- Run NCL script --> write results to stdout
#--
#-- 23.07.13
#------------------------------------------

eexample=${0%.*}
ncl_script=${eexample}.ncl

#-- clean up
e rm -rf ex01C.c ex01C.o ex01C.c~ ex01C.stub ex01CW.c ex01CW.o WRAPIT.stub
e rm -rf WRAPIT.c WRAPIT.o ex01C.so NUG_use_C_subroutines.ncl objects
e rm -rf WRAPIT_debug_output Makefile

#-- create the C code. It is the same code as in the Fortran
#-- example for the functions cquad and arcln but implemented
#-- in C.
#------------------------------------------
cat << EOF > ex01C.c
```
void *cquad(float a, float b, float c, int nq, float *x, float *quad)
{
    int i;
    /* Calculate quadratic polynomial values. */
    for(i = 0; i < nq; i++ ) quad[i] = a*pow(x[i],2) + b*x[i] + c;
}

float arcln(int numpnt, float *pointx, float *pointy)
{
    int i;
    float pdist, a;
    /* Calculate arc lengths. */
    if(numpnt < 2) {
        printf("arcln: number of points must be at least 2\n");
        return;
    }
    a = 0.;
    for( i=1; i < numpnt; i++ ) {
        pdist = sqrt(pow(pointx[i]-pointx[i-1],2) + pow(pointy[i]-pointy[i-1],2));
        a += pdist;
    }
    return(a);
}

/* http://www.ncl.ucar.edu/Document/Tools/WRAPIT.shtml#Example_6 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void *cquad(float a, float b, float c, int nq, float *x, float *quad)
{
    int i;
    /* Calculate quadratic polynomial values. */
    for(i = 0; i < nq; i++ ) quad[i] = a*pow(x[i],2) + b*x[i] + c;
}

float arcln(int numpnt, float *pointx, float *pointy)
{
    int i;
    float pdist, a;
    /* Calculate arc lengths. */
    if(numpnt < 2) {
        printf("arcln: number of points must be at least 2\n");
        return;
    }
    a = 0.;
    for( i=1; i < numpnt; i++ ) {
        pdist = sqrt(pow(pointx[i]-pointx[i-1],2) + pow(pointy[i]-pointy[i-1],2));
        a += pdist;
    }
    return(a);
}

#-- create a Fortran stub file containing the same calling
#-- sequence and types for the C subroutines and functions
#----------------------------------------------------------
cat << EOF > ex01C.stub
C NCLFORTSTART
    subroutine cquad (a, b, c, nq, x, quad)
    real a, b, c
    real x(nq), quad(nq)
C NCLEND
C NCLFORTSTART
    function arcln (numpnt, pointx, pointy)
    integer numpnt
    real pointx(numpnt),pointy(numpnt)
C NCLEND
EOF

echo "-- write C code - done"

echo "-- write Fortran stub file - done"

#-- run the NCL wrapper on the Fortran stub to create the
#-- C wrapper
#----------------------------------------------------------
wrapit77 < ex01C.stub > ex01CW.c

echo "-- create the C wrapper - done"
#-- modify ex01CW.c to make a few changes. The lines with
#-- NGCALLF should be changed:
#-----------------------------------------------
cat ex01CW.c | sed -e 's/NGCALLF(cquad,CQUAD)(a,b,c,nq,x,quad)/(void)cquad(*a,*b,*c,*nq,x,quad)/g' >
tmp.c
cat tmp.c | sed -e 's/extern float NGCALLF(arcln,ARCLN)()/extern float arcln(int
numpnt, float *pointx, float *pointy)/g' > tmp1.c
cat tmp1.c | sed -e 's/arcln_ret = NGCALLF(arcln,ARCLN)(numpnt,pointx,pointy)/arcln_ret =
arcln(*numpnt,pointx,pointy)/g' > tmp2.c
cat tmp2.c | sed -e 's/NhlErrorTypes cquad_W( void ) {/extern NhlErrorTypes
quad_W( void ) {/g' > tmp3.c
rm -rf tmp.c tmp1.c tmp2.c
mv tmp3.c ex01CW.c
echo "-- modify ex01CW.c - done"

#-- run the NCL wrapper
#-----------------------------------------------
WRAPIT -d ex01C.stub > WRAPIT_debug_output

#-- create a Makefile and run it
#-----------------------------------------------
compline=$(cat WRAPIT_debug_output | grep gcc | grep WRAPIT.c)
compline1=$(echo ${compline} | sed -e 's/WRAPIT.c/ex01C.c/g')
compline2=$(echo ${compline} | sed -e 's/WRAPIT.c/ex01CW.c/g')
linkline=$(cat WRAPIT_debug_output | grep gcc | grep WRAPIT.o)
linkline1=$(echo ${linkline} | sed -e 's/WRAPIT.o/ex01CW.o ex01C.o/g')
cat << EOF > Makefile
ex01C.so: ex01CW.o ex01C.o
 ${linkline1}
ex01C.o: ex01C.c
 ${compline1}
ex01CW.o: ex01CW.c
 ${compline2}
EOF
echo "-- write Makefile - done"
make >> /dev/null
echo "-- make - done"

#-- write the NCL script.
#-- Use the external functions cquad and arcln
#-----------------------------------------------
cat << EOF > $(ncl_script)
external EX01C "./ex01C.so"
begin
  print(""
  ;-- Calculate three values of a quadratic equation
  nump = 3
  x    = (/ -1., 0.0, 1.0 /)
  qval = new(nump,float)
  ;-- Call the NCL version of your Fortran subroutine.
EX01C::cquad(-1., 2., 3., nump, x, qval)
  print("Polynomial value = " + qval) ;-- should be (/0,3,4/)
;-- Calculate an arc length.
  xc = (/ 0., 1., 2. /)
  yc = (/ 0., 1., 0. /)
;-- Call the NCL version of your Fortran function.
  arclen = EX01C::arcln(nump,xc,yc)
  print("Arc length = " + arclen) ;-- should be 2.82843
  print(""
end
EOF

echo "-- write NCL script    - done"
#
#---------------------------------------------
#-- run ncl
#---------------------------------------------
echo "--write C code    - done
-- write Fortran stub file - done
-- create the C wrapper  - done
-- modify ex01CW.c      - done
-- write Makefile       - done
-- make                 - done
-- write NCL script     - done
---------------------------------------------
Copyright (C) 1995-2017 - All Rights Reserved
University Corporation for Atmospheric Research
NCAR Command Language Version 6.4.0
The use of this software is governed by a License Agreement.
See http://www.ncl.ucar.edu/ for more details.

Polynomial value = 0
Polynomial value = 3
Polynomial value = 4
Arc length = 2.82843
11 Creating Images for PowerPoint, Keynote, Web

For importing NCL visualizations to PowerPoint, Keynote or web pages, it is recommended to use the output format 'png'. For printable documents you should select larger width and height values to get a better resolution:

```plaintext
wkstype = "png"
wkstype@wkWidth = 2500
wkstype@wkHeight = 2500
wks = gsn_open_wks(wkstype, "plot_file_name")
```

If the output format "png" doesn't produce good results, it may be better to use "ps" or "pdf" output format. The PS or PDF file can be converted using the free ImageMagick software package, which is installed on most machines.

```plaintext
convert -geometry 2500x2500 -density 300 -trim plot.ps plot.png
```

For posters you have to increase the values for geometry and density.

To crop white space from the plot:

```plaintext
convert -alpha off -background white -geometry 1000x1000 \ -density 300 -trim plot.ps plot.png
```
12 Customizing the NCL Graphics Environment

NCL requires only one environment variable, which is NCARG_ROOT, which should be set to the parent directory of the NCL installation.

Here some of the important NCL environment variables which can be changed in order to customize your NCL environment:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCARG_ROOT</td>
<td>parent directory of NCL installation</td>
</tr>
<tr>
<td></td>
<td>default: /usr/local</td>
</tr>
<tr>
<td>NCARG_NCARG</td>
<td>location of supplemental directories, like databases, examples, resource files, etc.</td>
</tr>
<tr>
<td></td>
<td>default: $NCARG_ROOT/lib/ncarg</td>
</tr>
<tr>
<td>NCARG_USRRESFILE</td>
<td>NCL HLU resource file</td>
</tr>
<tr>
<td></td>
<td>default: ~/.hluresfile</td>
</tr>
<tr>
<td>NCARG_COLORMAP_PATH</td>
<td>list of paths of color map files</td>
</tr>
<tr>
<td></td>
<td>default: $NCARG_NCARG/colormaps</td>
</tr>
<tr>
<td>NCARG_RANGS</td>
<td>point to the RANGS/GSHHS database, if installed</td>
</tr>
<tr>
<td></td>
<td>default: $NCARG_NCARG/rangs</td>
</tr>
<tr>
<td>NCARG_EXAMPLES</td>
<td>point to the low-level examples</td>
</tr>
<tr>
<td></td>
<td>Default: $NCARG_NCARG/examples</td>
</tr>
<tr>
<td>NCL_GRIB_PTABLE_PATH</td>
<td>point to GRIB parameter table file</td>
</tr>
<tr>
<td></td>
<td>default: none</td>
</tr>
<tr>
<td>NIO_GRIB2_CODETABLES</td>
<td>location of the GRIB2 code tables</td>
</tr>
<tr>
<td></td>
<td>default: $NCARG_ROOT/lib/ncarg/grib2_codetables</td>
</tr>
</tbody>
</table>

All NCL environment variables are listed on the web page:

13 Tips

13.1 Reverse Latitudes in Data File

Some functions require that the order of the latitudes has to be **South to North**. If needed, it is very easy to reverse the latitudes array by re-ordering it with the following command:

\[ \text{latitudes} = \text{var}[@\text{lat}(:,::-1)] \]

Pre-conditions: variable \textit{var} has a named dimension called \textit{lat}

13.2 Convert NaNs (not a number) to \_FillValue

Some datasets contain non-numeric values called **NaNs**, which can cause problems when trying to use a function or graphically display the data. NCL provides two built-in functions for dealing with NaNs: \textit{isnan_ieee} for checking for the existence of NaNs, and \textit{replace_ieeenan} for converting all NaNs to a desired value.

If the variable \textit{var} is of type float:

\[
\text{if} \ (\text{any(isnan_ieee(var))) then} \\
\quad \text{value} = -99999.9 \\
\quad \text{replace_ieeenan (var, value, 0)} \\
\quad \text{var[@\_FillValue} = \text{value} \\
\quad \text{var[@missing_value} = \text{value} \\
\text{end if}
\]

Here is an example to convert variables from an input netCDF file which contain NaNs and write the converted variables to a new netCDF file.

\text{NUG\_change\_NaNs\_to\_FillValue.ncl}

```ncl
begin
  infile  = "data_with_NaNs.nc" 
  outfile = "nan_to_FillValue.nc"

;------------------------------------------------
;-- read data
;------------------------------------------------
  f      = addfile(infile,"r")
  time   = f->time           ;-- get dimension time
  lat    = f->lat            ;-- get dimension lat
  lon    = f->lon            ;-- get dimension lon
  ntim   = dimsizes(time)    ;-- get dimension sizes of time
  nlat   = dimsizes(lat)     ;-- get dimension sizes of lat
  nlon   = dimsizes(lon)     ;-- get dimension sizes of lon

;------------------------------------------------
;-- set correct units
;------------------------------------------------
  lat@units = "degrees_north"
  lon@units = "degrees_east"

;------------------------------------------------
;-- get variables
;------------------------------------------------
  var1 = f->COR_SST_FC
```

203
var2 = f->COR_SST_FC_SIGN

;-- copy variables

var1_miss    = tofloat(var1)
var2_miss    = tofloat(var2)

;-- set named coordinates for var1_miss and var2_miss

var1_miss!0   = "time"
var1_miss!1   = "lat"
var1_miss!2   = "lon"
var1_miss&time = time
var1_miss&lat  = lat
var1_miss&lon  = lon

var2_miss!0   = "time"
var2_miss!1   = "lat"
var2_miss!2   = "lon"
var2_miss&time = time
var2_miss&lat  = lat
var2_miss&lon  = lon

;-- convert NaNs to _FillValue/missing_value

if (any(isnan_ieee(var1_miss))) then
  value = -99999.9
  replace_ieeenan (var1_miss, value, 0)
  var1_miss@_FillValue    = value
  var1_miss@missing_value = value
end if

if (any(isnan_ieee(var2_miss))) then
  value = -99999.9
  replace_ieeenan (var2_miss, value, 0)
  var2_miss@_FillValue    = value
  var2_miss@missing_value = value
end if

printVarSummary(var1_miss)
printVarSummary(var2_miss)

;-- write var1_miss and var2_miss to new file

;-- create new netCDF file

system("rm -rf "+outfile)
fout = addfile(outfile,"c")

;-- begin output file settings

setfileoption(fout,"DefineMode",True) ;-- explicitly declare file
                                     ;-- definition mode

;-- create global attributes of the file

fAtt      = True                   ;-- assign file attributes
fAtt@title = "NCL convert NaNs to _FillValue"
fAtt@source_file = infile
fAtt@Conventions = "CF"
fAtt@creation_date = systemfunc ("date")
fAtt@history   = "NCL script: change_NaN_to_FillValue.ncl"
fAtt@comment   = "convert NaNs to _FillValue"
fileattdef(fout,fAtt)        ;-- copy file attributes to new file

--;-- predefine the coordinate variables and their dimensionality
dimNames = ("time", "lat", "lon")    ;-- define dimension names
dimSizes = (-1, nlat, nlon)    ;-- time unlimited (-1)
dimUnlim = (True, False, False)    ;-- True: unlimited
filedimdef(fout,dimNames,dimSizes,dimUnlim) ;-- copy to new file

--;-- predefine the the dimensionality of the variables to be
--;-- written out
filevardef(fout, "time", typeof(time), getvardims(time))
filevardef(fout, "lat", typeof(lat), getvardims(lat))
filevardef(fout, "lon", typeof(lon), getvardims(lon))
filevardef(fout, "COR_SST_FC", typeof(var1_miss), getvardims(var1_miss))
filevardef(fout, "COR_SST_FC_SIGN", typeof(var2_miss), \
    getvardims(var2_miss))

--;-- copy attributes associated with each variable to the file
filevarattdef(fout,"time", time)       ;-- copy time attributes
filevarattdef(fout,"lat", lat)        ;-- copy lat attributes
filevarattdef(fout,"lon", lon)        ;-- copy lon attributes
filevarattdef(fout,"COR_SST_FC", var1_miss) ;-- copy var1_miss
    ;-- attributes to new file
filevarattdef(fout,"COR_SST_FC_SIGN", var2_miss) ;-- copy
    ;-- var2_miss attributes to new file

--;-- explicitly exit file definition mode (not required)
setfileoption(fout,"DefineMode",False)

--;-- output only the data values since the dimensionality and
--;-- such have been predefined. The "(/ /)" syntax tells NCL to only
--;-- output the data values to the predefinedlocations on the file
fout->time      = (/time/)    ;-- write time to new netCDF file
fout->lat       = (/lat/)    ;-- write lat to new netCDF file
fout->lon       = (/lon/)    ;-- write lon to new netCDF file
fout->COR_SST_FC = (/var1_miss/) ;-- write variable to new netCDF file
fout->COR_SST_FC_SIGN = (/var2_miss/) ;-- write variable to new netCDF
--;-- file

;==================================================
;-- open new netcdf file and plot the two variabl
;==================================================
g = addfile(outfile,"r")

v1 = g->COR_SST_FC
v2 = g->COR_SST_FC_SIGN

;==================================================
;-- open workstation
;==================================================

wks_type = "png"                    ;-- plot output type
wks = gsn_open_wks(wks_type,"plot_change_NaNs_to_FillValue")
    ;-- open a workstation

;==================================================
;-- set resources
;==================================================

res = True
res@cnFillOn = True
res@gsnDraw = False
res@gsnFrame = False

;==================================================
;-- create the plot, don't draw yet
;==================================================

plot1 = gsn_csm_contour_map(wks,v1(0,:,:),res)

;==================================================

}
plot2 = gsn_csm_contour_map(wks,v2(0,:,:),res)
;-----------------------------------------------
;-- create the panel plot
;-----------------------------------------------
pres = True
pres@gsnMaximize = True
gsn_panel(wks,(/plot1,plot2/),(/2,1/),pres)
end
14 PyNGL and PyNIO

PyNGL (pronounced ‘pingle’) is a Python module used to visualize scientific data, with an emphasis on high quality 2D visualizations. PyNGL is based on NCL graphics. A working knowledge of Python is assumed.

PyNIO is a Python module that allows read and/or write access to the same variety of data formats using an interface modeled on netCDF.

See also: http://www.pyngl.ucar.edu/

The example scripts in this chapter will show you how to work with the PyNGL/PyNIO modules similar to some NCL examples in this User Guide.

PyNGL and PyNIO can be installed using "conda":

    conda install -c dbrown -c khallock pyngl pynio

14.1 XY-Plot

The first PyNGL example creates an xy-plot.

NUG_xy_plot_simple_PyNGL.py:

```python
import numpy as np
import Ngl, Nio

#-- create x-values
x2 = np.arange(100)

#-- create y-values
data = np.arange(1,40,5)
linear = np.arange(100)
square = [v ** 2 for v in np.arange(0,10,0.1)]

#-- retrieve maximum size of plotting data
maxdim = max(len(data),len(linear),len(square))

#-- create 2D arrays to hold 1D arrays above
y = -999.*np.ones((3,maxdim),'f')

y[0,0:(len(data))] = data
y[1,0:(len(linear))] = linear
y[2,0:(len(square))] = square

#-- open a workstation
wks = Ngl.open_wks("png","plot_xy_simple_ngl")

#-- set resources
res = Ngl.Resources() #-- generate an res object #-- for plot
res.tiMainString = "Title string" #-- set x-axis label
res.tiXAxisString = "x-axis label" #-- set x-axis label
res.tiYAxisString = "y-axis label" #-- set y-axis label

res.vpWidthF = 0.9  #-- viewport width
res.vpHeightF = 0.6  #-- viewport height
```
14.2 Contour Plot – Rectilinear Gridded Data

The first PyNGL example creates a filled contour plot of rectilinear gridded data.

PyNGL _rectilinear_contour.py:

import Ngl,Nio

#-- define variables
diri = "./"       #-- data directory
fname = "rectilinear_grid_2D.nc" #-- data file name

res.caXMissingV = -999.      #-- indicate missing value
res.caYMissingV = -999.      #-- indicate missing value

#-- marker and line settings
res.xyLineColors = ["blue","green","red"] #-- set line colors
res.xyLineThicknessF = 3.0     #-- define line thickness
res.xyDashPatterns = [0,0,2]   #-- ( none, solid, cross )
res.xyMarkLineModes = ["Markers","Lines","Markers"] #-- marker mode
#-- for each line
res.xyMarkers = [16,0,2]      #-- marker type of each line
res.xyMarkerSizeF = 0.01      #-- default is 0.01
res.xyMarkerColors = ["blue","green","red"] #-- set marker colors

#-- legend settings
res.xyExplicitLegendLabels = [" data"," linear"," square"] #-- set explicit
#-- legend labels
res.pmLegendDisplayMode = "Always" #-- turn on the drawing
res.pmLegendOrthogonalPosF = -1.13 #-- move the legend upwards
res.pmLegendParallelPosF = 0.15   #-- move the legend to the
#-- right
res.pmLegendWidthF = 0.2        #-- change width
res.pmLegendHeightF = 0.10      #-- change height
res.lgBoxMinorExtentF = 0.16    #-- legend lines shorter

#-- draw the plot
plot = Ngl.xy(wks,x2,y,res)

#-- the end
Ngl.end()
minval = 250.          #-- minimum contour level
maxval = 315            #-- maximum contour level
inc = 5.                #-- contour level spacing

#-- open file and read variables
f = Nio.open_file(diri + fname,"r")    #-- open data file
temp = f.variables["tsurf"][0,:,:]     #-- first time step
lat = f.variables["lat"][:,]           #-- all latitudes
lon = f.variables["lon"][:,]           #-- all longitudes

tempac,lon = Ngl.add_cyclic(temp,lon)

#-- open a workstation
wks_type = "png"                      #-- graphics output type
wkres = Ngl.Resources()               #-- generate an res object
wkres.wkWidth = 2500                  #-- plot res 2500 pixel width
wkres.wkHeight = 2500                 #-- plot resolution 2500
wks = Ngl.open_wks(wks_type,"plot_contour_ngl",wkres)   #-- open workstation

#-- set resources
res = Ngl.Resources()                 #-- generate an resource

if hasattr(f.variables["tsurf"],"long_name"):
    res.tiMainString = f.variables["tsurf"].long_name #-- set main title

res.cnFillOn = True                   #-- turn on contour fill.
res.cnLinesOn = False                 #-- turn off contour lines
res.cnLineLabelsOn = False            #-- turn off line labels.
res.cnInfoLabelOn = False             #-- turn off info label.
res.cnLevelSelectionMode = "ManualLevels" #-- select manual level
                                        #-- selection mode
res.cnMinLevelValF = minval           #-- minimum contour value
res.cnMaxLevelValF = maxval           #-- maximum contour value
res.cnLevelSpacingF = inc             #-- contour increment
res.cnFillPalette = "rainbow"         #-- choose color map
res.mpGridSpacingF = 30               #-- map grid spacing
res.sfXArray = lon                    #-- longitude locations of data
res.sfYArray = lat                    #-- latitude locations of data
res.lbOrientation = "Horizontal"     #-- labelbar orientation

map = Ngl.contour_map(wks,tempac,res)  #-- draw contours over a map.

#-- end
Ngl.end()
14.3 Vector Plot – Rectilinear Gridded Data

The next example script shows how to create a colored vector plot and a curly vector plot using some well known NCL resources.

NUG_rectilinear_vector_PyNGL.py:

```python
import numpy, os
import Nio
import Ngl

#-- define variables
diri   = "./"                             #-- data directory
fname  = "rectilinear_grid_2D.nc"         #-- data file name

#-- open file and read variables
f    = Nio.open_file(diri + fname,"r")    #-- open data file
temp = f.variables["tsurf"][:,:,]        #-- first time step, reverse lat
u    = f.variables["u10"][:,:,]          #-- first time step, reverse lat
v    = f.variables["v10"][:,:,]          #-- first time step, reverse lat
lat  = f.variables["lat"][:]             #-- reverse latitudes
lon  = f.variables["lon"][:]             #-- all longitudes

nlon   = len(lon)                         #-- number of longitudes
nlat   = len(lat)                         #-- number of latitudes

#-- open a workstation
wkres   = Ngl.Resources()                 #-- generate an resources object
                                           #-- for workstation
wkres.wkWidth   = 2500                    #-- plot res 2500 pixel width
wkres.wkHeight  = 2500                    #-- plot res 2500 pixel height
wks_type      = "png"                     #-- graphics output type
wks           = Ngl.open_wks(wks_type,"rectilinear_vector_PyNGL",wkres)

#-- create 1st plot: vectors on global map
res        = Ngl.Resources()
res.tiMainString = "F25-Wind velocity vectors"  #-- title string
res.tiMainFontHeightF   = 0.024                   #-- decrease title font size
res.mpLimitMode   = "Corners"                    #-- select a sub-region
```
res.mpLeftCornerLonF = float(lon[0])  #-- left longitude value
res.mpRightCornerLonF = float(lon[nlon-1])  #-- right lon value
res.mpLeftCornerLatF = float(lat[0])  #-- left latitude value
res.mpRightCornerLatF = float(lat[nlat-1])  #-- right lat value

res.mpPerimOn = True  #-- turn on map perimeter

res.vcMonoLineArrowColor = False  #-- draw vectors in color
res.vcMinFracLengthF = 0.33  #-- increase length of vectors
res.vcMinMagnitudeF = 0.001  #-- increase length of vectors
res.vcRefLengthF = 0.045  #-- set reference vector length
res.vcRefMagnitudeF = 20.0  #-- set reference magnitude value
res.vcLineArrowThicknessF = 6.0  #-- thicker vector lines (default: 1.0)

res.pmLabelBarDisplayMode = "Always"  #-- turn on a labelbar
res.lbOrientation = "Horizontal"  #-- labelbar orientation
res.lbLabelFontHeightF = 0.008  #-- labelbar label font size
res.lbBoxMinorExtentF = 0.22  #-- decrease height of labelbar boxes

res.vfXArray = lon[::3]  #-- longitude
res.vfYArray = lat[::3]  #-- latitudes

map1 = Ngl.vector_map(wks,u[::3,::3],v[::3,::3],res)  #-- draw a vector plot

#-- create 2nd plot: sub-region colored by temperature variable
tempa = (temp-273.15)*9.0/5.0+32.0  #-- convert from Kelvin to Fahrenheit

res.mpLimitMode = "LatLon"  #-- change the area of the map
res.mpMinLatF = 18.0  #-- minimum latitude
res.mpMaxLatF = 65.0  #-- maximum latitude
res.mpMinLonF = -128.  #-- minimum longitude
res.mpMaxLonF = -58.  #-- minimum longitude

res.mpFillOn = True  #-- turn on map fill
res.mpLandFillColor = "navyblue"  #-- change land color to navy
res.mpOceanFillColor = "transparent"  #-- set ocean/inlandwater color to transparent

res.mpInlandWaterFillColor = "transparent"  #-- set ocean/inlandwater color to transparent
res.mpGridMaskMode = "MaskNotOcean"  #-- draw grid over ocean, not land
res.mpOutlineBoundarySets = "GeophysicalAndUSStates"  #-- outline US States

res.vcFillArrowsOn = True  #-- fill the vector arrows
res.vcMonoFillArrowFillColor = False  #-- draw vectors with colors
res.vcFillArrowEdgeColor = "black"  #-- draw the edges in black
res.vcGlyphStyle = "CurlyVector"  #-- draw nice curly vectors
res.vcLineArrowThicknessF = 5.0  #-- make vector lines thicker (default:1.0)

res.tiMainString = "~F25~Wind velocity vectors"  #-- title string
res.lbTitleString = "TEMPERATURE (~S~o~N~F)"  #-- labelbar title string
res.lbTitleFontHeightF = 0.010  #-- labelbar title font size
res.lbBoxMinorExtentF = 0.18  #-- decrease height of labelbar boxes

res.vfXArray = lon[::3]  #-- longitude
res.vfYArray = lat[::3]  #-- latitudes

map2 = Ngl.vector_scalar_map(wks,u,v,tempa,res)  #-- the end
Ngl.end()
14.4 Slice Plot – Rectilinear Gridded Data

Creating a slice plot at a specified latitude index over all levels can be done in the same way as the index slicing in NCL.

NUG_rectilinear_slice_PyNGL.py:

```python
import numpy as np
import sys,os
import Nio
import Ngl

def nice_lon_labels(lons):
    lonstrs = []
    for l in lons:
        if l < 0:
            lonstrs.append("%i~S~o~N~W" % np.fabs(l))
        elif l > 0:
            lonstrs.append("%i~S~o~N~E" % l)
        else:
            lonstrs.append("EQ" % l)
    return lonstrs

#-- define variables
diri = "./" #-- data directory
fname = "rectilinear_grid_3D.nc" #-- data file name

#-- open file and read variables
f = Nio.open_file(diri + fname,"r") #-- open data file
t = f.variables["t"] #-- get whole "t" variable
t26 = t[0,:,26,:].copy() #-- variable at lat index 26
lev = f.variables["lev"][0.01].copy() #-- all levels, convert to hPa
lat = f.variables["lat"][::-1].copy() #-- reverse latitudes
lon = f.variables["lon"][::-1].copy() #-- all longitudes

t26,lon = Ngl.add_cyclic(t26,lon)
strlat26 = lat[26] #-- retrieve data of lat index 26

#-- get the minimum and maximum of the data
minval = int(np.amin(t[:])) #-- minimum value
```
maxval = int(np.amax(t[:]))  #-- maximum value
inc = 5  #-- contour level spacing

#-- values on which to place tickmarks on X and Y axis
lons = np.arange(-180, 240, 60)
levs = [1000, 700, 500, 400, 300, 200, 150, 100, 70, 50, 30, 10]

#-- open a workstation
wkres = Ngl.Resources()  #-- generate an res object for workstation
wkres.wkWidth = 2500  #-- plot res 2500 pixel width
wkres.wkHeight = 2500  #-- plot res 2500 pixel height
wks_type = "png"  #-- output type
wks = Ngl.open_wks(wks_type, "plot_rectilinear_slice", wkres)

#-- set resources
res = Ngl.Resources
res.tiMainString = "%s (%s) at lat %.2f degrees" % 
(t.long_name, t.units, strlat26)
res.cnLevelSelectionMode = "ManualLevels"  #-- select manual levels
res.cnMinLevelValF = minval  #-- minimum contour value
res.cnMaxLevelValF = maxval  #-- maximum contour value
res.cnLevelSpacingF = inc  #-- contour increment
res.cnFillOn = True  #-- turn on contour fill
res.cnLineLabelsOn = False  #-- turn off line labels
res.cnInfoLabelOn = False  #-- turn off info label
res.cnFillPalette = "BlueWhiteOrangeRed"  #-- set color map
res.pmLabelBarOrthogonalPosF = -0.03  #-- move labelbar close to plot
res.sfXArray = lon  #-- scalar field x
res.sfYArray = lev  #-- scalar field y
res.trYReverse = True  #-- reverse the Y axis
res.nglYAxisType = "LogAxis"  #-- y axis log
res.tiYAxisString = "%s (hPa)" % f.variables["lev"].long_name
res.nglPointTickmarksOutward = True  #-- point tickmarks out
res.tmYLMode = "Explicit"  #-- set y axis tickmark labels
res.tmXBMode = "Explicit"  #-- set x axis tickmark labels
res.tmYValues = levs
res.tmXValues = lons
res.tmYLabels = map(str, levs)
res.tmXLabels = nice_lon_labels(lons)
res.tmYLabelFontHeightF = 0.015  #-- make font smaller
res.tmXLabelFontHeightF = 0.015

map = Ngl.contour(wks, t26, res)  #-- draw contours

#-- end
Ngl.end()}
14.5 Contour Plot – Curvilinear Gridded Data

PyNGL is also able to plot curvilinear gridded data which is shown in the following example.

NUG_curvilinear_contour_PyNGL.py:

```python
import numpy
import sys,os
import Nio
import Ngl

#-- define variables
diri   = "/"       #-- data directory
fname  = "tos_ocean_bipolar_grid.nc"   #-- curvilinear data

#-- open file and read variables
f      = Nio.open_file(diri + fname,"r")
var    = f.variables["tos"][0,:,:]       #-- first time step, reverse lat
lat2d  = f.variables["lat"][::]         #-- 2D latitudes
lon2d  = f.variables["lon"][::]         #-- 2D longitudes

#-- open a workstation
wkres   = Ngl.Resources()      #-- generate an resources object for workstation
wkres.wkWidth         =  2500            #-- width of workstation
wkres.wkHeight        =  2500            #-- height of workstation
wks_type              = "png"            #-- output type
wks =  Ngl.open_wks(wks_type,"Py_curvilinear_contour",wkres)  #-- open workstation

#-- set resources
res     = Ngl.Resources()    #-- generate an resources object for plot
res.cnFillOn          =  True            #-- turn on contour fill
res.cnLinesOn         =  False           #-- don't draw contour lines
res.cnLineLabelsOn    =  False           #-- don't draw line labels
res.cnFillPalette     = "BlueWhiteOrangeRed"       #-- set color map
res.cnFillMode        = "CellFill"         #-- change contour fill mode
res.cnCellFillEdgeColor = "black"          #-- edges color
res.cnCellFillMissingValEdgeColor = "gray50" #-- missing value edges color
res.cnMissingValFillColor = "gray50"       #-- missing value fill color
```
res.lbOrientation = "Horizontal"          #-- labelbar orientation
res.tiMainString  = "Curvilinear grid: MPI-ESM-LR (2D lat/lon arrays)" #-- title string
res.tiMainFontHeightF = 0.022             #-- main title font size
res.sfXArray       = lon2d                #-- longitude grid cell center
res.sfYArray       = lat2d                #-- latitude grid cell center
res.mpFillOn       = False                #-- don't draw filled map
res.mpGridLatSpacingF = 10.              #-- grid lat spacing
res.mpGridLonSpacingF = 10.              #-- grid lon spacing
res.mpDataBaseVersion = "MediumRes"      #-- map database
res.mpLimitMode    = "LatLon"            #-- must be set using minLatF / maxLatF / minLonF / maxLonF
res.mpMinLatF      = -10.                #-- sub-region minimum latitude
res.mpMaxLatF      = 80.                 #-- sub-region maximum latitude
res.mpMinLonF      = -120.               #-- sub-region minimum longitude
res.mpMaxLonF      = 60.                 #-- sub-region maximum longitude

#-- create the plot
plot = Ngl.contour_map(wks,var,res)       #-- create the contour plot

#-- end
Ngl.end()
#-- MAIN
#-- retrieve start time
print ""

#-- define variables
diri = "/"  # data path
fname = "ta_ps_850.nc"  # data file
gname = "grids/r2b4_amip.nc"  # grid info file

#-- open file and read variables
f = Nio.open_file(diri + fname,"r")  # add data file
g = Nio.open_file(diri + gname,"r")  # add grid file

#-- read a timestep of "ta"
var =  f.variables["ta"][0,0,:]  # first time step, lev, ncells
print ""  # output like printVarSummary
print ""

#-- title string
#-- data minimum
#-- data maximum
#-- data increment
#-- set levels array

#-- define the x-, y-values and the polygon points
rad2deg = 45./np.arctan(1.)  # radians to degrees
x =  g.variables["clon"][,:]  # read clon
y =  g.variables["clat"][,:]  # read clat
vlon =  g.variables["clon_vertices"][,:]  # read clon_vertices
vlat =  g.variables["clat_vertices"][,:]  # read clat_vertices
ncells =  vlon.shape[0]  # number of cells
nv =  vlon.shape[1]  # number of edges

#-- cell center, lon
#-- cell center, lat
#-- cell latitude vertices
#-- cell longitude vertices

#-- longitude values -180. - 180.
for j in range(1,ncells):
    for i in range(1,nv):
        if vlon[j,i] < -180. :
        if vlon[j,i] > 180. :

#-- information
print ""
print "Cell points: ", nv
print "Cells: ", str(ncells)
print "Variable ta min/max: %.2f " % np.min(var) + "/" + " %.2f" % np.max(var)
print ""
#--- open a workstation
wks_type = "png"  #-- graphics output type
wks = Ngl.open_wks(wks_type,"plot_contour_unstructured")  #-- open a workstation

#--- set resources
res = Ngl.Resources()  #-- plot mods desired
res.nglDraw = False  #-- turn off plot draw
res.nglFrame = False  #-- don't advance frame
res.cnFillOn = True  #-- color plot desired
res.cnFillMode = "CellFill"  #-- set fill mode
res.cnFillPalette = "BlueWhiteOrangeRed"  #-- set color map
res.cnLinesOn = False  #-- turn off contour lines
res.cnLineLabelsOn = False  #-- turn off contour labels
res.cnLevelSelectionMode = "ExplicitLevels"  #-- use explicit levels
res.cnLevels = levels  #-- set levels
res.lbOrientation = "Horizontal"  #-- vertical by default
res.lbBoxLinesOn = False  #-- turn off labelbar boxes
res.lbLabelFontHeightF = 0.01  #-- labelbar label font size
res.mpFillOn = False  #-- don't use filled map
res.mpGridAndLimbOn = False  #-- don't draw grid lines
res.sfXArray = x  #-- transform x to mesh scalar field
res.sfYArray = y  #-- transform y to mesh scalar field
res.sfXCellBounds = vlon  #-- needed if set "CellFill"
res.sfYCellBounds = vlat  #-- needed if set "CellFill"
res.tiMainString = "ICON grid - CellFill"  #-- title string
res.tiMainOffsetYF = 0.03  #-- move main title towards plot

#--- create the plot
plot = Ngl.contour_map(wks,var,res)

#--- draw the plot and advance the frame
Ngl.draw(plot)
Ngl.frame(wks)

#--- get wallclock time
t2 = time.time()
print "Wallclock time: %0.3f seconds" % (t2-t1)
print ""
Ngl.end()
14.7 Triangles Plot – ICON Data

For some model data we want to see the cells colored by the data values. The next example first creates a contour plot to get the levels and colors which are used to create the second plot drawing triangles using Ngl.add_polygon function.

NUG_unstructured_ICON_triangles_PyNGL.py:

```python
import numpy as np
import math, time, sys, os
import Nio, Ngl

t1 = time.time()  #-- retrieve start time

#-- define variables
diri = './'  
fname = 'ta_ps_850.nc'  #-- data path and file name
gname = 'r2b4_amip.nc'  #-- grid info file
VarName = 'ta'  #-- variable name

#-- open file and read variables
f = Nio.open_file(diri + fname,'r') #-- add data file
g = Nio.open_file(diri + gname,'r') #-- add grid file (not contained in data file!!!)

#-- read a timestep of 'ta'
variable = f.variables['ta']  #-- first time step, lev, ncells
data = variable[0,0,:]  #-- ta [time,lev,ncells]; miss _FillValue
var = data - 273.15  #-- convert to degrees Celsius

#-- define _FillValue and missing_value if not existing
missing = -1e20
if not hasattr(var,'_FillValue'):
    var._FillValue = missing  #-- set _FillValue
if not hasattr(var,'missing_value'):
    var.missing_value = missing  #-- set missing_value

varM = np.ma.array(var, mask=np.equal(var,missing))  #-- mask array with missing values
nummissing = np.count_nonzero(varM.mask)  #-- number of missing values

#-- set data intervals, levels, labels, color indices
varMin, varMax, varInt = -32, 28, 4  #-- set data minimum, maximum, interval
levels = range(varMin,varMax,varInt)  #-- set levels array
nlevs = len(levels)  #-- number of levels
labels = [ '{:.2f}'.format(x) for x in levels ]  #-- convert list of floats to list of strings

#-- print info to stdout
print ''
print 'min/max:    %.2f' %np.min(varM) + ' / ' + ' %.2f' %np.max(varM)
print ''
print 'varMin:    %3d' %varMin
print 'varMax:    %3d' %varMax
print 'varInt:    %3d' %varInt
print ''
print 'missing_value:    ', missing
print 'missing values:   ', nummissing
```

```
rad2deg = 45./np.arctan(1.)  #-- radians to degrees

x, y       =  g.variables['clon'][:], g.variables['clat'][:,
                   g.variables['clon_vertices'][:,
                   g.variables['clat_vertices'][:,

x, y       =  x*rad2deg,  y*rad2deg  #-- cell center, lon, lat
vlon, vlat =  vlat*rad2deg, vlon * rad2deg #-- cell latitude/longitude
                   #-- vertices
ncells, nv =  vlon.shape #-- ncells: number of cells; nv:
                   #-- number of edges

#-- print information to stdout
print ''
print 'cell points:      ', nv
print 'cells:            ', str(ncells)
print ''

#-- rearrange the longitude values to -180.-180.
def rearrange(vlon):
    less_than    = vlon < -180.
    greater_than = vlon >  180.
    vlon[less_than]    = vlon[less_than] + 360.
    return vlon

vlon = rearrange(vlon)  #-- set longitude values to
                   #-- -180. to 180. degrees
print 'min/max vlon:     ', np.min(vlon), np.max(vlon)
print 'min/max vlat:     ', np.min(vlat), np.max(vlat)
print ''

#-- open a workstation for second plot:  triangles plot
wkres = Ngl/Resources()
wkres.wkWidth, wkres.wkHeight  =  2500, 2500
wks_type = 'png'
wks =  Ngl.open_wks(wks_type,'unstructured_ICON_triangles_ngl',wkres)

#-- define colormap
colormap =  Ngl.read_colormap_file('WhiteBlueGreenYellowRed')[22::12,:]
                   #-- RGB ! [256,4] -> [20,4]
                   #-- select every 12th color
colormap[19,:] = [1.,1.,1.,0.,]  #-- white for missing values
print ''
print 'levels:           ',levels
print 'labels:           ',labels
print ''
print 'nlevs:            %3d' %nlevs
print ''

#-- set map resources
mpres                             =  Ngl/Resources()
mpres.nglDraw                      =  False  #-- turn off plot draw and
                   #-- frame advance. We will
mpres.nglFrame                     =  False  #-- do it later after adding
                   #-- subtitles.
mpres.mpGridAndLimbOn              =  False
mpres.mpGeophysicalLineThicknessF  =  2.
mpres.pmTitleDisplayMode           = 'Always'
mpres.tiMainString = 'PyNGL: unstructured grid ICON'

#-- create only a map
map = Ngl.map(wks,mpres)
Ngl.draw(map)

#-- assign and initialize array which will hold the color indices of the cells
gscolors = -1*(np.ones((ncells,),dtype=np.int)) #-- assign array containing zeros; init to transparent: -1

#-- set color index of all cells in between levels
for m in xrange(0,nlevs):
    vind = [] #-- empty list for color indices
    for i in xrange(0,ncells-1):
        if (varM[i] >= levels[m] and varM[i] < levels[m+1]):
            gscolors[i] = m+1 # 1 to nlevs
            vind.append(i)

    print 'finished level %3d' % m , ' -- %5d ' % len(vind) , ' polygons considered - gscolors %3d' % (m+1)
    del vind

gscolors[varM < varMin] = 0 #-- set color index for cells less than level[0]
gscolors[varM >= varMax] = nlevs+1 #-- set color index for cells greater than levels[nlevs-1]
gscolors[np.nonzero(varM.mask)] = -1 #-- set color index for missing locations

#-- set polygon resources
pgres = Ngl.Resources()
pgres.gsEdgesOn = True #-- draw the edges
pgres.gsFillIndex = 0 #-- solid fill
pgres.gsLineColor = 'black' #-- edge line color
pgres.gsLineThicknessF = 0.7 #-- line thickness
pgres.gsColors = colormap[gscolors,:] #-- use color array
pgres.gsSegments = range(0,len(vlon[:,0])*3,3) #-- define segments array for fast draw
lon1d, lat1d = np.ravel(vlon), np.ravel(vlat) #-- convert to 1D-arrays

#-- add polygons to map
polyg = Ngl.add_polygon(wks,map,lon1d,lat1d,pgres)

#-- add a labelbar
lbrres = Ngl.Resources()
lbrres.vpWidthF = 0.85
lbrres.vpHeightF = 0.15
lbrres.lbOrientation = 'Horizontal'
lbrres.lbFillPattern = 'SolidFill'
lbrres.lbMonoFillPattern = 21 #-- must be 21 for color solid fill
lbrres.lbMonoFillColor = False #-- use multiple colors
lbrres.lbFillColors = colormap
lbrres.lbLabelFontHeightF= 0.014
lbrres.lbLabelAlignment = 'InteriorEdges'
lbrres.lbLabelStrings = labels

lb = Ngl.labelbar_ndc(wks,nlevs+1,labels,0.1,0.24,lbrres)

#-- maximize and draw the plot and advance the frame
Ngl.draw(map)
Ngl.frame(wks)

#-- get wallclock time
t2 = time.time()
print ''
print 'Wallclock time: \%0.3f seconds' % (t2-t1)
print ''
Ngl.end()
15 Common Error Messages

The first rule of NCL in avoiding errors and warnings is (as Dennis Shea would say):

**Look at your data!** You can avoid many errors and warnings if you are familiar with your data. Keep that in mind.

To see what's in your file, you can type the following on the shell command line:

```
ncl_filedump <your_filename>
or
ncdump -h <your_filename>
or
cdo infov <your_filename>
```

In comparison to ncdump and ncl_filedump, CDO provides somewhat different information such as the date, time as well as minimum, maximum and mean value of each 2D data field in your file.

Use `printVarSummary` within your NCL script to get the information about variables and dimensions:

```
printVarSummary(tsurf)
```

**Common language error messages:**

<table>
<thead>
<tr>
<th>Error message:</th>
<th>Cause:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fatal: Subscript out of range, error in subscript #0</strong></td>
<td>Subscripting an array using an index that is out-of-bounds. Remember the first element index is 0.</td>
</tr>
<tr>
<td><strong>Error message:</strong></td>
<td><strong>Cause:</strong></td>
</tr>
<tr>
<td>The following error message points to the subscript #1:</td>
<td>Subscripting an array using an index that is out-of-bounds. Subscript numbers start at 0 and go from left to right, so subscript “#1” refers to the second dimension from the left.</td>
</tr>
<tr>
<td><strong>fatal: Subscript out of range, error in subscript #1</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Sample code that causes the error:</strong></td>
<td>Check your subscript indexes to make sure they are in the range of your array size. Use print and printVarSummary.</td>
</tr>
</tbody>
</table>
| ```
  x = random_uniform(-100,100,(/10,20,30/))
  print(x(5,20,5)) ; index '20' is invalid
``` | |
<p>| <strong>Fix:</strong> | |
| Check your subscripts to make sure they are in-range of your array size. Use &quot;print&quot; and &quot;printVarSummary&quot; to get more information about a variable. |</p>
<table>
<thead>
<tr>
<th>Error message:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fatal:Number of subscripts on right-hand-side do not match number of dimensions of variable: (4), Subscripts used: (3)</td>
</tr>
</tbody>
</table>

**Cause:**
Subscripting an array using the wrong dimensionality

**Example:**
```plaintext
x = random_uniform(-50,50,(/5,32,64/))
y = x(0,:,:,:)
```

**Fix:**
Check your subscript syntax.

<table>
<thead>
<tr>
<th>Error message:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fatal:Assignment type mismatch, right hand side can't be coerced to type of left hand side</td>
</tr>
</tbody>
</table>

**Cause:**
Reassigning a variable using a different type or dimensionality

**Example:**
```plaintext
x = 5
x = "Now I'm a string"
```

**Fix:**
Use reassignment operator, or delete variable first.

<table>
<thead>
<tr>
<th>Error message:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fatal:syntax error: possibly an undefined procedure</td>
</tr>
</tbody>
</table>

**Cause:**
Referencing a function or procedure that doesn't exist

**Example:**
```plaintext
i = 5
prnt(i)
```

**Fix:**
Check the spelling of function/procedure and whether you need load another NCL script that defines it.

<table>
<thead>
<tr>
<th>Error message:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fatal:syntax error: function fspan expects 3 arguments, got 2</td>
</tr>
</tbody>
</table>

**Cause:**
Calling a function or procedure with the wrong number of arguments

**Example:**
```plaintext
x = fspan(0,10)
```

**Fix:**
Check your function or procedure arguments. Read the documentation.
| Cause: | You have an unclosed code block, like a "begin" without an end" or an "if" without an "end if"
| Example: | if (x.lt.0) then  
  x = 5  
| Fix: | Check for unclosed code blocks and close them.

| Error message: | fatal:Dimension sizes of left hand side and right hand side of assignment do not match
| Cause: | Assigning a set of array elements to another array with a different number of elements
| Example: | x = (/1,2,3,4/)  
y = (/1,2,3,4,5/)  
x = y  
| Fix: | Check the array sizes on the left and right side of the ".=".

| Error message: | fatal:The result of the conditional expression yields a missing value. NCL can not determine branch, see ismissing function
| Cause: | Using a missing value in an "if" statement or some other conditional statement.
| Example: | x = new(1,float)  
  if(x.eq.5) then  
    print("x is 5")  
  end if  
| Fix: | Use "ismissing" to test for missing values.  
  x = new(1,float)  
  if(.not.ismissing(x).and.x.eq.5) then  
    print("x is 5")  
  end if  

| Error message: | fatal:Variable (x1) is undefined
| Cause: | Referencing a variable that doesn't exist
| Example: |
\begin{verbatim}
x = 5
print(x)

Fix:
Check the spelling.
\end{verbatim}

**Error message:**
*warning:Attempt to reference attribute (FillValue) which is undefined*

**Cause:**
Referencing an attribute variable that doesn't exist

**Example:**
\begin{verbatim}
x = new(1,float)
print(x@FillValue)
\end{verbatim}

**Fix:**
Check the spelling.

**Error message:**
*Argument 0 of the current function or procedure was coerced to the appropriate type and thus will not change if the function or procedure modifies its value*

**Cause:**
Calling a function or procedure with the wrong argument type. This commonly happens if the function is expecting a string and you give it a numerical value.

**Example:**
\begin{verbatim}
x = 12.34
str = str_split(x,"."
\end{verbatim}

**Fix:**
Use one of the toxxx functions to coerce the argument
\begin{verbatim}
x = tostr(12.34)
str = str_split(x,"."
\end{verbatim}

**Common graphics error messages:**

**Error message:**
*gsn_csm_contour_map: Fatal: the input data array must be 1D or 2D
fatal:Illegal right-hand side type for assignment*

**Cause:**
Calling the function gsn_csm_dontour_map to plot variable with the wrong dimension (> 2D).

**Example:**
\begin{verbatim}
plot = gsn_csm_contour_map(wks, tsurf, res)
\end{verbatim}

**Fix:**
*Look at your data! How many dimensions does your variable have? If you don't know it make a printVarSummary to see the number of dimensions of you variable.*
assumed: float tsurf ( time, lev, lat, lon )

You can choose one timestep and one level to use the gsn_csm_contour_map function to create the plot

plot = gsn_csm_contour_map(wks,tsurf(0,0,:,:), res)

Error message:
gsn_add_cyclic: Warning: The range of your longitude data is not 360. You may want to set gsnAddCyclic to False to avoid a warning message from the Spline function.

Cause:
Your longitude value range is less than 360 degrees

Fix:
res@gsnAddCyclic = False

Error message:
is_valid_lat_ycoord: Warning: The units attribute of the Y coordinate array is not set to one of the allowable units values (i.e. 'degrees_north'). Your latitude labels may not be correct.

is_valid_lat_xcoord: Warning: The units attribute of the X coordinate array is not set to one of the allowable units values (i.e. 'degrees_east'). Your longitude labels may not be correct.

Cause:
The dimensions of latitude and longitude don’t have the correct units.

Fix:
Do a printVarSummary of your variable to be plotted or of your lat, lon arrays to see if the units of the dimensions lat and lon are correct. If not add the units attribute to the lat and lon coordinate arrays

lat@units = "degrees_north"
lon@units = "degrees_east"

Error message:
warning:ScalarFieldSetValues: 2d coordinate array sfXArray has an incorrect dimension size: defaulting sfXArray

warning:ScalarFieldSetValues: 2d coordinate array sfYArray has an incorrect dimension size: defaulting sfYArray

Cause:
If you do not have a global grid, the problem is that, by default, the gsn_csm_xxxx routines set the resource gsnAddCyclic to True.

Example:
res@sfXArray = lon ;-- range of lon values < 360
res@sfYArray = lat

Fix:
res@gsnAddCyclic = False

Error message:
warning:tmXBStride is not a valid resource in ex07-1_xy at this time
**Cause:**
The name of a resource is misspelled or it makes no sense to use the resources with the chosen plot function.

**Example 1:**
res@tmXBStride = 2

**Fix 1:**
Take a look at the name of the resource, in this case it is misspelled and it should be tres@tmXBLabelStride = 2

**Example 2:**
res@xyLineColor = "grey"
plot = gsn_csm_contour_map(wks, t, res)

**Fix 2:**
The setting of the resource xyLineColor makes no sense when plotting contours. If you want the contour lines in black use cnLineColor for contours.
res@cnLineColor = "grey"
# List of Example Scripts

The example scripts can be found in `$NCARG_ROOT/lib/ncarg/nclex/nug/`.

<table>
<thead>
<tr>
<th>#</th>
<th>Example</th>
<th>Script File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Read ASCII File 1</td>
<td>NUG_read_ASCII_1.ncl</td>
</tr>
<tr>
<td>4.4</td>
<td>Read ASCII File 2</td>
<td>NUG_read_ASCII_2.ncl</td>
</tr>
<tr>
<td>4.4</td>
<td>Read ASCII File 3</td>
<td>NUG_read_ASCII_3.ncl</td>
</tr>
<tr>
<td>4.5</td>
<td>Read CSV File 1</td>
<td>NUG_readCSV.ncl</td>
</tr>
<tr>
<td>4.5</td>
<td>Read CSV File 2</td>
<td>NUG_read_CSV_2.ncl</td>
</tr>
<tr>
<td>4.6</td>
<td>Read Binary File 1</td>
<td>NUG_read_Binary_1.ncl</td>
</tr>
<tr>
<td>4.6</td>
<td>Read Binary GrADS file</td>
<td>NUG_read_Binary_GrADS.ncl</td>
</tr>
<tr>
<td>4.7</td>
<td>Write ASCII File 1</td>
<td>NUG_write_ASCII_1.ncl</td>
</tr>
<tr>
<td>4.7</td>
<td>Write ASCII File 2</td>
<td>NUG_write_ASCII_2.ncl</td>
</tr>
<tr>
<td>4.7</td>
<td>Write ASCII File 3</td>
<td>NUG_write_ASCII_3.ncl</td>
</tr>
<tr>
<td>4.7</td>
<td>Write ASCII File 4</td>
<td>NUG_write_ASCII_4.ncl</td>
</tr>
<tr>
<td>4.7</td>
<td>Write ASCII File 5</td>
<td>NUG_write_ASCII_5.ncl</td>
</tr>
<tr>
<td>4.9</td>
<td>Write Binary File 1</td>
<td>NUG_write_Binary_1.ncl</td>
</tr>
<tr>
<td>4.9</td>
<td>Write Binary File 2</td>
<td>NUG_write_Binary_2.ncl</td>
</tr>
<tr>
<td>4.10</td>
<td>Write netCDF</td>
<td>NUG_write_NetCDF_1.ncl</td>
</tr>
<tr>
<td>4.10</td>
<td>Write netCDF detailed version</td>
<td>NUG_write_NetCDF_2.ncl</td>
</tr>
<tr>
<td>6.5</td>
<td>Statistics Linear Regression</td>
<td>NUG_statistics_linear_regression.ncl</td>
</tr>
<tr>
<td>6.6</td>
<td>Statistics Running Mean</td>
<td>NUG_statistics_running_mean.ncl</td>
</tr>
<tr>
<td>7.1</td>
<td>Mask Land or Ocean</td>
<td>NUG_example_mask_1.ncl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NUG_example_mask_2.ncl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NUG_example_mask_4.ncl</td>
</tr>
<tr>
<td>8.1</td>
<td>NCL Template Script</td>
<td>NUG_template.ncl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NUG_plot_in_5_steps.ncl</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Map Default</td>
<td>NUG_map_default.ncl</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Map Grids and Tickmarks</td>
<td>NUG_map_grid_and_tickmark_settings.ncl</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Map Settings for Land, Ocean, and Countries</td>
<td>NUG_map_land_ocean_settings.ncl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NUG_map_countries.ncl</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NUG_map_selected_countries.ncl</td>
</tr>
<tr>
<td>8.3.4</td>
<td>Map Projection Mollweide</td>
<td>NUG_projections_mollweide.ncl</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>File Name</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>8.3.5</td>
<td>Regional Map</td>
<td>NUG_map_settings.ncl</td>
</tr>
<tr>
<td>8.4</td>
<td>XY-plot</td>
<td>NUG_xy_plot.ncl</td>
</tr>
<tr>
<td>8.4.2</td>
<td>XY-plot time series</td>
<td>NUG_xy_plot_timeseries.ncl</td>
</tr>
<tr>
<td>8.5</td>
<td>Contour map</td>
<td>NUG_contour_map.ncl</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Contour filled map</td>
<td>NUG_contour_filled_map.ncl</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Contour fill pattern</td>
<td>NUG_contour_fillpattern.ncl</td>
</tr>
<tr>
<td>8.3</td>
<td>Simple map with Mollweide projection</td>
<td>NUG_projections_mollweide.ncl</td>
</tr>
<tr>
<td>8.3.6</td>
<td>Polar plot Northern Hemisphere</td>
<td>NUG_polar_NH.ncl</td>
</tr>
<tr>
<td>8.3.7</td>
<td>Map resolutions</td>
<td>NUG_maps_resolution.ncl</td>
</tr>
<tr>
<td>8.6</td>
<td>Vector using defaults</td>
<td>NUG_vector_default.ncl</td>
</tr>
<tr>
<td>8.6</td>
<td>Vectors curly look</td>
<td>NUG_vector_curly.ncl</td>
</tr>
<tr>
<td>8.6</td>
<td>Vectors colorized</td>
<td>NUG_vector_plot_colorized.ncl</td>
</tr>
<tr>
<td>8.6</td>
<td>Vectors overlay on contour map</td>
<td>NUG_vector_plot_overlay.ncl</td>
</tr>
<tr>
<td>8.7</td>
<td>Slice plot</td>
<td>NUG_slice_plot.ncl</td>
</tr>
<tr>
<td>8.8</td>
<td>Bar Charts</td>
<td>NUG_bar_chart.ncl</td>
</tr>
<tr>
<td>8.8</td>
<td>Bar Charts of multi data sets</td>
<td>NUG_bar_chart_multi.ncl</td>
</tr>
<tr>
<td>8.8</td>
<td>Bar Charts using a reference line</td>
<td>NUG_bar_chart_col_above_below.ncl</td>
</tr>
<tr>
<td>8.8</td>
<td>Histogram</td>
<td>NUG_histograms.ncl</td>
</tr>
<tr>
<td>8.9</td>
<td>Overlays</td>
<td>NUG_transparency_filled_contour.ncl</td>
</tr>
<tr>
<td>8.9</td>
<td>Time series of multi data sets</td>
<td>NUG_multi_timeseries.ncl</td>
</tr>
<tr>
<td>8.9</td>
<td>Grid resolutions comparison</td>
<td>NUG_grid_resolution_comparison.ncl</td>
</tr>
<tr>
<td>8.10</td>
<td>Panel plot</td>
<td>NUG_panel_plot.ncl</td>
</tr>
<tr>
<td>8.10</td>
<td>Panel plot 3 rows and 2 cols</td>
<td>NUG_panel_3x2_plot.ncl</td>
</tr>
<tr>
<td>8.10.1</td>
<td>Panel Plot Settings</td>
<td>NUG_panel_control.ncl</td>
</tr>
<tr>
<td>8.10.1</td>
<td>Panel of Plots with Different Sizes</td>
<td>NUG_panel_vp.ncl</td>
</tr>
<tr>
<td>8.11</td>
<td>Polyline, polygon and polymarker</td>
<td>NUG_polyline_polgons_polymarker.ncl</td>
</tr>
<tr>
<td>8.12</td>
<td>Shapefile plot</td>
<td>NUG_shapefile_plot.ncl</td>
</tr>
<tr>
<td>8.12</td>
<td>Shapefile plot + data</td>
<td>NUG_shapefile_plot_data.ncl</td>
</tr>
<tr>
<td>8.13</td>
<td>Color map definition</td>
<td>NUG_colormaps.ncl</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Example File</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>8.13.1</td>
<td>Convert GrADS coltab to NCL colmap</td>
<td>grads2ncl_coltab.ksh</td>
</tr>
<tr>
<td>8.13.2</td>
<td>Convert GMT coltab to NCL colmap</td>
<td>gmt2ncl_coltab.ksh</td>
</tr>
<tr>
<td>8.14</td>
<td>Curvilinear grid, quick look</td>
<td>NUG_curvilinear_basic.ncl</td>
</tr>
<tr>
<td>8.14</td>
<td>Curvilinear grid</td>
<td>NUG_curvilinear_grid.ncl</td>
</tr>
<tr>
<td>8.14.1</td>
<td>Bipolar grid – MPI-ESM</td>
<td>NUG_bipolar_grid_MPI-ESM.ncl</td>
</tr>
<tr>
<td>8.14.1</td>
<td>Bipolar grid – MPI-ESM – sub-region</td>
<td>NUG_bipolar_grid_MPI-ESM_subregion.ncl</td>
</tr>
<tr>
<td>8.14.2</td>
<td>Tripolar grid STORM</td>
<td>NUG_tripolar_grid_STORM.ncl</td>
</tr>
<tr>
<td>8.15</td>
<td>Unstructured grid</td>
<td>NUG_unstructured_grid.ncl</td>
</tr>
<tr>
<td>8.15.1</td>
<td>Tripolar grid ICON</td>
<td>NUG_tripolar_grid_ICON.ncl</td>
</tr>
<tr>
<td>8.16.1</td>
<td>Rotated grid on native projection</td>
<td>rotatedltln_2.ncl</td>
</tr>
<tr>
<td>8.16.2</td>
<td>Rotated grid transform rotated to unrotated grid</td>
<td>NUG_plot_rotated_grid.ncl</td>
</tr>
<tr>
<td>8.17</td>
<td>Globe with different grid resolutions</td>
<td>NUG_globe_orography_grid_resolution.ncl</td>
</tr>
<tr>
<td>8.18.1</td>
<td>Title string settings</td>
<td>NUG_title_strings.ncl</td>
</tr>
<tr>
<td>8.18.2</td>
<td>Text settings</td>
<td>NUG_text_settings.ncl</td>
</tr>
<tr>
<td>8.18.4</td>
<td>Axis annotations</td>
<td>NUG_axis_annotations.ncl</td>
</tr>
<tr>
<td>8.18.5</td>
<td>Contour labes</td>
<td>NUG_contour_labels.ncl</td>
</tr>
<tr>
<td>8.18.6</td>
<td>Colorize land and ocean</td>
<td>NUG_color_Land_Ocean.ncl</td>
</tr>
<tr>
<td>8.18.7</td>
<td>Colorize Countries</td>
<td>NUG_color_country_user.ncl</td>
</tr>
<tr>
<td>8.18.8</td>
<td>Labelbar settings</td>
<td>NUG_labelbars.ncl</td>
</tr>
<tr>
<td>8.18.9</td>
<td>Legend settings</td>
<td>NUG_legends.ncl</td>
</tr>
<tr>
<td>8.18.11</td>
<td>Date formats</td>
<td>NUG_date_format.ncl</td>
</tr>
<tr>
<td>9.1.1</td>
<td>ESMF regrid curvilinear to rectilinear grid with weights</td>
<td>NUG_regrid_curvilinear_to_rectilinear_bilinear_weights_ESMF.ncl</td>
</tr>
<tr>
<td>9.1.2</td>
<td>ESMF regrid curvilinear to rectilinear grid with weights using given destination grid</td>
<td>NUG_regrid_curvilinear_to_rectilinear_bilinear_wgts_destgrid_ESMF.ncl</td>
</tr>
<tr>
<td>9.1.3</td>
<td>ESMF regrid unstructured to rectilinear grid with weights</td>
<td>NUG_regrid_unstructured_to_rectilinear_bilinear_wgts_ESMF.ncl</td>
</tr>
<tr>
<td>9.1.4</td>
<td>ESMF regrid unstructured to rectilinear grid with weights using given destination grid</td>
<td>NUG_regrid_unstructured_to_rectilinear_bilinear_wgts_destgrid_ESMF.ncl</td>
</tr>
<tr>
<td>9.1.5</td>
<td>ESMF regrid rectilinear to curvilinear grid with weights using given destination grid</td>
<td>NUG_regrid_rectilinear_to_curvilinear_bilinear_wgts_destgrid_ESMF.ncl</td>
</tr>
<tr>
<td>9.1.6</td>
<td>NCL regrid bilinear</td>
<td>NUG_regrid_bilinear_CMIP5_grid_to_1x1deg_grid.ncl</td>
</tr>
<tr>
<td>9.2</td>
<td>CDO remap bilinear</td>
<td>NUG_remap_bilinear_CMIP5_grid_to_1x1deg_grid.ksh</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Code File</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>10.1</td>
<td>Use external Fortran code</td>
<td>NUG_use_Fortran_subroutines.ksh</td>
</tr>
<tr>
<td>10.2</td>
<td>Use external C code</td>
<td>NUG_use_C_subroutines.ksh</td>
</tr>
<tr>
<td>13.2</td>
<td>Convert NaNs to _FillValue</td>
<td>NUG_change_NaNs_to_FillValue.ncl</td>
</tr>
<tr>
<td>14.1</td>
<td>PyNGL: Simple xy-Plot</td>
<td>NUG_xy_plot_simple_PyNGL.py</td>
</tr>
<tr>
<td>14.2</td>
<td>PyNGL: contour plot rectilinear data</td>
<td>NUG_rectilinear_contour_PyNGL.py</td>
</tr>
<tr>
<td>14.3</td>
<td>PyNGL: vector plot rectilinear data</td>
<td>NUG_rectilinear_vector_PyNGL.py</td>
</tr>
<tr>
<td>14.4</td>
<td>PyNGL: slice plot rectilinear data</td>
<td>NUG_rectilinear_slice_PyNGL.py</td>
</tr>
<tr>
<td>14.5</td>
<td>PyNGL: contour plot curvilinear data</td>
<td>NUG_curvilinear_contour_PyNGL.py</td>
</tr>
<tr>
<td>14.6</td>
<td>PyNGL: contour plot unstructured data</td>
<td>NUG_unstructured_contour_cellfill_PyNGL.py</td>
</tr>
<tr>
<td>14.7</td>
<td>PyNGL: polygon plot unstructured data</td>
<td>NUG_unstructured_ICON_triangles_PyNGL.py</td>
</tr>
</tbody>
</table>
17 Appendix A - Plot Types

In this tutorial we have not covered the full variety of different plot types possible with ncl. However, here we give a more comprehensive listing which might inspire you to dig deeper into NCL’s capabilities.

<table>
<thead>
<tr>
<th>Plot Type</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar charts</td>
<td><img src="image1.png" alt="Bar chart" /></td>
</tr>
<tr>
<td>Box plots</td>
<td><img src="image2.png" alt="Box plot" /></td>
</tr>
<tr>
<td>Contours</td>
<td><img src="image3.png" alt="Contour" /></td>
</tr>
<tr>
<td>Latitude vs. Time</td>
<td><img src="image4.png" alt="Latitude vs. Time" /></td>
</tr>
<tr>
<td>Overlay plots</td>
<td><img src="image5.png" alt="Overlay" /></td>
</tr>
<tr>
<td>Plot Type</td>
<td>Image</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Panel plot</td>
<td><img src="image1" alt="Panel plot" /></td>
</tr>
<tr>
<td>Phase plots</td>
<td><img src="image2" alt="Phase plots" /></td>
</tr>
<tr>
<td>Press/height vs. latitude/longitude/time</td>
<td><img src="image3" alt="Press/height vs. latitude/longitude/time" /></td>
</tr>
<tr>
<td>Radar (r,theta) plots</td>
<td><img src="image4" alt="Radar (r,theta) plots" /></td>
</tr>
<tr>
<td>Scatter plots</td>
<td><img src="image5" alt="Scatter plots" /></td>
</tr>
<tr>
<td>Slices</td>
<td><img src="image6" alt="Slices" /></td>
</tr>
<tr>
<td>Streamlines</td>
<td>![Streamlines Image]</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Time vs. longitude/latitude</td>
<td>![Time vs. longitude/latitude Image]</td>
</tr>
<tr>
<td>Trajectories</td>
<td>![Trajectories Image]</td>
</tr>
<tr>
<td>Tropical strip plots</td>
<td>![Tropical strip plots Image]</td>
</tr>
<tr>
<td>Vectors</td>
<td>![Vectors Image]</td>
</tr>
<tr>
<td>WRF-VAPOR (3D) using Vapor</td>
<td>![WRF-VAPOR (3D) using Vapor Image]</td>
</tr>
<tr>
<td>XY plots</td>
<td><img src="image1.png" alt="XY plot example" /></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Evans plots</td>
<td><img src="image2.png" alt="Evans plot example" /></td>
</tr>
<tr>
<td>Histograms</td>
<td><img src="image3.png" alt="Histogram example" /></td>
</tr>
<tr>
<td>Iso levels</td>
<td><img src="image4.png" alt="Iso level example" /></td>
</tr>
<tr>
<td>Meteograms</td>
<td><img src="image5.png" alt="Meteogram example" /></td>
</tr>
<tr>
<td>Pie charts</td>
<td><img src="image6.png" alt="Pie chart example" /></td>
</tr>
<tr>
<td>Plot Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Skew-T</td>
<td></td>
</tr>
<tr>
<td>T-S diagram</td>
<td></td>
</tr>
<tr>
<td>Taylor diagram</td>
<td></td>
</tr>
<tr>
<td>3D plots (TDPACK)</td>
<td></td>
</tr>
<tr>
<td>Wind barbs</td>
<td></td>
</tr>
<tr>
<td>Wind rose</td>
<td></td>
</tr>
</tbody>
</table>

(* plots from the NCL web page - all examples - [http://www.ncl.ucar.edu/Applications/](http://www.ncl.ucar.edu/Applications/))

236
# Appendix B - Projections

<table>
<thead>
<tr>
<th>Projection</th>
<th>Description</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aitoff</td>
<td>Projection: Aitoff</td>
<td><img src="image" alt="Aitoff Diagram" /></td>
</tr>
<tr>
<td>Azimuthal Equidistant</td>
<td>Projection: Azimuthal Equidistant</td>
<td><img src="image" alt="Azimuthal Equidistant Diagram" /></td>
</tr>
<tr>
<td>Cylindrical Equal Area</td>
<td>Projection: Cylindrical Equal Area</td>
<td><img src="image" alt="Cylindrical Equal Area Diagram" /></td>
</tr>
<tr>
<td>Cylindrical Equidistant (default)</td>
<td>Projection: Cylindrical Equidistant (default)</td>
<td><img src="image" alt="Cylindrical Equidistant (default) Diagram" /></td>
</tr>
<tr>
<td>Projection</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>Gnomonic</td>
<td>[Image]</td>
<td></td>
</tr>
<tr>
<td>Hammer</td>
<td>[Image]</td>
<td></td>
</tr>
<tr>
<td>LambertConformal</td>
<td>[Image]</td>
<td></td>
</tr>
<tr>
<td>LambertEqualArea</td>
<td>[Image]</td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Mercator</td>
<td>Projection: Mercator</td>
<td></td>
</tr>
<tr>
<td>Mollweide</td>
<td>Projection: Mollweide</td>
<td></td>
</tr>
<tr>
<td>Orthographic</td>
<td>Projection: Orthographic</td>
<td></td>
</tr>
<tr>
<td>PseudoMollweide</td>
<td>Projection: PseudoMollweide</td>
<td></td>
</tr>
<tr>
<td>Projection</td>
<td>Description</td>
<td>Image</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Robinson</td>
<td></td>
<td><img src="image" alt="Robinson Projection" /></td>
</tr>
<tr>
<td>RotatedMercator</td>
<td></td>
<td><img src="image" alt="RotatedMercator Projection" /></td>
</tr>
<tr>
<td>Satellite</td>
<td></td>
<td><img src="image" alt="Satellite Projection" /></td>
</tr>
<tr>
<td>Stereographic</td>
<td></td>
<td><img src="image" alt="Stereographic Projection" /></td>
</tr>
</tbody>
</table>
To see how to generate all of these map projection plots take a look at the example script NUG_projections.ncl.

To change the map projection, set the "mpProjection" resource, e.g. to use Mollweide projection:

```
res = True
res@mpProjection = "Mollweide"
```
Seventeen dash patterns are available to use with the contour and polyline functions and their resources: cnLineDashPattern, cnLineDashPatterns, xyDashPattern, xyDashPatterns and gsLineDashPattern. You can create your own dash pattern using the NhlNewDashPattern function.

Dash pattern:

| i = 17 | --------------------------------- |
| i = 16 | - - - - - - - - - - - - - - - - |
| i = 15 | --------------------------------- |
| i = 14 | - - - - - - - - - - - - - - - - |
| i = 13 | --------------------------------- |
| i = 12 | - - - - - - - - - - - - - - - - |
| i = 11 | --------------------------------- |
| i = 10 | --------------------------------- |
| i = 9  | --------------------------------- |
| i = 8  | --------------------------------- |
| i = 7  | --------------------------------- |
| i = 6  | --------------------------------- |
| i = 5  | --------------------------------- |
| i = 4  | --------------------------------- |
| i = 3  | --------------------------------- |
| i = 2  | --------------------------------- |
| i = 1  | --------------------------------- |
| i = 0  | --------------------------------- |
20 Appendix D - Fill Pattern Table

Eighteen fill patterns are available to use with the contour and polygon functions and their resources: gsFillIndex, cnFillPattern and cnFillPatterns.

Fill pattern:
21 Appendix E – Marker Table

NCL provides 16 marker styles which can be set with the `gsMarkerIndex` resource. The default is an asterisk. You can also define your own marker style using the `NhlNewMarker` function.

Marker styles:
# Appendix F - Important Built-in Functions and Procedures

NCL provide a huge bunch of built-in functions and procedures which are explained in detail with examples

alphabetically: [http://ncl.ucar.edu/Document/Functions/list_alpha.shtml](http://ncl.ucar.edu/Document/Functions/list_alpha.shtml)

by category: [http://ncl.ucar.edu/Document/Functions/](http://ncl.ucar.edu/Document/Functions/)

The table below contains an overview of the important built-in functions and procedures.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addfile</td>
<td>Opens a data file that is (or is to be) written in a supported file format</td>
</tr>
<tr>
<td>addfiles</td>
<td>Creates a reference that spans multiple data files</td>
</tr>
<tr>
<td>all / any</td>
<td>Returns True if all/any of the values of its input evaluates as True</td>
</tr>
<tr>
<td>cd_calendar</td>
<td>Converts a mixed Julian/Gregorian date to a UT-referenced date</td>
</tr>
<tr>
<td>conform</td>
<td>Conforms an array to the shape of another array</td>
</tr>
<tr>
<td>delete</td>
<td>Deletes variables, attributes, and coordinate variables</td>
</tr>
<tr>
<td>dimsizes</td>
<td>Returns dimension sizes of input variable</td>
</tr>
<tr>
<td>exit</td>
<td>Forces an NCL script to exit immediately</td>
</tr>
<tr>
<td>fileexists</td>
<td>Checks for existence of any UNIX file</td>
</tr>
<tr>
<td>fspan</td>
<td>Creates an array of evenly-spaced floating point numbers</td>
</tr>
<tr>
<td>ind</td>
<td>Returns indices where the input is True</td>
</tr>
<tr>
<td>ismissing</td>
<td>Returns True for every element of the input that contains a missing value</td>
</tr>
<tr>
<td>ispan</td>
<td>Creates an array of equally-spaced integer, long, or int64 values</td>
</tr>
<tr>
<td>new</td>
<td>Creates a NCL variable</td>
</tr>
<tr>
<td>num</td>
<td>Counts the number of True values in input</td>
</tr>
<tr>
<td>print</td>
<td>Prints the value of a variable or expression</td>
</tr>
<tr>
<td>printVarSummary</td>
<td>Prints a summary of a file variable's information</td>
</tr>
<tr>
<td>sprintf</td>
<td>Converts floats or doubles into formatted strings</td>
</tr>
<tr>
<td>sprinti</td>
<td>Converts integers into formatted strings</td>
</tr>
<tr>
<td>system</td>
<td>Executes a shell command</td>
</tr>
<tr>
<td><strong>systemfunc</strong></td>
<td>Executes a shell command and returns output</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td><strong>typeof</strong></td>
<td>Returns type of input variable</td>
</tr>
<tr>
<td><strong>unique_string</strong></td>
<td>Returns a unique string given the input string as a prefix</td>
</tr>
<tr>
<td><strong>where</strong></td>
<td>Performs array assignment based on a conditional array</td>
</tr>
</tbody>
</table>
Appendix G - Important Resources

These are a list of some common resources by topic. They are by no means exhaustive. The term dynamic means that the value is determined by NCL.

For an exhaustive list of resources, see: 
http://www.ncl.ucar.edu/Document/Graphics/Resources/


<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Default</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>trYReverse</td>
<td>Reverse x or y axis</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>trXReverse</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trYMinF</td>
<td>Set minimum of x or y axis</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>trXMinF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trYMaxF</td>
<td>Set maximum of x or y axis</td>
<td>1.0</td>
<td>900</td>
</tr>
<tr>
<td>trXMaxF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trYLog</td>
<td>Turn on/off log axis</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>trXLog</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Default</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnFillOn</td>
<td>Turn on/off filled contours</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>cnFillPalette</td>
<td>Set color map</td>
<td>None</td>
<td>&quot;rainbow&quot;</td>
</tr>
<tr>
<td>cnFillOpacityF</td>
<td>Specifies fill opacity</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>cnMaxOn</td>
<td>Turn on/off contour lines</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>cnFillMode</td>
<td>Set type of contour fill</td>
<td>&quot;AreaFill&quot;</td>
<td>&quot;RasterFill&quot;</td>
</tr>
<tr>
<td>cnLevelSelectionMode</td>
<td>Control contour levels</td>
<td>&quot;AutomaticLevels&quot;</td>
<td>&quot;ExplicitLevels&quot;</td>
</tr>
<tr>
<td>cnMinLevelValF</td>
<td>Set minimum/maximum contour level</td>
<td>Dynamic</td>
<td>5</td>
</tr>
<tr>
<td>cnMaxLevelValF</td>
<td></td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>cnLevelSpacingF</td>
<td>Set contour spacing</td>
<td>Dynamic</td>
<td>2</td>
</tr>
<tr>
<td>cnLineThicknessF</td>
<td>Set thickness of contour lines</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>cnFillPatterns</td>
<td>Set pattern fills</td>
<td>&quot;SolidFill&quot;</td>
<td>(/1,3,-1/) -1 is transparent</td>
</tr>
<tr>
<td>cnInfoLabelOn</td>
<td>Turn on/off the contour info label</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Default</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnFillOn</td>
<td>Turn on/off filled contours</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>cnFillMode</td>
<td>Set type of contour fill</td>
<td>&quot;AreaFill&quot;</td>
<td>&quot;RasterFill&quot;</td>
</tr>
<tr>
<td>cnLabelBarEndStyle</td>
<td>Set style for end labels</td>
<td>&quot;IncludeOuterBoxes&quot;</td>
<td>&quot;ExcludeOuterBoxes&quot;</td>
</tr>
<tr>
<td>lbLabelBarOn</td>
<td>Turn on/off the labelbar</td>
<td>True for gsn_csm interfaces</td>
<td>False</td>
</tr>
<tr>
<td>lbOrientation</td>
<td>Set orientation of labelbar</td>
<td>&quot;horizontal&quot; for gsn_csm interfaces</td>
<td>&quot;vertical&quot;</td>
</tr>
<tr>
<td>lbLabelAutoStride</td>
<td>Automatically pick nice labelbar label stride</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>lbTitleOn</td>
<td>Turn on/off a labelbar title</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>lbTitleString</td>
<td>Set labelbar title</td>
<td>Null</td>
<td>&quot;m/s&quot;</td>
</tr>
<tr>
<td>Name</td>
<td>Function</td>
<td>Default</td>
<td>Example</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>gsnAddCyclic</td>
<td>Turn on/off the addition of a cyclic point to the longitude coordinate values</td>
<td>True for data that has 1D coordinate variables</td>
<td>False</td>
</tr>
<tr>
<td>gsnDraw</td>
<td>Draw the plot</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>gsnFrame</td>
<td>Advance the frame (page)</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>gsnCenterString</td>
<td>Set center string above the plot</td>
<td>N/A</td>
<td>“string here”</td>
</tr>
<tr>
<td>gsnLeftString</td>
<td>Set left string above the plot</td>
<td>long_name (if exists) in gsn_csm interfaces</td>
<td>“Salinity”</td>
</tr>
<tr>
<td>gsnRightString</td>
<td>Set right string above the plot</td>
<td>units (if exists) in gsn_csm interfaces</td>
<td>“ppm”</td>
</tr>
<tr>
<td>gsnMaximize</td>
<td>Maximize the plot and rotate to landscape if necessary</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>gsnSpreadColors</td>
<td>Span full range of color map</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>gsnSpreadColorStart</td>
<td>Begin color map at particular index</td>
<td>2</td>
<td>46</td>
</tr>
<tr>
<td>gsnSpreadColorEnd</td>
<td>End color map at particular index</td>
<td>ncolors-1</td>
<td>89</td>
</tr>
<tr>
<td>gsnScalarContour</td>
<td>Force vector/scalar gsn_csm interfaces to draw vectors over the scalar field</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>gsnXYBarChart</td>
<td>Changes an xy-line into a bar chart</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>gsnXRefLine</td>
<td>Add a vertical reference line to a plot</td>
<td>None</td>
<td>1.0</td>
</tr>
<tr>
<td>gsnXRefLineColor</td>
<td>Change color of X reference line</td>
<td>Foreground color</td>
<td>“green”</td>
</tr>
<tr>
<td>gsnYRefLine</td>
<td>Add a horizontal reference line to a plot</td>
<td>None</td>
<td>0.0</td>
</tr>
<tr>
<td>gsnYRefLineColor</td>
<td>Change color of Y reference line</td>
<td>Foreground color</td>
<td>“blue”</td>
</tr>
<tr>
<td>gsnPanelLabelBar</td>
<td>Turn on/off a common labelbar in a panel plot</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>gsnPanelFigureStrings</td>
<td>Add a series of strings to the upper left corner of each plot in a panel plot</td>
<td>N/A</td>
<td>(“a”, “b”, “c”?)</td>
</tr>
</tbody>
</table>


### Name | Function | Default | Example
--- | --- | --- | ---
`pmLegendWidthF` | Set width of a legend | Dynamic | 0.6
`pmLegendHeightF` | Set height of a legend | Dynamic | 0.3
`lgTitleOn` | Turn on a legend title | False | True
`lgTitleString` | Set title string | N/A | “Profiles”
`lgOrientation` | Set orientation of the legend | “horizontal” | “vertical”
`lgPerimOn` | Turn on/off the legend | True | False
`xyExplicitLegendLabels` | Change default legend labels | N/A | (/"line1","line2","line3"/)
`pmLegendOrthogonalPosF` | Adjust the legend orthogonally | N/A | -0.03
`pmLegendParallelPosF` | Adjust the legend perpendicularly | N/A | 0.2


### Name | Function | Default | Example
--- | --- | --- | ---
`xyDashPatterns` | Set line pattern | Solid | (/0,2/) (/"solid","dash")
`xyLineThicknesses` | Set the line thicknesses | 1.0 | (/2.0,3.0,4.0/)  
`xyLineColors` | Set line color | Foreground color | (/"red","green")
`xyMarkLineModes` | Set whether lines contain markers, lines, or both markers and lines | “Lines” | “Markers” | “MarkLines”
`xyMarkers` | Set marker styles | Asterisk | 5
`xyMarkerColor` | Set marker color | Foreground color | “green”
`xyMarkerSizeF` | Set marker size | 0.01 | 0.03


The second through fifth resources are to be used when zooming in on a cylindrical equidistant or polar stereographic projection. They are the limits set when using `mpLimitMode = “LatLon”`. This resource is set for the user by the high-level plot interfaces. Other projections, such as lambert conformal, require a different limit mode (`mpLimitMode = “Corners”`).

### Name | Function | Default | Example
--- | --- | --- | ---
`mpLimitMode` | Determine how a map is zoomed in | Depends on the projection | “LatLon” “Corners”
`mpMinLatF` | Set the minimum/maximum latitude for map zoom | Dynamic | 30. 60.
`mpMaxLatF` | | |  
`mpMinLonF` | Set the minimum/maximum longitude for map zoom | Dynamic | -70. 89.
`mpMaxLonF` | | |  
`mpFillOn` | Turn on/off map fill | True for gsn_csm | False
`mpCenterLonF` | Set center longitude of projection | 0 | 120.
`mpDataBaseVersion` | Set map database resolution | “LowRes” | “MediumRes” | “HighRes” (must be downloaded extra)
`mpLandFillColor` | Set color of land areas | “gray” for gsn_csm interfaces | “brown”
`mpOceanFillColor` | Set color of ocean areas | “transparent” | “SkyBlue”
`mpInlandWaterFillColor` | Set color of inland water areas | “transparent” | “blue”
`mpOutlineOn` | Turn on/off the map outlines | True | False
<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Default</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpOutlineBoundarySets</td>
<td>Set various continental outlines on and off</td>
<td>“Geophysical”</td>
<td>“GeophysicalAndUSStates”</td>
</tr>
<tr>
<td>mpGeophysicalLineThicknessesF</td>
<td>Set line thickness of map outlines</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>mpGeophysicalLineColor</td>
<td>Set color of map outlines</td>
<td>Foreground color</td>
<td>“red”</td>
</tr>
<tr>
<td>mpUSStateLineColor</td>
<td>Set color of US state boundaries</td>
<td>foreground</td>
<td>“blue”</td>
</tr>
</tbody>
</table>

## Appendix H - Named Colors

<table>
<thead>
<tr>
<th>RGB</th>
<th>Hex</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>250</td>
<td>snow</td>
</tr>
<tr>
<td>248</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>248</td>
<td>245</td>
<td>GhostWhite</td>
</tr>
<tr>
<td>245</td>
<td>245</td>
<td>white smoke</td>
</tr>
<tr>
<td>245</td>
<td>245</td>
<td>WhiteSmoke</td>
</tr>
<tr>
<td>220</td>
<td>220</td>
<td>gainsboro</td>
</tr>
<tr>
<td>255</td>
<td>250</td>
<td>floral white</td>
</tr>
<tr>
<td>255</td>
<td>250</td>
<td>FloralWhite</td>
</tr>
<tr>
<td>253</td>
<td>245</td>
<td>old lace</td>
</tr>
<tr>
<td>253</td>
<td>245</td>
<td>OldLace</td>
</tr>
<tr>
<td>250</td>
<td>240</td>
<td>linen</td>
</tr>
<tr>
<td>250</td>
<td>235</td>
<td>antique white</td>
</tr>
<tr>
<td>250</td>
<td>235</td>
<td>AntiqueWhite</td>
</tr>
<tr>
<td>255</td>
<td>239</td>
<td>papaya whip</td>
</tr>
<tr>
<td>255</td>
<td>239</td>
<td>PapayaWhip</td>
</tr>
<tr>
<td>255</td>
<td>235</td>
<td>blanched almond</td>
</tr>
<tr>
<td>255</td>
<td>235</td>
<td>BlanchedAlmond</td>
</tr>
<tr>
<td>255</td>
<td>228</td>
<td>bisque</td>
</tr>
<tr>
<td>255</td>
<td>218</td>
<td>peach puff</td>
</tr>
<tr>
<td>255</td>
<td>218</td>
<td>PeachPuff</td>
</tr>
<tr>
<td>255</td>
<td>222</td>
<td>navajo white</td>
</tr>
<tr>
<td>255</td>
<td>222</td>
<td>NavajoWhite</td>
</tr>
<tr>
<td>255</td>
<td>228</td>
<td>moccasin</td>
</tr>
<tr>
<td>255</td>
<td>248</td>
<td>cornsilk</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>ivory</td>
</tr>
<tr>
<td>255</td>
<td>250</td>
<td>lemon chiffon</td>
</tr>
<tr>
<td>255</td>
<td>250</td>
<td>LemonChiffon</td>
</tr>
<tr>
<td>255</td>
<td>245</td>
<td>seashell</td>
</tr>
<tr>
<td>240</td>
<td>255</td>
<td>honeydew</td>
</tr>
<tr>
<td>245</td>
<td>255</td>
<td>mint cream</td>
</tr>
<tr>
<td>245</td>
<td>255</td>
<td>MintCream</td>
</tr>
<tr>
<td>240</td>
<td>255</td>
<td>azure</td>
</tr>
<tr>
<td>240</td>
<td>248</td>
<td>alice blue</td>
</tr>
<tr>
<td>240</td>
<td>248</td>
<td>AliceBlue</td>
</tr>
<tr>
<td>230</td>
<td>230</td>
<td>lavender</td>
</tr>
<tr>
<td>255</td>
<td>240</td>
<td>lavender blush</td>
</tr>
<tr>
<td>255</td>
<td>245</td>
<td>LavenderBlush</td>
</tr>
<tr>
<td>255</td>
<td>228</td>
<td>misty rose</td>
</tr>
<tr>
<td>255</td>
<td>228</td>
<td>MistyRose</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>white</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>black</td>
</tr>
<tr>
<td>47</td>
<td>79</td>
<td>dark slate gray</td>
</tr>
<tr>
<td>47</td>
<td>79</td>
<td>DarkSlateGray</td>
</tr>
<tr>
<td>47</td>
<td>79</td>
<td>dark slate grey</td>
</tr>
<tr>
<td>47</td>
<td>79</td>
<td>DarkSlateGrey</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>dim gray</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>DimGray</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>dim grey</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>DimGrey</td>
</tr>
<tr>
<td>112</td>
<td>128</td>
<td>slate gray</td>
</tr>
<tr>
<td>112</td>
<td>128</td>
<td>SlateGray</td>
</tr>
<tr>
<td>119</td>
<td>136</td>
<td>slate grey</td>
</tr>
<tr>
<td>119</td>
<td>136</td>
<td>SlateGrey</td>
</tr>
<tr>
<td>119</td>
<td>136</td>
<td>light slate grey</td>
</tr>
<tr>
<td>119</td>
<td>136</td>
<td>LightSlateGrey</td>
</tr>
<tr>
<td>190</td>
<td>190</td>
<td>gray</td>
</tr>
<tr>
<td>190</td>
<td>190</td>
<td>grey</td>
</tr>
<tr>
<td>211</td>
<td>211</td>
<td>light grey</td>
</tr>
<tr>
<td>211</td>
<td>211</td>
<td>LightGrey</td>
</tr>
<tr>
<td>211</td>
<td>211</td>
<td>light gray</td>
</tr>
<tr>
<td>RGB</td>
<td>Hex</td>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>----------------</td>
</tr>
<tr>
<td>211</td>
<td>112</td>
<td>LightGray</td>
</tr>
<tr>
<td>25</td>
<td>112</td>
<td>midnight blue</td>
</tr>
<tr>
<td>25</td>
<td>112</td>
<td>MidnightBlue</td>
</tr>
<tr>
<td>0</td>
<td>128</td>
<td>navy</td>
</tr>
<tr>
<td>0</td>
<td>128</td>
<td>navy blue</td>
</tr>
<tr>
<td>0</td>
<td>128</td>
<td>NavyBlue</td>
</tr>
<tr>
<td>149</td>
<td>237</td>
<td>cornflower blue</td>
</tr>
<tr>
<td>149</td>
<td>237</td>
<td>CornflowerBlue</td>
</tr>
<tr>
<td>61</td>
<td>139</td>
<td>dark slate blue</td>
</tr>
<tr>
<td>61</td>
<td>139</td>
<td>DarkSlateBlue</td>
</tr>
<tr>
<td>90</td>
<td>205</td>
<td>slate blue</td>
</tr>
<tr>
<td>90</td>
<td>205</td>
<td>SlateBlue</td>
</tr>
<tr>
<td>104</td>
<td>238</td>
<td>medium slate blue</td>
</tr>
<tr>
<td>104</td>
<td>238</td>
<td>MediumSlateBlue</td>
</tr>
<tr>
<td>112</td>
<td>255</td>
<td>light slate blue</td>
</tr>
<tr>
<td>112</td>
<td>255</td>
<td>LightSlateBlue</td>
</tr>
<tr>
<td>0</td>
<td>205</td>
<td>medium blue</td>
</tr>
<tr>
<td>0</td>
<td>205</td>
<td>MediumBlue</td>
</tr>
<tr>
<td>105</td>
<td>225</td>
<td>royal blue</td>
</tr>
<tr>
<td>105</td>
<td>225</td>
<td>RoyalBlue</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>blue</td>
</tr>
<tr>
<td>144</td>
<td>255</td>
<td>dodger blue</td>
</tr>
<tr>
<td>144</td>
<td>255</td>
<td>DodgerBlue</td>
</tr>
<tr>
<td>191</td>
<td>255</td>
<td>deep sky blue</td>
</tr>
<tr>
<td>191</td>
<td>255</td>
<td>DeepSkyBlue</td>
</tr>
<tr>
<td>206</td>
<td>235</td>
<td>sky blue</td>
</tr>
<tr>
<td>206</td>
<td>235</td>
<td>SkyBlue</td>
</tr>
<tr>
<td>206</td>
<td>250</td>
<td>light sky blue</td>
</tr>
<tr>
<td>206</td>
<td>250</td>
<td>LightSkyBlue</td>
</tr>
<tr>
<td>130</td>
<td>180</td>
<td>steel blue</td>
</tr>
<tr>
<td>130</td>
<td>180</td>
<td>SteelBlue</td>
</tr>
<tr>
<td>196</td>
<td>222</td>
<td>light steel blue</td>
</tr>
<tr>
<td>196</td>
<td>222</td>
<td>LightSteelBlue</td>
</tr>
<tr>
<td>216</td>
<td>230</td>
<td>light blue</td>
</tr>
<tr>
<td>216</td>
<td>230</td>
<td>LightBlue</td>
</tr>
<tr>
<td>224</td>
<td>230</td>
<td>powder blue</td>
</tr>
<tr>
<td>224</td>
<td>230</td>
<td>PowderBlue</td>
</tr>
<tr>
<td>238</td>
<td>238</td>
<td>pale turquoise</td>
</tr>
<tr>
<td>238</td>
<td>238</td>
<td>PaleTurquoise</td>
</tr>
<tr>
<td>206</td>
<td>209</td>
<td>dark turquoise</td>
</tr>
<tr>
<td>206</td>
<td>209</td>
<td>DarkTurquoise</td>
</tr>
<tr>
<td>209</td>
<td>204</td>
<td>medium turquoise</td>
</tr>
<tr>
<td>209</td>
<td>204</td>
<td>MediumTurquoise</td>
</tr>
<tr>
<td>224</td>
<td>208</td>
<td>turquoise</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>cyan</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>light cyan</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>LightCyan</td>
</tr>
<tr>
<td>158</td>
<td>160</td>
<td>cadet blue</td>
</tr>
<tr>
<td>158</td>
<td>160</td>
<td>CadetBlue</td>
</tr>
<tr>
<td>205</td>
<td>170</td>
<td>medium aquamarine</td>
</tr>
<tr>
<td>205</td>
<td>170</td>
<td>MediumAquamarine</td>
</tr>
<tr>
<td>255</td>
<td>212</td>
<td>aquamarine</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>dark green</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>DarkGreen</td>
</tr>
<tr>
<td>107</td>
<td>47</td>
<td>dark olive green</td>
</tr>
<tr>
<td>107</td>
<td>47</td>
<td>DarkOliveGreen</td>
</tr>
<tr>
<td>188</td>
<td>143</td>
<td>dark sea green</td>
</tr>
<tr>
<td>188</td>
<td>143</td>
<td>DarkSeaGreen</td>
</tr>
<tr>
<td>139</td>
<td>87</td>
<td>sea green</td>
</tr>
<tr>
<td>139</td>
<td>87</td>
<td>SeaGreen</td>
</tr>
<tr>
<td>179</td>
<td>113</td>
<td>medium sea green</td>
</tr>
<tr>
<td>179</td>
<td>113</td>
<td>MediumSeaGreen</td>
</tr>
<tr>
<td>178</td>
<td>170</td>
<td>light sea green</td>
</tr>
<tr>
<td>178</td>
<td>170</td>
<td>LightSeaGreen</td>
</tr>
<tr>
<td>251</td>
<td>152</td>
<td>pale green</td>
</tr>
<tr>
<td>RGB</td>
<td>Hex</td>
<td>Name</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>spring green</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>PaleGreen</td>
</tr>
<tr>
<td>124</td>
<td>252</td>
<td>lawn green</td>
</tr>
<tr>
<td>124</td>
<td>252</td>
<td>LawnGreen</td>
</tr>
<tr>
<td>127</td>
<td>255</td>
<td>green</td>
</tr>
<tr>
<td>127</td>
<td>255</td>
<td>chartreuse</td>
</tr>
<tr>
<td>173</td>
<td>255</td>
<td>medium spring green</td>
</tr>
<tr>
<td>173</td>
<td>255</td>
<td>MediumSpringGreen</td>
</tr>
<tr>
<td>50</td>
<td>205</td>
<td>lime green</td>
</tr>
<tr>
<td>50</td>
<td>205</td>
<td>LimeGreen</td>
</tr>
<tr>
<td>154</td>
<td>205</td>
<td>yellow green</td>
</tr>
<tr>
<td>154</td>
<td>205</td>
<td>YellowGreen</td>
</tr>
<tr>
<td>34</td>
<td>139</td>
<td>forest green</td>
</tr>
<tr>
<td>34</td>
<td>139</td>
<td>ForestGreen</td>
</tr>
<tr>
<td>107</td>
<td>142</td>
<td>olive drab</td>
</tr>
<tr>
<td>107</td>
<td>142</td>
<td>OliveDrab</td>
</tr>
<tr>
<td>189</td>
<td>183</td>
<td>dark khaki</td>
</tr>
<tr>
<td>189</td>
<td>183</td>
<td>DarkKhaki</td>
</tr>
<tr>
<td>240</td>
<td>230</td>
<td>khaki</td>
</tr>
<tr>
<td>238</td>
<td>232</td>
<td>pale goldenrod</td>
</tr>
<tr>
<td>238</td>
<td>232</td>
<td>PaleGoldenrod</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>light goldenrod yellow</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>LightGoldenrodYellow</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>light yellow</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>LightYellow</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>yellow</td>
</tr>
<tr>
<td>255</td>
<td>215</td>
<td>gold</td>
</tr>
<tr>
<td>238</td>
<td>221</td>
<td>light goldenrod</td>
</tr>
<tr>
<td>238</td>
<td>221</td>
<td>LightGoldenrod</td>
</tr>
<tr>
<td>218</td>
<td>165</td>
<td>goldenrod</td>
</tr>
<tr>
<td>184</td>
<td>134</td>
<td>dark goldenrod</td>
</tr>
<tr>
<td>184</td>
<td>134</td>
<td>DarkGoldenrod</td>
</tr>
<tr>
<td>188</td>
<td>143</td>
<td>rosy brown</td>
</tr>
<tr>
<td>188</td>
<td>143</td>
<td>RosyBrown</td>
</tr>
<tr>
<td>205</td>
<td>92</td>
<td>Indian red</td>
</tr>
<tr>
<td>205</td>
<td>92</td>
<td>IndianRed</td>
</tr>
<tr>
<td>139</td>
<td>69</td>
<td>saddle brown</td>
</tr>
<tr>
<td>139</td>
<td>69</td>
<td>SaddleBrown</td>
</tr>
<tr>
<td>160</td>
<td>82</td>
<td>sienna</td>
</tr>
<tr>
<td>205</td>
<td>133</td>
<td>Peru</td>
</tr>
<tr>
<td>222</td>
<td>184</td>
<td>burlywood</td>
</tr>
<tr>
<td>245</td>
<td>245</td>
<td>beige</td>
</tr>
<tr>
<td>245</td>
<td>222</td>
<td>wheat</td>
</tr>
<tr>
<td>244</td>
<td>164</td>
<td>sandy brown</td>
</tr>
<tr>
<td>244</td>
<td>164</td>
<td>SandyBrown</td>
</tr>
<tr>
<td>210</td>
<td>180</td>
<td>tan</td>
</tr>
<tr>
<td>210</td>
<td>105</td>
<td>sandy</td>
</tr>
<tr>
<td>178</td>
<td>34</td>
<td>chocolate</td>
</tr>
<tr>
<td>165</td>
<td>42</td>
<td>firebrick</td>
</tr>
<tr>
<td>233</td>
<td>150</td>
<td>brown</td>
</tr>
<tr>
<td>233</td>
<td>150</td>
<td>DarkSalmon</td>
</tr>
<tr>
<td>250</td>
<td>128</td>
<td>salmon</td>
</tr>
<tr>
<td>255</td>
<td>160</td>
<td>light salmon</td>
</tr>
<tr>
<td>255</td>
<td>160</td>
<td>LightSalmon</td>
</tr>
<tr>
<td>255</td>
<td>165</td>
<td>orange</td>
</tr>
<tr>
<td>255</td>
<td>140</td>
<td>dark orange</td>
</tr>
<tr>
<td>255</td>
<td>140</td>
<td>DarkOrange</td>
</tr>
<tr>
<td>255</td>
<td>127</td>
<td>coral</td>
</tr>
<tr>
<td>240</td>
<td>128</td>
<td>light coral</td>
</tr>
<tr>
<td>240</td>
<td>128</td>
<td>LightCoral</td>
</tr>
<tr>
<td>255</td>
<td>99</td>
<td>tomato</td>
</tr>
<tr>
<td>255</td>
<td>69</td>
<td>orange red</td>
</tr>
<tr>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>255</td>
<td>69</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>255</td>
<td>105</td>
<td>180</td>
</tr>
<tr>
<td>255</td>
<td>105</td>
<td>180</td>
</tr>
<tr>
<td>255</td>
<td>20</td>
<td>147</td>
</tr>
<tr>
<td>255</td>
<td>20</td>
<td>147</td>
</tr>
<tr>
<td>255</td>
<td>192</td>
<td>203</td>
</tr>
<tr>
<td>255</td>
<td>182</td>
<td>193</td>
</tr>
<tr>
<td>219</td>
<td>112</td>
<td>147</td>
</tr>
<tr>
<td>219</td>
<td>112</td>
<td>147</td>
</tr>
<tr>
<td>176</td>
<td>48</td>
<td>96</td>
</tr>
<tr>
<td>199</td>
<td>21</td>
<td>133</td>
</tr>
<tr>
<td>199</td>
<td>21</td>
<td>133</td>
</tr>
<tr>
<td>208</td>
<td>32</td>
<td>144</td>
</tr>
<tr>
<td>208</td>
<td>32</td>
<td>144</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>238</td>
<td>130</td>
<td>238</td>
</tr>
<tr>
<td>221</td>
<td>160</td>
<td>221</td>
</tr>
<tr>
<td>218</td>
<td>112</td>
<td>214</td>
</tr>
<tr>
<td>186</td>
<td>85</td>
<td>211</td>
</tr>
<tr>
<td>186</td>
<td>85</td>
<td>211</td>
</tr>
<tr>
<td>153</td>
<td>50</td>
<td>204</td>
</tr>
<tr>
<td>153</td>
<td>50</td>
<td>204</td>
</tr>
<tr>
<td>148</td>
<td>0</td>
<td>211</td>
</tr>
<tr>
<td>148</td>
<td>0</td>
<td>211</td>
</tr>
<tr>
<td>138</td>
<td>43</td>
<td>226</td>
</tr>
<tr>
<td>138</td>
<td>43</td>
<td>226</td>
</tr>
<tr>
<td>160</td>
<td>32</td>
<td>240</td>
</tr>
<tr>
<td>147</td>
<td>112</td>
<td>219</td>
</tr>
<tr>
<td>147</td>
<td>112</td>
<td>219</td>
</tr>
<tr>
<td>216</td>
<td>191</td>
<td>216</td>
</tr>
<tr>
<td>255</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>238</td>
<td>233</td>
<td>233</td>
</tr>
<tr>
<td>205</td>
<td>201</td>
<td>201</td>
</tr>
<tr>
<td>139</td>
<td>137</td>
<td>137</td>
</tr>
<tr>
<td>255</td>
<td>245</td>
<td>238</td>
</tr>
<tr>
<td>238</td>
<td>229</td>
<td>222</td>
</tr>
<tr>
<td>205</td>
<td>197</td>
<td>191</td>
</tr>
<tr>
<td>139</td>
<td>134</td>
<td>130</td>
</tr>
<tr>
<td>255</td>
<td>239</td>
<td>219</td>
</tr>
<tr>
<td>238</td>
<td>223</td>
<td>204</td>
</tr>
<tr>
<td>205</td>
<td>192</td>
<td>176</td>
</tr>
<tr>
<td>139</td>
<td>131</td>
<td>120</td>
</tr>
<tr>
<td>255</td>
<td>228</td>
<td>196</td>
</tr>
<tr>
<td>238</td>
<td>213</td>
<td>183</td>
</tr>
<tr>
<td>205</td>
<td>183</td>
<td>158</td>
</tr>
<tr>
<td>139</td>
<td>125</td>
<td>107</td>
</tr>
<tr>
<td>255</td>
<td>218</td>
<td>185</td>
</tr>
<tr>
<td>238</td>
<td>203</td>
<td>173</td>
</tr>
<tr>
<td>205</td>
<td>175</td>
<td>149</td>
</tr>
<tr>
<td>139</td>
<td>119</td>
<td>101</td>
</tr>
<tr>
<td>255</td>
<td>222</td>
<td>173</td>
</tr>
<tr>
<td>238</td>
<td>207</td>
<td>161</td>
</tr>
<tr>
<td>205</td>
<td>179</td>
<td>139</td>
</tr>
<tr>
<td>139</td>
<td>121</td>
<td>94</td>
</tr>
<tr>
<td>255</td>
<td>250</td>
<td>205</td>
</tr>
<tr>
<td>238</td>
<td>233</td>
<td>191</td>
</tr>
<tr>
<td>205</td>
<td>201</td>
<td>165</td>
</tr>
<tr>
<td>139</td>
<td>137</td>
<td>112</td>
</tr>
<tr>
<td>255</td>
<td>248</td>
<td>220</td>
</tr>
<tr>
<td>238</td>
<td>232</td>
<td>205</td>
</tr>
<tr>
<td>205</td>
<td>200</td>
<td>177</td>
</tr>
<tr>
<td>139</td>
<td>136</td>
<td>120</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>240</td>
</tr>
<tr>
<td>Color 1</td>
<td>Color 2</td>
<td>Color 3</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>ivory2</td>
<td>ivory3</td>
<td>ivory4</td>
</tr>
<tr>
<td>honeydew1</td>
<td>honeydew2</td>
<td>honeydew3</td>
</tr>
<tr>
<td>LavenderBlush1</td>
<td>LavenderBlush2</td>
<td>LavenderBlush3</td>
</tr>
<tr>
<td>MistyRose1</td>
<td>MistyRose2</td>
<td>MistyRose3</td>
</tr>
<tr>
<td>azure1</td>
<td>azure2</td>
<td>azure3</td>
</tr>
<tr>
<td>SlateBlue1</td>
<td>SlateBlue2</td>
<td>SlateBlue3</td>
</tr>
<tr>
<td>RoyalBlue1</td>
<td>RoyalBlue2</td>
<td>RoyalBlue3</td>
</tr>
<tr>
<td>blue1</td>
<td>blue2</td>
<td>blue3</td>
</tr>
<tr>
<td>DodgerBlue1</td>
<td>DodgerBlue2</td>
<td>DodgerBlue3</td>
</tr>
<tr>
<td>SteelBlue1</td>
<td>SteelBlue2</td>
<td>SteelBlue3</td>
</tr>
<tr>
<td>SkyBlue1</td>
<td>SkyBlue2</td>
<td>SkyBlue3</td>
</tr>
<tr>
<td>LightSkyBlue1</td>
<td>LightSkyBlue2</td>
<td>LightSkyBlue3</td>
</tr>
<tr>
<td>SlateGray1</td>
<td>SlateGray2</td>
<td>SlateGray3</td>
</tr>
<tr>
<td>LightSteelBlue1</td>
<td>LightSteelBlue2</td>
<td>LightSteelBlue3</td>
</tr>
<tr>
<td>LightBlue1</td>
<td>LightBlue2</td>
<td>LightBlue3</td>
</tr>
<tr>
<td>LightCyan1</td>
<td>LightCyan2</td>
<td>LightCyan3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>180</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>122</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>187</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>174</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>150</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>102</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>152</td>
<td>245</td>
<td>255</td>
</tr>
<tr>
<td>142</td>
<td>229</td>
<td>238</td>
</tr>
<tr>
<td>122</td>
<td>197</td>
<td>205</td>
</tr>
<tr>
<td>83</td>
<td>134</td>
<td>139</td>
</tr>
<tr>
<td>0</td>
<td>245</td>
<td>255</td>
</tr>
<tr>
<td>0</td>
<td>229</td>
<td>238</td>
</tr>
<tr>
<td>0</td>
<td>197</td>
<td>205</td>
</tr>
<tr>
<td>0</td>
<td>134</td>
<td>139</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>0</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>0</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>0</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>151</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>141</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>121</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>82</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>127</td>
<td>255</td>
<td>212</td>
</tr>
<tr>
<td>118</td>
<td>238</td>
<td>198</td>
</tr>
<tr>
<td>102</td>
<td>205</td>
<td>170</td>
</tr>
<tr>
<td>69</td>
<td>139</td>
<td>116</td>
</tr>
<tr>
<td>193</td>
<td>255</td>
<td>193</td>
</tr>
<tr>
<td>180</td>
<td>238</td>
<td>180</td>
</tr>
<tr>
<td>155</td>
<td>205</td>
<td>155</td>
</tr>
<tr>
<td>105</td>
<td>139</td>
<td>105</td>
</tr>
<tr>
<td>84</td>
<td>255</td>
<td>159</td>
</tr>
<tr>
<td>78</td>
<td>238</td>
<td>148</td>
</tr>
<tr>
<td>67</td>
<td>205</td>
<td>128</td>
</tr>
<tr>
<td>46</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>154</td>
<td>255</td>
<td>154</td>
</tr>
<tr>
<td>144</td>
<td>238</td>
<td>144</td>
</tr>
<tr>
<td>124</td>
<td>205</td>
<td>124</td>
</tr>
<tr>
<td>84</td>
<td>139</td>
<td>84</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>127</td>
</tr>
<tr>
<td>0</td>
<td>238</td>
<td>118</td>
</tr>
<tr>
<td>0</td>
<td>205</td>
<td>102</td>
</tr>
<tr>
<td>0</td>
<td>139</td>
<td>69</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>238</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>205</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>127</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>118</td>
<td>238</td>
<td>0</td>
</tr>
<tr>
<td>102</td>
<td>205</td>
<td>0</td>
</tr>
<tr>
<td>69</td>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>192</td>
<td>255</td>
<td>62</td>
</tr>
<tr>
<td>179</td>
<td>238</td>
<td>58</td>
</tr>
<tr>
<td>154</td>
<td>205</td>
<td>50</td>
</tr>
<tr>
<td>105</td>
<td>139</td>
<td>34</td>
</tr>
<tr>
<td>202</td>
<td>255</td>
<td>112</td>
</tr>
<tr>
<td>188</td>
<td>238</td>
<td>104</td>
</tr>
<tr>
<td>162</td>
<td>205</td>
<td>90</td>
</tr>
<tr>
<td>110</td>
<td>139</td>
<td>61</td>
</tr>
<tr>
<td>255</td>
<td>246</td>
<td>143</td>
</tr>
<tr>
<td>238</td>
<td>230</td>
<td>133</td>
</tr>
<tr>
<td>205</td>
<td>198</td>
<td>115</td>
</tr>
<tr>
<td>139</td>
<td>134</td>
<td>78</td>
</tr>
<tr>
<td>255</td>
<td>236</td>
<td>139</td>
</tr>
<tr>
<td>238</td>
<td>220</td>
<td>130</td>
</tr>
<tr>
<td>205</td>
<td>190</td>
<td>112</td>
</tr>
<tr>
<td>Color Name</td>
<td>Value1</td>
<td>Value2</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>LightGoldenrod4</td>
<td>139</td>
<td>129</td>
</tr>
<tr>
<td>LightYellow1</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>LightYellow2</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>LightYellow3</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>LightYellow4</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>yellow1</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>yellow2</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>yellow3</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>yellow4</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>gold1</td>
<td>255</td>
<td>215</td>
</tr>
<tr>
<td>gold2</td>
<td>238</td>
<td>201</td>
</tr>
<tr>
<td>gold3</td>
<td>205</td>
<td>173</td>
</tr>
<tr>
<td>gold4</td>
<td>139</td>
<td>117</td>
</tr>
<tr>
<td>goldenrod1</td>
<td>255</td>
<td>193</td>
</tr>
<tr>
<td>goldenrod2</td>
<td>238</td>
<td>180</td>
</tr>
<tr>
<td>goldenrod3</td>
<td>205</td>
<td>155</td>
</tr>
<tr>
<td>goldenrod4</td>
<td>139</td>
<td>105</td>
</tr>
<tr>
<td>DarkGoldenrod1</td>
<td>255</td>
<td>185</td>
</tr>
<tr>
<td>DarkGoldenrod2</td>
<td>238</td>
<td>173</td>
</tr>
<tr>
<td>DarkGoldenrod3</td>
<td>205</td>
<td>149</td>
</tr>
<tr>
<td>DarkGoldenrod4</td>
<td>139</td>
<td>101</td>
</tr>
<tr>
<td>RosyBrown1</td>
<td>255</td>
<td>193</td>
</tr>
<tr>
<td>RosyBrown2</td>
<td>238</td>
<td>180</td>
</tr>
<tr>
<td>RosyBrown3</td>
<td>205</td>
<td>155</td>
</tr>
<tr>
<td>RosyBrown4</td>
<td>139</td>
<td>105</td>
</tr>
<tr>
<td>IndianRed1</td>
<td>255</td>
<td>106</td>
</tr>
<tr>
<td>IndianRed2</td>
<td>238</td>
<td>99</td>
</tr>
<tr>
<td>IndianRed3</td>
<td>205</td>
<td>85</td>
</tr>
<tr>
<td>IndianRed4</td>
<td>139</td>
<td>58</td>
</tr>
<tr>
<td>sienna1</td>
<td>255</td>
<td>130</td>
</tr>
<tr>
<td>sienna2</td>
<td>238</td>
<td>121</td>
</tr>
<tr>
<td>sienna3</td>
<td>205</td>
<td>104</td>
</tr>
<tr>
<td>sienna4</td>
<td>139</td>
<td>71</td>
</tr>
<tr>
<td>burlywood1</td>
<td>255</td>
<td>211</td>
</tr>
<tr>
<td>burlywood2</td>
<td>238</td>
<td>197</td>
</tr>
<tr>
<td>burlywood3</td>
<td>205</td>
<td>170</td>
</tr>
<tr>
<td>burlywood4</td>
<td>139</td>
<td>115</td>
</tr>
<tr>
<td>wheat1</td>
<td>255</td>
<td>231</td>
</tr>
<tr>
<td>wheat2</td>
<td>238</td>
<td>216</td>
</tr>
<tr>
<td>wheat3</td>
<td>205</td>
<td>186</td>
</tr>
<tr>
<td>wheat4</td>
<td>139</td>
<td>128</td>
</tr>
<tr>
<td>tan1</td>
<td>255</td>
<td>165</td>
</tr>
<tr>
<td>tan2</td>
<td>238</td>
<td>154</td>
</tr>
<tr>
<td>tan3</td>
<td>205</td>
<td>133</td>
</tr>
<tr>
<td>tan4</td>
<td>139</td>
<td>90</td>
</tr>
<tr>
<td>chocolate1</td>
<td>255</td>
<td>127</td>
</tr>
<tr>
<td>chocolate2</td>
<td>238</td>
<td>118</td>
</tr>
<tr>
<td>chocolate3</td>
<td>205</td>
<td>102</td>
</tr>
<tr>
<td>chocolate4</td>
<td>139</td>
<td>69</td>
</tr>
<tr>
<td>firebrick1</td>
<td>255</td>
<td>48</td>
</tr>
<tr>
<td>firebrick2</td>
<td>238</td>
<td>44</td>
</tr>
<tr>
<td>firebrick3</td>
<td>205</td>
<td>38</td>
</tr>
<tr>
<td>firebrick4</td>
<td>139</td>
<td>26</td>
</tr>
<tr>
<td>brown1</td>
<td>255</td>
<td>64</td>
</tr>
<tr>
<td>brown2</td>
<td>238</td>
<td>59</td>
</tr>
<tr>
<td>brown3</td>
<td>205</td>
<td>51</td>
</tr>
<tr>
<td>brown4</td>
<td>139</td>
<td>35</td>
</tr>
<tr>
<td>salmon1</td>
<td>255</td>
<td>140</td>
</tr>
<tr>
<td>salmon2</td>
<td>238</td>
<td>130</td>
</tr>
<tr>
<td>salmon3</td>
<td>205</td>
<td>112</td>
</tr>
<tr>
<td>salmon4</td>
<td>139</td>
<td>76</td>
</tr>
<tr>
<td>LightSalmon1</td>
<td>255</td>
<td>160</td>
</tr>
<tr>
<td>LightSalmon2</td>
<td>238</td>
<td>149</td>
</tr>
<tr>
<td>LightSalmon3</td>
<td>205</td>
<td>129</td>
</tr>
<tr>
<td>LightSalmon4</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>RGB</td>
<td>Hex</td>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>---------------</td>
</tr>
<tr>
<td>255</td>
<td>165</td>
<td>orange1</td>
</tr>
<tr>
<td>238</td>
<td>154</td>
<td>orange2</td>
</tr>
<tr>
<td>205</td>
<td>133</td>
<td>orange3</td>
</tr>
<tr>
<td>139</td>
<td>90</td>
<td>orange4</td>
</tr>
<tr>
<td>255</td>
<td>127</td>
<td>DarkOrange1</td>
</tr>
<tr>
<td>238</td>
<td>118</td>
<td>DarkOrange2</td>
</tr>
<tr>
<td>205</td>
<td>102</td>
<td>DarkOrange3</td>
</tr>
<tr>
<td>139</td>
<td>69</td>
<td>DarkOrange4</td>
</tr>
<tr>
<td>255</td>
<td>114</td>
<td>coral1</td>
</tr>
<tr>
<td>238</td>
<td>106</td>
<td>coral2</td>
</tr>
<tr>
<td>205</td>
<td>91</td>
<td>coral3</td>
</tr>
<tr>
<td>139</td>
<td>62</td>
<td>coral4</td>
</tr>
<tr>
<td>255</td>
<td>99</td>
<td>tomato1</td>
</tr>
<tr>
<td>238</td>
<td>92</td>
<td>tomato2</td>
</tr>
<tr>
<td>205</td>
<td>79</td>
<td>tomato3</td>
</tr>
<tr>
<td>139</td>
<td>54</td>
<td>tomato4</td>
</tr>
<tr>
<td>255</td>
<td>69</td>
<td>OrangeRed1</td>
</tr>
<tr>
<td>238</td>
<td>64</td>
<td>OrangeRed2</td>
</tr>
<tr>
<td>205</td>
<td>55</td>
<td>OrangeRed3</td>
</tr>
<tr>
<td>139</td>
<td>37</td>
<td>OrangeRed4</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>red1</td>
</tr>
<tr>
<td>238</td>
<td>0</td>
<td>red2</td>
</tr>
<tr>
<td>205</td>
<td>0</td>
<td>red3</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>red4</td>
</tr>
<tr>
<td>255</td>
<td>20</td>
<td>DeepPink1</td>
</tr>
<tr>
<td>238</td>
<td>18</td>
<td>DeepPink2</td>
</tr>
<tr>
<td>205</td>
<td>16</td>
<td>DeepPink3</td>
</tr>
<tr>
<td>139</td>
<td>10</td>
<td>DeepPink4</td>
</tr>
<tr>
<td>255</td>
<td>110</td>
<td>HotPink1</td>
</tr>
<tr>
<td>238</td>
<td>106</td>
<td>HotPink2</td>
</tr>
<tr>
<td>205</td>
<td>96</td>
<td>HotPink3</td>
</tr>
<tr>
<td>139</td>
<td>58</td>
<td>HotPink4</td>
</tr>
<tr>
<td>255</td>
<td>181</td>
<td>pink1</td>
</tr>
<tr>
<td>238</td>
<td>169</td>
<td>pink2</td>
</tr>
<tr>
<td>205</td>
<td>145</td>
<td>pink3</td>
</tr>
<tr>
<td>139</td>
<td>99</td>
<td>pink4</td>
</tr>
<tr>
<td>255</td>
<td>174</td>
<td>LightPink1</td>
</tr>
<tr>
<td>238</td>
<td>162</td>
<td>LightPink2</td>
</tr>
<tr>
<td>205</td>
<td>140</td>
<td>LightPink3</td>
</tr>
<tr>
<td>139</td>
<td>95</td>
<td>LightPink4</td>
</tr>
<tr>
<td>255</td>
<td>130</td>
<td>PaleVioletRed1</td>
</tr>
<tr>
<td>238</td>
<td>121</td>
<td>PaleVioletRed2</td>
</tr>
<tr>
<td>205</td>
<td>104</td>
<td>PaleVioletRed3</td>
</tr>
<tr>
<td>139</td>
<td>71</td>
<td>PaleVioletRed4</td>
</tr>
<tr>
<td>255</td>
<td>52</td>
<td>maroon1</td>
</tr>
<tr>
<td>238</td>
<td>48</td>
<td>maroon2</td>
</tr>
<tr>
<td>205</td>
<td>41</td>
<td>maroon3</td>
</tr>
<tr>
<td>139</td>
<td>28</td>
<td>maroon4</td>
</tr>
<tr>
<td>255</td>
<td>62</td>
<td>VioletRed1</td>
</tr>
<tr>
<td>238</td>
<td>58</td>
<td>VioletRed2</td>
</tr>
<tr>
<td>205</td>
<td>50</td>
<td>VioletRed3</td>
</tr>
<tr>
<td>139</td>
<td>34</td>
<td>VioletRed4</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>magenta1</td>
</tr>
<tr>
<td>238</td>
<td>0</td>
<td>magenta2</td>
</tr>
<tr>
<td>205</td>
<td>0</td>
<td>magenta3</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>magenta4</td>
</tr>
<tr>
<td>255</td>
<td>131</td>
<td>orchid1</td>
</tr>
<tr>
<td>238</td>
<td>122</td>
<td>orchid2</td>
</tr>
<tr>
<td>205</td>
<td>105</td>
<td>orchid3</td>
</tr>
<tr>
<td>139</td>
<td>71</td>
<td>orchid4</td>
</tr>
<tr>
<td>255</td>
<td>187</td>
<td>plum1</td>
</tr>
<tr>
<td>238</td>
<td>174</td>
<td>plum2</td>
</tr>
<tr>
<td>205</td>
<td>150</td>
<td>plum3</td>
</tr>
<tr>
<td>139</td>
<td>102</td>
<td>plum4</td>
</tr>
<tr>
<td>224</td>
<td>102</td>
<td>MediumOrchid1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>209</td>
<td>95</td>
<td>238</td>
</tr>
<tr>
<td>180</td>
<td>82</td>
<td>205</td>
</tr>
<tr>
<td>122</td>
<td>55</td>
<td>139</td>
</tr>
<tr>
<td>191</td>
<td>62</td>
<td>255</td>
</tr>
<tr>
<td>178</td>
<td>58</td>
<td>238</td>
</tr>
<tr>
<td>154</td>
<td>50</td>
<td>205</td>
</tr>
<tr>
<td>104</td>
<td>34</td>
<td>139</td>
</tr>
<tr>
<td>155</td>
<td>48</td>
<td>255</td>
</tr>
<tr>
<td>145</td>
<td>44</td>
<td>238</td>
</tr>
<tr>
<td>125</td>
<td>38</td>
<td>205</td>
</tr>
<tr>
<td>85</td>
<td>26</td>
<td>139</td>
</tr>
<tr>
<td>171</td>
<td>130</td>
<td>255</td>
</tr>
<tr>
<td>159</td>
<td>121</td>
<td>238</td>
</tr>
<tr>
<td>137</td>
<td>104</td>
<td>205</td>
</tr>
<tr>
<td>93</td>
<td>71</td>
<td>139</td>
</tr>
<tr>
<td>255</td>
<td>225</td>
<td>255</td>
</tr>
<tr>
<td>238</td>
<td>210</td>
<td>238</td>
</tr>
<tr>
<td>205</td>
<td>181</td>
<td>205</td>
</tr>
<tr>
<td>139</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>224</td>
<td>224</td>
<td>224</td>
</tr>
<tr>
<td>224</td>
<td>224</td>
<td>224</td>
</tr>
<tr>
<td>227</td>
<td>227</td>
<td>227</td>
</tr>
<tr>
<td>227</td>
<td>227</td>
<td>227</td>
</tr>
<tr>
<td>229</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>229</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>232</td>
<td>232</td>
<td>232</td>
</tr>
<tr>
<td>232</td>
<td>232</td>
<td>232</td>
</tr>
<tr>
<td>235</td>
<td>235</td>
<td>235</td>
</tr>
<tr>
<td>235</td>
<td>235</td>
<td>235</td>
</tr>
<tr>
<td>237</td>
<td>237</td>
<td>237</td>
</tr>
<tr>
<td>237</td>
<td>237</td>
<td>237</td>
</tr>
<tr>
<td>240</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>240</td>
<td>240</td>
<td>240</td>
</tr>
<tr>
<td>242</td>
<td>242</td>
<td>242</td>
</tr>
<tr>
<td>242</td>
<td>242</td>
<td>242</td>
</tr>
<tr>
<td>245</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>245</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>247</td>
<td>247</td>
<td>247</td>
</tr>
<tr>
<td>247</td>
<td>247</td>
<td>247</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>252</td>
<td>252</td>
<td>252</td>
</tr>
<tr>
<td>252</td>
<td>252</td>
<td>252</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>0</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>0</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>144</td>
<td>238</td>
<td>144</td>
</tr>
<tr>
<td>144</td>
<td>238</td>
<td>144</td>
</tr>
</tbody>
</table>


25 Glossary

A

adjustable array
An array that is a dummy argument in a Fortran subroutine or function whose dimensionality is determined at runtime. The dimensionality of an adjustable array is supplied in the argument list in which the dummy array name appears, or by values in a COMMON block.

animation
A sequence of two or more images that, when displayed in a rapid sequence, provide the illusion of continuous motion.

annotation
A viewable object whose location and usually size are set relative to the viewport or data coordinate space of a base plot. There are three kinds of annotations: intrinsic annotations, embedded annotations, and external annotations.

annotation functions
The functions used to add and remove annotations to and from plot objects. Specifically, these functions are NhlAddAnnotation and NhlRemoveAnnotation for the C and Fortran interfaces.

annotation plot
An annotation that is a plot object and not simply a viewable object. An annotation plot is a subordinate base plot.

ANSI
The American National Standards Institute, an independent non-profit organization that creates and publishes U.S. national standards (such as Fortran, C, CGM, and so forth) taking input from all sectors of the technical community and the public at large. ANSI also works in collaboration with other standards organizations such as ISO (the International Standards Organization) and IEEE (the Institute of Electrical and Electronics Engineers).

API
Application Programming Interface.

application
A program written in C, Fortran, or NCL that utilizes any of the functionality of NCAR Graphics.

application class
Refers specifically to the class App. Objects that are instances of this class are used to keep track of resource databases. Every application must create at least one App object (this will be done automatically for you if you use NhlOpen).

Application Programming Interface
The programming interfaces (C, Fortran, and NCL) to the NCAR Graphics package. These interfaces provide access to the support functions defined in the classes as well as provide additional useful functionality.

application resource file
A resource file that is specific to a particular application. There are two application-specific resource files: a system application-specific resource file and a user application-specific resource file. The user can specify what directories the application-specific resource files are in. By default, the user application-specific resource file is in the local current directory and the system application-specific resource file is in the directory specified by the setting of the environment variable NCARG_SYSAPPRES. Resources defined in the user application-specific resource file will override resources defined in the system application resource file.

area fill pattern
A pattern to use for filling a polygonal area. The patterns are selected by using an integer fill index into a table of patterns.
arithmetic operator
An operator that applies to variables having a numeric data type. Examples are "+" (addition) and "*" (multiplication).

ASCII
Stands for "American Standard Code for Information Interchange." This is an ANSI Standard specifying a set of 128 characters with their associated coded integer representations.

ASCII file
NCL: A data file that contains integers or floating point data values in ASCII format.

aspect ratio
Specifies the height-to-width ratio of a plot. This term is also applied to characters. For example, characters with an aspect ratio of 2.0 are twice as tall as they are wide.

associative operator
A binary operator that obeys the law of associativity: i.e. a binary operator "R" such that (aRb)Rc = aR(bRc) for all legal operands a, b, and c.

attribute
NCL: A singly-dimensioned datum of any type that is assigned to a variable using the ' @ ' operator. An attribute of a variable contains descriptive information about the variable.

background color
The color that will be used as a background color for the entire viewable surface of a physical workstation when plots are drawn on it.

base plot
A plot object responsible for setting the viewport of zero or more plot members relative to its own viewport. There are two kinds of base plot: primary base plots and subordinate base plots. At creation, any plot object is a primary base plot. A plot object ceases to be a base plot when added to another plot object as an overlay. When added as an annotation a plot object becomes a subordinate base plot. A plot object must be a primary base plot for users to draw it or change its workstation.

binary file
A file whose contents are to be interpreted as a sequence of bits, rather than characters. There are different flavors of binary files. A "flat" binary file is a sequence of bits with no ancillary information about the file contents. This type of file is created and read by C programs. Fortran creates and reads flat binary files only when in direct-access mode. All records are the same size in a flat binary file. By default, Fortran creates another type of binary file which can contain variable-length records. This is called a sequential-access binary file. In a sequential-access binary file, record length information is embedded prior to each record.

block statement
A statement that requires one or more individual statements bracketed by delimiters indicating the beginning and end of the block. Examples of block statements are: do-end do, if-then-end if, setvalues-end setvalues.

bounding box
For View class objects, the bounding box for such objects is the smallest rectangle in NDC space that contains all of the marks that would appear on an output workstation if the object were drawn.

built-in function or procedure
A built-in function or procedure in NCL is one that is built into the code for NCL, and hence you don't need to load any NCL scripts to use it. Examples of built-in functions include fspan and addfile. Examples of procedures include system and delete. An example of a function that is not considered a built-in function is gsn_open_wks.
cairo
From the wikipedia entry: a software library used to provide a vector-based graphics, device independent API for software developers. In NCL V5.2.0, new cairo workstations were added to provide PNG, TIFF, and alternate PS and PDF output.

cartesian grid
A cartesian grid is the simplest form of a structured grid. It simply consists of square cells arranged uniformly in a matrix. The grid cells are evenly spaced in all directions, and for every column there is the same number of rows and vice versa.

See also rectilinear and curvilinear grids.

CCM history tape format
A proprietary data format used by atmospheric climate simulation models developed at NCAR. (CCM stands for Community Climate Model.)

C function prototype
A C function declaration that declares a function's return type, how many arguments the function takes, and the types of the arguments.

CGM
Computer Graphics Metafile.

child
A relationship that holds between objects. If "A" and "B" are objects, then B is a child of A provided that when B was created, A was specified as being its parent (either in the fourth argument of an NhlCreate call, or in a NCL create expression), or B was made a child of A by using the NhlChangeWorkstation function. If B is a child of A, then the following conditions apply:

1. B inherits the resource database of A. If viewable, B will display to the same workstation as A; if A is a workstation, then B will draw to A.
2. Destroying A will destroy B.
3. Resources can be specified in resource files as: {App obj name} . {parent of A} . {Name of A} . {Name of B} . {resource of B} : {value}.

child/parent hierarchy
The tree structure determined by the child/parent relationship existing among all current objects in an application. A child can have only a single parent, but a parent may have many children. The child/parent hierarchy should not be confused with the class hierarchy. See also class hierarchy.

class
A template for defining objects that specifies variables, and procedures that operate on those variables. In the context of the NCAR HLU library, the class variables are called resources and the class procedures are called support functions. Objects are members, or instances, of a class formed by assigning specific values to the variables in the class.

class hierarchy
Each class, except the base class, is derived from some other class. The tree structure determined by the derived-class/superclass relationships among all of the classes is called the class hierarchy. The class hierarchy should not be confused with the child/parent hierarchy.

color index
An integer value that represents an index into the current color table. Index 0 represents the background color and 1 the foreground color. See also named color and gsn_draw_colormap.

color map
Same as color table.
color table
A table that associates integer values (called color indices) with RGB color values. In NCL, color tables contain up to a maximum of 256 colors (including the background and foreground colors). There are several predefined color tables, or you can define your own. See also gsn_draw_colormap.

command
NCL: Same as an NCL statement.

comment line
A line in an NCL code beginning with a semi-colon (;). A line in an NCL resource file beginning with an exclamation point (!). Comment lines contain descriptive information about the code.

composite class
A class that combines the resources of other classes with its own. A composite class inherits resources and functions from its superclass and it shares the resources from its composite members by the process of resource forwarding. If the composite class members have support functions, these functions do not apply to the composite class.

composite class member
A class used as part of the functionality of a composite class.

composite class resource
A resource available to a composite class by way of resource forwarding from a member class of the composite class.

Computer Graphics Metafile
A graphics metafile is a file that contains encoded vector graphics elements such as lines, colors, dash patterns, markers, and so forth. The Computer Graphics Metafile (CGM) is a precisely-defined formatting for a graphics metafile as defined and standardized by ANSI. NCL produces a version of the CGM that is called a conforming private encoding that can easily be converted to and from standard CGM by using the filters ncgm2cgm and cgm2ncgm. The NCAR private encoding is also called NCAR CGM, or NCGM.

contour plot
A plot of 2D data containing contour lines (lines marking points of equal elevation) to indicate surface shape. Contour plots may have color fill between contour lines and may have label bars and annotations.

coordinate addressing
A way of indexing array elements by specifying coordinate values rather than the normal integer array indices. Coordinate addressing is effected by using coordinate variables.

coordinate subscripts
Coordinate subscripts use the coordinate variables associated with a variable to determine which indexes are used in the selection. When specifying a coordinate subscript, braces ‘{‘ and ‘}’ indicate the start and end values of the coordinate variable that will be used to select the indexes.

coordinate variable
Another word for coordinate variable.

coordinate variable
NCL: A value associated with a named dimension of a variable or file variable that contains numerical coordinate information for each index of the dimension. Coordinate variables must be singly-dimensional values. Warnings are produced if the array of values assigned is not monotonically increasing or decreasing.

ctrans
The NCGM interpreter distributed with NCL.

curvilinear grid
A curvilinear grid is one which cannot be uniquely accessed by a pair of one-dimensional coordinate arrays. These grids require a pair of two-dimensional arrays to describe grid point locations.
See also cartesian and rectilinear grids.

D

dash pattern
A pattern (such as "solid", "dotted", and so forth) to use as a line style when lines are plotted using the NhlDraw function. Dash patterns are selected by using an integer index into a table of dash patterns.

data classes
Any of the classes that are used to provide user input data to any of the objects that utilize such data. These classes are the CoordArrays class, the CoordArrTable class, and the ScalarField class.

data conversion
The process of converting data stored in one format to another format, such as converting data stored as integers to data stored as floating point numbers. Some objects, such as ScalarField, perform automatic data conversions.

data coordinate space
The coordinate space that is appropriate to input data. Transformations can be effected between data coordinate space and NDC (see Normalized Device Coordinates).

data specific resource
A resource of a particular class, such as the XyPlot class, that can be used to modify the attributes of data supplied via a DataSpec object. Data specific resources can be used to control attributes such as curve colors, dash patterns, marker sizes, marker colors, and so forth.

data transformation
A process that transforms data from one coordinate space to another, such as transforming data in logarithmic space to data in linear space.

data type
A data type is a representation of data that defines a size and valid range for numerical data or provides a reference to a file or HLU graphical object.

decision statement
A language construct allowing for conditional program execution based on the truth or falsity of an expression. The basic decision statement in NCL is if.

derived class
See subclass.

DODs
Distributed Ocean Data. Now referred to as OPeNDAP.

drawable object
See viewable object.

draw function
Specifically, either the NhlDraw function of the C or Fortran interfaces, or the draw function of NCL, that is invoked to plot a View object.

E

embedded annotation
An annotation that may be incorporated as part of the functionality of subclasses of the Transform class. It is managed internally by the controlling PlotManager.

Encapsulated PostScript
Encapsulated PostScript (EPS) is a subset of regular PostScript. The restrictions placed on EPS files are for making it an appropriate format for importing into applications that import PostScript.

**Encapsulated PostScript Interchange Format**

Encapsulated PostScript Interchange Format (EPSI) files are Encapsulated PostScript files that have a "preview bitmap" that represents the PostScript image contained in the file. The bitmap (and it is a bitmap and not a color map) can be used by an importing application to quickly display a picture of the imported file.

**Endian**

[Note: this definition was taken straight from Wikipedia]

Endianness generally refers to sequencing methods used in a one-dimensional system (such as writing or computer memory). The two main types of endianness are known as big-endian and little-endian. Systems which exhibit aspects of both conventions are often described as middle-endian. When specifically talking about bytes in computing, endianness is also referred to as byte order.

**Numeric**

NCL: Starting from 5.2.0, NCL has added int64 (aka long long), uint64 (unsigned long long), ulong (unsigned long), uint (unsigned int), and ushort (unsigned short). These newly added data types are named as numeric.

**EPS**

see Encapsulated PostScript

**EPSI**

see Encapsulated PostScript Interchange Format

**Error class**

A class that is used to configure the error reporting module of the HLU library. For any application, there is precisely one error object created, and it is created automatically for you. The error class defines several resources for controlling error reporting.

**Expression**

NCL: Any sequence of NCL operators and operands that results in the computation of a value. In particular, any literal value is an expression and any variable is an expression. Also, arrays are expressions as well as functions. Operators applied to expressions are expressions.

**External annotation**

An annotation consisting of an arbitrary user-created viewable object added to a plot object. The user controls the location and size relative to the base plot by manipulating the resources of a user-accessible AnnoManager object.

**File**

NCL: A data file residing external to NCL in one of NCL’s supported data formats.

**File name suffix**

A suffix appended to a file name to indicate its type. NCL recognizes the following supported suffixes: ".nc" for netCDF, ".hdf" for HDF, ".h5" for HDF5, ".he" ".he2" for HDFEOS, ".he5" for HDFEOS5, ".grb" ".grib" ".grb2" ".grib2" for GRIB, and ".ccm" for CCM History Tape. The obsolete suffix ".cdf" for a netCDF file is also recognized.

**File variable**

NCL: A variable, created by the NCL addfile function, that contains a reference to a file.

**Fill value**

NCL: Same as missing value.

**Fixed grid**
A fixed grid is a type of a rectilinear grid where each grid point can be uniquely accessed by one-dimensional, monotonically increasing or decreasing arrays (i.e. the coordinates are orthogonal). In cartesian coordinates, these may refer to the "x" and "y" coordinates, while on the globe these are longitude and latitude arrays. The grid spacing may be different in the latitude (y) and longitude (x) coordinates, but it is constant.

The special case where the grid spacing is the same in the latitude/longitude directions is called an "equally spaced" grid. Pole points may or may not be present. Some examples include: 1x1, 2x5, and 2.5x2.5 degree grids.

**Fixed offset grid**
A fixed-offset grid is analogous to the fixed grid, but refers to the special case where the latitude/longitude grids are offset for the traditional Greenwich Meridian or poles.

**Fontcap**
A file that contains detailed information used to plot characters. Fontcaps have a human-readable ASCII form and a binary form that is readable by ctrans.

**Foreground color**
The color associated with color index 1. This is used as the default color in drawing viewable objects.

**Fortran 90 interface block**
A sequence of Fortran 90 statements (bracketed by special delimiting statements) used to describe a procedure interface. The statements in the interface block contain a declaration for the procedure and declarations for the dummy arguments and no executable statements.

**Function**
- **HLU:** Any member of the NCL functions or the HLU API.
- **NCL:** An identifier with a list of parameters separated by commas and enclosed in parentheses. Functions return values when called. A function is defined by NCL source unlike the NCL intrinsic function.

**Gaussian grid**
A Gaussian grid is a type of a rectilinear grid one where each grid point can be uniquely accessed by one-dimensional latitude and longitude arrays (i.e. the coordinates are orthogonal). The longitudes are equally spaced while the latitudes are unequally spaced according to the Gaussian quadrature. There are no grid points at the poles. See also rectilinear and curvilinear grids.

**GIF**
A file format used for the storage and on-line retrieval of bitmapped graphical data. GIF stands for "Graphical Interchange Format"; it was created by the CompuServe Corporation in 1987.

**Graphcap**
A file that contains detailed information used to define the capabilities of a specific plotting device. Graphcaps have a human-readable ASCII form or a binary form that is readable by ctrans.

**Graphical object**
- **NCL:** An NCL value of type graphic. A graphical object is an identifier for an HLU object.

**Graphical User Interface**
A non-programmatic, graphical, interface to the functionality of NCAR Graphics. Such an interface is sometimes referred to as a "point-and-click" interface, since that is how the interaction is accomplished.

**GRIB**
GRIB (GRIdded Binary) is a data format used for the storage of historical and forecasted weather data. The format is standardized by the World Meteorological Organization (WMO). There are two versions: GRIB1 and GRIB2. NCL supports both versions.
**GSUN**
Acronym for "Getting Started Using NCL."

**GSUN scripts**
NCL scripts that provide an "easy" interface to the graphics capabilities of NCL. Some examples are gsn_xy and gsn_csm_contour_map.

**GUI**
Graphical User Interface.

**H**

**HDF**
HDF (Hierarchical Data Format) is a multi-object file format, developed at NCSA, that facilitates the transfer of various types of data between machines and operating systems.

**High Level Utilities**
Objects, like XyPlot objects, Contour objects, TextItem objects, and so forth, that can be created and manipulated by a set of library functions, callable from either a C program, a Fortran program, the NCAR Command Language, or a GUI. High Level Utilities are also referred to as HLUs and are to be distinguished from the Low Level Utilities, or LLUs.

**HLU resource string**
A character string identifying a particular resource of a class. These are the resources listed in the descriptions of the classes.

**HLUs**
High Level Utilities.

**HSV**
Acronym for Hue/Saturation/Value. An additive color system based on the attributes of color (hue), percentage of white (saturation), and value (brightness or intensity).

**I**

**identifier**
NCL: The name of a variable, function, or procedure.

**immediate mode**
A mode used by certain API functions that produces immediate drawing without invoking a draw function.

**instance**
When specific values are assigned to all the resources defined in a class, the result is an instance of that class. Any instance of any class is called an object. Default values exist for all resources; before creating an object, users may override any default value.

**instance hierarchy**
Same as child/parent hierarchy.

**inheritance**
A class is said to inherit resources or support functions from its superclasses, since those functions and resources are available to the subclass. See superclass, and composite class.

**interpreter**
A program that transforms statements into machine code a statement at a time. The ncl executable is an interpreter of the NCL language.

**intrinsic annotation**
An annotation available as composite class member of the PlotManager class. Intrinsic annotations are available to any class of plot object and include TickMark, Title, LabelBar, and Legend annotations. The PlotManager manages these annotations internally.

**intrinsic function**
- **NCL**: An identifier with a list of parameters, the parameters being separated by commas enclosed and in parentheses. Intrinsic functions return values when called. An intrinsic function is not defined by NCL source; it is a C or Fortran routine that has been added to the NCL function set. Intrinsic functions often perform operations that NCL source does not support.

**irregular rectangular coordinate space**
- A 2-dimensional rectangular grid that has unequal spacing along the X and/or Y axes.

**ISO**
- The International Standards Organization that publishes international standards. (see ANSI).

**intrinsic procedure**
- **NCL**: An identifier with a list of parameters, the parameters being separated by commas enclosed in parentheses. An intrinsic procedure is not defined by NCL source; it is a C or Fortran routine that has been added to the NCL procedure set. Intrinsic procedures often perform operations that NCL source does not support.

**J**

**K**

**keyword**
- **NCL**: A word reserved by NCL that not allowed to be used as a variable or function name.

**L**

**label bar**
- A specialized label consisting of a bar of filled rectangular areas that are labeled to correspond with areas from an adjoining plot. Label bars can be filled with black-and-white patterns, with color, or with both. Label bars are commonly used with contour plots and with other types of plots where area pattern fills or color are used to differentiate values in the plot.

**landscape**
- See portrait

**lazy evaluation**
- **NCL**: The process whereby relational expressions are assigned a value as soon as it is possible to do so, without necessarily evaluating all of the components in the expression. For example, the expression `(1 .lt. 3) .or. (2 .lt. 1)` can be assigned the value True immediately after evaluating `(1 .lt. 3)` without having to evaluate `(2 .lt. 1)`.

**legend**
- A specialized annotation that formats a series of lines or markers of varying styles along with adjoining explanatory labels. Legends are designed to serve as “keys” for an associated plot.

**literal array**
- **NCL**: An array of values specified using literal values, these values being separated by commas and enclosed in `'(/' and ')/'`.

**literal value**
- **NCL**: A single scalar value expressed by its actual string value (i.e. not referenced by a variable). For example, 1, 1.414 and "string" are literal values.

**LLUs**
- Low Level Utilities.
local resources,
Resources defined in a particular class that are not inherited from another class.

logical operator
A operator that returns a true value or a false value depending on the truth or falsity of its operands. The logical operators in NCL are: .and., .or., .xor., .not..

loop statement
A language construct that allows for code repetition with incremental values set for a variable or variables. The looping statements in NCL are: do and do while.

Low Level Utilities
Traditional NCAR Graphics as it existed before the HLUs or NCL were developed. It is a package of about 500 graphics routines. User entries have both C and Fortran interfaces.

M

machine-independent data format
Same as Network-transparent data format.

marker
See polymarker.

member class
One of the class components of a composite class.

metadata
NCL: Information used to describe data, such as dimension names, variable attributes, valid ranges, and so forth.

metafile
A file containing encoded graphical elements. Metafiles are used for storing and transporting graphics images. In the context of NCL and NCAR Graphics "metafile" is generally synonymous with NCGM.

missing value
NCL: A special value for a variable or array element indicating that no legal data has been specified for that quantity. See the section on missing data in the evaluation of expressions in the NCL Language documentation details on how these missing values are handled.

monotone
A sequence of numeric values is monotone (or monotonic) if either: each element in the sequence is larger than (or equal to) its predecessor, or each element in the sequence is smaller than (or equal to) its predecessor. A sequence is monotonically increasing if each element in the sequence is larger than its predecessor; a sequence is monotonically decreasing if each element in the sequence is smaller than its predecessor. A sequence is monotonically non-decreasing if each element in the sequence is larger than, or equal to, its predecessor. A sequence is monotonically non-increasing if each element in the sequence is smaller than, or equal to, its predecessor.

N

named color
A string representing a predefined color. Named colors can be used with just about any graphical resource that defines the color of a plot attribute (like a line color or a polygon fill color). In order to use a named color, that color must be part of your current color table. See also color index.

named dimension
NCL: A dimension of a variable or file variable that has been assigned a name using the '!' operator.

NCAR Command Language
A language written for the purpose of interactive data manipulation and display. NCL has a command line interface and will accept netCDF, HDF, HDFEOS, HDF5, HDFEOS5, GRIB1, GRIB2, or ASCII input files. NCL also provides an easy interface to the HLUs.

**NCAR Computer Graphics Metafile**

The NCAR private binary encoding of a Computer Graphics Metafile.

**NCGM**

NCAR Computer Graphics Metafile.

**NCL**

NCAR Command Language.

**NCL resource list**

NCL: An NCL resource list is a list of HLU resource strings followed by a " : " followed by a valid NCL expression.

**NDC**

Normalized Device Coordinates. A coordinate system that describes positions on a virtual plotting device. The lower left corner corresponds to (0,0), and the upper right corner corresponds to (1,1). NDC space will be mapped onto the largest square which will fit on an actual plotting device. PostScript output is centered on the page by default, but options exist for positioning PostScript output anywhere on the page. See the PSWorkstation for details.

**ncl**

Refers to the interpreter that interprets NCL statements.

**netCDF**

NetCDF (network Common Data Form) is an interface for scientific data access and a library that provides an implementation of the interface. There are different kinds of NetCDF files: classic, 64-bit offset, netCDF-4 classic, and netCDF-4. See the NetCDF FAQ for more information.

**netCDF 64-bit offset**

In 2004, the 64-bit offset format variant was added. Nearly identical to the netCDF classic format, it allows users to create and access far larger datasets than were possible with the original format. (A 64-bit platform is not required to write or read 64-bit offset netCDF files.)

**netCDF classic**

The classic format was the only format for netCDF data created between 1989 and 2004. As of NetCDF version 4.2.x, it is still the default format for new netCDF data files, and the form in which most netCDF data is stored. Some users think of this as "NetCDF-3".

**netCDF-4 classic**

At the same time that the netCDF-4 format was as introduced, the "netCDF-4 classic" format was added for users who needed the performance benefits of the new format (such as compression) without the complexity of a new programming interface or enhanced data model.

**netCDF-4**

In 2008, the netCDF-4 format was added to support per-variable compression, multiple unlimited dimensions, more complex data types, and better performance, by layering an enhanced netCDF access interface on top of the HDF5 format.

**network-transparent data format**

A format for encoding data that removes any machine dependencies that might be involved in encoding the data. Typical examples of such data formats are netCDF and HDF.

**numeric**

NCL: Any data type that represents a numerical value. The numeric data types are: double, float, int64 (aka long long), uint64 (unsigned long long), long, ulong (unsigned long), integer, uint (unsigned int), short, ushort (unsigned short), and byte.
Starting from 5.2.0, NCL has added int64 (aka long long), uint64 (unsigned long long), ulong (unsigned long), uint (unsigned int), and ushort (unsigned short). These newly added data types are named as numeric, means extra-numeric.

In order to make NCL backward compatible, we keep the name numeric refers to data types: double, float, long, integer, short, and byte, but create a new name snumeric, means super-numeric, which includes both numeric and enumeric.

**numeric data type**
- A data type for numeric quantities. In NCL the numeric data types are: double, float, int64, uint64, long, ulong, integer, uint, short, ushort, and byte.

**non-numeric data type**
- A data type for non-numeric quantities. In NCL the non-numeric data types are: string, character, graphic, file, and logical.

**O**

**object**
- An object is created from a class by assigning specific values for the class resources. See class and instance.

**OPeNDAP**
- OPeNDAP, an acronym for "Open-source Project for a Network Data Access Protocol", is a data transport architecture and protocol widely used by earth scientists that simplifies all aspects of scientific data networking, allowing simple access to remote data. Visit www.opendap.org for more information.

**output primitive**
- Procedures and functions for producing graphics output at the lowest level. GSUN procedures/functions exist for drawing lines (gsn_polyline, gsn_polyline_ndc, gsn_add_polyline), text (gsn_text, gsn_text_ndc, gsn_add_text), filled areas (gsn_polygon, gsn_polygon_ndc, gsn_add_polygon), and markers (gsn_polymarker, gsn_polymarker_ndc, gsn_add_polymarker).

**overlay**
- A transform overlaid on a base plot using the add overlay function. The base plot sets the viewport of the overlay to match its own and transforms the coordinate data of the overlay into its own coordinate space. Only that portion of the overlay's coordinate space that intersects the coordinate space of the managing plot will be visible in the plot output. If the overlay is a plot object, it gives up its base plot status. The base plot to which the overlay is added assumes responsibility for managing the overlay's plot members.

**overlay functions**
- The functions used to add and remove overlays to and from plot objects. Specifically, these functions are NhlAddOverlay and NhlRemoveOverlay for the C and Fortran interfaces.

**overlay plot**
- An overlay created from a plot object rather than from a simple transform.

**overlay sequence**
- The ordering of the transforms in a plot or subplot that contains overlays. The base plot is always first, followed by each overlay in an order that may be manipulated through the overlay functions. The overlay sequence determines the basic drawing order of the plot. The base plot is drawn first; each succeeding overlay is drawn on top of the preceding transforms. Annotations are not affected by the overlay sequence; they always drawn after all the transforms.

**P**

**parent**
- A relationship that exists between objects. If A and B are objects, then A is a parent of B if and only if B is a child of A. See child for more information.
PDF
An acronym for Portable Document Format, a file format created by Adobe Systems, Inc. It uses the PostScript printer description language and is highly portable across computer platforms. PDF documents are created with Adobe Acrobat or other programs and can be viewed with Adobe Acrobat Reader and other PDF reader programs.

plot
Depending on context, the word plot may be used to mean:

- A plot object.
- A primary base plot and all its plot members.
- The output resulting from drawing a primary base plot and all its plot members.
- The output resulting from drawing any arbitrary collection of viewable objects.

plot_class
NCL: The plot_class in NCL is the same as the HLU class pointer used in the HLU API to specify what type of object to create. The NCL plot_class identifier is spelled the same as the HLU class pointer.

plot member
A viewable object managed by a base plot. If the base plot is a subordinate base plot then the object is indirectly a plot member of the complete plot managed by the primary base plot. Drawing the primary base plot causes all its plot members to be drawn. A plot member is either an overlay or an annotation. Although an annotation can be any arbitrary view, an overlay must be a view belonging to the Transform class. A plot member must belong to the same Workstation as its base plot and cannot be drawn independently. A view cannot belong as a plot member to more than one base plot at a time.

plot object
A Transform object instantiated with an active PlotManager. In general, unless otherwise restricted by their specific class, plot objects have the ability both to manage other viewable objects as plot members and to be managed as plot members themselves. A plot object that manages plot members is called a base plot. If the managing plot object is itself managed as a plot member, it is a subordinate base plot. If it manages itself (i.e. is not a plot member) it is a primary base plot. At creation, all plot objects are primary base plots.

polymarker
An array of coordinates specifying positions where certain specified symmetric symbols (markers) such as circles, dots, and so forth will be plotted.

portrait mode
(Definition taken from Wikipedia) Portrait mode and landscape mode refer to the orientation of text (and pictures) on a printed page. (The paper must be a rectangle, however in practice square sheets are hardly ever used.) In portrait mode the text is printed on the paper such that the reader will turn the long side of the paper vertical and the short side horizontal. In landscape mode on the other hand, the long side is horizontal, and the short side vertical (like most landscape paintings).

PostScript
A general-purpose programming language that contains a rich set of graphics operators. PostScript is produced by many popular word processing and graphics packages and can be displayed on a wide variety of printers, plotters, and workstation screens.

primary base plot
A self-managing plot object. A base plot that directly manages any number of plot members, consisting of overlays and annotations, but is not itself a plot member. At creation all plot objects are primary base plots. A plot object must be a primary base plot for the user to draw it or change its workstation.

primitive
see output primitive

procedure
NCL: An identifier with a list of parameters, these parameters being separated by commas and enclosed in parentheses. Procedures do not return values when called.

PS

see PostScript

R

rectilinear grid

A rectilinear grid is very similar to a cartesian grid in that it is a basic rectangular matrix arrangement of data. The one difference is that the uniform spacing restriction is lifted. Rectilinear grids are usually represented by one-dimensional coordinate variables.

See also cartesian, gaussian, and curvilinear grids.

relational operator

A operator that returns a true value or false value depending on a relation between its operands. The relational operators in NCL are: .le., .lt., .ge., .gt., .ne., .eq..

resource

A variable defined as part of the definition of a class. Resource values in objects can be set by using a NhlCreate function or a NhlSetValues function. Resource values can be retrieved using a NhlGetValues function.

resource file

A file that can be used to set values for resources. There are four different resource files. See system resource file and user resource file.

resource forwarding

A technique that makes the resources of member classes available to a composite class.

RGB

Stands for the red, green, blue color space where colors are specified as triplets of floating point numbers between 0.0 and 1.0 inclusive. The triplet gives the percentage intensities for the red, green, and blue components of a color. The triplet <1.,.0.,0.> would indicate a full percentage of red, and no green or blue percentage, for example.

RGBA

Stands for the red, green, blue, alpha color space where colors are specified as quadruplets of floating point numbers between 0.0 and 1.0 inclusive. The first three values of the quadruplet gives the percentage intensities for the red, green, and blue components of a color. The fourth value gives the percentage of opaqueness of that color. An opacity value of 1.0 means the color is fully opaque, and a value of 0.0 means it is fully transparent. The quadruplet <1.,0.,0.,0.5> would indicate the color red at half opacity. The usage of RGBA colors was introduced in NCL V6.1.0.

S

scalar

NCL: A single element of data of any type is referred to as a scalar value.

scalar_logical_expression

NCL: A single element value of the logical data type with no missing values.

scope

NCL: The range or area within a program in which an identifier is meaningful.

script

A file containing a sequence of program statements that can be submitted to an interpreter for execution.
**self-describing data format**
A format for encoding data that can contain information that describes the data being encoded. Typical examples of such data formats are netCDF, HDF, and HDF5.

**shape**
The number of dimensions of an array. The statement `a = new(/2, 3, 5/), float)` would create an array `a` of shape 3 (i.e. `a` has three dimensions).

**simple overlay**
An overlay created from a simple transform.

**simple transform**
A transform object created without an active PlotManager. Unlike a plot object, a simple transform cannot manage any plot members, either as annotations or as overlays. Therefore, the only elements that appear when a simple transform is drawn are those implemented within the object itself. However, unless restricted by its particular class, a simple transform may itself become an annotation or an overlay of a base plot. Since a plot object has all the capabilities of a simple transform and none of the limitations, the main reason for creating a simple transform would be to conserve system resources when PlotManager capabilities are not required.

**size**
The number of elements in array dimensions. The statement `a = new(/2, 3, 5/), float)` would create an array `a` that has a first dimension of size 2, a second dimension of size 3, and a third dimension of size 5.

**statement**
A non-comment line of code (or a line of code prior to any commenting). A line of NCL code contains all continuations resulting from use of the "\" symbol.

**stipple**
To cover an area with small dots.

**streamline**
The path an idealized particle would follow if introduced into a wind or fluid flow. For example (as an approximation to the ideal), the path a speck of dust would take in a wind.

**Streamline plot**
A plot representing a vector field using streamlines, based upon 2-dimensional data. It may also contain tick marks and titles.

**stride**
The increment indicator in a subscript specifier. Using `m:n:i` as a subscript means to take the individual subscript values starting with `m` and ending with `n` in strides of `i`. The stride must always be an integer and should be thought of as a skip indicator rather than an additive increment value, since coordinate subscripts may not always be integers. A stride of 2 means to take every second value after the first, a stride of 3 means take every third value, and so forth.

**snumeric**
NCL: Starting from 5.2.0, NCL has a new name snumeric, means super-numerica, which includes both numeric and enumeric. snumeric includes: double, float, int64 (aka long long), uint64 (unsigned long long), long, ulong (unsigned long), integer, uint (unsigned int), short, ushort (unsigned short), and byte.

**statement**
NCL: A single language construct within NCL that performs a specific task.

**statement list**
NCL: A sequence of statements separated by a carriage returns (\n).

**structured grid**
Structured grids are the exact opposite of unstructured grids. A structured grid gets its name from the nature of having a structure implicitly defined by the arrangement of the data. A structured grid has a basic rectangular matrix structure that makes storage and use easy as integer offsets (Typically named i, j, and k) can be used to access individual data points. Data points are arranged into rectangular or cubic structures by simply connecting them to their neighboring i, j, and k cells. Three types of structured grids include cartesian, rectilinear, and curvilinear grids.

**subclass**
A class B is a subclass of A if B has in it all of the resources and support functions of A (B may have additional resources and support functions as well). If B is a class derived from A, then B is said to inherit its resources and support functions from A. If B is derived from A, then it is also said that B is a subclass of A.

**subordinate base plot**
A plot member that is an annotation plot. A subordinate base plot sets the viewport of the plot members it controls, while its own viewport is set by the base plot that controls it. Unlike a primary base plot, the user cannot directly draw or change the workstation of a subordinate base plot. A subordinate base plot and the plot members it manages are known collectively as a subplot.

**subplot**
The portion of a plot that is managed by a subordinate base plot.

**superclass**
A class A is a superclass of class B if A is on the same branch of the class hierarchy tree and A is higher on that branch.

**superclass resource**
A resource that one class inherits from a superclass.

**supported data format**
Any of the formats that can be read by the NCL addfile function. The supported formats are: netCDF, HDF, HDFEOS, HDF5, HDFEOS5, GRIB (1 and 2), and CCM History Tape. You do not have to know the details of these formats in order to use them with NCL.

**support function**
A function defined as part of a class definition.

**system resource file**
There are four files where resources can be set - two of these files are system resource files and the other two are user resource files. The name of one of the system resource files is specified by the setting of the environment variable NCARG_SYSRESFILE; the other system resource file is specified in an application-specific manner. See application resource file.

**text function codes**
Special characters embedded in a text string, usually starting and ending with the colon (":") character. Text function codes provide the capability for selecting various fonts, doing superscripts and subscripts, taking complete control of the positioning of characters relative to one another, and last but not least, "zooming" characters in either width or height, or both.

**tick marks**
Marks along an axis of a plot that are perpendicular to the axis and serve to divide the axis (or parts of the axis) into equal or logarithmetically spaced parts.

**transform**
A Transform object.

**Transform class**
The Transform class is a subclass of the View class that supports transformations from data coordinate space into the NDC space occupied by the Transform class instance's viewport. Transform
subclasses may include the PlotManager class as a composite class member. The Transform class provides a resource for activating or deactivating the PlotManager when a Transform instance is created. In addition, the Transform class provides support functions for converting between data coordinate space and NDC space, for drawing immediate mode graphics primitives, and for adding and removing overlays and annotations from plot objects.

**Transform object**
An object that is an instance of the Transform class. Transform objects become plot objects when created with an active PlotManager instance. A Transform object created without an active PlotManager is called a simple transform.

**Unstructured mesh or grid**
Unstructured meshes are the exact opposite of structured grids, where the connectivity between points must be explicitly defined for every set of points. This makes them significantly more difficult and complex, and the nice relationships between neighboring cells or edges is no longer automatic and must be constructed manually. However, they are much more flexible in their ability to define complex shapes because they have no constraints on their arrangement.

Unstructured meshes are typically defined as points and cells. Cells are collections of points to define basic 2D or 3D primitives such as triangles, cubes, and tetrahedra.

**User resource file**
There are four files where resources can be set - two of these files are user resource files and the other two are system resource files. The name of one of the user resource files is specified by the setting of the environment variable NCARG_USRRESFILE; the other user resource file is specified in an application-specific manner. See application resource file.

**Variable**
NCL: A name that can contain a singly-dimensioned or multi-dimensioned data array, dimension names, coordinate variables, attributes, and so forth.

**Vector plot**
A plot representing a vector field by drawing glyphs that represent magnitude and direction at grid points based on 2-dimensional data. It may also contain tick marks, titles, and/or a label bar. Three glyph styles are available: a basic line-drawn arrow, a filled arrow with an option edge, and a standard wind barb.

**View**
A viewable object.

**Viewable object**
An object that is an instance of the View class.

**View class**
An object can be drawn only if it is an instance of the View class. The View class provides resources for sizing and positioning objects on an output device (workstation). The View class also provides a support function for determining the bounding box of a given object in the class. A View class object must have a Workstation class parent.

**View object**
A viewable object.

**Viewport**
For View objects, the viewport is a rectangular subregion of NDC space that specifies where the View object will be placed when drawn. The precise meaning of the viewport depends on the View object. For example, for XyPlot objects, the viewport specifies where the grid containing the curves will
be placed, and the labeling (if any) will be drawn outside of the viewport. On the other hand, for TextItem objects, the viewport will be a rectangle surrounding the text string.

**visualization block**
NCL: A group of NCL resources specified in either an NCL create, setvalues, or getvalues statement. Visualization blocks are used to create, modify, or inquire about the values of resources of objects.

**workstation**
Used in NCL and HLU terminology to mean a valid output device such as an X Window System display, a PostScript file, a PDF files, or an NCGM.

**workstation class**
A class that provides interfaces to specific output devices.

**wrapit interface block**
A sequence of Fortran 77 statements that specify a procedure and its arguments, similar to C function prototypes and Fortran 90 interface blocks. Wrapit interface blocks are used by wrapit77, a program for generating wrappers.

**wrapper function**
A C function that provides an interface between NCL and an existing Fortran or C subroutine or function. The wrapper intercepts an NCL function or procedure, does the appropriate argument checks and conversions, then calls the existing code.

**XY plot**
A plot containing curves made up of X/Y coordinate pairs. It may also contain tick marks, titles, and/or a legend.
26 Index

<table>
<thead>
<tr>
<th>!</th>
<th>character</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>character</td>
<td>12</td>
</tr>
<tr>
<td>_FillValue</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>addfile</td>
<td>27</td>
</tr>
<tr>
<td>addfiles</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Algebraic operators</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Arrays</td>
<td>Coordinate subscripting</td>
<td>23</td>
</tr>
<tr>
<td>Named subscripting</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Standard subscripting</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Subscription</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>asciiread</td>
<td>30, 33</td>
<td></td>
</tr>
<tr>
<td>ascifwrite</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Bar Charts</td>
<td>94</td>
</tr>
<tr>
<td>begin</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>break</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Built-in functions and procedures</td>
<td>244</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>cbinread</td>
<td>36</td>
</tr>
<tr>
<td>cbinwrite</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>cd_calendar</td>
<td>57, 170</td>
<td></td>
</tr>
<tr>
<td>cd_convert</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>cd_inv_calendar</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>cd_string</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>CDO</td>
<td>detrend</td>
<td>54</td>
</tr>
<tr>
<td>fldmean</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>runmean</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>seltimestep</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>selvar</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>timmean</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>timstd</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>yearmean</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Colormaps</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>Colors</td>
<td>named colors</td>
<td>250</td>
</tr>
<tr>
<td>continue</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>
Contour Line and Label Settings ................................................................. 157
Contours ................................................................................................. 83
craybinread ...................................................................................... 36
Crop White Space from Plot File ............................................................. 201
CSV
csv
reading ................................................................................................. 33
writing .................................................................................................. 42
Curvilinear Grids ................................................................................... 126

<table>
<thead>
<tr>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dash Pattern .................................................................................. 241</td>
</tr>
<tr>
<td>Data types</td>
</tr>
<tr>
<td>numeric ............................................................................................ 10</td>
</tr>
<tr>
<td>non-numeric .................................................................................... 10</td>
</tr>
<tr>
<td>numeric ............................................................................................ 10</td>
</tr>
<tr>
<td>dim_avg ........................................................................................... 50</td>
</tr>
<tr>
<td>dim_avg_Wrap ................................................................................ 50</td>
</tr>
<tr>
<td>dim_stddev ..................................................................................... 50</td>
</tr>
<tr>
<td>dim_stddev_n_Wrap ...................................................................... 50</td>
</tr>
<tr>
<td>dim_stddev_Wrap .......................................................................... 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>end ................................................................................................. 24</td>
</tr>
<tr>
<td>Environment Variables ................................................................. 202</td>
</tr>
<tr>
<td>Errors .............................................................................................. 221</td>
</tr>
<tr>
<td>ESMF Regridding ........................................................................... 175</td>
</tr>
<tr>
<td>Examples</td>
</tr>
<tr>
<td>Axis Annotations ........................................................................... 156</td>
</tr>
<tr>
<td>Bar Chart Example ........................................................................ 94</td>
</tr>
<tr>
<td>Bar Chart Multi Plot .................................................................... 95</td>
</tr>
<tr>
<td>Bar Chart Plot Displaying Values Above Or Below ...................... 96</td>
</tr>
<tr>
<td>Bipolar Grid MPI-ESM .................................................................. 129</td>
</tr>
<tr>
<td>Bipolar Grid MPI-ESM_subregion ............................................... 130</td>
</tr>
<tr>
<td>C Wrapper ..................................................................................... 197</td>
</tr>
<tr>
<td>Color Land and Ocean .................................................................. 158</td>
</tr>
<tr>
<td>Colormaps .................................................................................... 123</td>
</tr>
<tr>
<td>Compare Grid Resolutions .......................................................... 104</td>
</tr>
<tr>
<td>Contour Fill Pattern Plot ............................................................. 87</td>
</tr>
<tr>
<td>Contour Filled Plot ....................................................................... 85</td>
</tr>
<tr>
<td>Contour Plot .................................................................................. 83</td>
</tr>
<tr>
<td>Convert GMT colortable to NCL colormap .................................. 125</td>
</tr>
<tr>
<td>Convert Grads Colortable to NCL Colormap .................................. 125</td>
</tr>
<tr>
<td>Convert NaNs to _FillValue .......................................................... 203</td>
</tr>
<tr>
<td>Curvilinear Grid ............................................................................ 127</td>
</tr>
<tr>
<td>Curvilinear Data basic ................................................................... 126</td>
</tr>
<tr>
<td>Date Format ................................................................................... 171</td>
</tr>
<tr>
<td>Fortran Wrapper ............................................................................ 195</td>
</tr>
<tr>
<td>Globe with different grid resolutions .......................................... 145</td>
</tr>
<tr>
<td>Histogram Example ....................................................................... 97</td>
</tr>
<tr>
<td>Labelbars ....................................................................................... 163</td>
</tr>
<tr>
<td>Legends .......................................................................................... 165</td>
</tr>
<tr>
<td>Linear Regression .......................................................................... 51</td>
</tr>
<tr>
<td>Map Resolution Plot ...................................................................... 77</td>
</tr>
<tr>
<td>Map Settings .................................................................................. 74</td>
</tr>
<tr>
<td>Multiple Timeseries Plot ............................................................... 101</td>
</tr>
<tr>
<td>Panel Plot ...................................................................................... 108</td>
</tr>
<tr>
<td>Panel Plot 3 rows x 2 columns ...................................................... 110</td>
</tr>
<tr>
<td>Polar Plot NH ............................................................................... 110</td>
</tr>
<tr>
<td>Polylines, Polygon and Polymarker .............................................. 116</td>
</tr>
<tr>
<td>Projections Mollweide Plot ............................................................ 72</td>
</tr>
</tbody>
</table>

282
<table>
<thead>
<tr>
<th>Function Code</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbdirread</td>
<td></td>
</tr>
<tr>
<td>fbdirwrite</td>
<td></td>
</tr>
<tr>
<td>fbfindrecread</td>
<td></td>
</tr>
<tr>
<td>fbfindrecrewrite</td>
<td></td>
</tr>
<tr>
<td>fbfindwrite</td>
<td></td>
</tr>
<tr>
<td>Fill Pattern</td>
<td></td>
</tr>
<tr>
<td>function</td>
<td></td>
</tr>
<tr>
<td>Function Code</td>
<td>~ 149</td>
</tr>
<tr>
<td>subsctipts</td>
<td></td>
</tr>
<tr>
<td>superscript</td>
<td></td>
</tr>
</tbody>
</table>

Export file formats ................................................................. 6
external C code .................................................................. 195, 197
external Fortran code ................................................... 195
Loops

Logical operators and expressions

G

gsn_add_polygon .................................................. 116
gsn_add_polyline .................................................. 116, 119
gsn_add_polymarker ................................................. 116
gsn_add_shapefile_polygons ...................................... 119
gsn_add_shapefile_polylines ..................................... 119
gsn_add_shapefile_polymarkers .................................. 119
gsn_add_text ....................................................... 116, 151
gsn_csm_contour ................................................... 83, 93
gsn_csm_contour_map .............................................. 83
gsn_csm_contour_map_polar ..................................... 75
gsn_csm_map .......................................................... 75
gsn_csm_vector_map ................................................ 88
gsn_csm_xy ............................................................ 79, 94
gsn_define_colormap ............................................... 123
gsn_draw_colormap .................................................. 123
gsn_histogram ........................................................ 97
gsn_open_wks ......................................................... 201
gsn_panel .............................................................. 108
gsn_polygon ........................................................... 116
gsn_polygon_ndc ..................................................... 116
gsn_polyline .......................................................... 116
gsn_polyline_ndc .................................................... 116
gsn_polymarker ....................................................... 116
gsn_polymarker_ndc ................................................. 116
gsn_text ............................................................... 116
gsn_text_ndc .......................................................... 116

H

Histogram ..................................................................... 97

I

if - statement ................................................................ 24
Import file formats .................................................... 6
Insert JPEG or PNG ................................................... 172

L

Labelbars ..................................................................... 162
Legends ....................................................................... 165
ListSetType .................................................................. 29
cat ............................................................................. 29
join ............................................................................ 29
Logical operators and expressions ................................ 9
Logos .......................................................................... 172
Loops ........................................................................... 24
### M

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps</td>
<td>67</td>
</tr>
<tr>
<td>resolution</td>
<td>77</td>
</tr>
<tr>
<td>Marker</td>
<td>243</td>
</tr>
<tr>
<td>mask</td>
<td>56</td>
</tr>
<tr>
<td>Metadata</td>
<td>62</td>
</tr>
<tr>
<td>define dimensions and attributes</td>
<td>64</td>
</tr>
<tr>
<td>missing_value</td>
<td>203</td>
</tr>
<tr>
<td>month_to_annual</td>
<td>49</td>
</tr>
</tbody>
</table>

### N

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td>203</td>
</tr>
<tr>
<td>NCARG_ROOT environment variable</td>
<td>2</td>
</tr>
<tr>
<td>NetCDF</td>
<td>6</td>
</tr>
<tr>
<td>new</td>
<td>22</td>
</tr>
</tbody>
</table>

### O

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>overlay</td>
<td>99</td>
</tr>
<tr>
<td>Overlay Plots</td>
<td>99</td>
</tr>
</tbody>
</table>

### P

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panel Plots</td>
<td>108</td>
</tr>
<tr>
<td>Plot Types</td>
<td>231</td>
</tr>
<tr>
<td>print</td>
<td>25</td>
</tr>
<tr>
<td>print_table</td>
<td>26</td>
</tr>
<tr>
<td>printFileVarSummary</td>
<td>26</td>
</tr>
<tr>
<td>printMinMax</td>
<td>26</td>
</tr>
<tr>
<td>printVarSummary</td>
<td>25</td>
</tr>
<tr>
<td>procedure</td>
<td>60</td>
</tr>
<tr>
<td>Projections</td>
<td>236</td>
</tr>
<tr>
<td>PyNGL</td>
<td>207</td>
</tr>
<tr>
<td>contour plot example</td>
<td>208</td>
</tr>
<tr>
<td>contour plot of unstructured data</td>
<td>215</td>
</tr>
<tr>
<td>contour plot on curvilinear gridded data</td>
<td>213</td>
</tr>
<tr>
<td>polygon plot of unstructured data (ICON)</td>
<td>217</td>
</tr>
<tr>
<td>slice plot example</td>
<td>212</td>
</tr>
<tr>
<td>vector plot example</td>
<td>210</td>
</tr>
<tr>
<td>xy-plot example</td>
<td>207</td>
</tr>
<tr>
<td>PyNIO</td>
<td>207</td>
</tr>
</tbody>
</table>

### Q

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>quit</td>
<td>5</td>
</tr>
</tbody>
</table>

### R

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>5</td>
</tr>
<tr>
<td>regline</td>
<td>51</td>
</tr>
<tr>
<td>Reserved keywords</td>
<td>175</td>
</tr>
<tr>
<td>Resources</td>
<td>26</td>
</tr>
<tr>
<td>Resources</td>
<td>148, 246</td>
</tr>
<tr>
<td>runave_n_wrap</td>
<td>52</td>
</tr>
</tbody>
</table>
S

setfileoption ................................................................. 36
Shapefile Plots .................................................................. 119
sleep .................................................................................. 59
status_exit .......................................................................... 59
str_capitalize ..................................................................... 58
str_fields_count ............................................................... 33, 58
str_get_field ....................................................................... 33, 58
str_left_strip ...................................................................... 58
str_lower .............................................................................. 58
str_right_strip .................................................................... 58
str_split .............................................................................. 58
str_split_csv ....................................................................... 58
str_squeeze ......................................................................... 58
str_strip .............................................................................. 58
str_upper .............................................................................. 58
stringtoDouble ................................................................... 59
stringtoFloat ....................................................................... 59
stringToLong ....................................................................... 59
stringToShort ..................................................................... 59
Syntax characters .............................................................. 8
system ................................................................................ 59
systemfunc .......................................................................... 59

T

Tickmarks .............................................................................. 80, 168, 169
Time series ........................................................................... 81
time_axis_labels .................................................................. 81

U

Unstructured Grids .................................................................. 133

V

Variables ................................................................................ 10
Vector Plots ............................................................................ 88
Viewport ................................................................................ 67
  vpHeightF .......................................................................... 67
  vpWidthF .......................................................................... 67
  vpXF .................................................................................. 67
  vpYF .................................................................................. 67

W

wgt_areaave .......................................................................... 50
Workstation
  height ................................................................................ 201
  width ................................................................................ 201
Wrapit .................................................................................. 195
write_matrix .......................................................................... 37
write_table ............................................................................. 37
XY-Plots