Package 'CCAMLRGIS'

September 8, 2021

Type Package

Title Antarctic Spatial Data Manipulation

Version 3.2.0

Date 2021-09-08

Description Loads and creates spatial data, including layers and tools that are relevant to the activities of the Commission for the Conservation of Antarctic Marine Living Resources. Provides two categories of functions: load functions and create functions. Load functions are used to import existing spatial layers from the online CCAMLR GIS such as the ASD boundaries. Create functions are used to create layers from user data such as polygons and grids.

Depends R (>= 3.6), sp

License GPL-3

URL https://github.com/ccamlr/CCAMLRGIS

Encoding UTF-8

LazyData true

Imports dplyr, raster, methods, rgdal, rgeos, geosphere, graphics, grDevices, magrittr

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation no

Author Stephane Thanassekos [aut, cre],
Keith Reid [aut],
Lucy Robinson [aut],
Michael D. Sumner [ctb],
Roger Bivand [ctb]

Maintainer Stephane Thanassekos <stephane.thanassekos@ccamlr.org>

Repository CRAN

Date/Publication 2021-09-08 06:40:02 UTC

R topics documented:

add_col *Add colors*

Description

Given an input variable, generates either a continuous color gradient or color classes. To be used in conjunction with add_Cscale.

Usage

```
add_col(var, cuts = 100, cols = c("green", "yellow", "red"))
```

Arguments

var	numeric vector of the variable to be colorized. Either all values (in which case all values will be assigned to a color) or only two values (in which case these are considered to be the range of values).
cuts	numeric, controls color classes. Either one value (in which case cuts equally spaced color classes are generated) or a vector (in which cases irregular color classes are generated e.g. c(-10,0,100,2000)).
cols	character vector of colors (see R standard color names here). cols are interpolated along cuts. Color codes as such generated, for example, by rgb may also be used.

Value

list containing the colors for the variable var (given as $varcol in the output) as well as the single cols and cuts, to be used as inputs in add_Cscale.

See Also

add_Cscale, create_PolyGrids, R colors.

Examples

```
#Example 1: add color to points

MyPoints=create_Points(PointData)
MyCols=add_col(MyPoints$Nfishes) #With default cols and cuts
plot(MyPoints,pch=21,bg=MyCols$varcol,cex=2)

MyCols=add_col(MyPoints$Nfishes,cols=c('blue','red')) #With custom colors - here from blue to red
plot(MyPoints,pch=21,bg=MyCols$varcol,cex=2)

MyCols=add_col(MyPoints$Nfishes,cols=c('blue','red'),cuts=3) #With custom colors and fewer classes
plot(MyPoints,pch=21,bg=MyCols$varcol,cex=2)
```

```
#Example 2: add color to a grid, using custom cuts and colors, and adding a color scale (add_Cscale)

#Step 1: Generate your grid
MyGrid=create_PolyGrids(GridData,dlon=2,dlat=1)

#Step 2: Inspect your grid data (e.g. sum of Catch) to determine whether irregular cuts are required
hist(MyGrid$Catch_sum,100)
#In this case (heterogeneously distributed data) irregular cuts would be preferable

#Step 3: Generate colors according to the desired classes (cuts)
Gridcol=add_col(MyGrid$Catch_sum,cuts=c(0,50,100,500,2000,3500),cols=c('yellow','purple'))

#Step 4: Plot result and add color scale
Mypar=par(mai=c(0,0,0,2)) #Figure margins as c(bottom, left, top, right)
plot(MyGrid,col=Gridcol$varcol) #Use the colors generated by add_col
#Add color scale using cuts and cols generated by add_col
add_Cscale(title='Sum of Catch (t)',cuts=Gridcol$cuts,cols=Gridcol$cols,width=33)
par(Mypar)
```

`add_Cscale` *Add a color scale*

Description

Adds a color scale to plots. Default behavior set for bathymetry. May also be used to place a legend.

Usage

```
add_Cscale(
  pos = "1/1",
  title = "Depth (m)",
  width = 18,
  height = 70,
  cuts = Depth_cuts,
  cols = Depth_cols,
  minVal = NA,
  maxVal = NA,
  fontsize = 1,
  offset = 100,
  lwd = 1,
  mode = "Cscale"
)
```

Arguments

`pos`	character, indicating the vertical position of the color scale (which is always on the right side of plots). if `pos="1/1"`, the color scale will be centered.

	if pos="1/2", the color scale will in the top half of the plotting regions. if pos="2/2", the color scale will in the bottom half of the plotting regions.
title	character, title of the color scale.
width	width of the color scale box, expressed in % of the width of the plotting region.
height	height of the color scale box, expressed in % of the height of the plotting region.
cuts	numeric vector of color classes. May be generated via add_col.
cols	character vector of color names. May be generated via add_col.
minVal	If desired, the color scale may be generated starting from the value minVal. See examples.
maxVal	If desired, the color scale may be generated up to the value maxVal. See examples.
fontsize	Size of the text in the color scale.
offset	Controls the horizontal position of the color scale. Increase to distance from the plotting region.
lwd	thickness of lines.
mode	if 'Cscale', the default, the function builds a color scale. if 'Legend', the function gives you the location of a legend, arguments pos, offset and height may be used for adjustments. See examples.

See Also

load_Bathy, SmallBathy, Depth_cuts, Depth_cols, Depth_cuts2, Depth_cols2, add_col, R colors, legend.

Examples

```
#Example 1: simple bathymetry plot with color scale
Mypar=par(mai=c(0,0,0,1)) #plot margins as c(bottom, left, top, right)
plot(SmallBathy, breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(height=95)
par(Mypar)

#' #Example 2: simple bathymetry plot with Fishable Depth range highlight and color scale
Mypar=par(mai=c(0,0,0,1)) #plot margins as c(bottom, left, top, right)
plot(SmallBathy, breaks=Depth_cuts2, col=Depth_cols2, legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(height=95,cuts=Depth_cuts2,cols=Depth_cols2)
par(Mypar)

#Example 3: Show only values greater than 'minVal'
Mypar=par(mai=c(0,0,0,1)) #plot margins as c(bottom, left, top, right),
plot(SmallBathy, breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(minVal=-3200)
par(Mypar)

#Example 4: Show only values between 'minVal' and 'maxVal'
Mypar=par(mai=c(0,0,0,1)) #plot margins as c(bottom, left, top, right)
```

```
plot(SmallBathy, breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(minVal=-3200,maxVal=-400)
par(Mypar)

#Example 5: Adding two color scales

#Bathymetry
Mypar=par(mai=c(0,0,0,1)) #plot margins as c(bottom, left, top, right)

plot(SmallBathy,breaks=Depth_cuts,col=Depth_cols,legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(pos='1/2',height=45,maxVal=-1,minVal=-4000,fontsize=0.8)
#Some gridded data
MyGrid=create_PolyGrids(GridData,dlon=2,dlat=1)
Gridcol=add_col(MyGrid$Catch_sum,cuts=10)
plot(MyGrid,col=Gridcol$varcol,add=TRUE)
#Add color scale using cuts and cols generated by add_col, note the use of 'round'
add_Cscale(pos='2/2',height=45,title='Catch (t)',
           cuts=round(Gridcol$cuts,1),cols=Gridcol$cols,fontsize=0.8)
par(Mypar)

#Example 6: Adding a color scale and a legend

#Create some point data
MyPoints=create_Points(PointData)

#Crop the bathymetry to match the extent of MyPoints (extended extent)
BathyCr=raster::crop(SmallBathy,raster::extend(raster::extent(MyPoints),100000))
Mypar=par(mai=c(0,0,0,1)) #plot margins as c(bottom, left, top, right)
plot(BathyCr,breaks=Depth_cuts,col=Depth_cols,legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(pos='1/2',height=45,maxVal=-1,minVal=-4000,fontsize=0.8)

#Plot points with different symbols and colors (see ?points)
Psymbols=c(21,22,23,24)
Pcolors=c('red','green','blue','yellow')
plot(MyPoints[MyPoints$name=='one',],pch=Psymbols[1],bg=Pcolors[1],add=TRUE)
plot(MyPoints[MyPoints$name=='two',],pch=Psymbols[2],bg=Pcolors[2],add=TRUE)
plot(MyPoints[MyPoints$name=='three',],pch=Psymbols[3],bg=Pcolors[3],add=TRUE)
plot(MyPoints[MyPoints$name=='four',],pch=Psymbols[4],bg=Pcolors[4],add=TRUE)

#Add legend with position determined by add_Cscale
Loc=add_Cscale(pos='2/2',height=45,mode='Legend')
legend(Loc,legend=c('one','two','three','four'),title='Vessel',pch=Psymbols,pt.bg=Pcolors,xpd=TRUE)
par(Mypar)
```

`add_labels` *Add labels*

Description

Adds labels to plots. Three modes are available: In `'auto'` mode, labels are placed at the centres of polygon parts of spatial objects loaded via the `load_` functions. Internally used in conjunction with

Labels. In 'manual' mode, users may click on their plot to position labels. An editable label table is generated to allow fine-tuning of labels appearance, and may be saved for external use. To edit the label table, double-click inside one of its cells, edit the value, then close the table. In 'input' mode, a label table that was generated in 'manual' mode is re-used.

Usage

```
add_labels(
  mode = NULL,
  layer = NULL,
  fontsize = 1,
  fonttype = 1,
  angle = 0,
  col = "black",
  LabelTable = NULL
)
```

Arguments

mode	either 'auto', 'manual' or 'input'. See Description above.
layer	in 'auto' mode, single or vector of characters, may only be one, some or all of: c("ASDs","SSRUs","RBs","SSMUs","MAs","RefAreas","MPAs","EEZs").
fontsize	in 'auto' mode, size of the text.
fonttype	in 'auto' mode, type of the text (1 to 4), where 1 corresponds to plain text, 2 to bold face, 3 to italic and 4 to bold italic.
angle	in 'auto' mode, rotation of the text in degrees.
col	in 'auto' mode, color of the text.
LabelTable	in 'input' mode, a label table that was generated in 'manual' mode.

Value

Adds labels to plot. To save a label table generated in 'manual' mode, use: MyLabelTable=add_labels(mode='auto'). To re-use that label table, use: add_labels(mode='input',LabelTable=MyLabelTable).

See Also

Labels, load_ASDs, load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_EEZs, load_MPAs, R colors.

Examples

```
#Example 1: 'auto' mode
#label ASDs in bold and red
ASDs=load_ASDs()
plot(ASDs)
add_labels(mode='auto',layer='ASDs',fontsize=1,fonttype=2,col='red')
```

```
#add MPAs and EEZs and their labels in large, green and vertical text
MPAs=load_MPAs()
EEZs=load_EEZs()
plot(MPAs,add=TRUE,border='green')
plot(EEZs,add=TRUE,border='green')
add_labels(mode='auto',layer=c('EEZs','MPAs'),fontsize=2,col='green',angle=90)


#Example 2: 'manual' mode (you will have to do it yourself)
#Examples 2 and 3 below are commented (remove the # to test)
#plot(SmallBathy)
#ASDs=load_ASDs()
#plot(ASDs,add=TRUE)
#MyLabels=add_labels(mode='manual')


#Example 3: Re-use the label table generated in Example 2
#plot(SmallBathy)
#plot(ASDs,add=TRUE)
#add_labels(mode='input',LabelTable=MyLabels)
```

---

add_PieLegend *Add a legend to Pies*

Description

Adds a legend to pies created using create_Pies.

Usage

```
add_PieLegend(
  Pies = NULL,
  PosX = 0,
  PosY = 0,
  Size = 25,
  lwd = 1,
  Boxexp = c(0.2, 0.2, 0.12, 0.3),
  Boxbd = "white",
  Boxlwd = 1,
  Labexp = 0.3,
  fontsize = 1,
  LegSp = 0.5,
  Horiz = TRUE,
  PieTitle = "Pie chart",
```

```
  SizeTitle = "Size chart",
  PieTitleVadj = 0.5,
  SizeTitleVadj = 0.3,
  nSizes = 3,
  SizeClasses = NULL
)
```

Arguments

Pies	Spatial object created using create_Pies.
PosX	numeric, horizontal adjustment of legend.
PosY	numeric, vertical adjustment of legend.
Size	numeric value controlling the size of pies.
lwd	numeric value controlling the line thickness of pies.
Boxexp	numeric vector of length 4 controlling the expansion of the legend box, given as c(xmin,xmax,ymin,ymax).
Boxbd	color of the background of the legend box.
Boxlwd	numeric value controlling the line thickness of the legend box.
Labexp	numeric value controlling the distance of the pie labels to the center of the pie.
fontsize	Size of the legend font.
LegSp	Numeric, spacing between the pie and the size chart (only used if SizeVar was specified in create_Pies).
Horiz	Logical. Set to FALSE if vertical layout is desired (only used if SizeVar was specified in create_Pies).
PieTitle	Character, title of the pie chart.
SizeTitle	Character, title of the size chart (only used if SizeVar was specified in create_Pies).
PieTitleVadj	numeric value controlling the vertical adjustment of the title of the pie chart.
SizeTitleVadj	numeric value controlling the vertical adjustment of the title of the size chart (only used if SizeVar was specified in create_Pies).
nSizes	integer, number of size classes to display in the size chart. Minimum and maximum sizes are displayed by default. (only used if SizeVar was specified in create_Pies).
SizeClasses	numeric vector (e.g. c(1,10,100)) to select the size classes to display in the size chart (only used if SizeVar was specified in create_Pies). If set, overrides nSizes.

Value

Adds a legend to a pre-existing plot.

See Also

create_Pies, PieData, PieData2.

Examples

```
#Example 1. Pies of constant size, all classes displayed:
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=50
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.1,PosY=-1,Boxexp=c(0.5,0.45,0.12,0.45),
              PieTitle="Species")


#Example 2. Pies of constant size, selected classes displayed:
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=50,
                   Classes=c("TOP","TOA","ANI")
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.1,PosY=-1,Boxexp=c(0.6,0.6,0.12,0.55),
              PieTitle="Selected species")


#Example 3. Pies of constant size, proportions below 25% are grouped in a 'Other' class
#(N.B.: unlike Example 2, the 'Other' class may contain classes that are displayed in the legend.
#Please compare Example 1 and Example 3)
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=50,
                   Other=25
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.1,PosY=-1,Boxexp=c(0.55,0.55,0.12,0.45),
              PieTitle="Other (%) class")


#Example 4. Pies of variable size (here, their area is proportional to 'Catch'),
#all classes displayed, horizontal legend:
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=18,
                   SizeVar="Catch"
                   )
```

```
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.3,PosY=-0.8,Boxexp=c(0.16,0.1,0.1,0.4),
              PieTitle="Species",SizeTitle="Catch (t.)")


#Example 5. Pies of variable size (here, their area is proportional to 'Catch'),
#all classes displayed, vertical legend:
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=18,
                   SizeVar="Catch"
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-1.7,PosY=0.1,Boxexp=c(0.35,0.32,0.02,0.15),
              PieTitle="Species",SizeTitle="Catch (t.)",Horiz=FALSE,LegSp=0.6)


#Example 6. Pies of constant size, all classes displayed.
#Too many pies (see next example for solution):
#Create pies
MyPies=create_Pies(Input=PieData2,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=5
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.8,PosY=-0.1,Boxexp=c(0.5,0.45,0.12,0.45),
              PieTitle="Species")


#Example 7. Pies of constant size, all classes displayed. Gridded locations (in which case numerical
#variables in the 'Input' are summed for each grid point):
#Create pies
MyPies=create_Pies(Input=PieData2,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=5,
                   GridKm=250
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.8,PosY=-0.1,Boxexp=c(0.5,0.45,0.12,0.45),
              PieTitle="Species")


#Example 8. Pies of variable size (here, their area is proportional to 'Catch'),
#all classes displayed, vertical legend, gridded locations (in which case numerical
```

```
#variables in the 'Input' are summed for each grid point):
MyPies=create_Pies(Input=PieData2,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=3,
                   GridKm=250,
                   SizeVar='Catch'
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-1.2,PosY=0.3,Boxexp=c(0.38,0.32,0.08,0.18),
              PieTitle="Species",Horiz=FALSE,SizeTitle="Catch (t.)",
              SizeTitleVadj=0.8,nSizes=2)
```

`add_RefGrid` *Add a Reference grid*

Description

Add a Latitude/Longitude reference grid to maps.

Usage

```
add_RefGrid(
  bb,
  ResLat = 1,
  ResLon = 2,
  LabLon = NA,
  lwd = 1,
  fontsize = 1,
  offset = NA
)
```

Arguments

`bb`	bounding box of the first plotted object. for example, `bb=bbox(SmallBathy)` or `bb=bbox(MyPolys)`.
`ResLat`	Latitude resolution in decimal degrees.
`ResLon`	Longitude resolution in decimal degrees.
`LabLon`	Longitude at which Latitude labels should appear. if set, the resulting Reference grid will be circumpolar.
`lwd`	Line thickness of the Reference grid.
`fontsize`	Font size of the Reference grid's labels.
`offset`	offset of the Reference grid's labels (distance to plot border).

See Also

load_Bathy, SmallBathy.

Examples

```
#Example 1: Circumpolar grid with Latitude labels at Longitude 0

Mypar=par(mai=c(1,1.5,0.5,0)) #Figure margins as c(bottom, left, top, right)
par(Mypar)
plot(SmallBathy,breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_RefGrid(bb=bbox(SmallBathy),ResLat=10,ResLon=20,LabLon = 0)

#Example 2: Local grid around created polygons

MyPolys=create_Polys(PolyData,Densify=TRUE)
BathyC=raster::crop(SmallBathy,MyPolys) #crop the bathymetry to match the extent of MyPolys
Mypar=par(mai=c(0.5,0.5,0.5,0.5)) #Figure margins as c(bottom, left, top, right)
par(Mypar)
plot(BathyC,breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_RefGrid(bb=bbox(BathyC),ResLat=2,ResLon=6)
plot(MyPolys,add=TRUE,col='orange',border='brown',lwd=2)
```

---

assign_areas *Assign point locations to polygons*

Description

Given a set of polygons and a set of point locations (given in decimal degrees), finds in which polygon those locations fall. Finds, for example, in which ASD the given fishing locations occurred.

Usage

```
assign_areas(
  Input,
  Polys,
  AreaNameFormat = "GAR_Long_Label",
  Buffer = 0,
  NamesOut = NULL,
  NamesIn = NULL
)
```

Arguments

Input	dataframe containing - at the minimum - Latitudes and Longitudes to be assigned to polygons. **If** NamesIn **is not provided, the columns in the** Input **must be in the following order: Latitude, Longitude, Variable 1, Variable 2, ... Variable x.**.

Polys	character vector of spatial objects names (e.g., Polys=c('ASDs','RBs')). **Must be matching the names of the pre-loaded spatial objects (loaded via e.g.,** ASDs=load_ASDs()**)**
AreaNameFormat	dependent on the polygons loaded. For the Secretariat's spatial objects loaded via 'load_' functions, we have the following: 'GAR_Name' e.g., 'Subarea 88.2' 'GAR_Short_Label' e.g., '882' 'GAR_Long_Label' (default) e.g., '88.2' Several values may be entered if desired (if several Polys are used), e.g.: c('GAR_Short_Label','GAR_Name'), in which case AreaNameFormat must be given in the same order as Polys.
Buffer	distance in nautical miles to be added around the Polys of interest. Can be specified for each of the spatial objects named in Polys (e.g., Buffer=c(2,5)). Useful to determine whether locations are within Buffer nautical miles of a polygon.
NamesOut	names of the resulting area columns in the output dataframe, with order matching that of Polys (e.g., NamesOut=c('Recapture_ASD','Recapture_RB')). If not provided will be set as equal to Polys.
NamesIn	character vector of length 2 specifying the column names of Latitude and Longitude fields in the Input. **Latitudes name must be given first, e.g.:** NamesIn=c('MyLatitudes','MyLongitudes').

Value

dataframe with the same structure as the Input, with additional columns corresponding to the Polys used and named after NamesOut.

See Also

load_ASDs, load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_MPAs, load_EEZs.

Examples

```
#Generate a dataframe
MyData=data.frame(Lat=runif(100,min=-65,max=-50),
                  Lon=runif(100,min=20,max=40))

#Assign ASDs and SSRUs to these locations (first load ASDs and SSRUs)
ASDs=load_ASDs()
SSRUs=load_SSRUs()

MyData=assign_areas(Input=MyData,Polys=c('ASDs','SSRUs'),NamesOut=c('MyASDs','MySSRUs'))

#View(MyData)
table(MyData$MyASDs) #count of locations per ASD
```

```
table(MyData$MySSRUs) #count of locations per SSRU
```

CCAMLRGIS *Loads and creates spatial data, including layers and tools that are relevant to CCAMLR activities. All operations use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs).*

Description

This package provides two broad categories of functions: load functions and create functions.

Load functions

Load functions are used to import CCAMLR geo-referenced layers and include:

- load_ASDs
- load_SSRUs
- load_RBs
- load_SSMUs
- load_MAs
- load_Coastline
- load_MPAs
- load_EEZs
- load_Bathy

Create functions

Create functions are used to create geo-referenced layers from user-generated data and include:

- create_Points
- create_Lines
- create_Polys
- create_PolyGrids
- create_Stations
- create_Pies

Vignette

To learn more about CCAMLRGIS, start with the vignette: `browseVignettes(package = "CCAMLRGIS")`

See Also

The CCAMLRGIS package relies on several other package which users may want to familiarize themselves with, namely sp, raster, rgeos and rgdal.

`CCAMLRp` *CCAMLRGIS Projection*

Description

The CCAMLRGIS package uses the Lambert azimuthal equal-area projection. Source: `http://gis.ccamlr.org/`. In order to align with recent developments within Geographic Information Software, this projection will be accessed via EPSG code 6932 (see: epsg dot org).

Usage

```
data(CCAMLRp)
```

Format

Character string

Value

"+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs"

`Clip2Coast` *Clip Polygons to the Antarctic coastline*

Description

Clip Polygons to the Coast (removes polygon parts that fall on land) and computes the area of the resulting polygon. Uses a SpatialPolygon as input which may be user-generated or created via buffered points (see create_Points), buffered lines (see create_Lines) or polygons (see create_Polys).

Usage

```
Clip2Coast(Input)
```

Arguments

`Input`	Polygon(s) to be clipped.

Value

SpatialPolygon carrying the same data as the `Input`.

See Also

Coast, create_Points, create_Lines, create_Polys, create_PolyGrids.

Examples

```
#Example 1: Use Clip2Coast on a pre-generated polygon

MyPolys=create_Polys(PolyData,Densify=TRUE,Buffer=c(10,-15,120))
plot(MyPolys,col='red')
plot(Coast[Coast$ID=='All',],col='grey',add=TRUE)
MyPolysClipped=Clip2Coast(MyPolys)
plot(MyPolysClipped,col='blue',add=TRUE)
#View(MyPolysClipped)

#Example 2: Use Clip2Coast while creating a polygon, with Clip=TRUE in create_Polys().

MyPolysBefore=create_Polys(PolyData,Buffer=c(10,-15,120))
MyPolysAfter=create_Polys(PolyData,Buffer=c(10,-15,120),Clip=TRUE)
plot(MyPolysBefore,col='green')
plot(Coast[Coast$ID=='All',],add=TRUE)
plot(MyPolysAfter,col='red',add=TRUE)
```

Coast *Simplified and subsettable coastline*

Description

Coastline polygons generated from load_Coastline and sub-sampled to only contain data that falls within the CCAMLR boundaries. This spatial object may be subsetted to plot the coastline for selected ASDs or EEZs. Source: http://gis.ccamlr.org/

Usage

```
data(Coast)
```

Format

SpatialPolygonsDataFrame

See Also

Clip2Coast.

Examples

```
#Complete coastline:
plot(Coast[Coast$ID=='All',],col='grey')

#ASD 48.1 coastline:
plot(Coast[Coast$ID=='48.1',],col='grey')
```

`create_Lines` *Create Lines*

Description

Create Lines to display, for example, fishing line locations or tagging data.

Usage

```
create_Lines(
  Input,
  OutputFormat = "ROBJECT",
  OutputName = NULL,
  Buffer = 0,
  Densify = FALSE,
  Clip = FALSE,
  SeparateBuf = TRUE,
  NamesIn = NULL
)
```

Arguments

`Input`	the name of the `Input` data as a .csv file or an R dataframe. If a .csv file is used as input, this file must be in your working directory and its name given in quotes e.g. "DataFile.csv". **If `NamesIn` is not provided, the columns in the `Input` must be in the following order:** **Line name, Latitude, Longitude.** **If a given line is made of more than two points, the locations of points must be given in order, from one end of the line to the other.**
`OutputFormat`	can be an R object or an ESRI Shapefile. if `OutputFormat` is specified as "ROBJECT" (the default), a spatial object is created in your R environment. if `OutputFormat` is specified as "SHAPEFILE", an ESRI Shapefile is exported in your working directory.
`OutputName`	if `OutputFormat` is specified as "SHAPEFILE", the name of the output shapefile in quotes (e.g. "MyLines") must be provided.
`Buffer`	Distance in nautical miles by which to expand the lines. Can be specified for each line (as a numeric vector).

Densify	If set to TRUE, additional points between points of equal latitude are added prior to projection (see examples).
Clip	if set to TRUE, polygon parts (from buffered lines) that fall on land are removed (see Clip2Coast).
SeparateBuf	If set to FALSE when adding a Buffer, all spatial objects are merged, resulting in a single spatial object.
NamesIn	character vector of length 3 specifying the column names of line identifier, Latitude and Longitude fields in the Input. **Names must be given in that order, e.g.:** NamesIn=c('Line ID','Line Latitudes','Line Longitudes').

Value

Spatial object in your environment or ESRI shapefile in your working directory. Data within the resulting spatial object contains the data provided in the Input plus additional "LengthKm" and "LengthNm" columns which corresponds to the lines lengths, in kilometers and nautical miles respectively.

To see the data contained in your spatial object, type: View(MyLines@data).

See Also

create_Points, create_Polys, create_PolyGrids, create_Stations, add_RefGrid.

Examples

```
#If your data contains line end locations in separate columns, you may reformat it as follows:

#Example data:
MyData=data.frame(
  Line=c(1,2),
  Lat_Start=c(-60,-65),
  Lon_Start=c(-10,5),
  Lat_End=c(-61,-66),
  Lon_End=c(-2,2)
)

#Reformat to us as input as:
Input=data.frame(
  Line=c(MyData$Line,MyData$Line),
  Lat=c(MyData$Lat_Start,MyData$Lat_End),
  Lon=c(MyData$Lon_Start,MyData$Lon_End)
)

#Create lines and plot them
plot(create_Lines(Input=Input))
```

```
#Example 1: Simple and non-densified lines

MyLines=create_Lines(Input=LineData)
plot(MyLines,lwd=2,col=rainbow(length(MyLines)))

#Example 2: Simple and densified lines (note the curvature of the purple line)

MyLines=create_Lines(Input=LineData,Densify=TRUE)
plot(MyLines,lwd=2,col=rainbow(length(MyLines)))

#Example 3: Densified, buffered and clipped lines

MyLines=create_Lines(Input=LineData,Densify=TRUE,Buffer=c(10,40,50,80,100),Clip=TRUE)

plot(MyLines,lwd=2,col=rainbow(length(MyLines)))
plot(Coast[Coast$ID=='All',],col='grey',add=TRUE)

#Example 4: Buffered and grouped lines
MyLines=create_Lines(Input=LineData,Densify=TRUE,Buffer=30,SeparateBuf=FALSE)
plot(MyLines,lwd=2,border='blue')
```

`create_Pies` *Create Pies*

Description

Generates pie charts that can be overlaid on maps. The `Input` data must be a dataframe with, at least, columns for latitude, longitude, class and value. For each location, a pie is created with pieces for each class, and the size of each piece depends on the proportion of each class (the value of each class divided by the sum of values). Optionally, the area of each pie can be proportional to a chosen variable (if that variable is different than the value mentioned above, the `Input` data must have a fifth column and that variable must be unique to each location). If the `Input` data contains locations that are too close together, the data can be gridded by setting `GridKm` Once pie charts have been created, the function add_PieLegend may be used to add a legend to the figure.

Usage

```
create_Pies(
  Input,
  NamesIn = NULL,
  Classes = NULL,
  cols = c("green", "red"),
  Size = 50,
  SizeVar = NULL,
  GridKm = NULL,
  Other = 0,
```

```
  Othercol = "grey"
)
```

Arguments

Input	the name of the Input data as an R dataframe.
NamesIn	character vector of length 4 specifying the column names of Latitude, Longitude, Class and value fields in the Input. **Names must be given in that order, e.g.:** NamesIn=c('Latitude','Longitude','Class','Value').
Classes	optional character vector of classes to be displayed. If this excludes classes that are in the Input, those excluded classes will be polled in a 'Other' class.
cols	vector of two or more color names to colorize pie pieces.
Size	numeric value controlling the size of pies.
SizeVar	optional, name of the field in the Input that should be used to scale the area of pies. Must be unique to locations.
GridKm	optional, cell size of the grid in kilometers, if gridding is desired (in which case locations a pooled by grid cell and values are summed for each class).
Other	optional, percentage threshold below which classes are pooled in a 'Other' class.
Othercol	optional, color of the pie piece for the 'Other' class.

Value

Spatial object in your environment, ready to be plotted.

See Also

add_PieLegend, PieData, PieData2.

Examples

```
#Example 1. Pies of constant size, all classes displayed:
#Create pies
MyPies=create_Pies(Input=PieData,NamesIn=c("Lat","Lon","Sp","N"),Size=50)
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.1,PosY=-1,Boxexp=c(0.5,0.45,0.12,0.45),
            PieTitle="Species")


#Example 2. Pies of constant size, selected classes displayed:
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=50,
                   Classes=c("TOP","TOA","ANI")
```

```
                  )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.1,PosY=-1,Boxexp=c(0.6,0.6,0.12,0.55),
              PieTitle="Selected species")


#Example 3. Pies of constant size, proportions below 25% are grouped in a 'Other' class
#(N.B.: unlike Example 2, the 'Other' class may contain classes that are displayed in the legend.
#Please compare Example 1 and Example 3)
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=50,
                   Other=25
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.1,PosY=-1,Boxexp=c(0.55,0.55,0.12,0.45),
              PieTitle="Other (%) class")


#Example 4. Pies of variable size (here, their area is proportional to 'Catch'),
#all classes displayed, horizontal legend:
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=18,
                   SizeVar="Catch"
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.3,PosY=-0.8,Boxexp=c(0.16,0.1,0.1,0.4),
              PieTitle="Species",SizeTitle="Catch (t.)")


#Example 5. Pies of variable size (here, their area is proportional to 'Catch'),
#all classes displayed, vertical legend:
#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=18,
                   SizeVar="Catch"
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-1.7,PosY=0.1,Boxexp=c(0.35,0.32,0.02,0.15),
              PieTitle="Species",SizeTitle="Catch (t.)",Horiz=FALSE,LegSp=0.6)
```

```
#Example 6. Pies of constant size, all classes displayed.
#Too many pies (see next example for solution):
#Create pies
MyPies=create_Pies(Input=PieData2,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=5
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.8,PosY=-0.1,Boxexp=c(0.5,0.45,0.12,0.45),
              PieTitle="Species")


#Example 7. Pies of constant size, all classes displayed. Gridded locations (in which case numerical
#variables in the 'Input' are summed for each grid point):
#Create pies
MyPies=create_Pies(Input=PieData2,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=5,
                   GridKm=250
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.8,PosY=-0.1,Boxexp=c(0.5,0.45,0.12,0.45),
              PieTitle="Species")


#Example 8. Pies of variable size (here, their area is proportional to 'Catch'),
#all classes displayed, vertical legend, gridded locations (in which case numerical
#variables in the 'Input' are summed for each grid point):
MyPies=create_Pies(Input=PieData2,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=3,
                   GridKm=250,
                   SizeVar='Catch'
                   )
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-1.2,PosY=0.3,Boxexp=c(0.38,0.32,0.08,0.18),
              PieTitle="Species",Horiz=FALSE,SizeTitle="Catch (t.)",
              SizeTitleVadj=0.8,nSizes=2)
```

`create_Points` *Create Points*

Description

Create Points to display point locations. Buffering points may be used to produce bubble charts.

Usage

```
create_Points(
  Input,
  OutputFormat = "ROBJECT",
  OutputName = NULL,
  Buffer = 0,
  Clip = FALSE,
  SeparateBuf = TRUE,
  NamesIn = NULL
)
```

Arguments

`Input`	the name of the `Input` data as a .csv file or an R dataframe. If a .csv file is used as input, this file must be in your working directory and its name given in quotes e.g. "DataFile.csv". **If `NamesIn` is not provided, the columns in the `Input` must be in the following order:** **Latitude, Longitude, Variable 1, Variable 2, ... Variable x**
`OutputFormat`	can be an R object or an ESRI Shapefile. if `OutputFormat` is specified as "ROBJECT" (the default), a spatial object is created in your R environment. if `OutputFormat` is specified as "SHAPEFILE", an ESRI Shapefile is exported in your working directory.
`OutputName`	if `OutputFormat` is specified as "SHAPEFILE", the name of the output shapefile in quotes (e.g. "MyPoints") must be provided.
`Buffer`	Radius in nautical miles by which to expand the points. Can be specified for each point (as a numeric vector).
`Clip`	if set to TRUE, polygon parts (from buffered points) that fall on land are removed (see Clip2Coast).
`SeparateBuf`	If set to FALSE when adding a `Buffer`, all spatial objects are merged, resulting in a single spatial object.
`NamesIn`	character vector of length 2 specifying the column names of Latitude and Longitude fields in the `Input`. **Latitudes name must be given first, e.g.:** `NamesIn=c('MyLatitudes','MyLongitudes')`.

Value

Spatial object in your environment or ESRI shapefile in your working directory. Data within the resulting spatial object contains the data provided in the `Input` plus additional "x" and "y" columns which corresponds to the projected points locations and may be used to label points (see examples).

To see the data contained in your spatial object, type: `View(MyPoints@data)`.

See Also

create_Lines, create_Polys, create_PolyGrids, create_Stations, add_RefGrid.

Examples

```
#Example 1: Simple points with labels

MyPoints=create_Points(Input=PointData)
plot(MyPoints)
text(MyPoints$x,MyPoints$y,MyPoints$name,adj=c(0.5,-0.5),xpd=TRUE)

#Example 2: Simple points with labels, highlighting one group of points with the same name

MyPoints=create_Points(Input=PointData)
plot(MyPoints)
text(MyPoints$x,MyPoints$y,MyPoints$name,adj=c(0.5,-0.5),xpd=TRUE)
plot(MyPoints[MyPoints$name=='four',],bg='red',pch=21,cex=1.5,add=TRUE)

#Example 3: Buffered points with radius proportional to catch

MyPoints=create_Points(Input=PointData,Buffer=0.5*PointData$Catch)
plot(MyPoints,col='green')
text(MyPoints$x,MyPoints$y,MyPoints$name,adj=c(0.5,0.5),xpd=TRUE)

#Example 4: Buffered points with radius proportional to catch and clipped to the Coast

MyPoints=create_Points(Input=PointData,Buffer=2*PointData$Catch,Clip=TRUE)
plot(MyPoints,col='cyan')
plot(Coast[Coast$ID=='All',],add=TRUE,col='grey')
```

---

create_PolyGrids *Create a Polygon Grid*

Description

Create a polygon grid to spatially aggregate data in cells of chosen size. Cell size may be specified in degrees or as a desired area in square kilometers (in which case cells are of equal area).

Usage

```
create_PolyGrids(
  Input,
```

```
  OutputFormat = "ROBJECT",
  OutputName = NULL,
  dlon = NA,
  dlat = NA,
  Area = NA,
  cuts = 100,
  cols = c("green", "yellow", "red"),
  NamesIn = NULL
)
```

Arguments

Input	the name of the Input data as a .csv file or an R dataframe. If a .csv file is used as input, this file must be in your working directory and its name given in quotes e.g. "DataFile.csv". **If NamesIn is not provided, the columns in the Input must be in the following order:** **Latitude, Longitude, Variable 1, Variable 2 ... Variable x.**
OutputFormat	can be an R object or an ESRI Shapefile. if OutputFormat is specified as "ROBJECT" (the default), a SpatialPolygonDataFrame is created in your R environment. if OutputFormat is specified as "SHAPEFILE", an ESRI Shapefile is exported in your working directory.
OutputName	if OutputFormat is specified as "SHAPEFILE", the name of the output shapefile in quotes (e.g. "MyGrid") must be provided.
dlon	width of the grid cells in decimal degrees of longitude.
dlat	height of the grid cells in decimal degrees of latitude.
Area	area, in square kilometers, of the grid cells.
cuts	Number of desired color classes.
cols	Desired colors. If more that one color is provided, a linear color gradient is generated.
NamesIn	character vector of length 2 specifying the column names of Latitude and Longitude fields in the Input. **Latitudes name must be given first, e.g.:** NamesIn=c('MyLatitudes','MyLongitudes').

Value

Spatial object in your environment or ESRI shapefile in your working directory. Data within the resulting spatial object contains the data provided in the Input after aggregation within cells. For each Variable, the minimum, maximum, mean, sum, count, standard deviation, and, median of values in each cell is returned. In addition, for each cell, its area (AreaKm2), projected centroid (Centrex, Centrey) and unprojected centroid (Centrelon, Centrelat) is given.

To see the data contained in your spatial object, type: View(MyGrid@data).

Finally, colors are generated for each aggregated values according to the chosen cuts (numerical classes) and cols (colors).

To generate a custom color scale after the grid creation, refer to add_col and add_Cscale. See Example 4 below.

See Also

create_Points, create_Lines, create_Polys, create_Stations, add_RefGrid, add_col, add_Cscale.

Examples

```
#Example 1: Simple grid, using automatic colors

MyGrid=create_PolyGrids(Input=GridData,dlon=2,dlat=1)
#View(MyGrid@data)
plot(MyGrid,col=MyGrid$Col_Catch_sum)

#Example 2: Equal area grid, using automatic colors

MyGrid=create_PolyGrids(Input=GridData,Area=10000)
plot(MyGrid,col=MyGrid$Col_Catch_sum)

#Example 3: Equal area grid, using custom cuts and colors

MyGrid=create_PolyGrids(Input=GridData,
       Area=10000,cuts=c(0,50,100,500,2000,3500),cols=c('blue','red'))

plot(MyGrid,col=MyGrid$Col_Catch_sum)

#Example 4: Equal area grid, using custom cuts and colors, and adding a color scale (add_Cscale)

#Step 1: Generate your grid
MyGrid=create_PolyGrids(Input=GridData,Area=10000)

#Step 2: Inspect your gridded data (e.g. sum of Catch) to
#determine whether irregular cuts are required
hist(MyGrid$Catch_sum,100)
#In this case (heterogeneously distributed data) irregular cuts would be preferable

#Step 3: Generate colors according to the desired classes (cuts)
Gridcol=add_col(MyGrid$Catch_sum,cuts=c(0,50,100,500,2000,3500),cols=c('yellow','purple'))

#Step 4: Plot result and add color scale
Mypar=par(mai=c(0,0,0,2)) #Figure margins as c(bottom, left, top, right)
plot(MyGrid,col=Gridcol$varcol) #Use the colors generated by add_col
#Add color scale using cuts and cols generated by add_col
add_Cscale(title='Sum of Catch (t)',cuts=Gridcol$cuts,cols=Gridcol$cols,width=24)
par(Mypar)
```

`create_Polys` *Create Polygons*

Description

Create Polygons such as proposed Research Blocks or Marine Protected Areas.

Usage

```
create_Polys(
  Input,
  OutputFormat = "ROBJECT",
  OutputName = NULL,
  Buffer = 0,
  Densify = TRUE,
  Clip = FALSE,
  SeparateBuf = TRUE,
  NamesIn = NULL
)
```

Arguments

`Input`	the name of the `Input` data as a .csv file or an R dataframe. If a .csv file is used as input, this file must be in your working directory and its name given in quotes e.g. "DataFile.csv". **If `NamesIn` is not provided, the columns in the `Input` must be in the following order:** **Polygon name, Latitude, Longitude.** **Latitudes and Longitudes must be given clockwise.**
`OutputFormat`	can be an R object or an ESRI Shapefile. if `OutputFormat` is specified as "ROBJECT" (the default), a SpatialPolygonDataFrame is created in your R environment. if `OutputFormat` is specified as "SHAPEFILE", an ESRI Shapefile is exported in your working directory.
`OutputName`	if `OutputFormat` is specified as "SHAPEFILE", the name of the output shapefile in quotes (e.g. "MyPolygons") must be provided.
`Buffer`	Distance in nautical miles by which to expand the polygons. Can be specified for each polygon (as a numeric vector).
`Densify`	If set to TRUE, additional points between points of equal latitude are added prior to projection (see examples).
`Clip`	if set to TRUE, polygon parts that fall on land are removed (see Clip2Coast).
`SeparateBuf`	If set to FALSE when adding a `Buffer`, all spatial objects are merged, resulting in a single spatial object.
`NamesIn`	character vector of length 3 specifying the column names of polygon identifier, Latitude and Longitude fields in the `Input`. **Names must be given in that order, e.g.:** `NamesIn=c('Polygon ID','Poly Latitudes','Poly Longitudes')`.

Value

Spatial object in your environment or ESRI shapefile in your working directory. Data within the resulting spatial object contains the data provided in the `Input` plus an additional "AreaKm2" column which corresponds to the areas, in square kilometers, of your polygons. Also, columns "Labx" and "Laby" which may be used to add labels to polygons.

To see the data contained in your spatial object, type: `View(MyPolygons@data)`.

See Also

`create_Points`, `create_Lines`, `create_PolyGrids`, `create_Stations`, `add_RefGrid`.

Examples

```
#Example 1: Simple and non-densified polygons

MyPolys=create_Polys(Input=PolyData,Densify=FALSE)
plot(MyPolys,col='blue')
text(MyPolys$Labx,MyPolys$Laby,MyPolys$ID,col='white')

#Example 2: Simple and densified polygons (note the curvature of iso-latitude lines)

MyPolys=create_Polys(Input=PolyData)
plot(MyPolys,col='red')
text(MyPolys$Labx,MyPolys$Laby,MyPolys$ID,col='white')

#Example 3: Buffered and clipped polygons

MyPolysBefore=create_Polys(Input=PolyData,Buffer=c(10,-15,120))
MyPolysAfter=create_Polys(Input=PolyData,Buffer=c(10,-15,120),Clip=TRUE)
plot(MyPolysBefore,col='green')
plot(Coast[Coast$ID=='All',],add=TRUE)
plot(MyPolysAfter,col='red',add=TRUE)
text(MyPolysAfter$Labx,MyPolysAfter$Laby,MyPolysAfter$ID,col='white')

#Example 4: Buffered and grouped polygons
MyPolys=create_Polys(Input=PolyData,Buffer=80,SeparateBuf=FALSE)
plot(MyPolys,border='blue',lwd=3)
```

`create_Stations` *Create Stations*

Description

Create random point locations inside a polygon and within bathymetry strata constraints. A distance constraint between stations may also be used if desired.

Usage

```
create_Stations(
  Poly,
  Bathy,
  Depths,
  N = NA,
  Nauto = NA,
  dist = NA,
  Buf = 1000,
  ShowProgress = FALSE
)
```

Arguments

Poly	single polygon inside which stations will be generated. May be created using create_Polys.
Bathy	bathymetry raster with the appropriate projection, such as this one.
Depths	vector of depths. For example, if the depth strata required are 600 to 1000 and 1000 to 2000, Depths=c(-600,-1000,-2000).
N	vector of number of stations required in each depth strata, therefore length(N) must equal length(Depths)-1.
Nauto	instead of specifying N, a number of stations proportional to the areas of the depth strata may be created. Nauto is the maximum number of stations required in any depth stratum.
dist	if desired, a distance constraint in nautical miles may be applied. For example, if dist=2, stations will be at least 2 nautical miles apart.
Buf	distance in meters from isobaths. Useful to avoid stations falling on strata boundaries.
ShowProgress	if set to TRUE, a progress bar is shown (create_Stations may take a while).

Value

Spatial object in your environment. Data within the resulting object contains the strata and stations locations in both projected space ("x" and "y") and degrees of Latitude/Longitude.

To see the data contained in your spatial object, type: View(MyStations@data).

See Also

create_Polys, SmallBathy, add_RefGrid.

Examples

```
#First, create a polygon within which stations will be created

MyPolys=create_Polys(Input=PolyData,Densify=TRUE)
plot(MyPolys)
text(MyPolys$Labx,MyPolys$Laby,MyPolys$ID)
#Subsample MyPolys to only keep the polygon with ID 'one'
MyPoly=MyPolys[MyPolys$ID=='one',]
plot(MyPoly,col='green',add=TRUE)

#Second (optional), crop your bathymetry raster to match the extent of your polygon

BathyCroped=raster::crop(SmallBathy,MyPoly)


#Example 1: Set numbers of stations, no distance constraint

MyStations=create_Stations(Poly=MyPoly,
           Bathy=BathyCroped,Depths=c(-550,-1000,-1500,-2000),N=c(20,15,10))
Mypar=par(mai=c(0,0,0,2)) #Figure margins as c(bottom, left, top, right)
plot(BathyCroped,breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(offset = 50,height = 90,fontsize = 0.8,width=25)
plot(MyPoly,add=TRUE,border='red',lwd=2)
raster::contour(BathyCroped,levels=c(-550,-1000,-1500,-2000),add=TRUE)
plot(MyStations,add=TRUE,col='orange')
par(Mypar)

#Example 2: Set numbers of stations, with distance constraint

MyStations=create_Stations(Poly=MyPoly,
           Bathy=BathyCroped,Depths=c(-550,-1000,-1500,-2000),N=c(20,15,10),dist=10)
Mypar=par(mai=c(0,0,0,2)) #Figure margins as c(bottom, left, top, right)
plot(BathyCroped,breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(offset = 50,height = 90,fontsize = 0.8,width=25)
plot(MyPoly,add=TRUE,border='red',lwd=2)
raster::contour(BathyCroped,levels=c(-550,-1000,-1500,-2000),add=TRUE)
plot(MyStations[MyStations$Stratum=='550-1000',],pch=21,bg='yellow',add=TRUE)
plot(MyStations[MyStations$Stratum=='1000-1500',],pch=21,bg='orange',add=TRUE)
plot(MyStations[MyStations$Stratum=='1500-2000',],pch=21,bg='red',add=TRUE)
par(Mypar)

#Example 3: Automatic numbers of stations, with distance constraint

MyStations=create_Stations(Poly=MyPoly,
           Bathy=BathyCroped,Depths=c(-550,-1000,-1500,-2000),Nauto=30,dist=10)
Mypar=par(mai=c(0,0,0,2)) #Figure margins as c(bottom, left, top, right)
plot(BathyCroped,breaks=Depth_cuts, col=Depth_cols, legend=FALSE,axes=FALSE,box=FALSE)
add_Cscale(offset = 50,height = 90,fontsize = 0.8,width=25)
plot(MyPoly,add=TRUE,border='red',lwd=2)
```

```
raster::contour(BathyCroped,levels=c(-550,-1000,-1500,-2000),add=TRUE)
plot(MyStations[MyStations$Stratum=='550-1000',],pch=21,bg='yellow',add=TRUE)
plot(MyStations[MyStations$Stratum=='1000-1500',],pch=21,bg='orange',add=TRUE)
plot(MyStations[MyStations$Stratum=='1500-2000',],pch=21,bg='red',add=TRUE)
par(Mypar)
```

Depth_cols *Bathymetry colors*

Description

Set of standard colors to plot bathymetry, to be used in conjunction with Depth_cuts.

Usage

```
data(Depth_cols)
```

Format

Character vector

See Also

Depth_cols2, add_col, add_Cscale, SmallBathy.

Examples

```
plot(SmallBathy,breaks=Depth_cuts,col=Depth_cols,axes=FALSE,box=FALSE)
```

Depth_cols2 *Bathymetry colors with Fishable Depth range*

Description

Set of colors to plot bathymetry and highlight Fishable Depth range, to be used in conjunction with Depth_cuts2.

Usage

```
data(Depth_cols2)
```

Format

Character vector

See Also

Depth_cols, add_col, add_Cscale, SmallBathy.

Examples

```
plot(SmallBathy,breaks=Depth_cuts2,col=Depth_cols2,axes=FALSE,box=FALSE)
```

Depth_cuts *Bathymetry depth classes*

Description

Set of depth classes to plot bathymetry, to be used in conjunction with Depth_cols.

Usage

```
data(Depth_cuts)
```

Format

Numeric vector

See Also

Depth_cuts2, add_col, add_Cscale, SmallBathy.

Examples

```
plot(SmallBathy,breaks=Depth_cuts,col=Depth_cols,axes=FALSE,box=FALSE)
```

Depth_cuts2 *Bathymetry depth classes with Fishable Depth range*

Description

Set of depth classes to plot bathymetry and highlight Fishable Depth range, to be used in conjunction with Depth_cols2.

Usage

```
data(Depth_cuts2)
```

Format

Numeric vector

See Also

Depth_cuts, add_col, add_Cscale, SmallBathy.

Examples

```
plot(SmallBathy,breaks=Depth_cuts2,col=Depth_cols2,axes=FALSE,box=FALSE)
```

---

get_depths *Get depths of locations from bathymetry raster*

Description

Given a bathymetry raster and an input dataframe of point locations (given in decimal degrees), computes the depths at these locations using bilinear interpolation (using extract). Optionally can also compute the horizontal distance of locations to chosen isobaths.

Usage

```
get_depths(
  Input,
  Bathy,
  d = 10000,
  Isobaths = NA,
  IsoLocs = FALSE,
  ShowProgress = FALSE,
  NamesIn = NULL
)
```

Arguments

Input	dataframe with, at least, Latitudes and Longitudes. **If** NamesIn **is not provided, the columns in the** Input **must be in the following order: Latitude, Longitude, Variable 1, Variable 2, ... Variable x**
Bathy	bathymetry raster with the appropriate projection, such as this one. It is recommended to use a raster of higher resolution than SmallBathy (see load_Bathy).
d	distance in meters, used to group locations by distance and speed up computations (by cutting the bathymetry raster into small pieces matching the extent of grouped locations). Lower values make computations faster but at the risk of not finding distances to isobaths (when desired).
Isobaths	Depths to which the horizontal distances to locations are computed, if required.
IsoLocs	If TRUE, the locations on the Iosbaths that are closest to the locations given in the Input are added to the exported dataframe.
ShowProgress	if set to TRUE, a progress bar is shown (get_depths may take a while).
NamesIn	character vector of length 2 specifying the column names of Latitude and Longitude fields in the Input. **Latitudes name must be given first, e.g.:** NamesIn=c('MyLatitudes','MyLongitudes').

Value

dataframe with the same structure as the Input with additional columns, where 'x' and 'y' are the projected locations, 'd' is the depth, 'D_iso', 'X_iso' and 'Y_iso' are the horizontal distances and closest point location on 'Isobaths'. All units are in meters.

See Also

load_Bathy, SmallBathy, create_Points, create_Stations, extract.

Examples

```
#Generate a dataframe
MyData=data.frame(Lat=PointData$Lat,
Lon=PointData$Lon,
Catch=PointData$Catch)

#Example 1: get depths of locations
MyDataD=get_depths(Input=MyData,Bathy=SmallBathy)
#View(MyDataD)
plot(MyDataD$d,MyDataD$Catch,xlab='Depth',ylab='Catch',pch=21,bg='blue') #Plot of catch vs depth

#Example 2: get depths of locations and distance to isobath -3000m

MyDataD=get_depths(Input=MyData,Bathy=SmallBathy,
        Isobaths=-3000,IsoLocs=TRUE,d=200000,ShowProgress=TRUE)
plot(MyDataD$x,MyDataD$y,pch=21,bg='green')
raster::contour(SmallBathy,levels=-3000,add=TRUE,col='blue',maxpixels=10000000)
segments(x0=MyDataD$x,
         y0=MyDataD$y,
         x1=MyDataD$X_3000,
         y1=MyDataD$Y_3000,col='red')
```

GridData *Example dataset for create_PolyGrids*

Description

To be used in conjunction with create_PolyGrids.

Usage

```
data(GridData)
```

Format

DataFrame

See Also

create_PolyGrids.

Examples

```
#View(GridData)

MyGrid=create_PolyGrids(Input=GridData,dlon=2,dlat=1)
plot(MyGrid,col=MyGrid$Col_Catch_sum)
```

Labels *Polygon labels*

Description

Labels for the layers obtained via 'load_' functions. Positions correspond to the centroids of polygon parts. Can be used in conjunction with add_labels.

Usage

```
data(Labels)
```

Format

dataframe

See Also

add_labels, load_ASDs, load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_EEZs, load_MPAs.

Examples

```
#View(Labels)

ASDs=load_ASDs()
plot(ASDs)
add_labels(mode='auto',layer='ASDs',fontsize=1,fonttype=2)
```

LineData — *Example dataset for create_Lines*

Description

To be used in conjunction with create_Lines.

Usage

```
data(LineData)
```

Format

DataFrame

See Also

create_Lines.

Examples

```
#View(LineData)

MyLines=create_Lines(LineData)
plot(MyLines,lwd=2)
```

load_ASDs — *Load CCAMLR statistical Areas, Subareas and Divisions*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs)

Usage

```
load_ASDs()
```

See Also

load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_Coastline, load_MPAs, load_EEZs.

Examples

```
#When online:
ASDs=load_ASDs()
plot(ASDs)

#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
ASDs=load_ASDs()
EEZs=load_EEZs()
save(list=c('ASDs','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(ASDs,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

`load_Bathy` *Load Bathymetry data*

Description

Download the up-to-date projected GEBCO data from the online CCAMLRGIS and load it to your environment. This functions can be used in two steps, to first download the data and then use it. If you keep the downloaded data, you can then re-use it in all your scripts.

Usage

```
load_Bathy(LocalFile, Res = 5000)
```

Arguments

`LocalFile`	To download the data, set to `FALSE`. To re-use a downloaded file, set to the full path of the file (e.g., "C:/Desktop/GEBCO2020_5000.tif").
`Res`	Desired resolution in meters. May only be one of: 500, 1000, 2500 or 5000.

Details

To download the data, you must either have set your working directory using `setwd`, or be working within an Rproject. In any case, your file will be downloaded to the folder path given by `getwd`.

It is strongly recommended to first download the lowest resolution data (set `Res=5000`) to ensure it is working as expected.

To re-use the downloaded data, you must provide the full path to that file, for example:

"C:/Desktop/GEBCO2020_5000.tif".

This data was reprojected from the original GEBCO Grid after cropping at 40 degrees South. Projection was made using the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea

+lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs), and the data was aggregated at a several resolutions.

Value

Bathymetry raster.

References

GEBCO Compilation Group (2020) GEBCO 2020 Grid (doi:10.5285/a29c5465-b138-234d-e053-6c86abc040b9)

See Also

add_col, add_Cscale, Depth_cols, Depth_cuts, Depth_cols2, Depth_cuts2, get_depths, create_Stations, SmallBathy, load_SSRUs, load_RBs, load_ASDs, load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_Coastline, load_MPAs, load_EEZs.

Examples

```
#The examples below are commented. To test, remove the '#'.

##Download the data. It will go in the folder given by getwd():
#Bathy=load_Bathy(LocalFile = FALSE,Res=5000)
#plot(Bathy, breaks=Depth_cuts,col=Depth_cols,axes=FALSE,box=FALSE,legend=F)

##Re-use the downloaded data (provided it's here: "C:/Desktop/GEBCO2020_5000.tif"):
#Bathy=load_Bathy(LocalFile = "C:/Desktop/GEBCO2020_5000.tif")
#plot(Bathy, breaks=Depth_cuts,col=Depth_cols,axes=FALSE,box=FALSE,legend=F)
```

---

load_Coastline *Load the full CCAMLR Coastline*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs) Note that this coastline expands further north than Coast.

Usage

```
load_Coastline()
```

See Also

load_ASDs, load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_MPAs, load_EEZs.

Examples

```
#When online:
Coastline=load_Coastline()
plot(Coastline)

#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
Coastline=load_Coastline()
EEZs=load_EEZs()
save(list=c('Coastline','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(Coastline,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

---

load_EEZs *Load Exclusive Economic Zones*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs)

Usage

```
load_EEZs()
```

See Also

load_ASDs, load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_Coastline, load_MPAs.

Examples

```
#When online:
EEZs=load_EEZs()
plot(EEZs)
```

```
#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
MPAs=load_MPAs()
EEZs=load_EEZs()
save(list=c('MPAs','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(MPAs,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

`load_MAs` *Load CCAMLR Management Areas*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs)

Usage

```
load_MAs()
```

See Also

`load_ASDs`, `load_SSRUs`, `load_RBs`, `load_SSMUs`, `load_Coastline`, `load_MPAs`, `load_EEZs`.

Examples

```
#When online:
MAs=load_MAs()
plot(MAs)

#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
MAs=load_MAs()
EEZs=load_EEZs()
save(list=c('MAs','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(MAs,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

load_MPAs *Load CCAMLR Marine Protected Areas*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs)

Usage

```
load_MPAs()
```

See Also

load_ASDs, load_SSRUs, load_RBs, load_SSMUs, load_MAs, load_Coastline, load_EEZs.

Examples

```
#When online:
MPAs=load_MPAs()
plot(MPAs)

#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
MPAs=load_MPAs()
EEZs=load_EEZs()
save(list=c('MPAs','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(MPAs,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

load_RBs *Load CCAMLR Research Blocks*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs)

Usage

```
load_RBs()
```

See Also

load_ASDs, load_SSRUs, load_SSMUs, load_MAs, load_Coastline, load_MPAs, load_EEZs.

Examples

```
#When online:
RBs=load_RBs()
plot(RBs)

#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
RBs=load_RBs()
EEZs=load_EEZs()
save(list=c('RBs','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(RBs,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

---

load_SSMUs *Load CCAMLR Small Scale Management Units*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs)

Usage

```
load_SSMUs()
```

See Also

load_ASDs, load_SSRUs, load_RBs, load_MAs, load_Coastline, load_MPAs, load_EEZs.

Examples

```
#When online:
SSMUs=load_SSMUs()
plot(SSMUs)

#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
SSMUs=load_SSMUs()
EEZs=load_EEZs()
save(list=c('SSMUs','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(SSMUs,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

---

load_SSRUs *Load CCAMLR Small Scale Research Units*

Description

Download the up-to-date spatial layer from the online CCAMLRGIS and load it to your environment. See examples for offline use. All layers use the Lambert azimuthal equal-area projection (EPSG:6932; CRS:+proj=laea +lat_0=-90 +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs)

Usage

```
load_SSRUs()
```

See Also

load_ASDs, load_RBs, load_SSMUs, load_MAs, load_Coastline, load_MPAs, load_EEZs.

Examples

```
#When online:
SSRUs=load_SSRUs()
plot(SSRUs)

#If going offline in the future: load and save as RData when online,
# then reload RData when offline:
SSRUs=load_SSRUs()
```

```
EEZs=load_EEZs()
save(list=c('SSRUs','EEZs'), file = file.path(tempdir(), "CCAMLRLayers.RData"))
rm(SSRUs,EEZs)
load(file.path(tempdir(), "CCAMLRLayers.RData"))
```

PieData *Example dataset for create_Pies*

Description

To be used in conjunction with create_Pies. Count of species per location.

Usage

```
data(PieData)
```

Format

DataFrame

See Also

create_Pies.

Examples

```
#View(PieData)

#Create pies
MyPies=create_Pies(Input=PieData,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=50
)
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.1,PosY=-1.6,Boxexp=c(0.5,0.45,0.12,0.45),
              PieTitle="Species")
```

PieData2 *Example dataset for create_Pies*

Description

To be used in conjunction with create_Pies. Count of species per location.

Usage

```
data(PieData2)
```

Format

DataFrame

See Also

create_Pies.

Examples

```
#View(PieData2)

MyPies=create_Pies(Input=PieData2,
                   NamesIn=c("Lat","Lon","Sp","N"),
                   Size=5,
                   GridKm=250
)
#Plot Pies
plot(MyPies,col=MyPies$col)
#Add Pies legend
add_PieLegend(Pies=MyPies,PosX=-0.8,PosY=-0.3,Boxexp=c(0.5,0.45,0.12,0.45),
              PieTitle="Species")
```

PointData *Example dataset for create_Points*

Description

To be used in conjunction with create_Points.

Usage

```
data(PointData)
```

Format

DataFrame

See Also

create_Points.

Examples

```
#View(PointData)

MyPoints=create_Points(PointData)
plot(MyPoints)
text(MyPoints$x,MyPoints$y,MyPoints$name,adj=c(0.5,-0.5),xpd=TRUE)
plot(MyPoints[MyPoints$name=='four',],bg='red',pch=21,cex=1.5,add=TRUE)
```

PolyData *Example dataset for create_Polys*

Description

To be used in conjunction with create_Polys.

Usage

```
data(PolyData)
```

Format

DataFrame

See Also

create_Polys.

Examples

```
#View(PolyData)

MyPolys=create_Polys(PolyData,Densify=TRUE)
plot(MyPolys,col='green',add=TRUE)
text(MyPolys$Labx,MyPolys$Laby,MyPolys$ID)
plot(MyPolys[MyPolys$ID=='three',],border='red',lwd=3,add=TRUE)
```

`project_data` *Project user-supplied locations*

Description

Given an input dataframe containing locations (given in decimal degrees or meters), projects these locations and, if desired, appends them to the dataframe. May also be used to back-project to Latitudes/Longitudes provided the input was projected using a Lambert azimuthal equal-area projection.

Usage

```
project_data(
  Input,
  NamesIn = NULL,
  NamesOut = NULL,
  append = TRUE,
  inv = FALSE
)
```

Arguments

`Input`	dataframe containing - at the minimum - Latitudes and Longitudes to be projected (or Y and X to be back-projected).
`NamesIn`	character vector of length 2 specifying the column names of Latitude and Longitude fields in the `Input`. **Latitudes (or Y) name must be given first, e.g.:** `NamesIn=c('MyLatitudes','MyLongitudes')`.
`NamesOut`	optional. Names of the resulting columns in the output dataframe, with order matching that of `NamesIn` (e.g., `NamesOut=c('Y','X')`).
`append`	TRUE or FALSE. Should the projected locations be appended to the `Input`?
`inv`	TRUE or FALSE. Should a back-projection be performed? In such case, locations must be given in meters and have been projected using a Lambert azimuthal equal-area projection.

See Also

`assign_areas`.

Examples

```
#Generate a dataframe
MyData=data.frame(Lat=runif(100,min=-65,max=-50),
                  Lon=runif(100,min=20,max=40))
```

```
#Project data using a Lambert azimuthal equal-area projection
MyData=project_data(Input=MyData,NamesIn=c("Lat","Lon"))
```

---

seabed_area *Calculate planimetric seabed area*

Description

Calculate planimetric seabed area within polygons and depth strata in square kilometers.

Usage

```
seabed_area(Bathy, Polys, depth_classes = c(-600, -1800))
```

Arguments

Bathy	bathymetry raster with the appropriate projection, such as this one. It is recommended to use a raster of higher resolution than SmallBathy, see load_Bathy.
Polys	polygon(s) within which the areas of depth strata are computed.
depth_classes	numeric vector of strata depths. for example, depth_classes=c(-600,-1000,-2000). If the values -600,-1800 are given within depth_classes, the computed area will be labelled as 'Fishable_area'.

Value

dataframe with the name of polygons in the first column and the area for each strata in the following columns. Note that polygon names are taken from the first column in the data of the input SpatialPolygonDataframe.

See Also

load_Bathy, SmallBathy, create_Polys, load_RBs.

Examples

```
#Example 1: Compute fishable area in Research Blocks using the SmallBathy (not recommended)

RBs=load_RBs()
RBs@data[,1]=RBs$GAR_Short_Label #Take the 'GAR_Short_Label' as polygon names
FishDepth=seabed_area(SmallBathy,RBs)
#View(FishDepth)

#Example 2: Compute various strata areas within user-generated polygons
```

```
MyPolys=create_Polys(PolyData,Densify=TRUE)
FishDepth=seabed_area(SmallBathy,MyPolys,depth_classes=c(0,-200,-600,-1800,-3000,-5000))
#View(FishDepth)
```

SmallBathy *Small bathymetry dataset*

Description

Bathymetry dataset derived from the GEBCO 2020 dataset. Subsampled using raster's resample function, using the nearest neighbor method at a 10,000m resolution. Projected using the CCAMLR standard projection (CCAMLRp). To highlight the Fishable Depth range, use Depth_cols2 and Depth_cuts2. **To be only used for large scale illustrative purposes. Please refer to load_Bathy to get higher resolution data.**

Usage

```
data(SmallBathy)
```

Format

raster

References

GEBCO Compilation Group (2020) GEBCO 2020 Grid (doi:10.5285/a29c5465-b138-234d-e053-6c86abc040b9)

See Also

load_Bathy, add_col, add_Cscale, Depth_cols, Depth_cuts, Depth_cols2, Depth_cuts2, get_depths, create_Stations.

Examples

```
plot(SmallBathy,breaks=Depth_cuts,col=Depth_cols,axes=FALSE,box=FALSE)
```

Index