oceanmap: Mapping oceanographic data

Presentation · March 2017

CITATIONS
0

READS
2,712

1 author:

Robert Klaus Bauer
Institute of Research for Development (IRD), Sète, France
30 PUBLICATIONS **67** CITATIONS

SEE PROFILE

www.umr-marbec.fr/bauer-robert.html
robert.bauer@ird.fr

oceanmap

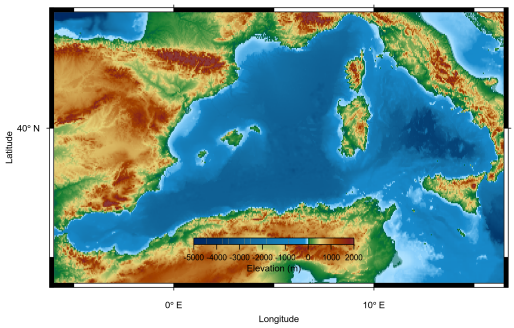
Mapping oceanographic data

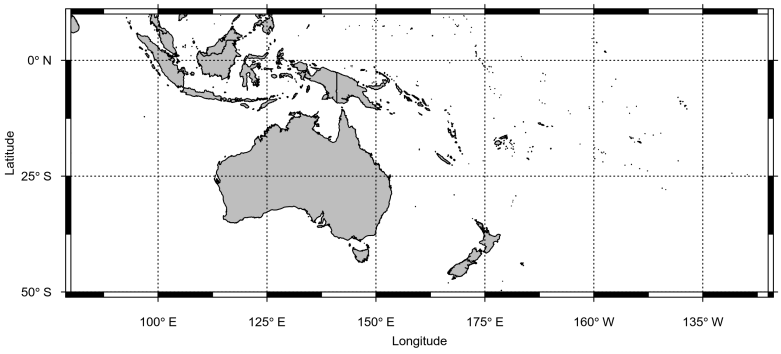
by Robert K. Bauer

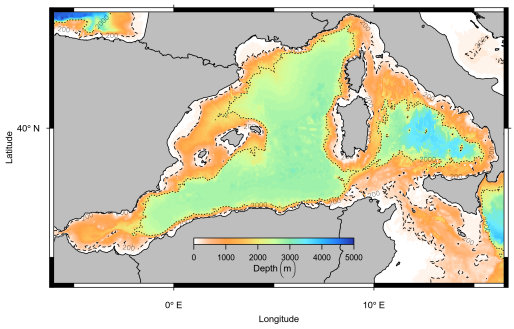
R-package tutorial v.0.0.4

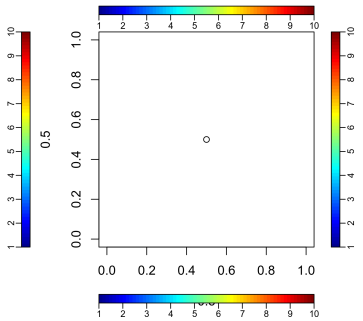
March 16, 2017

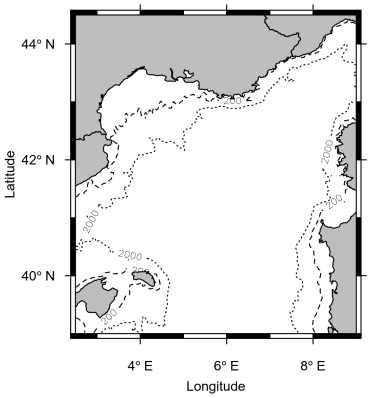
Sète, France











Introduction

Visualizing data is a crucial step in analyzing and exploring data. During the last two decades the statistical programming language R has become a major tool for data analyses and visualization across different fields of science. However, creating figures ready for scientific publication can be a tricky and time consuming task.
The oceanmap package provides some helpful functions to facilitate and optimize the visualization of geographic and oceanographic data, such as satellite and bathymetric data sets. Its major functions are

- ***plotmap***(), to plot land masks
- ***set.colorbar***(), to define the colorbar (position, ticks, colormap)
- ***v***(), an optimized tool for creating image or contour plots from different data formats (matrix, array, raster, netcdf, binary)
- ***figure***(), to facilitate resizing and saving of figures in diverse formats (jpeg, png, eps, pdf and eps)

These functions were written in a way that they do not require a large amount of their numerous arguments to be specified but still return nice plots. Each of these functions will be subsequently discussed.

A closely related problem represents the **extraction of spatial data** at defined position-s/areas. Some examples on this will be discussed in an extra section.

Getting oceanmap started

Installing oceanmap

```
install.packages('oceanmap')

library(oceanmap)
?oceanmap::oceanmap
```

Package 'oceanmap'

March 15, 2017

Type Package

Title A Plotting Toolbox for 2D Oceanographic Data

Version 0.0.4

Date 2017-03-15

Author Robert K. Bauer

Maintainer Robert K. Bauer <robert.bauer@ird.fr>

Depends R (>= 2.10), maps, mapdata, raster, extrafont

Imports abind, fields, marmap, plotrix, methods, utils, grDevices, maptools, sp, ncdf4, stats

Description Plotting toolbox for 2D oceanographic data (satellite data, sst, chla, ocean fronts & bathymetry). Recognized classes and formats include ncdf4, Raster, '.nc' and '.gz' files.

License GPL (>= 3)

NeedsCompilation no

Repository CRAN

Date/Publication 2017-03-15 16:03:02

SystemRequirements ImageMagick

R topics documented:

1

`.get.worldmap` *World Map*

Description

Creates a world map database that allows longitude ranges between -180 and 360 degrees, and thus from the Pacific to the Atlantic and vise versa. It is based on the worldHires database (which itself is based on CIA World Data Bank II data and contains approximately 2 million points representing the world coastlines and national boundaries), from which polygon irritations of the Antarctic were also corrected.

Usage

```
.get.worldmap(resolution)
```

Arguments

`resolution`	number that specifies the resolution with which to draw the map. Resolution 0 is the full resolution of the database [default]. Otherwise, just before polylines are plotted they are thinned: roughly speaking, successive points on the polyline that are within resolution device pixels of one another are collapsed to a single point (see the Reference for further details). Thinning is not performed if plot = FALSE or when polygons are drawn (fill = TRUE or database is a list of polygons).

Examples

```
owd <- getwd()
setwd(system.file("test_files", package="oceanmap"))
nfile <- Sys.glob('herring*.nc') # load sample-'.nc'-files
head(nc2time(nfile))

library('ncdf4')
nc <- nc_open(nfile)
head(nc2time(nc))

setwd(owd)
```

`oceanmap` *oceanmap - plot tools for 2D oceanographic data*

Description

`oceanmap` is a plotting toolbox for oceanographic data. Visualizing data is a crucial step in analyzing and exploring data. During the last two decades the statistical programming language R has become a major tool for data analyses and visualization across different fields of science. However, creating figures ready for scientific publication can be a tricky and time consuming task.

The `oceanmap` package provides some helpful functions to facilitate and optimize the visualization of geographic and oceanographic data, such as satellite and bathymetric data sets. Its plotting functions are written in a way that they do not require a large amount of their numerous arguments to be specified but still return nice plots. Its major functions are:

Major functions:

- plotmap: plots landmask as basis or overlay
- v: plots oceanographic data (fronts, SST, chla, bathymetry, etc.) from raster-objects, ncdf4- or gz-files
- set.colorbar: adds a colorbar to current figure, allowing several placement methods
- get.bathy: download bahymetric data at user defined resolution from the NOAA ETOPO1 database
- add.region: generate region definitions to facilitate land mask and colorbar plotting using plotmap and v
- figure & close_fig: generate and save graphic devices in flexible file formats (jpeg, png, eps, pdf and eps)

Getting Started
Check out some examples of the principle functions, listed above.

Author(s)

Robert K. Bauer

Figure 1. Extracts of the oceanmap R-package manual

The *plotmap*-function

Often it is required to add a land mask as geographical reference to an image- or scatter-plot. To add a landmask it is required to provide information about the region extent. As for the entire *oceanmap*-package, the idea of the ***plotmap***-function is to reduce the user effort when creating figures. To allow a high level of user flexibility ***plotmap*** **can produce a may by 5 different ways**. By providing the required region extent as:

1. geographical coordinates (longitude and latitude),
2. a raster-object
3. an extent-object
4. a region-keyword
5. or add a land mask to an existing plot

Many additional arguments can passed by the user. The idea of a region-keyword (option 4) is to use a short character string instead of requiring axes limits. This keyword feature is also implemented in the ***v***-function and permits here the rapid visualization of subregions of interest (e.g. from a raster object). Region keywords can be set up by the ***add.region***-function. The latter function is interactive and also defines the default colorbar placement that is further used by the ***v***-function.

5 different ways to produce a map with *plotmap*() - examples

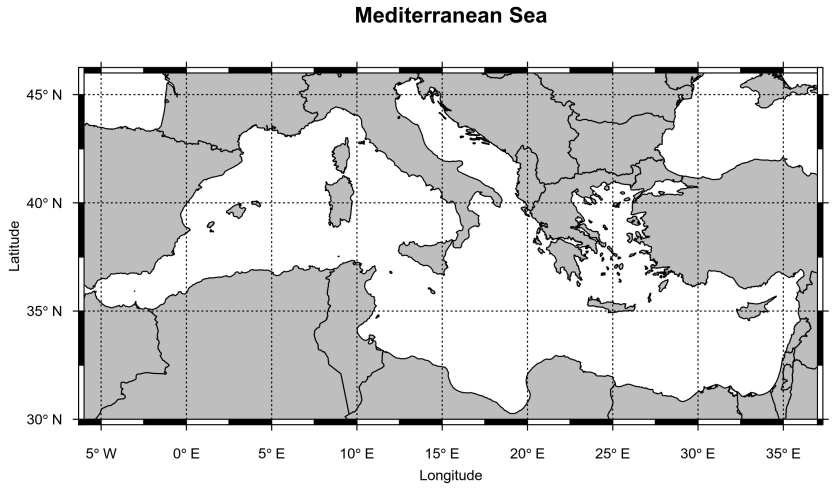


Figure 2. map of the Mediterranean Sea produced by ***plotmap***

```
#### Example 1: plot landmask of the Mediterranean Sea
## a) by using longitude and latitude coordinates:
lon <- c(-6, 37)
lat <- c(30, 46)
figure(width=9.75,height=5.28)
plotmap(lon=lon, lat=lat, main="Mediterranean Sea")
plotmap(xlim=lon, ylim=lat, main="Mediterranean Sea")

## b) plot landmask of the Mediterranean Sea by using an extent-object
   :
library('raster')
ext <- extent(lon, lat)
plotmap(ext, main="Mediterranean Sea") # extent-object

## c) plot landmask of the Mediterranean Sea by using a raster-object:
r <- raster(ext)
plotmap(r, main="Mediterranean Sea") # raster-object

## d) plot landmask of the Mediterranean Sea by using a region label:
plotmap('med4', main="Mediterranean Sea") # region-label
# regions() ## check preinstalled region label

## e) add landmask to an existing plot:
plot(3.7008, 43.4079, xlim=lon, ylim=lat)
plotmap(add=T)
points(3.7008, 43.4079, pch=19)
```

```
#### Example 2: subplots and some additional arguments of plotmap()
par(mfrow=c(2, 1))
plotmap('medw4', main="Western Mediterranean Sea",col.bg="darkblue")
plotmap('medw4', main="Western Mediterranean Sea", bwd=3, border='grey
    ', grid=FALSE)
```

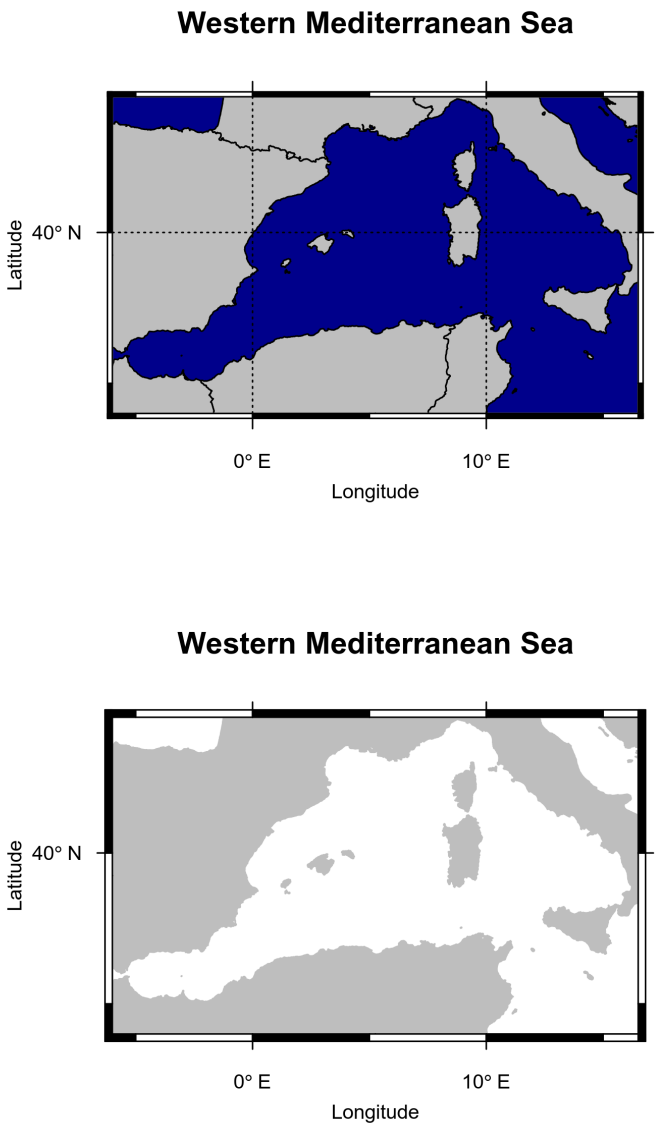


Figure 3. maps of the western Mediterranean Sea produced by ***plotmap***, changing scale bar width, background and country border colours.

Using *plotmap*() with *figure*(): aspect ratio when resizing figures

Note that you can, unlike with the basic ***map***-package, resize figures generated by ***plotmap*** (and ***v***)-function calls. (In fact, only the font and border width size will change when resizing figures.)

Recommendation:

Use ***dev.size()*** to find the most appropriate dimensions for your plot and **check out *figure()* to switch between the plotting window and diverse figure file formats: *jpeg, png, eps*** or ***pdf*.** Formats are selected according to the type statement ***(e.g. type=jpeg)***.
ATTENTION: To save plots with figure(), the device needs to be closed by a seperate function (***dev.off***() or ***close_fig***(TRUE))

```
#### Example 3: plotmap() and figure()
do.save <- FALSE ## open a plotting window
figure("Gulf_of_Lions_extended", do.save=do.save, width=5, height=5,
   type="pdf")
plotmap("lion",col.bg='darkblue',grid=FALSE)
close_fig(do.save)

## now resize figure manually and get new figure dimensions:
width <- dev.size()[1]
height <- dev.size()[2]

do.save <- TRUE ## do NOT open a plotting window, but save figure
   internally
figure("Gulf_of_Lions_extended", do.save=do.save, width=width, height=
   height, type="pdf")
plotmap("lion",col.bg='darkblue',grid=FALSE)
close_fig(do.save)
```

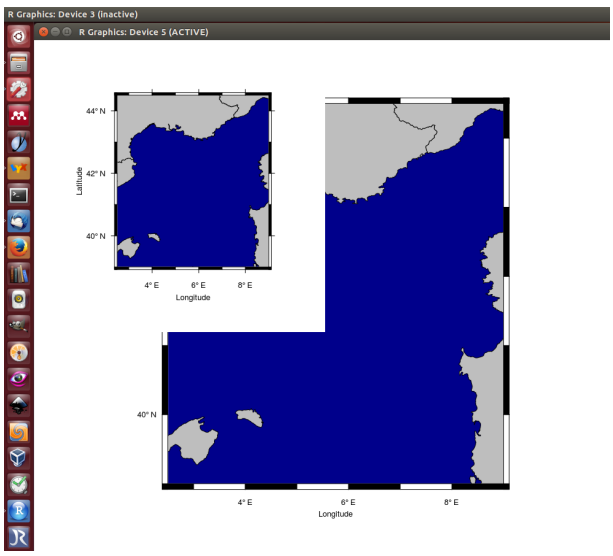


Figure 4. ***plotmap*** keeps the plot dimensions when resizing figures.

Drawing maps from *West to East* or *East to West:*

Particularly tricky but not that uncommon is the case of drawing maps that cover the region between the "Western" or "Eastern" hemispheres (i.e. that run from 180°W to 180°E or 180°E to 180°W). Whether ***plotmap*** draws a map from the Western to the Eastern or Eastern to Western hemisphere depends on the order and magnitude of the defined longitude values. In general, ***xlim* and *lon*-values < 0 or > 180 are always treated as Western longitudes**. Accordingly, maps are drawn from ***East to West*** **if** `xlim[1] > xlim[2]` and from ***West to East*** if `xlim[2] > xlim[1]`. Here an easy example:

```
#### Example 4: between hemispheres
par(mfrow=c(2,1))
plotmap(lon=c(80, -120), lat=c(-50, 10), main="map from East to West")
plotmap(lon=c(-120, 80), lat=c(-50, 10), main="map from West to East")
```

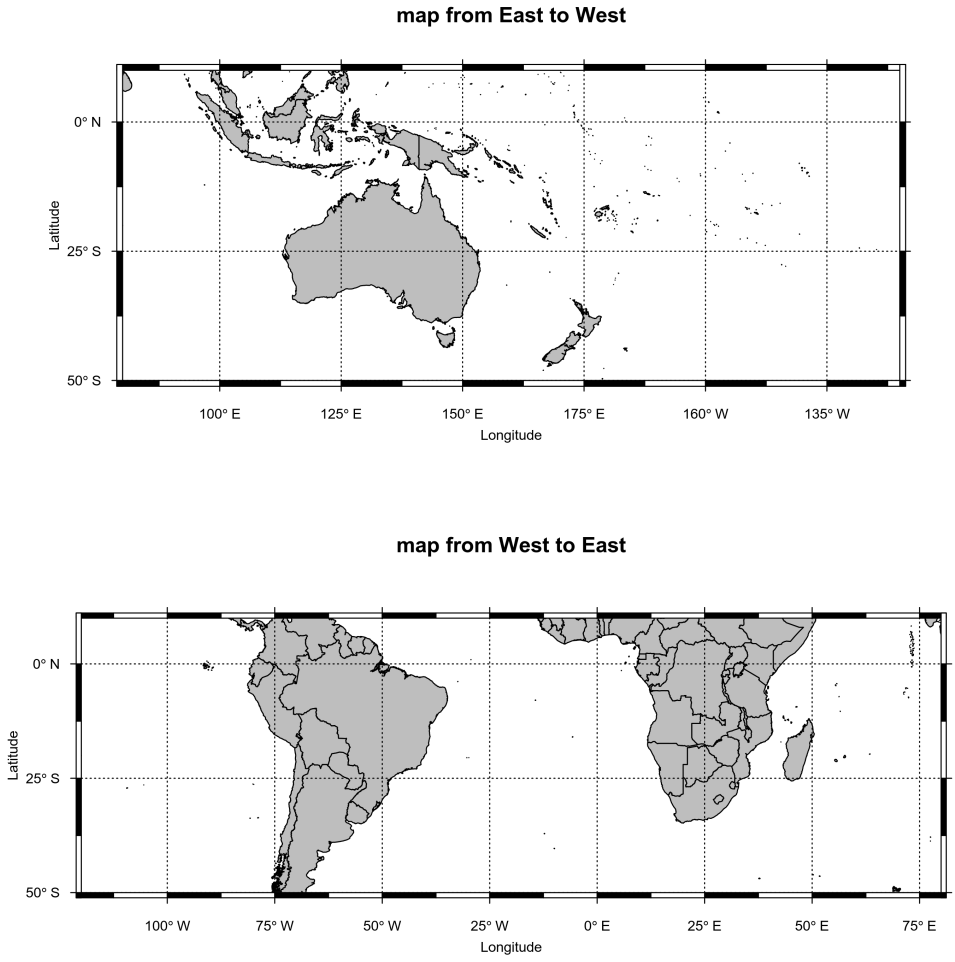


Figure 5. Drawing maps between hemispheres with ***plotmap***.

The *set.colorbar*-function

Setting up the colorbar for an image plot in R is particularly tricky. Common plot functions, such as ***image.plot*** of the ***fields***-package set the colorbar to a defined margin, but lack manipulation procedures. Setting up a colorbar by hand requires a lot of data input and is hence time consuming. **In particular four different arguments are needed:**

- the limits of the colorbar, defining its position, height and width
- the axis where to set tick labels (bottom, left, top, right)
- the colormap and its gradient (horizontal or vertical)
- the labels of the colorbar

The ***set.colorbar***-function of the *oceanmap*-package comes with **3 ways to define the colorbar** position:

1. defining its spatial extent (arguments **cbx, cby**)
2. defining argument **cbpos** with a single letter ("b", "l", "t", "r") that indicates the position of the colorbar (bottom, left, top, right)
3. set the position of colorbar **manually with the mouse cursor** (run ***set.colorbar***() without defining cbx, cby or cbpos).

In either case, the spatial extent of the thus defined colorbar position and extent will be returned (as cbx and cby) that can be reused in later function calls. This feature is further included in the ***add.region***- and ***v***-functions to set up the (default) colorbar placement of regions.
Note that the **colormap** and **ticks** of the colorbar can be defined through additional arguments of ***set.colorbar***()

```
## Example 1: plot colorbars manually
par(mar=c(8,8,8,8))
plot(0.5,0.5,xlim=c(0,1),ylim=c(0,1))
set.colorbar(cbx=c(0, 1), cby=c(-.3, -.4)) # bottom
set.colorbar(cby=c(0, 1), cbx=c(-.4, -.3)) # left
set.colorbar(cbx=c(0, 1), cby=c(1.2, 1.3)) # top
set.colorbar(cby=c(0, 1), cbx=c(1.2, 1.3)) # right


## Example 2: use cbpos
par(mar=c(8,8,8,8))
plot(0.5,0.5,xlim=c(0,1),ylim=c(0,1))
set.colorbar(cbpos='b') # bottom
set.colorbar(cbpos='l') # left
set.colorbar(cbpos='t') # top
set.colorbar(cbpos='r') # right
```

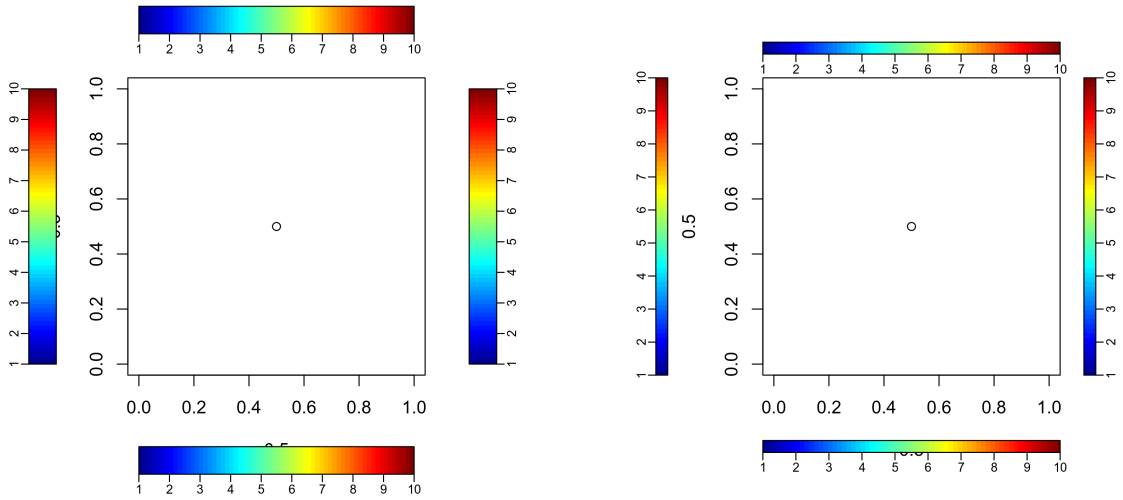


Figure 6. example colorbars created with the ***cbx/cby*** (left) and ***cbpos***-arguments (right) of the ***set.colorbar***()- function

```
## Example 3: interactive placement
# par(mar=c(8,8,8,8))
# plot(0.5,0.5,xlim=c(0,1),ylim=c(0,1))
# cb <- set.colorbar() # interactive
# plot(0.5,0.5,xlim=c(0,1),ylim=c(0,1))
# set.colorbar(cbx=cb$cbx, cby=cb$cby) # reuse stored colorbar
   positions
```

Interactive colorbar placement

```
set.colorbar()
Warning in set.colorbar() :
  colorbar positions missing or not of correct length (cbx and/or cby)!

Please select the lower left colorbar-position by the mouse cursor.
Please select the upper right colorbar-position by the mouse cursor.
```

The *v*-function: Creating image-plots of spatial data

The ***v***-function combines the advantages of the ***plotmap*** and ***set.colorbar*** **to** ***v***isualize **2D (oceanographic) data.** To facilitate plotting, **it is further capable of handling different input formats**. Valid formats are:

- ***matrix*** and ***array*** objects
- objects of class '***Raster***' ('RasterLayer', 'RasterStack' or 'RasterBrick'),
- ***netcdf***-files '***ncdf4***'-objects,
- '***.gz***'-files (Hervé Demarq)

Note that for plotting the input data will be internally transformed into a **Raster-object**, no matter which of the valid input format was selected. Examples to visualize ***matrix*** and ***array*** objects will not be discussed here but are given in `?matrix2raster()`. The other input data formats are discussed in the following sections.

v() and *RasterLayers:*

The standard object type to handle spatial data in R are Raster-objects. Here some examples to visualize such objects with *v*().

```
## Example 1: load & plot a sample Raster-object
setwd(system.file("test_files", package="oceanmap"))
load("medw4_modis_sst2_4km_1d_20020705_20020705.r2010.0.qual0.Rdata",
   verbose=TRUE)
dat <- raster::crop(dat,extent(c(0,10,40,44))) ## crop data, xlim/ylim
    not yet implemented in v()
print(dat)
```

print(dat)-output

```
class       : RasterLayer
dimensions  : 96, 240, 23040  (nrow, ncol, ncell)
resolution  : 0.04166667, 0.04166667  (x, y)
extent      : 0, 10, 40, 44  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +ellps=WGS84
data source : in memory names       : layer
values      : 16.8, 25.8  (min, max)
```

```
v(dat, main="Raster-object", cbpos='r')
```

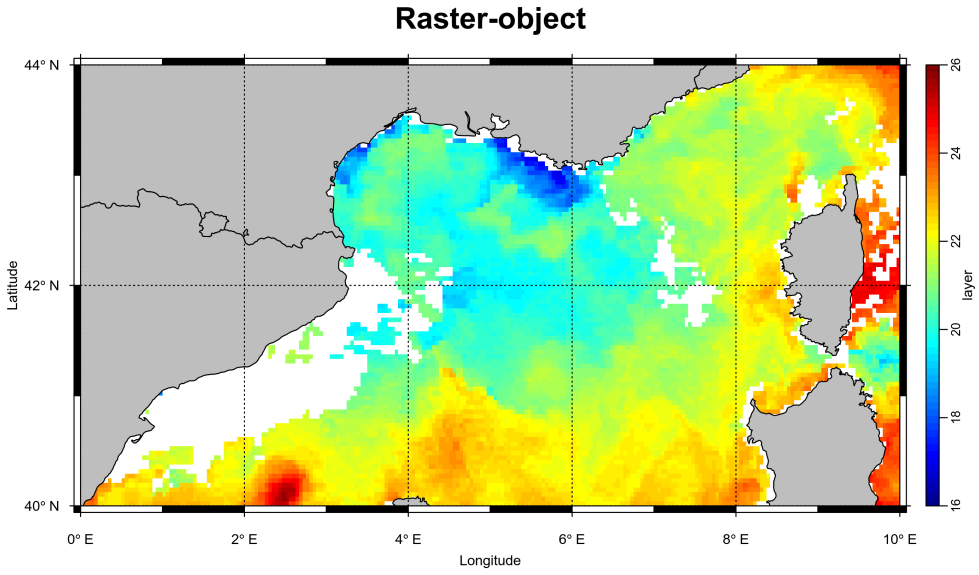


Figure 7. plotting a Raster-object with *v*()

v() and *netcdf*-files

```
## Example 2: load & plot sample netcdf-file ('.nc'-file)
setwd(system.file("test_files", package="oceanmap"))
nfiles <- Sys.glob('*.nc') # load list of sample-'.nc'-files
head(nfiles)

### option a) load netcdf-file with ncdf4-package and plot it
library('ncdf4')
ncdf <- nc_open(nfiles[1])
print(ncdf)
```

print(ncdf)-output

```
File herring_lavae.nc (NC_FORMAT_CLASSIC):
     1 variables (excluding dimension variables):
        float Conc[lon,lat,time]
               _FillValue: -1

     3 dimensions:
        lon  Size:352
        lat  Size:310
        time  Size:4   *** is unlimited ***
            units: seconds since 2006-03-01 00:00:00
```

```
v(obj = ncdf, cbpos="r")

### option b) load and plot netcdf-file as RasterStack object
nc <- nc2raster(nfiles[1])
```

ATTENTION: nc2raster()-calls require a "varname" selection if the netcdf-file holds multiple spatial variables.

```
v(nc,cbpos="r") # plot RasterStack object

### option c) plot netcdf-file directly
v(nfiles[1], cbpos="r")
v(nfiles[1], cbpos="r", replace.na=TRUE)
```

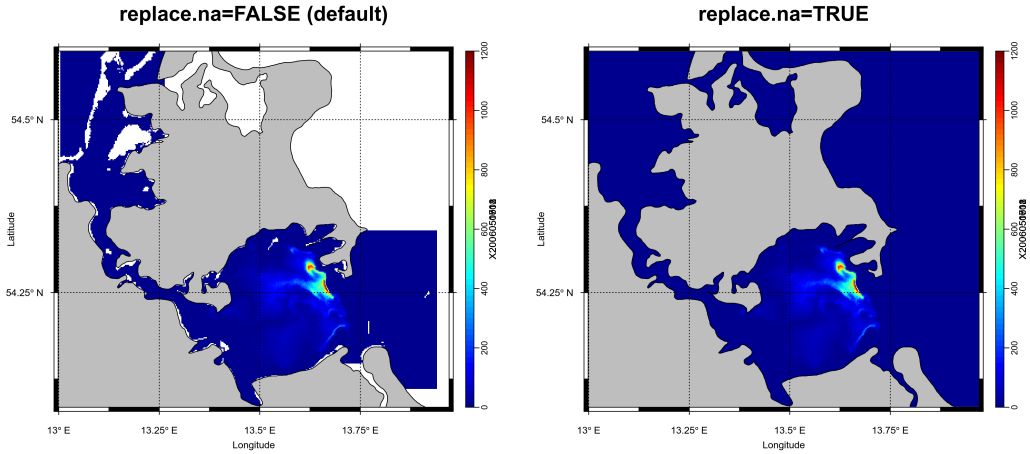


Figure 8. plotting ***netcdf***-files with ***v***()

```
###### plot multiple layers:
par(mfrow=c(2,2))
v(nfiles[1], t=1:4, cbpos="r", replace.na=TRUE, subplot = TRUE)
```

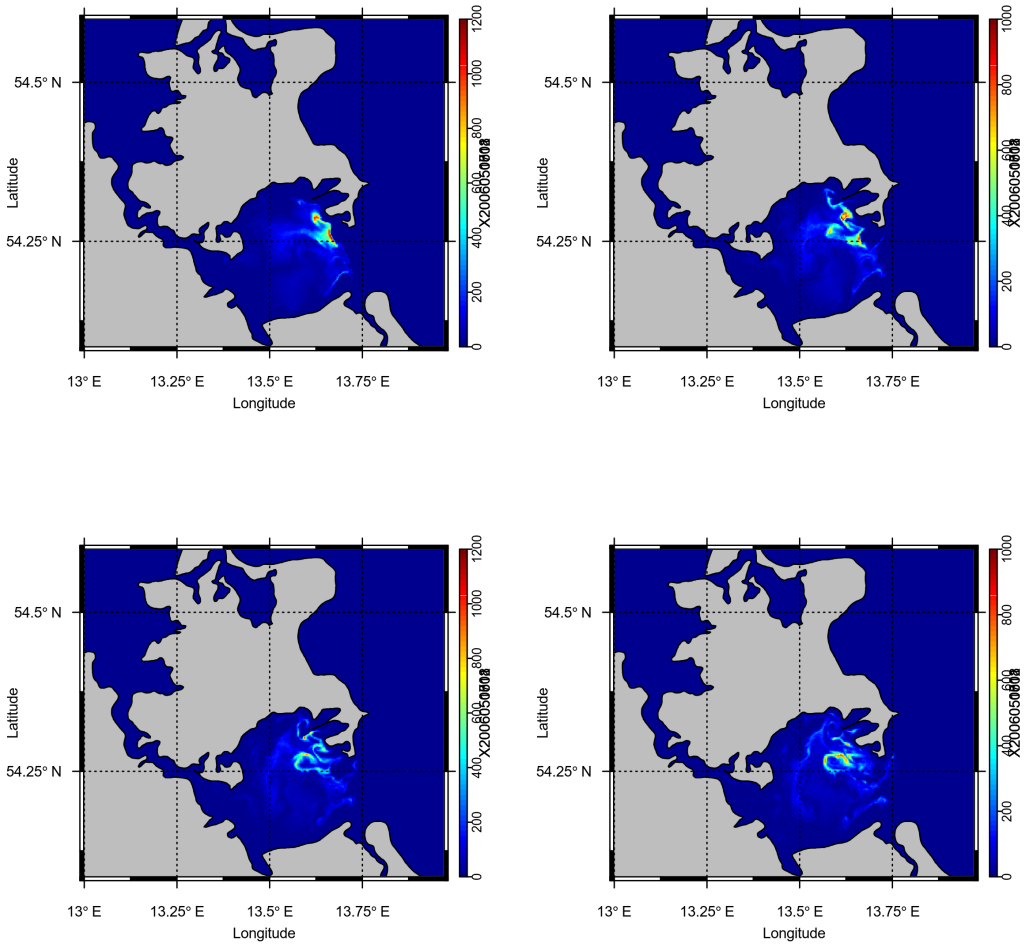


Figure 9. plotting multiple layers of ***netcdf***-files or RasterStack objects with ***v***

v() and *get.bathy*():

Gathering oceanographic data is not yet implemented in oceanmap. However, bathymetric data can already be obtained from the ***NOAA ETOPO1*** data base (https://data.noaa.gov/dataset/etopo1-1-arc-minute-global-relief-model) thanks to the ***get.bathy***() function of **oceanmap**. In order to download the bathymetry of a region, this function requires information on the region extent that can be provided as

1. longitude and latitude coordinates (lon, lat)
2. an extent (raster)-object
3. a region-keyword

Given this information ***get.bathy***() downloads the bathymetric data, plots and converts it in a **RasterLayer**-object. The ***default resolution of the grid is 4 minutes***, but can be changed by through the resolution statement of ***get.bathy***(). Note that for resolutions < 1 min, the NOAA server will interpolate the bathymetry, which is very time consuming and can be done the same way, but much faster with the function ***disaggregate***() of the raster package.

calling get.bathy() with coordinates (lon, lat):

```
## Example 1: load & plot bathymetry of the Baltic Sea, defined by
   longitudes and latidtues
lon <- c(9, 31)
lat <- c(53.5, 66)
# get.bathy(lon=lon, lat=lat, main="Baltic Sea", cbpos='r')
```

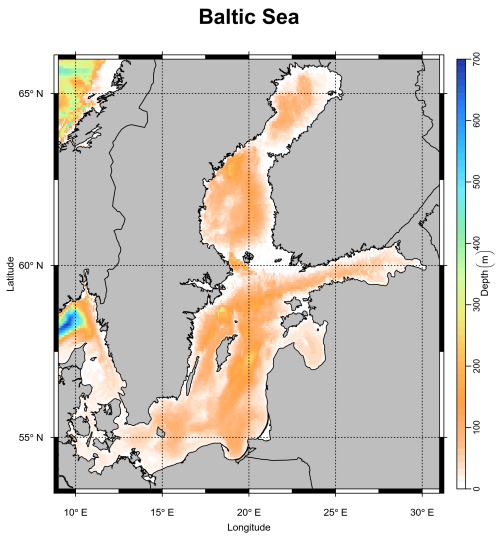


Figure 10. ***get.bathy*** & ***v***-examples for bathymetry plots - part I

***calling get.bathy*() with a region-keyword:**

```
# ## Example 2: load & plot bathymetry data from the NOAA-ETOPO1
   database
# par(mfrow=c(2,1))
# bathy <- get.bathy("medw4", terrain=T, res=3, keep=T, visualize=T,
   subplot = TRUE, grid=F)
# # load('bathy_medw4_res.3.dat',verbose = T); bathy <- h
# v(bathy, param="bathy", subplot = TRUE, terrain=F, levels=c
   (200,2000)) # show contours
```

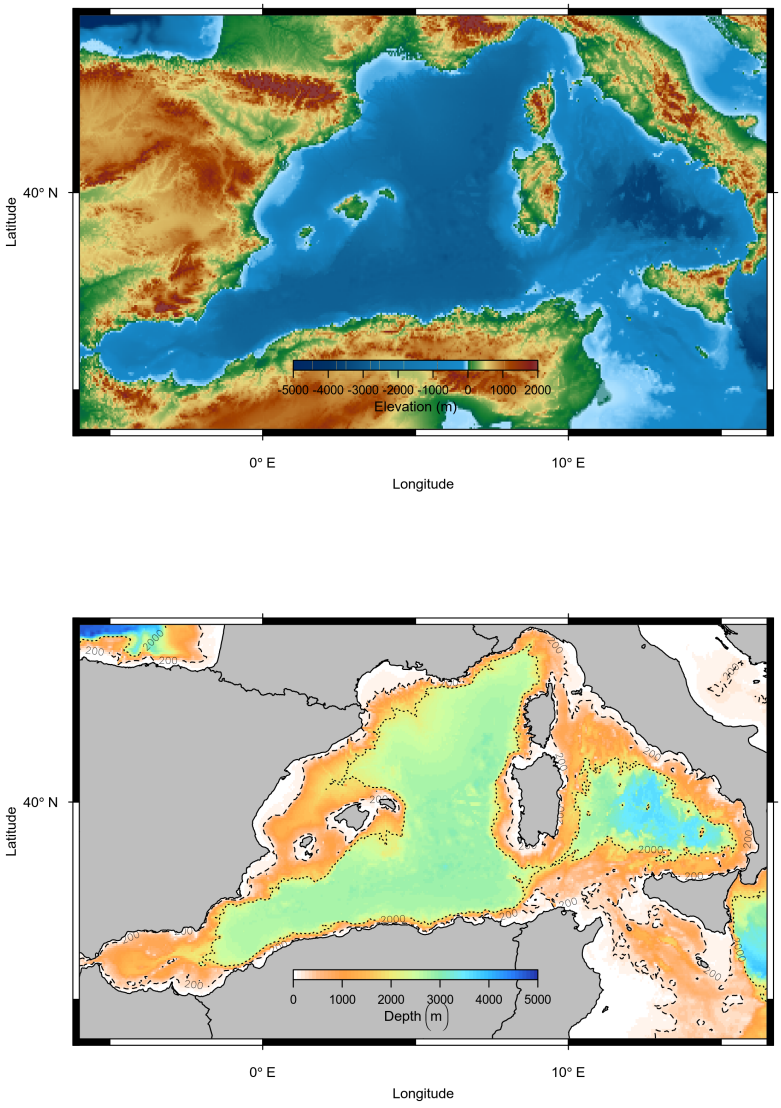


Figure 11. ***get.bathy*** & ***v***-examples for bathymetry plots - part II

How to produce only contour plots from bathymetry data: set `v_image=FALSE`

```
# ## b) only contour lines:
# par(mfrow=c(1,2))
# h <- get.bathy("lion",terrain=F,res=3, visualize=T,
#                subplot=T, v_image = FALSE, levels=c(200,2000))
#
# ## use v-function for same plot but on subregion:
# v(h,v_area = "survey", param="bathy",
#   subplot=T, v_image = FALSE, levels=c(200,2000))
```

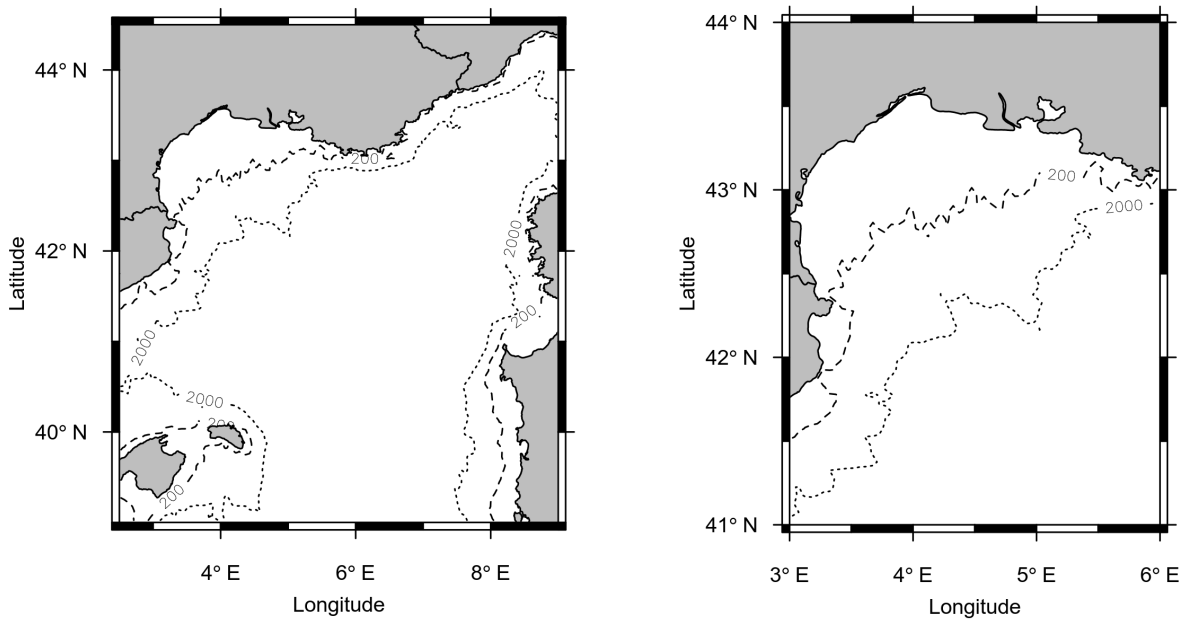


Figure 12. ***get.bathy*** & ***v***-examples for contour line plots

v() and *gz*-files:

```
## Example 3: plot sample-'.gz'-file
owd <- getwd()
setwd(system.file("test_files", package="oceanmap"))
gz.files <- Sys.glob('*.gz')
v(gz.files[2]) ## plot content of gz-file
```

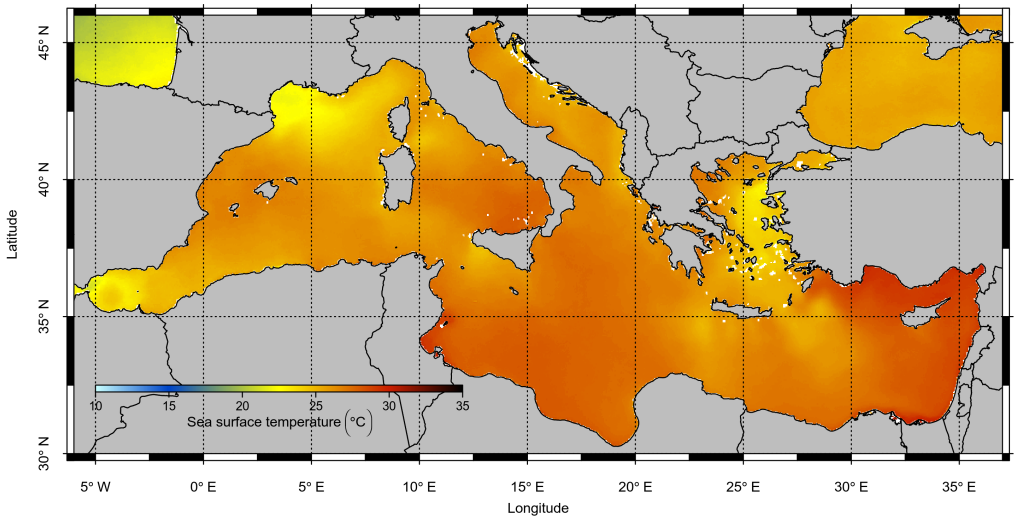


Figure 13. plotting a ***gz***-file with ***v***()

load and plot *gz*-file as a Raster-object (standard spatial object in R):

```
## Example 4: load sample-'.gz'-file manually as Raster-object and
   plot it
obj <- readbin(gz.files[2],area='lion')
par(mfrow=c(1,2))
v(obj,param="sst",subplot = TRUE)
v(obj,param="Temp",subplot = TRUE) ## note unset "pal" (colormap) for
   unkown "param"-values!
```

Missing or non-defined "param"-definitions mean also undefined "pal"-values (colormap)

```
Warning message:
In .v.plot(b = b, minv = minv, maxv = maxv,
zlim = zlim, adaptive.vals = adaptive.vals,  :
  "pal" not defined, "jet" selected! available colormaps:
...
```

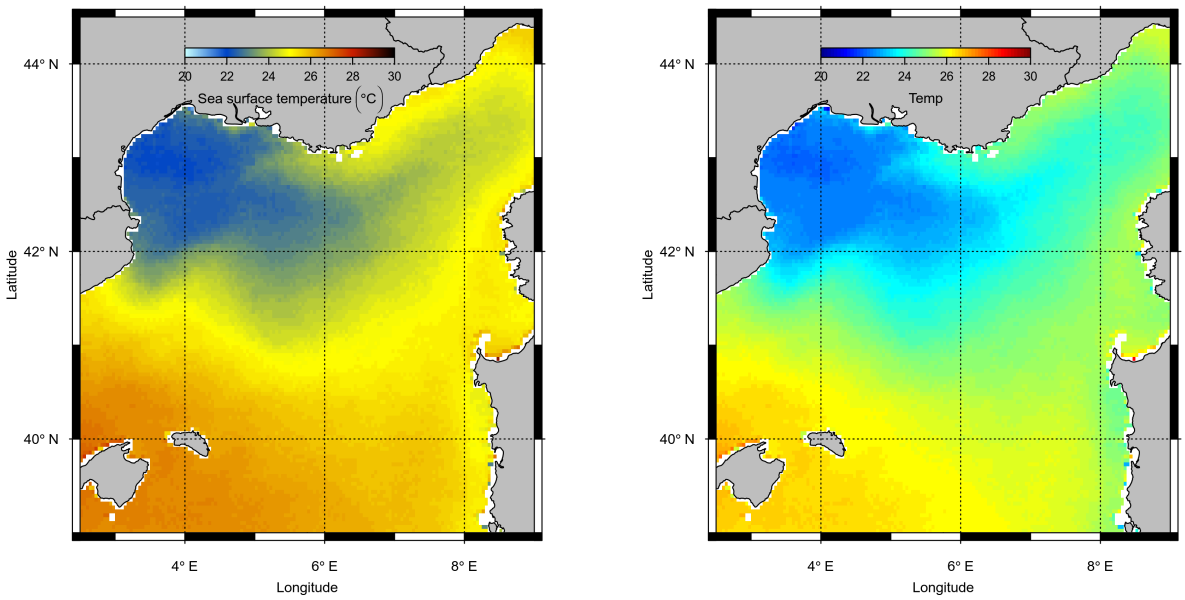


Figure 14. plotting ***Raster***-objects with ***v***()

available colormaps

oceanmap comes with a set of pre-installed colormaps. However, the user can apply own colormaps in ***v***-function calls.

```
## Example 5: available color maps
data('cmap') # load color maps data
names(cmap) # list available color maps

setwd(system.file("test_files", package="oceanmap"))
gz.files <- Sys.glob('*.gz')
figure(width=15,height=15)
par(mfrow=c(4,5))
for(n in names(cmap)) v(gz.files[2], v_area='lion', subplot=TRUE,
                        pal=n, adaptive.vals=TRUE, main=n)

## define new color maps from blue to red to white:
n <- colorRampPalette(c('blue','red','white'))(100)
v(gz.files[2], v_area='lion', subplot=TRUE,
  pal=n, adaptive.vals=TRUE, main="own colormap")
```

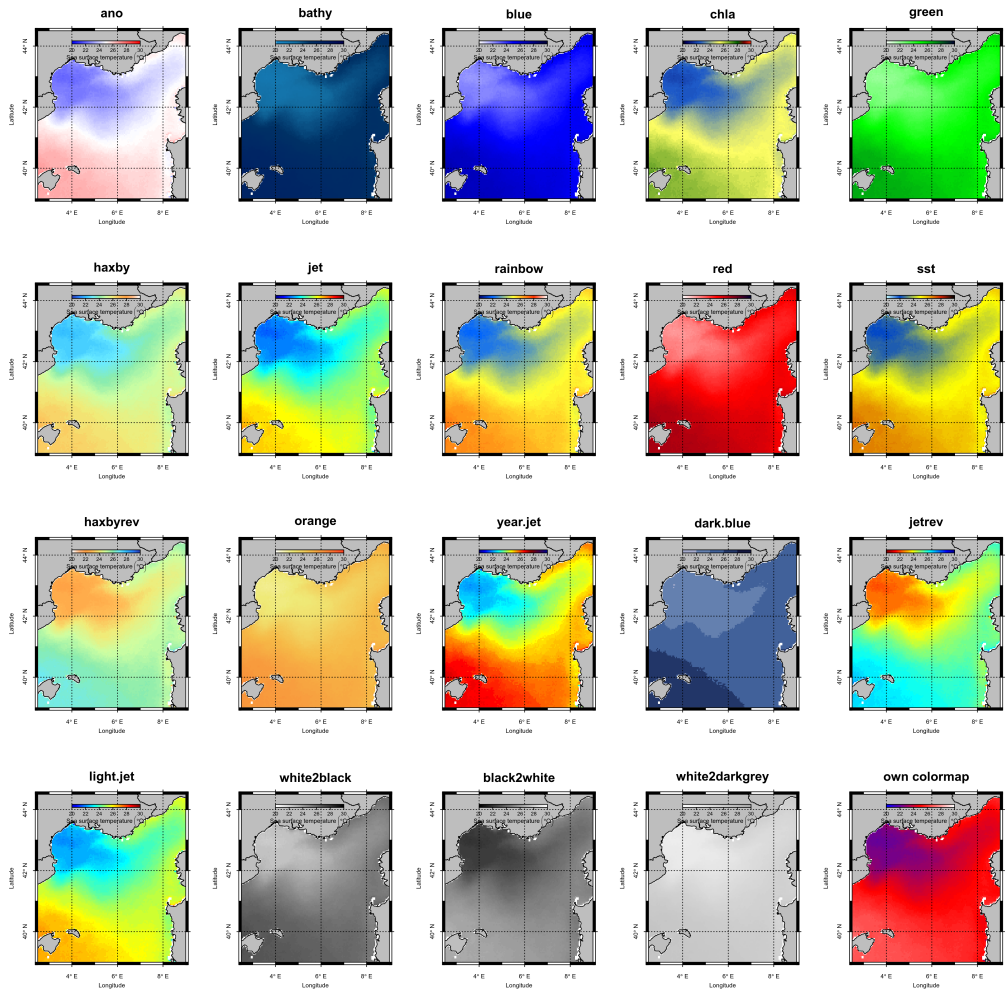


Figure 15. plotting available and new defined colormaps with ***v***()

available oceanographic parameters

The standard ***param*** values required in ***v***() to define the title and colormap of the colorbar are stored in the ***parameter_definitions*** dataset. Here some examples of pre-installed definitions.

```
## Example 6: available parameters
data(parameter_definitions)
names(parameter_definitions)
# ?parameter_definitions

setwd(system.file("test_files", package="oceanmap"))
figure(width=12, height=6.2)
par(mfrow=c(2,3))
v('*sst2*707*',v_area="medw4",main="sst", subplot=TRUE)
v('*chla*531*',v_area="medw4",main="chla", subplot=TRUE)
v('*chlagrad*',v_area="medw4",main="chlagrad",subplot=TRUE)
v('*p100*',v_area="medw4",main="p100 (oceanic fronts)",subplot=TRUE)
v('*sla*',v_area="medw4",main="sla",subplot=TRUE)
# h <- get.bathy("medw4",visualize=TRUE,terrain=F,res=4, subplot=TRUE,
   main="bathy")
```

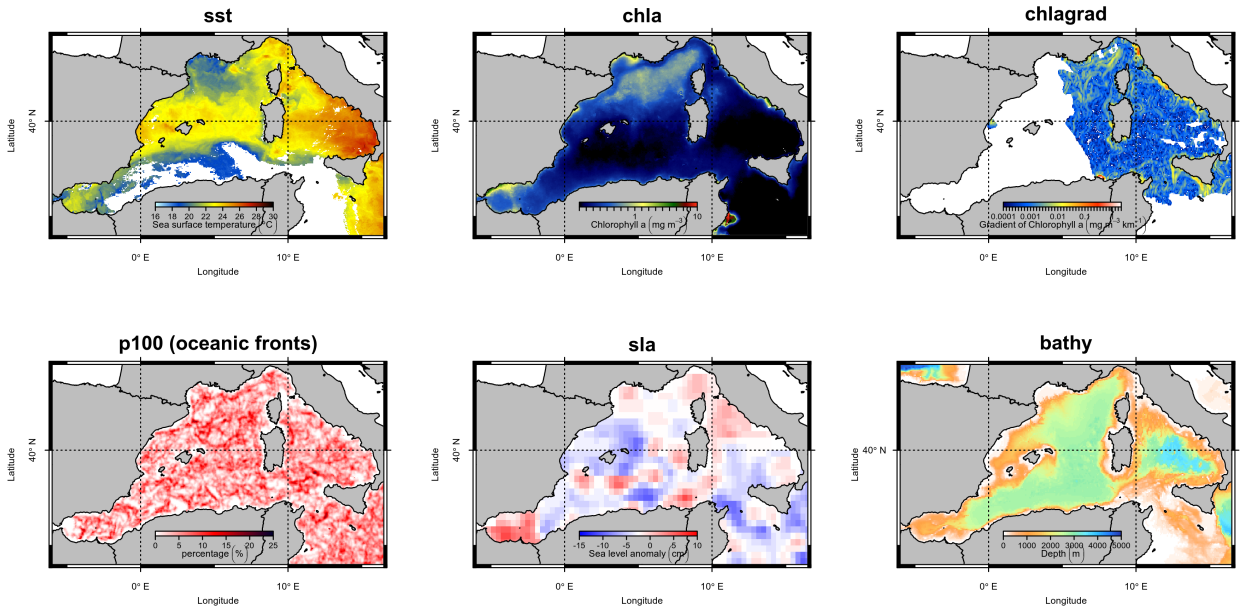


Figure 16. Sample images of some available parameters in ***v***().

available region keywords & definitions

Researchers often need to analyse spatial data of a fixed study region (e.g. the Western Mediterranean Sea). To facilitate plotting of such data, ***oceanmap*** comes with a ***region-keyword*** feature. The idea of this is to link all information required to produce land mask or image plots to a short keyword (label). Apart from its keyword, each region-definition includes a long version of the label, the spatial extent of the region (longitudes and latitudes), its grid resolution, as well as the default colorbar position and default figure dimensions. Available region-definitions are stored as a data frame in the ***region_definitions*** file of the package.

```
data(region_definitions)
head(region_definitions)
region_definitions$label
# ?region_definitions

figure(width=15,height=15)
par(mfrow=c(5,6))
for(n in region_definitions$label) plotmap(region = n,main=n)
```

The figure, produced by this R-code, illustrates the pre-installed region-definitions on the next page.

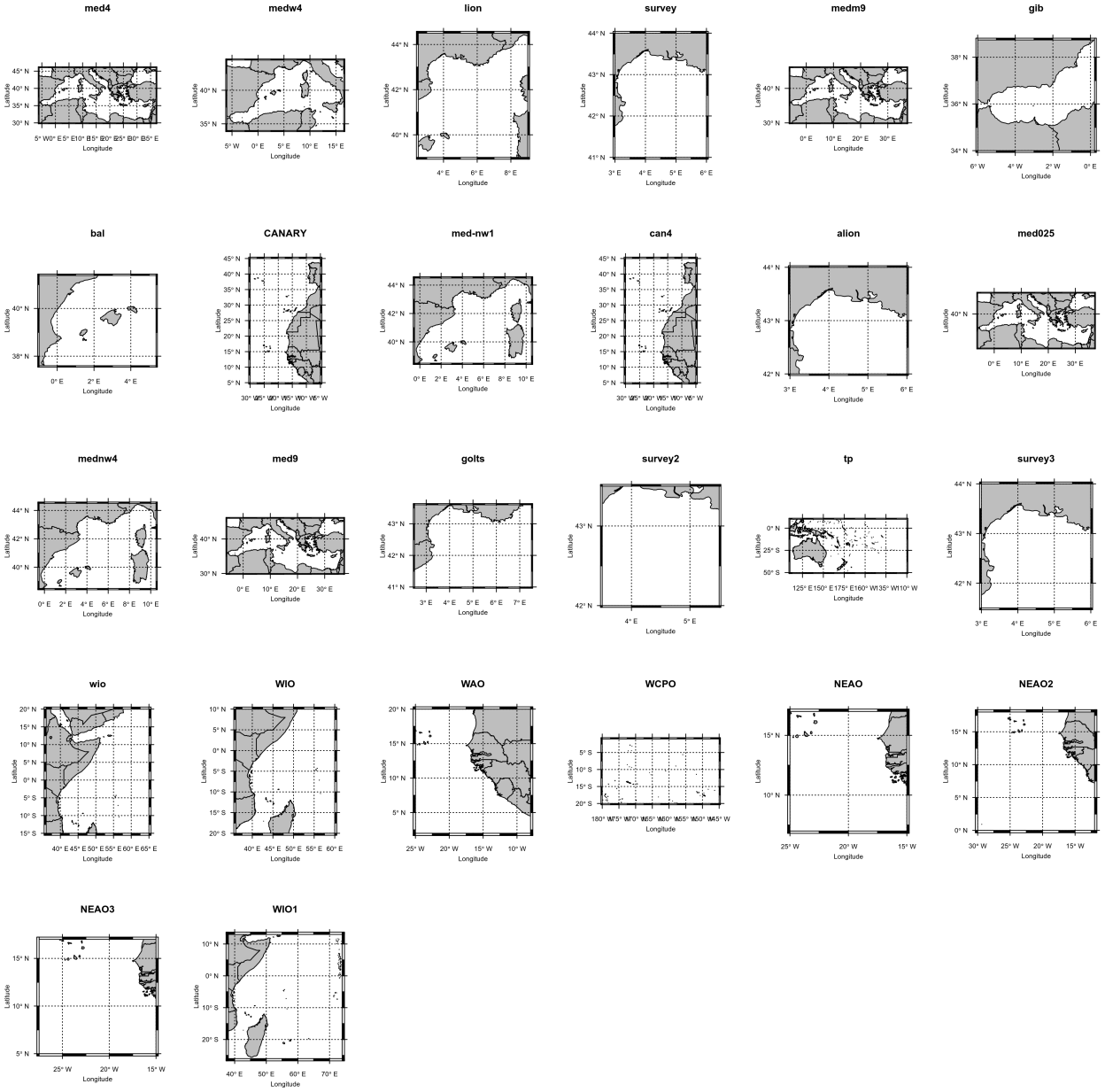


Figure 17. Sample images of pre-defined regions in ***plotmap***() and ***v***(). Checkout add.region() to define own region extents with default colorbar placement.

How to create and delete region-definitions with *add.region*() - part I

The function ***add.region***() can be used to set up new region-definitions (and -keywords), to **store** or **restore** a backup. ATTENTION: Backups are necessary since new region-definitions will be lost when updating ***oceanmap!*** In general, new regions can be supplied as a one-row data frame, e.g. by modifying existing region definition entries, or as a second option, through an interactive process. Here an example on how to modify existing definitions (the interactive way will be discussed on the next page):

```
## Example 1: Add region by supplying a one-row data.frame
##            that holds the entire required information
# data(region_definitions) # load region_definitions
# lion <- region_definitions[region_definitions$label == 'lion',] #
   selecting Gulf of Lions region
# lion
# junk <- lion
# junk$label <- 'junk' # rename region label
# add.region(junk) # add junk region
# data(region_definitions) # reload region_definitions
# region_definitions[,1:9]

## Example 2: Delete region
#delete.region("junk") # delete junk region
#data(region_definitions) # reload region_definitions
#region_definitions[,1:9]
```

How to create new region-definitions with *add.region*() - part II: the interactive way

Calling ***add.region***() without or insufficient arguments starts an interactive session that prompts the user to complete the new region definition. The single steps of this way listed below:

Adding a region definition: all interactive steps during an add()-region call

```
Please define the keyword of the new region, coded as 'label':

Please define the long name of the new region, coded as 'name':

Please define the northern most latitude
(negative values for the southern hemisphere) of the new region, coded as 'latn':

Please define the southern most latitude
(negative values for the southern hemisphere) of the new region, coded as 'lats':

Please define the western most longitude of the new region
(negative values for the western hemisphere), coded as 'lonw':

Please define the western most longitude of the new region
(negative values for the western hemisphere), coded as 'lone':

Please type 'm' if you want to perform colorbar placement by hand (mouse cursor)
or a letter (b, l, t, r) referring to a side of the plot (bottom, left, top, right).

If 'm' (manual placement) was selected:
Please select the lower left colorbar-position by the mouse cursor.
Please select the upper right colorbar-position by the mouse cursor.

You can resize the window to appropriate size. Try to avoid white space.
Press <Enter> when done.

Please enter the default grid resolution

Press <Enter> to save the new region definition
or any other key to abort the operation
```

Extra: Extracting data from spatial Raster-objects:

Extracting spatial data from ***Raster***-objects can be easily done with a bunch of functions implemented in the ***sp*** and ***maptools*** packages (in particular with the function ***extract***). Here some examples, for points, polygons and circles:

```
# bathy <- get.bathy('lion') # get bathymetry

#' extract values
# Points <- locator()
Points <- cbind(lon=c(6,8),lat=c(40,40))
points(Points,pch=19,col=c('black','red'))
extract(bathy, Points) # extract points

#' polygon
inst.pkg('rgeos')
cds1 <- rbind(c(3,42), c(3,43), c(7,43), c(3,42)) # polygon need to be
    closed (first point = last point)
poly <- SpatialPolygons(list(Polygons(list(Polygon(cds1)), 1)))
plot(poly[1],add=T,lwd=2,border="blue")
extract(bathy, poly[1])

#
poly2 <- gBuffer(poly[1], width = 10/60) # increase polygon by 10 nm
plot(poly2,add=T,lwd=2,border="blue",lty='dotted')

# source("SpatialCircle.r")
library(maptools)
circ <- SpatialCircle(Points[1,1],Points[1,2],r = 1)
circ_poly <- PolySet2SpatialPolygons(SpatialLines2PolySet(circ)) #
   convert circle from spatialline to spatialpolygon
plot(circ_poly, add=T)
extract(bathy, circ_poly)
```

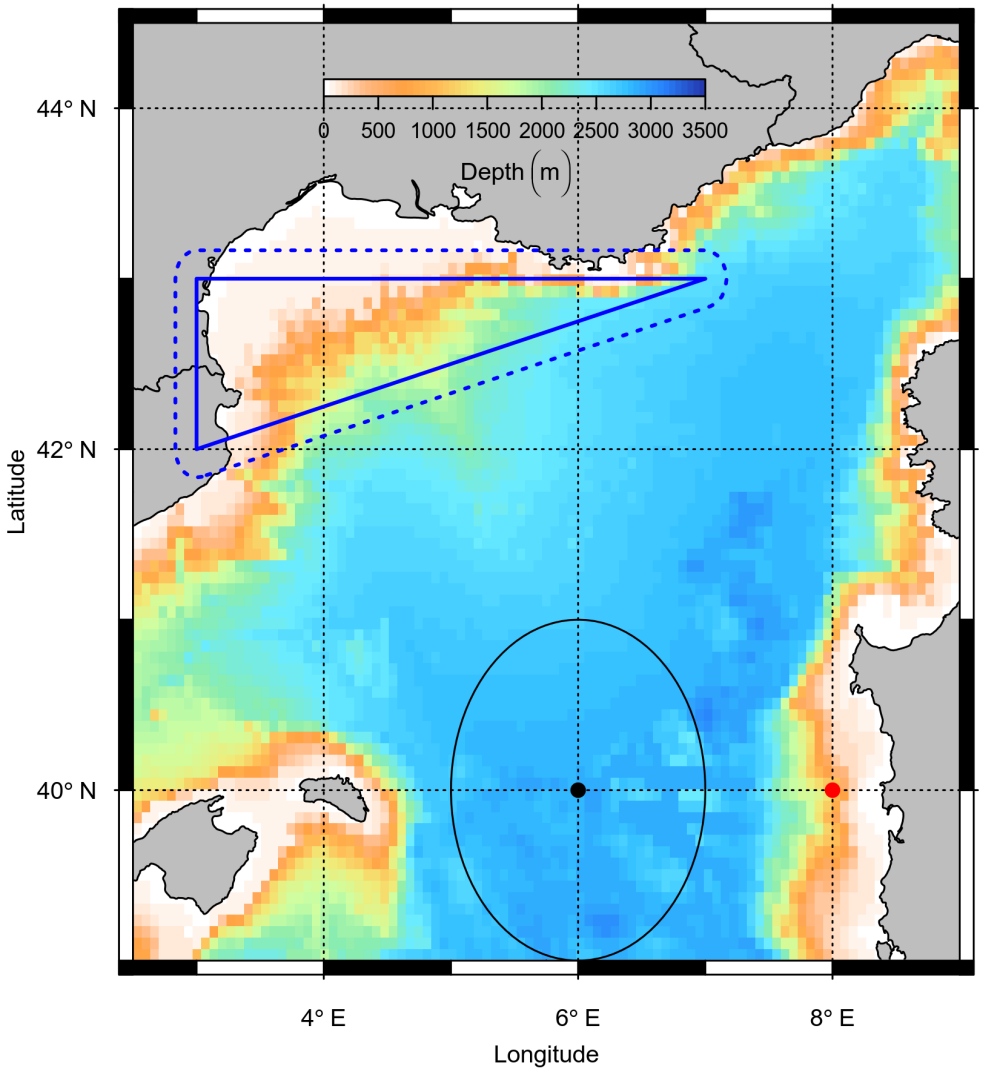


Figure 18. Extracing data from points, polygons and circles