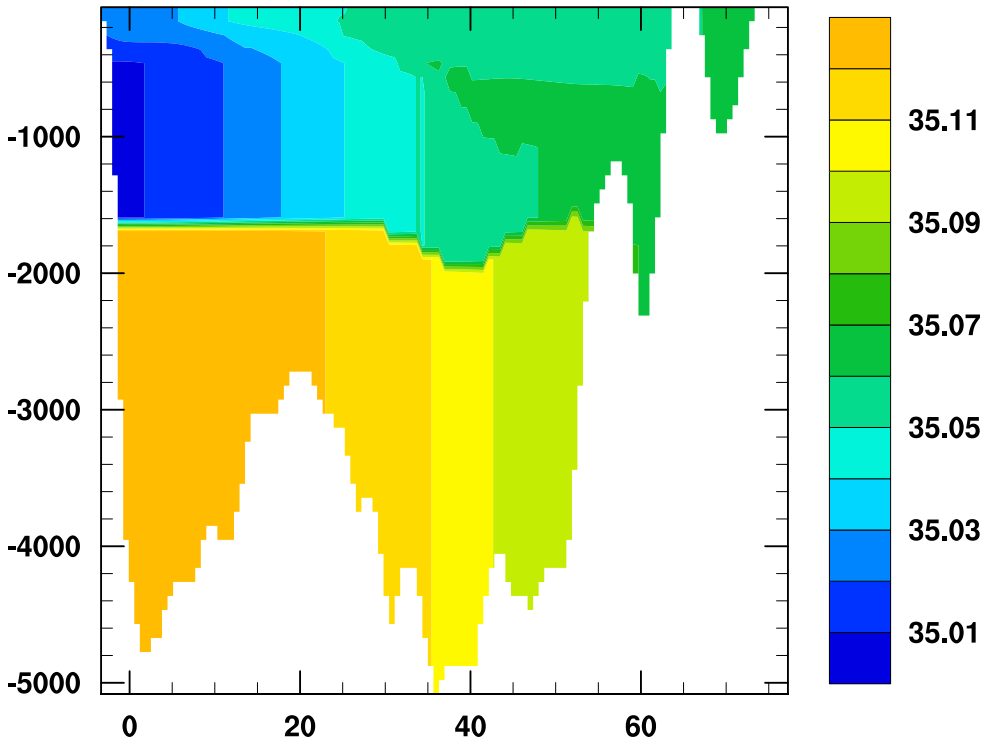
Norwegian Meteorological Institute
met.no

met.no note

Note No. 8/2005
Oceanography
Oslo, 27th April 2005

The ncl-metno shell script collection[1]

Arne Melsom



[1]If viewed with Acrobat Reader®, the pdf file with this document contains hyperlinks that are active when the an Internet connection is open.

Norwegian Meteorological Institute
met.no

note

Number	Subject	Date	Classification	ISSN
8/2005	Oceanography	27th April 2005	☒ Open ☐ Restricted ☐ Confidential	-

Title

The ncl-metno shell script collection

Authors

Arne Melsom

Client(s)	Client reference
Norwegian Science Council	146476/120, 155972/720

Abstract

This note documents a set of shell scripts that make it easy to create a variety of depictions with filled contours and/or vectors. This software only works for fields that have been stored on netCDF files. Each of these shell scripts produce a script for the NCAR Command Language (NCL), and NCL is executed with the NCL script as input. NCL then produces figures, predominantly as a set of png and eps files. The depiction is also displayed in a terminal window. The software that is documented in this note has been developed for UNIX platforms, and the scripts are written in the "Bourne again shell" (bash).

Keywords

NCL, netCDF, visualization, wrapper

Disciplinary signature	Responsible signature
Øystein Hov, Head R&D Department	Eivind A. Martinsen, Head, Section Oceanography

Postal address PO Box 43 Blindern, N-0313 Oslo, Norway
Office Niels Henrik Abels vei 40
Telephone +47 2296 3000
Telefax +47 2296 3050
e-mail: met.inst@met.no
Web: met.no
Bank account 7695 05 00601
Swift code DNBANOKK

Contents

1. Introduction

The main purpose of the software that is documented in this note, is to provide an easy-to-use, command line based working environment for visualization of results that are stored on netCDF formatted files. In order to achieve this, a set of shell scripts that utilizes the NCAR Command Language (NCL) has been developed. Thus, the software that is documented here is merely a tool for swift production of graphics, and the tool (wrapper scripts) is entirely dependent on NCL. NCL, which is available for free in binary, is a product of the Scientific Computing Division at the National Center for Atmospheric Research (NCAR). In the present context, it should be stressed that NCL is not only a visualization tool, but may also favorably be used for data processing purposes. Documentation, NCL sample scripts, and much more, are available from NCL's web site at `http://www.ncl.ucar.edu/`.

A second purpose of ncl-metno is to provide the user with an NCL script that can subsequently be edited by a user with knowledge of NCL, in order to modify the depiction to his or her needs. A third purpose is to aid a potential NCL user in getting started using the NCL software, since inspection of NCL scripts that are made by the ncl-metno package may be useful as a starting point for learning NCL. In the latter respect, it must be mentioned that excellent information for new users are available from the NCL web site, in particular, Getting Started Using NCL is useful.

Some of the terms that are used later in this document, are explained here. When referring to the sequence of array dimensions, the term "Fortran style" is used frequently. This term means that the order of variation in array indices run from left to right, and starts with index 1. Thus, in "Fortran style" a two-dimensional array *var* is stored as
var(1,1) var(2,1) ... var(m,1) var(1,2) var(2,2) ... var(m,2) ... var(m,n)
However, the order of variation is reversed in NCL scripts, and the index starts from 0. Thus, translating "Fortran style" above to the format used in NCL scripts, we have
var(0,0) var(0,1) ... var(0,m-1) var(1,0) var(1,1) ... var(1,m-1) ... var(n-1,m-1)

Further, it is customary to store 3-dimensional and 4-dimensional fields in the order $x-y-z$, $x-y-t$, and $x-y-z-t$ (in "Fortran style"), so crossections with a constant z (and t) are horizontal slices. The vertical coordinate is sometimes a layer number rather than a vertical z-level number. Fields that are stored on a geographical grid on netCDF files typically use the horizontal dimensions *lon* and *lat* in places of x and y. For *ncl-metno*, the dimensions are only relevant if you wish to include a map in the depictions. Then, horizontal 1-dimensional variables *lon* and *lat* must exist on the same netCDF file as the one that contains the field you wish to visualize.

This note is organized as follows: First, installation instructions are provided in section 2. Then, the various types of shell scripts are described in sections 3-6. Next, the user is guided through some easy steps for modifying the depictions in section 7, and some frequently occurring error messages are explained in section 8. Finally, each of the *ncl-metno* shell scripts are documented in appendices A-N. This documentation is in part based on the hypertext documentation of *ncl-metno,* and in part based on help texts that are available for the various shell script in this software.

DISCLAIMER: This software has been written by a programmer who's not really a programmer, but a scientist. Mistakes may well have been made, so USE THIS SOFTWARE AT YOUR OWN RISK! . . . I should add that using earlier versions of *ncl-metno* for more than a year, I have never experienced any serious problems.

WARNING: When these shell scripts are executed, some files will be generated in the working directory. This is typically a NCL script file named with a leading "dot" (e.g. *.contour.ncl*), and a set of *png* and *eps* picture files. The m*.sh scripts produce sets of *pnm* files.

2. Installation

Before you can install and run the present software, you must download and install NCL on your system (unless it exists already). Next, you must download the *ncl-metno* software, which is available from `http://ensemble.met.no/ncl-metno/`. The tar-ball that you can download from this site also includes documentation of the shell scripts (in hypertext, and this document). In order to install *ncl-metno*, you must first set some paths in the INSTALL file: *$ncldir* is the directory where the *ncl-metno* shell scripts will be stored, and *$linkdir* is the directory where binaries reside (typically, this is the directory included in your $PATH). You may optionally set *$htdir* to a path where you wish to store the hypertext documentation of *ncl-metno*. Then, you should complete the installation by typing *./INSTALL* on the command line prompt. This rather liberal implementation allows a user to install this software locally if he or she is working in a network and does not have super user privileges (provided that such an installation is in accord with the user's institute's/company's policy!)

Note that for this document, *$ncldir* was set to */usr/local/selfmade/ncl* so paths will generaly be different in your implementation than they appear here.

3. Filled contours

The *ncl-metno* software comes with nine scripts that produce depictions with filled contours:

- **contour.sh**
  makes a depiction of a 2-dimensional field, or of a "horizontal" slice from a 3-dimensional or 4-dimensional field
- **Dcontour.sh**
  makes a depiction of differences between two fields that must have the same dimensions; this may e.g. be useful for inspection of trends
- **Scontour.sh** makes a depiction of the sum of two fields that have the same dimensions
- **mcontour.sh**
  produces multiple pnm files that may later be turned into an animation (see **makemovie.sh** in section 6 and its documentation in appendix N for details); otherwise, this is the same as **contour.sh**

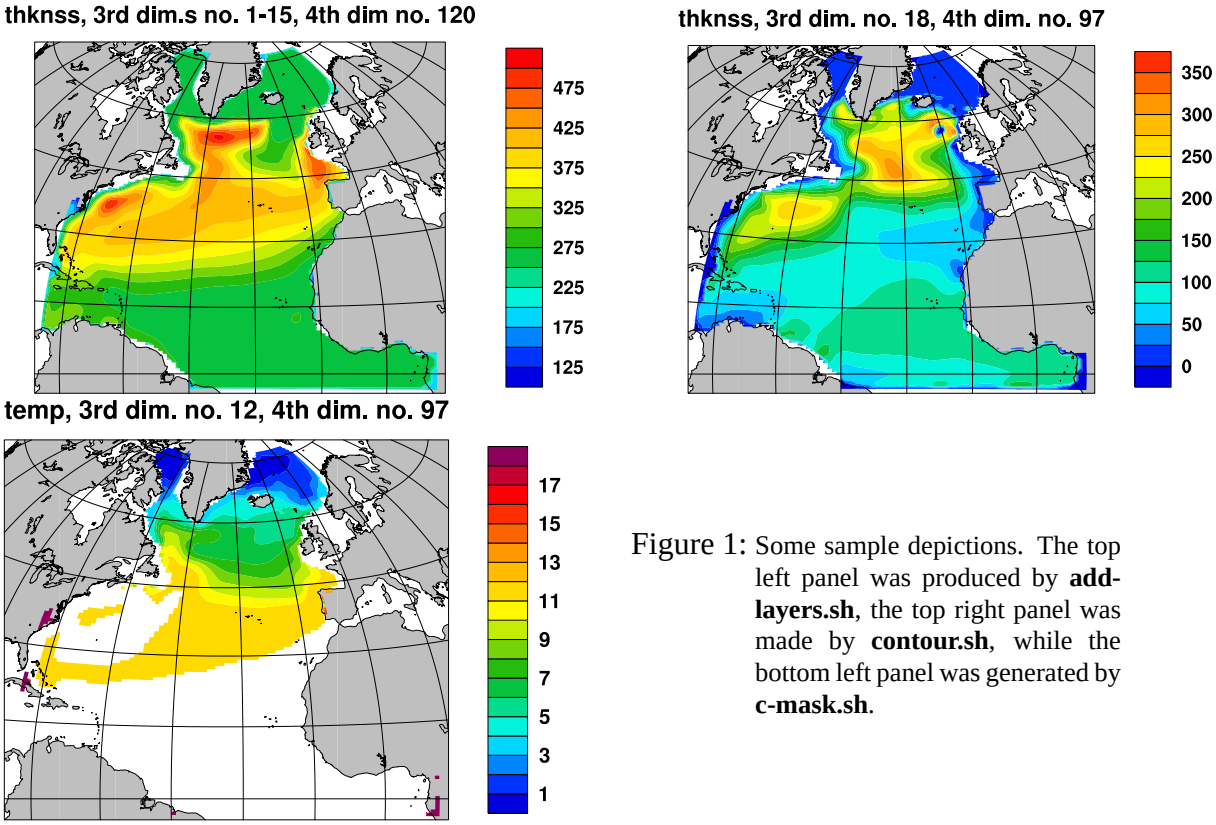


Figure 1: Some sample depictions. The top left panel was produced by **addlayers.sh**, the top right panel was made by **contour.sh**, while the bottom left panel was generated by **c-mask.sh**.

- **c-mask.sh**
  makes a depiction of one field, but with a masking that is set based on an interval from another field; otherwise, this is the same as **contour.sh**

- **section.sh**
  makes a depiction of a crossection from a 3-dimensional or 4-dimensional field, along constant nodes in the dimensions that are not displayed; this may e.g. be useful for making Hovmøller plots

- **layersection.sh**
  makes a depiction of a x-z or y-z crossection when the third dimension is layer no.; layer thickness values must also be available

- **mlayersection.sh**
  produces multiple pnm files that may later be turned into an animation (see **makemovie.sh** in section 6 and its documentation in appendix N for details); otherwise, this is the same as **layersection.sh**

- **addlayers.sh**
  adds values in adjacent levels/layers from 3-dimensional or 4-dimensional fields; this may e.g. be useful for depicting the level of isopycnals when the vertical coordinate is density

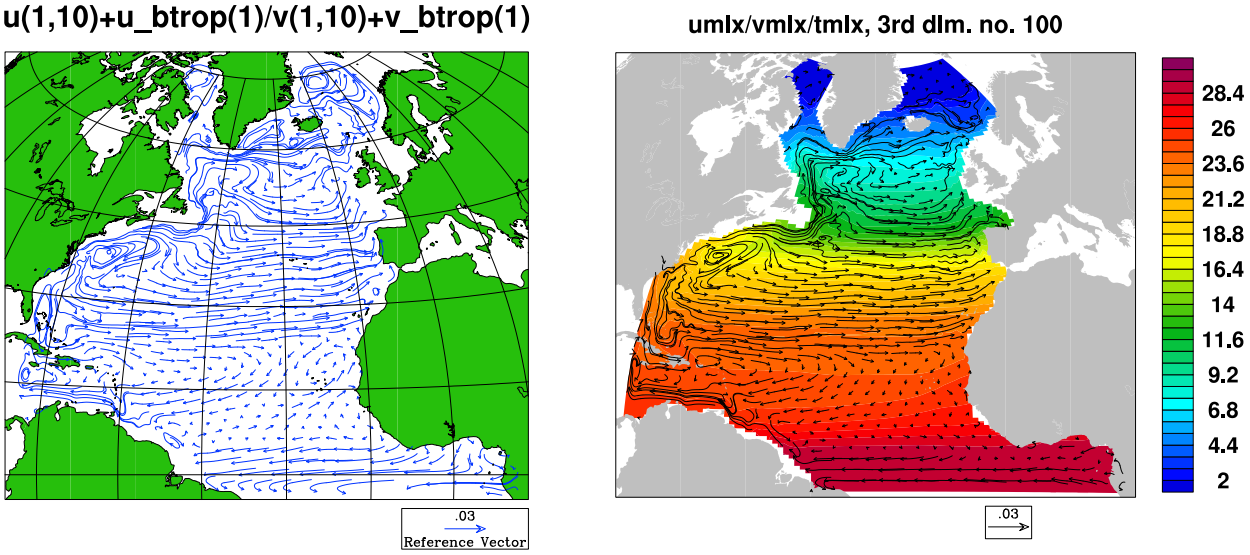


Figure 2: Some sample depictions. The left panel was produced by **Svector.sh**, while the right panel was generated by **c-on-v.sh**.

Documentation of these scripts is provided as appendices A-H.

4. Vectors

The *ncl-metno* software comes with three scripts that produce vector depictions:

- **vector.sh**
  depicts a vectors based on u- and v-components from 2-dimensional fields, or from a "horizontal" slice from 3-dimensional or 4-dimensional fields
- **Svector.sh**
  adds two u-component fields and two v-component fields; otherwise this is the same as **vector.sh**
- **mvector.sh**
  produces multiple pnm files that may later be turned into an animation (see **makemovie.sh** in section 6 and its documentation in appendix N for details); otherwise, this is the same as **vector.sh**

Documentation of these scripts is provided as appendices I-K.

5. Filled contours and vectors

The *ncl-metno* software comes with two scripts that produce vector depictions on top of filled contours:

- **transport.sh**
  computes transports as the product of a scalar field and a vector fields, and depicts the result as vectors on top of transport (flux) values
- **v-on-c.sh**
  depicts a vector field on top of a filled contours for a scalar field

Documentation of these scripts is provided in appendices L-M.

6. Miscellaneous scripts

There are two remaining scripts in the *ncl-metno* software:

- **cropone.sh**
  crops a pnm image
- **makemovie.sh**
  converts a set of pnm files to an animation, in the *mpeg* format or in the *fli* format

Documentation of **makemovie.sh** is provided in appendix N.

7. Modifying the ncl scripts

A number of plot specification are collected in the file *userdef.ncl*, which was written to the directory *$ncl-metno* that was specified during installation, see section-2 for details. When one of the shell scripts is run, it first looks for a *userdef.ncl* in the working directory. If this file can't be found in the working directory, the script resorts to *$ncl-metno/userdef.ncl* for the default specifications. In the documentation of the individual scripts in appendices A-N, *$ncl-metno* was set to */usr/local/selfmade/ncl*.

Hence, in order to change the specifications in *$ncl-metno/userdef.ncl*, the user must copy this file to the working directory, then edit the local *userdef.ncl* file, and finally run the *ncl-metno* shell script. For editing purposes, note that a semi-colon (';') marks the start of a comment in NCL. Various aspects of the depiction that may be modified are described in the subsections below. Note that each of these aspects in general only has an affect on results from a subset of the *ncl-metno* shell scripts (e.g., "Vector specifications" only affects visualizations that include vectors).

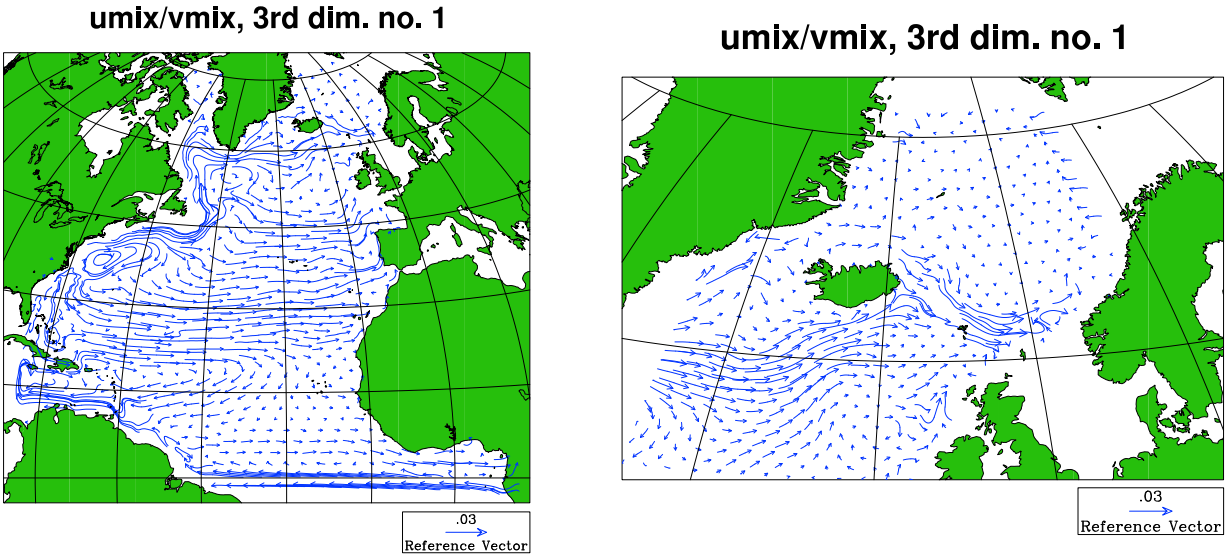


Figure 3: The right panel displays results after zooming in on a north eastern sub-domain in the left panel. See the text for details.

7.1. Title

The plot's title is given by *mytitle*. By default, this is set to "auto", then, the shell script will produce a title. All but one of the figures in this document have been made with this setting. Reset *mytitle* to whatever you like, if you don't want a title to appear (as on the cover of this document), leave the string empty. But don't delete (or comment) the *mytitle* line!

7.2. Zooming

From time to time, the user may wish to inspect detailed results in a subregion. This can be done by resetting *x1*, *y1* to the subregion's lower left corner, and *x2*, *y2* to its upper right corner. These values should be set in "Fortran style". These may be set as absolute numbers, or fractions of 'nx' and 'ny' (e.g. *y1*=12, *y2*=3*ny/5). By default, the entire domain is depicted.

In Figure 3, results are displayed before and after three lines in the *userdef.ncl* file was edited:

```
x1= nx-50  ; Leftmost  grid point to depict, for dimension  x  or  lon
x2= nx     ; Rightmost grid point to depict
y1= ny-40  ; Lowermost grid point to depict, for dimension  y  or  lat
y2= ny     ; Uppermost grid point to depict
```

7.3. Plot size

The maximum paper size of the plot is set as *maxsize*. For depictions without a map, the shell scripts will set the height and width of the plot so that one grid cell has the same size in both directions. Further, *xp* and *yp* is the x- and y-coordinate of the upper left corner of the plot.

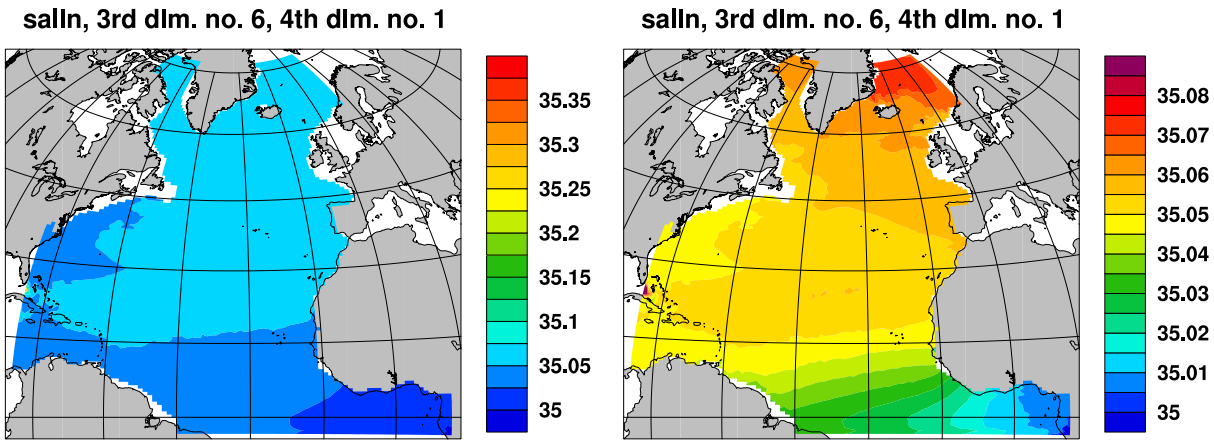


Figure 4: The right panel displays results after the color coding was reset. See the text for details.

7.4. Contouring specifications

The lower and upper isopleths are given by *v1* and *v2*, respectively. The number of isopleths is given by *nv*. Note that the depictions that are produced by *ncl-metno* fills regions between the isopleths with colors, the actual isopleths are not drawn. So, if *v1*=1., *v2*=10., and *nv*=10, a total of 11 colors will be used, for the intervals $\langle -\infty,1\rangle$, $\langle 1,2\rangle \ldots \langle 9,10\rangle$, $\langle 10,\infty\rangle$. By default, NCL will automatically select a modest no. of isopleth values. As long as *nv*=0 or 1, NCL will resort to its default method.

The range that is automatically generated by NCL when *nv*=0 or 1 is not always the best. It may be a good idea to manually set the color specification in*userdef.ncl*, particularly when there are outliers. An example of this is displayed in Figure 4. Here, results are displayed before and after three lines in the *userdef.ncl* file was edited:

```
v1= 35.00 ; Low  value for isopleths, disregarded when  nv  is 0 or 1
v2= 35.085; High value for isopleths, disregarded when  nv  is 0 or 1
nv= 17    ; No. of isopleths, there will be  nv+1  colors
```

7.5. Vector specifications

The physical size of a reference vector is given by *vsz*, corresponding to a speed *vsp*. Increase *vsp* (or decrease *vsz* to shorten the length of vectors. (Their preset values are for ocean currents, so vectors will be unpleasantly long for e.g. wind speeds.) Further, *vd* is a measure of the distance between neighboring vectors. Increase its value to decrease the no. of displayed vectors. Finally, when *curly_on* is set to 1 (its default value), pieces of streamlines are displayed rather than standard vectors. Set *curly_on*=0 for standard vectors.

In Figure 5, the size and density of the vectors were changed by resetting *vsp* and *vd*, while *vsz* (and *curly_on*) were left unchanged:

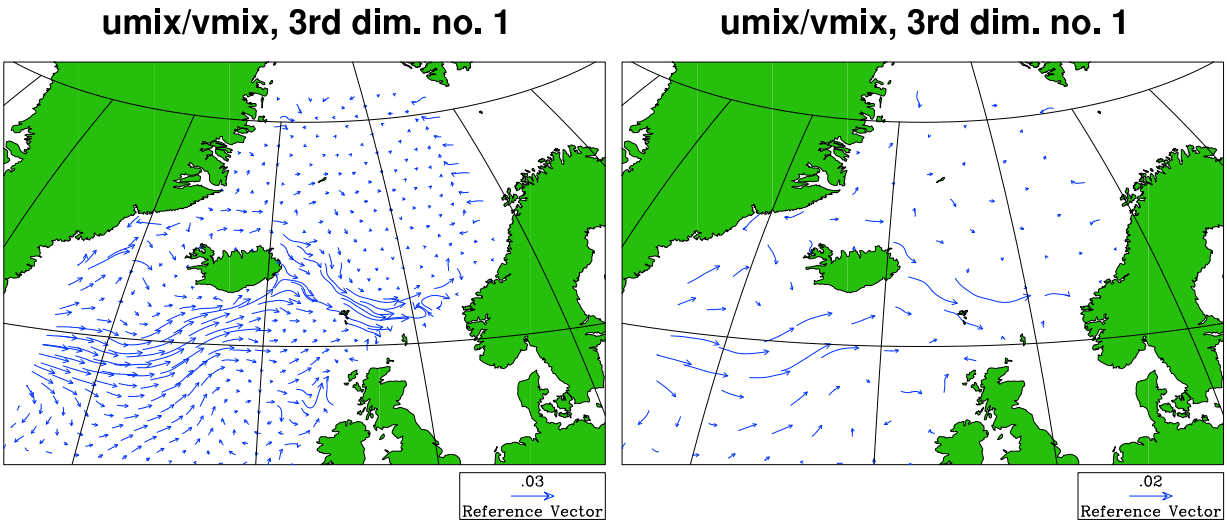


Figure 5: The right panel displays results after the size and density of the vectors were modified. See the text for details.

```
vsz= 0.05   ; Size (length) of reference vector
vsp= 0.02   ; Speed         of reference vector
vd = 0.04   ; Distance between vectors
curly_on= 1 ; =1: Use curly vectors, otherwise, use standard vectors
```

The default values of *vsp* and *vd* are 0.03 and 0.015, respectively.

7.6. Color map (palette)

NCL comes with a range of predefined color maps, and the user may also define his or her own color maps. A selection of the predefined palettes are listed in *userdef.ncl*. The preselected color map is a "rainbow style" palette with 18 different colors. The user may change this setting by un-commenting the appropriate line in *userdef.ncl*.

Now, reconsider the results that are displayed in Figure 4, and the contouring specification used in the right panel. These results are redisplayed in Figure 6, using two of the alternative color maps. The left panel was produced after the relevant section in *userdef.ncl* was rewritten to

```
;mapname="LR BkBlAqGrYeOrReViWh200"; rainbow style,  nv <= 17
;mapname="HR BkBlAqGrYeOrReViWh200"; rainbow style,  nv <= 35
mapname="LR BlWhRe"                 ; blue/white/red, nv <= 17
;mapname="HR BlWhRe"                ; blue/white/red, nv <= 35
;mapname="nrl_sirkes"               ; NRL,            nv <= 17
;mapname="gsdtol"                   ; grayscale,      nv <= 17
;mapname="default"                  ; tigerstripes,   nv <= 17
```

i.e., the top line in the default setting on *userdef.ncl* was commented by introducing a leading semi-colon, while the third line was un-commented.

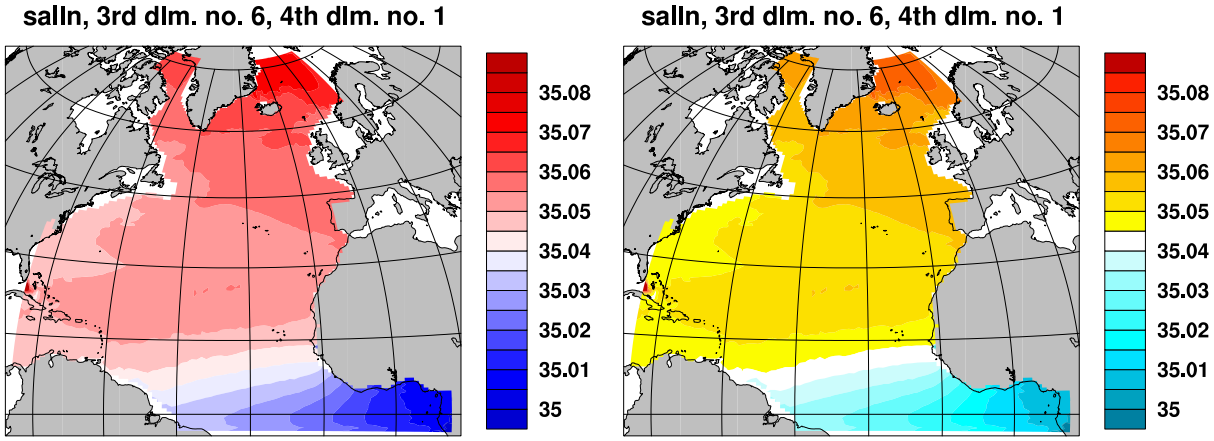


Figure 6: Results in the left and right panels are displayed using the color maps "blue/white/red" (BlWhRe) and "NRL" (nrl_sirkes), respectively. See the text for details.

7.7. Map projection

Most of the *ncl-metno* shell scripts support the use of maps in NCL. When a map is requested, the user must specify the map projection to be used. A list of map projections is provided in *userdef.ncl*, which by default will plot the selected field on a "LambertEqualArea" projection. The user may choose one of the alternative projections by un-commenting the appropriate line in *userdef.ncl*.

7.8. Coastline details

NCL presently handles three different levels of map details, namely "Ncarg4_0", "Ncarg4_1" and "RANGS_GSHHS" for coarse, intermediate and high resolutions, respectively. The use of "RANGS_GSHHS" may require a separate installation, see the NCL installation instructions. The "Ncarg4_1" medium resolution map is used as the default in *userdef.ncl*. Alternative maps may be chosen by un-commenting the appropriate line in *userdef.ncl*.

8. Error messages

Some errors occur more frequently than others. Here are a couple.

First, lets try to plot a horizontal slice of the variable *salt* from the netCDF file *expt_004_S.nc*:
mycomputer:∼% contour.sh 4d expt_004_S.nc salt 1 1

```
fatal:Either file (ncfile) isn't defined or variable (salt)
 is not a variable in the file
fatal:Execute: Error occurred at or near line 12
```

This error message is easy to understand. In this particular case, the netCDF file exist, but there are no variable *salt* on the file.

Now, let's request a depiction for the variable *salin* on the same netCDF file:
mycomputer:∼% contour.sh 3d expt_004_S.nc salin 1

```
fatal:Number of subscripts do not match number of dimensions of variable,
 (3) subscripts used, (4) subscripts expected
fatal:Execute: Error occurred at or near line 11
```

In the case above, the user requested a depiction of a 3-dimensional variable, but the variable *salin* was stored on the netCDF file as a 4-dimensional variable. If the user wishes to inspect the NCL script, he or she may set a flag for retaining the NCL script:
mycomputer:∼% contour.sh 3d CONMAN_004_S.nc salin -1
Obviously, the same error occurs, but the NCL script is retained:

```
Saving ncl script file as ~/.contour.ncl
```

We inspect the NCL script:
mycomputer:∼% head -12 .contour.ncl | tail -3
and find that the error is actually in line 12 (or on line‘11, if the top line is "NCL style" line 0):

```
d30 = 1-1

ncvar0 = ncfile->salin(d30,:,:)
```

A depiction for the first level and the first output time for this 4-dimensional variable is obtained by issuing
mycomputer:∼% contour.sh 4d CONMAN_004_S.nc salin 1 1
Finally, let's try to plot the salinity field on a map:
mycomputer:∼% contour.sh 4dmap CONMAN_004_S.nc salin 1 1
Now we got the error message:

```
fatal:Number of subscripts do not match number of dimensions of variable,
 (1) subscripts used, (2) subscripts expected
fatal:Execute: Error occurred at or near line 274
```

This one is a bit harder to grasp. The problem here is with the dimensional variables *lon* and *lat*. These variables existed on the netCDF file *CONMAN_004_S.nc* (as they must for the *4dmap* option to work), but as 2-dimensional variables. The *salin* field was not stored on a geographical grid, and *contour.sh* is unable to plot the field on a map.

Acknowledgement. This software is entirely dependent on the NCAR Command Language (NCL). I am indebted to the developers of NCL. This software has been written in connection with two projects that were funded by the Norwegian Research Council, under contracts no. 146476/120 and 155972/720, respectively.

A. contour.sh, syntax

```
 contour.sh <option> <file> <variable> [<d3node> (<d4node>)]
where
   <option>    specifies dimensions and geo- or nongeo-grid
                implemented:
                 2d    - 2D fields
                 3d    - 3D fields
                 4d    - 4D fields
                 2dmap - 2D fields, dims. are lon & lat
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...2/3/4dmap will be displayed on a lon-lat grid
                  with a map
   <file>      name of the netcdf file
   <variable>  name of requested variable on the netcdf file
                (case sensitive)
   <d3node>  node no. of third dimension
               if <option> is one of 2d, 2dmap and a fourth
                 argument is present, or if <d3node> is negative,
                 this will be interpreted as a flag that will cause
                 the ncl script to remain (see examples below)
   <d4node>  node no. of fourth dimension

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'contour.sh' is given,
 the user may specify
  * title
  * zooming
  * map projection (lon-lat grids only)
  * color map (palette)
  * no. of colors
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).

Examples:
=========
 contour.sh 4dmap hydrography.nc temp 1 10
   will produce a depiction on a lon-lat grid w/ a map,
    for the first node in the third dimension (usually
    the top vertical level) and the tenth node in the
    fourth dimension (usually time step no. 10),
    of the variable 'temp' on the file 'hydrography.nc'
 contour.sh 3d surface.nc sst -1
   will produce a depiction on a x-y grid of the first node
    in the third dimension, of the variable 'sst' on the
    file 'surface.nc'; and the ncl-script will be retained
 contour.sh 2dmap topography.nc Depth a
   will produce a depiction on a lon-lat grid w/ a map,
   of the variable 'Depth' on the file 'topography.nc';
    and the ncl-script will be retained
```

B. Dcontour.sh / Scontour.sh, syntax

```
-> for 2D fields:
 Dcontour.sh <option> <file1> <file2> <var1> <var2>
-> for 3D fields:
 Dcontour.sh <option> <file1> <file2> <var1> <var2> [<d3node1> <d3node2>]
-> for 4D fields:
 Dcontour.sh <option> <file1> <file2> <var1> <var2> \
             [<d3node1> <d3node2> (<d4node1> <d4node2>)]
where
   <option>    specifies dimensions and geo- or nongeo-grid
                implemented:
                 2d    - 2D fields
                 3d    - 3D fields
                 4d    - 4D fields
                 2dmap - 2D fields, dims. are lon & lat
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...2/3/4dmap will be displayed on a lon-lat grid
                  with a map
   <file1>     name of netcdf file w/<var1>
   <file2>     name of netcdf file w/<var2>,
                you may type '.' if <file1> & <file2> are the same
   <var1>      name of requested variable on <file1> (case sensitive)
   <var2>      name of requested variable on <file2> (case sensitive)
                you may type '.' if <var1> & <var2> are the same
   <d3node1>, <d3node2>
               node no.s of third dimension for <var1> and <var2>,
                respectively
               you may type '.' for <d3node2> if <d3node1>
                and <d3node2> are the same
               if <option> is one of 2d, 2dmap and a fourth
                 argument is present, or if <d3node1> is negative,
                 this will be interpreted as a flag that will cause
                 the ncl script to remain (see examples below)
   <d4node1>, <d4node2>
               node no.s of fourth dimension for <var1> and <var2>,
                respectively
               you may type '.' for <d4node2> if <d4node1>
                and <d4node2> are the same

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'Dcontour.sh' is given,
 the user may specify
  * title
  * zooming
  * map projection (lon-lat grids only)
  * color map (palette)
  * no. of colors
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).
```

```
Examples:
=========
 Dcontour.sh 4dmap hydro1.nc hydro2.nc temp T 1 1 10 10
   will produce a depiction on a lon-lat grid w/ a map,
    of the difference at the first vertical level and
    the tenth time step of the variables 'temp' and 'T',
    between these fields on 'hydro1.nc' and 'hydro2.nc'
    (positive where  hydro1.nc->temp .gt. hydro2.nc->T;
    when the third and fourth dimensions for both variables
    are vertical level and time, respectively
 Dcontour.sh 4dmap hydro1.nc . salt . 4 3 10 .
   will produce a depiction on a lon-lat grid w/ a map,
    of the difference at the tenth time step on 'hydro1.nc'
    between 'salt' at the fourth and third vertical level
    (positive where  salt(,,4,) .gt. salt(,,3,);
    when the third and fourth dimensions are vertical level
    and time, respectively
```

The syntax for Scontour.sh is identical to the syntax for Dcontour.sh.

C. mcontour.sh, syntax

```
NOTE: The user is **STRONGLY** recommended to copy
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'mcontour.sh' is given,
 and at least specify
  * color map (palette)
 -otherwise, the series of files produced by  mcontour.sh
  will (usually) NOT have the same color map (see more information below)
NOTE: This script will provide a set of output files, where
 the value of the final (3rd or 4th) dimension changes from
 one output file to the next. Below, we assume that this
 dimension is time.

Syntax:
=======
...if  <option> is  4d  or  4dmap:
 mcontour.sh <option> <file> <variable> <depth> <first> <last> (<step>)
...if  <option> is  3d  or  3dmap :
 mcontour.sh <option> <file> <variable> <first> <last> (<step>)
where
   <option>    specifies dimensions and geo- or nongeo-grid
                implemented:
                 3d    - 3D fields
                 4d    - 4D fields
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...3/4dmap will be displayed on a lon-lat grid
                  with a map
   <file>      name of the netcdf file
   <variable>  name of requested variable on the netcdf file
                (case sensitive)
   <depth>     vertical level no.
   <first>     first time step no.
   <last>      last  time step no.
   <step>      time step between consequtive frames
                (optional, set to  1  if not specified by user)

 The script will produce a set of pnm-files.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'mcontour.sh' is given,
 the user may specify
  * title
  * zooming
  * map projection (lon-lat grids only)
  * color map (palette)
  * no. of colors
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).
```

```
Examples:
=========
 mcontour.sh 4dmap hydrography.nc temp 1 10 15
   will produce 6 pnm-files for time steps 10-15
    of the first vertical level of the variable 'temp'
    on the file 'hydrography.nc', on a lon-lat grid w/ a map
 mcontour.sh 3d surface.nc sst 1 9 2
   will produce 5 pnm-files for time steps 1, 3, 5, 7 and 9
   on a x-y grid of the variable 'sst' on the file 'surface.nc'
```

D. c-mask.sh, syntax

```
 c-mask.sh <option> <file1> <file2> <var1> <var2> <val1> <val2> \
         [<d3node> (<d4node>)]
where
   <option>  specifies dimensions and geo- or nongeo-grid
               2d   /2dr    - 2D fields
               3d   /3dr    - 3D fields
               4d   /4dr    - 4D fields
               2dmap/2dmapr - 2D fields, dims. are lon & lat
               3dmap/3dmapr - 3D fields, first two dims.
                                are lon & lat
               4dmap/4dmapr - 4D fields, first two dims.
                                are lon & lat
               ...2/3/4dmap(r) requires that hor. dim.s
                                are lon & lat
               options *r mask values inside the range
                [<val1>,<val2>], other options mask values
                outside of the range
   <file1>   name of the netcdf file w/ variable to depict
   <file2>   name of the netcdf file w/ masking variable
              you may type '.' if <file1> & <file2> are the same
   <var1>    name of variable on the netcdf file to depict
              (case sensitive)
          NOTE! This script requires the existence of an
                attribute 'missing_value' to <var1>
   <var2>    name of variable to use for masking <var1>
              (case sensitive)
              you may type '.' if <var1> & <var2> are the same
   <val1>, <val2>
             limits for masking:
              <option> = *r  :
                  <var1> will be masked for values  INSIDE the range
                         (<val1>,<val2>)
              otherwise     :
                  <var1> will be masked for values OUTSIDE the range
                         (<val1>,<val2>)
   <d3node>  node no. of third dimension
               if <option> is one of   2d(r), 2dmap(r) and a fourth
                 argument is present, or if <option> is negative,
                 this will be interpreted as a flag that will cause
                 the ncl script to remain (see examples below)
   <d4node>  node no. of fourth dimension

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'c-mask.sh' is given,
 the user may specify
  * title
  * zooming
  * map projection (lon-lat grids only)
  * color map (palette)
  * no. of colors
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).
```

```
Examples:
=========
 c-mask.sh 4dmap hydrography.nc temp salt 34 35 1 10
   will produce a depiction on a lon-lat grid w/ a map,
    of the first vertical level and the tenth time step
    of the variable 'temp' on the file 'hydrography.nc'
   the 'temp' field will be masked wherever 'salt' is outside
    the range <34, 35>
 c-mask.sh 3dr surface.nc ssh sst 0 10 -1
   will produce a depiction on a x-y grid for the first
   node in the third dimension time step, of the variable
    'ssh' on the file 'surface.nc'; and the ncl script
    will be retained; the 'ssh' field will be masked
    wherever 'sst' is negative or >10
```

E. section.sh, syntax

```
 section.sh <file> <variable> <ndims> <dim1> <dim2> <node1> (<node2>)
where
   <file>      name of netCDF file
   <variable>  name of variable to depict
   <ndims>     no. of dimensions of the variable (3 or 4; for 2,
                use contour.sh xy ...)
   <dim1>      crossection's 1. dimension no. (1-3) [Fortran style]
   <dim2>      crossection's 2. dimension no. (2-4) [Fortran style]
   <node1>     node no. of first  non-depicted dimension
                a negative <node1> value is interpreted as a flag
                that stops the ncl script from being deleted
   <node2>     node no. of second non-depicted dimension
                (if <ndims> is 4)

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'section.sh' is given,
 the user may specify
  * title
  * zooming
  * color map (palette)
  * no. of colors
  * plot size limits
  (look up, or copy, this file to edit your own 'userdef' file).

Example:
=========
 section.sh sst.nc sst 3 1 3 60
   if the first and second dimensions are longitude and
    latitude, and the third is time, this will produce
    a Hovmoller diagram of sst variability along latitude
    node no. 60, based on results on the file 'sst.nc'
```

F. layersection.sh, syntax

```
NOTE: This script assumes that the variable to depict has been
 stored with dimensions in the order x  -y  -layer(-time)
                                      or lon-lat-layer(-time)
 (w/ Fortran style sequence of dimensions).

Syntax:
=======
 layersection.sh <hfile> <varfile> <hname> <varname> <dimname> <node> <time>
where
   <hfile>     name of netCDF file w/thickness results
   <varfile>   name of netCDF file w/requested variable
                you may use '.' if both variables are on the same file
   <hname>     name of thickness variable
   <varname>   name of variable to depict
            NOTE! This script requires the existence of an
                  attribute 'missing_value' to <varname>
   <dimname>   name of crossection's horizontal dimension
                e.g.,  lat  for a lat-z (meridional) crossection
   <node>      node of non-depicted dimension
                if <dimname> is  lat , this is the  lon  grid no.
   <time>      time step no. to depict (use  0  for x-  y-z &
                                                  lon-lat-z fields)
               a negative <time> value is interpreted as a flag that
                stops the ncl script from being deleted

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'layersection.sh' is given,
 the user may specify
  * title
  * zooming
  * color map (palette)
  * no. of colors
  * plot size limits
  (look up, or copy, this file to edit your own 'userdef' file).

Examples:
=========
 layersection.sh hycom_expt007.nc . thknss salin lat 30 49
   will produce a lat-z crossection normal to longitude node no. 30,
    of the variable 'salin' on the file 'hycom_expt007.nc',
    w/ layers given by 'thknss' on the same file,
    from timestep no. 49
 layersection.sh hycom_expt007.nc . thknss . lat 30 49
   same as above, but here, thknss is requested; this is a
    special case where the layer no. will be contoured
    (the thickess will correspond to the distance between
        layer interfaces)
```

G. mlayersection.sh, syntax

```
NOTE: The user is **STRONGLY** recommended to copy
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'mlayersection.sh' is given,
 and at least specify
  * color map (palette)
 -otherwise, the series of files produced by  mlayersection.sh
  will (usually) NOT have the same color map (see more information below)

NOTE: This script assumes that the variable to depict has been
 stored with dimensions in the order x  -y  -layer(-time)
                                       or lon-lat-layer(-time)
 (w/ Fortran style sequence of dimensions).

Syntax:
=======
 mlayersection.sh <hfile> <varfile> <hname> <varname> <dimname> <node> <first>
where
   <hfile>     name of netCDF file w/thickness results
   <varfile>   name of netCDF file w/requested variable
                you may use '.' if both variables are on the same file
   <hname>     name of thickness variable
   <varname>   name of variable to depict
            NOTE! This script requires the existence of an
                  attribute 'missing_value' to <varname>
   <dimname>   name of crossection's horizontal dimension
                e.g.,  lat  for a lat-z (meridional) crossection
   <node>      node of non-depicted dimension
                if <dimname> is  lat , this is the  lon  grid no.
   <first>     first time step no.
   <last>      last  time step no.
   <step>      time step between consequtive frames
                (optional, set to  1  if not specified by user)

 The script will produce a set of ppm-files.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'mlayersection.sh' is given,
 the user may specify
  * title
  * zooming
  * color map (palette)
  * no. of colors
  * plot size limits
  (look up, or copy, this file to edit your own 'userdef' file).

Example:
========
 mlayersection.sh hycom_expt007.nc . thknss salin lat 30 47 49
   will produce a lat-z crossection normal to longitude node no. 30,
    of the variable 'salin' on the file 'hycom_expt007.nc',
    w/ layers given by 'thknss' on the same file,
    from timestep no. 47, 48 and 49
```

H. addlayers.sh, syntax

```
 addlayers.sh <option> <file> <variable> <firstlevel> <lastlevel> \
            (<d4node>)
where
   <option>    specifies dimensions and geo- or nongeo-grid
                implemented:
                 3d    - 3D fields
                 4d    - 4D fields
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...3/4dmap will be displayed on a lon-lat grid
                  with a map
   <file>      name of the netcdf file
   <variable>  name of requested variable on the netcdf file
                (case sensitive)
   <firstlevel> this is the first vertical level no. in the addition
   <lastlevel>  this is the  last vertical level no. in the addition
   <d4node>  node no. of fourth dimension (usually time)

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'addlayers.sh' is given,
 the user may specify
  * title
  * zooming
  * map projection (lon-lat grids only)
  * color map (palette)
  * no. of colors
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).

Example:
========
 addlayers.sh 4dmap hydrography.nc thknss 1 5 10
   will produce a depiction on a lon-lat grid w/ a map,
    of the sum of values for the variable 'thknss' for
    layers 1-5 from the tenth time step, provided that
    the third and fourth dimensions are layer no. and
    time, respectively. Values will be read from the
    file 'hydrography.nc'
```

I. vector.sh, syntax

```
 vector.sh <option> <file> <u> <v> [<d3node> (<d4node>)]
where
   <option>    specifies dimensions and geo- or nongeo-grid
                implemented:
                 2d    - 2D fields
                 3d    - 3D fields
                 4d    - 4D fields
                 2dmap - 2D fields, dims. are lon & lat
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...2/3/4dmap will be displayed on a lon-lat grid
                  with a map
   <file>      name of the netcdf file
   <u>         name of variable w/ velocity in the x-direction
                on the netcdf file (case sensitive)
   <v>         name of variable w/ velocity in the y-direction
                on the netcdf file (case sensitive)
   <d3node>  node no. of third dimension
               if <option> is one of 2d, 2dmap and a fourth
                 argument is present, or if <d3node> is negative,
                 this will be interpreted as a flag that will cause
                 the ncl script to remain (see examples below)
   <d4node>  node no. of fourth dimension

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'vector.sh' is given,
 the user may specify
  * title
  * zooming
  * vector spec.s (size, distance, curly/regular vectors)
  * map projection (lon-lat grids only)
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).

Examples:
=========
 vector.sh 4dmap hydrography.nc u v 1 10
   will produce vectors on a lon-lat grid w/ a map, for
    the first node in the third dimension (usually the top
    vertical level) and the tenth node in the fourth
    dimension (usually time step no. 10), based on
    variables 'u' and 'v' on the file 'hydrography.nc'
 vector.sh 3d surface.nc u-vel v-vel -1
   will produce vectors on a x-y grid of the first node
    in the third dimension, based on variables 'u-vel' and
    'v-vel' on the file 'surface.nc'; and the ncl-script
     will be retained
 vector.sh 2dmap topography.nc ubaro vbaro a
   will produce vectors on a lon-lat grid w/ a map,
   based on variables 'ubaro' and vbaro' on the file
   'topography.nc'; and the ncl-script will be retained
```

J. Svector.sh, syntax

```
 Svector.sh <option> <file> <u> <v> <u_bt> <v_bt> [ <d3node> \
                                                    (<d4node>) ]
where
   <option>    specifies dimensions and geo- or nongeo-grid
                implemented:
                 2d    - 2D fields
                 3d    - 3D fields
                 4d    - 4D fields
                 2dmap - 2D fields, dims. are lon & lat
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...2/3/4dmap will be displayed on a lon-lat grid
   <file>      name of the netcdf file
   <u>         name of variable w/ baroclinic velocity in the
                x-direction on the netcdf file (case sensitive)
   <v>         name of variable w/ baroclinic velocity in the
                y-direction
   <u_bt>      name of variable w/ barotropic velocity in the
                x-direction
   <v_bt>      name of variable w/ barotropic velocity in the
                y-direction
   <d3node>  node no. of third dimension
               if <option> is one of 2d, 2dmap and a fourth
                 argument is present, or if <d3node> is negative,
                 this will be interpreted as a flag that will cause
                 the ncl script to remain
   <d4node>  node no. of fourth dimension

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'Svector.sh' is given,
 the user may specify
  * title
  * zooming
  * vector spec.s (size, distance, curly/regular vectors)
  * map projection (lon-lat grids only)
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).

Example:
========
 Svector.sh 4dmap hydrography.nc u v u_btrop v_btrop 1 10
   will produce vectors on a lon-lat grid w/ a map, of
    the sums 'u'+'u_btrop' and 'v'+'v_btrop' in the
    x- and y-directions, respectively; 'u' and 'v' will be
    extracted at the first first node in the third dimension
    (usually the top vertical level) and the tenth node in
    the fourth dimension (usually time step no. 10),
    whereas 'u_btrop' and 'v_btrop' are extracted at the
    tenth node in their third dimension; all variables
    will be read from the file 'hydrography.nc'
```

K. mvector.sh, syntax

```
NOTE: This script will provide a set of output files, where
 the value of the final (3rd or 4th) dimension changes from
 one output file to the next. Below, we assume that this
 dimension is time.

Syntax:
=======
...if  <option> is  4d  or  4dmap:
 mvector.sh <option> <file> <u> <v> <depth> <first> <last> (<step>)
...if  <option> is  3d  or  3dmap:
 mvector.sh <option> <file> <u> <v> <first> <last> (<step>)
where
   <option>    specifies dimensions and geo- or nongeo-grid
               implemented:
                3d    - 3D fields
                4d    - 4D fields
                3dmap - 3D fields, first two dims. are lon & lat
                4dmap - 4D fields, first two dims. are lon & lat
                ...3/4dmap will be displayed on a lon-lat grid
                 with a map
   <file>      name of the netcdf file
   <u>         name of variable w/ velocity in the x-direction
                on the netcdf file (case sensitive)
   <v>         name of variable w/ velocity in the y-direction
                on the netcdf file (case sensitive)
   <depth>     vertical level no.
   <first>     first time step no.
   <last>      last  time step no.
   <step>      time step between consequtive frames
                (optional, set to  1  if not specified by user)

 The script will produce a set of pnm-files.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'mvector.sh' is given,
 the user may specify
  * title
  * zooming
  * vector spec.s (size, distance, curly/regular vectors)
  * map projection (lon-lat grids only)
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).

Example:
=========
 mvector.sh 4dmap hydrography.nc u v 1 5 15 2
   will produce vectors on a lon-lat grid w/ a map, of
    the first vertical level and time steps no. 5, 7, 9, 11, 13 and 15,
    based on variables 'u' and 'v' on the file 'hydrography.nc'
```

L. transport.sh, syntax

```
NOTE: The user is **STRONGLY** recommended to copy
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'transport.sh' is given,
 and at least consider altering
  * vector spec.
 -otherwise, lengths of vector will be scaled as ocean currents
  rather than a transport quantity

Syntax:
=======
 transport.sh <option> <uvfile> <varfile> <u> <v> <var> [ <d3node> \
                                                          (<d4node>) ]
where
   <option>    specifies dimensions and geo- or nongeo-grid
               implemented:
                 2d    - 2D fields
                 3d    - 3D fields
                 4d    - 4D fields
                 2dmap - 2D fields, dims. are lon & lat
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...2/3/4dmap will be displayed on a lon-lat grid
                  with a map
   <uvfile>    name of the netcdf file w/ <u> and <v>
   <varfile>   name of the netcdf file w/ <var>
               you may type '.' if <uvfile> & <varfile> are the same
   <u>         name of variable w/ velocity in the x-direction
               on the netcdf file (case sensitive)
   <v>         name of variable w/ velocity in the y-direction
               on the netcdf file (case sensitive)
            NOTE! special case: if <v> is set to  1 ,
                                speed is contoured
   <var>       name of variable to base filled contours on
               on the netcdf file (case sensitive)
            NOTE! This script requires the same dimensions for
                  <u>, <v> and <var>, i.e., they must all be
                  2d, or 2dmap, etc.
   <d3node>  node no. of third dimension
               if <option> is one of 2d, 2dmap and a fourth
                 argument is present, or if <d3node> is negative,
                 this will be interpreted as a flag that will cause
                 the ncl script to remain (see examples below)
   <d4node>  node no. of fourth dimension

 The script will produce an eps-file and a png-file.

NOTE! Unless a user spec. file exists, this script will give
      rise to unreasonably long or short vectors if variable
      values are of a different order than 1.
```

```
User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'transport.sh' is given,
 the user may specify
  * title
  * zooming
  * vector spec.s (size, distance, curly/regular vectors)
  * map projection (lon-lat grids only)
  * color map (palette)
  * no. of colors
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).

Examples:
=========
 transport.sh 2dmap uv.nc topography.nc ubaro_mean vbaro_mean depth
   will produce transport vectors on top of filled contours
    for the volume transport, on a lon-lat grid w/ a map,
    based on variables 'ubaro', vbaro' on the file 'uv.nc',
    and 'depth' on the file 'topography.nc'
 transport.sh 4dmap hydrography.nc . u v temp 1 10
   will produce transport vectors on top of filled contours
    for temperature transport, on a lon-lat grid w/ a map,
    for the first node in the third dimension (usually
    the top vertical level) and the tenth node in the
    fourth dimension (usually time step no. 10), based on
    variables 'u', 'v' and 'var' on the file 'hydrography.nc'
 transport.sh 4dmap hydrography.nc . u v 1 1 10
   special case (variable to contour is set to '1'): will
    produce velocity vectors on top of filled contours for
    the current speed, for the first node in the third dimension
    and the tenth node in the fourth dimension, on a lon-lat
    grid w/ a map, based on variables 'u' and 'v' on the file
    'hydrography.nc'
```

M. v-on-c.sh, syntax

```
 v-on-c.sh <option> <uvfile> <varfile> <u> <v> <var> [ <d3node> \
                                                      (<d4node>) ]
where
   <option>    specifies dimensions and geo- or nongeo-grid
                implemented:
                 2d    - 2D fields
                 3d    - 3D fields
                 4d    - 4D fields
                 2dmap - 2D fields, dims. are lon & lat
                 3dmap - 3D fields, first two dims. are lon & lat
                 4dmap - 4D fields, first two dims. are lon & lat
                 ...2/3/4dmap will be displayed on a lon-lat grid
                  with a map
   <uvfile>    name of the netcdf file w/ <u> and <v>
   <varfile>   name of the netcdf file w/ <var>
                you may type '.' if <uvfile> & <varfile> are the same
   <u>         name of variable w/ velocity in the x-direction
                on the netcdf file (case sensitive)
   <v>         name of variable w/ velocity in the y-direction
                on the netcdf file (case sensitive)
   <var>       name of variable to base filled contours on
                on the netcdf file (case sensitive)
             NOTE! This script requires the same dimensions for
                   <u>, <v> and <var>, i.e., they must all be
                   2d, or 2dmap, etc.
   <d3node>  node no. of third dimension
               if <option> is one of 2d, 2dmap and a fourth
                argument is present, or if <d3node> is negative,
                this will be interpreted as a flag that will cause
                the ncl script to remain (see examples below)
   <d4node>  node no. of fourth dimension

 The script will produce an eps-file and a png-file.

User specifications:
====================
 By copying the default spec.s from
   /usr/local/selfmade/ncl/userdef.ncl
 to the directory where the command 'v-on-c.sh' is given,
 the user may specify
  * title
  * zooming
  * vector spec.s (size, distance, curly/regular vectors)
  * map projection (lon-lat grids only)
  * color map (palette)
  * no. of colors
  * plot size limits
  * coastline detail level
  (look up, or copy, this file to edit your own 'userdef' file).
```

```
Examples:
=========
 v-on-c.sh 4dmap uv.nc hydrography.nc u v temp 1 10
   will produce vectors on top of filled contours for
    temperature, on a lon-lat grid w/ a map, of the first node
    in the third dimension (usually the top vertical level)
    and the tenth node in the fourth dimension (ususally time
    step no. 10) based on variables
    'u' and 'v' on the file 'uv.nc', and
    'temp'      on          'hydrography.nc'
 v-on-c.sh 3d surface.nc . u-vel v-vel ssh -1
   will produce vectors on top of filled contours for
    sea surface height, on a x-y grid of the first node in
    the third dimension, based on variables 'u-vel', 'v-vel'
    and 'ssh' on the file 'surface.nc'; and the ncl script
    will be retained
 v-on-c.sh 2dmap ave.nc topography.nc ubaro vbaro topo a
   will produce vectors on top of filled contours for
    the bottom topography, on a lon-lat grid w/ a map,
    based on variables
    'ubaro' and 'vbaro' on the file 'ave.nc', and
    'topo'              on          'topography.nc';
    the ncl script will be retained
```

N. makemovie.sh, syntax

```
makemovie.sh will convert a set of pnm files to an animation,
             either in the mpeg format or in the fli format

Syntax:
=======
 makemovie.sh <format> <file root> <first> <last> (<step>)
where
  <format>          is the animation format,
                     either  mpeg  or  fli
  <file root>       is the file root, i.e. the file name without
                     the frame no. and suffix
                     (if  salt0001.png - salt0012.png is to be
                      animated, the file root is 'salt')
   <first>     first time step no.
   <last>      last  time step no.
   <step>      time step between consequtive frames
                (optional, set to  1  if not specified by user)

Examples:
=========
 makemovie.sh fli salt 19 30
   will make a fli movie  salt.fli  w/ 12 frames, based on images
    salt0019.pnm, salt0020.pnm, ..., salt0030.pnm

 makemovie.sh mpeg temp 12 120 12
   will make a mpeg movie  temp.mpg  w/ 10 frames, based on images
    temp0012.pnm, temp0024.pnm, ..., temp0120.pnm
```