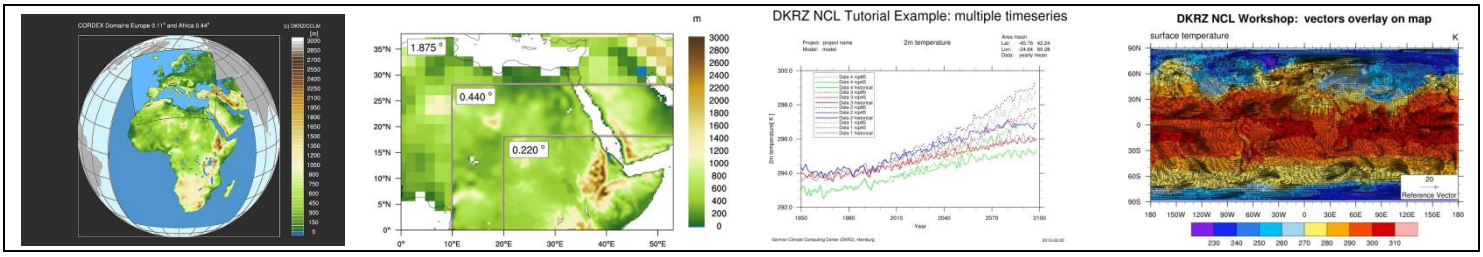
NCL

TUTORIAL

High Quality Graphics with NCL 6.1.2

Karin Meier-Fleischer
Michael Böttinger
DKRZ

Version: 1.0 2013/10/22



Contact: **Karin Meier-Fleischer**

Deutsches Klimarechenzentrum (DKRZ)
Bundesstrasse 45a
D-20146 Hamburg
Germany

Email: meier-fleischer(at)dkrz.de
http://www.dkrz.de/

Thanks to
Mary Haley (UCAR)
and Dennis Shea (NCAR)

The NCAR Command Language (Version 6.1.2) [Software]. (2013).
Boulder, Colorado: UCAR/NCAR/CISL/VETS. http://dx.doi.org/10.5065/D6WD3XH5

http://www.ncl.ucar.edu

Contents

1 Introduction

NCL (**N**CAR **C**ommand **L**anguage) 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/

The NCAR Command Language (Version 6.1.2) [Software]. (2013). Boulder, Colorado: UCAR/NCAR/CISL/VETS. http://dx.doi.org/10.5065/D6WD3XH5

1.1 Documents from NCAR

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

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

Manuals web page: http://www.ncl.ucar.edu/Document/Manuals/

Getting Started: http://www.ncl.ucar.edu/Document/Manuals/Getting_Started/
Reference Manual: http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/

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 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. 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

Email support information:

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

1.3 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/

2 Preliminary Work

The NCARG_ROOT environment variable must be set to the directory containing the NCL software installation and the NCL binary directory. Furthermore, $NCARG_ROOT/bin must be added to the PATH environment variable of the user.

At **DKRZ,** NCL is installed on different platforms in different versions. The environment settings can be loaded using the 'module load' function. After loading the module file of NCL, the user does not need to set any further environment variables, as this is done by the module function.

The function call

```
module apropos ncl
```

will return the name of the NCL environment module file which can be loaded by

```
module load <NCL-name>
```

For example: `module load NCL/6.1.2`

The Tutorial scripts and data files used in this Document can be found on the DKRZ computer in the Tutorial main directory

/work/kv0653/NCL_Tutorial

To make working with the DKRZ Tutorial scripts more convenient, the variable **NCL_TUT** must be set:

For bash or ksh:

```
export NCL_TUT=/work/kv0653/NCL_Tutorial
```

For csh or tcsh:

```
setenv NCL_TUT /work/kv0653/NCL_Tutorial
```

2.1 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-6.1.0.* Then add the bin directory of the NCARG_ROOT path to your PATH environment variable.

For sh:

```
NCARG_ROOT="/opt/local/ncl-6.1.0"
export NCARG_ROOT
PATH=$NCARG_ROOT/bin:$PATH
```

For bash or ksh:

```
export NCARG_ROOT="/opt/local/ncl-6.1.0"
export PATH=$NCARG_ROOT/bin:$PATH
```

For csh and tcsh:

```
setenv NCARG_ROOT "/opt/local/ncl-6.1.0"
setenv PATH "$NCARG_ROOT/bin:$PATH"
```

2.2 Editor Settings

Many different editor plugins or settings are available to support syntax specific highlighting for NCL scripts. For more information about nedit, vim, emacs, kate, netbeans et al., please see:

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

3 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.

3.1 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 record an interactive session:

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

3.2 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:

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

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
```

3.3 Input, Export and Graphics Output File Formats

NCL can import the following data file formats:

- ✓ netCDF3, netCDF4
- ✓ HDF4-SDS, HDF4-EOS HDF5, HDF5-EOS
- ✓ GRIB 1, GRIB 2
- ✓ CCM History Tape
- ✓ shapefiles

- ✓ binary
- ✓ ASCII

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

- ✓ netCDF3, 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
- ✓ NCGM (an older format that is generally not recommended)
- ✓ X11 (graphics output only to a X11 window)

3.3.1 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.

3.3.2 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 important 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".

3.4 Data Operators: CDO’s and NCO’s

It might be necessary (and faster) to extract some variables or time steps of your data before you import it in NCL, especially when you work with large file sizes (> 2 GB). This can be done by using the Climate Data Operators (CDO) developed by the Max-Planck-Institute for

Meteorology. CDO is a collection of command line operators that manipulate and analyse climate data sets:

https://code.zmaw.de/embedded/cdo/1.5.9/cdo.html

On DKRZ's systems, the module **cdo** is aleady installed, but first needs to be loaded before running it:

```
module load cdo
```

Its advantage is running on the command line and the capability to combine operators by piping the output of one operator to the next operator.

```
cdo [options] operator_1 [operator_2 [operator_n]]
```

Manipulation of netCDF data including the change of attributes and dimension names can be done with the NetCDF Operators (NCO):

http://nco.sourceforge.net/

These operators are also already installed on DKRZ's systems. The module **nco** first needs to be loaded before running it:

```
module load nco
```

3.5 Tutorial Datasets

The example data sets are the same used for our AvizoGreen Hands-On Tutorials and stored on DKRZ's GPFS file system (/work). Users do not have write perimissions to overwrite these files or to store their own data there; they have read permissions only.

The example data files are available in the directory */work/kv0653/NCL_Tutorial* on blizzard, wizard, Passat and the halos:

```
NCL_Tutorial/ECHAM5_OM_A1B_2001_2D.nc
NCL_Tutorial/ECHAM5_OM_A1B_2001_3D.nc
NCL_Tutorial/EH_OM_A1B_3Dvectors.nc
NCL_Tutorial/MITgcm_2007.nc
NCL_Tutorial/MITgcm_2007s.nc
NCL_Tutorial/MITgcm_current_1960.nc
```

The most important rule in data processing is to pay attention to 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:

ncl_filedump <*your_filename*>

4 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 can be found in the chapters which may readily be executed.

4.1 Syntax Characters

Commonly used syntax characters include:

=	assignment syntax
;	begins a comment
@	create or reference attributes
!	create or reference named dimensions
&	create or reference a coordinate variable
$	enclose strings when importing or exporting variables via *addfile*
{…}	used for coordinate subscripting
[…]	subscript variables of type *list*
(/…/)	constructs an array
:	used in array syntax
|	used as a separator for named dimensions
****	continue statement for spanning multiple lines
::	used as separator when calling external codes
:=	reassignment operator (new since version 6.1.1)
→	used for inputting/outputting supported data formats

4.2 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

4.2.1 Algebraic Operators

+	Addition
-	Substraction / Negation
*****	Multiplication
/	Division
%	Modulus
>	Greater than

<	Less than
^	Exponentiation
#	Matrix multiplication

Note:

- Use parentheses to circumvent precedence rules:
  - `x = (2 + 3) * 3` → `15`
  - `print(- (3+2)^2)` → `25`
  - `print(- ((3+2)^2))` → `-25`
- The + sign is an overloaded operator, which means that it has two different applications:
  - Addition: `x = 2.3 + 5.8` → `x = 8.1`
  - String concatenation: `"Value: "+12.7` → `"Value: 12.7"`
- The – sign can also be used in two different ways:
  - negation has the highest precedence:
    - `x = - 3^2` is equal to `x = (-3)^2` → `9`
  - treated as a minus:
    - `x = 8 - 3^2` → `-1`

4.2.2 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 evaluated left to right. It would be a good idea to 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
```

4.2.3 Array Expressions

For arrays, the arithmetic operators (add, multiply, divide, compare and so forth) can be applied just as well as for scalars. *See Arrays in chapter 4.5.*

4.2.4 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 8.5.*

4.3 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

Non-numeric data types:

- string
- character
- graphic
- file
- logical
- list

Enumeric 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)

4.4 Variables

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

To assign a variable:

```
x     = 0.12                                  ; float
y     = -4                                    ; integer
z     = 20.d                                  ; double
title = "This is the title string"            ; string
a     = True                                  ; logical

a     = (/1, 2, 3, 4/)                        ; integer array
```

```
b     = (/1, 2.0, 3.0, 4./)                ; float array
c     = (/1., 2, 3., 4.0/) * 1d5           ; double array
d     = (/"hello", "world"/)               ; string array
e     = (/True, False, False, True/)       ; logical array
f     = (/ (/1,2/), (/3,4/), (/5,6/) /)    ; 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.

4.4.1 Attributes

The attributes of a variable, dimension or coordinate variable contain information about the variable itself. Attributes can be *units, long_name, standard_name, coordinates, scale_factor, add_offset, valid_min, valid_max, axis* and many more. Here are some examples:

```
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
```

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

4.4.2 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 data referenced, and the leftmost dimension is numbered 0. This also applies to arrays.

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

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

4.4.3 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:

```
lon_pts@units    = "degrees_east"
lat_pts@units    = "degrees_north"
```

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.

```
temp&lon    = lon_pts
temp&lat    = lat_pts
```

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

4.5 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 = (/ 1, 2, 3, 4 /)          4 elements; index 0-3
b = (/ 0, 1, 1, 0 /)          4 elements; index 0-3

c = a + b          →    c = (/ 1, 3, 4 , 4 /)
c = a - b          →    c = (/ 1, 1, 2 , 4 /)
c = a * b          →    c = (/ 0, 2, 3 , 0 /)
c = a /(b+0.1)     →    c = (/10,1.818182,2.727273,40 /)
```

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
right dimension index 2 with 4 elements (varying fastest)

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(dimsizes(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.

4.5.1 Standard Subscripting

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, /)`

```
new_1 = a              → new_1 contains 0,1,2,3,4,5,6,7,8,9
new_2 = a(3)           → new_2 contains 3
new_3 = a(1:4)         → new_3 contains 1,2,3,4
new_4 = a(0:9:3)       → new_4 contains 0,3,6,9
new_5 = a(:5)          → new_5 contains 0,1,2,3,4,5
new_6 = a(:7)          → new_6 contains 7,8,9
new_7 = a(1:6:-1)      → new_7 contains 6,5,4,3,2,1
new_8 = a(8:4)         → new_8 contains 8,7,6,5,4
                           note: no need to set the stride to -1
new_9 = 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)`

```
new_T = T(0:11:3, :19, :)   → defines a 4 x 20 x 120 array
```

4.5.2 Named Subscripting

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_reord1 = pres(lon|:, lat|:)
                                →    p_reord1(40,21) ~ p_reord1(lon,lat)

p_reord2 = pres(lon|19:39, lat|0:9)
                                →    define an array 20x10 p_reord2(20,10)
```

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

4.5.3 Coordinate Subscripting

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   = "lat"                  →    name the dimension
m&lat = m                      →    associate the array
mw    = m({ -5.0 : 25.0 : 2})  →    contains the values –
                                    5.0,15.0,25.0
```

4.6 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.

4.6.1 Block

Blocks can be used to bundle many statements into a group of statements. The statements between the ***begin*** and ***end*** statement will be executed.

```
begin
      statement 1
```

```
      statement 2
      ……
end
```

4.6.2 If - Statement

```
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:

```
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:

```
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
```

➔ (0) if-statement 1 else

4.6.3 Loops

Loops are useful but not efficient and should be minimally used in any interpreted language. To be more efficient, it would sometimes be better to write a wrapper in Fortran or C and load it into your NCL script (see chapter 11).

Loop over n-times:

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

The stride value is optional.

Loop while a logical expression is True:

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

Special statements:

break	jump to first statement after "*end do*"
continue	proceed directly to the next iteration

4.6.4 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 4.4 and 4.5 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 must 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
```

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.

4.7 Metadata and Attributes

Metadata is the information associated with a variable or file that describes the data. The metadata of a variable can be attributes like *units*, *long_name*, *_FillValue,* and for a file it can be *creation_date* and *history*.

To assign or access variable attributes, use the @ character:

```
var = (/ 277.0, 281.5, 278.7, 291.0 /)
```

```
var@units = "K"
var@long_name = "Temperature"
var@_FillValue = -9999

title = var@long_name
```

To verify whether an attribute of a variable exists, use the function *isatt:*

```
if(isatt(slp, "units")) then …… end if
```

To get the attributes of a variable of a file, use the function *getfilevaratts*:

file_atts = getfilevaratts (fin, "slp")

4.8 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

printMinMax(*data_variable,0*)
prints the minimum and maximum value of a variable

printFileVarSummary(*file,varname*)
prints a summary of a file variable's information

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

4.9 Reserved Keywords

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

begin	break	byte	character
continue	create	defaultapp	do
double	else	end	external
False	file	float	function
getvalues	graphic	if	integer
load	local	logical	long

new	noparent	numeric	procedure
quit	quit	QUIT	record
return	setvalues	short	string
then	True	undef	while

All built-in function and procedure names are also reserved keywords.

5 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 facilitates data processing and the creation of generic functions.

5.1 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.

```
f = addfile(filename, status)

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

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

```
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 /)
```

5.2 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                   ; default is 'cat'
   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

   ```
   ListSetType(f, "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:

```
varnames = getfilevarnames(f[0])
```

4. It is only possible to write to an individual file of the *file list* `f:`

```
f = addfiles(files,"w")
```

5.3 Assign a variable from a File Variable

```
f = addfile("dataT.nc", "w")
T = f->T                    → set T to file variable T
```

or

```
T = f->T(0,:,:)             → set T to file variable T,
                              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 = "RCP85 MPI-ESM tas"
var  = f->$tnam$
lon  = x&"lon-1"
```

6 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`'.

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

Variable: f
Type: file
filename:          ECHAM5_OM_A1B_2001_2D
path:     ECHAM5_OM_A1B_2001_2D.nc
   file global attributes:
     CDI : Climate Data Interface version 1.0.8
     Conventions : CF-1.0
     history : Wed Feb 27 10:05:05 2008: cdo merge ……
     CDO : Climate Data Operators version 1.0.9 available from http://www.mpimet.mpg.de/cdo
     source : ECHAM5.2
     institution : Max-Planck-Institute for Meteorology
   dimensions:
     lon = 192
     lat = 96
     time = 1460  // unlimited
   variables:
     double lon ( lon )
       long_name :          longitude
       units :        degrees_east
       standard_name :      longitude

     double lat ( lat )
       long_name :          latitude
       units :        degrees_north
       standard_name :      latitude

     double time ( time )
       units :        hours since 2001-01-01 00:00

     float tsurf ( time, lat, lon )
       long_name :          surface temperature
       units :        K
       code :         169
       table :        128
       grid_type :    gaussian
……
     float slp ( time, lat, lon )
       long_name :          mean sea level pressure
       units :        Pa
       code :         151
       table :        128
       grid_type :    gaussian
```

- **ncl_convert2nc** Converts GRIB1, GRIB2, shapefile, HDF, or HDF-EOS files to NetCDF files
- **ng4ex** a 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

7 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

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)

7.1 Using NCL

1) Compute yearly means from monthly data

To calculate the yearly (annual) mean, use the function *month_to_annual* in which can be found in the contributed.ncl library (don't forget to load it):

```
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)
```

2) Compute the time average at each grid point

To compute the area mean without weights of the variable tas, use the functions *dim_avg_n* or *dim_avg_n_Wrap*:

Example code: assume tas(time,lat,lon)

Without transferring the metadata -

```
tasAvg = dim_avg_n(dim_avg_n(tas,2),1)
```

➔ Results 1D array of area mean values
➔ First average over lon (= index 2) and then lat (= index 1)

To preserve the metadata of tas, use the _Wrap version of this function

```
tasAvg = dim_avg_n_Wrap(dim_avg_n_Wrap(tas,2),1)
    ;-- 1D array of area mean values with the attributes
    ;   and coordinate arrays of tas
```

3) Compute the standard deviation of a dimension using *dim_stddev*

```
ret_var = dim_stddev(var)
```

var:	variable of numerical type and any dimension
ret_var:	variable of same type as var, dimension rank is reduced by one

To specify which dimension should be used to compute the standard deviation, use *dim_stddev_n*. If you want to retain the metadata, use *dim_stddev_Wrap*. Use the function *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:

```
tasStdT = dim_stddev(tas(lat|:,lon|:,time|:)
                                        ;--  tasStdLon(194,201),
                                        ;--  no metadata
tasStdT = dim_stddev_Wrap(tas(lat|:,lon|:,time|:)
                                        ;--  tasStdLon(194,201),
                                        ;--  with metadata
tasStdT = dim_stddev_n_Wrap(tas,0)      ;--  tasStdLon(194,201),
                                        ;--  with metadata
```

4) Compute the area average

```
rad      = 4.0*atan(1.0)/180.0
weights  = cos(lat*rad)
area_avg = wgt_areaave(var,weights,1.0, 1)
```

7.2 Using CDOs

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').

1) Compute annual means from monthly data

   To calculate the yearly mean use the operator *yearmean*:

   ```
   cdo –r –f nc yearmean <input file> <output file>
   ```

2) 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*:

   ```
   cdo –r –f nc timmean <input file> <output file>
   ```

3) To calculate the temporal standard deviation at each grid point, use *timstd*:

   ```
   cdo –r –f nc timstd <input file> <output file>
   ```

4) Compute the area average

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

   ```
   cdo –r –f nc fldmean <input file> <output file>
   ```

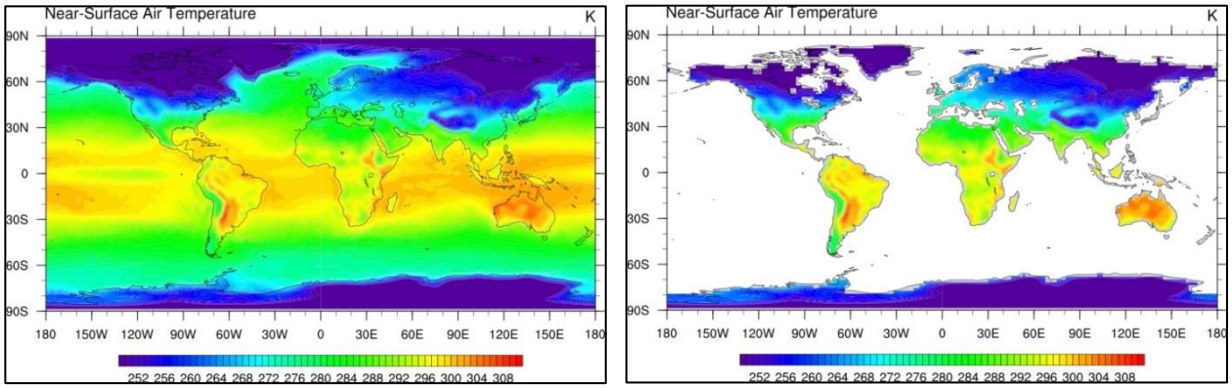
Tip: For large data sets it would be more efficient to do the computations first.

8 Advanced NCL features

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

8.1 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.



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  = var            → xLand has the same metadata as var
xOcean = var            → xOcean has same metadata as var
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)

```
res@mpAreaMaskingOn       =  1
res@mpMaskAreaSpecifiers  = "France"
res@mpFillAreaSpecifiers  = (/ "water", "land" /)
res@mpSpecifiedFillColors = (/7,2/)
```

- change the order of the plotting elements

  → see NCL example mask_2.ncl

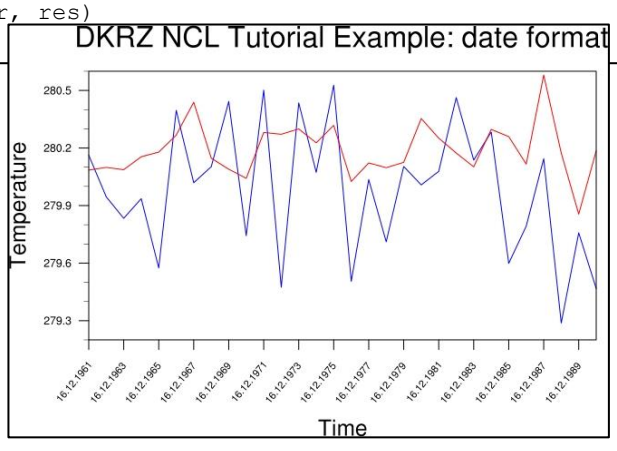
8.2 Date Conversion

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

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

```
….
  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         = tointeger(utc_date(:,0))
   month        = tointeger(utc_date(:,1))
   day          = tointeger(utc_date(:,2))
   hour         = tointeger(utc_date(:,3))
   minute       = tointeger(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@tmXBLabels            =  labels            ;- labels on axis ticks
….
  plot = gsn_csm_xy(wks, var&time, var, res)
….
```



8.3 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_capital(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()                     ; the quote character
print(qc + strlist + qc)
```

```
(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.

```
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:
      Number Of Attributes: 1
        _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**:

```
str = "20130101000000 53.33 10.0 278.32 t2m"

val = tofloat(str_get_field(str,4," "))
                    → val = 278.32  of type float

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

8.4 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:

```
system("rm -f tmp.asc")
system("export NCARG_COLORMAPS=$HOME/NCL/Colors")
```

- To execute a shell command and return the output of the call:

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

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

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

- Get the content of a shell environment variable:

```
ret = getenv("SHELL")
→
        Variable: ret
        Type: string
        Total Size: 8 bytes
                            1 values
        Number of Dimensions: 1
        Dimensions and sizes:   [1]
        Coordinates:
        (0)     /usr/bin/tcsh
```

8.5 User-defined Functions and Procedures

Unlike a function, a procedure is executed without a return value. 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:

```
;-- 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/my_lib/funs_and_procs.ncl"                 User defined lib
```

8.5.1 Procedures

Generally, procedures are used to perform a task (eg: draw a plot).

General structure:

```
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("wallClockElapseTime_ger")
procedure wallClockElapseTime_ger(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  "/your_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
```

8.5.2 Functions

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

General structure:

```
undef ("function_name")                      ;-- optional
procedure function_name(declaration_list)
local local_variables                        ;-- optional
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()
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
```

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
```

8.6 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)
→
      Variable: m
      Type: float
      Total Size: 155976 bytes
                  38994 values
      Number of Dimensions: 2
      Dimensions and sizes:   [201] x [194]
      Coordinates:

m@standard_name = "magnitude_of_wind_velocity"
m@long_name     = "magnitude of wind velocity"
m@units         = "m s-1"

→
      Variable: m
      Type: float
      Total Size: 155976 bytes
                  38994 values
      Number of Dimensions: 2
      Dimensions and sizes:   [201] x [194]
      Coordinates:
      Number Of Attributes: 3
```

```
        units :             m s-1
        long_name :         magnitude of wind velocity
        standard_name :     magnitude_of_wind_velocity
```

- Overwrite an attribute

```
m@long_name = "magnitude (sqrt(u^2+v^2)"
```

→

```
Variable: mm
Type: float
Total Size: 155976 bytes
       38994 values
Number of Dimensions: 2
Dimensions and sizes:   [201] x [194]
Coordinates:
Number Of Attributes: 3
  units :             m s-1
  long_name :   magnitude (sqrt(u^2+v^2)
  standard_name :     magnitude_of_wind_velocity
```

- Delete an attribute

```
delete(m@long_name)
delete(m@units)
```

→

```
Variable: mm
Type: float
Total Size: 155976 bytes
       38994 values
Number of Dimensions: 2
Dimensions and sizes:   [201] x [194]
Coordinates:
Number Of Attributes: 3
  standard_name :     magnitude_of_wind_velocity
```

Example:

```
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:

```
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:

```
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)"
```

9 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 colortable had been changed. This may cause differently colored results for the examples compared with older versions of NCL.

See also: http://www.ncl.ucar.edu/Document/Graphics/Resources/list_alpha_res.shtml

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

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/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.

9.1 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_Tutorial/scripts: TUT_template.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

end
```

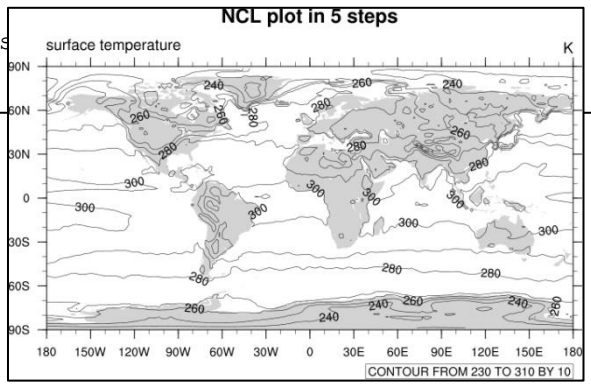
The first two lines tell NCL to load the graphic libraries *gsn_code.ncl* and *gsn_csm.ncl*. These library files contain high level procedures and functions for contours, vectors, legends, labelbars and so on, which are not yet included directly in NCL. Additional library files like *contributed.ncl* 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.

In the next NCL release (6.2) these libraries (gsn_code.ncl, gsn_csm.ncl and contributed.ncl) will no longer be required because it will happen automatically.

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

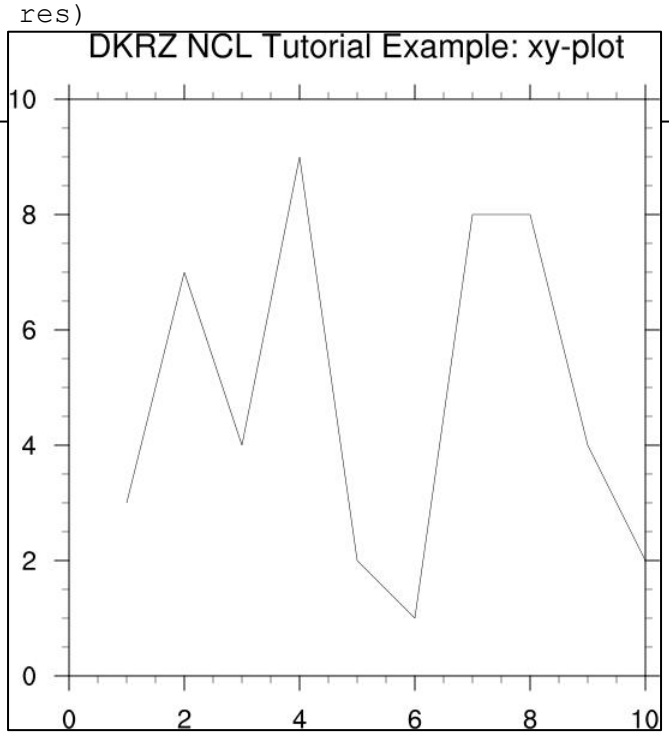
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

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
end
```



9.2 XY-Plots

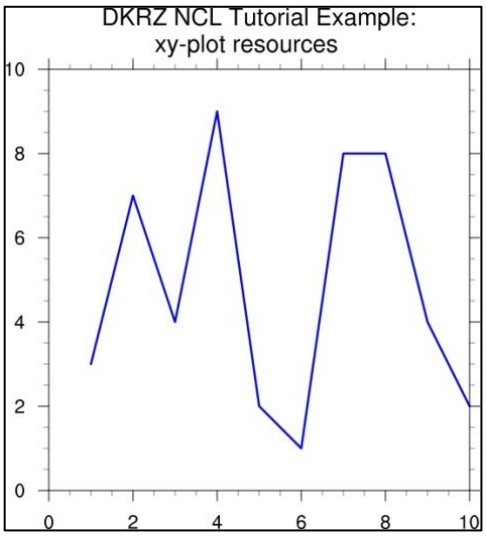
Simple xy-plot example: NCL_Tutorial/scripts/TUT_xy_plot.ncl

```
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("x11","xy_plot")

   res                          =  True
   res@tiMainString             = "DKRZ NCL Tutorial Example: xy-plot"

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

 end
```



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

```
   res@xyLineColor          =  "blue"
   res@xyLineThicknessF     =  5
```

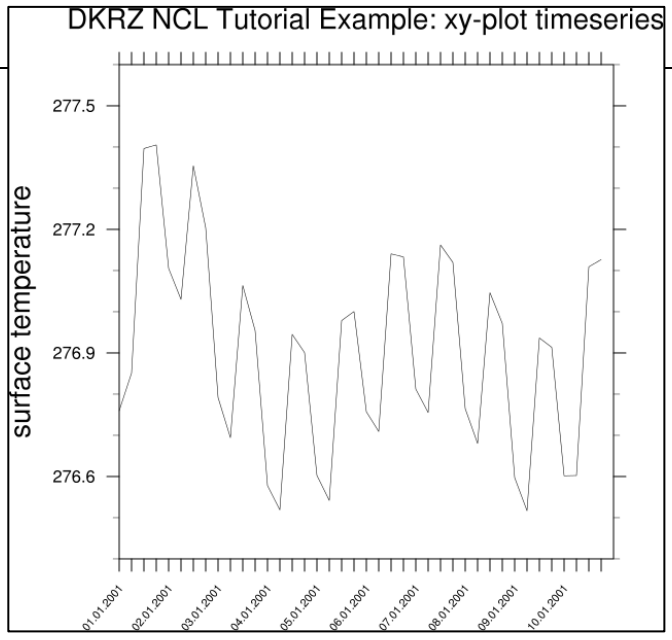


9.3 Time-series

A time-series plot is a little bit tricky because the time data format is commonly an integer representing the values of "seconds since…" or "days since …" for instance. To convert these integer values to a normal date format, NCL provides a bunch of "calendar" functions. See also section 8.2 and the special procedure "time_axis_labels".

Simple contour example: NCL_Tutorial/scripts/TUT_xy_plot_timeseries.ncl

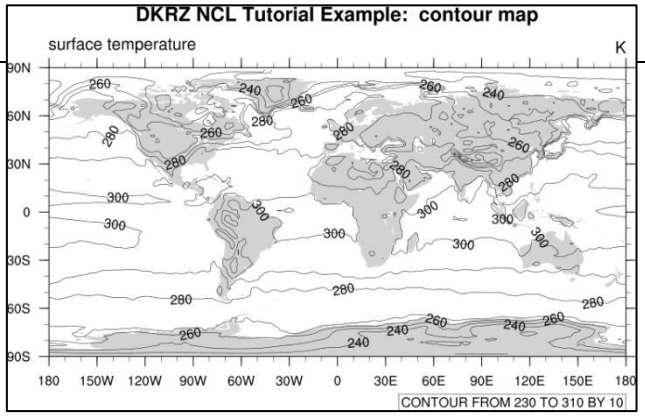
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/contrib/time_axis_labels.ncl"

begin
   f = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc", "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 = "DKRZ NCL Tutorial 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
```



9.4 Contours

Simple contour example: NCL_Tutorial/scripts/TUT_contour_map.ncl

```
;-- load pre-defined functions and procedures
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
   file1 = "$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc"

;-- read the data
   f    =  addfile(file1,"r")
   var  =  f->tsurf(0,:,:)

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

;-- set plot resources
   res                  =  True                    ; plot mods desired
   res@gsnMaximize      =  True                    ; maxmize plot in frame
   res@tiMainString     = "DKRZ NCL Tutorial Example: contour map"

;-- 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:

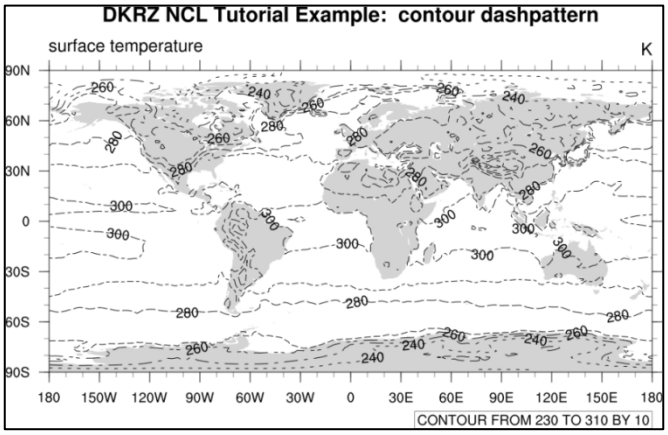
Insert

```
res@cnLineDashPattern      =  1      ; use dash pattern 1
```

or

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

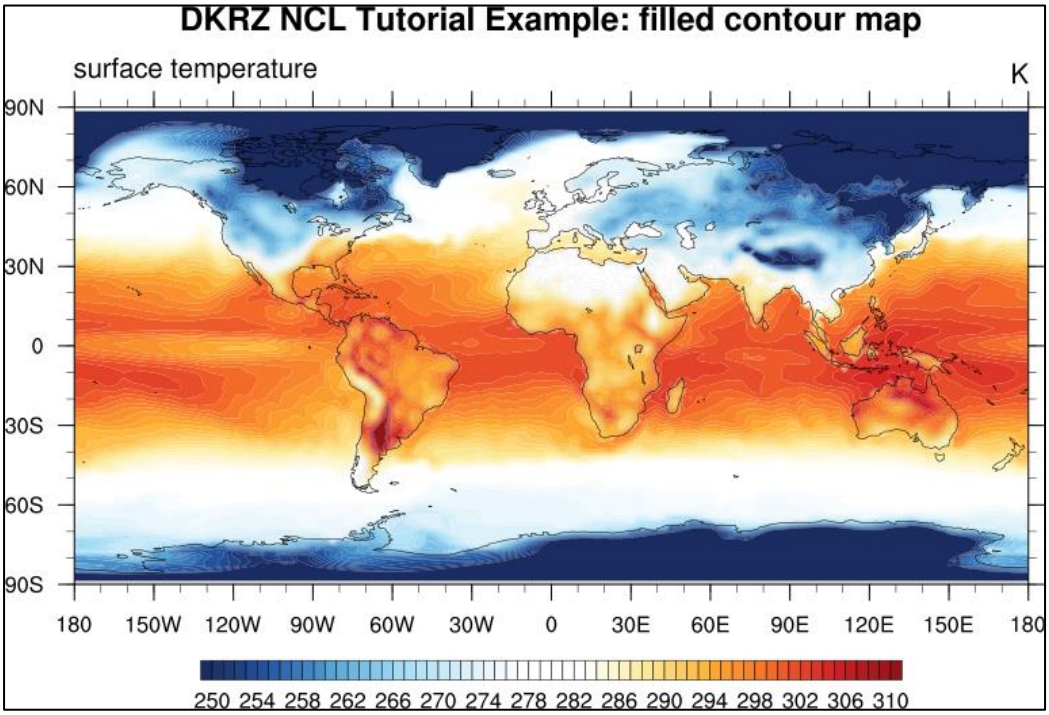
A table of all available dash Patterns can be found in the *Appendix C - Dash Pattern*.



9.4.1 Filled Contours

Simple filled contour example: NCL_Tutorial/scripts/TUT_contour_filled_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
  file1   =  addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
  var     =  file1->tsurf(0,:,:)

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

;-- set resources
  res                          =  True
  res@gsnMaximize              =  True

  res@cnFillOn                 =  True            ;-- turn off contour fill
  res@cnLinesOn                =  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          =  1               ;-- contour level spacing

  res@lbLabelStride            =  4
  res@lbBoxMinorExtentF        =  0.15            ;-- decrease the height of the
                                                  ;-- labelbar
  res@tiMainString             = "DKRZ NCL Tutorial Example: filled contour map"
  res@tiMainFontHeightF        =  0.02

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

end
```



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:

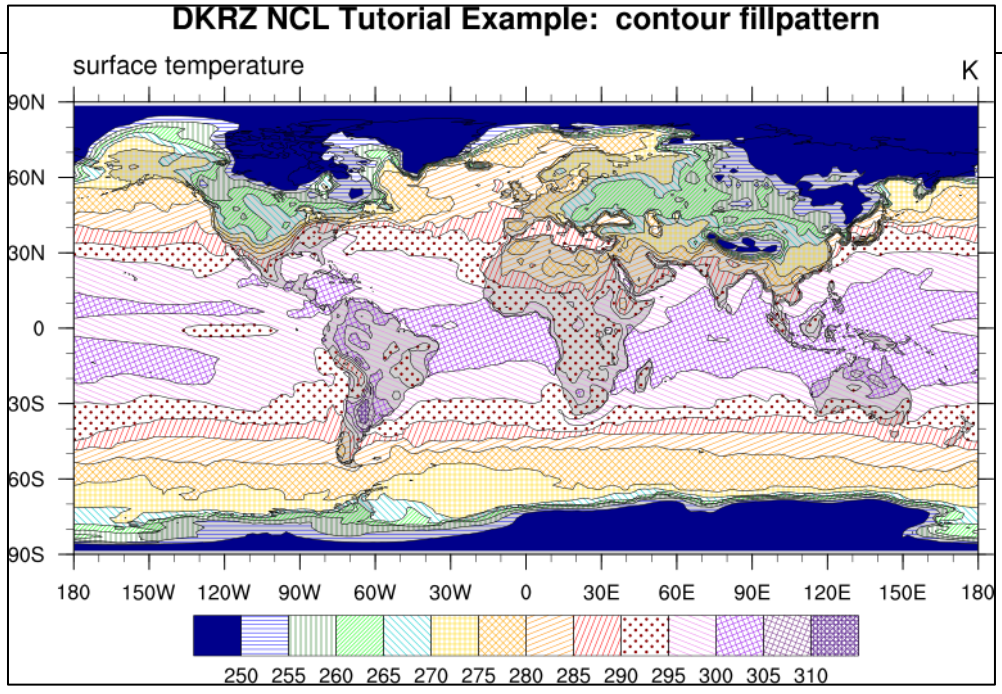
```
res@cnLevelSelectionMode  = "ExplicitLevels"
res@cnLevels              =  (/250,255,270,275,280,300,310/)
```

9.4.2 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: NCL_Tutorial/scripts/TUT_contour_fillpattern.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
  file1 = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
  var   = file1->tsurf(0,:,:)

;---- define the workstation (plot output type and name)
  wks = gsn_open_wks("X11","TUT_contour_fillpattern")

;---- set resources
  res                         =  True
  res@gsnMaximize             =  True
  res@tiMainString            = "DKRZ NCL Tutorial Example:  contour fillpattern"
  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,.5,1.,.5,.5,.5,.4/)

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

end
```



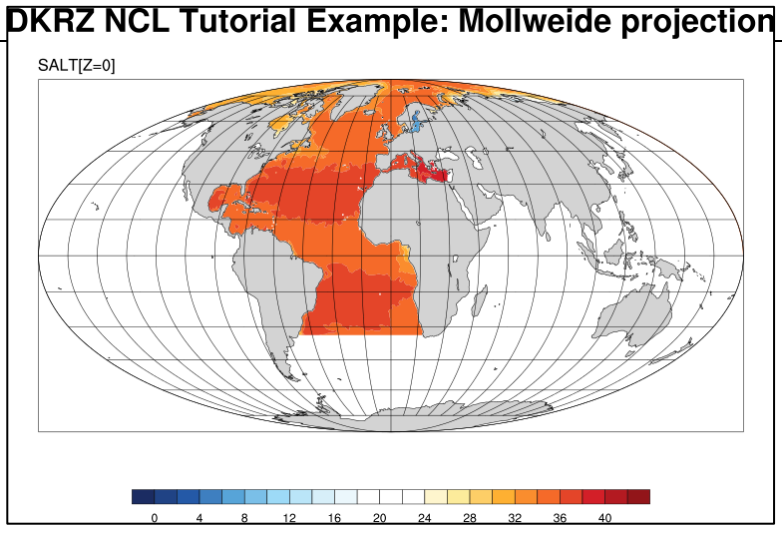
9.5 Maps

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.

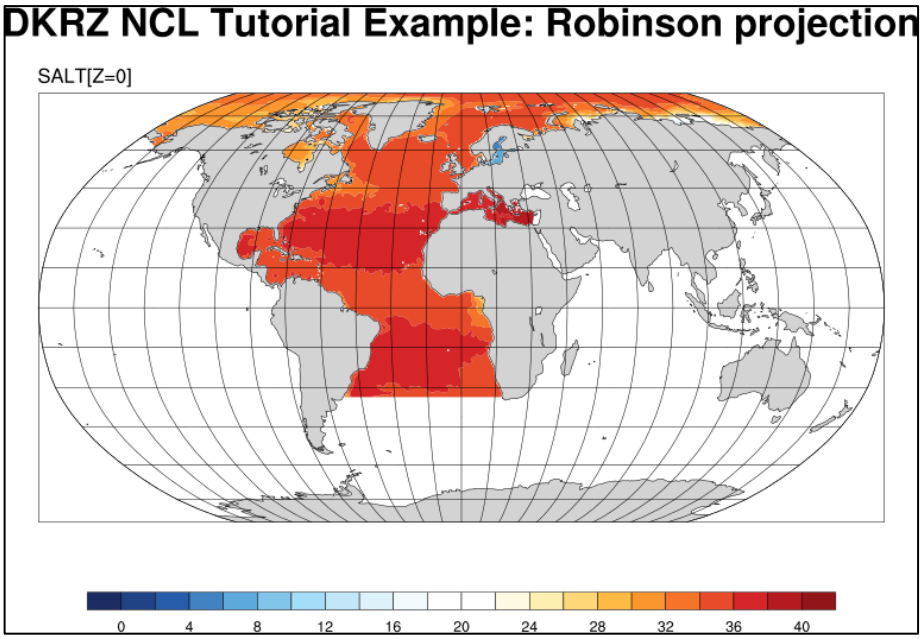
9.5.1 Simple Map

Simple filled contour example using Mollweide projection:
NCL_Tutorial/scripts/TUT_projections_mollweide.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
  file1 = addfile("$NCL_TUT/data/MITgcm_2007s-t1.nc","r")
  var   = file1->SSS(0,0,:,:)

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

;-- set resources
  res                         =  True
  res@gsnMaximize             =  True
  res@lbBoxMinorExtentF       =  0.15     ;-- decrease the height of the labelbar
  res@cnFillOn                =  True           ;-- turn off contour fill
  res@cnLinesOn               =  False          ;-- turn off contour lines
  res@cnLineLabelsOn          =  False          ;-- turn off line labels
  res@cnLevelSelectionMode    = "ManualLevels"  ;-- set contour levels manually
  res@cnMinLevelValF          =    0.           ;-- minimum contour level
  res@cnMaxLevelValF          =   40.           ;-- maximum contour level
  res@cnLevelSpacingF         =    2            ;-- contour level spacing

  res@mpProjection            = "Mollweide"     ;-- change projection
  res@mpGridAndLimbOn         =  True           ;-- plot grid lines

  res@tiMainString            = "DKRZ NCL Tutorial Example: Mollweide projection"
  res@tiMainFontHeightF       =  0.02

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

end
```



To change the map projection from "Mollweide" to "Robinson", just change the *res@mpProjection* setting to "Robinson".



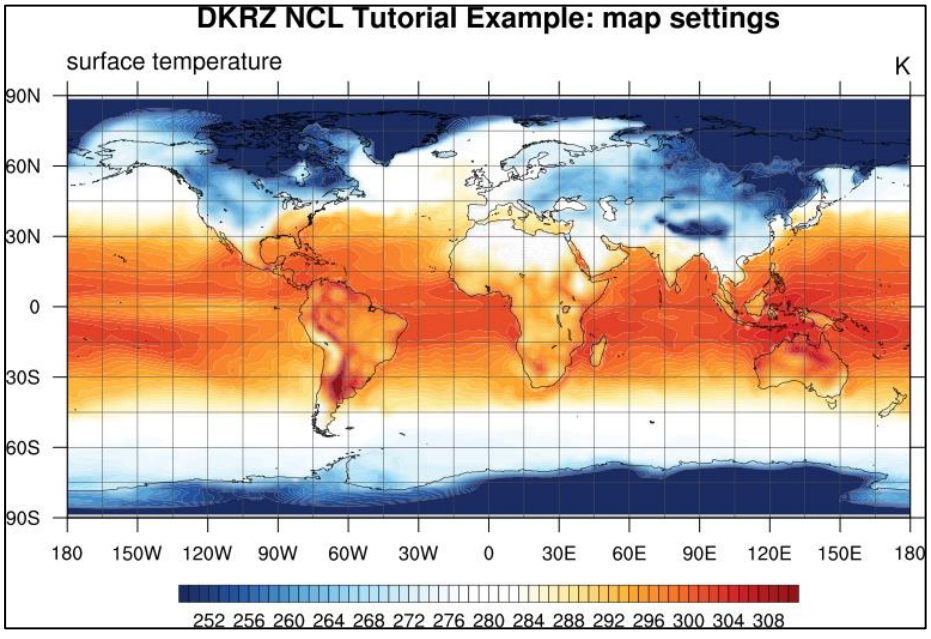
The default projection is "CylindricalEquidistant". To change back to the default projection, uncomment the *res@mpProjection* line or set it to "CylindricalEquidistant".

To change the color of the grid lines from the default to grey, do:

```
res@mpGridLineColor  = "grey30"       ;-- set grid line color
```

To switch off drawing the minor tickmarks:

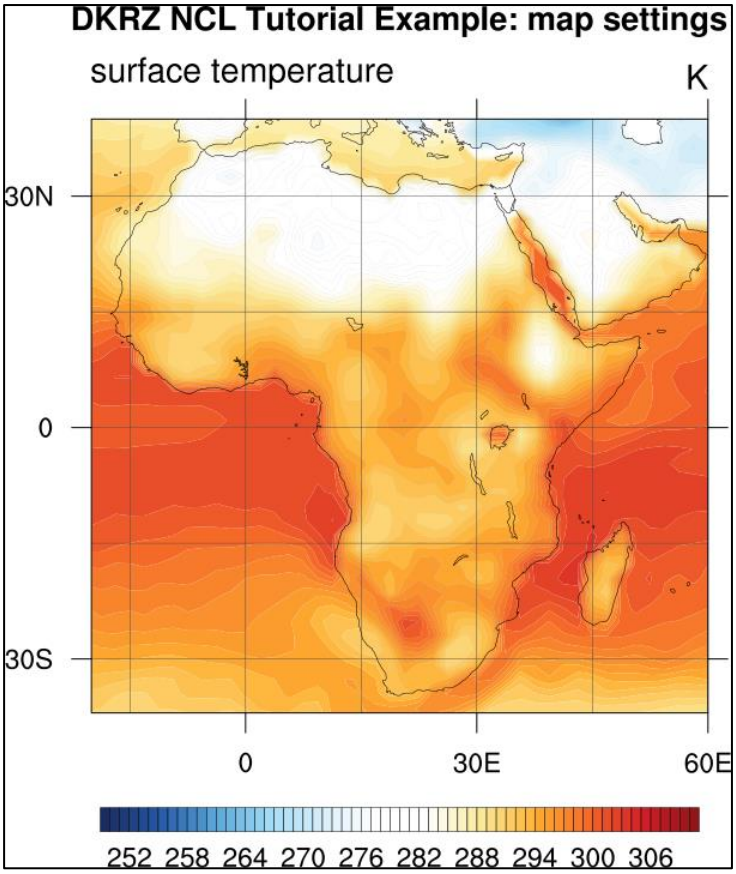
```
res@tmXBMinorOn      =  False         ;-- no minor x-tickmarks
res@tmYLMinorOn      =  False         ;-- no minor y-tickmarks
```



9.5.2 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:

```
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
```



9.5.3 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"`

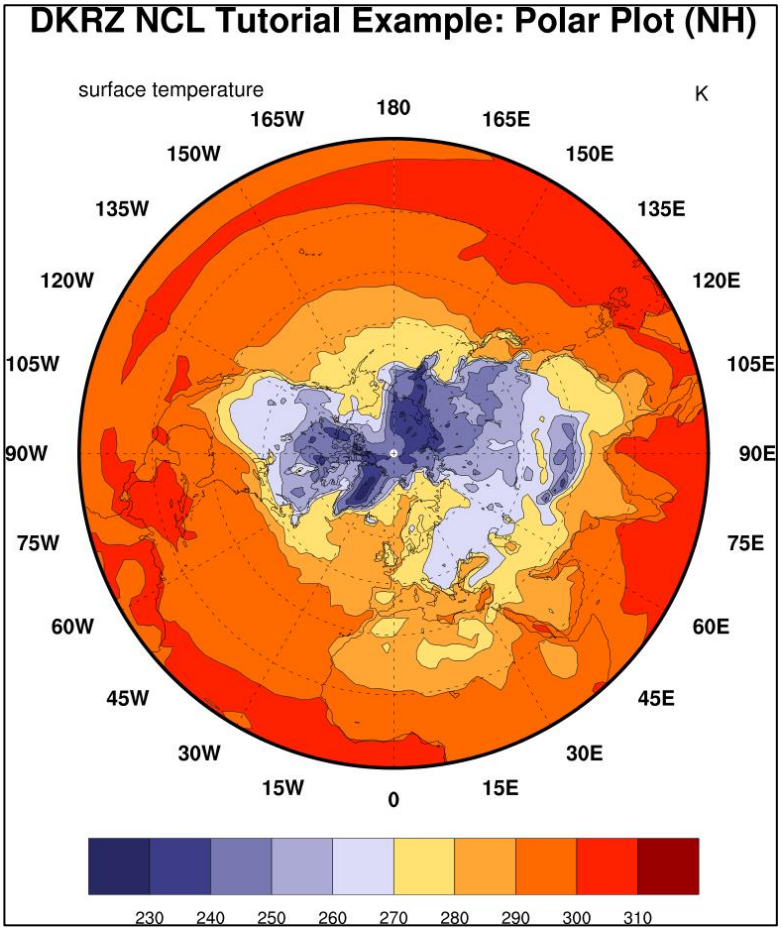
A procedure *polar_map_circle* is included to draw a thicker line around the map.

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 : NCL_Tutorial/scripts/TUT_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
;-- plot a circle around the polar map using width wsize and color col
;----------------------------------------------------------------------
undef("polar_map_circle")
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

;----------------------------------------------------------
;-- main program
;----------------------------------------------------------
begin

  a = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
```

```
  u = a->tsurf(0,:,:)

  wks = gsn_open_wks("X11" ,"polar_settings")
  gsn_define_colormap(wks,"NCL_BYR-03")

  res                             =  True
  res@gsnDraw                     =  False
  res@gsnFrame                    =  False
  res@gsnPolar                    = "NH"
  res@gsnPolarLabelSpacing        =  15
  res@gsnPolarLabelDistance       =  1.1 ; default is 1.04
  res@gsnPolarLabelFontHeightF    =  0.015
  res@gsnPolarLabelFont           = "helvetica-bold"
  res@gsnSpreadColorStart         =  14
  res@gsnSpreadColorEnd           =  -8

  res@cnFillOn                    =  True

  res@tiMainString           = "DKRZ NCL Tutorial Example: Polar Plot (NH)"

  plot = gsn_csm_contour_map_polar(wks,u,res)
                                          ;-- from contributed.ncl library
  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)

  frame(wks)

end
```



9.6 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.

http://www.ncl.ucar.edu/Document/Graphics/rangs.shtml

Low resolution: `res@mpDataBaseVersion = "LowRes"  ;-- the default`

Medium resolution: `res@mpDataBaseVersion = "MediumRes"`

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

High resolution:
```
res@mpDataBaseVersion = "HighRes"
res@mpDataResolution  = "<resolution>"
```

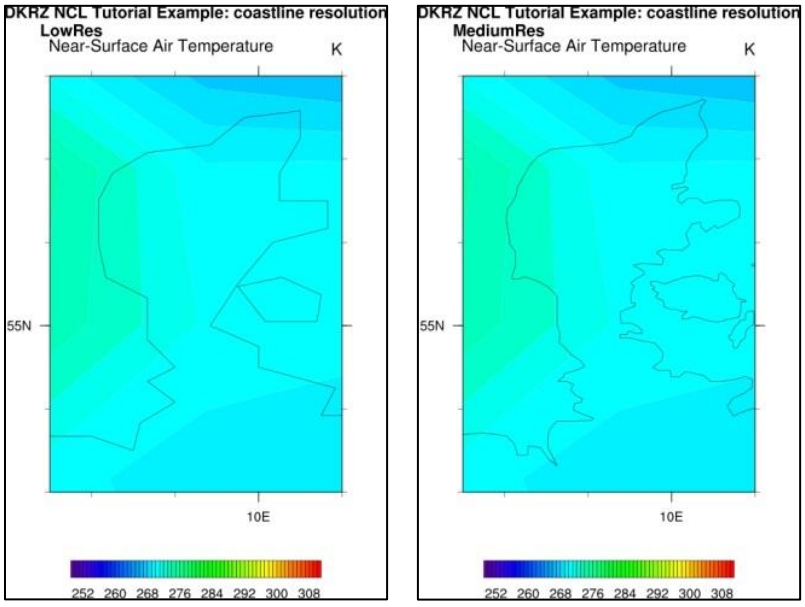
resolution =

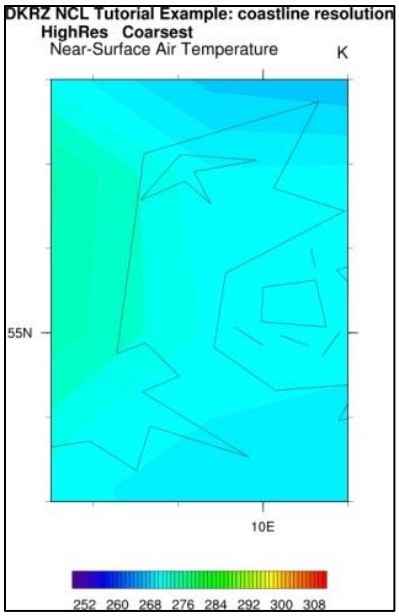
```
Unspecified          (default)
Coarsest
Coarse
Medium
Fine
Finest
```

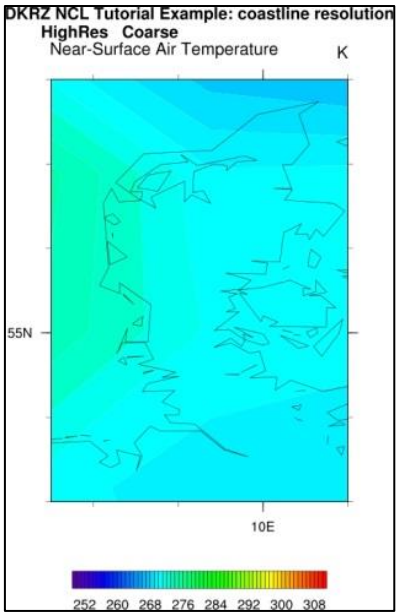
If “Unspecified” (default), the resolution is automatically set depending on the scale and size of the chosen region. A coarse resolution would be chosen for a map with a smaller scale, and a fine resolution for a map with a large scale.

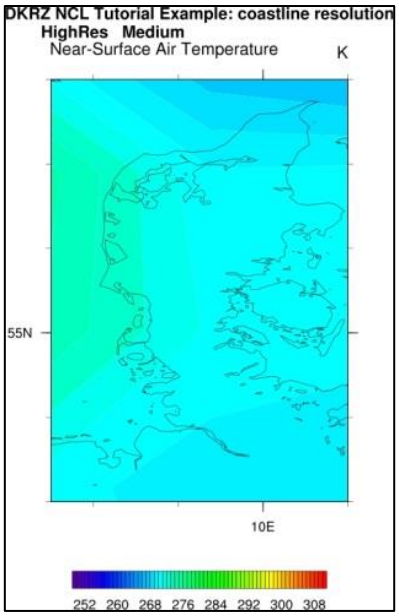
You can also turn off the NCL map outlines and draw your own map outlines read in from a shapefile. Shapefiles can be downloaded for free, and are provided in many different resolutions. See section 9.13.

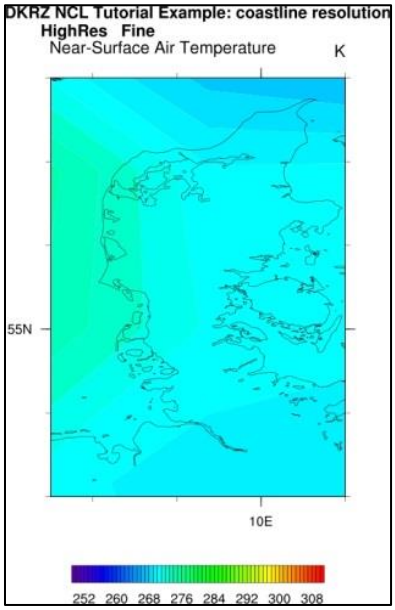
Map resolution example:
NCL_Tutorial/scripts/TUT_map_resolution.ncl

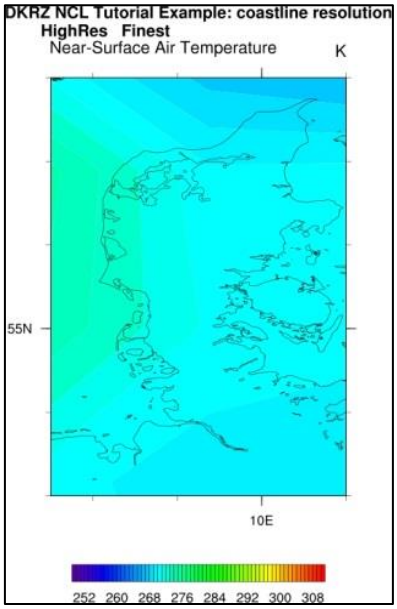



DKRZ NCL Tutorial Example: coastline resolution
HighRes Coarsest
Near-Surface Air Temperature
K
55N
10E
252 260 268 276 284 292 300 308


DKRZ NCL Tutorial Example: coastline resolution
HighRes Coarse
Near-Surface Air Temperature
K
55N
10E
252 260 268 276 284 292 300 308


DKRZ NCL Tutorial Example: coastline resolution
HighRes Medium
Near-Surface Air Temperature
K
55N
10E
252 260 268 276 284 292 300 308


DKRZ NCL Tutorial Example: coastline resolution
HighRes Fine
Near-Surface Air Temperature
K
55N
10E
252 260 268 276 284 292 300 308

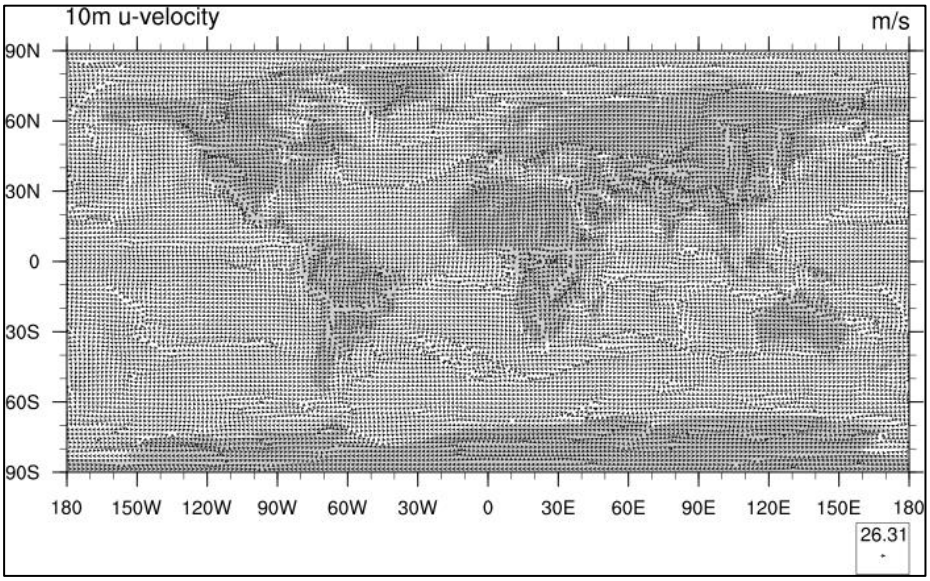

DKRZ NCL Tutorial Example: coastline resolution
HighRes Finest
Near-Surface Air Temperature
K
55N
10E
252 260 268 276 284 292 300 308

9.7 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: NCL_Tutorial/scripts/TUT_vector_default.ncl

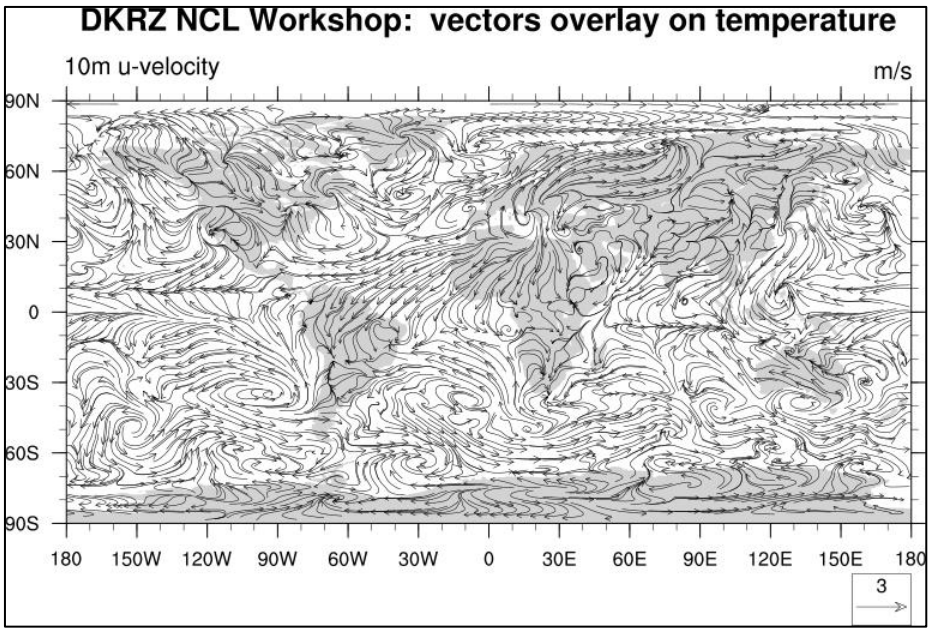
```
;-- load pre-defined functions and procedures
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin

   f    =  addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
   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","vector_default")

;-- draw the vectors

   plot = gsn_csm_vector_map_ce(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.

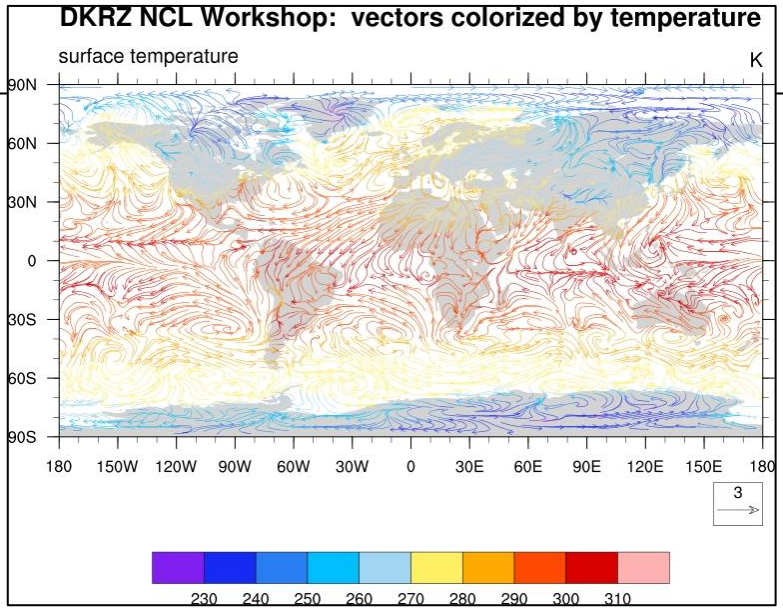
Vector field example: NCL_Tutorial/scripts/TUT_vector_curly.ncl

```
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")
   u   =  f->u10(0,:,:)                    ;-- first time step
   v   =  f->v10(0,:,:)                    ;-- first time step

;-- define the workstation (graphic will be written to a PNG file)

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

;-- set plot resources

   res                        =  True
   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@tiMainString           = "DKRZ NCL Workshop:  vector curly"

;-- draw the vectors

   plot = gsn_csm_vector_scalar_map_ce(wks,u,v,t,res)

end
```



Vector field colorized by surface temperature:
NCL_Tutorial/scripts/TUT_vector_plot_colorized.ncl

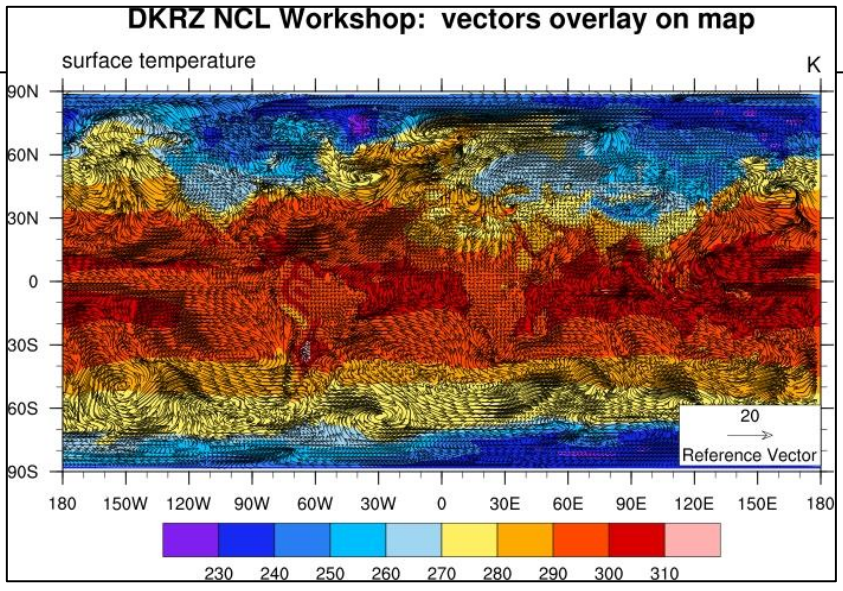
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin

   data     = "$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc"
   uname    = "u10"
   vname    = "v10"
   tname    = "tsurf"

;-- read the data

   f    =  addfile(data,"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","vector_colorized")

;-- set plot resources

   res                        =  True
   res@vcLevelPalette         =  "ncl_default" ;-- colormap for vectors
   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@tiMainString = "DKRZ NCL Workshop:  vectors colorized by temperature"

;-- draw the vectors

   plot = gsn_csm_vector_scalar_map_ce(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.

Vector field on filled contour map example:
NCL_Tutorial/scripts/TUT_vector_plot_overlay.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/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
   u   =  f->u10(0,:,:)                    ;-- first time step
   v   =  f->v10(0,:,:)                    ;-- first time step
   t   =  f->tsurf(0,:,:)                  ;-- first time step

;-- define the workstation (graphic will be written to a file)
   wks =  gsn_open_wks("png","vectors_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@mpFillOn              = False           ; no map fill
   cnres@gsnLeftString         = "surface temperature"  ; change left string
   cnres@gsnRightString        = t@units         ; assign right string
   cnres@tiMainString          = "DKRZ NCL Workshop:  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
   vcres@gsnLeftString         = ""              ; turn off left string
   vcres@tiXAxisString         = ""              ; turn off axis label

   cplot  = gsn_csm_contour_map_ce(wks,t,cnres)    ; contours over a map
   vplot  = gsn_csm_vector(wks,u,v,vcres)          ; vectors only
   overlay(cplot,vplot)
   draw(cplot)                    ; cplot now contains both itself, and vplot
   frame(wks)
end
```

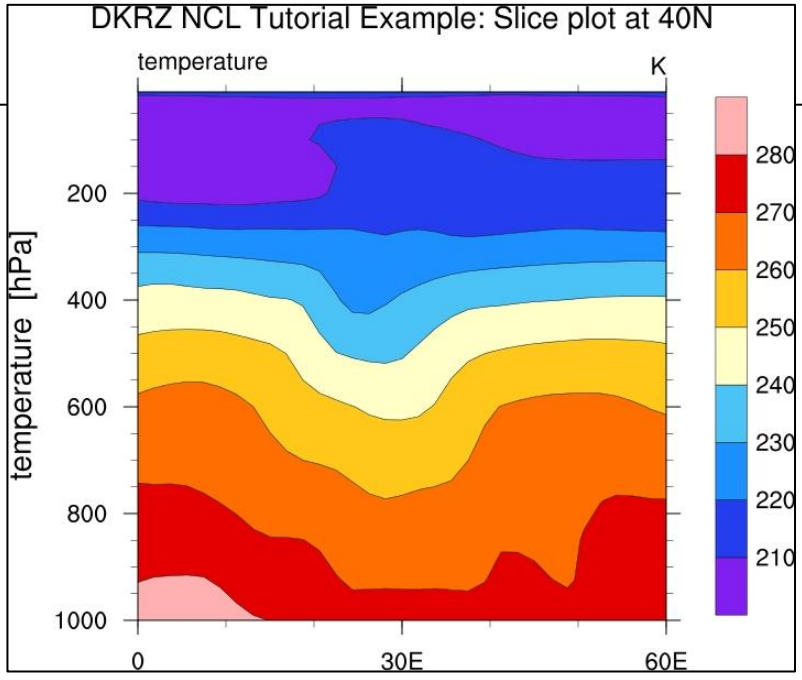


9.8 Slice Plots

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: NCL_Tutorial/scripts/TUT_slices.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_timestep1_3D.nc", "r")
   var   = f->t(0,:,{40},{0:60})     ;-- first time step, lat=40N, lon=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")
   gsn_define_colormap(wks,"ncl_default")     ;-- set the colormap to be used

;-- set resources
   res                          =  True
   res@tiMainString             = "DKRZ NCL Tutorial Example: Slice plot at 40N"
   res@cnFillOn                 =  True       ;-- turn on color fill
   res@cnLineLabelsOn           =  False      ;-- turns off contour line labels
   res@cnInfoLabelOn            =  False      ;-- turns off contour info label
   res@lbOrientation            = "vertical"  ;-- vertical label bar
   res@tiYAxisString            =  var@long_name+"  [hPa]"
                                              ;-- append units to y-axis label

   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@gsnYAxisIrregular2Linear =  True       ;-- converts irreg depth to linear
   res@trYReverse               =  True       ;-- reverses y-axis

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

end
```

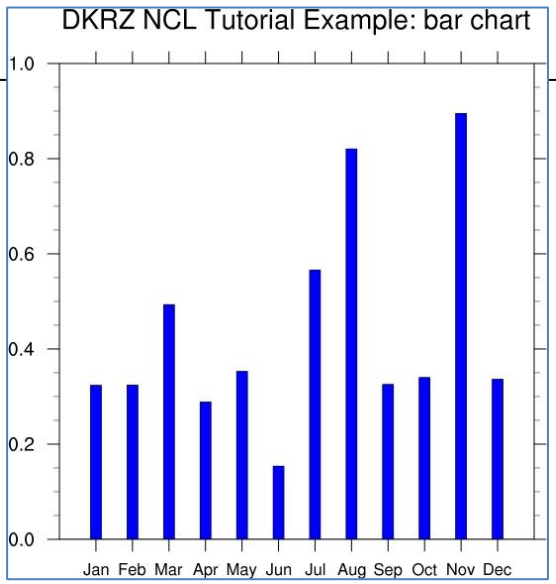


9.9 Bar Charts

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.

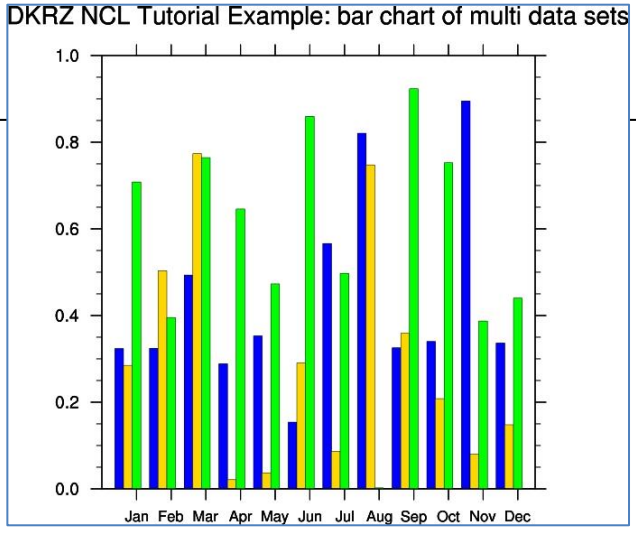
Bar chart example: NCL_Tutorial/scripts/TUT_bar_chart.ncl

```
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)                       ;-- generate x-values
   y = random_uniform(low, high, n)              ;-- generate y-values

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

   res                          =  True
   res@gsnXYBarChart            =  True          ;-- turn on bar chart mode
   res@gsnXYBarChartBarWidth    =  0.3           ;-- width of bins
   res@gsnXYBarChartColors      = "blue"         ;-- color of bins

   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              = "DKRZ NCL Tutorial Example: bar chart"

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

end
```



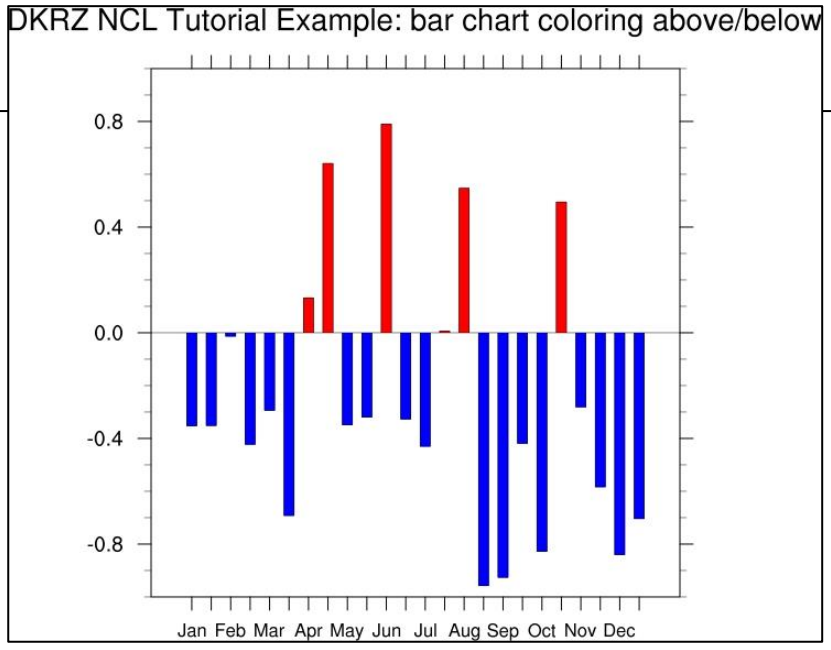
Bar chart example displaying 3 different data sets - a method which for example can be used to visualize 3 different realizations of an ensemble simulation:
NCL_Tutorial/scripts/TUT_bar_chart_multi.ncl

```
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)                    ;-- generate x-values
   y1 = random_uniform(low, high, n)            ;-- generate y1-values
   y2 = random_uniform(low, high, n)            ;-- generate y2-values
   y3 = random_uniform(low, high, n)            ;-- generate y3-values

   wks = gsn_open_wks("x11","bar_chart_multi")

   res                          =  True
   res@gsnXYBarChart            =  True         ;-- turn on bar chart mode
   res@gsnXYBarChartBarWidth =  0.25            ;-- width of bins

   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)
   res@tmXBLabels               = (/"Jan","Feb","Mar","Apr", \
                                    "May","Jun","Jul","Aug", \
                                    "Sep","Oct","Nov","Dec"/)
   res@tmXBLabelFontHeightF  = 0.015
   res@tiMainString="DKRZ NCL Tutorial 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)

end
```



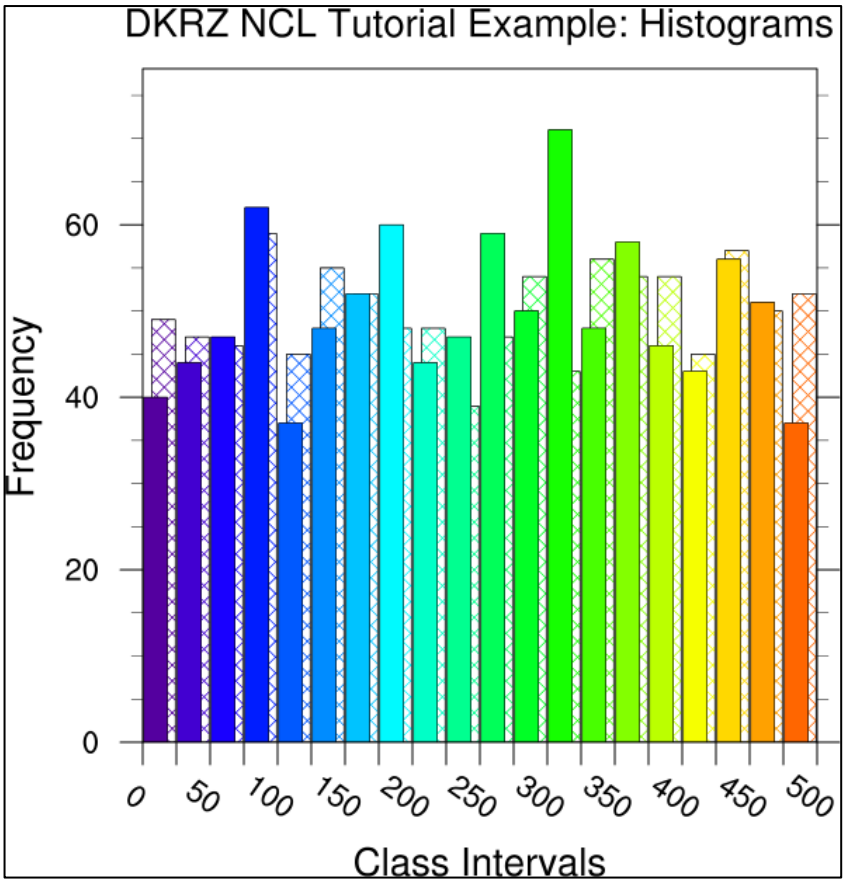
Bar chart example displaying values above or below a reference value with different colors:
NCL_Tutorial/TUT_bar_chart_col_above_below.ncl

```
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","bar_chart_col_above_below")

   res                          =  True
   res@gsnFrame                 =  False            ;-- don't advance the frame
   res@gsnXYBarChart            =  True             ;-- turn on bar chart mode
   res@gsnXYBarChartBarWidth =  0.25             ;-- width of bins

   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@tmXBLabels               =  (/"Jan","","Feb","","Mar","","Apr","",\
                                     "May","","Jun","","Jul","","Aug","","Sep",\
                                     "","Oct","","Nov","","Dec",""/)
   res@tmXBLabelFontHeightF  =  0.015
   res@tiMainString = "DKRZ NCL Tutorial 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

   plots1 = gsn_csm_xy(wks, x, y, res)

   frame(wks)
end
```



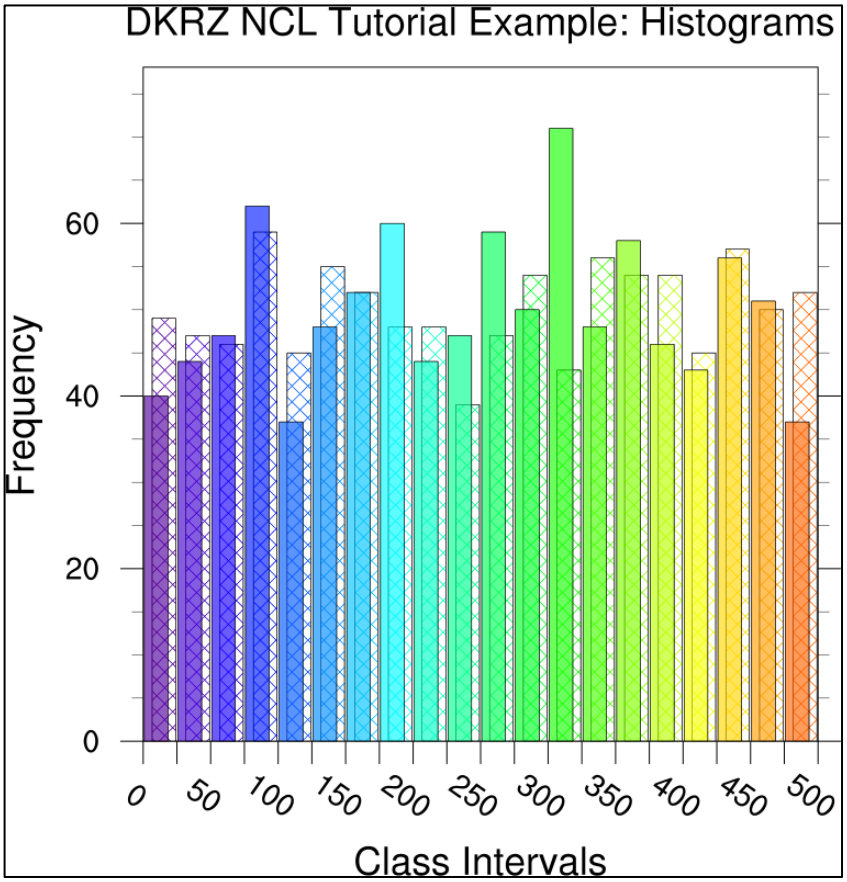
Histogram example :
NCL_Tutorial/TUT_histograms.ncl

```
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)            ;-- generate y0-data
  data(1,:) = random_uniform(0,500.,1000)            ;-- generate y1-data

  xint = ispan(0,500,25)                             ;-- generate x-values

;-- open workstation
  wks = gsn_open_wks("X11","TUT_histograms")         ;-- open workstation
  gsn_define_colormap(wks,"rainbow")                 ;-- choose colormap

  res                                    =  True     ;-- plot mods desired
  res@gsnHistogramBarWidthPercent        =  70.
  res@gsnHistogramClassIntervals         =  xint
  res@tmXBLabelAngleF                    =  325.     ;-- change label angle
  res@tmLabelAutoStride                  =  True     ;-- prevent label overlap

  res@tiMainString                      = "DKRZ NCL Tutorial Example: Histograms"

  plot=gsn_histogram(wks,data,res)                   ;-- create histogram
end
```



To change the filled histogram bars to a more transparent mode, the gsFillOpacityF resource can be used:

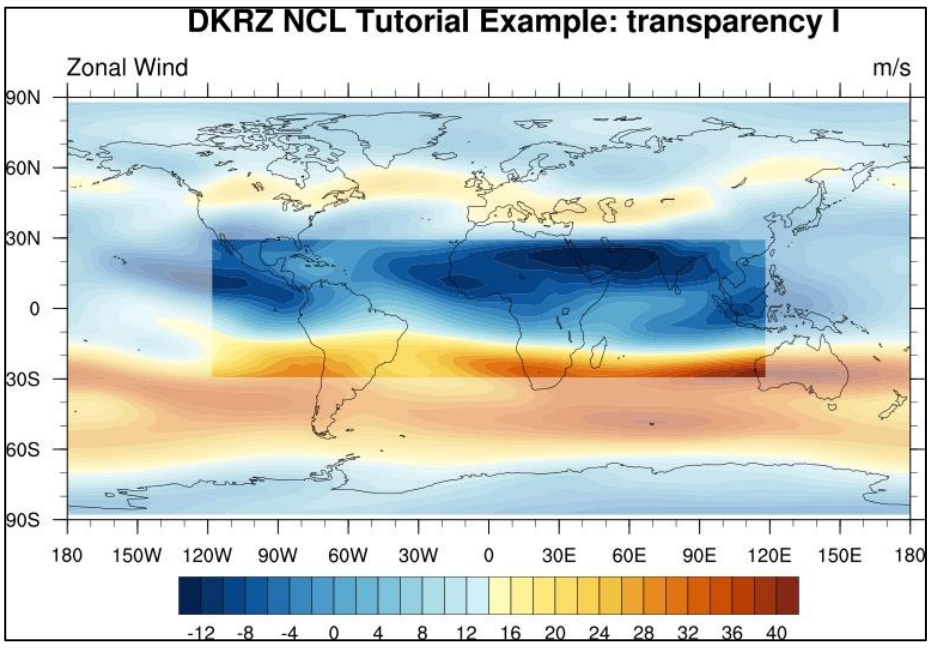
```
res@gsFillOpacityF        =  0.4
```



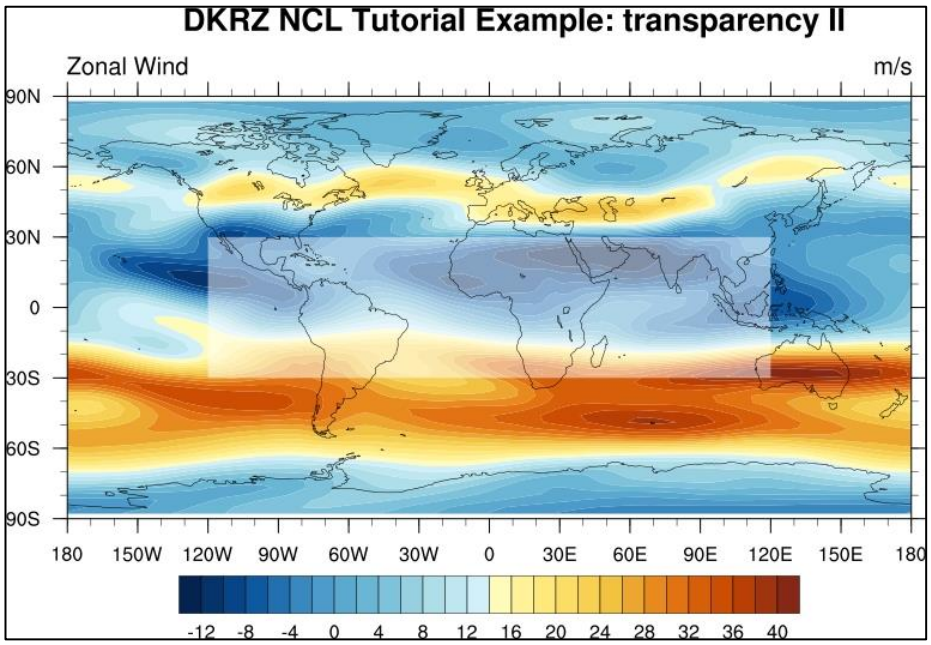
9.10 Overlay Plots

One great feature of NCL is its support for overlaying graphical elements on top of other elements. Take a look at the TUT_vector_plot_colorized.ncl example in section 9.6, 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:
NCL_Tutorial/scripts/TUT_transparent_filled_contour.ncl



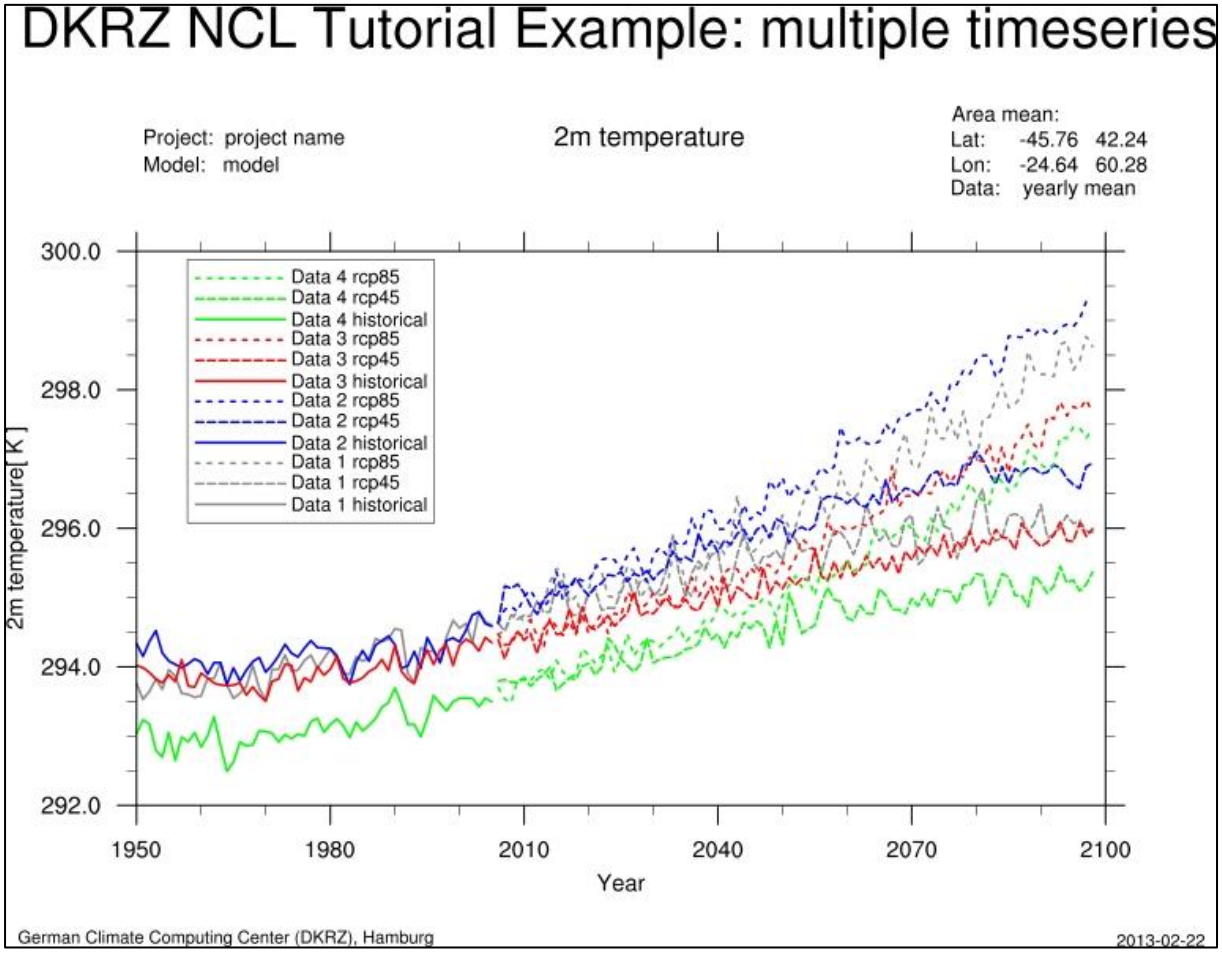
In the second image of this example, a semi transparent white rectangle is displayed on top of the opaque contours plot.



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

begin
;-- read data
  a   = addfile("$NCL_TUT/data/uv300.nc","r")
  u   = a->U(1,:,:)

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

;-- set resources
  res                         =  True
  res@gsnDraw                 =  False      ;-- don't draw the plot
  res@gsnFrame                =  False      ;-- don't advance the frame
  res@cnFillOn                =  True       ;-- turn on contour fill
  res@cnFillPalette           = "BlueYellowRed" ;-- use colormap
  res@cnLinesOn               =  False      ;-- don't draw contour lines
  res@cnLineLabelsOn          =  False      ;-- don't draw contour labels
  res@cnInfoLabelOn           =  False      ;-- don't draw info label box
  res@cnLevelSelectionMode    = "ExplicitLevels"  ;-- set explicit levels
  res@cnLevels                =  ispan(-12,40,2)  ;-- contour levels

;-- set resources only for main plot displaying the contours
  bres                        =  res
  bres@gsnMaximize            =  True       ;-- maximize main plot
  bres@mpFillOn               =  False      ;-- turn off map fill
  bres@tiMainString        = "      DKRZ NCL Tutorial 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      ;-- don't draw a labelbar

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

  overlay(main_plot,overlay_plot)    ;-- overlay overlay_plot on main_plot

  draw(main_plot)                    ;-- draw the plot
  frame(wks)                         ;-- advance 1. frame

;-- create a new plot with 100% opacity
  bres@tiMainString       = "      DKRZ NCL Tutorial 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 2. plot and advance the 2. frame
  draw(plot)
  frame(wks)
end
```

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 NCL_Tutorial/scripts/TUT_multiple_timeseries.ncl.



A very pretty example demonstrating the benefit of overlays is plotting three data sets with different grid resolutions on one map.

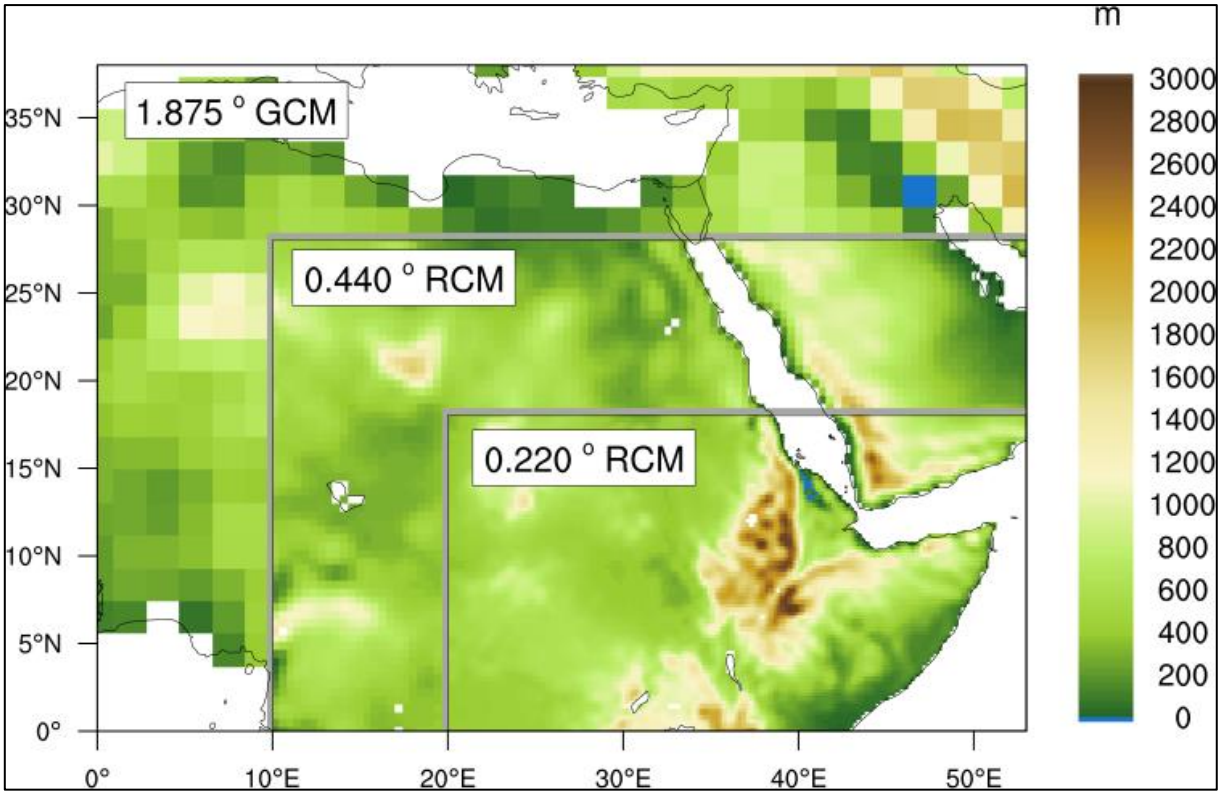
Compare grid resolution example script:
NCL_Tutorial/scripts/TUT_compare_grid_resolutions.ncl

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

begin
  lon0    =   0.0
  lon1    =  53.0
  lat0    =   0.0
  lat1    =  38.0
  border1 =  10.0
  border2 =  20.0

  file_1     = addfile("$NCL_TUT/data/orog_fx_MPI-ESM-LR_historical_r0i0p0.nc","r")
  variable1  = file_1->orog(:,:)

  mask_1 = addfile("$NCL_TUT/data/sftlf_fx_MPI-ESM-LR_historical_r0i0p0.nc","r")
  lsm1   = mask_1->sftlf(:,:)
  lsm1   = lsm1/100                 ;-- do this cause data in percent
  lsm1   = where(lsm1.gt.0.5,-9999,lsm1)

;-- mask data over ocean
  land_only1  = variable1
  land_only1  = mask(variable1,lsm1,-9999)

;-- second file
  file_2  = addfile("$NCL_TUT/data/orog_AFR-44_MPI-ESM-LR_historical_r1i1p1_CCLM_4-8-
17.nc","r")
  variable2            = file_2->orog(0,{lat0:lat1-border1},{lon0+border1:lon1})
  variable2&rlat@units = "degrees_north"
  variable2&rlon@units = "degrees_east"

  mask_2 = addfile("$NCL_TUT/data/sftlf_AFR-44_MPI-ESM-LR_historical_r1i1p1_CCLM_4-8-
17.nc","r")
  lsm2   = mask_2->sftlf(0,{lat0:lat1-border1},{lon0+border1:lon1})
  lsm2   = where(lsm2.gt.0.5,-9999,lsm2)

;-- mask data over ocean
  land_only2  = variable2
  land_only2  = mask(variable2,lsm2,-9999)

;-- third file
  file_3  = addfile("$NCL_TUT/data/orog_AFR-22_MPI-ESM-LR_historical_r1i1p1_CCLM_4-8-
17.nc","r")
  variable3            = file_3->orog(0,{lat0:lat1-border2},{lon0+border2:lon1})
  variable3&rlat@units = "degrees_north"
  variable3&rlon@units = "degrees_east"

  mask_3 = addfile("$NCL_TUT/data/sftlf_AFR-22_MPI-ESM-LR_historical_r1i1p1_CCLM_4-8-
17.nc","r")
  lsm3   = mask_3->sftlf(0,{lat0:lat1-border2},{lon0+border2:lon1})
  lsm3   = where(lsm3.gt.0.5,-9999,lsm3)

;-- mask data over ocean
  land_only3  = variable3
  land_only3  = mask(variable3,lsm3,-9999)

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

;-- define the global resources for all plots
  cnres                       =  True          ; plot mods desired
```

```
  cnres@gsnDraw                =  False
  cnres@gsnFrame               =  False
  cnres@gsnMaximize            =  True          ; Maximize plot in frame
  cnres@cnFillOn               =  True          ; turn on color fill
  cnres@cnLinesOn              =  False         ; turn of contour lines
  cnres@gsnAddCyclic           =  False         ; Don't add a cyclic point
  cnres@gsnLeftString          = ""
  cnres@gsnCenterString        = ""
  cnres@gsnRightStringFontHeightF  = -0.02
  cnres@gsnRightStringParallelPosF = 1.1       ; 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
  cnres@cnMaxLevelValF         =  3000.          ; Maximum contour level
  cnres@cnLevelSpacingF        =  20             ; Contour level spacing
  cnres@cnFillOn               =  True           ; Turn on contour fill
  cnres@cnFillMode             = "RasterFill"    ; Can be faster than "AreaFill"
  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,  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,  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)

;-- draw text on plot (map)
  txres                       =  True
  txres@txFontHeightF         =  0.02
  txres@txFontColor           = "Black"
  txres@txBackgroundFillColor = "White"
  txres@txPerimOn             =  True
```

```
  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
```

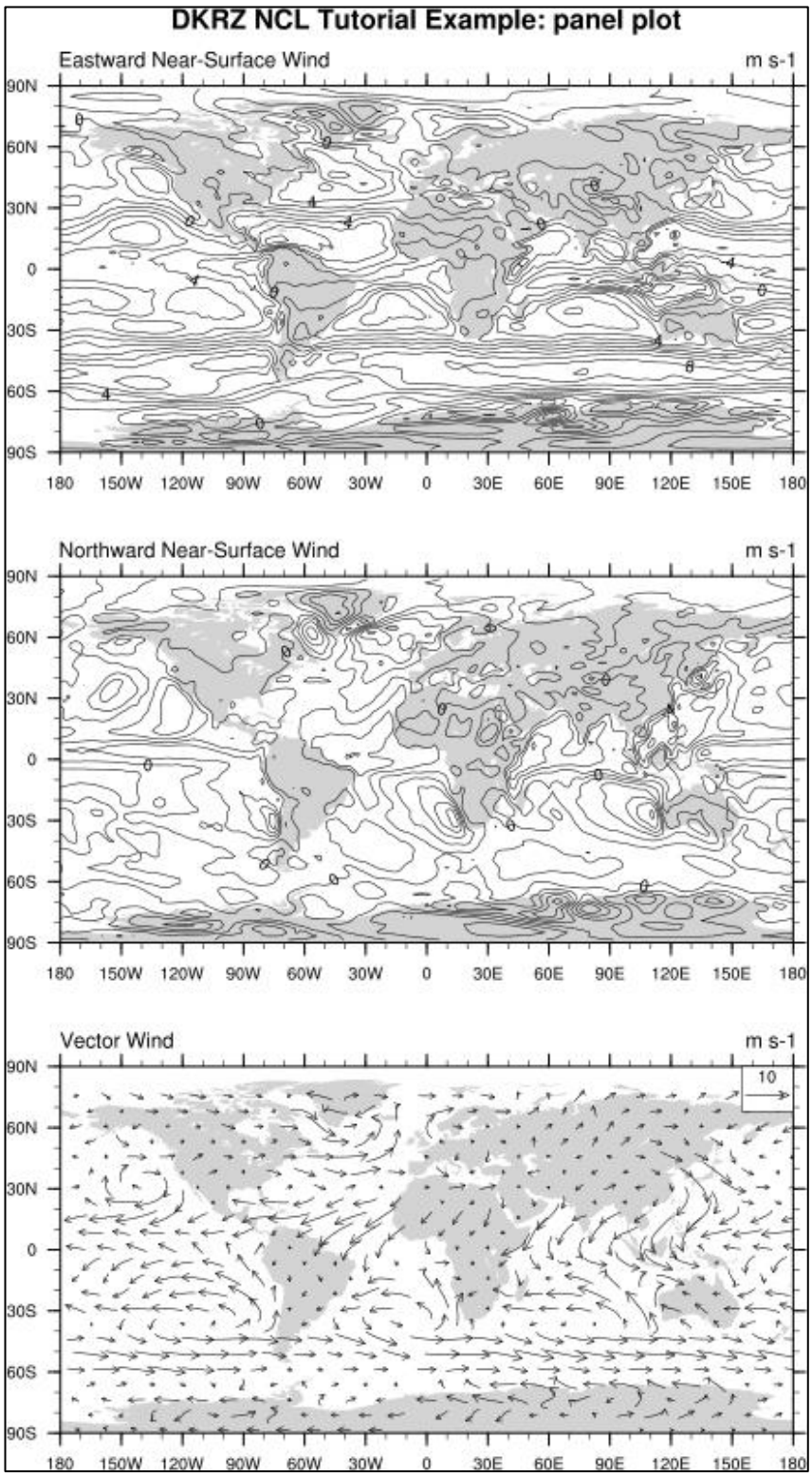


9.11 Panel Plots

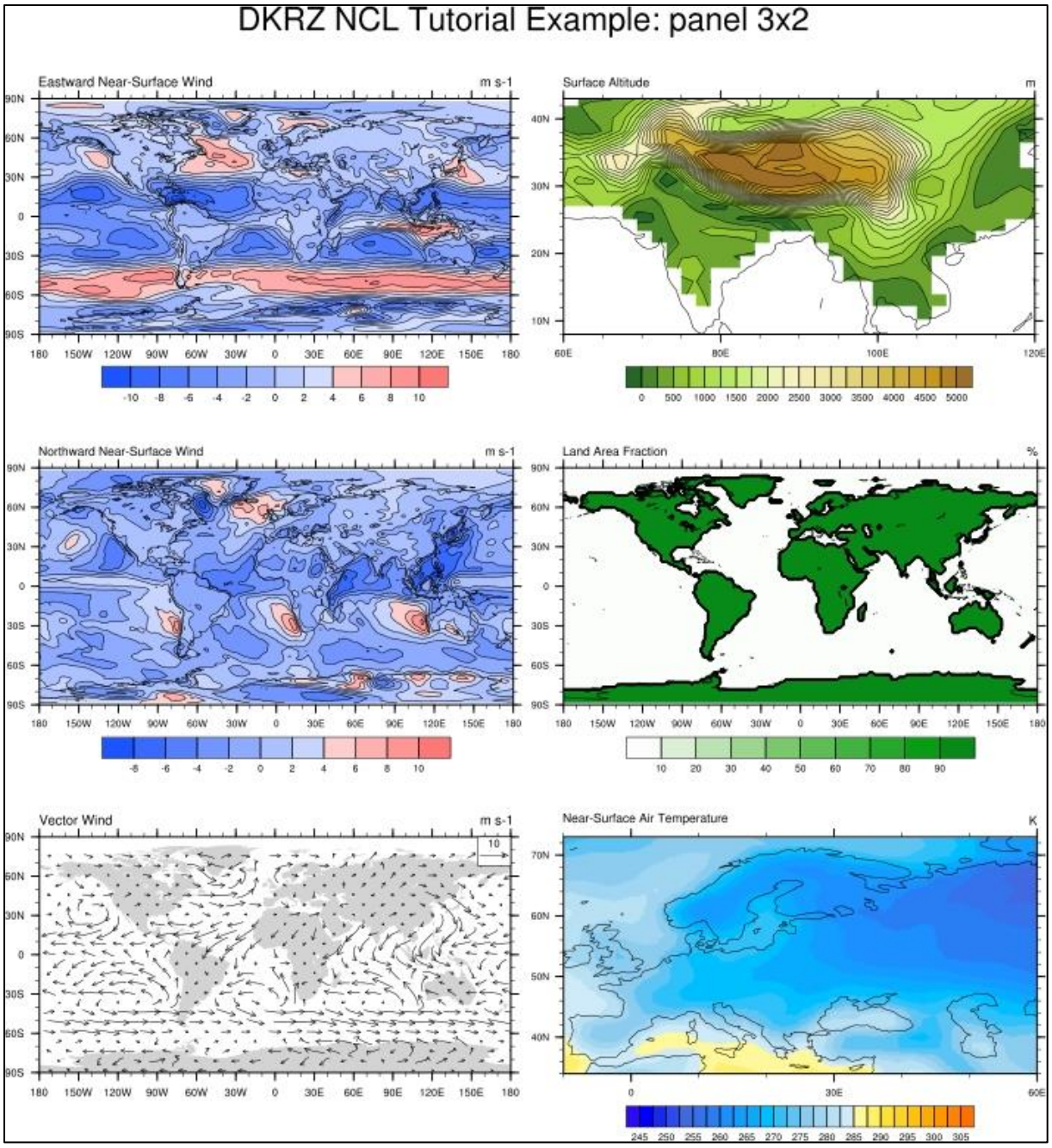
NCL allows the drawing of multiple plots on a single page (frame) using the procedure **gsn_panel**. The plots will be drawn from the left to right side of the page and from the top down to the bottom. A common labelbar or title for all plots can be included.

Panel plot example: NCL_Tutorial/scripts/TUT_panel_plot.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
  f1 = addfile("$NCL_TUT/data/orog_data1_historical_r0i0p0.nc","r")
  f2 = addfile("$NCL_TUT/data/sftlf_data1_historical_r0i0p0.nc","r")
  f3 = addfile("$NCL_TUT/data/uas_data1_historical_r1i1p1_2005.nc","r")
  f4 = addfile("$NCL_TUT/data/vas_data1_historical_r1i1p1_2005.nc","r")

  orog  = f1->orog
  sftlf = f2->sftlf
  u     = f3->uas
  v     = f4->vas

;-- open a PNG file
  wks = gsn_open_wks("png","panel_plot")

;-- create plot array
  plot = new(3,graphic)

;-- set resources for contour plots
  res                                =  True
  res@gsnDraw                        =  False
  res@gsnFrame                       =  False
  res@gsnAddCyclic                   =  True
  res@gsnMaximize                    =  True
  res@tiMainString          = "DKRZ NCL Tutorial Example: panel plot"
  res@cnInfoLabelOn                  =  False
  plot(0) = gsn_csm_contour_map_ce(wks,u(0,:,:),res)

  res@tiMainString                   = ""
  plot(1) = gsn_csm_contour_map_ce(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_ce(wks,u(0,::4,::4),v(0,::4,::4),vres)

;-- create panel plot
  gsn_panel(wks,plot,(/3,1/),False)
end
```


DKRZ NCL Tutorial Example: panel plot
Eastward Near-Surface Wind
m s-1
Northward Near-Surface Wind
m s-1
Vector Wind
m s-1
10
90N
60N
30N
0
30S
60S
90S
180 150W 120W 90W 60W 30W 0 30E 60E 90E 120E 150E 180

Panel plot example 3 rows x 2 columns: NCL_Tutorial/scripts/TUT_panel_plot_3x2.ncl



9.12 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.

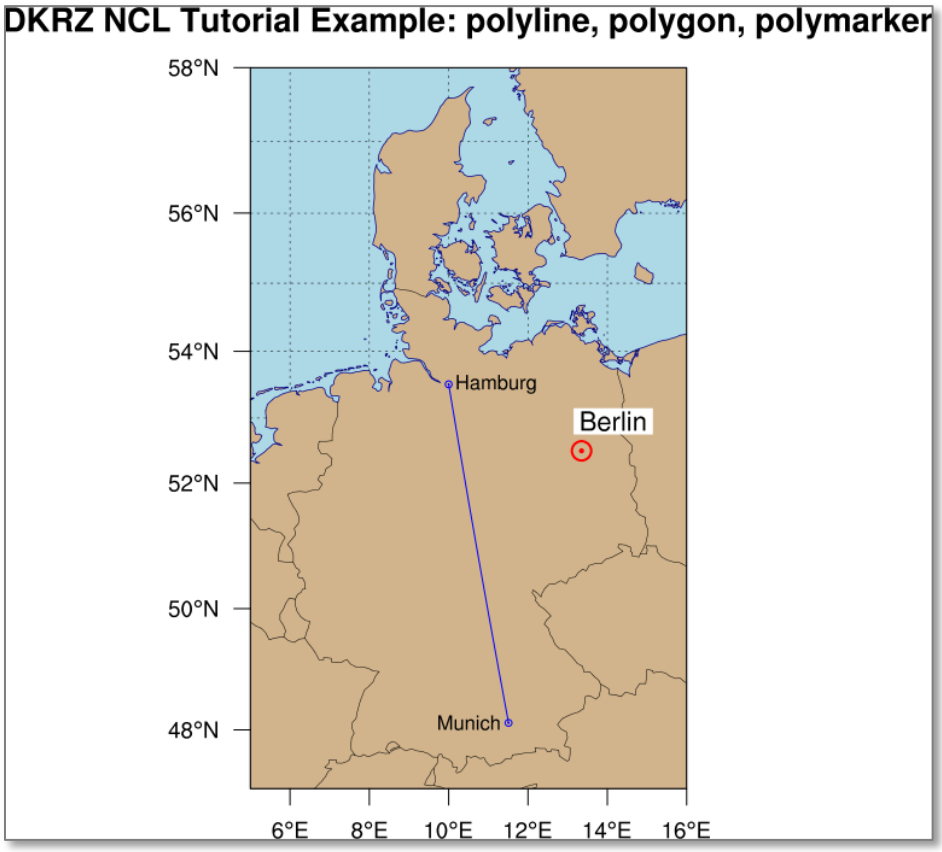
See also: http://www.ncl.ucar.edu/Applications/polyg.shtml

Example script for plotting markers at the coordinates of the cities Hamburg, Munich and Berlin and drawing a trace and a box for the text:
NCL_Tutorial/scripts/TUT_polyline_polygon_polymarker.ncl

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

begin
;-- define coordinates for the cities:  Hamburg, Munich and Berlin
  ham_lat = 53.51
  ham_lon = 10.0
  muc_lat = 48.12
  muc_lon = 11.51
  ber_lat = 52.5
  ber_lon = 13.35
  dx      = 1.0
  dy      = 0.2
  xoff    = 0.8
  yoff    = 0.45
  box_x   = (/ber_lon+xoff-dx, ber_lon+xoff+dx, \
              ber_lon+xoff+dx, ber_lon+xoff-dx, ber_lon+xoff-dx/)
  box_y   = (/ber_lat+yoff-dy, ber_lat+yoff-dy, \
              ber_lat+yoff+dy, ber_lat+yoff+dy, ber_lat+yoff-dy/)
```

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

;-- set map resources
  res                          =  True
  res@gsnDraw                  =  False
  res@gsnFrame                 =  False
  res@gsnMaximize              =  True

  res@mpFillOn                 =  True
  res@mpOutlineOn              =  True
  res@mpOutlineDrawOrder       = "Draw"
  res@mpGridAndLimbOn          =  True
  res@mpGridLineDashPattern    =  2
  res@mpGridMaskMode           = "MaskLand"
  res@pmTickMarkDisplayMode    = "Always"
  res@mpGridLatSpacingF        =  1
  res@mpGridLonSpacingF        =  2
  res@mpOceanFillColor         = "LightBlue"
  res@mpLandFillColor          = "tan"
  res@mpInlandWaterFillColor   = "Blue"
  res@mpDataBaseVersion        = "MediumRes"
  res@mpDataSetName            = "Earth..4"
  res@mpOutlineBoundarySets    = "National"     ;-- gives you country outlines
  res@mpGeophysicalLineColor   = "Navy"
  res@mpGeophysicalLineThicknessF = 1.5
  res@mpProjection             = "Mercator"
  res@mpLimitMode              = "LatLon"
  res@mpMinLonF                =   5.0
  res@mpMaxLonF                =  16.0
  res@mpMinLatF                =  47.0
  res@mpMaxLatF                =  58.0

  res@tiMainString = "DKRZ NCL Tutorial Example: polyline, polygon, polymarker"
  res@tiMainFontHeightF        =  0.02

  map = gsn_csm_map(wks, res)                 ;-- create the map

;-- add trace: polyline resources
  pres                         =  True
  pres@gsLineThicknessF        =  2.0
  pres@gsLineColor             = "blue"

  trace=gsn_add_polyline(wks, map,(/ham_lon, muc_lon/),(/ham_lat,
muc_lat/),pres)

;-- add cities: polymarker resources
  mkres                        =  True
  mkres@gsMarkerIndex          =  14
  mkres@gsMarkerColor          = "blue"
  mkres@gsMarkerThicknessF     =  2.0

  ham_city = gsn_add_polymarker(wks, map, ham_lon, ham_lat, mkres)
  muc_city = gsn_add_polymarker(wks, map, muc_lon, muc_lat, mkres)

  mkres@gsMarkerSizeF          =  0.02
  mkres@gsMarkerThicknessF     =  4.0
  mkres@gsMarkerColor          = "red"

  ber_city = gsn_add_polymarker(wks, map, ber_lon, ber_lat, mkres)
```

```
;-- add cities: text resources
  tres                          =  True
  tres@txFontHeightF            =  0.015

  ham_text = gsn_add_text(wks, map, "Hamburg", ham_lon+1.2, ham_lat, tres)
  muc_text = gsn_add_text(wks, map, "Munich",  muc_lon-1.0, muc_lat, tres)

;-- add text field: polygon  and text resources
  lres                          =  True
  lres@gsLineThicknessF         =  2.0
  lres@gsFillColor              = "white"

  ber_line = gsn_add_polygon(wks, map, box_x, box_y, lres)

  tres@txFontHeightF            =  0.02

  ber_text = gsn_add_text(wks, map, "Berlin", ber_lon+xoff, ber_lat+yoff, tres)

;-- draw the plot and advance the frame
;-- all the attached primitives will also be drawn
  draw(map)
  frame(wks)

end
```



9.13 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

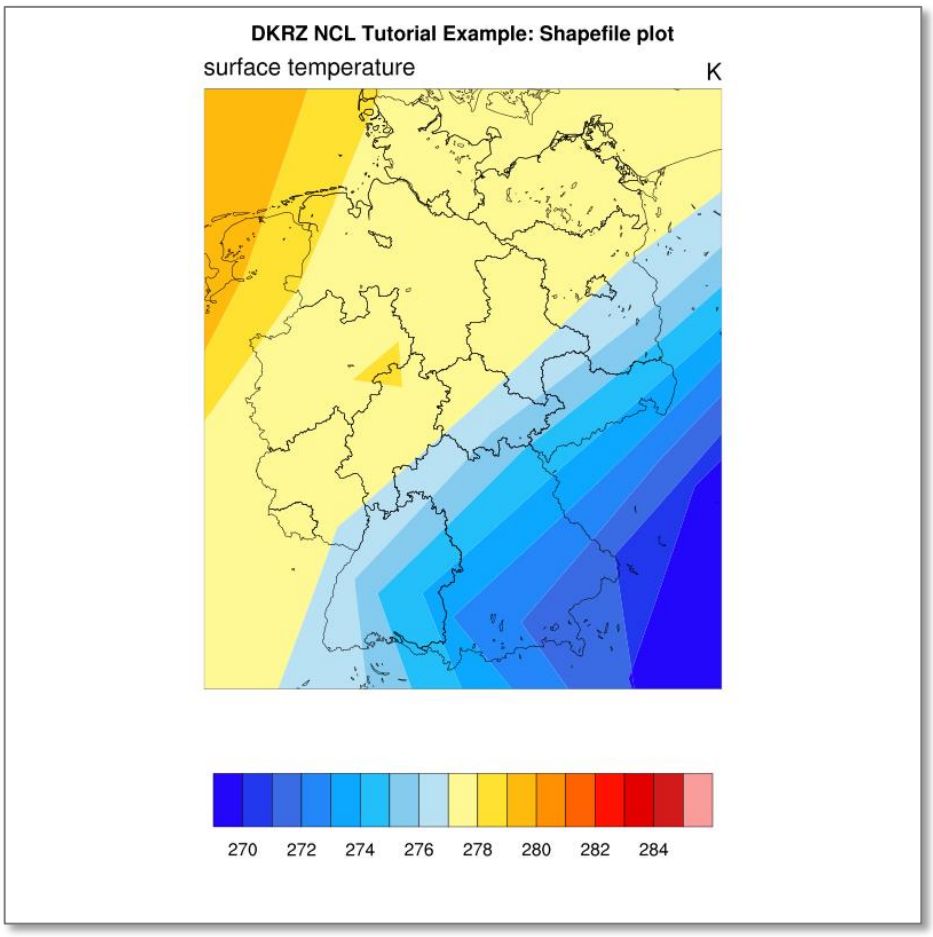
To attach shapefile points, lines, or polygons to a plot, NCL provides the **gsn** functions

- **gsn_add_shapefile_polymarkers**(wks, plot, filename, resource)
- **gsn_add_shapefile_polylines**(wks, plot, filename, resource)
- **gsn_add_shapefile_polygons**(wks, plot, filename, resource)

Simple shapefile example: NCL_Tutorial/scripts/TUT_shapefile_plot.ncl

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

begin
;-- read data to display
   g    = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
   var  = g->tsurf(0,:,:)

;-- open workstation
   wks = gsn_open_wks("X11","shapefile_plot_data")
   gsn_define_colormap(wks,"ncl_default")      ;-- choose color map

;-- 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@gsnSpreadColorStart  =  18              ;-- start at color 14
   res@gsnSpreadColorEnd    =  -3


   res@cnFillOn             =  True            ;-- turn on contour fill
   res@cnLinesOn            =  False           ;-- don't draw contour lines
   res@cnSmoothingOn        =  True            ;-- turn on smoothing
   res@cnLevelSelectionMode = "ManualLevels"   ;-- set manual levels
   res@cnMinLevelValF       =  270.0           ;-- minimum contour value
   res@cnMaxLevelValF       =  285.0           ;-- maximum contour value
   res@cnLevelSpacingF      =    1.0           ;-- contour increment

;-- select coordinates for Germany
   res@mpFillOn             =  False           ;-- turn off map fill
   res@mpLimitMode          = "LatLon"         ;-- use LatLon for Mercator
   res@mpMinLatF            =  47.             ;-- min lat
   res@mpMaxLatF            =  55.             ;-- max lat
   res@mpMinLonF            =  5.              ;-- min lon
   res@mpMaxLonF            =  16.             ;-- max lon
   res@mpDataBaseVersion    = "HighRes"        ;-- map data base
   res@mpDataResolution     = "Medium"         ;-- map resolution
   res@mpProjection         = "Mercator"       ;-- map projection
```

```
   res@tiMainString        = "DKRZ NCL Tutorial Example: Shapefile plot"
   res@tiMainFontHeightF   =  0.015

;-- generate map, but don't draw it
   plot = gsn_csm_contour_map(wks,var,res)

;-- add polylines from the shapefile to the plot
   shp_filename = ("$NCL_TUT/data/Shapefiles/DEU_adm/DEU_adm1.shp")
   poly = gsn_add_shapefile_polylines(wks,plot,shp_filename,False)

;-- draw the plot and the attached shapefile outlines and advance the frame
   draw(plot)
   frame(wks)

end
```



9.14 Colormaps

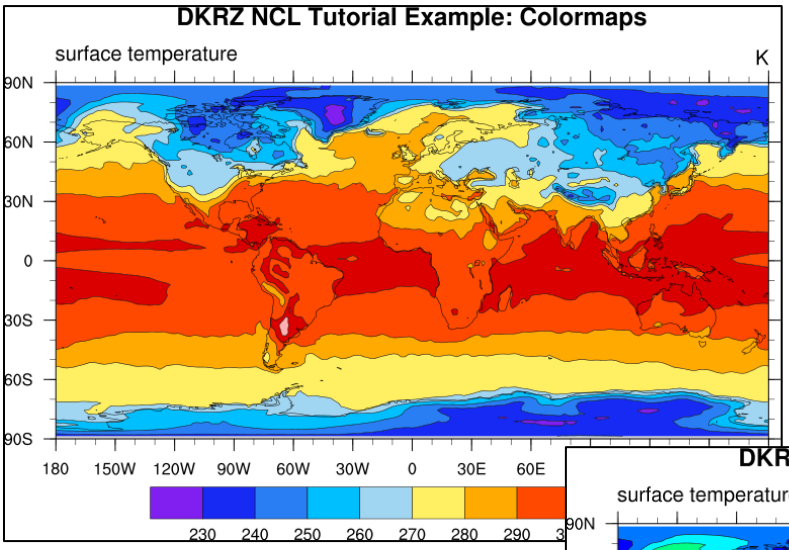
Many different color tables, also called colormaps, are available. Alternatively, you can define your own colormaps or convert color tables from other graphic packages, such as GrADS, into your own RGB (= Red Green Blue) or RGBA (= Red Green Blue and Alpha, the transparency channel) colormap in order to use it within NCL.

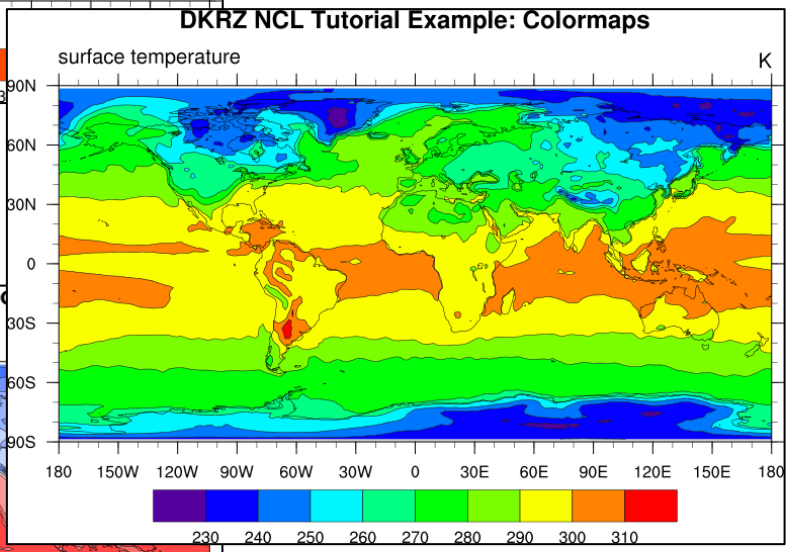
At this point, only a short description of how to handle different colormaps in NCL is given. A more detailed description can be found on the web page

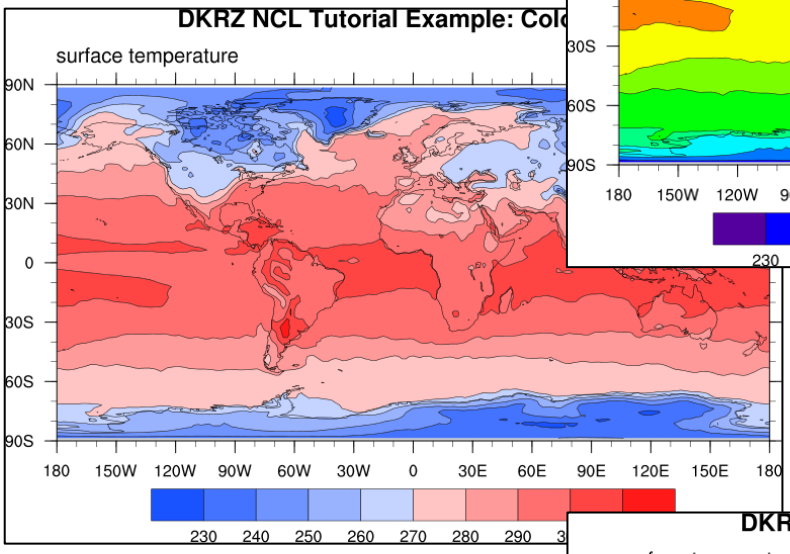
Color Table Gallery: http://www.ncl.ucar.edu/Document/Graphics/color_table_gallery.shtml

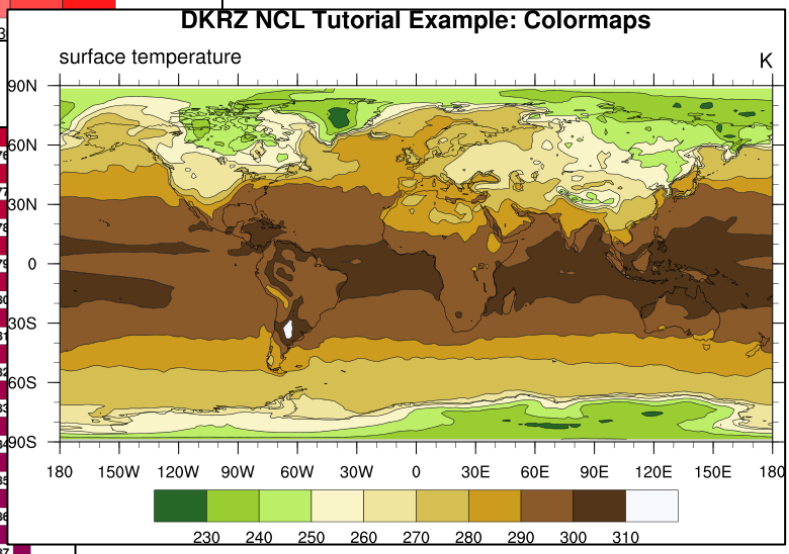
Simple colormap example: NCL_Tutorial/scripts/TUT_colormaps.ncl

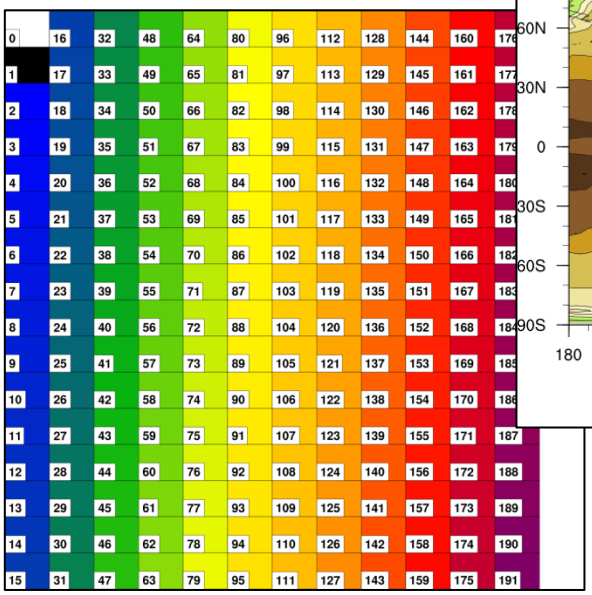
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
   file1 = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
   var   =  file1->tsurf(0,:,:)

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

;-- set resources
   res                        =  True
   res@gsnMaximize            =  True         ;-- maximize plot output
   res@cnFillOn               =  True         ;-- turn on contour fill
   res@tiMainString           = "DKRZ NCL Tutorial Example: Colormaps"
                                              ;-- title string
   res@tiMainFontHeightF      =  0.02

;-- 1: set colormap 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 colormap to "rainbow" and draw the contour map
   gsn_define_colormap(wks,"rainbow")
   plot = gsn_csm_contour_map(wks, var, res)

;-- 3: change the colormap to "BlueRed" and draw the contour map
   gsn_define_colormap(wks,"BlueRed")
   plot = gsn_csm_contour_map(wks, var, res)

;-- 4: colormap set by resource cnFillPalette instead of
;-- gsn_define_colormap
   res@cnFillPalette          = "OceanLakeLandSnow"
   plot = gsn_csm_contour_map(wks, var, res)

;-- 5: draw just the colormap - no data.
;-- !! Uses the last gsn_define_colormap setting to wks
   gsn_define_colormap(wks,"BlGrYeOrReVi200")
   gsn_draw_colormap(wks)

end
```


DKRZ NCL Tutorial Example: Colormaps
surface temperature
K
90N
60N
30N
0
30S
60S
90S
180 150W 120W 90W 60W 30W 0 30E 60E
230 240 250 260 270 280 290


DKRZ NCL Tutorial Example: Colormaps
surface temperature
K
90N
60N
30N
0
30S
60S
90S
180 150W 120W 90W 60W 30W 0 30E 60E 90E 120E 150E 180
230 240 250 260 270 280 290 300 310


DKRZ NCL Tutorial Example: Colo
surface temperature
90N
60N
30N
0
30S
60S
90S
180 150W 120W 90W 60W 30W 0 30E 60E 90E 120E 150E 180
230 240 250 260 270 280 290


DKRZ NCL Tutorial Example: Colormaps
surface temperature
K
90N
60N
30N
0
30S
60S
90S
180 150W 120W 90W 60W 30W 0 30E 60E 90E 120E 150E 180
230 240 250 260 270 280 290 300 310


0 16 32 48 64 80 96 112 128 144 160
1 17 33 49 65 81 97 113 129 145 161
2 18 34 50 66 82 98 114 130 146 162
3 19 35 51 67 83 99 115 131 147 163
4 20 36 52 68 84 100 116 132 148 164
5 21 37 53 69 85 101 117 133 149 165
6 22 38 54 70 86 102 118 134 150 166
7 23 39 55 71 87 103 119 135 151 167
8 24 40 56 72 88 104 120 136 152 168
9 25 41 57 73 89 105 121 137 153 169
10 26 42 58 74 90 106 122 138 154 170
11 27 43 59 75 91 107 123 139 155 171 187
12 28 44 60 76 92 108 124 140 156 172 188
13 29 45 61 77 93 109 125 141 157 173 189
14 30 46 62 78 94 110 126 142 158 174 190
15 31 47 63 79 95 111 127 143 159 175 191

9.14.1 Converting a GrADS color table

This shell script converts a GrADS color table to an NCL colormap:

```
#!/usr/bin/ksh
#------------------------------------------------
#-- convert GrADS color table to NCL colormap (RGB)
#------------------------------------------------
in=$1

grads_coltab=${in##*/}
coltab=NCL_${grads_coltab%.*}.rgb

ncols=$(cat ${in} | grep -v "\*" | grep -v "\#" | wc -l)
ncols=$(expr ${ncols} + 2)

#-- insert background (1) and foreground (0) colors; NCL starts with color index 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
```

9.14.2 Converting a GMT color table

In the next NCL release (6.2) the GMT color tables are automatically included.

This shell script can be used to convert a private GMT color table to an NCL colormap:

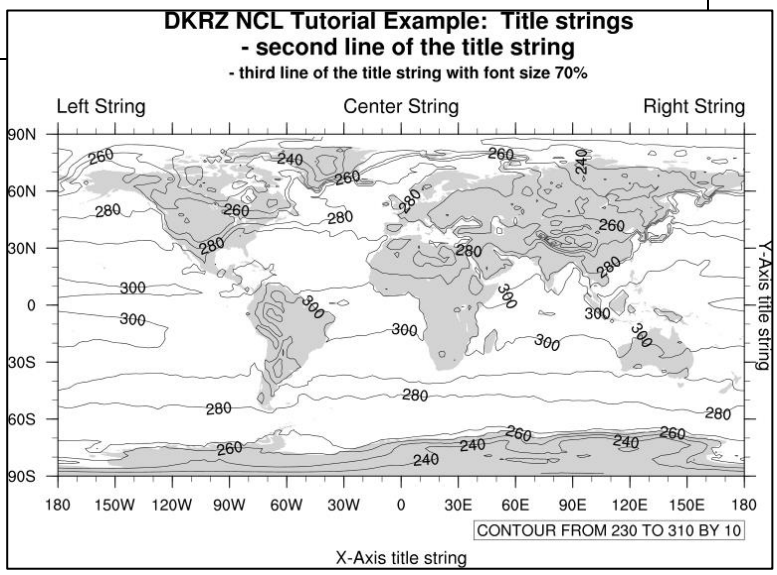
```
#!/usr/bin/ksh
#-------------------------------------------------------------------------
#-- convert GMT color tables to NCL color tables
#-------------------------------------------------------------------------
in=$1

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
```

9.15 Annotations and Labelbars

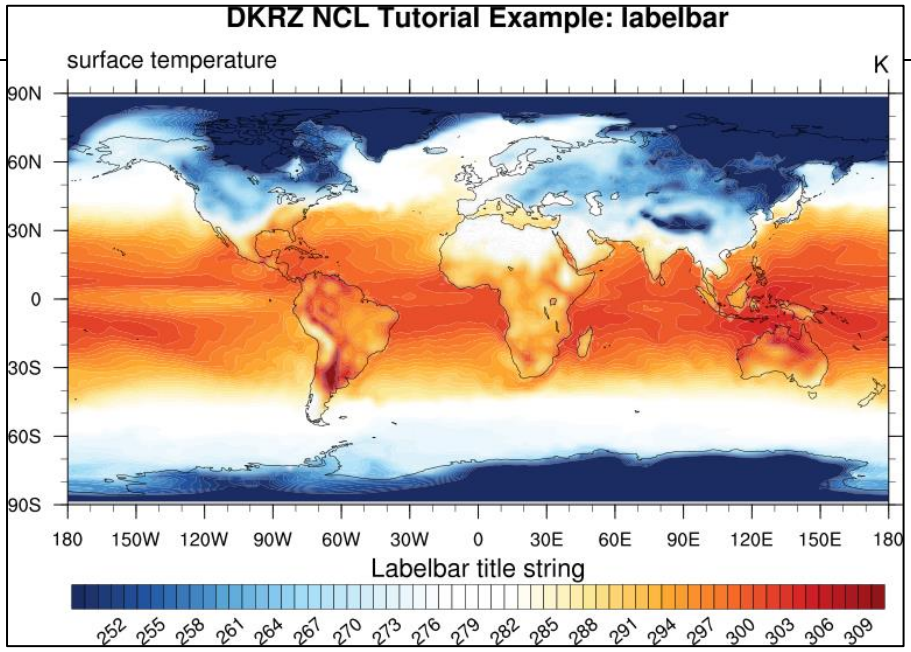
Title strings example: NCL_Tutorial/scripts/TUT_title_strings.ncl

```
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
  file1  =  addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
  var    =  file1->tsurf(0,:,:)

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

;-- set the title string. ~C~ insert a carriage return (no \ allowed).
  res@tiMainString           = "DKRZ NCL Tutorial Example:  Title strings ~C~
- second line of the title string ~C~ ~Z70~             - third line of the
title string with font size 70% ~C~  "

  res@tiMainFontHeightF      =  0.02

  res@tiXAxisString          = "X-Axis title string"
  res@tiYAxisString          = "Y-Axis title string"

  res@tiXAxisSide            = "Bottom"    ;-- X-Axis title on bottom
  res@tiYAxisSide            = "Right"     ;-- Y-axis title on right side
  res@tiYAxisAngleF          = 270         ;-- Y-axis title rotate 270 degrees

  res@tiXAxisFontHeightF     = 0.015       ;-- X-Axis title font size
  res@tiYAxisFontHeightF     = 0.015       ;-- Y-Axis title font size

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

end
```



Labelbar example: NCL_Tutorial/scripts/TUT_labelbar.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
  file1    = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
  var      = file1->tsurf(0,:,:)

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

;-- set resources
  res                          =  True
  res@gsnMaximize              =  True

  res@cnFillOn                 =  True            ;-- turn on contour fill
  res@cnLinesOn                =  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          =  1               ;-- contour level spacing

  res@lbLabelAngleF            =  40              ;-- rotate the labels
  res@lbTitleOn                =  True            ;-- turn on title
  res@lbTitleString            = "Labelbar title string"
  res@lbTitleFontHeightF       =  0.018           ;-- increase label title size
  res@pmLabelBarOrthogonalPosF = 0.12             ;-- move the labelbar downward
  res@pmLabelBarWidthF         = 0.8              ;-- increase labelbar width
  res@pmLabelBarHeightF        =  0.08            ;-- decrease labelbar height

  res@tiMainString             = "DKRZ NCL Tutorial Example: labelbar"   ;-- title
  res@tiMainFontHeightF        =  0.02

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

end
```

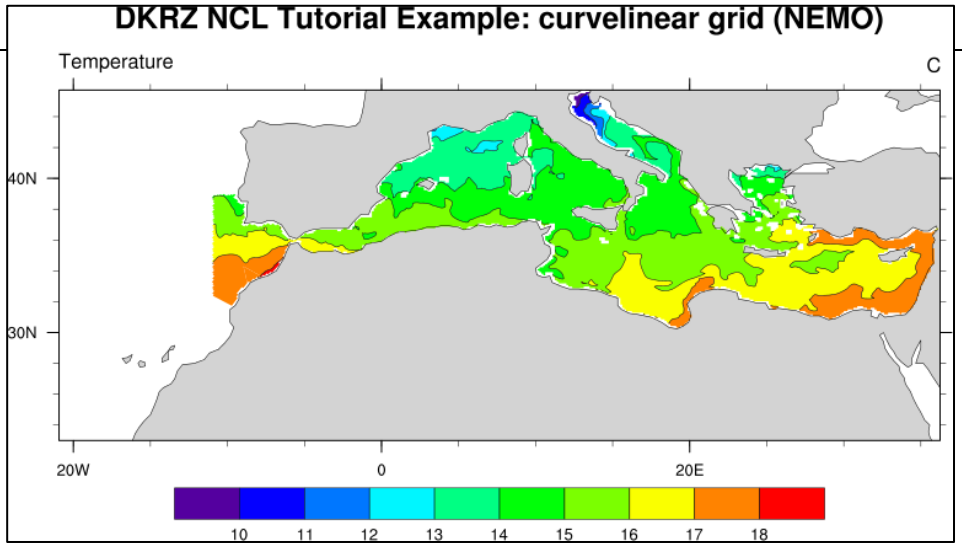


See also the chapter Helpful Resources 9.19.

9.16 Curvilinear grids

Curvilinear grids are those represented by two-dimensional latitude/longitude arrays. When plotting curvilinear grids, you need to set the special sfXArray and sfYArray resources to the two-dimensional longitude and latitude arrays, respectively.
Curvilinear grid example: NCL_Tutorial/scripts/TUT_curvilinear_grid.ncl

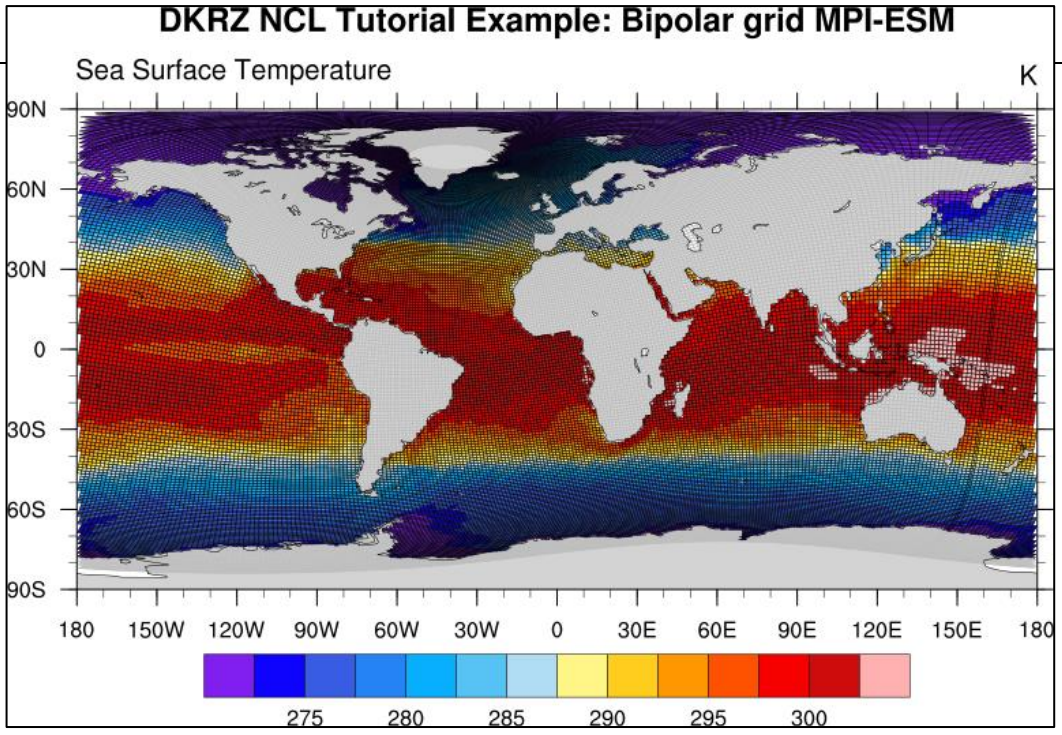
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
;-- read the data and define
   f       =  addfile("$NCL_TUT/data/CNTASN_1m_200103_grid_T.nc","r")

   var     =  f->votemper(0,0,:,:)
   lat2d   =  f->nav_lat
   lon2d   =  f->nav_lon

;-- define the workstation (plot type and name)
   wks = gsn_open_wks("X11","TUT_curvilinear")

;-- set resources
   res                        =  True
   res@gsnAddCyclic           =  False          ;-- don't add lon cyclic point
   res@gsnMaximize            =  True

   res@cnFillOn               =  True           ;-- turn off contour fill
   res@cnFillPalette          = "rainbow"       ;-- choose color map

   res@tiMainString      = "DKRZ NCL Tutorial Example: curvilinear grid (NEMO)"
   res@tiMainFontHeightF     =  0.02

;---Zoom in on map
   res@mpMinLatF              =  min(lat2d)      ;-- min lat
   res@mpMaxLatF              =  max(lat2d)      ;-- max lat
   res@mpMinLonF              =  min(lon2d)      ;-- min lon
   res@mpMaxLonF              =  max(lon2d)      ;-- max lon

;---Lat/lon arrays of curvilinear frid for overlaying on map
   res@sfXArray               =  lon2d
   res@sfYArray               =  lat2d

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

end
```



9.16.1 MPI-ESM-LR

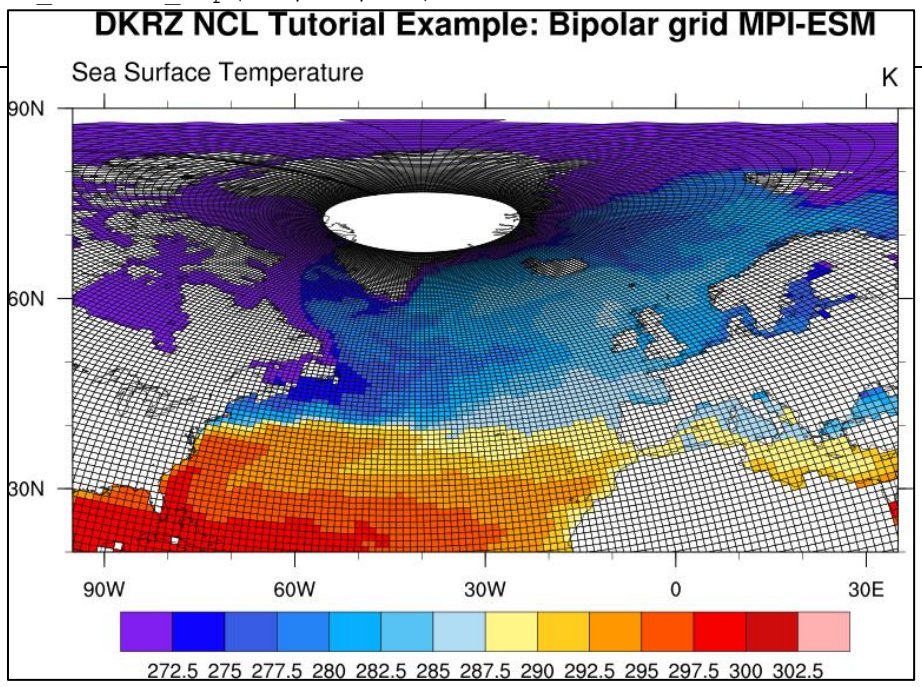
MPI-ESM bipolar grid example (MPI-OM TP04):
NCL_Tutorial/scripts/TUT_bipolar_grid_MPI-ESM.ncl

```
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
   f  =  addfile("$NCL_TUT/data/tos_Omon_MPI-ESM-LR_rcp45_r1i1p1_20060101.nc","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","bipolar_grid_MPI-ESM")

;-- set resources
   res                         =  True
   res@gsnMaximize             =  True            ;-- maximize plot output

   res@cnFillOn                =  True            ;-- turn on contour fill
   res@cnFillMode              = "CellFill"
   res@cnFillPalette           = "ncl_default"    ;-- choose a colormap
   res@cnLinesOn               =  False           ;-- turn lines off
   res@cnLineLabelsOn          =  False           ;-- turn labels off

   res@cnCellFillEdgeColor    = 1
   res@cnCellFillMissingValEdgeColor = "grey"    ;-- _FillValue color

   res@tiMainString            = "DKRZ NCL Tutorial Example: Bipolar grid MPI-ESM"
   res@tiMainFontHeightF       =  0.02

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

end
```



Plot a sub-region of the MPI-ESM bipolar grid example (MPI-OM TP04):
NCL_Tutorial/scripts/TUT_bipolar_grid_MPI-ESM_subregion.ncl

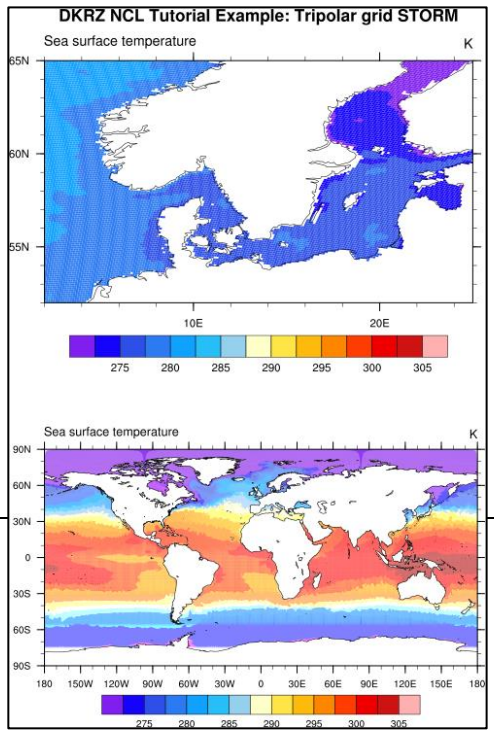
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
;-- read the data and define
   f = addfile("$NCL_TUT/data/tos_Omon_MPI-ESM-LR_rcp45_r1i1p1_20060101.nc","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 colormap
   res@cnLinesOn                =  False          ;-- turn lines off
   res@cnLineLabelsOn           =  False          ;-- turn labels off
   res@cnCellFillEdgeColor      = 1
   res@cnCellFillMissingValEdgeColor = "black"  ;-- _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             = "DKRZ NCL Tutorial Example: Bipolar grid MPI-ESM"
   res@tiMainFontHeightF        =  0.02

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

end
```



9.16.2 STORM

STORM: MPI-OM TP6M tripolar grid example:
NCL_Tutorial/scripts/TUT_tripolar_grid_STORM.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
   f  =  addfile("$NCL_TUT/data/tp6ml80_srtm30plus_omip_int_mpiom_data_mm_0031-01-01_0031-
06-30_t1_sst_sss.nc","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","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"       ;-- fill mode
   res@cnFillPalette          = "ncl_default"    ;-- choose a colormap
   res@cnLinesOn              =  False           ;-- Turn lines off
   res@cnLineLabelsOn         =  False           ;-- Turn labels off
   res@tiMainString           = "DKRZ NCL Tutorial Example: Tripolar grid STORM"
   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@tiMainString)

   plot(1) = gsn_csm_contour_map(wks,var,res)

;-- create panel plot
   gsn_panel(wks,plot,(/2,1/),False)

end
```

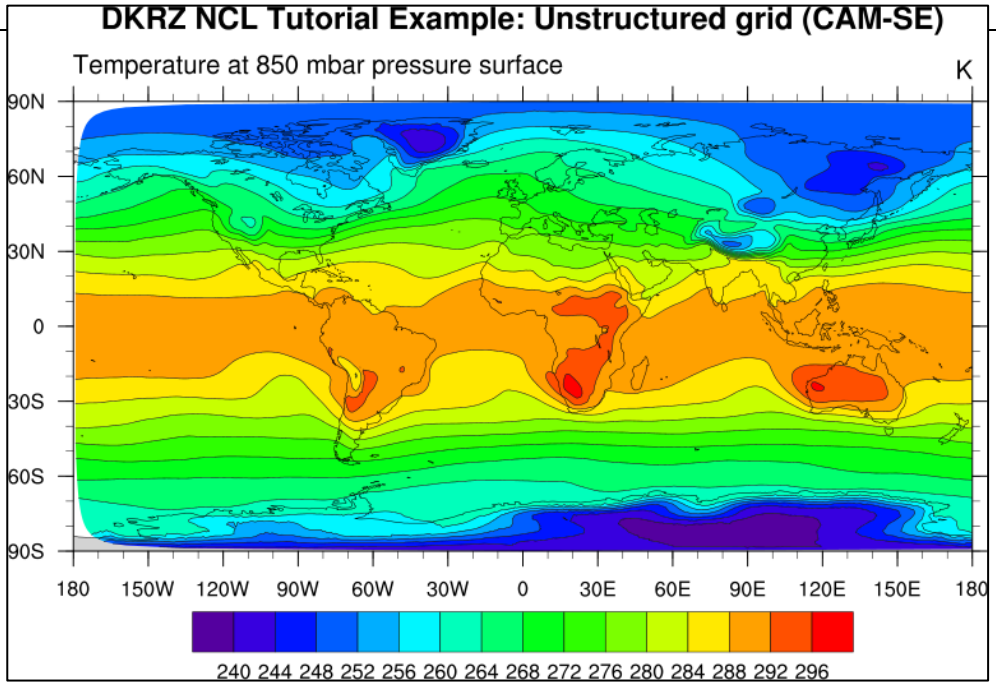


9.17 Unstructured Grids

Unstructured grids are typically defined as points or cells, and consist of one-dimensional arrays of values, latitude points, and longitude points. As with curvilinear grids, you also need to set the special sfXArray and sfYArray resources, except this time to the one-dimensional longitude and latitude arrays, respectively.

Unstructured grid example: NCL_Tutorial/scripts/TUT_unstructured_grid.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
   f       =  addfile("$NCL_TUT/data/b1850cam5cn_camse_01_climo_subset.nc","r")

   var     =  f->T850

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

;-- set resources
   res                   =  True
   res@gsnMaximize       =  True            ;-- maximize plot output

   res@cnFillOn          =  True            ;-- turn on contour fill
   res@cnFillPalette     = "rainbow"        ;-- choose a colormap

   res@tiMainString      = "DKRZ NCL Tutorial Example: Unstructured grid (CAM-SE)"
   res@tiMainFontHeightF =  0.02

;---Lat/lon arrays of unstructured grid for overlaying on map
   res@sfXArray          =  f->lon
   res@sfYArray          =  f->lat

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

end
```



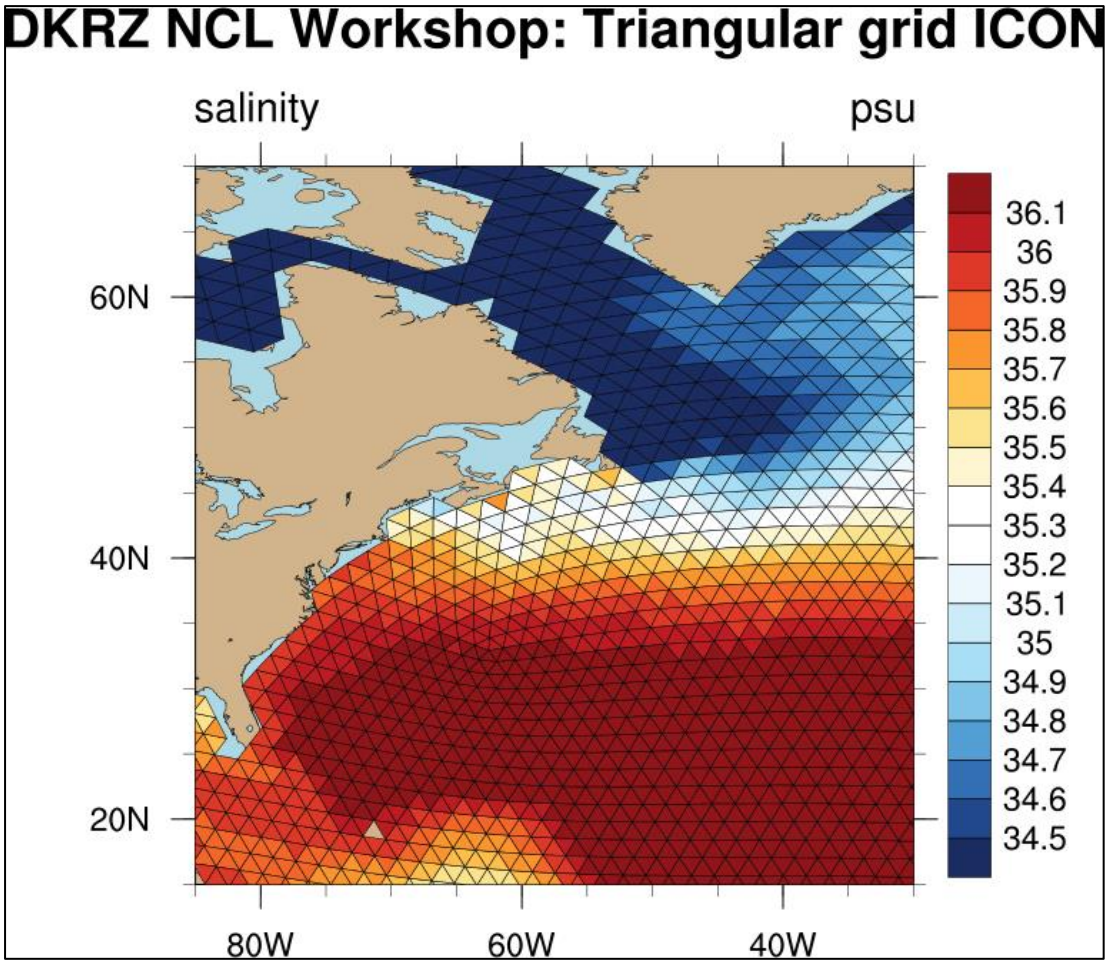
9.17.1 ICON

ICON icosahedral grid example: NCL_Tutorial/scripts/TUT_triangular_grid_ICON.ncl

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

begin
  f        = addfile("$NCL_TUT/data/ocedat.S.nc", "r" )
  var      = f->S(0,2,:)

  var@_FillValue = getVarFillValue(var)    ;-- retrieve missing value of var,
                                           ;-- otherwise, return default _FillValue

  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.                          ;-- minimum longitude
  lon_max  = -30.                          ;-- maximum longitude
  lat_min  =  15.                          ;-- minimum latitude
  lat_max  =  70.                          ;-- maximum latitude

  rad2deg = 45./atan(1.)                   ;-- 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. )
     var@_FillValue = -999.                      ;--  missing value
  else
     print("WARNING: variable wet_c doesn't exist  -->  no masking")
  end if

;-- lon: 0 - 360
  vlon = f->clon_vertices * rad2deg
  vlon = where(vlon.lt.0, vlon + 360, vlon)
  vlat = f->clat_vertices * rad2deg

  vminimum = min(var)
  vmaximum = max(var)

;-- open workstation and define colormap
  wks = gsn_open_wks("png", "triangular_grid_ICON")

  res = True
  res@gsnDraw                =  False
  res@gsnFrame               =  False
  res@gsnMaximize            =  True
  res@gsnAddCyclic           =  False
  res@gsnPaperOrientation    = "landscape"
  res@gsnRightString         =  var@units
  res@tiMainString           =  "DKRZ NCL Workshop: Triangular grid ICON"
  res@mpFillOn               =  True
  res@mpFillDrawOrder        = "PostDraw"
  res@mpProjection           = "CylindricalEquidistant"
  res@mpLimitMode            = "LatLon"
  res@mpMinLonF              =  lon_min
```

```
  res@mpMaxLonF              =  lon_max
  res@mpMinLatF              =  lat_min
  res@mpMaxLatF              =  lat_max
  res@mpGreatCircleLinesOn   =  True
  res@mpDataBaseVersion      = "MediumRes"
  res@mpLandFillColor        = "tan"
  res@mpOceanFillColor       = "LightBlue"
  res@mpInlandWaterFillColor = "LightBlue"
  res@sfXArray               =  x
  res@sfYArray               =  y
  res@cnFillOn               =  True
  res@cnFillPalette          = "BlueWhiteOrangeRed"
  res@cnLinesOn              =  False
  res@cnLineLabelsOn         =  False
  res@cnInfoLabelOn          =  False
  res@cnFillMode             = "RasterFill"
  res@cnRasterSmoothingOn    =  True
  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@lbOrientation          = "vertical"

;-- cut off values out of the given data range and set them to _FillValue
  plot = gsn_csm_contour_map(wks,var,res)

;-- get contour informations
  getvalues plot@contour
     "cnLevels"     : levels                     ;-- # 26 (n)
     "cnFillColors" : colors                     ;-- # 27 (n+1)
  end getvalues

;-- clear the underlying contour plot, but keep all the information
  plot = setColorContourClear(plot,vminimum,vmaximum)
  draw(plot)

;-- set resources for the triangles
  pres                       =  True
  pres@gsEdgesOn             =  True              ;-- turn on edges
  pres@gsFillIndex           =  0                 ;-- solid fill

;-- plot data less than
  k=1
  vind = ind(var .lt. var_min)
  do j = 0, dimsizes(vind) - 1
     pres@gsFillColor = colors(0)
     xp=vlon(vind(j),:)
     yp=vlat(vind(j),:)
     gsn_polygon(wks,plot,xp,yp,pres)
     delete([/xp,yp/])
     k=k+1
  end do
  delete(vind)

;-- plot all cells
  m=1
  do i = 0, dimsizes(levels) - 2
    vind = ind(var .ge. levels(i) .and. var .lt. levels(i+1))
    do j = 0, dimsizes(vind) - 1
       pres@gsFillColor = colors(i+1)
       xp=vlon(vind(j),:)
       yp=vlat(vind(j),:)
```

```
        gsn_polygon(wks,plot,xp,yp,pres)
        delete([/xp,yp/])
        m=m+1
     end do
     delete(vind)
  end do

;-- plot data greater than
  nl=num(colors)
  nl=nl-1
  n=1
  vind = ind(var .gt. var_max)
  do j = 0, dimsizes(vind) - 1
     pres@gsFillColor = colors(nl)
     xp=vlon(vind(j),:)
     yp=vlat(vind(j),:)
     gsn_polygon(wks,plot,xp,yp,pres)
     delete([/xp,yp/])
     n=n+1
  end do
  delete(vind)

  print("--> triangles calculated:  "+ (m+n+k))

;-- generate the plot output
  frame(wks)
end
```



9.18 Globe with different grid resolutions

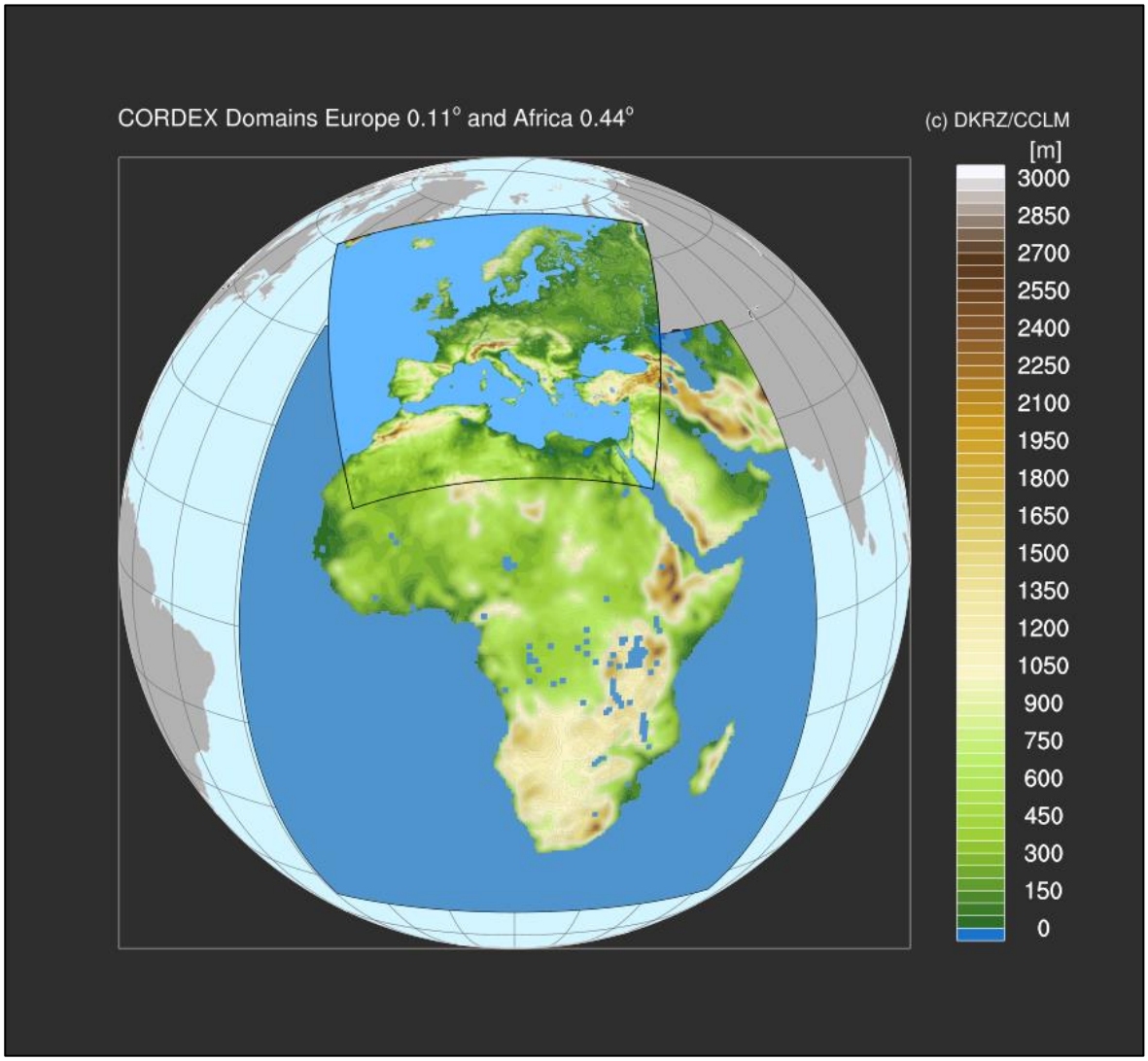
A very special kind of plot is the following example, which shows two regional model domains with different grid resolutions mapped onto one globe. The background color of the plotting frame is changed to dark grey and the foreground color to white. One single colormap is used to visualize the orography 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 distinguishuable. This is a good demonstration of the capabilities of NCL graphics.

Globe with different grid resolutions example:
NCL_Tutorial/scripts/TUT_globe_orography_grid_resolution.ncl

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

begin
  f1          = addfile("$NCL_TUT/data/HSURF_EUR-11_model_region.nc","r")
  var1        = f1->HSURF(0,:,:)
  mask1       = addfile("$NCL_TUT/data/FR_LAND_EUR-11_model_region.nc","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("$NCL_TUT/data/HSURF_AFR-44_model_region.nc","r")
  var2        = f2->HSURF(0,:,:)
  mask2       = addfile("$NCL_TUT/data/FR_LAND_AFR-44_model_region.nc","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, "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@trYReverse            =  True
  cnres2@lbOrientation         = "vertical"
  cnres2@lbLabelFontHeightF    =  0.013
  cnres2@gsnMaximize           =  True
  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               =
  cnres@cnMissingValFillColor  = "steelblue1"
  cnres@cnLinesOn              =  False
  cnres@cnLineLabelsOn         =  False
  cnres@cnLevelSelectionMode   = "ManualLevels"
  cnres@cnMinLevelValF         =  0.0
  cnres@cnMaxLevelValF         =  3000.
  cnres@cnLevelSpacingF        =  50.
  cnres@cnFillPalette          = "OceanLakeLandSnow"
  cnres@cnFillDrawOrder        = "PostDraw"
  cnres@gsnRightString              = "["+var1@units+"]"
  cnres@gsnRightStringFontHeightF    =  0.013
  cnres@gsnRightStringParallelPosF   =  1.19
  cnres@gsnRightStringOrthogonalPosF = -0.007
```

```
  cnres@tiXAxisString          = ""
  cnres@tiYAxisString          = ""

  cnres@gsnLeftString          = ""
  cnres@trYReverse             =  True
  cnres@tfDoNDCOverlay         =  False    ; transform to standard lat/lon
  cnres@sfXArray               =  lon2d
  cnres@sfYArray               =  lat2d
  cnres@lbOrientation          = "vertical"
  cnres@lbLabelFontHeightF     =  0.013
  cnres@gsnMaximize            =  True

;-- 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 = "CORDEX 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
```


CORDEX Domains Europe 0.11° and Africa 0.44°
(c) DKRZ/CCLM
[m]
3000
2850
2700
2550
2400
2250
2100
1950
1800
1650
1500
1350
1200
1050
900
750
600
450
300
150
0

9.19 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 file
res@cnFillOn           = True
res@tiMainFontHeightF = 0.02
```

The first two lowercase letters are the abbreviation of the resource type.

Resource types:

am	Annotation Manager (AnnoManager)
app	App (App)
ca	Coordinate Array (CoordArrays)
cn	Contour (ContourPlot)
ct	Coordinate Array Table (CoordArrTable)
dc	Data Comm (DataComm)
err	Error
gs	Graphic Style (GraphicStyle)
gsn	GSN High-level Interfaces (GSN)
lb	Label Bar (LabelBar)
lg	Legends (Legend)
mp	Maps (MapPlot and MapTransformation)
pm	Plot Manager (PlotManager)
pr	Primitives (Primitive)
sf	Scalar Field (ScalarField and MeshScalarField)
st	Streamline (StreamlinePlot)
tf	Transform
ti	Title
tm	Tickmark (TickMark)
tr	Transformation (Transformation, IrregularTransformation, LogLinTransformation)

tx	Text (TextItem)
vc	Vectors (VectorPLot)
vf	Vector Field (VectorField)
vp	View Port (View)
wk	Workstation (Workstation, DokumentWorkstation, ImageWorkstation, NcgmWorkstation, PDFWorkstation, PSWorkstation, XWorkstation)
ws	Workspace
xy	XY-Plots (XyPlot)

9.19.1 Title Strings and Function Codes

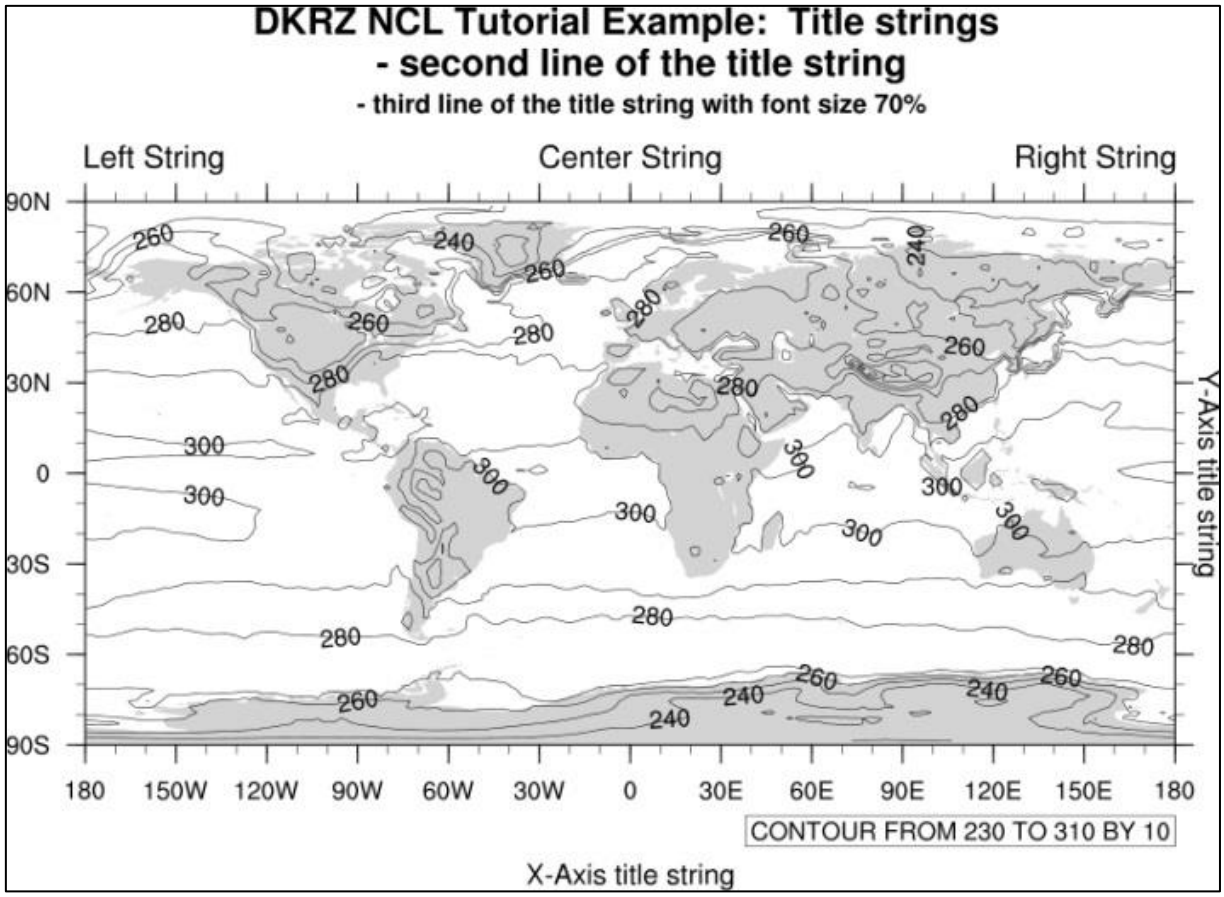
The "~" character in NCL has a special meaning for all text-based resources. It's a "function code" that tells NCL you want to do something special to the string, like create a super/subscript, insert a carriage return, or change the font:

Text/title examples: http://www.ncl.ucar.edu/Applications/text.shtml
Function code examples: 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 ^{o}
Subscript:	~B~2~N~	subscript 2 like $_{2}$
Select font:	~F33~	select greek font
Reset to default:	~N~	

More than one title string on top of the plot: NCL_Tutorial/scripts/TUT_title_strings.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
  file1    =  addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
  var      =  file1->tsurf(0,:,:)

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

;---- set resources
  res                       =  True
  res@gsnMaximize           =  True
  res@gsnLeftString         = "Left String"
  res@gsnCenterString       = "Center String"
  res@gsnRightString        = "Right String"

  res@tiMainString          = "DKRZ NCL Tutorial Example:  Title strings ~C~
- second line of the title string ~C~ ~Z70~             - third line of the
title string with font size 70% ~C~  "
  res@tiMainFontHeightF     =  0.02

  res@tiXAxisString         = "X-Axis title string"
  res@tiYAxisString         = "Y-Axis title string"
  res@tiXAxisSide           = "Bottom"    ;-- X-Axis title on bottom
  res@tiYAxisSide           = "Right"     ;-- Y-axis title on right side
  res@tiYAxisAngleF         =  270        ;-- Y-axis title rotate 270 degrees
  res@tiXAxisFontHeightF    =  0.015      ;-- X-Axis title font size
  res@tiYAxisFontHeightF    =  0.015      ;-- Y-Axis title font size

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

end
```

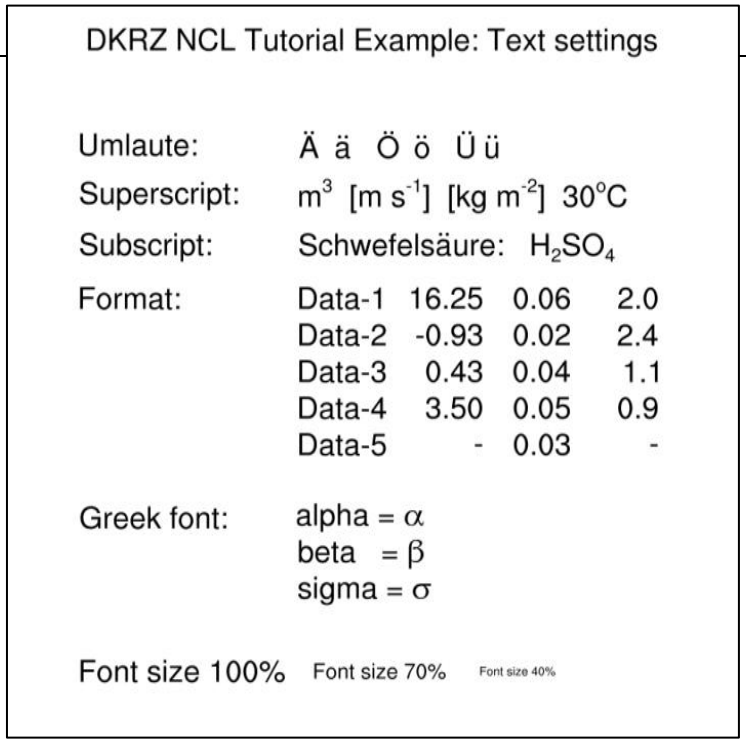

DKRZ NCL Tutorial Example: Title strings
- second line of the title string
- third line of the title string with font size 70%
Left String
Center String
Right String
90N
60N
30N
0
30S
60S
90S
180
150W
120W
90W
60W
30W
0
30E
60E
90E
120E
150E
180
Y-Axis title string
CONTOUR FROM 230 TO 310 BY 10
X-Axis title string

9.19.2 Function Codes for Creating Special Characters

Different text settings such as Umlaut, superscript and subscript:
NCL_Tutorial/scripts/TUT_text_settings.ncl

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

begin
;-- define german "Umlaute", super- and subscript variables
  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~"
  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","text_settings")

;-- x, y start point for writing
  x   = 0.1
  y   = 0.95
  inc = 0.07

;-- text resources
  txres                   = True
  txres@txFontHeightF     = 0.03
  txres@txJust            = "CenterCenter"
  str                     = "DKRZ NCL Tutorial 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
```

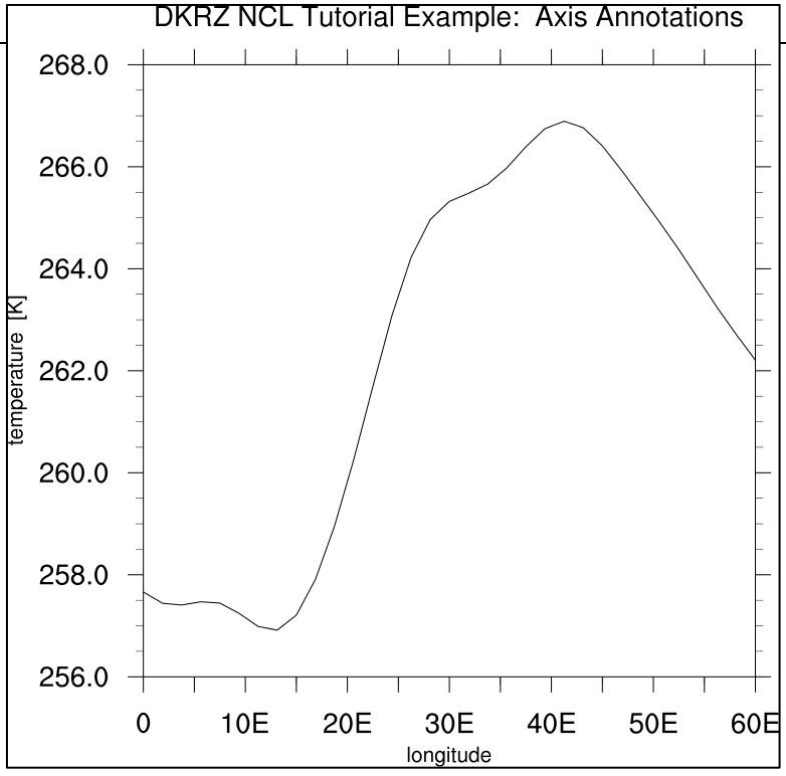
```
    ypos = y-5*inc-i*0.05
    if(diff(i).ne.0.0) then
      str     = sprintf("%6.2f",diff(i))
      gsn_text_ndc(wks,str,xpos,ypos,txres)
    else
      str     = "-"
      gsn_text_ndc(wks,str,xpos,ypos,txres)
    end if
    xpos = xpos + 0.12
    if(var(i).ne.0.0) then
      str     = sprintf("%5.2f",var(i))
      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 = ~F33~a~N~"
  gsn_text_ndc(wks,str2,xpos+0.27,ypos,txres)
  str2                    = "beta   = ~F33~b~N~"
  gsn_text_ndc(wks,str2,xpos+0.27,ypos-0.05,txres)
  str2                    = "sigma = ~F33~s~N~"
  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                    = "~Z70~Font size 70%~N~"
  gsn_text_ndc(wks,str2,xpos+0.3,ypos-3*inc,txres)
  str3                    = "~Z40~Font size 40%~N~"
  gsn_text_ndc(wks,str3,xpos+0.45,ypos-3*inc,txres)

  frame(wks)
end
```



9.19.3 Axis Annotations

Adding units and axis labels to the plot: NCL_Tutorial/scripts/TUT_axis_annotations.ncl

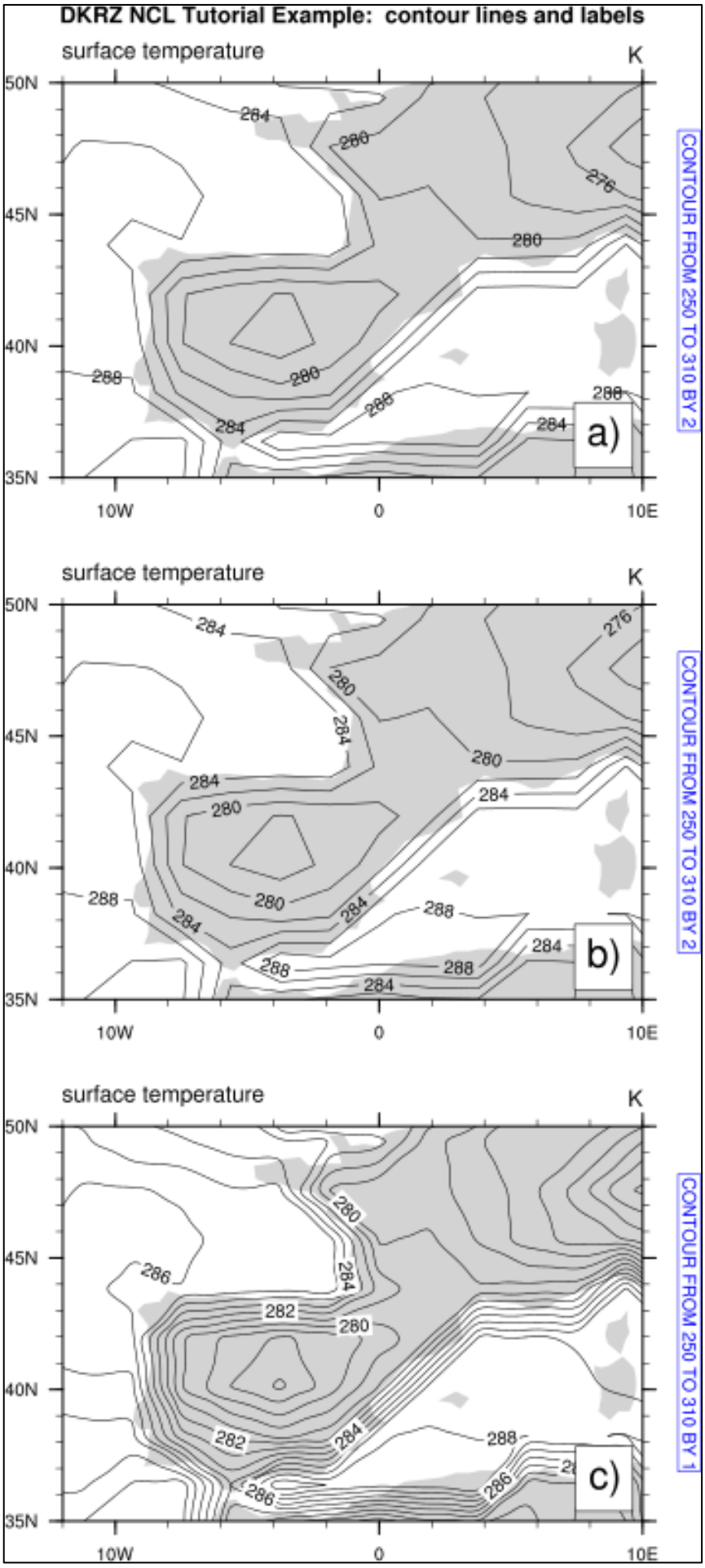
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
;---- read the data and define variable reference var
   f          = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_timestep1_3D.nc","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","axis_annotations")

;---- set resources
   res                         =  True
   res@gsnMaximize             =  True

   res@tiMainString            = "DKRZ NCL Tutorial 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
```



9.19.4 Contour Lines and Label Settings

The next example shows how to change the strings for the contour lines and their labels: $NCL_TUT/TUT_contour_labels.ncl

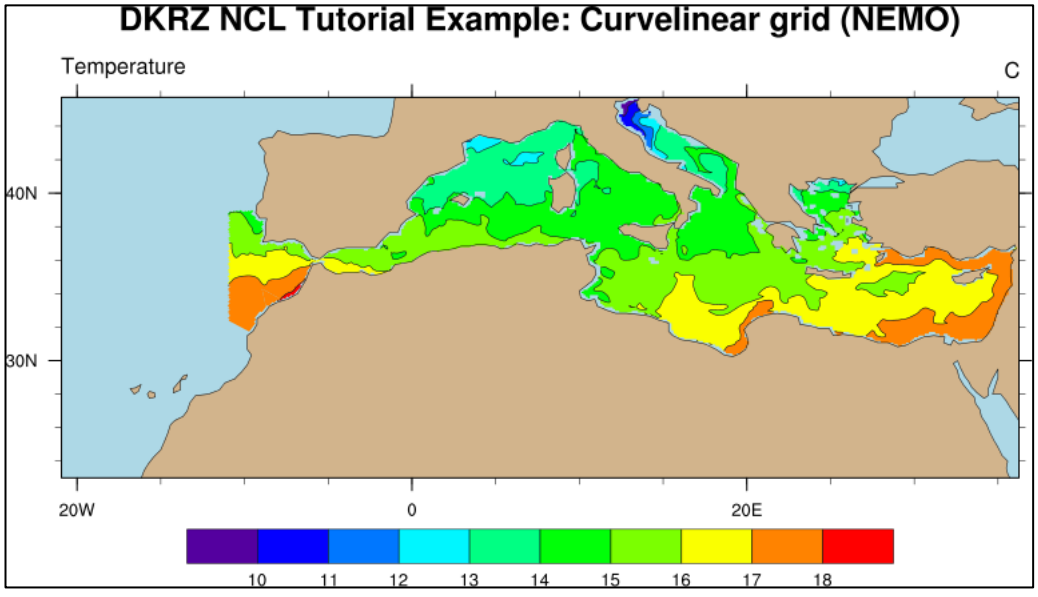
```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"

begin
;---- read the data and define variable reference var
  file1 =  addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_0101-1001_2D.nc","r")
  var   =  file1->tsurf(0,:,:)

;---- define the workstation (plot output type and name)
  wks = gsn_open_wks("png","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@mpMinLatF                 =   35.
  res@mpMaxLatF                 =   50.
  res@mpMinLonF                 =  -12.
  res@mpMaxLonF                 =   10.

  plot = new(3,graphic)        ;-- create 3 empty plots

  res@cnInfoLabelOrthogonalPosF  = -0.88
  res@cnInfoLabelParallelPosF    = 1.1
  res@cnInfoLabelAngleF          = -90.
  res@cnInfoLabelFontColor       = "blue"
  res@cnInfoLabelPerimColor      = "blue"
  res@tiMainString = "DKRZ NCL Tutorial Example:  contour lines and labels"
  res@cnLineLabelPlacementMode   = "computed"
  plot(0) = gsn_csm_contour_map(wks, var, res)

  res@tiMainString               = ""
  res@cnLineLabelPlacementMode   = "constant"
  plot(1) = gsn_csm_contour_map(wks, var, res)

  res@tiMainString               = ""
  delete(res@cnLineLabelPlacementMode)      ;-- set to default: randomized
  res@cnLineLabelBackgroundColor = "white"
  res@cnSmoothingOn              =  True
  res@cnSmoothingTensionF        =  1.
  res@cnSmoothingDistanceF       =  0.005
  plot(2) = gsn_csm_contour_map(wks, var, res)

;-- draw the panel plot
  pres = True
  pres@gsnPanelFigureStrings = (/"a)","b)","c)"/)
  gsn_panel(wks,plot,(/3,1/),pres)

end
```


DKRZ NCL Tutorial Example: contour lines and labels
surface temperature
K
50N
45N
40N
35N
10W
0
10E
a)
b)
c)
CONTOUR FROM 250 TO 310 BY 2
CONTOUR FROM 250 TO 310 BY 2
CONTOUR FROM 250 TO 310 BY 1

9.19.5 Colorizing Land, Ocean and Lakes

Colorize land and ocean areas differently: NCL_Tutorial/scripts/TUT_color_Land_Ocean.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
   f       =  addfile("$NCL_TUT/data/CNTASN_1m_200103_grid_T.nc","r")
   var     =  f->votemper(0,0,:,:)
   lat2d   =  f->nav_lat
   lon2d   =  f->nav_lon

;-- define the workstation (plot type and name)
   wks = gsn_open_wks("X11","color_Land_Ocean")

;-- set resources
   res                         =  True
   res@gsnAddCyclic            =  False
   res@gsnMaximize             =  True
   res@cnFillOn                =  True            ;-- turn on contour fill
   res@tiMainString      = "DKRZ NCL Tutorial Example: Curvilinear grid (NEMO)"
   res@tiMainFontHeightF       =  0.02
   res@mpMinLatF               =  min(lat2d)
   res@mpMaxLatF               =  max(lat2d)
   res@mpMinLonF               =  min(lon2d)
   res@mpMaxLonF               =  max(lon2d)

;-- color land and ocean (looks nicer)
   res@mpOceanFillColor        = "lightblue"
   res@mpInlandWaterFillColor= "lightblue"
   res@mpLandFillColor         = "tan"

;---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
```

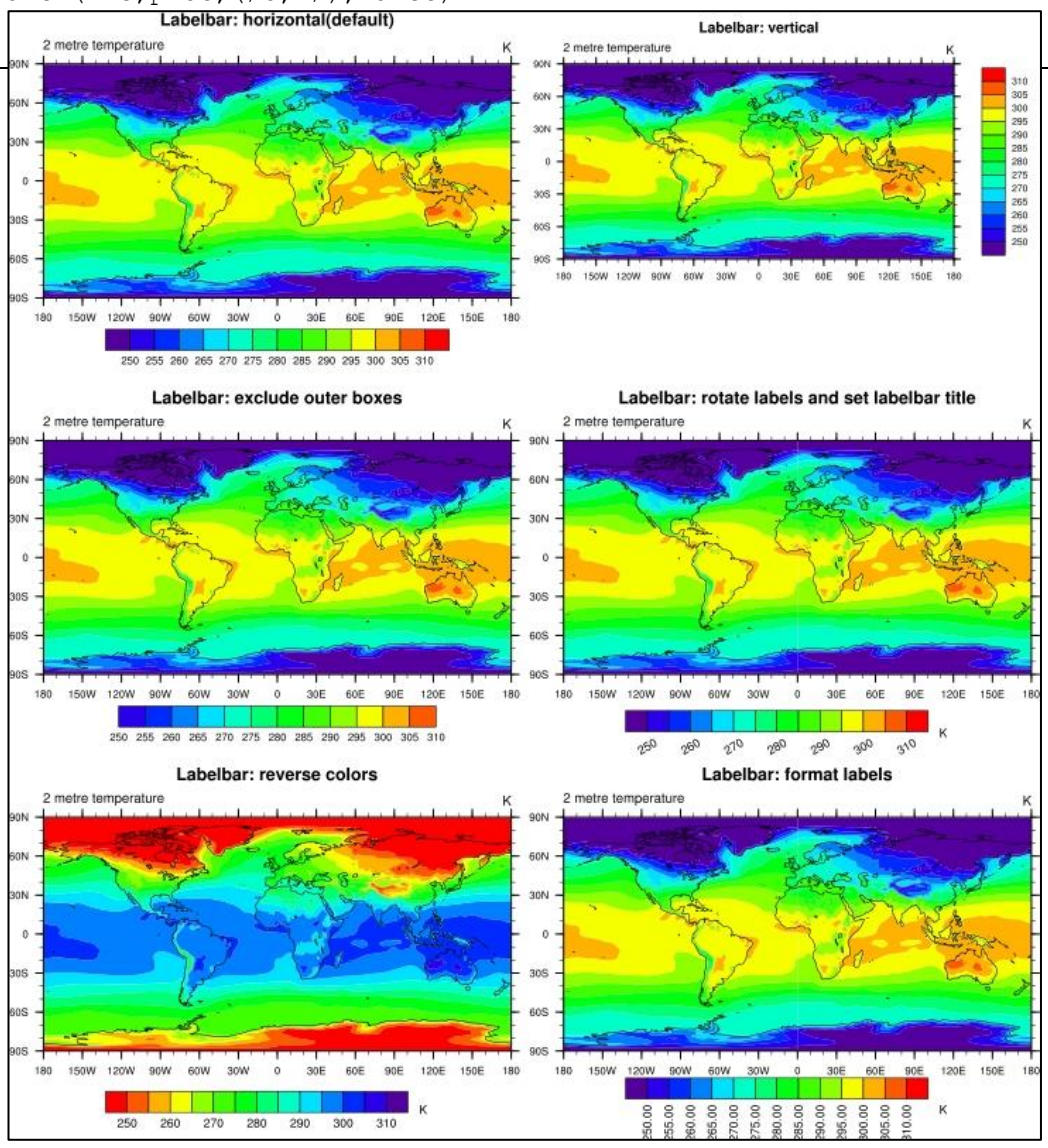


9.19.6 Labelbar Settings

Labelbars are often needed to show the mapping between physical values and colors. This example shows how to achieve different labelbar styles and annotation formats: NCL_Tutorial/scripts/TUT_labelbars.ncl

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

begin
   file1 = "$NCL_TUT/data/ERAIN_SFC00_t2m_ymm_1989-2005.nc"

;-- read the data
   f   =  addfile(file1,"r")
   var =  f->T2M(0,:,:)

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

;-- set plot resources
   res                  = True
   res@gsnDraw          = False
   res@gsnFrame         = False
   res@gsnMaximize      = True
   res@gsnSpreadColors  = True
   res@cnFillOn         = True
   res@cnFillPalette    = "rainbow"
   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)

;-- upper left
   res@tiMainString       = "Labelbar: horizontal(default)"
   plot(0) = gsn_csm_contour_map(wks,var,res)

;-- upper right
   res@tiMainString       = "Labelbar: vertical"
   res@lbOrientation      = "vertical"
   plot(1) = gsn_csm_contour_map(wks,var,res)

;-- middle left
   res@tiMainString       = "Labelbar: exclude outer boxes"
   res@lbOrientation      = "horizontal"
   res@cnLabelBarEndStyle = "ExcludeOuterBoxes"
   plot(2) = gsn_csm_contour_map(wks,var,res)

;-- middle right
   res@tiMainString       = "Labelbar: rotate labels and set labelbar title"
   res@lbTitleOn          =  True
   res@lbTitleString      = "K"
   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)

;-- lower left
   res@tiMainString        = "Labelbar: reverse colors"
   res@gsnSpreadColorStart =  -1
   res@gsnSpreadColorEnd   =  2
   delete(res@lbLabelAngleF)
   plot(4) = gsn_csm_contour_map(wks,var,res)

;-- lower right

   getvalues plot@contour
     "cnLevels" : levels
   end getvalues

   res@lbLabelStrings      =  sprintf("%3.2f",levels)
   res@lbLabelAngleF       =  90
   delete(res@gsnSpreadColorStart)
   delete(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
```



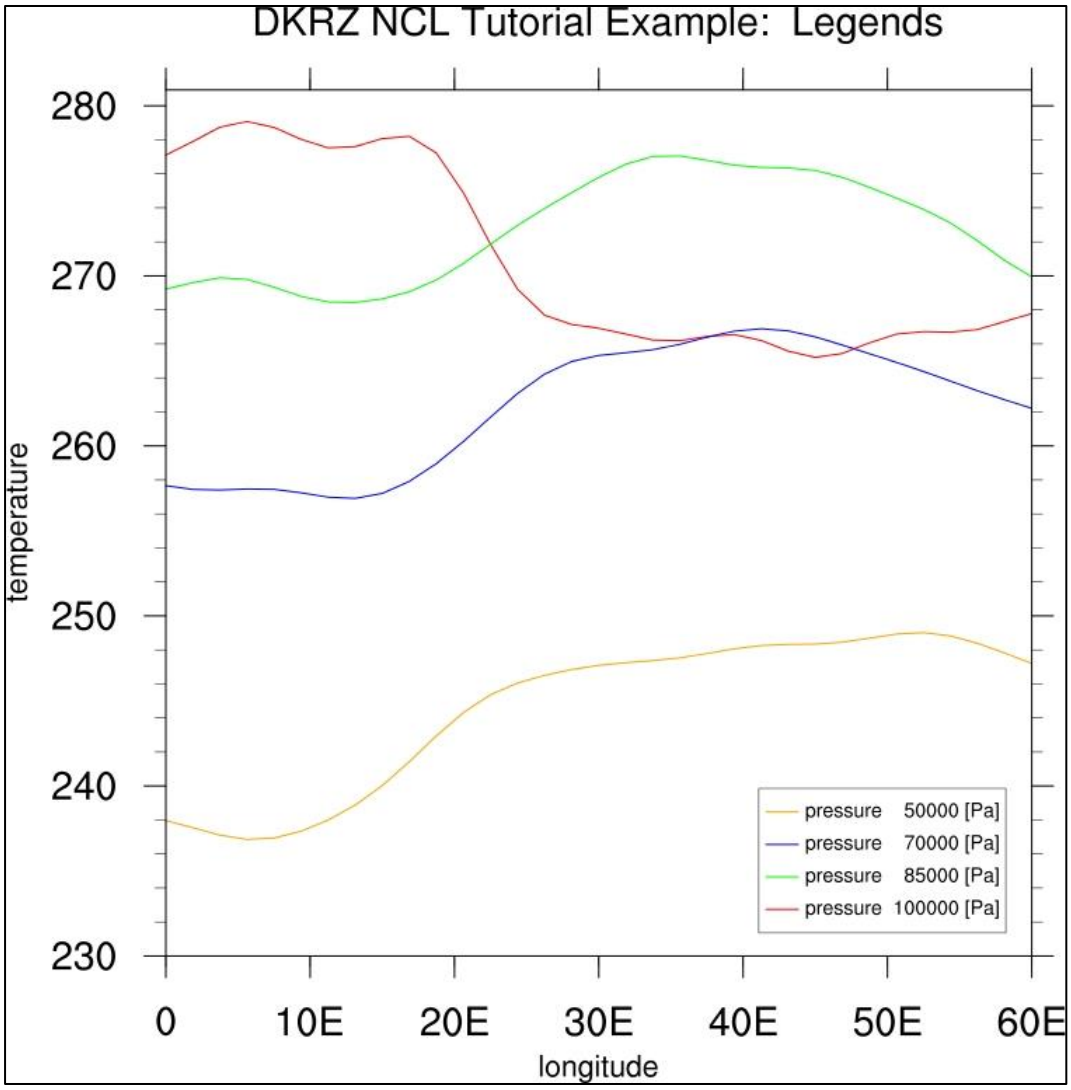
9.19.7 Legend Settings

For line plots with different line types or colors, legends are required to annotate the respective meanings of the different lines.

Legends example: NCL_Tutorial/scripts/TUT_legends.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
   f        = addfile("$NCL_TUT/data/ECHAM5_OM_A1B_2001_timestep1_3D.nc","r")
   var        = f->t(0,:,{55},{0:60})

   lon_t      = f->lon({0:60})              ;-- longitude=0-60E
   lev_t      = f->lev                      ;-- currently 17 levels

;-- define the data to be plotted
   levels  = new((/4,dimsizes(lon_t)/),float)

   levels(0,:) =  f->t(0,{100000},{55},{0:60})
   levels(1,:) =  f->t(0, {85000},{55},{0:60})
   levels(2,:) =  f->t(0, {70000},{55},{0:60})
   levels(3,:) =  f->t(0, {50000},{55},{0:60})

   name   = lev_t@long_name                 ;-- retrieve the long_name
   units  = lev_t@units                     ;-- retrieve the units

;-- define the colors of the lines and their labels
   colors = (/"red", "green", "blue", "orange"/)
   labels = (/ " "+name+"  100000 ["+units+"]", \
               " "+name+"    85000 ["+units+"]",\
               " "+name+"    70000 ["+units+"]", \
               " "+name+"    50000 ["+units+"]"/)

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

;---- set resources
   res                         =  True
   res@gsnMaximize             =  True

   res@trYMinF                 =  230
   res@trYMaxF                 =  max(var)

;-- set the title string. ~C~ insert a carriage return (no \ allowed).
   res@tiMainString            = "DKRZ NCL Tutorial Example:  Legends"
   res@tiMainFontHeightF       =  0.02
   res@tiXAxisString           =  lon_t@long_name
   res@tiYAxisString           =  var@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@lgLabelFontHeightF      =  0.01
    res@lgBoxMinorExtentF       =  0.16          ;-- legend lines shorter
    res@pmLegendDisplayMode     = "Always"
    res@pmLegendWidthF          =  0.15          ;-- set legend width
    res@pmLegendHeightF         =  0.1           ;-- set legend height
    res@pmLegendOrthogonalPosF = -0.34           ;-- move legend up
    res@pmLegendParallelPosF    =  0.97          ;-- move legend right

    plot = gsn_csm_xy(wks,lon_t,levels,res)

end
```



9.19.8 Date Format

The time steps of a data in a netCDF file are commonly stored like

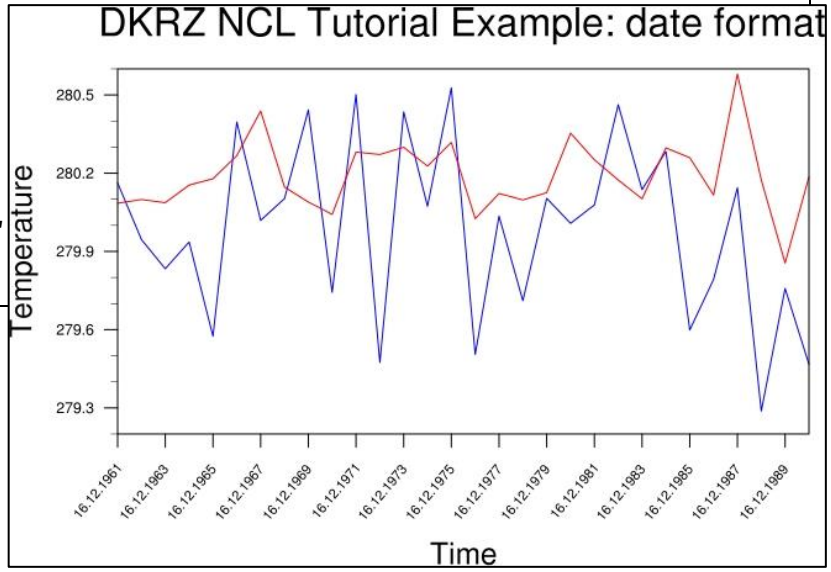
```
double time(time) ;
        time:standard_name = "time" ;
        time:units = "hours since 2001-01-01 00:00:00" ;
        time:calendar = "standard" ;
```

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 shouldn’t use the 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: NCL_Tutorial/scripts/TUT_date_format.ncl

```
undef ("getDate")
function getDate(time)
;----------------------
begin
   ;-- convert the time proleptic_gregorian calendar in UTC date

   utc_date = cd_calendar(time, 0)
   ;-- returns a utc_date variable which holds the following
   ;-- time elements of the time variable

   ;-- get the time elements from the utc_date variable
   year   = tointeger(utc_date(:,0))
   month  = tointeger(utc_date(:,1))
   day    = tointeger(utc_date(:,2))
   hour   = tointeger(utc_date(:,3))
   minute = tointeger(utc_date(:,4))
   second = utc_date(:,5)

  ;-- write date as string (DD.MM.YYYY) using the variables year,
  ;-- month and day
   date_str_i = sprinti("%0.2i",day) + "." + sprinti("%0.2i",month) + \
                "."  + sprinti("%0.4i",year)
   return(date_str_i)  ;-- return the readable date string
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
  system("cdo -s -r -f nc yearmean $NCL_TUT/data/tas_AFR-44_MPI-ESM-
LR_historical_r1i1p1_CCLM_4-8-17_mon_1961-1990.nc tser_tmp.nc")

  f1 = addfile("tser_tmp.nc","r")
  var                        = f1->tas(:,0,0,0)
  time                       = var&time
  timax                      = dimsizes(time) - 1

  f2 = addfile("$NCL_TUT/data/tas_AFR-44_MPI-ESM-LR_historical_r1i1p1_CCLM_4-
8-17_mon_1961-1990.nc","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) and colormap
  wks  = gsn_open_wks("png", "TUT_date_format")

;-- set resources
  res                         =  True
  res@gsnDraw                 =  False
  res@gsnFrame                =  False

  res@xyLineColor             = "blue"
  res@xyLineThicknessF        =  2
  res@xyDashPattern           =  0
  res@vpXF                    =  0.25
  res@vpYF                    =  0.6
  res@vpWidthF                =  0.7
  res@vpHeightF               =  0.37
  res@tiMainString            = "DKRZ NCL Tutorial 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@tmXBLabels              =  labels             ;-- write readable dates
  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
```



10 Regridding data with CDOs and ESMF

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, regiridded data files (see also 3.4). 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:

```
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, NCL ESMF)
- patch (NCL ESMF)
- conservative (CDOs, NCL ESMF)
- bicubic (CDOs)
- distance weighted (CDOs)
- nearest neighbor (CDOs, NCL 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

CDOs: http://www.mpimet.mpg.de/fileadmin/software/cdo/cdo.pdf

10.1 ESMF Bilinear Regridding

This example demonstrates the use of the ESMF regridding function to interpolate a global CMIP5 grid into a regular global 1x1^{o} grid.

NCL_Tutorial/scripts/TUT_ncl_regrid_bilinear_CMIP5_grid_to_1x1deg_grid.ncl

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

begin
    t1 = systemfunc("date +%s")

    fname = "$NCL_TUT/data/thetao_Omon_MPI-ESM-LR_piControl_r1i1p1_185001.nc"
    sfile        =  addfile(fname,"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 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
    Opt@SrcMask2D      =  where(.not.ismissing(thetao),1,0)
    Opt@DstGridType    = "1x1"                          ;-- Destination grid
    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 global 1x1 degree grid
    print("---------------------------------------------------------------------
")
    print("Generating interpolation weights from CMIP5 to World 1x1 degree
grid.")
    print("   Method: bilinear")
    print("---------------------------------------------------------------------
")

    thetao_regrid = ESMF_regrid(thetao,Opt)
    printVarSummary(thetao_regrid)
;-- write regridded data to file
    system("rm -rf regrid_bilinear_CMIP5_thetao.nc")
    fout = addfile("regrid_bilinear_CMIP5_thetao.nc", "c")
    fout->thetao = thetao_regrid

;-- compute the elapsed time
    t2   = systemfunc("date +%s")
    t1_i = toint(t1)
    t2_i = toint(t2)
    te   = t2_i - t1_i
    print("Elapsed time:  "+ te + "s")

end
```

The elapsed time: 6s

10.2 Bilinear Regridding with CDO

This example demonstrates the use of the CDO remapping function to interpolate a global CMIP5 grid into a regular world grid 1x1^{o} 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 *weights_bil.nc*, which will be used to interpolate the data.

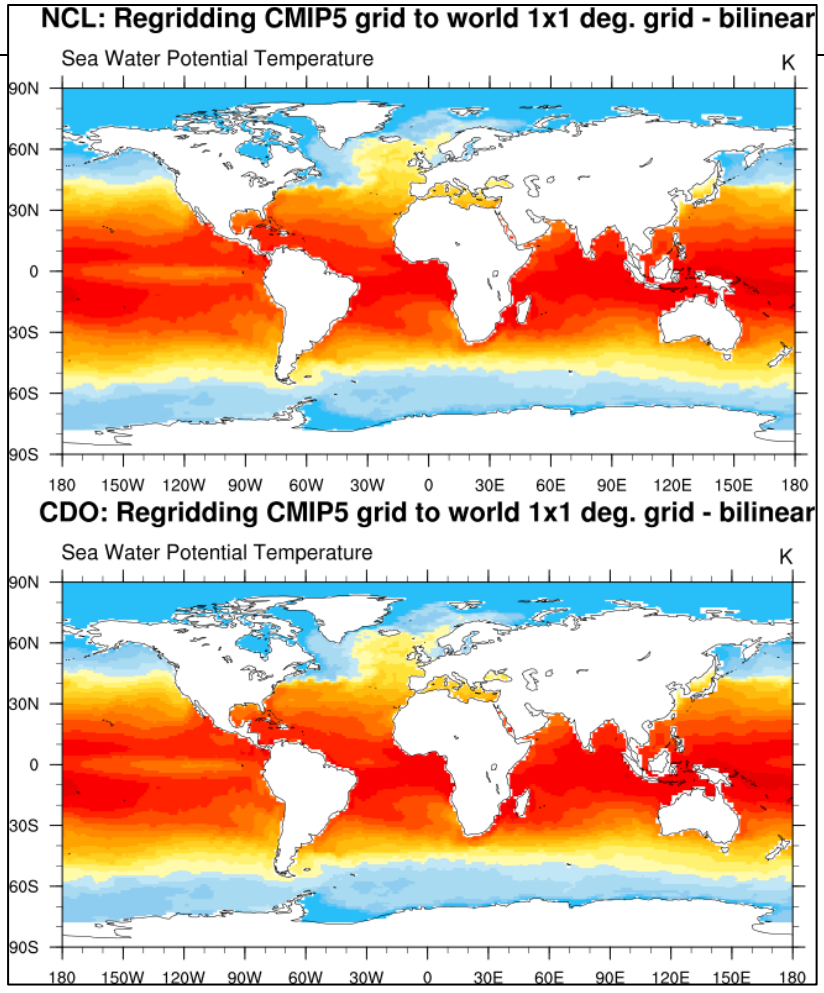
The next command line will generate a bilinear weight file, but only in memory, not saved on the disk:

```
cdo remapbil,r360x180 ${fin} ${fout}
```

NCL_Tutorial/scripts/TUT_cdo_remap_bilinear_CMIP5_grid_to_1x1deg_grid.ksh

```
#!/usr/bin/ksh

t1=$(date +%s)

fin="$NCL_TUT/data/thetao_Omon_MPI-ESM-LR_piControl_r1i1p1_185001.nc"
fout="remap_bilinear_CMIP5_thetao.nc"
wgts="weights_bilinear.nc"

cdo genbil,r360x180 ${fin} ${wgts}
cdo remap,r360x180,${wgts} ${fin} ${fout}

t2=$(date +%s)
td=$(expr $t2 - $t1)
echo "Elapsed time:  ${td}s"

exit
```

The elapsed time: 55s



11 Using External Fortran or C-Code

NCL supports invoking 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

11.1 Fortran

In this example, the korn shell script *TUT_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: NCL_Tutorial/scripts/TUT_use_Fortran_subroutines.ksh

```
#!/usr/bin/ksh
#------------------------------------------------------------
#-- DKRZ NCL Tutorial example script:  TUT_use_Fortran_subroutines.ksh
#--
#--     Write a short Fortran subroutine --> ex01.f
#--     Run the wrapper                  --> ex01.so
#--     Write NCL script                 --> TUT_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
EOF
```

```
#-------------------------------------------------------------
#-- fortran77 code
#-------------------------------------------------------------
cat<<EOF> ex01.f
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) 1995-2013 - All Rights Reserved
University Corporation for Atmospheric Research
NCAR Command Language Version 6.1.2
The use of this software is governed by a License Agreement.
See http://www.ncl.ucar.edu/ for more details.

Polynomial value = 0
Polynomial value = 3
Polynomial value = 4
Arc length = 2.82843
```

11.2 C-Code

The same exampled 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. Step write the C-code
2. Step create a Fortran stub file with the calling sequences and types
3. Step run "wrapit77" on the Fortran stub file to create the C wrapper
4. Step Make some changes in the C wrapper file
5. Step run "WRAPIT" with the '-d' option to get compiling informations
6. Step create a Makefile to compile the C-code and create the shared library
7. Step use the external subroutine and functions within NCL

All steps are done by the following Korn-Shell script.

C Wrapper example: NCL_Tutorial/scripts/TUT_use_C_subroutines.ksh

```
#!/bin/ksh
#-------------------------------------------------------------
#-- DKRZ NCL Tutorial example script:  TUT_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                 --> TUT_use_C_subroutines.ncl
#--     Run NCL script                   --> write results to stdout
#--
#-- 23.07.13
#-------------------------------------------------------------
example=${0%.*}
ncl_script=${example}.ncl

#-- clean up
rm -rf ex01C.c ex01C.o ex01C.c~ ex01C.stub ex01CW.c ex01CW.o WRAPIT.stub
rm -rf WRAPIT.c WRAPIT.o ex01C.so TUT_use_C_subroutines.ncl objects
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
/* http://www.ncl.ucar.edu/Document/Tools/WRAPIT.shtml#Example_6 */
#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);
}
EOF

echo "---------------------------------"
echo "-- write C code            - done"

#-------------------------------------------------------------
#-- 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 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 cquad_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 "---------------------------------"
ncl -n ${ncl_script}

exit
```

Result on stdout:

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

Polynomial value = 0
Polynomial value = 3
Polynomial value = 4
Arc length = 2.82843
```

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, e.g. using the modules function

wizard.dkrz.de: set by `'module load ncl/6.1.2-gccsys'`

NCARG_ROOT=/sw/centos58-x64/ncl-6.1.2-gccsys

Here some of the important NCL environment variables which can be changed in order to customize your NCL environment:

NCARG_ROOT	parent directory of NCL installation default: */usr/local*
NCARG_NCARG	location of supplemental directories, like databases, examples, resource files, etc. default: *$NCARG_ROOT/lib/ncarg*
NCARG_USRRESFILE	NCL HLU resource file default: *~/.hluresfile*
NCARG_COLORMAP_PATH	list of paths of colormap files default: *$NCARG_NCARG/colormaps*
NCARG_RANGS	point to the RANGS/GSHHS database, if installed default: $NCARG_NCARG/rangs
NCARG_EXAMPLES	point to the low-level examples Default: *$NCARG_NCARG/examples*
NCL_GRIB_PTABLE_PATH	point to GRIB parameter table file default: *none*
NIO_GRIB2_CODETABLES	location of the GRIB2 code tables default: *$NCARG_ROOT/lib/ncarg/grib2_codetables*

All NCL environment variables are listed on the web page:

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

13 Creating Images for PowerPoint, Keynote, Web

For importing NCL visualizations to PowerPoint, Keynote or web pages, it is useful to use the output format ‘png’. For printable documents you should select larger width and height values to get a better resolution:

```
wkstype          =  "png"
wkstype@wkWidth  =  2500
wkstype@wkHeight=  2500
wks = gsn_open_wks(wkstype, "plot_file_name")
```

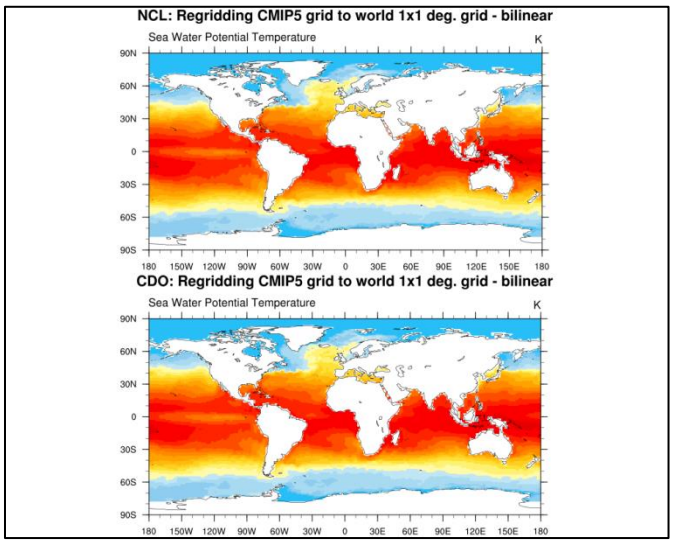
Sometimes the output format "png" won’t produce good results and it would be better to use PS or PDF output format. The PS or PDF file can be converted using the ImageMagick software package, which is installed on most machines.

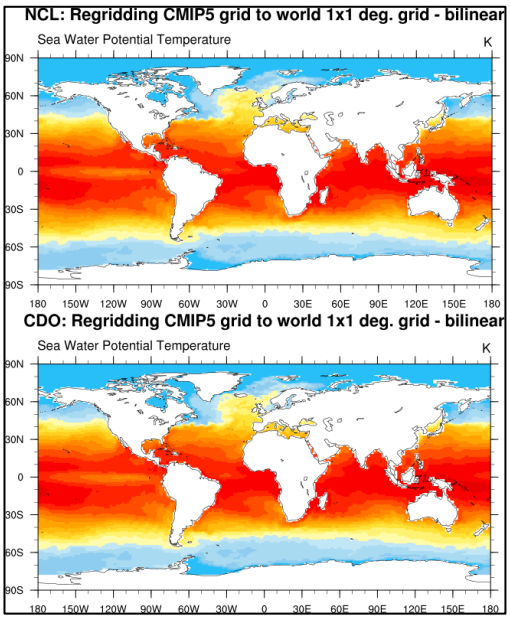
```
convert -geometry 2500x2500 -density 300 -trim plot.ps plot.png
```

For posters you have to increase the values for geometry and density.

To crop white space from the plot:

```
convert -alpha off -background white -geometry 1000x1000 \
                   -density 300 -trim plot.ps plot.png
```





14 Tips & Common Mistakes

The first rule of NCL in avoiding errors and warnings is (as Dennis Shea would say):

Pay attention to 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.

14.1 Errors and Warnings

Common error messages:

```
(0)   gsn_csm_contour_map_ce: Fatal: the input data array must be 1D or 2D
fatal:Illegal right-hand side type for assignment
```

Solution: ***Look at your data!*** How many dimensions does your variable have?

e.g. `float tsurf ( time, lev, lat, lon )`

you have to choose one timestep and one level to use the gsn_csm_contour_map function to create the plot

`tsurf(0,0,:,:)`

```
(0) 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.
```

Solution: set *res@gsnAddCyclic = False*

```
(0) 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.
(0) 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.
```

Solution: add the ***units*** attribute to the ***lat*** and ***lon*** coordinate arrays

```
(0) warning:ScalarFieldSetValues: 2d coordinate array sfXArray has an incorrect dimension
size: defaulting sfXArray
(0) warning:ScalarFieldSetValues: 2d coordinate array sfYArray has an incorrect dimension
size: defaulting sfYArray
```

Solution: try setting *res@gsnAddCyclic = False*

```
warning:tmXBStride is not a valid resource in ex07-1_xy at this time
```

Solution: in this case, if the resource is misspelled, it should be *tmXBLabelStride*

14.2 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:

```
latitudes = var&lat(::-1)
```

Pre-conditions: variable ***var*** has a named dimension called ***lat***

14.3 Reading Excel CSV Data Files

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

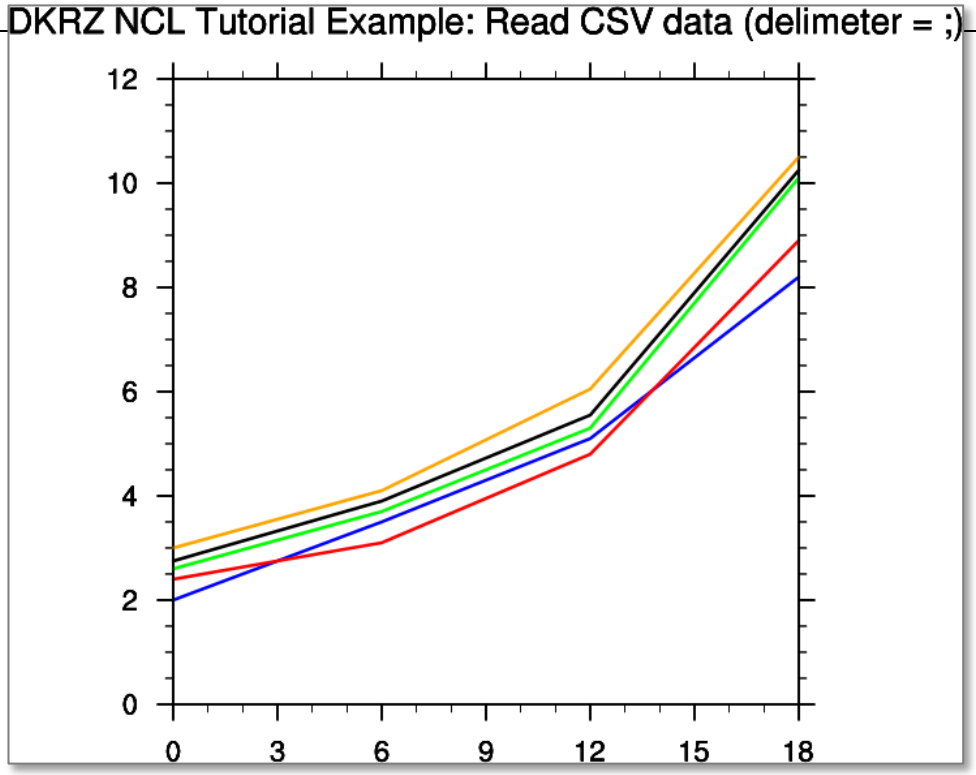
Example: the data file Test_6h.csv contains a value for 0h,6h,12h,18h per line:

```
2.00;3.50;5.10;8.20
2.40;3.10;4.80;8.90
2.60;3.70;5.30;10.10
2.75;3.90;5.55;10.25
3.00;4.10;6.05;10.50
```

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

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

begin

  fname = "$NCL_TUT/data/Test_6h.csv"

;-- read in file as array of strings so we can parse each line
  delim  = ";"

;-- read the data and count the fields (=columns)
  data   = asciiread(fname, -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 j=1,scount
     val(:,j-1) = tofloat(str_get_field(data,j,delim))
  end do
  print("Val: " + val)

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

;-- open a workstation
  wks = gsn_open_wks("X11","plot_csv")

;-- set resources
  res                         =  True
  res@gsnDraw                 =  False
  res@gsnFrame                =  False
  res@tiMainString            = "Read CSV data (delimeter = ;)"
  res@xyLineThicknessF        =   5
  res@trYMinF                 =   0.0
  res@trYMaxF                 =  12.0
  res@trXMinF                 =   0
  res@trXMaxF                 =  18

;-- assign emty plot array
  plot = new(nl,graphic)
```

```
;-- define line color
  color = (/"blue", "red", "green", "black", "orange"/)

;-- generate the plots
  do l=0,nl-1
    y = val(l,:)
    res@xyLineColor   =  color(l)
    plot(l) = gsn_csm_xy(wks, x, y, res)
  end do

;-- draw the plot and advance the frame
  draw(plot)
  frame(wks)

end
```



14.4 Exporting a 2-dimensional Array to new ASCII File (CSV)

The following question was asked in the ***ncl-talk*** mailing list:

"How can a 2-dimensional array be printed as comma-separated-values (CSV)?"

Example 2 of the **write_table** function in NCL 6.1.0 is one approach. Another approach: do your own line formatting.

```
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 spread sheet software. For single spaces between numbers, use " %0.2f,".

15 List of Example Scripts

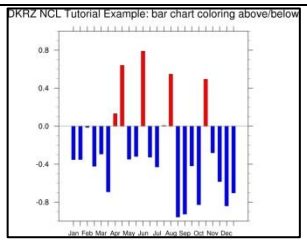
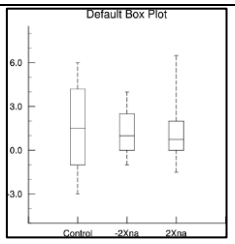
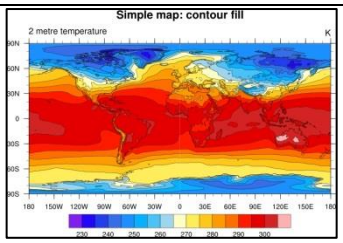
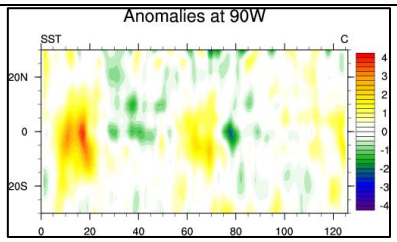
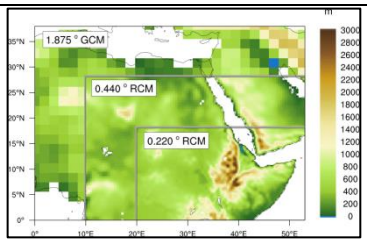
Chapter	Example	Script File Name
9.1	NCL Graphics - in 5 steps	TUT_plot_in_5_steps.ncl
9.2	XY-plot	TUT_xy_plot.ncl
9.2	XY-plot changing the defaults	TUT_xy_plot_res.ncl
9.3	XY-plot time series	TUT_xy_plot_timeseries.ncl
9.3	Time series of multi data sets	TUT_multi_timeseries.ncl
9.4	Contour map	TUT_contour_map.ncl
9.4.1	Contour filled map	TUT_contour_filled_map.ncl
9.5	Map settings	TUT_maps_settings.ncl
9.5.3	Polar plot Northern Hemisphere	TUT_polar_NH.ncl
9.6	Map resolutions	TUT_maps_resolution.ncl
0	Vector using defaults	TUT_vector_default.ncl
0	Vectors curly look	TUT_vector_curly.ncl
0	Vectors colorized	TUT_vector_plot_colorized.ncl
0	Vectors overlay on contour map	TUT_vector_plot_overlay.ncl
9.8	Slice plot	TUT_slice_plot.ncl
9.9	Bar Charts	TUT_bar_chart.ncl
9.9	Bar Charts of multi data sets	TUT_bar_chart_multi.ncl
9.9	Bar Charts using a reference line	TUT_bar_chart_col_above_below.ncl
9.10	Overlays	TUT_transparency_filled_contour.ncl
9.10	Grid resolutions comparison	TUT_grid_resolution_comparison.ncl
9.11	Panels plot	TUT_panel_plot.ncl
9.11	Panel plot 3 rows and 2 cols	TUT_panel_3x2_plot.ncl

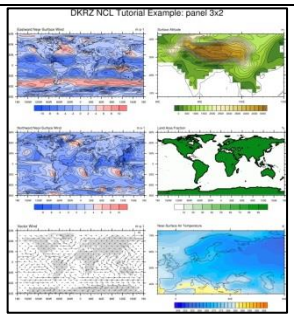
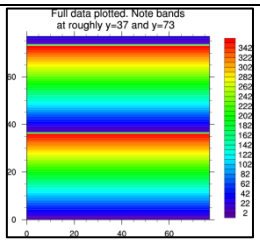
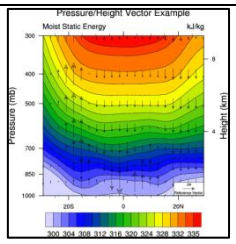
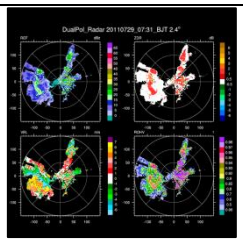
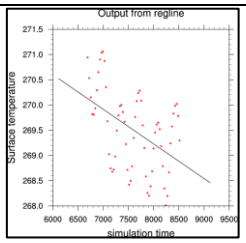
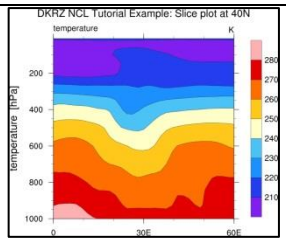
9.13	Shapefile plot	TUT_shapefile_plot.ncl
9.14	Colormap definition	TUT_colormaps.ncl
9.14.1	Convert GrADS coltab to NCL colmap	grads2ncl_coltab.ksh
9.14.2	Convert GMT coltab to NCL colmap	gmt2ncl_coltab.ksh
9.15	Title strings	TUT_title_strings.ncl
9.15	Axis labels	TUT_axis_annotations.ncl
9.15	Labelbars	TUT_labelbars.ncl
9.15	Legends	TUT_legends.ncl
9.15	Date format	TUT_date_format.ncl
9.16	Curvelinear grid	TUT_curvelinear_grid.ncl
9.16	Unstructured grid	TUT_unstructured_grid.ncl
9.16.1	Bipolar grid – MPI-ESM-LR	TUT_bipolar_grid_MPI-ESM.ncl
9.16.2	Tripolar grid STORM	TUT_tripolar_grid_STORM.ncl
9.17.1	Tripolar grid ICON	TUT_tripolar_grid_ICON.ncl
9.19.1	Title string settings	TUT_title_strings.ncl
Fehler! erweisquelle konnte nicht gefunden werden.	Text settings	TUT_text_settings.ncl
9.19.3	Axis annotations	TUT_axis_annotations.ncl
9.19.6	Labelbar settings	TUT_labelbars.ncl
9.19.7	Legend settings	TUT_legends.ncl
9.19.8	Date formats	TUT_date_format.ncl
10.1	NCL regrid bilinear	TUT_regrid_bilinear_CMIP5_grid_to_1x1deg_grid.ncl
10.2	CDO remap bilinear	TUT_remap_bilinear_CMIP5_grid_to_1x1deg_g

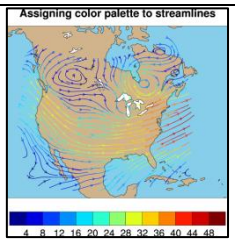
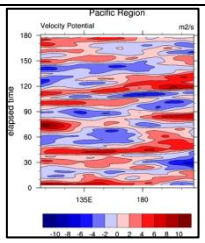
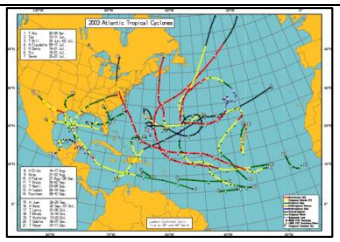
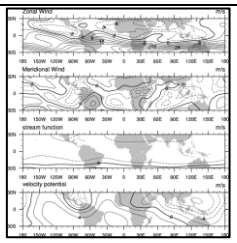
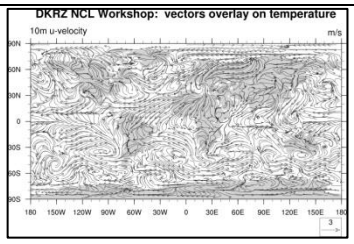
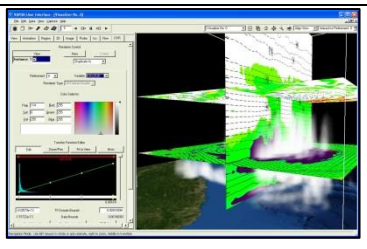
		rid.ksh

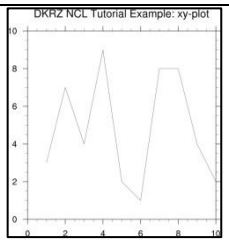
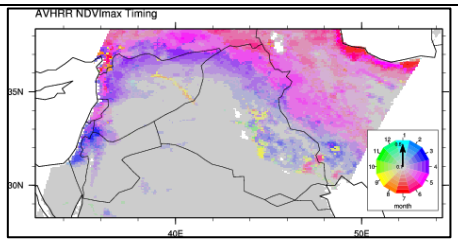
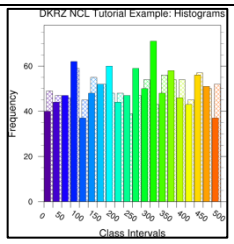
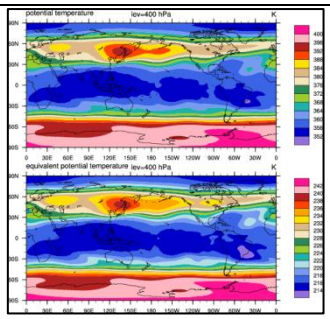
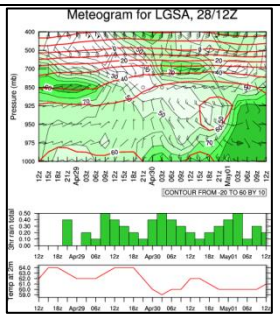
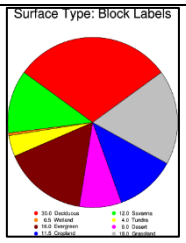
16 Appendix A - Plot Types

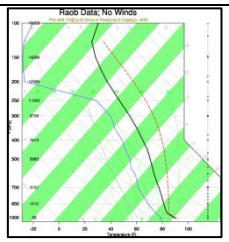
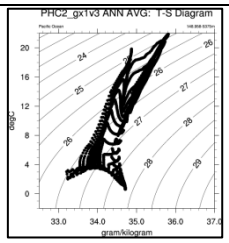
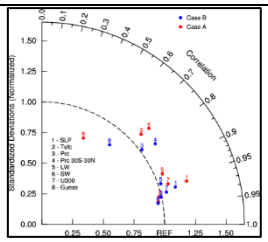
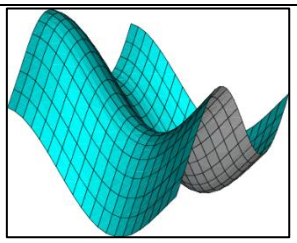
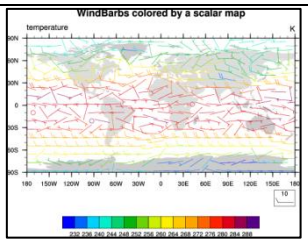
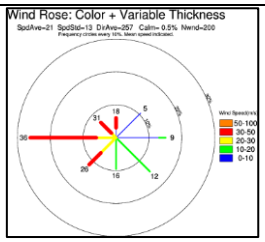
In this tutorial we have not covered the full variety of different plot types possible with ncl. However, here we give a more comprehensive listing which might inspire you to dig deeper into ncl's capabilities.

Bar charts	
Box plots	 *
Contours	
Latitude vs. Time	 *
Overlay plots	

Panel plot	
Phase plots	 *
Press/height vs. latitude/longitude/time	 *
Radar (r,theta) plots	 *
Scatter plots	 *
Slices	

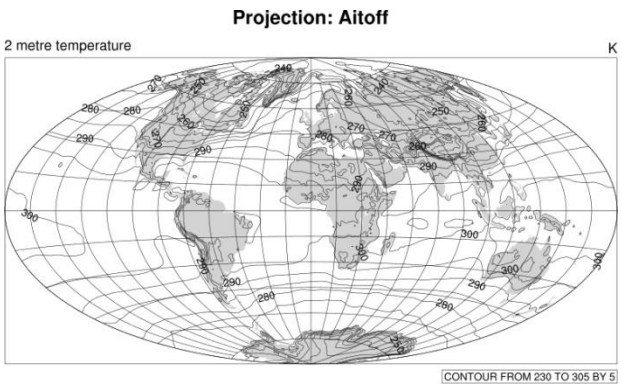
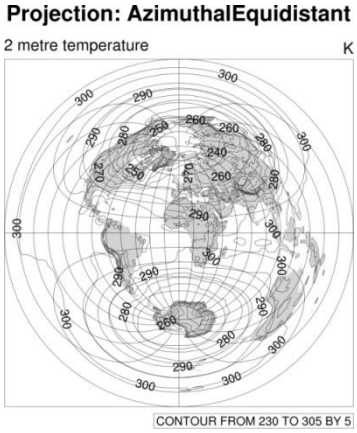
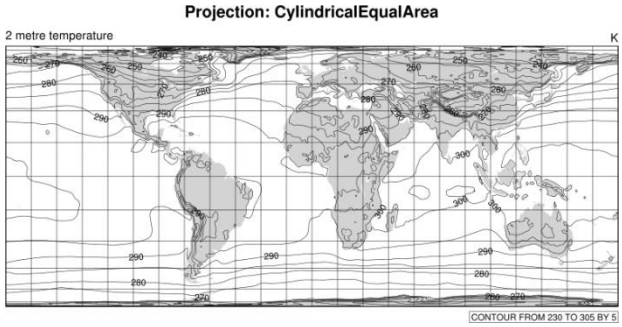
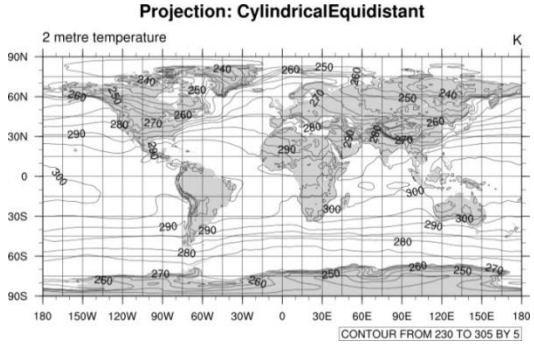
Streamlines	 *
Time vs. longitude/latitude	 *
Trajectories	 *
Tropical strip plots	 *
Vectors	
WRF-VAPOR (3D) using Vapor	 *

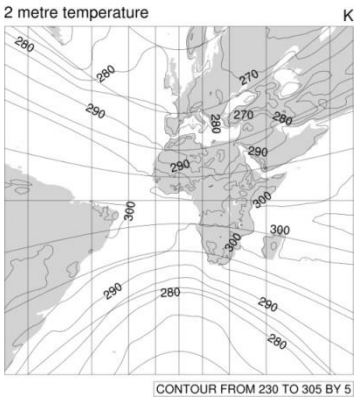
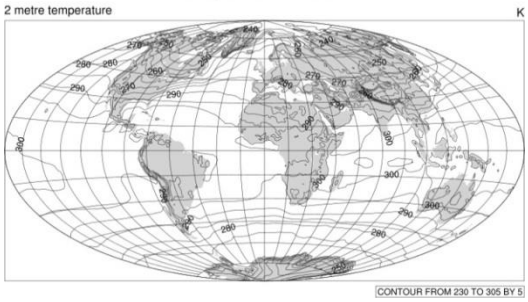
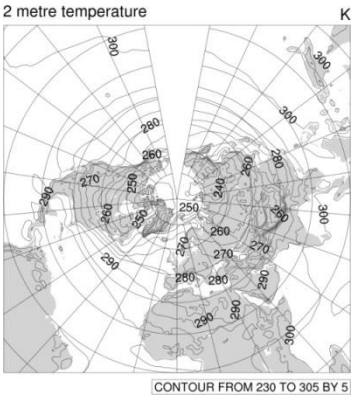
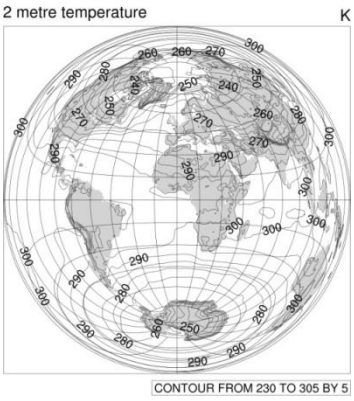
XY plots	
Evans plots	 *
Histograms	
Iso levels	 *
Meteograms	 *
Pie charts	 *

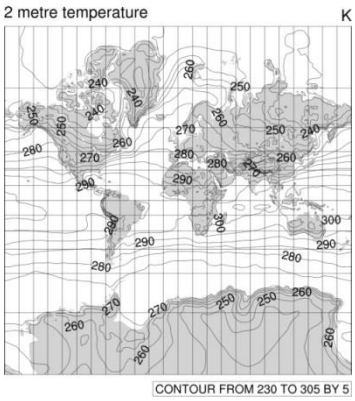
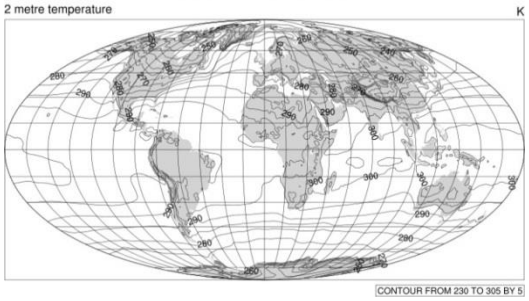
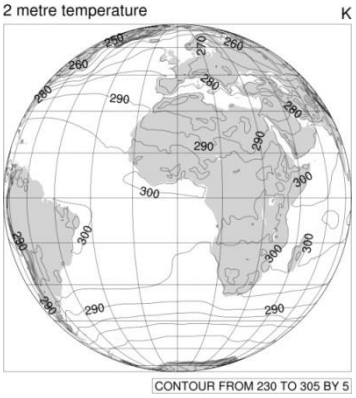
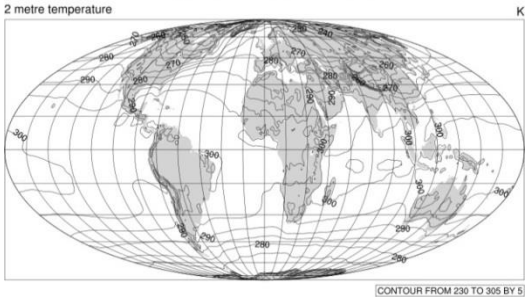
Skew-T	 *
T-S diagram	 *
Taylor diagram	 *
3D plots (TDPACK)	 *
Wind barbs	 *
Wind rose	 *

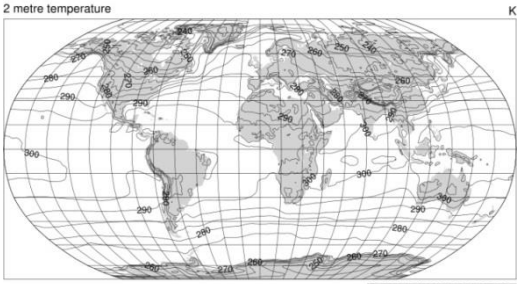
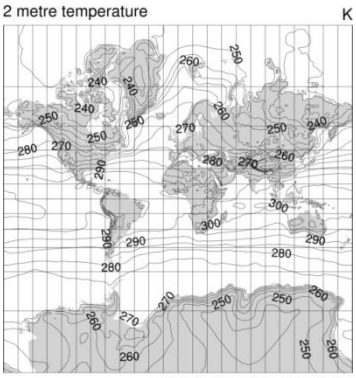
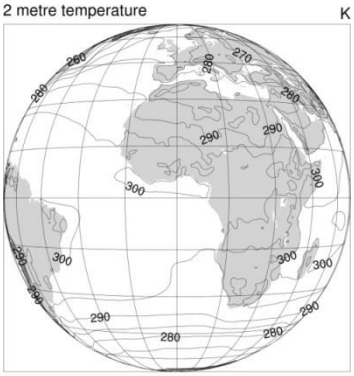
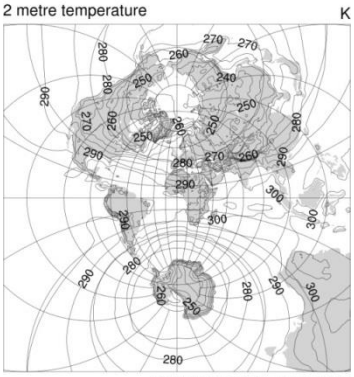
(* plots from the NCL web page - all examples - http://www.ncl.ucar.edu/Applications/)

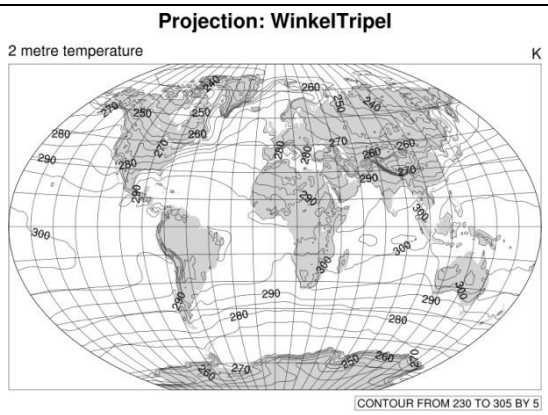
17 Appendix B - Projections

Aitoff	
AzimuthalEquidistant	
CylindricalEqualArea	
CylindricalEquidistant (default)	

Gnomonic	**Projection: Gnomonic** 
Hammer	**Projection: Hammer** 
LambertConformal	**Projection: LambertConformal** 
LambertEqualArea	**Projection: LambertEqualArea** 

Mercator	**Projection: Mercator** 
Mollweide	**Projection: Mollweide** 
Orthographic	**Projection: Orthographic** 
PseudoMollweide	**Projection: PseudoMollweide** 

Robinson	**Projection: Robinson** 2 metre temperature K  CONTOUR FROM 230 TO 305 BY 5
RotatedMercator	**Projection: RotatedMercator** 2 metre temperature K  CONTOUR FROM 230 TO 305 BY 5
Satellite	**Projection: Satellite** 2 metre temperature K  CONTOUR FROM 230 TO 305 BY 5
Stereographic	**Projection: Stereographic** 2 metre temperature K  CONTOUR FROM 230 TO 305 BY 5

WinkelTripel	

To see how to generate all these projection plots take a look at the example script NCL_Tutorial/scripts/TUT_projections.ncl.

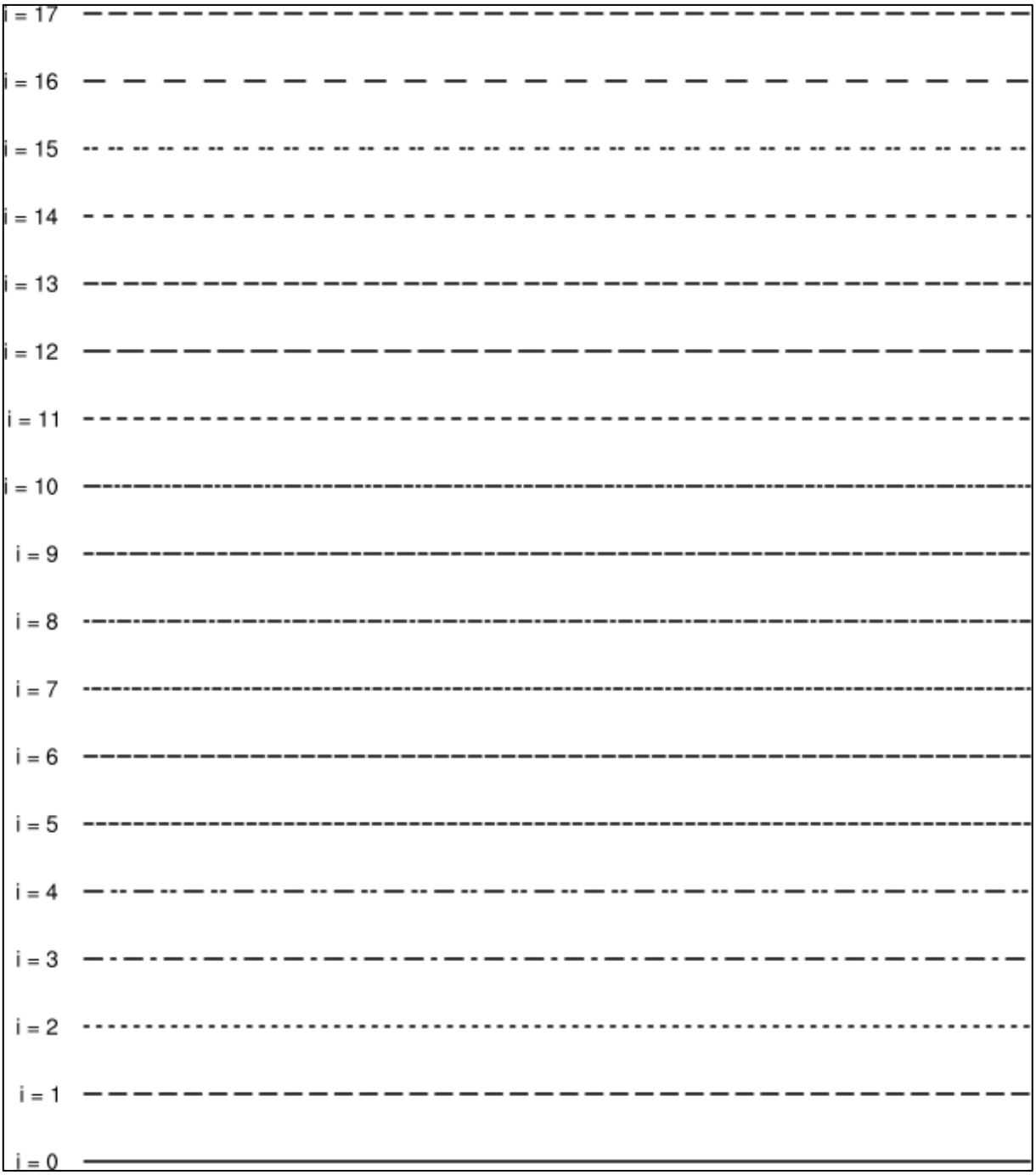
To change the map projection the user has to set it with the following resource, e.g. to use Mollweide projection:

```
res                     = True
res@mpProjection        = "Mollweide"
```

18 Appendix C - Dash Pattern

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.

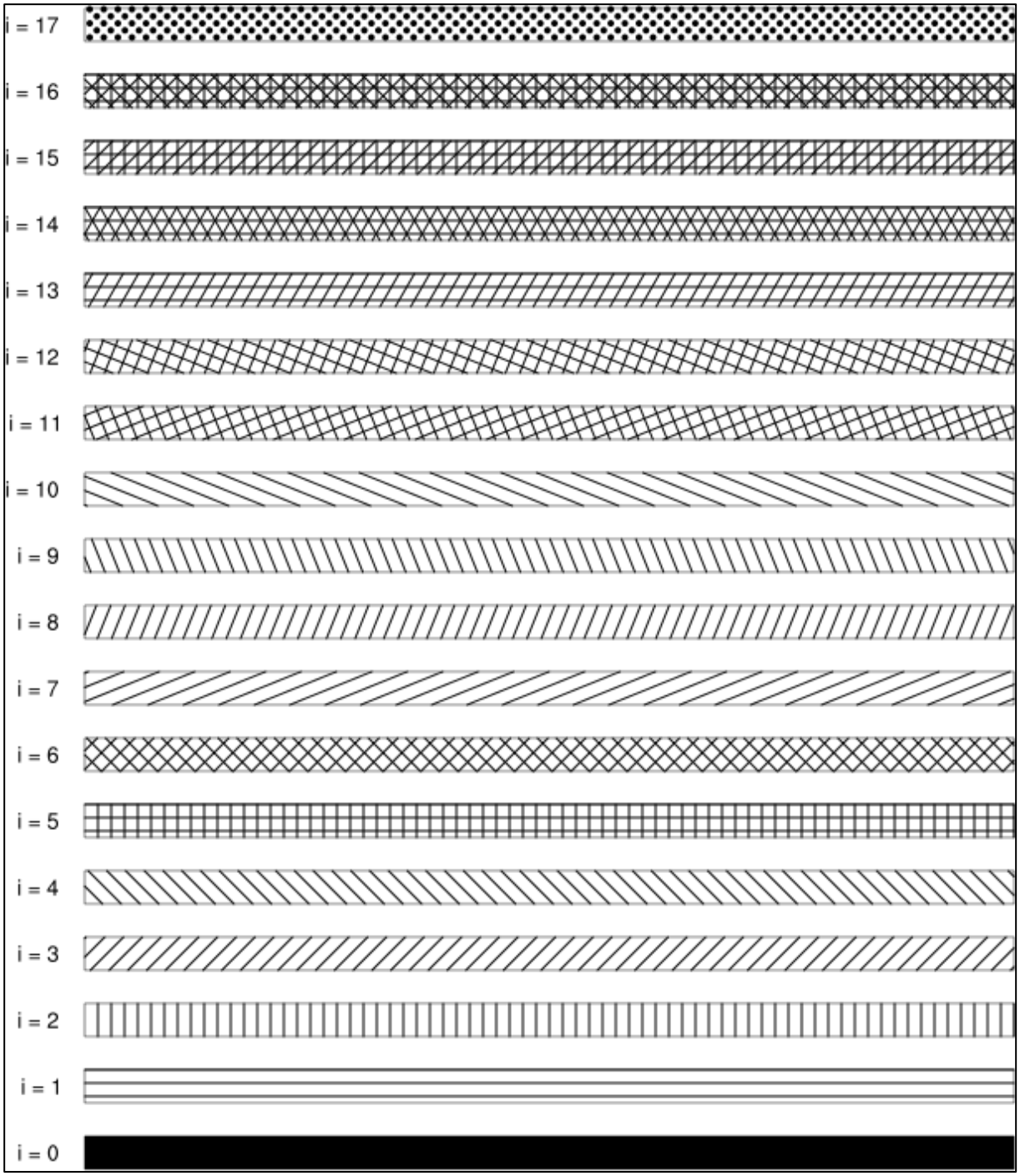
Dash pattern:



19 Appendix D - Fill Pattern

Eighteen fill patterns are available to use with the contour and polygon functions and their resources: *gsFillIndex*, *cnFillPattern* and *cnFillPatterns*.

Fill pattern:



20 Index

E

F

G

I

L

M

N

P

Q

R

S

V

W