Transition Guide NCL → PyNGL

Version 1.1
February 2019

Karin Meier-Fleischer, DKRZ

Table of Contents

1 Introduction

For most NCL users, the pivot from NCL to Python will likely be a big step which could take a significant amount of time. This guide was written to help users with the transition and hopefully ease some of the anxiety.

The first section provides a comparison table of NCL and Python language features, while the second section contains one-to-one mappings of NCL and Python arithmetic functions.

The next two sections provide NCL and Python examples of reading and writing ASCII and NetCDF files.

The rest of the guide contains a suite of graphical examples written in both NCL and Python, with the Python scripts using PyNGL for the graphics.

For the sections that contain NCL and Python scripts, you will find the NCL part on the left side and the equivalent Python part on the right side. You can directly compare the scripts line by line and you will see that there are not too many big differences as you might expect.

Many of the examples in this document can be found on the NCL website at:

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

Some of these examples use xarray instead of PyNIO, but the PyNIO code was commented out so you can use this if desired.

To run the example scripts, the easiest thing to do is use conda. You first need to install Miniconda via:

https://conda.io/en/latest/miniconda.html

You can then use conda to install all of the required packages. It is recommended that you install these packages to a separate environment:

```
conda create -n ncl_to_python -c conda-forge xarray netcdf4 scipy pyngl pynio ncl
source activate ncl_to_python
```

You can now download the NCL or Python scripts from the above website and any required data file(s) from:

http://www.ncl.ucar.edu/Document/Manuals/NCL_User_Guide/Data/

Finally, run the scripts with:

NCL: ncl *script_name.ncl*

PyNGL: python *script_name.py*

2 Basics of the Languages NCL and Python

Variable assignment, array indexing, loops, and conditional statements differ in NCL and Python but there are many similarities. To give a complete comparison would go beyond the scope of this document, therefore only the important things are mentioned.

To learn more about NCL, Python, Xarray, NumPy, etc. see

NCL documentation	http://www.ncl.ucar.edu/Document/
Python 2.x documentation	https://docs.python.org/2/index.html
Python 3.x documentation	https://docs.python.org/3/
Numpy and Scipy	https://docs.scipy.org/doc/
ESMPy	https://www.earthsystemcog.org/projects/esmpy/
xarray	http://xarray.pydata.org/en/stable/
xesmf	https://xesmf.readthedocs.io/en/latest/

NCL		PyNGL/PyNIO
`;-- this is a comment`		`#-- this is a comment`
`/;` `This is a block comment` `which can have multiple lines.` `;/`		`"""` `This is a block comment` `which can have multiple lines.` `"""`
`;-- load library file` `load "my_lib.ncl"`		`#-- import a module` `import my_lib`
`;-- define variable var` `var = 12` `print(typeof(var))`		`#-- define variable` `var = 12` `print(type(var))`
`;-- define variable vf of type float` `vf = 2.0` `print(typeof(vf))`		`#-- define variable vf of type float` `vf = 2.0` `print(type(vf))`
`;-- convert var to float` `var := tofloat(var)` `print(typeof(var))`		`#-- convert var to float` `var = float(var)` `print(type(var))`
`;-- convert var to string` `var := tostring(var)` `print(typeof(var))`		`#-- convert var to string` `var = str(var)` `print(type(var))`

;-- define an array array = (/0,3,6,9/) print(array)		#-- define an array array = [0,3,6,9] print(array) #-- its better to use Numpy for arrays import numpy as np array = np.array(array) print(array)
;-- overwrite array with zeros array = 0 print(array)		#-- overwrite array with zeros array[:] = 0 print(array)
;-- overwrite array with ones array = 1 print(array)		#-- overwrite array with ones array[:] = 1 print(array)
;-- addition of arrays a = (/1,2,3,4/) b = (/0,1,1,0/) c = a + b print(c)		#-- addition of arrays a = np.array([1,2,3,4]) b = np.array([0,1,1,0]) c = np.array(a + b) print(c)
;-- create some new arrays n = new(4,integer) n = 0 q = new((/2,3,5/),float) q = 0. l = new(100,float,1e20) print(n) print(q) print(l)		#-- create some new arrays n = np.zeros(4,np.int) q = np.zeros(shape=(2,3,5),dtype=float) l = np.full(100,1.0e20) print(n) print(q) print(l)
;-- subscripting a_sub = a(1:3) print(a_sub)		#-- array indexing (doesn't include the last index) a_sub = a[1:3] print(a_sub) #-- now it does a_sub = a[1:4] print(a_sub)
;-- reverse array a_rev = a(::-1)		#-- reverse array a_rev = a[::-1]

print(a_rev)		print(a_rev)
;-- select every second element a_sec = a(::2) print(a_sec)		#-- select every second element a_sec = a[::2] print(a_sec)
;-- find values in array ind_0 = ind(b .eq. 0) print(b(ind_0)		#-- find values in array ind_not0 = np.nonzero(b != 0) print(b[ind_not0])
;-- generate equaly spaced arrays i = ispan(0,10,1) print(i)		#-- generate equaly spaced arrays i = np.arange(0,10,1) print(i)
lat = fspan(-180.0,180.0,13) print(lat)		lat = np.linspace(-180.0, 180.0,13) print(lat) #-- or lat = np.arange(-180.0,**210.0**,**30.0**)
;-- dimension reshaping ra = (/(/(/1,2,3,4/),(/5,6,7,8/),(/5,4,2,0/)/),\ (/(/1,2,3,4/),(/5,6,7,8/),(/5,4,2,0/)/) /) print(**dimsizes(ra)**) ra1d = ndtooned(ra) print(""+ra1d) print(dimsizes(ra1d)) ra3d = onedtond(ra,dimsizes(ra)) print(dimsizes(ra3d))		#-- dimension reshaping ra = np.array([[[1,2,3,4],[5,6,7,8],[5,4,2,0]],\ [[1,2,3,4],[5,6,7,8],[5,4,2,0]]]) print(**ra.shape**) ra1d = np.ravel(ra) print(ra1d) print(ra.shape) ra3d = np.reshape(ra,ra.shape) print(ra3d.shape)
;-- if statements (NCL version >= 6.5.0) t = 99 if(t .eq. 0) then print("t = 0") elseif (t .eq. 1) then print("t = 1") else print("t = "+t) end if		#-- if statements t = 99 if t == 0: print("t = 0") elif t == 1: print("t = 1") else: print("t = {}".format(t))
;-- do loops do j=0,5 print("j = "+j) end do		#-- for loops for j in range(0,5,1): print("j = {}".format(j))

str_array = (/"Hamburg","Munich","Berlin"/) do k = 0,dimsizes(str_array)-1 print(str_array(k)+"") end do		str_array = ["Hamburg","Munich","Berlin"] for city in str_array: print(city)
;-- while loops j = 0 do while(j .lt. 5) print("j = "+j) if(j .eq. 3) then print("--> j = "+j) end if j = j + 1 end do		#-- while loops j = 0 while(j <= 5): print("j = ",j) if j == 3: print("--> j = {}".format(j)) j = j + 1

3 Arithmetic Functions

The module numpy provides numerous arithmetic functions to calculate e.g. averages, means, minimum, maximum, and statistics. The list below doesn't contain all functions but the important ones.

See also NCL http://ncl.ucar.edu/Document/Functions/math.shtml
Numpy https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html

Module load `import numpy as np`

Description	NCL	Numpy
Minimum	min, dim_min_n	np.min
Maximum	max, dim_max_n	np.max
Sum	dim_sum_n	np.sum
Product	dim_product_n	np.prod
Square-root	sqrt	np.sqrt
Absolut value	abs	np.abs
Sinus	sin	np.sin
Cosine	cos	np.cos
Tangent	tan	np.tan
Inverse sinus	asin	np.arcsin
Inverse cosine	acos	np.arccos
Inverse tangent	atan	np.arctan
Convert radians in degrees	get_r2d	np.degrees, np.rad2deg
Convert degrees in radians	get_d2r	np.radians, np.deg2rad
Average	dim_avg_n	np.average
Exponent	exp	np.exp
Logarithm	log	np.log
Standard deviation	stddev, dim_stdev_n	np.std
Variance	dim_variance_n	np.var
Return the ceiling	ceil	np.ceil
Return the floor	floor	np.floor
Remainder of division	mod	np.mod

4 Read a File

The best way to compare NCL and PyNGL is to oppose an NCL script to a PyNGL script. In this chapter we will give an overview of the differences and how similar the scripts are. Using PyNIO to open and read a file is straight forward to the addfiles handling, you don't have to take care if you have a GRIB, HDF or netCDF file. Reading an ASCII file is as with NCL a little bit different and will be explained afterwards.

4.1 Read GRIB, HDF or netCDF Data Sets

The start makes a script which opens a file, print the variables it contains, reads a variable and print some information about some variables.

NCL		PyNGL/PyNIO
;-- data file name fname = "rectilinear_grid_3D.nc" ;-- open file f = addfile(fname,"r") ;-- retrieve the variables stored in file print(getfilevarnames(f)) ;-- read variable, first time step temp = f->t(0,:,:) ;-- print variable summary printVarSummary(temp) ;-- print variable lat content print(""+f->lat) ;-- print variable lon content print(""+f->lon)		import Ngl,Nio #-- import PyNGL and PyNIO modules #-- data file name fname = "rectilinear_grid_3D.nc" #-- open file f = Nio.open_file(fname) #-- retrieve the variables stored in file print(f.variables) #-- read variable, first time step temp = f.variables["t"][0] #temp = f.variables["t"][0,:,:] # Also valid #-- print variable summary print(f.variables["t"]) #-- print variable lat content print(f.variables["lat"]) #-- print variable lon content print(f.variables["lon"])

4.2 Read an ASCII File

Usually ASCII files are used to store data in a readable format e.g. for observed data (station data). The ASCII data files can contain data written column-wise with delimiter separating the columns (CSV files), block-wise or any other combination. Therefor it is very important to have a description what is in the file.

Assume we want to read some data, store it in an array, print the array values and some more information like number of columns and lines, rank and shape. The input ASCII file is Test_6h.csv

```
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		PyNGL/PyNIO
;-- data file name fili = "Test_6h.csv" ;-- number of lines and columns in input file nrows = 5 ncols = 4 ;-- read all data vals = asciiread(fili,(/nrows,ncols/),"float") ;-- print information print("vals: "+vals) print("rank of vals: "+dimsizes(dimsizes(vals))) print("shape vals: "+dimsizes(vals))		import numpy as np import Ngl #-- data file name fili = "Test_6h.csv" #-- number of lines and columns in input file nrows = 5 ncols = 4 #-- read all data vals = Ngl.asciiread(fili,(nrows,ncols),"float",**sep=';'**) #-- print information print("vals: {}".format(vals)) print("rank of vals: {}".format(len(vals.shape))) print("shape vals: {}".format(vals.shape)) exit()

The next example will show you how to read an ASCII file which contains 21361 lines. One header line and 21360 lines with latitudes, longitudes, and temperature values. The file name is ***asc6.txt*** which can be downloaded from http://ncl.ucar.edu/Applications/Data/asc/asc6.txt.

```
Lat    Lon    Temp (C)
33.3   76.5   20.3
33.3   76.6   20.3
33.3   76.7   21.5
33.3   76.8   20.0
...
```

NCL		PyNGL/PyNIO
		import numpy as np import Ngl

```
;-- file has 21361 lines but 1 header line
;-- 3 columns
nrows   = 21360
ncols   =     3
num_lon =   240         ;-- number of longitudes
num_lat =    89         ;-- number of latitudes

;-- read all data from file
data = asciiread("asc6.txt",(/nrows,ncols/),"float")

;-- select lat, lon and temp data
lat    = data(::num_lon,0)
lon    = data(:num_lon-1,1)
temp1D = data(:,2)

;-- size of lat, lon and temp1D
nlats  = dimsizes(lat)
nlons  = dimsizes(lon)
ntemp  = dimsizes(temp1D)

;-- reshape temp1d to 2D-array temp2D array (89,240)
temp2D = onedtond(temp1D,(/nlats,nlons/))

;-- print information
print("rank  temp1D: " + dimsizes(dimsizes(temp1D)))
print("shape temp1D: " + ntemp)
print("rank  temp2D: " + dimsizes(dimsizes(temp2D)))
print("shape temp2D: " + dimsizes(temp2D))
```

```
#-- file has 21361 lines but 1 header line
#-- 3 columns
nrows   = 21360
ncols   =     3
num_lon =   240         #-- number of longitudes
num_lat =    89         #-- number of latitudes

#-- read all data from file
data = Ngl.asciiread("asc6.txt",(nrows,ncols),"float")

#-- select lat, lon and temp data
lat    = data[::num_lon,0]
lon    = data[:num_lon,1]
temp1D = data[:,2]

#-- size of lat, lon and temp1D
nlats  = len(lat)
nlons  = len(lon)
ntemp  = len(temp1D)

#-- reshape temp1d to 2D-array temp2d with size (89,240)
temp2D = np.reshape(temp1D,(nlats,nlons))

#-- print information
print("rank  temp1D: {}".format (len(temp1D.shape)))
print("shape temp1D: {}".format (ntemp))
print("rank  temp2D: {}".format(len(temp2D.shape)))
print("shape temp2D: {}".format (temp2D.shape))

exit()
```

4.3 Read Multiple Files

Sometimes a variable is stored in more than one file for different time steps. In this case NCL and Python's netCDF4 package are able to read the multiple files as one.

NCL

```
;-- list of files
file_list  = systemfunc("ls file_*.nc")

;-- open file
```

PyNGL/PyNIO

```
import netCDF4 as nc

#-- list of files
file_list  = "file_*.nc"
```

```
f = addfiles(file_list,"r")

;-- read variables
var = f[:]->tsurf
print(getvardimnames(var))

;-- read dimension time variable
time = f[:]->time
print(time)

#-- read dimensions lat and lon variables
lat = f[0]->lat
print(lat)

lon = f[0]->lon
print(lon)
```

```
#-- open file
f = nc.MFDataset(file_list)

#-- read variable
var = f.variables['tsurf']
print(var.dimensions)

#-- read dimension time variable
time = f.variables['time']
print(time[:])

#-- read dimensions lat and lon variables
lat = f.variables['lat']
print(lat[:])

lon = f.variables['lon']
print(lon[:])

exit()
```

5 Write a File

In many cases it is important to be able to save the computed data to a new file. The next two script comparisons show one case for a netCDF file and one case for an ASCII file.

5.1 Write a netCDF File

Let us assume that we have a data set ***rectilinear_grid_3D.nc*** and we want to read the variable t at level=0 and convert it from Kelvin to degrees Celsius. Afterwards, we want to write the converted data to a new netCDF file. There are two possible ways to write the data to a netCDF file, one short way and one more detailed way, the difference is the metadata of the netCDF files.

Note, that it is much more efficient if all dimensions, variables, and attributes are defined/created before any actual data is written to the output file.

The short way:

NCL

```
;-- data file name
fname = "rectilinear_grid_3D.nc"

;-- open file
f = addfile(fname,"r")

;-- read temperature, time, latitude and longitude
var  =  f->t
time =  f->time
lat  =  f->lat
lon  =  f->lon

;-- convert data from units Kelvin to degC
varC = var(:,0,:,:)    ;-- copy variable at level=0
                       ;   retain metadata
varC =  varC-273.15    ;-- convert to degC
varC@units = "degC"    ;-- change units attribute

;-- open new netCDF file
system("rm -rf t_degC_ncl_short.nc");-- delete file
outf = addfile("t_degC_ncl_short.nc","c")
```

PyNGL/PyNIO

```
import os
import numpy as np
import Ngl,Nio

#-- data file name
fname = "rectilinear_grid_3D.nc"

#-- open file
f = Nio.open_file(fname, "r")

#-- read temperature, time, latitude and longitude
var  = f.variables["t"]
time = f.variables["time"]
lat  = f.variables["lat"]
lon  = f.variables["lon"]

#-- convert data from units Kelvin to degC
varC = var[:,0,:,:]       #-- copy variable at level=0
                          #   retain metadata
varC = varC-273.15        #-- convert to degC


#-- open new netCDF file
os.system("rm -rf t_degC_py_short.nc")  #-- delete file
outf = Nio.open_file("t_degC_py_short.nc","c")
```

```
;-- write data to new file
outf->time = time
outf->lat  = lat
outf->lon  = lon
outf->t    = varC

;-- close output stream (not necessary)
delete(outf)
```

```
#-- create dimensions time, lat and lon
outf.create_dimension('time',None)
outf.create_dimension('lat',f.dimensions['lat'])
outf.create_dimension('lon',f.dimensions['lon'])

#-- create dimension variables
outf.create_variable('time',time.typecode(),time.dimensions)
outf.create_variable('lat',lat.typecode(),lat.dimensions)
outf.create_variable('lon',lon.typecode(),lon.dimensions)

#-- create variable t
outf.create_variable('t','f',('time','lat','lon'))

#-- write data to new file (assign values)
outf.variables['time'].assign_value(time)
outf.variables['lat'].assign_value(lat)
outf.variables['lon'].assign_value(lon)
outf.variables['t'].assign_value(varC)

#-- close output stream
outf.close()
```

The created netCDF files don't differ comparing the data (*cdo diff t_degC_ncl_short.nc t_degC_py_short.nc*) but the metadata of both files do.

ncdump -h t_degC_ncl_short.nc

```
netcdf t_degC_ncl_short {
dimensions:
     time = 1 ;
     lat = 96 ;
     lon = 192 ;
variables:
     double time(time) ;
          time:units = "hours since 2001-01-01 00:00:00" ;
          time:calendar = "standard" ;
     double lat(lat) ;
          lat:long_name = "latitude" ;
          lat:units = "degrees_north" ;
          lat:standard_name = "latitude" ;
          lat:axis = "Y" ;
     double lon(lon) ;
          lon:long_name = "longitude" ;
          lon:units = "degrees_east" ;
          lon:standard_name = "longitude" ;
```

ncdump -h t_degC_py_short.nc

```
netcdf t_degC_py_short {
dimensions:
     time = UNLIMITED ; // (1 currently)
     lat = 96 ;
     lon = 192 ;
variables:
     double time(time) ;
     double lat(lat) ;
     double lon(lon) ;
     float t(time, lat, lon) ;
}
```

```
          lon:axis = "X" ;
     float t(time, lat, lon) ;
          t:lev = 100000. ;
          t:grid_type = "gaussian" ;
          t:table = 128 ;
          t:code = 130 ;
          t:units = "degC" ;
          t:long_name = "temperature" ;
}
```

The more detailed way to keep or set the metadata:

NCL

```
;-- data file name
fname = "rectilinear_grid_3D.nc"

;-- open file
f = addfile(fname,"r")

;-- read temperature, time, latitude and longitude
var  =  f->t
time =  f->time
lat  =  f->lat
lon  =  f->lon

;-- convert data from units Kelvin to degC
varC = var(:,0,:,:)    ;-- copy variable at level=0
                       ;   retain metadata
varC =  varC-273.15      ;-- convert to degC
varC@units = "degC"      ;-- change units attribute

;-- open new netCDF file
system("rm -rf t_degC.nc")  ;-- delete file
outf = addfile("t_degC.nc","c")
```

PyNGL/PyNIO

```
import os
import numpy as np
import Ngl,Nio

#-- data file name
fname = "rectilinear_grid_3D.nc"

#-- open file
f = Nio.open_file(fname, "r")

#-- read temperature, time, latitude and longitude
var  = f.variables["t"]
time = f.variables["time"]
lat  = f.variables["lat"]
lon  = f.variables["lon"]

#-- convert data from units Kelvin to degC
varC = var[:,0,:,:]       #-- copy variable at level=0
                          #   retain metadata
varC = varC-273.15             #-- convert to degC


#-- open new netCDF file
os.system("rm -rf t_degC.nc")  #-- delete file
outf = Nio.open_file("t_degC.nc","c")

#-- create dimensions time, lat and lon
```

```
;-- write data to new file
outf->time = time
outf->lat  = lat
outf->lon  = lon
outf->t    = varC

;-- close output stream (not necessary)
delete(outf)
```

```
outf.create_dimension('time',None)
outf.create_dimension('lat',f.dimensions['lat'])
outf.create_dimension('lon',f.dimensions['lon'])

#-- create dimension variables
outf.create_variable('time',time.typecode(),time.dimensions)
varAtts = list(time.__dict__.keys())
varAtts.sort()
for att in varAtts:
      value = getattr(time,att)
      setattr(outf.variables['time'],att,value)

outf.create_variable('lat',lat.typecode(),lat.dimensions)
varAtts = list(lat.__dict__.keys())
varAtts.sort()
for att in varAtts:
      value = getattr(lat,att)
      setattr(outf.variables['lat'],att,value)

outf.create_variable('lon',lon.typecode(),lon.dimensions)
varAtts = list(lon.__dict__.keys())
varAtts.sort()
for att in varAtts:
      value = getattr(lon,att)
      setattr(outf.variables['lon'],att,value)

#-- create variable t
outf.create_variable('t','f',('time','lat','lon'))
setattr(outf.variables['t'], 'standard_name', 'temperature')
setattr(outf.variables['t'], 'units', 'degC')

#-- write data to new file (assign values)
outf.variables['time'].assign_value(time)
outf.variables['lat'].assign_value(lat)
outf.variables['lon'].assign_value(lon)
outf.variables['t'].assign_value(varC)

#-- close output stream
outf.close()
```

The metadata looks now like:

ncdump -h t_degC_ncl_short.nc		ncdump -h t_degC_py.nc

```
netcdf t_degC_ncl {
dimensions:
     time = UNLIMITED ; // (1 currently)
     lat = 96 ;
     lon = 192 ;
variables:
  double time(time) ;
     time:units = "hours since 2001-01-01 00:00:00" ;
     time:calendar = "standard" ;
  double lat(lat) ;
     lat:long_name = "latitude" ;
     lat:units = "degrees_north" ;
     lat:standard_name = "latitude" ;
     lat:axis = "Y" ;
  double lon(lon) ;
     lon:long_name = "longitude" ;
     lon:units = "degrees_east" ;
     lon:standard_name = "longitude" ;
     lon:axis = "X" ;
  float t(time, lat, lon) ;
     t:lev = 100000. ;
     t:grid_type = "gaussian" ;
     t:table = 128 ;
     t:code = 130 ;
     t:units = "degC" ;
     t:long_name = "temperature" ;
}
```

```
netcdf t_degC_py {
dimensions:
     time = UNLIMITED ; // (1 currently)
     lat = 96 ;
     lon = 192 ;
variables:
double time(time) ;
     time:calendar = "standard" ;
     time:units = "hours since 2001-01-01 00:00:00" ;
double lat(lat) ;
     lat:axis = "Y" ;
     lat:long_name = "latitude" ;
     lat:standard_name = "latitude" ;
     lat:units = "degrees_north" ;
double lon(lon) ;
     lon:axis = "X" ;
     lon:long_name = "longitude" ;
     lon:standard_name = "longitude" ;
     lon:units = "degrees_east" ;
float t(time, lat, lon) ;
     t:standard_name = "temperature" ;
     t:units = "degC" ;
}
```

5.2 Write an ASCII File

Now, we want to describe how to write the content of a variable to an ASCII file. Both, NCL and PyNGL work in a very similar way.

NCL

```
;-- data file name
fname = "rectilinear_grid_3D.nc"

;-- open file
f = addfile(fname, "r")
```

PyNGL/PyNIO

```
import os, sys
import numpy as np
import Ngl,Nio

#-- data file name
fname = "rectilinear_grid_3D.nc"

#-- open file
f = Nio.open_file(fname, "r")
```

```
;-- read variable, first time step, first level
var = f->t(0,0,:,:)

;-- convert var from K to degC
var = var - 273.15

print(var)

;-- write var to an ASCII file
asciiwrite("data_ncl.asc",sprintf("%10.6f",var(0:9,0:9)))
```

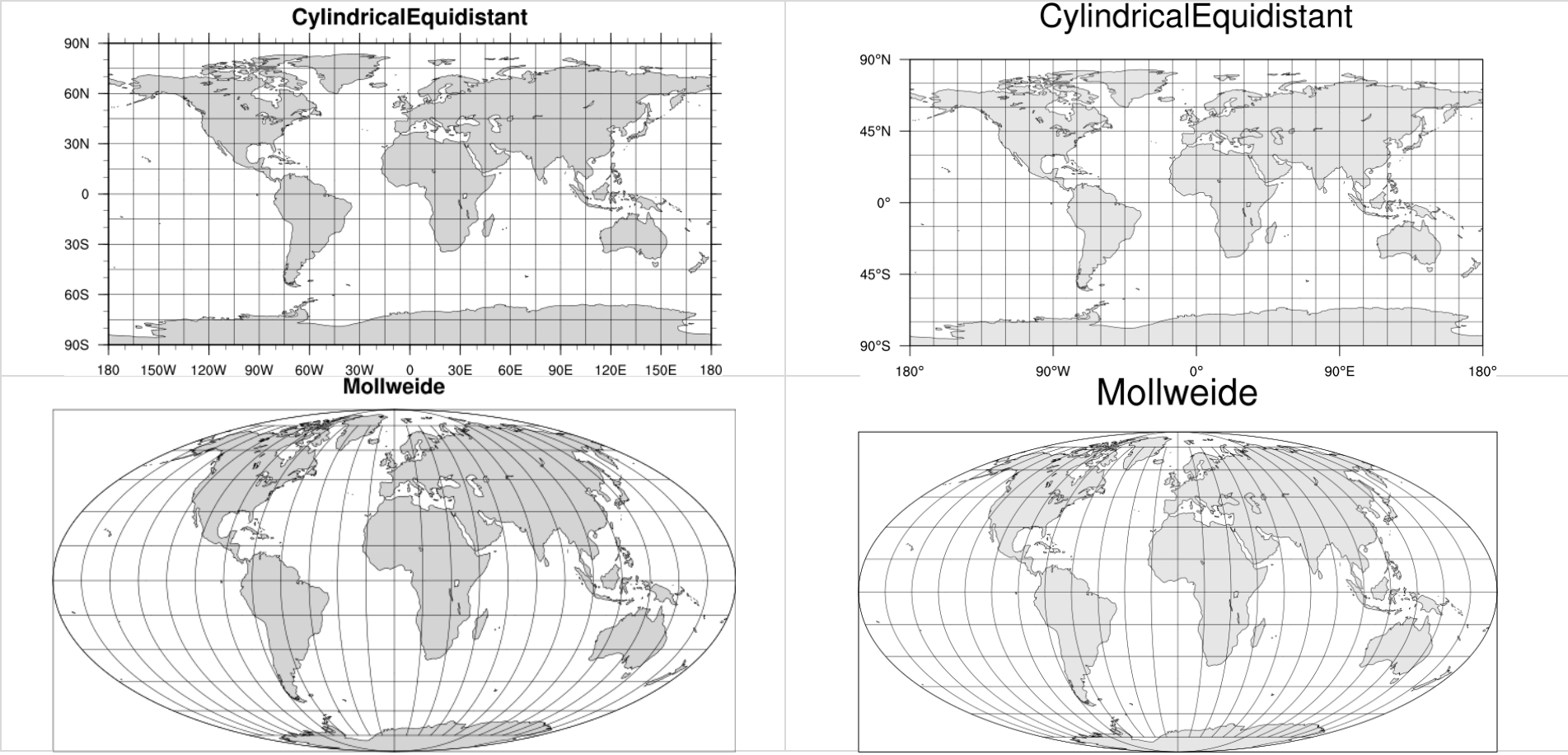
```
#-- read variable, first time step, first level
var = f.variables["t"][0,0,:,:]

#-- convert var from K to degC
var = var - 273.15

print(var)

#-- write var to an ASCII file
os.system("/bin/rm -f data_py.asc")    #-- delete file


#-- redirect stdout to file
sys.stdout = open("data_py.asc","w")
for i in range(0,10):
      for j in range(0,10):
            print("{:10.6f}".format(var[i,j])

exit()
```

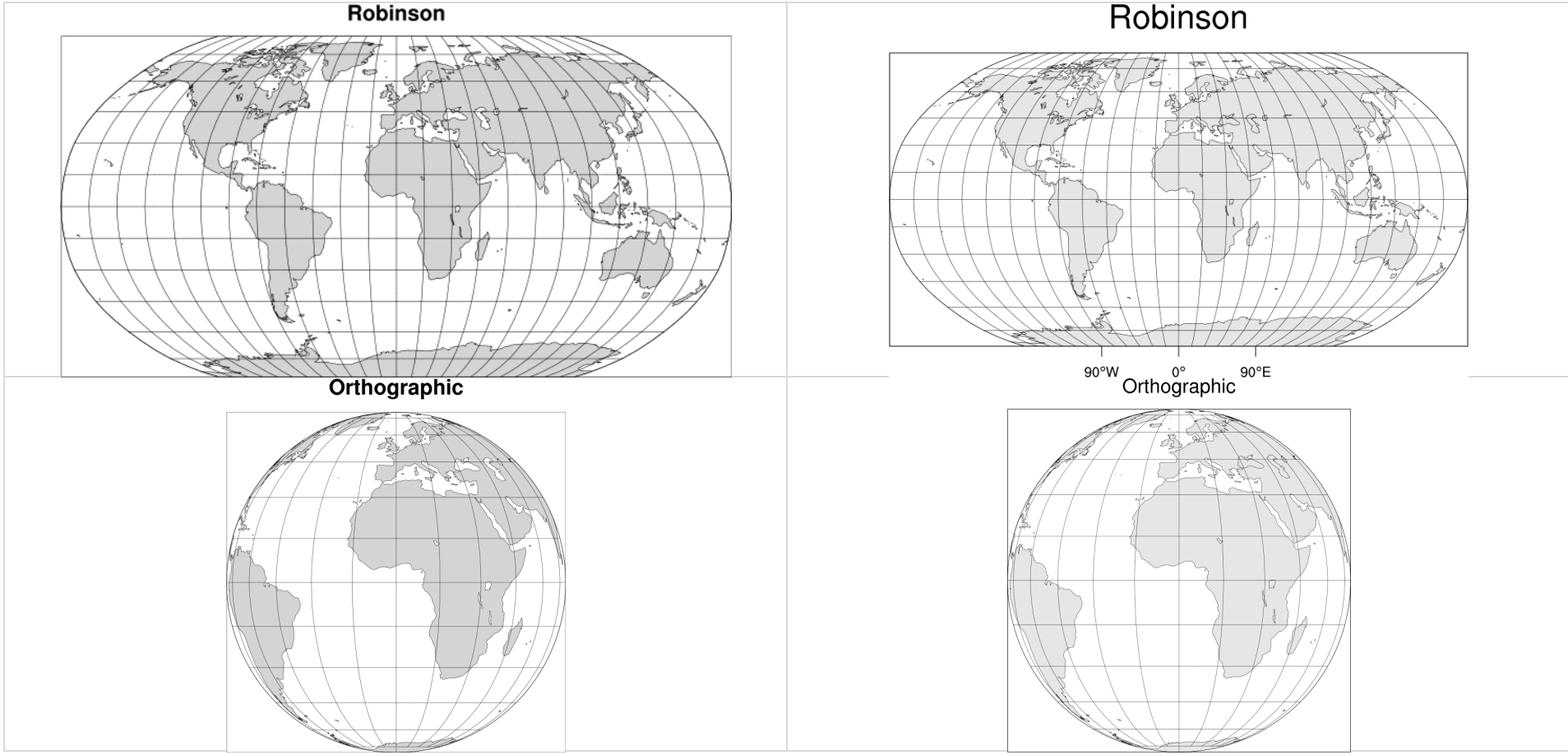
6 Plotting

Now, you should be ready to create some plots. As well as seen before there are many similarities between NCL and PyNGL graphic settings. You shouldn't have many problems to make this transition, too.

6.1 Maps

A good start for plotting is to draw simple maps with filled land areas. This example shows how to loop through a given array of different projections and create a plot for each.





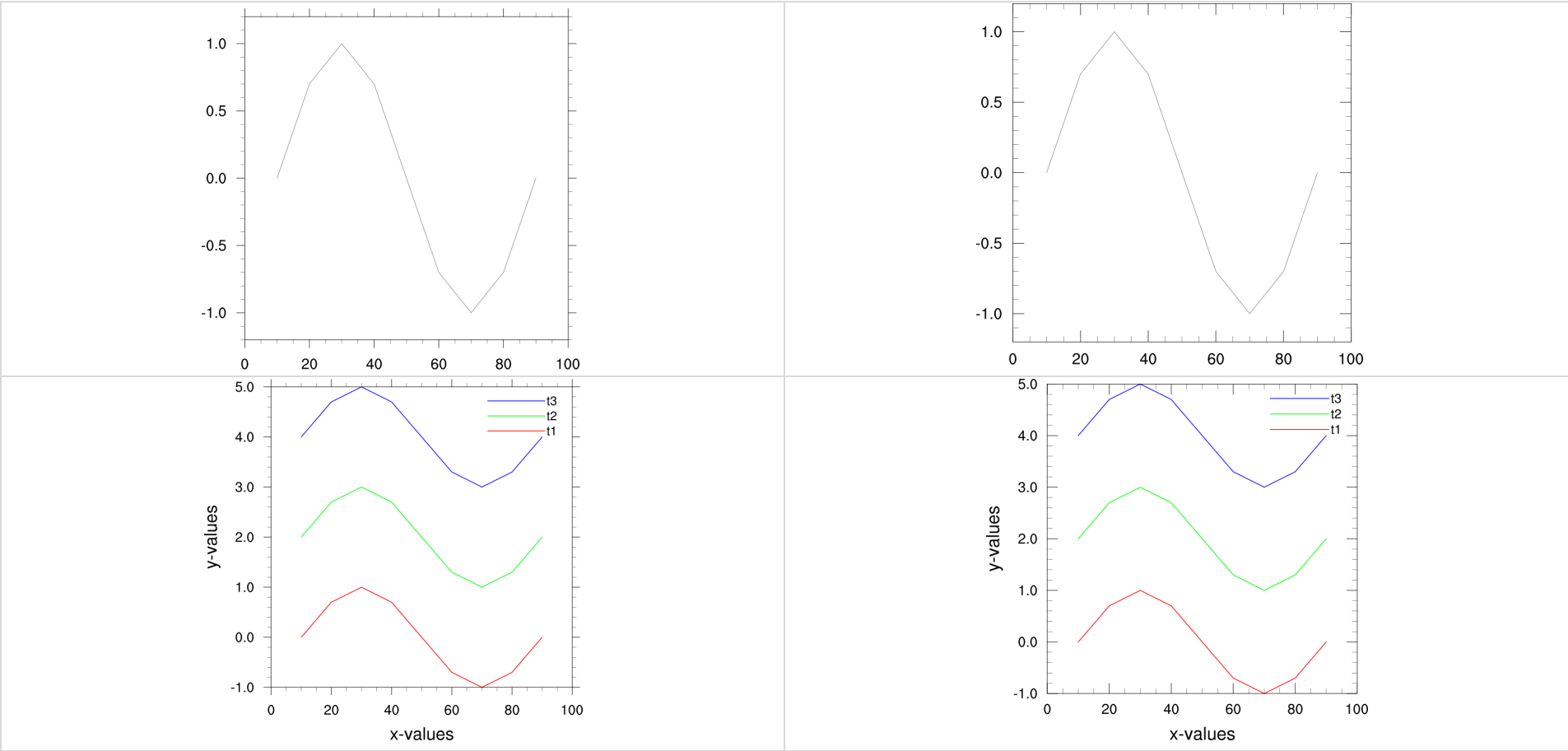
NCL		PyNGL/PyNIO
;-- open workstation wks_type = "png" wks = gsn_open_wks(wks_type,"plot_TRANS_maps_ncl") ;-- which projection do we want to plot projections = (/"CylindricalEquidistant","Mollweide",\ "Robinson","Orthographic"/) ;-- resource settings		import Ngl #-- open workstation wks_type = "png" wks = Ngl.open_wks(wks_type,"plot_TRANS_maps_py") #-- which projection do we want to plot projections = ["CylindricalEquidistant","Mollweide",\ "Robinson","Orthographic"] #-- resource settings

```
mpres              =  True          ;-- resource object

mpres@vpWidthF     =  0.8           ;-- viewport width
mpres@vpHeightF    =  0.8           ;-- viewport height

mpres@mpGridAndLimbOn =  True
mpres@mpPerimOn    =  True
mpres@mpOutlineOn  =  True


do i=0,dimsizes(projections)-1
     mpres@mpProjection = projections(i)
     mpres@tiMainString = projections(i)
     map = gsn_csm_map(wks,mpres)
end do
```

```
mpres              =  Ngl.Resources()   #-- resource object

mpres.vpWidthF     =  0.8               #-- viewport width
mpres.vpHeightF    =  0.8               #-- viewport height

mpres.mpFillOn     =  True
mpres.mpOceanFillColor = "Transparent"
mpres.mpLandFillColor  = "Gray90"
mpres.mpInlandWaterFillColor = "Gray90"

for proj in projections:
     mpres.mpProjection = proj
     mpres.tiMainString = proj
     map = Ngl.map(wks,mpres)

Ngl.end()
```

6.2 XY-Plot

An XY-plot shows the relation of two given arrays in a coordinate system. The first example shows how to create a simple XY-plot with 1D data arrays x and y.



NCL		PyNGL/PyNIO
;-- define x and y variables x = (/10.,20.,30.,40.0,50.,60.,70.,80.,90./) y = (/0.,0.7,1.,0.7,0.,-0.7,-1.,-0.7,0./)		import numpy as np import Ngl #-- define x and y variables x = [10., 20., 30., 40., 50., 60., 70., 80., 90.] y = np.array([0.,0.7,1.,0.7,0.,-0.7,-1.,-0.7,0.],np.float32)

```
;-- open a workstation
wks = gsn_open_wks("png","plot_TRANS_xy_0_ncl)



;-- set resources
res                =  True
res@gsnMaximize    =  True       ;-- max. plot size

;-- draw the plot
plot = gsn_csm_xy(wks,x,y,res)
```

```
#-- open a workstation
wkres            =  Ngl.Resources()
wks_type         = "png"
wks =  Ngl.open_wks(wks_type,"plot_TRANS_xy_0_py",wkres)

#-- set resources
res              =  Ngl.Resources()


#-- draw the plot
plot = Ngl.xy(wks,x,y,res)

#-- done
Ngl.end()
```

Assume, we have 3 data arrays which we want to plot in the same plot frame with different colors and a legend. The data values can be stored in one array y because they correspond to the same x-values.

NCL

```
;--  define x and y variables
x = (/10.,20.,30.,40.0,50.,60.,70.,80.,90./)
y = (/(/0.,0.7,1.,0.7,0.,-0.7,-1.,-0.7,0./),  \
      (/2.,2.7,3.,2.7,2.,1.3,1.,1.3,2./),  \
      (/4.,4.7,5.,4.7,4.,3.3,3.,3.3,4./)/)

;-- open a workstation
wks = gsn_open_wks("png","plot_TRANS_xy_1_ncl)



;-- set resources
res                    =  True
res@gsnMaximize        =  True      ;-- max. plot size
res@tiXAxisString      = "x-values" ;-- x-axis title
res@tiYAxisString      = "y-values" ;-- y-axis title

;-- xy-plot resources
res@xyLineColors       = (/"red","green","blue"/)
res@xyDashPattern      =  0         ;-- solid lines
res@xyLineThicknessF   =  3.0       ;-- line thickness
res@xyExplicitLegendLabels = (/"t1","t2","t3"/)

;-- legend resources
```

PyNGL/PyNIO

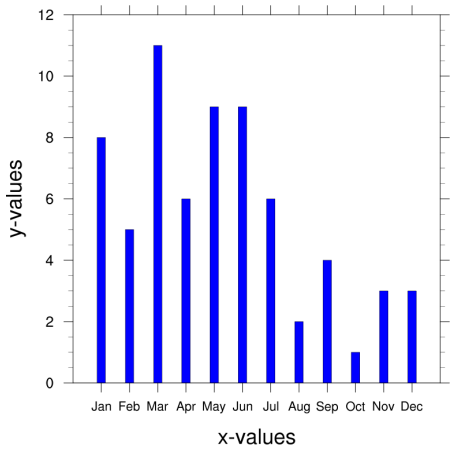
```
import numpy as np
import Ngl

#--  define x and y variables
x = [10., 20., 30., 40., 50., 60., 70., 80., 90.]
y = np.array([[0.,0.7,1.,0.7,0.,-0.7,-1.,-0.7,0.],  \
              [2.,2.7,3.,2.7,2.,1.3,1.,1.3,2.],  \
              [4.,4.7,5.,4.7,4.,3.3,3.,3.3,4.]],np.float32)

#-- open a workstation
wkres            =  Ngl.Resources()
wks_type         = "png"
wks =  Ngl.open_wks(wks_type,"plot_TRANS_xy_1_py",wkres)

#-- set resources
res                    =  Ngl.Resources()

res.tiXAxisString      = "x-values"  #-- x-axis title
res.tiYAxisString      = "y-values"  #-- y-axis title

#-- xy-plot resources
res.xyLineColors       = ["red","green","blue"]
res.xyLineThicknessF   =  3.0        #-- line thickness
res.xyExplicitLegendLabels = ["t1","t2","t3"]


#-- legend resources
```

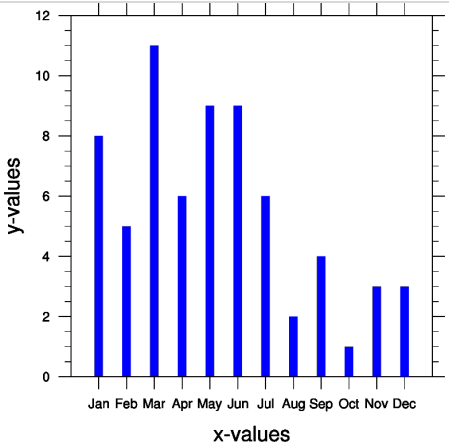
```
res@pmLegendDisplayMode = "Always";-- turn on drawing
res@pmLegendZone      =  0         ;-- topleft
res@pmLegendOrthogonalPosF =  0.32;-- move upwards
res@lgJustification   = "BottomRight";-- legend just.
res@pmLegendWidthF    =  0.2       ;-- change width
res@pmLegendHeightF   =  0.10      ;-- change height
res@pmLegendSide      = "Top"      ;-- Change location
res@lgPerimOn         =  False ;-- turn off perimeter

;-- draw the plot
plot = gsn_csm_xy(wks,x,y,res)
```

```
res.pmLegendDisplayMode = "Always" #-- turn on drawing
res.pmLegendZone      =  0          #-- topLeft
res.pmLegendOrthogonalPosF =  0.32 #-- move upwards
res.lgJustification   = "BottomRight" #-- legend just.
res.pmLegendWidthF    =  0.2        #-- change width
res.pmLegendHeightF   =  0.10       #-- change height
res.pmLegendSide      = "Top"       #-- change location
res.lgPerimOn         =  False      #-- turn off perimeter

#-- draw the plot
plot = Ngl.xy(wks,x,y,res)

#-- done
Ngl.end()
```

6.2.1 Bar Charts and Histograms

Bar charts are simply XY-plots that are drawn with rectangular bars proportional to the values they have. The example scripts use one 1D array x for the x-axis center position of the bars and another 1D array y for the heights of the bars. With the resources @gsnXYBarChart* the user can tell NCL how the rectangular bars should look like. Unfortunately, PyNGL doesn't provide these resources and we have to draw the bars using the function Ngl.polygon.





NCL		PyNGL/PyNIO
		import numpy as np import Ngl #-- function get_bar returns coordinates of a bar

```
;-- create random x- and y-values
x = fspan(1.0,12.0,12)
y = (/ 8, 5, 11, 6, 9, 9, 6, 2, 4, 1, 3, 3/)


;-- define color and x-axis labels
color  = "blue"
xlabels = (/"Jan","Feb","Mar","Apr","May","Jun",\
            "Jul","Aug","Sep","Oct","Nov","Dec"/)

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



;-- set resources
res                  =  True
res@gsnMaximize      =  True
res@gsnXYBarChart    =  True          ;-- bar chart style
res@gsnXYBarChartBarWidth =  0.3      ;-- width of bars
res@gsnXYBarChartColors = color       ;-- color



res@tiXAxisString   = "x-values"      ;-- x-axis title
res@tiYAxisString   = "y-values"      ;-- y-axis title

res@tmXBMode        = "Explicit"      ;-- explicit labels
res@tmXBValues      =  x              ;-- x-values
res@tmXBLabels      =  xlabels        ;-- x-axis labels
res@tmXBLabelFontHeightF =  0.015     ;-- x-axis font size

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         = 12.0            ;-- y-axis max value
```

```
def get_bar(x,y,dx,ymin,bar_width_perc=0.6):
     dxp  =  (dx * bar_width_perc)/2.
     xbar =  np.array([x-dxp, x+dxp, x+dxp, x-dxp, x-dxp])
     ybar =  np.array([ ymin, ymin,    y,    y, ymin])
     return xbar,ybar

#--------------
#     MAIN
#--------------
#-- create random x- and y-values
x  = np.arange(1,13,1)
y  = [8,5,11,6,9,9,6,2,4,1,3,3]
dx = min(x[1:-1]-x[0:-2])          #-- distance between x-values

#-- define color and x-axis labels
color   = 'blue'
xlabels = ["Jan","Feb","Mar","Apr","May","Jun", \
           "Jul","Aug","Sep","Oct","Nov","Dec"]

#-- open a workstation
wkres    =  Ngl.Resources()                 #-- resources object
wks_type = "png"                            #-- output type
wks =  Ngl.open_wks(wks_type,"plot_TRANS_bar_chart_py",wkres)

#-- set resources
res =  Ngl.Resources()                      #-- resources object
res.nglFrame      =  False                  #-- don't advance frame
res.nglPointTickmarksOutward = True         #-- tickmarks outward

#-- bar resources
barres             =  Ngl.Resources()       #-- resource object
barres.gsFillColor =  color                 #-- set bar color

res.tiXAxisString = "x-values"              #-- x-axis title
res.tiYAxisString = "y-values"              #-- y-axis title

res.tmXBMode      = "Explicit"              #-- tickmarks mode
res.tmXBValues    =  x                      #-- x-axis values
res.tmXBLabels    =  xlabels                #-- x-axis labels
res.tmXBLabelFontHeightF = 0.012            #-- x-axis font size

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       =  12.0                   #-- y-axis max value

#-- loop through each y point and create bars
for i in range(len(y)):
     xbar,ybar = get_bar(x[i], y[i], dx, res.trXMinF, 0.3)
```

```
;-- create the plot
plots = gsn_csm_xy(wks, x, y, res)
```

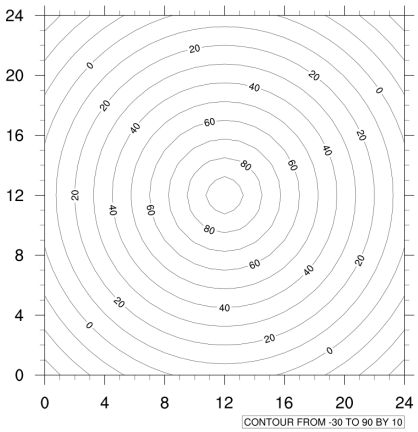
```
      plot = Ngl.xy(wks, xbar, ybar, res)
      Ngl.polygon(wks, plot,  xbar, ybar, barres) #-- filled bar

#-- create the plot
Ngl.frame(wks)                                    #-- advance frame

Ngl.end()
```

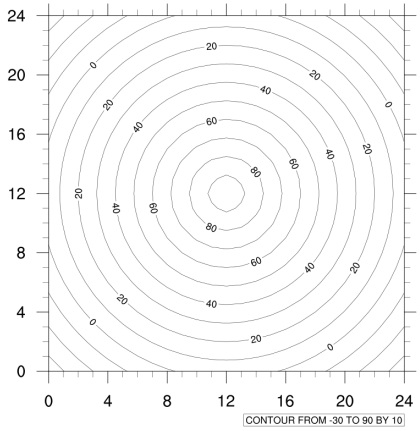
6.3 Contour Plots

This section shows how to draw contour lines and color filled areas between contour levels (= contour fill). The examples use random dummy data.

6.3.1 Contour Line Plot

The example shows how to create a simple contour line plot.





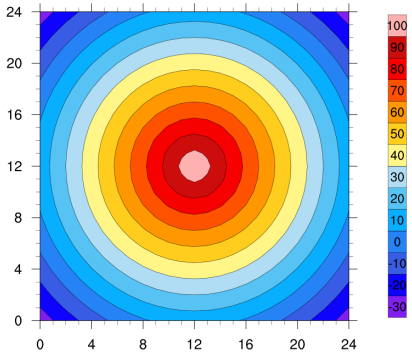
NCL		PyNGL/PyNIO
;-- create some dummy data to contour T = new((/25,25/),float) vals = ispan(-12,12,1) jspn = vals*vals ispn = vals*vals do i=0,dimsizes(jspn)-1 T(i,:) = tofloat(jspn + ispn(i)) end do T = 100.0 - sqrt(64 * T) ;-- start the graphics wks = gsn_open_wks("png","plot_TRANS_contour_line_ncl")		import numpy as np import Ngl #-- create some dummy data to contour T = np.zeros((25,25),np.float32) jspn = np.power(np.arange(-12,13),2) ispn = np.power(np.arange(-12,13),2) for i in range(0,len(ispn)): T[i,:] = (jspn + ispn[i]).astype(np.float32) T = 100.0 - np.sqrt(64 * T) #-- start the graphics wks = Ngl.open_wks("png","plot_TRANS_contour_line_py")

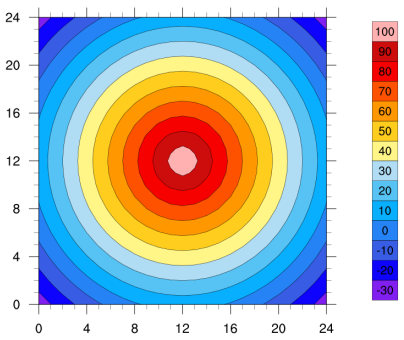
```
;-- resource settings
res                     =  True
res@gsnMaximize         =  True

;-- create the contour plot
plot = gsn_csm_contour(wks,T,res)
```

```
#-- resource settings
res                          =  Ngl.Resources()
res.nglPointTickmarksOutward = True        #-- tickmarks outward

#-- create the contour plot
plot = Ngl.contour(wks,T,res)

Ngl.end()
```

6.3.2 Contour Fill Plot

The example shows how to create a simple contour fill plot.





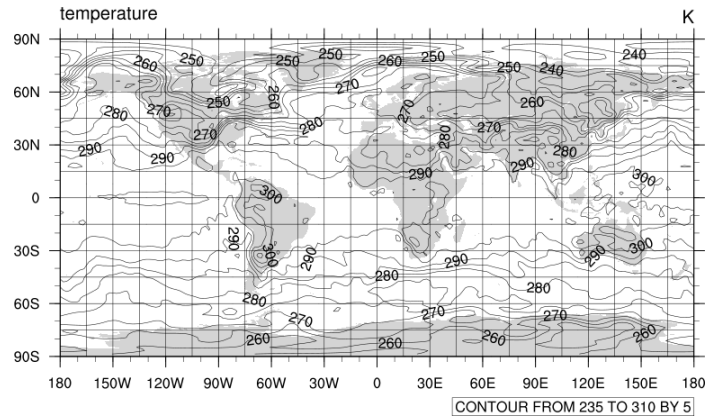
NCL		PyNGL/PyNIO
;-- create some dummy data to contour T = new((/25,25/),float) vals = ispan(-12,12,1) jspn = vals*vals ispn = vals*vals do i=0,dimsizes(jspn)-1 T(i,:) = tofloat(jspn + ispn(i))		import numpy as np import Ngl #-- create some dummy data to contour T = np.zeros((25,25),np.float32) jspn = np.power(np.arange(-12,13),2) ispn = np.power(np.arange(-12,13),2) for i in range(0,len(ispn)): T[i,:] = (jspn + ispn[i]).astype(np.float32)

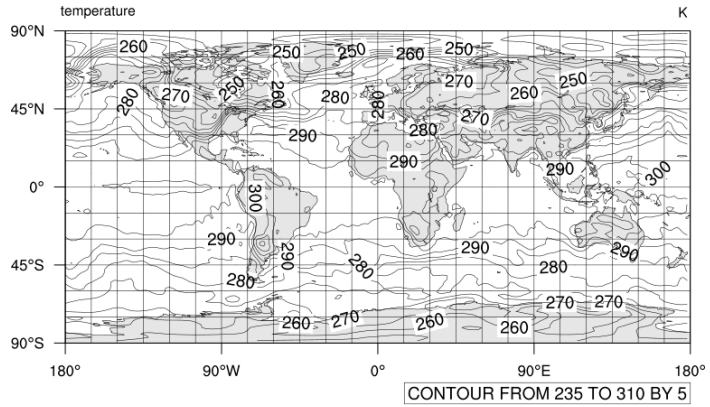
```
end do

T = 100.0 - sqrt(64 * T)

;-- start the graphics
wks = gsn_open_wks("png","plot_TRANS_contour_fill_ncl")

;-- resource settings
res                     =  True
res@gsnMaximize         =  True



res@cnFillOn            =  True         ;-- contour fill
res@cnLineLabelsOn      =  False        ;-- no line labels
res@cnFillPalette       = "ncl_default"

res@lbLabelFontHeightF =  0.018         ;-- lb font size
res@lbLabelAlignment    = "BoxCenters"
res@lbLabelPosition     = "Center"
res@lbLabelStrings      =  ispan(-30,110,10)
res@lbOrientation       = "vertical"    ;-- vertical lb

;-- create the contour plot
plot = gsn_csm_contour(wks,T,res)
```

```
T = 100.0 - np.sqrt(64 * T)



#-- start the graphics
wks = Ngl.open_wks("png","plot_TRANS_contour_fill_py")

#-- resource settings
res                     =  Ngl.Resources()
res.nglDraw             =  False         #-- don't draw plot
res.nglFrame            =  False         #-- don't advance  frame
res.nglPointTickmarksOutward = True      #-- tickmarks outward

res.cnFillOn            =  True          #-- contour fill
res.cnLineLabelsOn      =  False         #-- no line labels
res.cnFillPalette       = "ncl_default"

res.lbLabelPosition     = "Center"
res.lbLabelFontHeightF =  0.018          #-- lb font size
res.lbLabelAlignment    = "BoxCenters"
res.lbLabelStrings      =  list(range(-30,110,10))
                                         #-- doesn't appear to work

#-- create the contour plot
plot = Ngl.contour(wks,T,res)

#-- bug work-around (v1.5.2): apply the labels after the plot
#--                           has been created
nrlist                   =  Ngl.Resources()
nrlist.lbLabelStrings    =  list(range(-30,110,10))
Ngl.set_values(plot,nrlist)

#-- draw the plot and advance the frame
Ngl.draw(plot)
Ngl.frame(wks)

Ngl.end()
```

6.3.3 Contour Lines on Maps

The next example reads a netCDF file and plot the data as contour lines on a map.



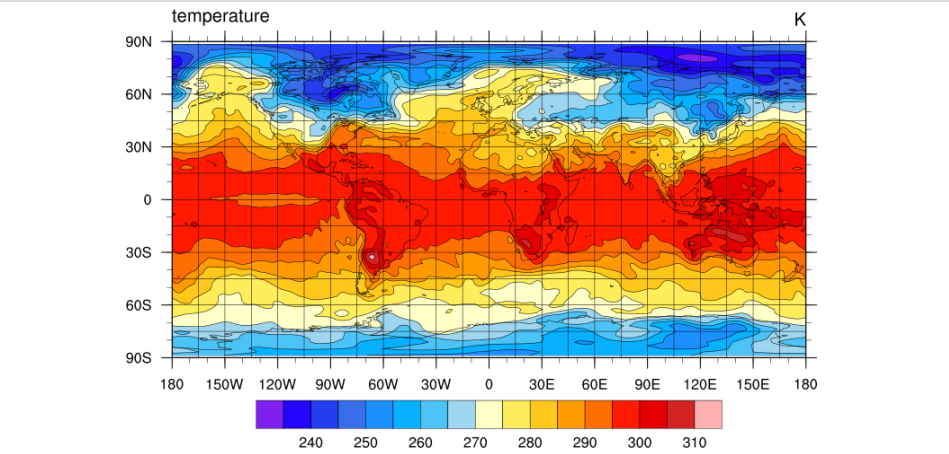


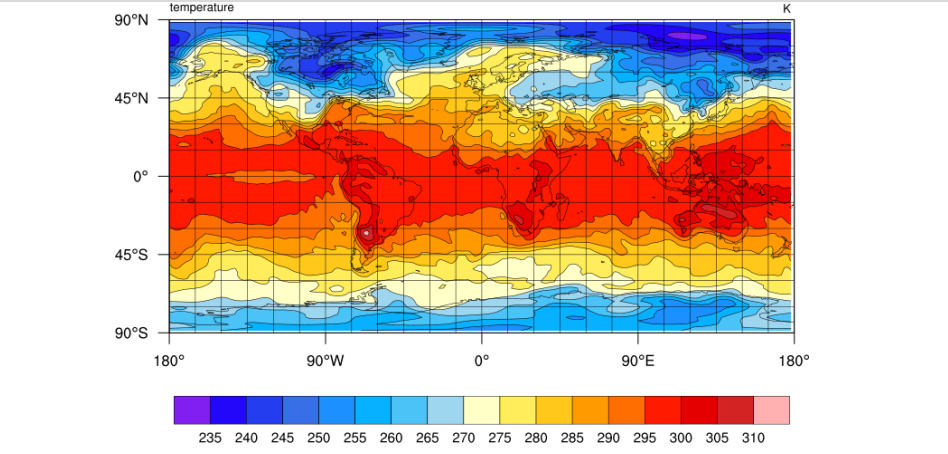
NCL		PyNGL/PyNIO
;-- open file and read variables f = addfile("../read_data/rectilinear_grid_3D.nc","r") var = f->t(0,0,:,:) ;-- start the graphics wks = gsn_open_wks("png","plot_TRANS_contour_lines_map_ncl") ;-- resource settings res = True res@gsnMaximize = True res@mpGridAndLimbOn = True ;-- draw grid lines ;-- create the contour plot plot = gsn_csm_contour_map(wks,var,res)		import numpy as np import Ngl,Nio #-- open file and read variables f = Nio.open_file("../read_data/rectilinear_grid_3D.nc","r") var = f.variables["t"][0,0,:,:] lat = f.variables["lat"][:] lon = f.variables["lon"][:] #-- start the graphics wks = Ngl.open_wks("png","plot_TRANS_contour_lines_on_map_py") #-- resource settings res = Ngl.Resources() res.nglFrame = False res.lbOrientation = "horizontal" #-- horizontal labelbar res.sfXArray = lon res.sfYArray = lat res.mpFillOn = True res.mpOceanFillColor = "Transparent" res.mpLandFillColor = "Gray90" res.mpInlandWaterFillColor = "Gray90" #-- create the contour plot plot = Ngl.contour_map(wks,var,res) #-- write variable long_name and units to the plot

```
txres               = Ngl.Resources()
txres.txFontHeightF = 0.014

Ngl.text_ndc(wks,f.variables["t"].attributes['long_name'],\
                 0.14,0.78,txres)
Ngl.text_ndc(wks,f.variables["t"].attributes['units'], \
                 0.95,0.78,txres)

#-- advance the frame
Ngl.frame(wks)

Ngl.end()
```

6.3.4 Contour Fill on Maps

The next example reads a netCDF file and plot the data as color filled contour areas on a map.





NCL		PyNGL/PyNIO
``;-- open file and read variables f = addfile("../read_data/rectilinear_grid_3D.nc","r") var = f->t(0,0,:,:) ;-- start the graphics wks = gsn_open_wks("png","plot_TRANS_contour_fill_on_map_ncl")```		```import numpy as np import Ngl,Nio #-- open file and read variables f = Nio.open_file("../read_data/rectilinear_grid_3D.nc", "r") var = f.variables["t"][0,0,:,:] lat = f.variables["lat"][:] lon = f.variables["lon"][:] #-- start the graphics```

```
;-- resource settings
res                  =  True
res@gsnMaximize      =  True

res@cnFillOn         =  True       ;-- contour fill
res@cnLineLabelsOn   =  False      ;-- contour line labels

res@mpGridAndLimbOn =  True     ;-- draw grid lines




;-- create the contour plot
plot = gsn_csm_contour_map(wks,var,res)
```

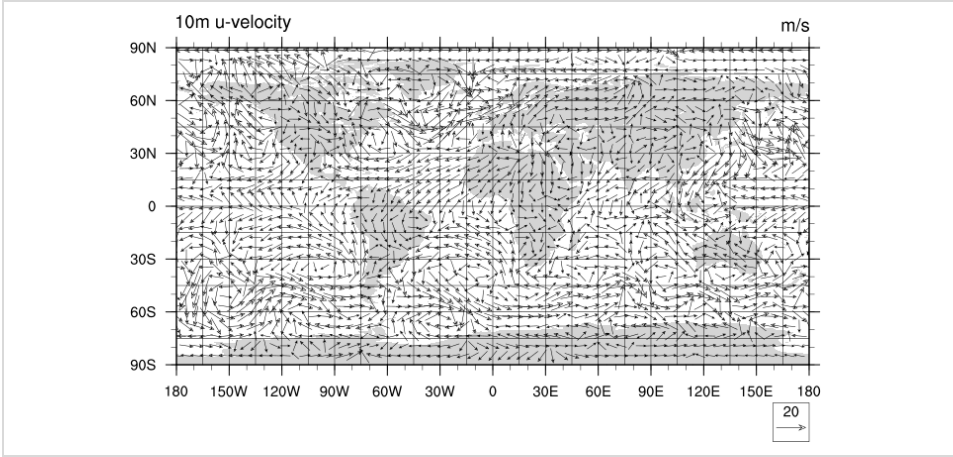
```
wks = Ngl.open_wks("png","plot_TRANS_contour_fill_on_map_py")

#-- resource settings
res                  =  Ngl.Resources()
res.nglFrame         =  False

res.cnFillOn         =  True
res.cnFillPalette    = "NCL_default"
res.cnLineLabelsOn   =  False

res.lbOrientation    = "horizontal"      #-- horizontal labelbar

res.sfXArray         =  lon
res.sfYArray         =  lat

#-- create the contour plot
plot = Ngl.contour_map(wks,var,res)

#-- write variable long_name and units to the plot
txres                = Ngl.Resources()
txres.txFontHeightF = 0.012

Ngl.text_ndc(wks,f.variables["t"].attributes['long_name'],\
                 0.14,0.82,txres)
Ngl.text_ndc(wks,f.variables["t"].attributes['units'], \
                 0.95,0.82,txres)

#-- advance the frame
Ngl.frame(wks)

Ngl.end()
```

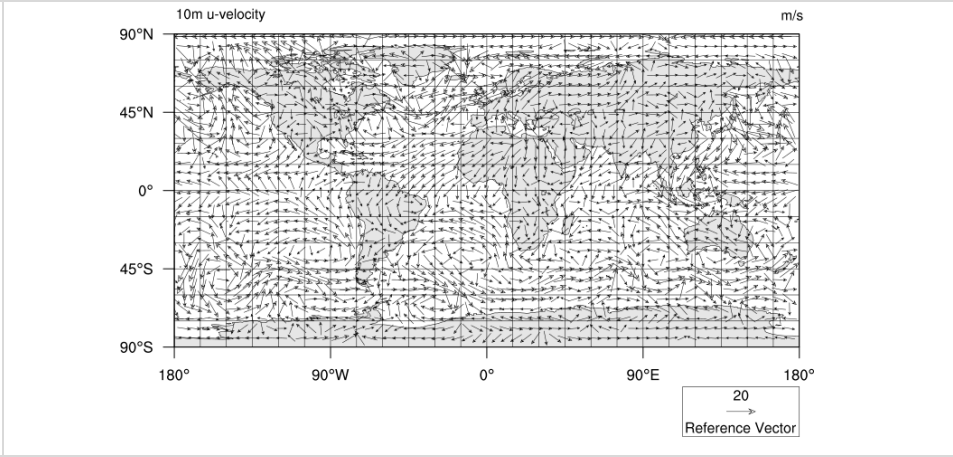
6.4 Vector and Streamline Plots

To plot vectors and streamlines two variables are needed which are in the following examples u-wind and v-wind.

6.4.1 Vector Plot on Maps

The example shows how to plot the variables u10 and v10 as vectors.





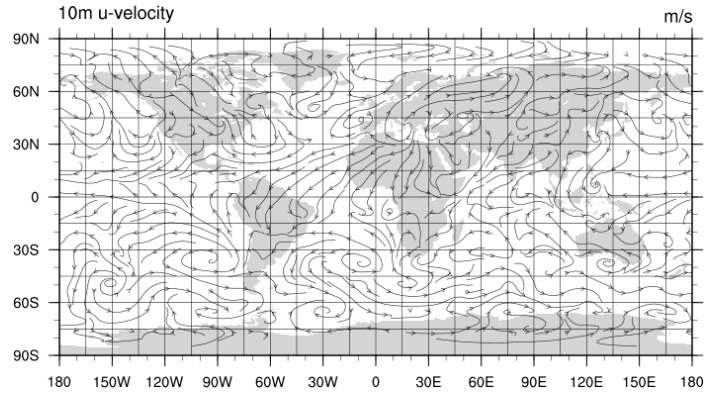
NCL		PyNGL/PyNIO
;-- open a file and read variables f = addfile("rectilinear_grid_2D.nc", "r") u = f->u10(0,:,:) ;-- first time step v = f->v10(0,:,:) ;-- first time step ;-- open a workstation wks = gsn_open_wks("png","plot_TRANS_vectors_ncl") ;-- resource settings		import Ngl,Nio #-- open a file and read variables f = Nio.open_file("rectilinear_grid_2D.nc", "r") u = f.variables["u10"] v = f.variables["v10"] ua = f.variables["u10"][0,:,:] va = f.variables["v10"][0,:,:] lat = f.variables["lat"] lon = f.variables["lon"] nlon = len(lon) nlat = len(lat) #-- open a workstation wks = Ngl.open_wks("png","plot_TRANS_vectors_py") #-- resource settings

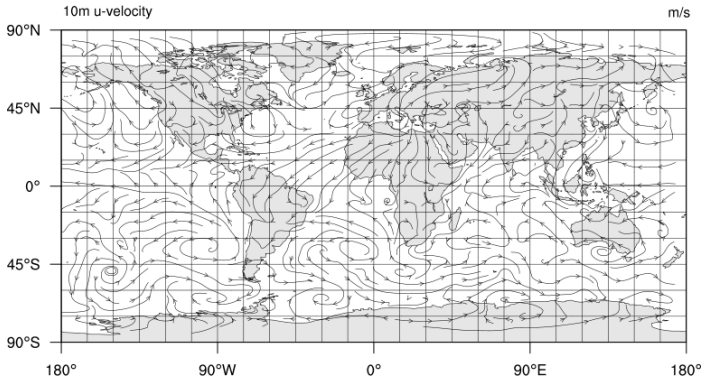
```
vcres                   = True

vcres@vcMinFracLengthF = 0.3    ;-- length of smallest vec
vcres@vcRefLengthF     = 0.05   ;-- length of ref vector
vcres@vcRefMagnitudeF  = 20.0   ;-- vector ref mag
vcres@vcRefLengthF     = 0.035  ;-- length of vec ref

vcres@mpGridAndLimbOn  = True   ;-- draw grid lines




;-- create the plot
plot = gsn_csm_vector_map(wks,u(::3,::3),v(::3,::3),vcres)
```

```
vcres                   =  Ngl.Resources()
vcres.nglFrame          =  False

vcres.vfXArray          =  lon[::3]
vcres.vfYArray          =  lat[::3]

vcres.vcMinFracLengthF =  0.3        #-- length of smallest vec
vcres.vcRefLengthF     =  0.05       #-- length of ref vec
vcres.vcRefMagnitudeF  =  20.0       #-- vector ref mag
vcres.vcRefLengthF     =  0.035      #-- length of vec ref

vcres.mpFillOn          =  True
vcres.mpOceanFillColor = "Transparent"
vcres.mpLandFillColor  = "Gray90"
vcres.mpInlandWaterFillColor = "Gray90"

#-- create the plot
plot = Ngl.vector_map(wks,ua[::3,::3],va[::3,::3],vcres)

#-- write variable long_name and units to the plot
txres               = Ngl.Resources()
txres.txFontHeightF = 0.014

Ngl.text_ndc(wks,f.variables["u10"].attributes['long_name'],\
                 0.16,0.8,txres)
Ngl.text_ndc(wks,f.variables["u10"].attributes['units'],\
                 0.95,0.8,txres)

#-- advance the frame
Ngl.frame(wks)

Ngl.end()
```

6.4.2 Streamline Plot on Maps

The example shows how to plot the variables u10 and v10 as streamlines.





NCL

```
;-- open a file and read variables
f =  addfile("rectilinear_grid_2D.nc","r")

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




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

;-- resource settings
stres                  =  True

stres@mpGridAndLimbOn =  True            ;-- draw grid lines
```

PyNGL/PyNIO

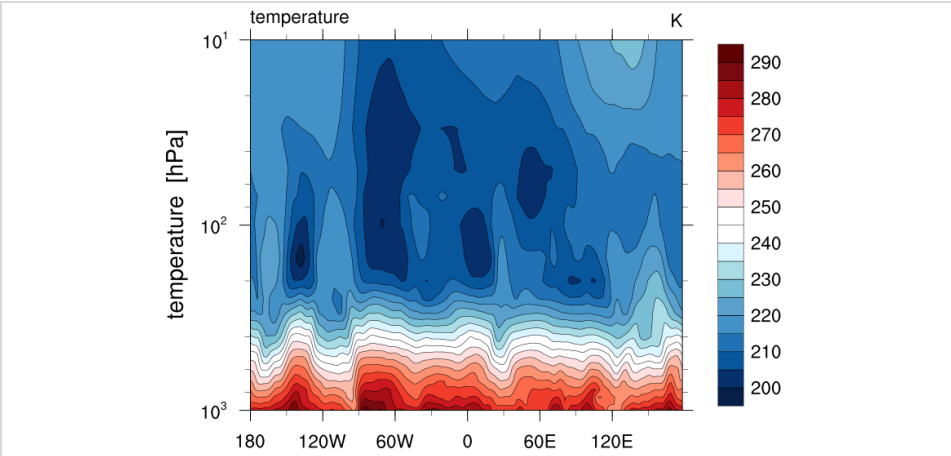
```
import Ngl,Nio

#-- open a file and read variables
f    = Nio.open_file("rectilinear_grid_2D.nc","r")

u    = f.variables["u10"]
v    = f.variables["v10"]
ua   = f.variables["u10"][0,:,:]
va   = f.variables["v10"][0,:,:]

lat  = f.variables["lat"]
lon  = f.variables["lon"]
nlon = len(lon)
nlat = len(lat)

#-- open a workstation
wks = Ngl.open_wks("png","plot_TRANS_streamline_py")

#-- resource settings
stres                  =  Ngl.Resources()
stres.nglFrame         =  False

stres.vfXArray         = lon[::3]
stres.vfYArray         = lat[::3]

stres.mpFillOn         =  True
stres.mpOceanFillColor = "Transparent"
stres.mpLandFillColor  = "Gray90"
stres.mpInlandWaterFillColor = "Gray90"
```

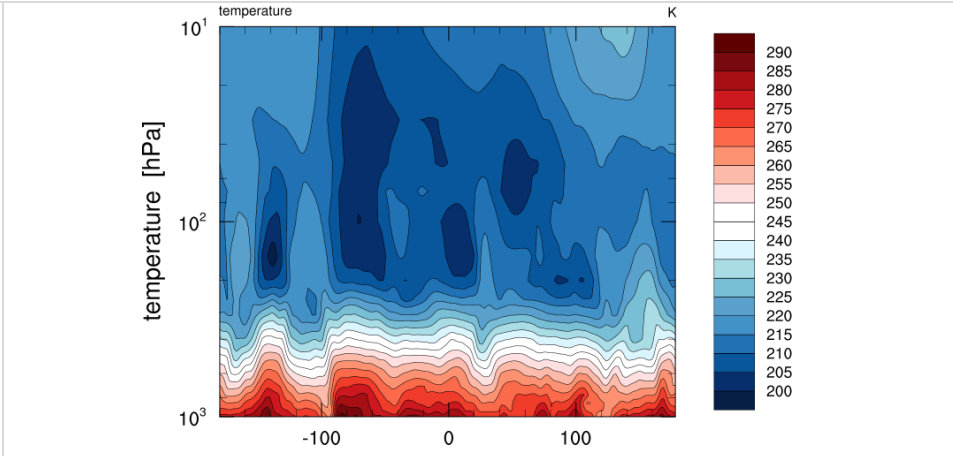
```
;-- create the plot
plot = gsn_csm_streamline_map(wks,u(::3,::3),v(::3,::3),stres)
```

```
#-- create the plot
plot = Ngl.streamline_map(wks,ua[::3,::3],va[::3,::3],stres)

#-- write variable long_name and units to the plot
txres               = Ngl.Resources()
txres.txFontHeightF = 0.014

Ngl.text_ndc(wks,f.variables["u10"].attributes['long_name'],\
                 0.16,0.76,txres)
Ngl.text_ndc(wks,f.variables["u10"].attributes['units'],\
                 0.95,0.76,txres)

#-- advance the frame
Ngl.frame(wks)

Ngl.end()
```

6.5 Slices

A slice is a cut through a 3D variable's space. The example shows how to define the slice itself and plot it as a contour fill plot.





NCL

```
;-- open file and read variables
fname  = rectilinear_grid_3D.nc"

f      =  addfile(fname, "r")
var    =  f->t(0,:,{40},:)     ;-- first time step,lat=40N


lon_t =  f->lon                ;-- lon
lev_t =  f->lev                ;-- currently 17 levels









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

PyNGL/PyNIO

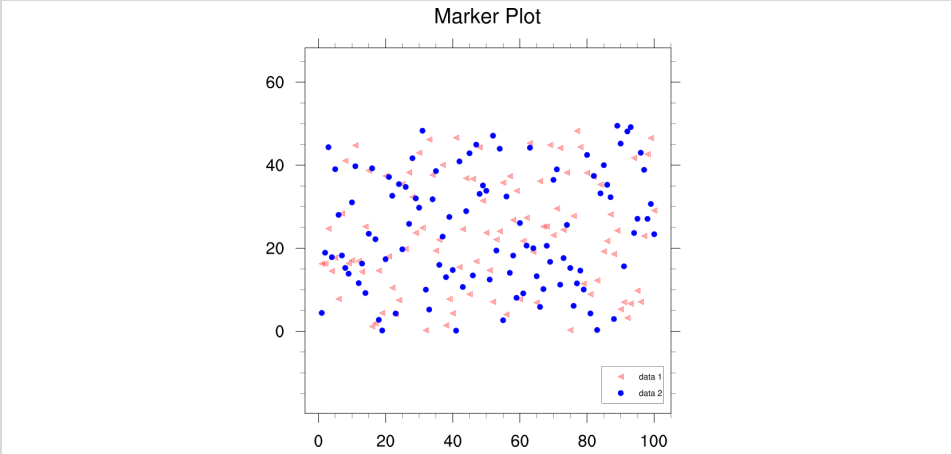
```
import numpy as np
import Ngl,Nio

#-- open file and read variables
fname  = rectilinear_grid_3D.nc"     #-- data file name

f    =  Nio.open_file(fname,"r")     #-- open data file
t    =  f.variables["t"][0,:,::-1,:] #-- first time step,
                                     #-- rev. lat
lev  =  f.variables["lev"][:]        #-- all levels
lat  =  f.variables["lat"][::-1]     #-- reverse latitudes
lon  =  f.variables["lon"][:]        #-- all longitudes
nlat =  len(lat)                     #-- number of latitudes

longname = f.variables["t"].attributes['long_name']
units    = f.variables["t"].attributes['units']

ind40    = 69                    #-- index close to lat 40 deg
t40      = t[:,ind40,:]          #-- variable at lat ~40 deg
strlat40 = lat[ind40]            #-- retrieve data of lat ~40
degrees

#-- open a workstation
wks =  Ngl.open_wks("png","plot_TRANS_slice_py")
```

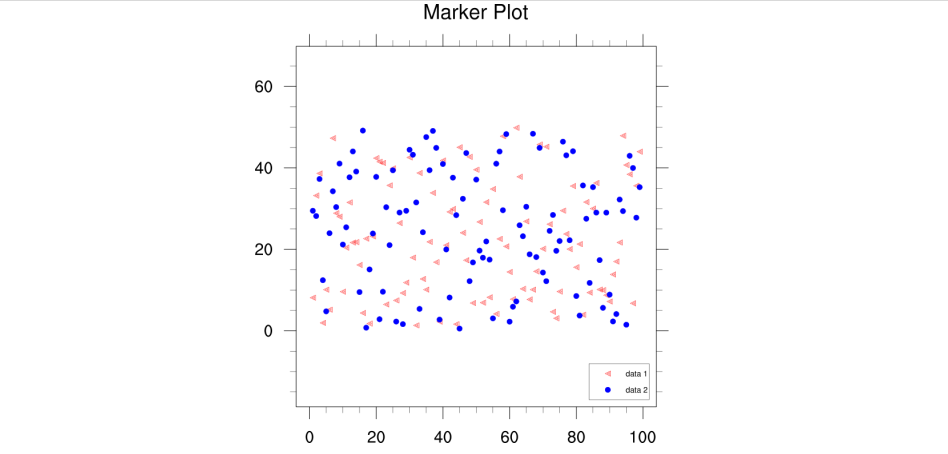
```
;-- set resources
res                  =  True
res@gsnMaximize      =  True

;-- viewport resources
res@vpXF             =  0.1          ;-- viewport x-pos
res@vpYF             =  0.9          ;-- viewport y-pos
res@vpWidthF         =  0.7          ;-- viewport width
res@vpHeightF        =  0.6          ;-- viewport height

res@cnFillOn         =  True         ;-- turn on color fill
res@cnFillPalette    = "temp_diff_18lev"
res@cnLineLabelsOn   =  False        ;-- no line labels
res@cnInfoLabelOn    =  False        ;-- no info label
res@cnLevelSelectionMode = "ManualLevels"   ;--manual levs
res@cnMinLevelValF   =  200.         ;-- min contour value
res@cnMaxLevelValF   =  290.         ;-- max contour value
res@cnLevelSpacingF =  5.            ;-- contour increment

res@lbOrientation    = "vertical"    ;-- vertical label bar
res@tiYAxisString    =  var@long_name+"  [hPa]"


res@sfXArray         =  lon_t       ;-- lon_t as plot x-axis
res@sfYArray         =  lev_t/100   ;-- lev_t in hPa as plot
                                    ;-- y-axis

res@trYReverse       =  True         ;-- reverses y-axis
res@gsnYAxisIrregular2Log =  True    ;-- irregular to linear
                                    ;-- depth

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

```
#-- set resources
res                  =  Ngl.Resources() #-- res object for plot
res.nglFrame         =  False

#-- viewport resources
res.vpXF             =  0.1             #-- x-pos of viewport
res.vpYF             =  0.9             #-- y-pos of viewport
res.vpWidthF         =  0.7             #-- viewport width
res.vpHeightF        =  0.6             #-- viewport height

res.cnFillOn         =  True            #-- turn on contour fill
res.cnFillPalette    = "temp_diff_18lev"
res.cnLineLabelsOn   =  False           #-- turn off line labels
res.cnInfoLabelOn    =  False           #-- turn off info label
res.cnLevelSelectionMode = "ManualLevels"#-- select manual levs
res.cnMinLevelValF   =  200.            #-- minimum contour value
res.cnMaxLevelValF   =  290.            #-- maximum contour value
res.cnLevelSpacingF =  5.                 #-- contour increment

res.tiYAxisString    =  longname+"  [hPa]"



res.sfXArray         =  lon               #-- scalar field x
res.sfYArray         =  lev/100           #-- scalar field y


res.trYReverse       =  True              #-- reverse the Y axis
res.nglYAxisType     = "LogAxis"          #-- y axis log


#-- draw slice contour plot
plot = Ngl.contour(wks,t40,res)

#-- write variable long_name and units to the plot
txres                = Ngl.Resources()
txres.txFontHeightF = 0.014

Ngl.text_ndc(wks,longname,0.18,0.81,txres)
Ngl.text_ndc(wks,units,   0.77,0.81,txres)

#-- advance the frame
Ngl.frame(wks)

#-- done
Ngl.end()
```

6.6 Scatter Plots

A scatter plot is an xy-plot where the data is plotted as points. The example shows how to plot two data sets with different markers and colors.





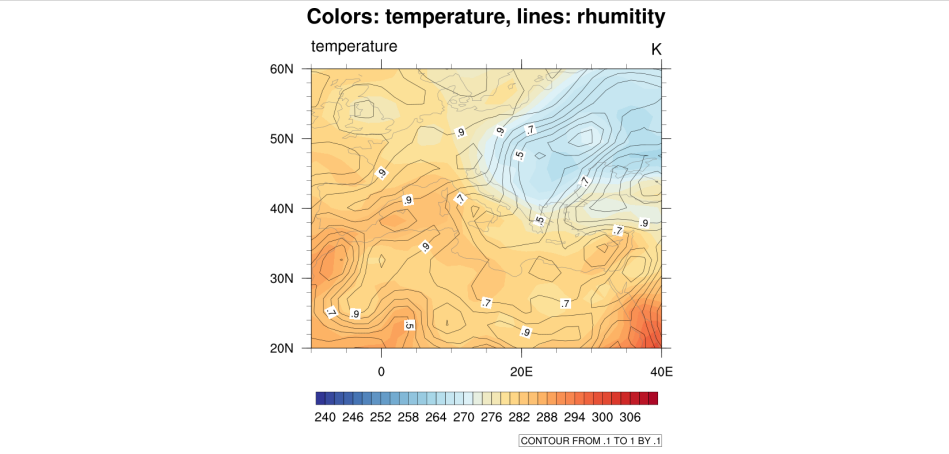
NCL		PyNGL/PyNIO
`` ;-- generate random data npts = 100 x = ispan(1,npts,1) data1 = random_uniform(0.1,50,npts) data2 = random_uniform(0.1,50,npts) ;-- create data array containing data1 and data2 data = new((/2,npts/),typeof(data1)) data(0,:) = data1 data(1,:) = data2 ;-- set explicit labels for legend labels = (/"data 1","data 2"/) ;-- open workstation wks = gsn_open_wks("png","plot_scatter_ncl") ;-- set resources res = True res@gsnMaximize = True ;-- maximize plot output res@tiMainString = "Marker Plot" ;-- add title ```		``` import numpy as np import Ngl #-- generate random data npts = 100 x = np.arange(1,npts,1) data1 = np.random.uniform(0.1,50,npts) data2 = np.random.uniform(0.1,50,npts) #-- create data array containing data1 and data2 data = np.zeros([2,npts],np.float32) data[0,:] = data1 data[1,:] = data2 #-- set explicit labels for legend labels = ["data 1","data 2"] #-- open workstation wks = Ngl.open_wks("png","plot_TRANS_scatter_py") #-- set resources res = Ngl.Resources() res.nglPointTickmarksOutward = True #-- point tickmarks outward ```

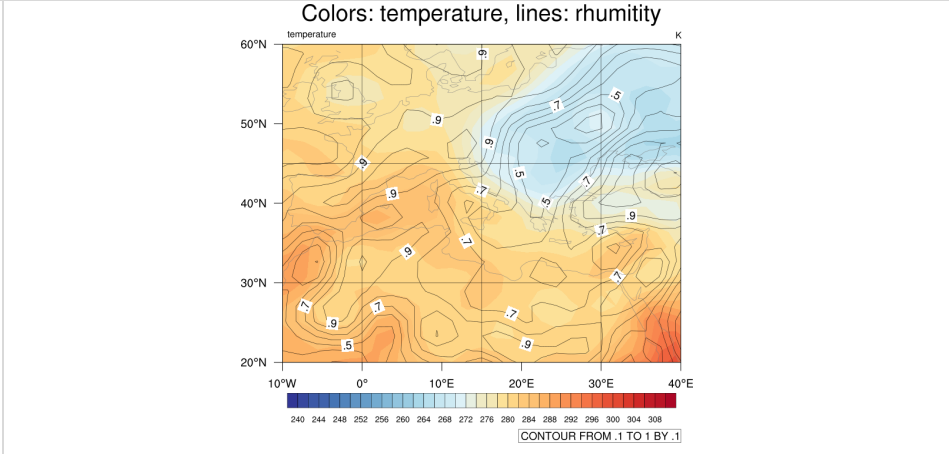
```
;-- make plot space larger to have enough space for legend
res@trYMinF          = min(data1)-20.   ;-- y-axis minimum
res@trYMaxF          = max(data1)+20.   ;-- y-axis maximum
res@trXMinF          = min(x)-5.        ;-- y-axis minimum
res@trXMaxF          = max(x)+5.        ;-- y-axis maximum

res@tmLabelAutoStride = True ;-- use nice tick mark labels

res@xyMarkLineModes  = (/"Markers","Markers"/)
                ;-- set mark line mode for both variables
res@xyMarkers        = (/10,16/)          ;-- marker types
res@xyMarkerColors   = (/"red","blue"/)  ;-- marker colors

res@lgJustification  = "TopRight"      ;-- legend position
res@lgLabelFontHeightF = 0.01   ;-- legend label font size
res@lgItemOrder      = (/1,0/)      ;-- reverse the legend
res@xyExplicitLabels = labels       ;-- use explicit
                                    ;-- legend labels

res@pmLegendDisplayMode = "Always"  ;-- display legend
res@pmLegendWidthF   = 0.10         ;-- legend width
res@pmLegendHeightF  = 0.06         ;-- legend height
res@pmLegendOrthogonalPosF = -0.22  ;-- move legend up
res@pmLegendParallelPosF = 0.98     ;-- move legend right

;-- create the plot
plot = gsn_csm_xy(wks,x,data,res)
```

```
res.tiMainString     = "Marker Plot"   #-- add title

#-- make plot space larger to have enough space for legend
res.trYMinF          = min(data1)-20.  #-- y-axis minimum
res.trYMaxF          = max(data1)+20.  #-- y-axis maximum
res.trXMinF          = min(x)-5.       #-- y-axis minimum
res.trXMaxF          = max(x)+5.       #-- y-axis maximum

res.tmLabelAutoStride = True       #-- use nice tick mark labels

res.xyMarkLineModes  = ["Markers","Markers"]
                                       #-- set mark line mode
res.xyMarkers        = [10,16]         #-- choose marker types
res.xyMarkerColors   = ["red","blue"] #-- choose marker colors

res.lgJustification  = "TopRight"      #-- position of legend
res.lgLabelFontHeightF = 0.01          #-- legend label font size
res.lgItemOrder      = [1,0]           #-- reverse the legend
res.xyExplicitLabels = labels     #-- use explicit legend labels


res.pmLegendDisplayMode = "Always"     #-- display legend always
res.pmLegendWidthF   = 0.10            #-- legend width
res.pmLegendHeightF  = 0.06            #-- legend height
res.pmLegendOrthogonalPosF = -0.22     #-- move legend up
res.pmLegendParallelPosF = 0.98        #-- move legend right

#-- create the plot
plot = Ngl.xy(wks,x,data,res)
```

6.7 Overlays

The example shows how to plot one variable as contour fill plot on a map and the second variable on top of the plot as contour lines.





NCL

```
#-- open file and read variables
f     = addfile("../read_data/rectilinear_grid_3D.nc", "r")
t     = f->t(0,0,:,:)          ;-- 1st time step, 1st level
rhum  = f->rhumidity(0,0,:,:)  ;-- 1st time step, 1st level



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

;-- set resources
res                  = True
res@gsnDraw          = False      ;-- don't draw plot
res@gsnFrame         = False      ;-- don't advance frame

;-- plot resources for the map
tres@mpGeophysicalLineColor = "gray50";-- map outline color
tres@mpMinLatF       =  20.0      ;-- min lat
tres@mpMaxLatF       =  60.0      ;-- max lat
tres@mpMinLonF       = -10.0      ;-- min lon
tres@mpMaxLonF       =  40.0      ;-- max lon
```

PyNGL/PyNIO

```
import numpy as np
import Ngl,Nio

#-- open file and read variables
f    = Nio.open_file("rectilinear_grid_3D.nc", "r")
t    = f.variables["t"][0,0,:,:]
rhum = f.variables["rhumidity"][0,0,:,:]
lat  = f.variables["lat"][:]
lon  = f.variables["lon"][:]

#-- define workstation
wks =  Ngl.open_wks("png","plot_TRANS_overlay_py")

#-- plot resources for the map
mpres                =  Ngl.Resources()
mpres.nglDraw        =  False   #-- don't draw plot
mpres.nglFrame       =  False   #-- don't advance frame
mpres.mpOutlineOn    =  True    #-- turn on map outlines
mpres.mpGeophysicalLineColor = "gray50"
                                #-- map outline color
mpres.mpLimitMode    = "LatLon" #-- limit map via lat/lon
mpres.mpMinLatF      =  20.0    #-- min lat
mpres.mpMaxLatF      =  60.0    #-- max lat
mpres.mpMinLonF      = -10.0    #-- min lon
```

```
;-- plot resources for the temperature plot
tres = res
tres@gsnMaximize    = True       ;-- maximize plot output

tres@cnFillOn       = True       ;-- turn on color fill
tres@cnFillPalette  = "cmp_b2r"  ;-- set the colormap to
                                 ;-- be used
tres@cnLinesOn      = False      ;-- turn off contour
                                 ;-- line labels
tres@cnLineLabelsOn = False;-- turn off contour line labels
tres@cnInfoLabelOn  = False ;-- turn off contour info label
tres@cnLevelSelectionMode = "ManualLevels"
                                 ;-- select manual levels
tres@cnMinLevelValF =  240.      ;-- minimum contour value
tres@cnMaxLevelValF =  310.      ;-- maximum contour value
tres@cnLevelSpacingF =  2.       ;-- contour increment

tres@tiMainString = "Colors: temperature, lines: rhumitity"
                                  ;-- title string
tres@lbBoxMinorExtentF =  0.17    ;-- decrease height of
                                  ;-- labelbar box




;-- plot resources for the rhumidity plot
rres = res

rres@gsnLeftString   = ""      ;-- don't draw left string
rres@gsnRightString  = ""      ;-- don't draw right string

rres@cnInfoLabelOrthogonalPosF = -0.05 ;-- move info \
                                       ;-- label upward




;-- generate tplot, but don't draw it yet
tplot = gsn_csm_contour_map(wks,t,tres)

;-- generate plot2, but don't draw it yet
rplot = gsn_csm_contour(wks,rhum,rres)

;-- overlay rplot on tplot
```

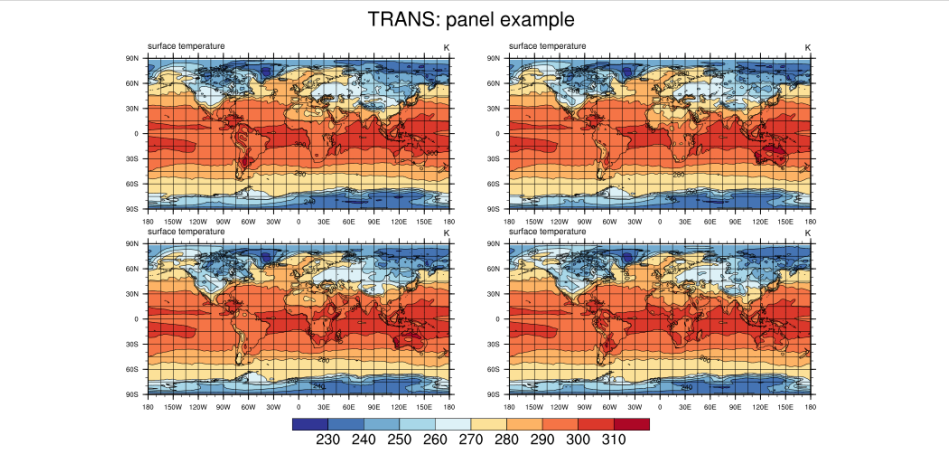
```
mpres.mpMaxLonF     =  40.0    #-- max lon

#-- plot resources for the temperature plot
tres                =  Ngl.Resources()
tres.nglDraw        =  False  #-- don't draw plot
tres.nglFrame       =  False  #-- don't advance frame
tres.cnFillOn       =  True   #-- turn on color fill
tres.cnFillPalette  = "cmp_b2r"#-- set the colormap to be used
tres.cnLinesOn      =  False  #-- turn off contour line labels
tres.cnLineLabelsOn =  False  #-- turn off contour line labels
tres.cnInfoLabelOn  =  False  #-- turn off contour info label

tres.cnLevelSelectionMode = "ManualLevels"
                                #-- select manual levels
tres.cnMinLevelValF  =  240.   #-- minimum contour value
tres.cnMaxLevelValF  =  310.   #-- maximum contour value
tres.cnLevelSpacingF =  2.     #-- contour increment

tres.pmLabelBarOrthogonalPosF = -0.26 #-- move labelbar upward
tres.lbLabelFontHeightF =  0.009  #-- labelbar labe font size
tres.lbBoxMinorExtentF  =  0.17   #-- decrease height of
                                  #-- labelbar box
tres.lbOrientation = "horizontal" #-- horizontal labelbar

tres.tiMainString  = "Colors: temperature, lines: rhumitity"
                                  #-- title string
tres.sfXArray      =  lon
tres.sfYArray      =  lat

#-- plot resources for the rhumidity plot
rres                =  Ngl.Resources()
rres.nglDraw        =  False       #-- don't draw plot
rres.nglFrame       =  False       #-- don't advance frame

rres.cnInfoLabelOrthogonalPosF = 0.13 #-- move info label
                                  #-- upward
rres.sfXArray      =  lon
rres.sfYArray      =  lat

#-- generate tplot, but don't draw it yet
map   = Ngl.map(wks,mpres)

#-- generate tplot, but don't draw it yet
tplot = Ngl.contour(wks,t,tres)

#-- generate plot2, but don't draw it yet
rplot = Ngl.contour(wks,rhum,rres)

#-- overlay rplot on tplot
```

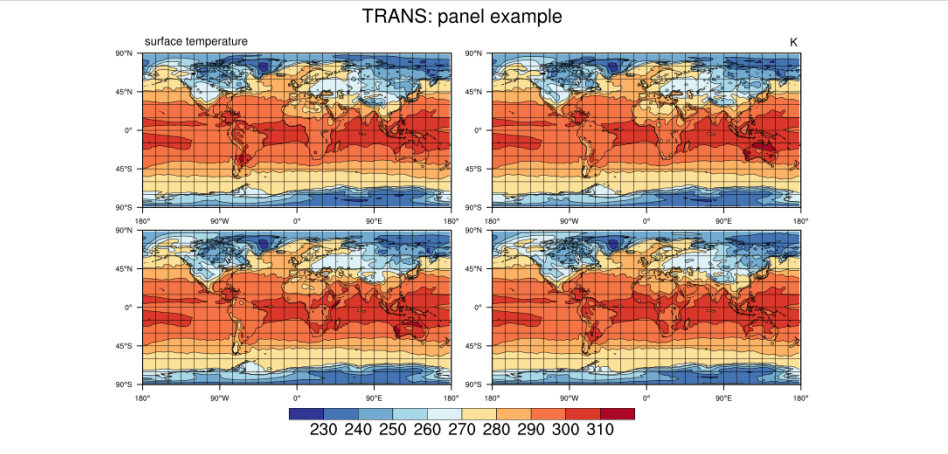
```
overlay(tplot, rplot)

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

```
Ngl.overlay(map, tplot)
Ngl.overlay(map, rplot)

#-- draw the plot
Ngl.draw(map)

#-- write variable long_name and units to the plot
txres               = Ngl.Resources()
txres.txFontHeightF = 0.014
Ngl.text_ndc(wks,f.variables["t"].attributes['long_name'],\
                 0.17,0.88,txres)
Ngl.text_ndc(wks,f.variables["t"].attributes['units'], \
                 0.95,0.88,txres)

#-- advance the frame
Ngl.frame(wks)
```

6.8 Panel Plots

A panel plot means two or more plots are positioned on the same page (= frame). The example shows how to create a panel for four plots, two per row and two per column.





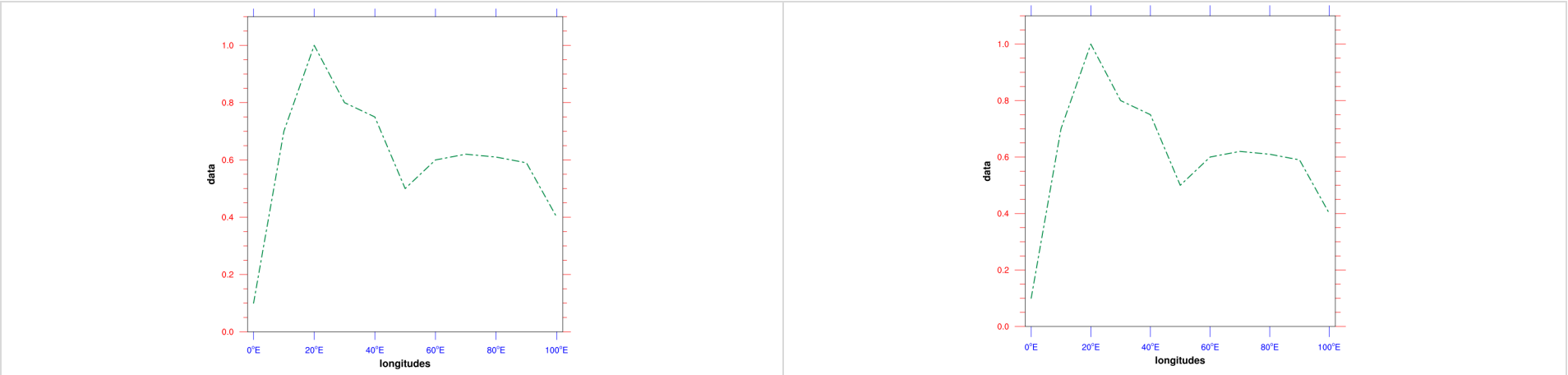
NCL		PyNGL/PyNIO
`` ;-- read data and set variable references f = addfile("rectilinear_grid_2D.nc", "r") t = f->tsurf ;-- open a PNG file wks = gsn_open_wks("png","plot_TRANS_panel_ncl") ;-- set resources for contour plots res = True res@gsnDraw = False ;-- don't draw plot, yet res@gsnFrame = False ;-- don't advance frame res@gsnMaximize = True ;-- maximize plots res@cnFillOn = True ;-- contour fill res@cnFillPalette = "cmp_b2r" ;-- choose color map res@mpGridAndLimbOn = True ;-- draw grid lines res@lbLabelBarOn = False ;-- don't draw a labelbar ```		``` import Ngl, Nio #-- open file and read variables f = Nio.open_file("rectilinear_grid_2D.nc", "r") var = f.variables["tsurf"] lat = f.variables["lat"][:] lon = f.variables["lon"][:] #-- start the graphics wks = Ngl.open_wks("png","plot_TRANS_panel_py") #-- set resources for contour plots res = Ngl.Resources() res.nglDraw = False #-- don't draw plots res.nglFrame = False #-- don't advance the frame res.cnFillOn = True #-- contour fill res.cnFillPalette = "cmp_b2r" #-- choose color map res.cnLineLabelsOn = False #-- no line labels res.lbLabelBarOn = False #-- don't draw a labelbar ```

```
;-- create 4 plots time step 1 to 4 (NCL index 0-3)
plot_1 = gsn_csm_contour_map(wks,t(0,:,:),res)
plot_2 = gsn_csm_contour_map(wks,t(1,:,:),res)
plot_3 = gsn_csm_contour_map(wks,t(2,:,:),res)
plot_4 = gsn_csm_contour_map(wks,t(3,:,:),res)


;-- panel resources
pnlres                  =  True
pnlres@gsnPanelLabelBar =  True         ;-- common labelbar
pnlres@gsnPanelXWhiteSpacePercent =  5
pnlres@gsnPanelMainFontHeightF =  0.020 ;-- text font size
pnlres@gsnPanelMainString = "TRANS: panel example"

;-- create panel plot
gsn_panel(wks,(/plot_1,plot_2,plot_3,plot_4/),(/2,2/),pnlres)
```

```
res.sfXArray        =  lon       #-- coordinates for the x-axis
res.sfYArray        =  lat       #-- coordinates for the y-axis

#-- create the contour plots
plot = []
for i in range(0,4):
     p = Ngl.contour_map(wks,var[i,:,:],res)
     plot.append(p)

#-- panel resources
pnlres = Ngl.Resources()
pnlres.nglFrame         = False   #-- don't advance the frame
pnlres.nglPanelLabelBar = True    #-- common labelbar
pnlres.txString         = "TRANS: panel example"#-- panel title
pnlres.txFontHeightF    =  0.02   #-- text font size


#-- create the panel plot
Ngl.panel(wks,plot[0:4],[2,2],pnlres)

#-- add title string,long_name and units string to panel
txres = Ngl.Resources()
txres.txFontHeightF = 0.020
Ngl.text_ndc(wks,"TRANS: panel example",0.5,0.825,txres)

txres.txFontHeightF = 0.012
Ngl.text_ndc(wks,f.variables["tsurf"].attributes['long_name'],\
                 0.12,0.79,txres)
Ngl.text_ndc(wks,f.variables["tsurf"].attributes['units'], \
                 0.975,0.79,txres)

#-- advance the frame
Ngl.frame(wks)

Ngl.end()
```

6.9 Annotations

The following example shows how to change the appearance of the tickmarks and labels of the axes.



NCL		PyNGL/PyNIO
;-- define x and y variables x = (/ 0.0, 10.0, 20.0, 30.0, 40.0, \ 50.0, 60.0, 70.0, 80.0, 90.0, 100.0/) y = (/ 0.1, 0.7, 1.0, 0.8, 0.75, \ 0.5, 0.6, 0.62, 0.61, 0.59, 0.4/) ;-- x-axis spacing value, values and labels xvalues = ispan(0,100,20) ;-- define x-axis labels xlabels = xvalues+"~S~o~N~E" ;-- add "oE" to x-axis labels ;-- open a workstation wks = gsn_open_wks("png","plot_TRANS_anno_0_ncl") ;-- set resources		import numpy as np import Ngl #-- define x and y variables x = [0.0, 10.0, 20.0, 30.0, 40.0, \ 50.0, 60.0, 70.0, 80.0, 90.0, 100.0] y = [0.1, 0.7, 1.0, 0.8, 0.75, \ 0.5, 0.6, 0.62, 0.61, 0.59, 0.4] xmin = min(x) xmax = max(x) ymin = min(y) ymax = max(y) #-- x-axis spacing value, values and labels xvalues = np.arange(0,120,20) #-- define x-axis labels xlabels = xvalues.astype('str') #-- convert to type string xlabels = [xl + "~S~o~N~E" for xl in xlabels] #-- add 'oE' #-- open a workstation wkres = Ngl.Resources() wks = Ngl.open_wks("png","plot_TRANS_anno_0_py",wkres) #-- set resources

```
res                   =  True
res@gsnMaximize       =  True       ;-- maximize plot size

res@xyLineThicknessF =  4.0        ;-- line width
res@xyLineColor      = "springgreen4" ;-- line color
res@xyDashPattern    =  3          ;-- line patter 3

res@tiXAxisFont      = 22
res@tiXAxisFontHeightF = 0.014
res@tiXAxisString    = "longitudes";-- x-axis title
res@tiYAxisFont      = 22
res@tiYAxisFontHeightF = 0.014
res@tiYAxisString    = "data"      ;-- y-axis title

res@trXMinF          = min(x)-2.   ;-- x-axis min value
res@trXMaxF          = max(x)+2.   ;-- x-axis max value
res@trYMinF          = min(y)-.1   ;-- y-axis min value
res@trYMaxF          = max(y)+.1   ;-- y-axis max value

;-- x-axis tickmark resources
res@tmXBMode         = "Explicit" ;-- use explicit labels
res@tmXBValues       =  xvalues   ;-- values for x-axis
res@tmXBLabels       =  xlabels   ;-- labels for x-axis
res@tmXBLabelFontColor = "blue"   ;-- x-axis label color
res@tmXBLabelFontHeightF = 0.012  ;-- x-axis font size

res@tmXBMajorLineColor  = "blue"  ;-- color major tms
res@tmXBMinorLineColor  = "blue"  ;-- color minor tms
res@tmXBMajorThicknessF =  2.     ;-- thickness major tms
res@tmXBMinorThicknessF =  2.     ;-- thickness minor tms









;-- y-axis tickmark resources
res@tmYLLabelFontColor = "red"    ;-- x-axis label color
res@tmYLLabelFontHeightF = 0.012  ;-- x-axis font size

res@tmYLMajorLineColor   = "red"  ;-- color major tms
res@tmYLMinorLineColor   = "red"  ;-- color minor tms
res@tmYLMajorThicknessF  =  2.    ;-- thickness major tms
res@tmYLMinorThicknessF  =  2.    ;-- thickness minor tms
```

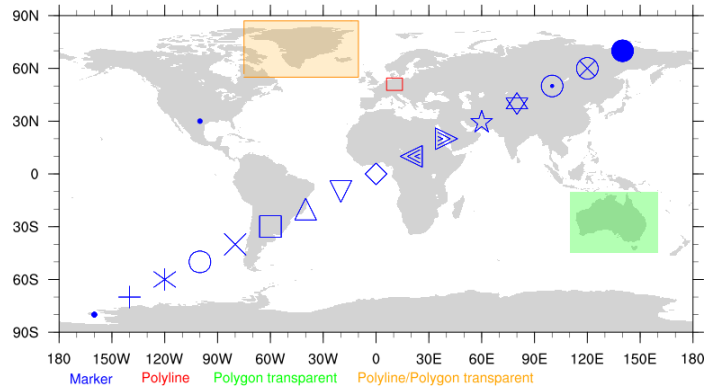
```
res =  Ngl.Resources()


res.xyLineThicknessF =  4.0 #-- line width
res.xyLineColor      = "springgreen4"   #-- line color
res.xyDashPattern    =  3   #-- line patter 3: dash-dot-dash

res.tiXAxisFont      = 22
res.tiXAxisFontHeightF = 0.014
res.tiXAxisString    = "longitudes"#-- x-axis title
res.tiYAxisFont      = 22
res.tiYAxisFontHeightF = 0.014
res.tiYAxisString    = "data"     #-- y-axis title

res.trXMinF          = xmin - 2.  #-- x-axis min value
res.trXMaxF          = xmax + 2.  #-- x-axis max value
res.trYMinF          = ymin - .1  #-- y-axis min value
res.trYMaxF          = ymax + .1  #-- y-axis max value

#-- x-axis tickmark resources
res.tmXBMode           = "Explicit"#-- set x-axis labeling to
explicit
res.tmXBValues         =  xvalues  #-- values for x-axis
res.tmXBLabels         =  xlabels  #-- labels for x-axis
res.tmXBLabelFontColor = "blue" #-- x-axis label color
res.tmXBLabelFontHeightF = 0.012 #-- x-axis font size

res.tmXBMajorLineColor  = "blue" #-- color major tms
res.tmXBMinorLineColor  = "blue" #-- color minor tms
res.tmXBMajorThicknessF =  2.    #-- thickness major tms
res.tmXBMinorThicknessF =  2.    #-- thickness minor tms

#-- draw the tick marks outward the plot
res.tmXBMajorOutwardLengthF = 0.02
res.tmXTMajorOutwardLengthF = 0.02
res.tmXBMinorOutwardLengthF = 0.01
res.tmXTMinorOutwardLengthF = 0.01

#-- y-axis tickmark resources
res.tmYLLabelFontColor   = "red" #-- x-axis label color
res.tmYLLabelFontHeightF = 0.012 #-- x-axis font size

res.tmYLMajorLineColor   = "red"  #-- color major tms
res.tmYLMinorLineColor   = "red"  #-- color minor tms
res.tmYLMajorThicknessF  =  2.    #-- thickness major tms
res.tmYLMinorThicknessF  =  2.    #-- thickness minor tms

#-- draw the tick marks outward the plot
res.tmYLMajorOutwardLengthF = 0.02
```

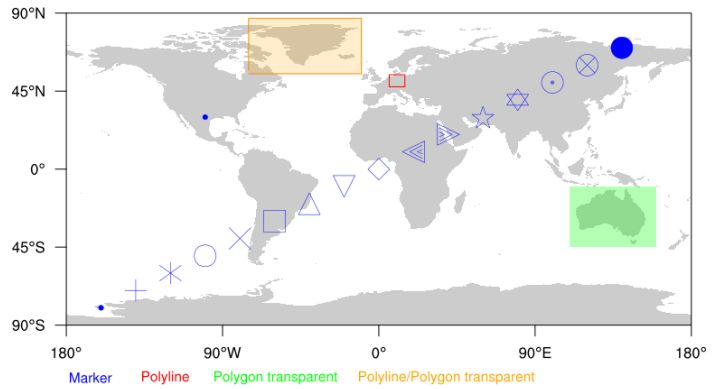
```
;-- draw the plot
plot = gsn_csm_xy(wks,x,y,res)
```

```
res.tmYRMajorOutwardLengthF = 0.02
res.tmYLMinorOutwardLengthF = 0.01
res.tmYRMinorOutwardLengthF = 0.01

#-- draw the plot
plot = Ngl.xy(wks,x,y,res)

#-- done
Ngl.end()
```

6.10 Polylines, Polygons, and Polymarker

The example shows how to use polylines, polygons and polymarker within a plot.





NCL

```
;-- open a workstation and define colormap
wks = gsn_open_wks("png","plot_TRANS_poly_ncl")

;-- set resources
res              =  True
res@gsnDraw      =  False    ;-- don't draw plot yet
res@gsnFrame     =  False    ;-- don't advance frame yet
res@gsnMaximize =  True

res@mpFillOn     =  True




;-- create the map, but don't draw it yet
map = gsn_csm_map(wks,res)

;---------------------------------------------------------
;-- polyline coordinates (Germany)
```

PyNGL/PyNIO

```
import Ngl

#-- open a workstation and define colormap
wks = Ngl.open_wks("png","plot_TRANS_poly_py")

#-- set resources
res                 =  Ngl.Resources()
res.nglDraw         =  False       #-- don't draw plot yet
res.nglFrame        =  False       #-- don't advance frame yet

res.mpFillOn        =  True
res.mpOutlineOn     =  False
res.mpOceanFillColor = "Transparent"
res.mpLandFillColor = "Gray80"
res.mpInlandWaterFillColor = "Gray80"
res.mpGridAndLimbOn =  False       #-- don't draw grid lines

res.pmTickMarkDisplayMode = "Always" #-- turn on map tickmark

#-- create the map, but don't draw it yet
map = Ngl.map(wks,res)

#---------------------------------------------------------
#-- polyline coordinates (Germany)
```

```
;----------------------------------------------------------
x = (/ 6.,  15.,  15.,   6.,   6./)
y = (/47.5, 47.5, 54.5, 54.5, 47.5/)

;-- polyline resources
plres                  =  True
plres@gsLineThicknessF =  2.0    ;-- set line thickness
plres@gsLineColor      = "red"   ;-- set line color

;-- add polyline to map
box_1 = gsn_add_polyline(wks, map, x, y, plres)

;----------------------------------------------------------
;-- polygon coordinates (Australia)
;----------------------------------------------------------
x = (/110., 160., 160., 110., 110./)
y = (/-45., -45., -10., -10., -45./)

;-- polygon resources
pgres                  =  True
pgres@gsFillColor    = "green" ;-- fill color
pgres@gsFillOpacityF =  0.3    ;-- set opacity of polygon

;-- add filled polygon to map
gon_1 = gsn_add_polygon(wks, map, x, y, pgres)

;----------------------------------------------------------
;-- polygon coordinates (Greenland)
;----------------------------------------------------------
x = (/-75., -10., -10., -75., -75./)
y = (/ 55.,  55.,  87.,  87.,  57./)

;-- polygon resources
pres                 =  True
pres@gsFillColor    = "orange"     ;-- fill color
pres@gsFillOpacityF =  0.2         ;-- set opacity

;-- add polygon to map
gon_2 = gsn_add_polygon(wks, map, x, y, pres)

plres@gsLineColor   = "darkorange" ;-- set line color
box_2 = gsn_add_polyline(wks, map, x, y, plres)

;----------------------------------------------------------
;-- polymarker resources
;----------------------------------------------------------
pmres                  =  True
pmres@gsMarkerColor    = "blue" ;-- marker color
pmres@gsMarkerIndex    =  1     ;-- use marker 1
```

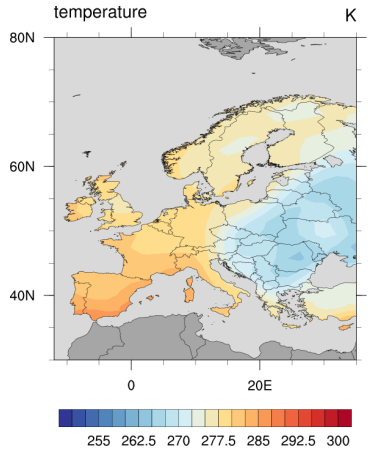
```
#----------------------------------------------------------
x = [ 6.,  15.,  15.,   6.,   6.]
y = [47.5, 47.5, 54.5, 54.5, 47.5]

#-- polyline resources
plres                  =  Ngl.Resources()
plres.gsLineThicknessF =  2.0     #-- set line thickness
plres.gsLineColor      = "red"    #-- set line color

#-- add polyline to map
box_1 = Ngl.add_polyline(wks, map, x, y, plres)

#----------------------------------------------------------
#-- polygon coordinates (Australia)
#----------------------------------------------------------
x = [110., 160., 160., 110., 110.]
y = [-45., -45., -10., -10., -45.]

#-- polygon resources
pgres                  =  Ngl.Resources()
pgres.gsFillColor      = "green"  #-- fill color
pgres.gsFillOpacityF   =  0.3     #-- set opacity of polygon

#-- add filled polygon to map
gon_1 = Ngl.add_polygon(wks, map, x, y, pgres)

#----------------------------------------------------------
#-- polygon coordinates (Greenland)
#----------------------------------------------------------
x = [-75., -10., -10., -75., -75.] #-- define polygon x-array
y = [ 55.,  55.,  87.,  87.,  57.] #-- define polygon y-array

#-- polygon resources
pres                 =  Ngl.Resources()
pres.gsFillColor    = "orange"     #-- fill color
pres.gsFillOpacityF =  0.2         #-- set opacity

#-- add polygon and polyline to map
gon_2 = Ngl.add_polygon(wks, map, x, y, pres)

plres.gsLineColor   = "darkorange" #-- set line color
box_2 = Ngl.add_polyline(wks, map, x, y, plres)

#----------------------------------------------------------
#-- polymarker resources
#----------------------------------------------------------
pmres                  =  Ngl.Resources()
pmres.gsMarkerColor    = "blue"    #-- marker color
pmres.gsMarkerIndex    =  1        #-- use marker 1
```

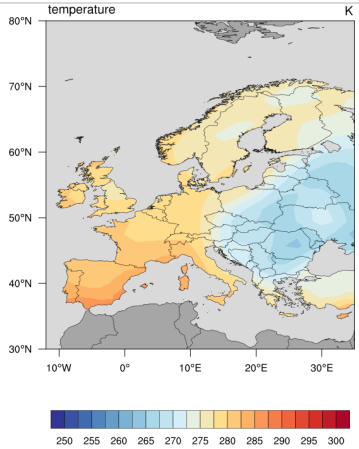
```
pmres@gsMarkerSizeF    =  0.03  ;-- set size of marker
pmres@gsLineThicknessF =  3.    ;-- marker line thickness

;-- unique identifier name for polymarker drawing,
;-- here marker_1
marker_1 = gsn_add_polymarker(wks, map, -100., 30., pmres)

;---------------------------------------------------------
;-- draw all 16 marker on plot using unique identifier
;-- name and additional map attribute settings
;---------------------------------------------------------
x = -160.                 ;-- x-position of first marker
y =  -80.                 ;-- y-position of first marker


do i = 0,15               ;-- 16 different marker
   pmres@gsMarkerIndex  =  i+1
   str = unique_string("poly")   ;-- result is poly0-poly15
;-- add marker to map
   map@$str$ = gsn_add_polymarker(wks, map, x+(i*20.), \
                                  y+(i*10.), pmres)
end do

;---------------------------------------------------------
;-- write strings at the bottom of the plot
;---------------------------------------------------------
txres               =  True
txres@txFontHeightF =  0.014   ;-- default size is HUGE!
txres@txFontColor   = "blue"
txres@txJust    = "CenterLeft"  ;-- puts text on top of bars

dty = 0.23
gsn_text_ndc(wks,"Marker", 0.1, dty, txres)

txres@txFontColor   = "red"
gsn_text_ndc(wks,"Polyline", 0.2, dty, txres)

txres@txFontColor   = "green"
gsn_text_ndc(wks,"Polygon transparent", 0.3, dty, txres)

txres@txFontColor   = "orange"
gsn_text_ndc(wks,"Polyline/Polygon transparent", 0.5, \
             dty, txres)

;-- create the plot and advance the frame
draw(map)
frame(wks)
```

```
pmres.gsMarkerSizeF    =  0.03     #-- set size of marker
pmres.gsLineThicknessF =  8.       #-- marker line thickness

#-- unique identifier name for polymarker drawing,
#-- here marker_1
marker_1 = Ngl.add_polymarker(wks, map, -100., 30., pmres)

#---------------------------------------------------------------
#-- draw all 16 marker on plot using unique identifier name
#-- and additional map attribute settings
#---------------------------------------------------------------
x     = -160.               #-- x-position of first marker
y     =  -80.               #-- y-position of first marker
idstr = "poly"

for i in range(0,16):       #-- 16 different marker
   pmres.gsMarkerIndex = i+1
   id = idstr + str(i)      #-- result is poly0-poly15
#-- add marker to map
   map.id = Ngl.add_polymarker(wks,map,x+(i*20.), \
                               y+(i*10.),pmres)



#---------------------------------------------------------------
#-- write strings at the bottom of the plot
#---------------------------------------------------------------
txres               =  Ngl.Resources()
txres.txFontHeightF =  0.014       #-- default size is HUGE!
txres.txFontColor   = "blue"
txres.txJust        = "CenterLeft" #-- puts text on top of bars

dty = 0.23
Ngl.text_ndc(wks,"Marker", 0.1, dty, txres)

txres.txFontColor   = "red"
Ngl.text_ndc(wks,"Polyline", 0.2, dty, txres)

txres.txFontColor   = "green"
Ngl.text_ndc(wks,"Polygon transparent", 0.3, dty, txres)

txres.txFontColor   = "orange"
Ngl.text_ndc(wks,"Polyline/Polygon transparent", 0.5, \
             dty, txres)

#-- create the plot and advance the frame
Ngl.draw(map)
Ngl.frame(wks)
```

6.11 Masking

The example shows one way to mask the data for selected countries.





NCL

```
;-- countries where contours should appear
;-- too small countries:
;-- "Monaco","San Marino","Vatican City"
mask_specs = (/\"Albania","Andorra","Armenia", \
    "Austria","Azerbaijan","Belarus","Belgium", \
    "Bosnia and Herzegovina","Bulgaria","Croatia", \
    "Cyprus","Czech Republic","Denmark","Estonia", \
    "Finland","France","Georgia","Germany","Greece", \
    "Hungary","Iceland","Ireland","Italy","Kazakhstan", \
    "Latvia","Liechtenstein","Lithuania","Luxembourg", \
    "Macedonia","Malta","Moldova","Montenegro", \
    "Netherlands","Norway","Poland","Portugal","Romania",\
    "Russia","Serbia","Slovakia","Slovenia", \
    "Spain","Sweden","Switzerland","Turkey","Ukraine", \
    "United Kingdom"/)

;-- open file and read variables
f = addfile("rectilinear_grid_3D.nc","r")
var = f->t(0,0,:,:)
lon = f->lon
lat = f->lat
```

PyNGL/PyNIO

```
import Ngl, Nio

#-- countries where contours should appear
#-- too small countries:
#-- "Monaco","San Marino","Vatican City"
mask_specs = ["Albania","Andorra","Armenia", \
      "Austria","Azerbaijan","Belarus","Belgium", \
      "Bosnia and Herzegovina","Bulgaria","Croatia", \
      "Cyprus","Czech Republic","Denmark","Estonia", \
      "Finland","France","Georgia","Germany","Greece", \
      "Hungary","Iceland","Ireland","Italy","Kazakhstan", \
      "Latvia","Liechtenstein","Lithuania","Luxembourg", \
      "Macedonia","Malta","Moldova","Montenegro", \
      "Netherlands","Norway","Poland","Portugal","Romania",\
      "Russia","Serbia","Slovakia","Slovenia", \
      "Spain","Sweden","Switzerland","Turkey","Ukraine", \
      "United Kingdom"]

#-- open file and read variables
f    = Nio.open_file("../read_data/rectilinear_grid_3D.nc", "r")
var  = f.variables["t"][0,0,:,:]
lat  = f.variables["lat"][:]
lon  = f.variables["lon"][:]
```

```
minlat =  30.0        ;-- minimum latitude
maxlat =  80.0        ;-- maximum latitude
minlon = -12.0        ;-- minimum longitude
maxlon =  35.0        ;-- maximum longitude

;-- start the graphics
wks = gsn_open_wks("png","plot_TRANS_masking_ncl")

;-- resource settings
res                 =  True
res@gsnMaximize     =  True

res@cnFillOn        =  True     ;-- contour fill
res@cnFillPalette   = "cmp_b2r" ;-- choose color map
res@cnLinesOn       =  False    ;-- no  contour lines
res@cnLineLabelsOn  =  False    ;-- no contour line labels
res@cnLevelSelectionMode = "ManualLevels" ;-- set levels
res@cnMinLevelValF   =  250.0   ;-- contour min. value
res@cnMaxLevelValF   =  300.0    ;-- contour max. value
res@cnLevelSpacingF  =    2.5    ;-- contour interval
res@cnFillDrawOrder  = "Predraw" ;-- contours first

res@lbBoxMinorExtentF =  0.2   ;-- decrease labelbar height



res@mpDataBaseVersion = "MediumRes" ;-- alias to Ncarg4_1
res@mpDataSetName    = "Earth..4"   ;-- choose map dataset
res@mpMinLatF        =  minlat      ;-- minimum latitude
res@mpMaxLatF        =  maxlat      ;-- maximum latitude
res@mpMinLonF        =  minlon      ;-- minimum longitude
res@mpMaxLonF        =  maxlon      ;-- maximum longitude




res@mpOutlineBoundarySets = "National"
res@mpFillBoundarySets   = "NoBoundaries"
res@mpAreaMaskingOn      =  True
res@mpMaskAreaSpecifiers = mask_specs
res@mpFillAreaSpecifiers = (/"land","water"/)

res@mpOceanFillColor       = "gray85"
res@mpLandFillColor        = "gray65"
res@mpInlandWaterFillColor = "gray65"
```

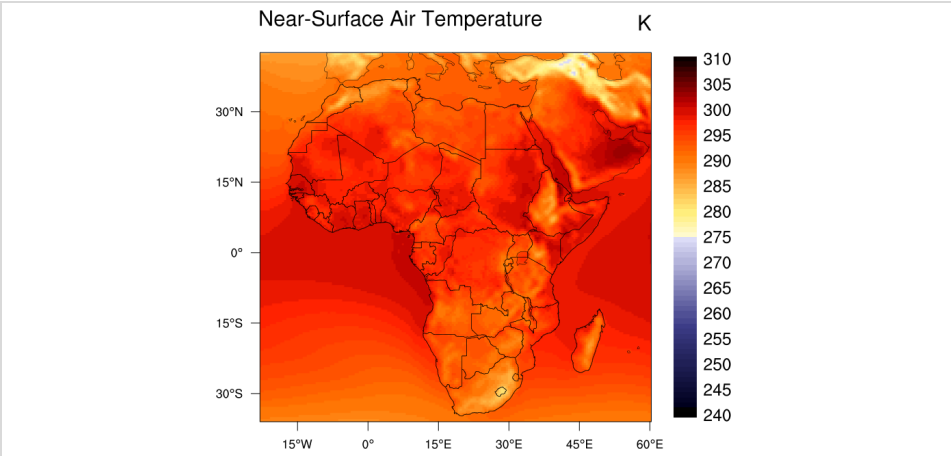
```
minlat =  30.0
maxlat =  80.0
minlon = -12.0
maxlon =  35.0

#-- start the graphics
wks = Ngl.open_wks("png","plot_TRANS_masking_py")

#-- resource settings
res                 =  Ngl.Resources()
res.nglFrame        =  False

res.cnFillOn        =  True        #-- turn on contour level fill
res.cnFillPalette   = "cmp_b2r"    #-- choose color map
res.cnLinesOn       =  False       #-- don't draw contour lines
res.cnLineLabelsOn  =  False     #-- turn off contour line labels
res.cnLevelSelectionMode = "ManualLevels"  #-- set levels
res.cnMinLevelValF  =  250.0       #-- contour min. value
res.cnMaxLevelValF  =  300.0       #-- contour max. value
res.cnLevelSpacingF =    2.5       #-- contour interval
res.cnFillDrawOrder = "Predraw"    #-- contours first

res.lbBoxMinorExtentF  =  0.2      #-- height of labelbar boxes
res.lbOrientation   = "horizontal" #-- horizontal labelbar
res.lbLabelFontHeightF = 0.014

res.mpDataBaseVersion = "MediumRes" #-- alias to Ncarg4_1
res.mpDataSetName     = "Earth..4"
res.mpLimitMode       = "LatLon"
res.mpMinLatF         =  minlat
res.mpMaxLatF         =  maxlat
res.mpMinLonF         =  minlon
res.mpMaxLonF         =  maxlon
res.mpGridAndLimbOn   =  False

res.mpFillOn               = True         #-- turn on map fill
res.mpOutlineBoundarySets = "National"
res.mpFillBoundarySets    = "NoBoundaries"
res.mpAreaMaskingOn       =  True
res.mpMaskAreaSpecifiers  =  mask_specs
res.mpFillAreaSpecifiers  = ["land","water"]
res.mpSpecifiedFillColors = ["gray65","gray85","gray65"]
                                      #-- Land,Ocean,InlandWater

res.sfXArray          =  lon
res.sfYArray          =  lat

#-- viewport resources
```

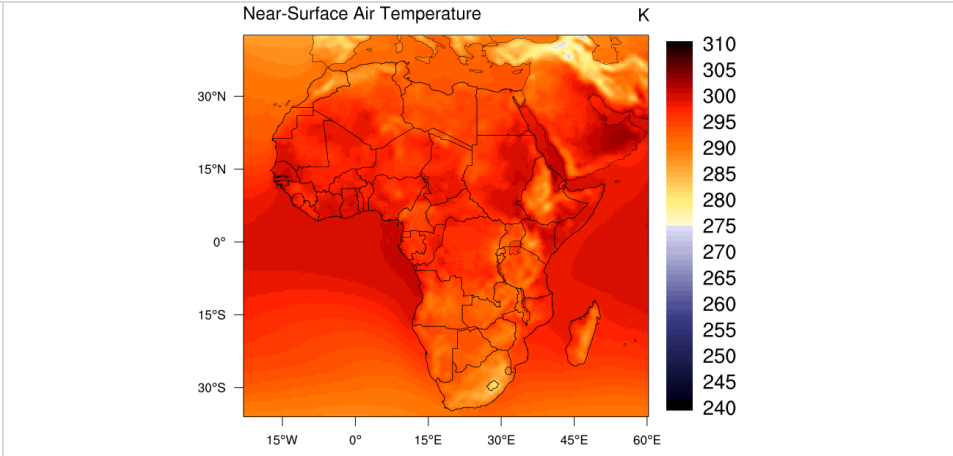
```
;-- create the contour plot
plot = gsn_csm_contour_map(wks,var,res)
```

```
res.nglMaximize       =  False   #-- don't maximize plot
res.vpYF              =  0.9    #-- start y-position of viewport
res.vpWidthF          =  0.65    #-- width of viewport
res.vpHeightF         =  0.65    #-- height of viewport

#-- create the contour plot
plot = Ngl.contour_map(wks,var,res)

#-- write variable long_name and units to the plot
txres               = Ngl.Resources()
txres.txFontHeightF = 0.018

Ngl.text_ndc(wks,f.variables["t"].attributes['long_name'],\
             0.29,0.92,txres)
Ngl.text_ndc(wks,f.variables["t"].attributes['units'], \
             0.82,0.92,txres)

#-- draw the frame
Ngl.frame(wks)
```

6.12 Shapefiles

The example shows how to read and use a shapefile to plot the high resolution borderline of the African countries.





NCL

```
;-- open file and read variables
f = addfile("tas_AFR-44_CNRM-CM5_rcp45_r1i1p1_CCLM_4-8-
17_ym_20060101-20981231.nc", "r")
;-- the input file has no coordinate units degrees_north and
;-- degrees_east this must be set in the script
var             =  f->tas(0,0,:,:)
var&rlat@units = "degrees_north"
var&rlon@units = "degrees_east"

;-- open graphic object
wks =  gsn_open_wks("png","plot_TRANS_shapefile_ncl")

;-- resource settings
res                 =  True
res@gsnDraw         =  False
res@gsnFrame        =  False
res@gsnMaximize     =  True     ;-- maximize plot in frame
res@gsnAddCyclic    =  False    ;-- don't add a cyclic point

res@cnFillOn        =  True     ;-- turn on contour fill
```

PyNGL/PyNIO

```
from __future__ import print_function
import numpy as np
import Ngl, Nio

#--  open file and read variables
f = Nio.open_file("tas_AFR-44_CNRM-CM5_rcp45_r1i1p1_CCLM_4-8-
17_ym_20060101-20981231.nc", "r")


var  = f.variables["tas"][0,0,:,:]
lat  = f.variables["rlat"][:]
lon  = f.variables["rlon"][:]

#-- open graphic object
wks = Ngl.open_wks("png","plot_TRANS_shapefile_py")

#-- resource settings
res                 =  Ngl.Resources()
res.nglFrame        =  False       #-- don't advance frame
res.nglDraw         =  False       #-- don't draw plot



res.cnFillOn        =  True
```

```
res@cnFillPalette   = "NCL_BYR-03" ;-- choose color map
res@cnFillMode      = "RasterFill" ;-- turn on contour fill
res@cnLinesOn       =  False    ;-- turn off contour lines

res@cnLevelSelectionMode = "ManualLevels" ;-- set levels
res@cnMinLevelValF  =  240.0    ;-- minimum contour level
res@cnMaxLevelValF  =  310.0    ;-- maximum contour level
res@cnLevelSpacingF =  0.5      ;-- contour level spacing

res@lbBoxLinesOn    =  False ;-- turn off labelbar box lines
res@lbLabelStride   =  10       ;-- skip every other label
res@lbOrientation   = "Vertical";-- labelbar orientation
                               ;-- is vertical


res@mpLimitMode    = "LatLon"
res@mpMinLatF      = -36.0
res@mpMaxLatF      =  42.6
res@mpMinLonF      = -23.0
res@mpMaxLonF      =  60.3

res@pmTickMarkDisplayMode = "Always"   ;-- turn on tickmarks




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

;-- open shapefile
shpf = addfile("$HOME/data/Shapefiles/act4567/country.shp",\
               "r")
lon = shpf->x
lat = shpf->y

;-- read data off shapefile
segments = shpf->segments
geometry = shpf->geometry
segsDims = dimsizes(segments)
geomDims = dimsizes(geometry)

;-- read global attributes
geom_segIndex = shpf@geom_segIndex
geom_numSegs  = shpf@geom_numSegs
segs_xyzIndex = shpf@segs_xyzIndex
segs_numPnts  = shpf@segs_numPnts
numFeatures   = geomDims(0)

;-- add polylines to map
```

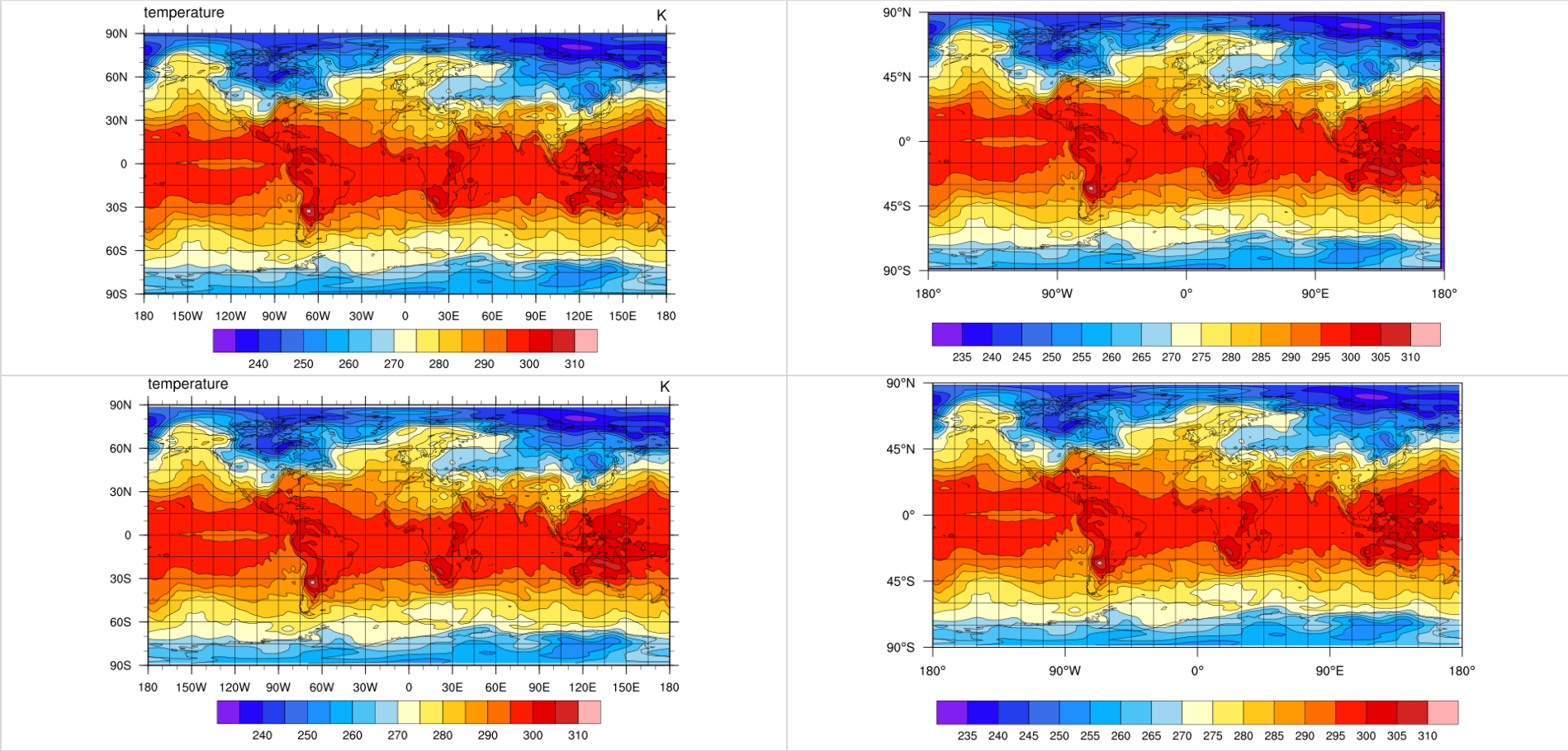
```
res.cnFillPalette   = "NCL_BYR-03" #-- choose color map
res.cnFillMode      = "RasterFill" #-- turn on contour fill
res.cnLinesOn       =  False
res.cnLineLabelsOn  =  False
res.cnLevelSelectionMode = "ManualLevels"#-- set levels
res.cnMinLevelValF  =  240.0       #-- minimum contour level
res.cnMaxLevelValF  =  310.0       #-- maximum contour level
res.cnLevelSpacingF =  0.5         #-- contour level spacing

res.lbBoxLinesOn    =  False     #-- turn off labelbar box lines
res.lbLabelStride   =  10        #-- skip every other label
res.lbBoxMinorExtentF =  0.24    #-- decrease height of labelbar
                                 #-- boxes
res.pmLabelBarOrthogonalPosF = -0.05 #-- move labelbar upward

res.mpLimitMode     = "LatLon"
res.mpMinLatF       = -36.0
res.mpMaxLatF       =  42.6
res.mpMinLonF       = -23.0
res.mpMaxLonF       =  60.3
res.mpGridAndLimbOn =  False         #-- don't draw grid lines


res.sfXArray        =  lon
res.sfYArray        =  lat

#-- create the contour plot
plot = Ngl.contour_map(wks,var,res)

#-- open shapefile
shpf     = Nio.open_file("act4567/country.shp", "r")
lon      = np.ravel(shpf.variables["x"][:])
lat      = np.ravel(shpf.variables["y"][:])


#-- read data off shapefile
segments = shpf.variables["segments"][:,0]
```

```
lines = new(segsDims(0),graphic) ;-- array to hold polylines

;-- polyline resource settings
plres              = True         ;-- resources for polylines
plres@gsLineColor = "black"


;-- add shapefile polylines to the plot
segNum = 0       ; Counter for adding polylines
do i=0, numFeatures-1
   startSegment = geometry(i, geom_segIndex)
   numSegments  = geometry(i, geom_numSegs)
   do seg=startSegment, startSegment+numSegments-1
      startPT = segments(seg, segs_xyzIndex)
      endPT   = startPT + segments(seg, segs_numPnts) - 1
      lines(segNum) = gsn_add_polyline(wks, plot,\
                      lon(startPT:endPT),\
                      lat(startPT:endPT), plres)
      segNum = segNum + 1
   end do
end do









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

```
#-- polyline resource settings
plres              =  Ngl.Resources() #-- resources for polylines
plres.gsLineColor = "black"
plres.gsSegments  =  segments

#-- add shapefile polylines to the plot
id = Ngl.add_polyline(wks, plot, lon, lat, plres)









#-- write variable long_name and units to the plot
txres               = Ngl.Resources()
txres.txFontHeightF = 0.022

Ngl.text_ndc(wks,f.variables["tas"].attributes['long_name'],0.30
,0.88,txres)
Ngl.text_ndc(wks,f.variables["tas"].attributes['units'],
0.78,0.88,txres)



#-- draw the plot and advance the frame
Ngl.draw(plot)
Ngl.frame(wks)

Ngl.end()
```

6.13 Regridding

The next example shows how to regrid a data set from a coarse grid to a higher resolution grid.



NCL		PyNGL/PyNIO
		`from __future__ import print_function` `import numpy as np` `import xarray as xr` `import xesmf as xe` `import Ngl`
`;-- open file and read variables`		`#--  open file and read variables`

```
data = addfile("../read_data/rectilinear_grid_3D.nc","r")

t    = data->t(0,0,:,:)
lat  = data->lat
lon  = data->lon

;-- destination lat/lon
dst_lat = fspan(-89.5,89.5,180)
dst_lon = fspan(-179.5,179.5,360)


;-- set resources
Opt                =  True
Opt@InterpMethod   = "bilinear"                          ;--
interpolation method
Opt@ForceOverwrite =  True                               ;--
force overwrite
Opt@SrcMask2D      =  where(.not.ismissing(t),1,0)       ;--
what to mask
Opt@DstGridType    = "rectilinear"                       ;--
Destination grid
Opt@DstGridLon     =  dst_lon
Opt@DstGridLat     =  dst_lat

;-- call ESMF_regrid
t_regrid = ESMF_regrid(t,Opt)




;-- plotting
wks = gsn_open_wks("png","plot_TRANS_regrid_ESMF_ncl")

res                 =  True
res@gsnFrame        =  False
res@cnFillOn        =  True
res@cnFillPalette   = "NCL_default"
res@cnLineLabelsOn  =  False
res@cnLevelSelectionMode = "ManualLevels"
res@cnMinLevelValF  =  235.0
res@cnMaxLevelValF  =  310.0
res@cnLevelSpacingF =    5.0
res@lbOrientation   = "horizontal"




plot = gsn_csm_contour_map(wks,t_regrid,res)
```

```
data = xr.open_dataset("../read_data/rectilinear_grid_3D.nc")

t    = data['t'][0,0,:,:]
lat  = np.array(data['lat'][:])
lon  = np.array(data['lon'][:])

#-- destination lat/lon
dst_lat = np.arange(-89.5,90.5,1.0)
dst_lon = np.arange(-179.4,180.5,1.0)


#-- init destination grid
dstgrid = xr.Dataset({'lat': (['lat'], dst_lat),
                      'lon': (['lon'], dst_lon),})









#-- regrid
regridder = xe.Regridder(data, dstgrid, 'bilinear')

t_regrid   = regridder(t)
lat_regrid = regridder(t).lat
lon_regrid = regridder(t).lon

#-- plotting
wks = Ngl.open_wks("png","plot_TRANS_regrid_xesmf_py")

res                 =  Ngl.Resources()
res.nglFrame        =  False
res.cnFillOn        =  True
res.cnFillPalette   = "NCL_default"
res.cnLineLabelsOn  =  False
res.cnLevelSelectionMode = "ManualLevels"
res.cnMinLevelValF  =  235.0
res.cnMaxLevelValF  =  310.
res.cnLevelSpacingF =    5.
res.lbOrientation   = "horizontal"

res.sfXArray        =  dst_lon
res.sfYArray        =  dst_lat

plot = Ngl.contour_map(wks,t_regrid,res)
```

```
frame(wks)



plot = gsn_csm_contour_map(wks,t,res)

frame(wks)

;-- clean-up
system("rm -rf source_grid_file.nc destination_grid_file.nc")
system("rm -rf weights_file.nc PET0.RegridWeightGen.Log")
```

```
Ngl.frame(wks)

res.sfXArray        =  lon
res.sfYArray        =  lat

plot = Ngl.contour_map(wks,t,res)

Ngl.frame(wks)

#-- clean-up
regridder.clean_weight_file()
```