Package: maps (via r-universe)

September 3, 2024

Title Draw Geographical Maps

Version 3.4.2

Date 2023-12-14

Author Original S code by Richard A. Becker and Allan R. Wilks. R version by Ray Brownrigg. Enhancements by Thomas P Minka and Alex Deckmyn.

Description Display of maps. Projection code and larger maps are in separate packages ('mapproj' and 'mapdata').

Depends R (>= 3.5.0)

Imports graphics, utils

LazyData yes

Suggests mapproj (>= 1.2-0), mapdata (>= 2.3.0), sp, rnaturalearth

License GPL-2

Maintainer Alex Deckmyn <alex.deckmyn@meteo.be>

NeedsCompilation yes

Repository https://adeckmyn.r-universe.dev

RemoteUrl https://github.com/adeckmyn/maps

RemoteRef HEAD

RemoteSha 3fbbe684e6a9fe21b09c7fa8d908ee7855a97065

Contents

`area.map` *Area of projected map regions*

Description

Computes the areas of regions in a projected map.

Usage

```
area.map(m, regions = ".", sqmi=TRUE, ...)
```

Arguments

`m`	a map object containing named polygons (created with `fill = TRUE`).
`regions`	a character vector naming one of more regions, as in `map`.
`sqmi`	If `TRUE`, measure area in square miles. Otherwise keep the units of `m`.
`...`	additional arguments to `match.map`

Details

The area of each matching region in the map is computed, and regions which match the same element of `regions` have their areas combined. Each region is assumed planar, with vertices specified by the `x` and `y` components of the map object.

The correct use of this function is to first use `map` to create polygons and project the coordinates onto a plane, then apply `area.map` to compute the area of the projected regions. If the projection is area-preserving (such as `albers`), then these areas will match the area on the globe, up to a constant. To get an absolute area in square miles, the `sqmi` option will scale the result, depending on the projection.

The coordinates from `map` are affected by its `resolution` argument, so use `resolution=0` for the most accurate areas.

Value

a named vector of region areas.

NOTE

The `sqmi` option assumes the coordinates have been projected with the mapproject function.

Author(s)

Tom Minka

See Also

`area.polygon`, `apply.polygon`

Examples

```
# because the projection is rectangular, these are not true areas on the globe.
m = map("state", fill = TRUE, plot = FALSE)
area.map(m)
area.map(m, ".*dakota")
area.map(m, c("North Dakota", "South Dakota"))

if(require(mapproj)) {
  # true areas on the globe
  m = map("state", proj="bonne", param=45, fill=TRUE, plot=FALSE)
  # North Dakota is listed as 70,704 square miles
  area.map(m, "North Dakota")
}
```

canada.cities *Database of Canadian cities*

Description

This database is of Canadian cities of population greater than about 1,000. Also included are province capitals of any population size.

Format

A list with 6 components, namely "name", "country.etc", "pop", "lat", "long", and "capital", containing the city name, the province abbreviation, approximate population (as at January 2006), latitude, longitude and capital status indication (0 for non-capital, 1 for capital, 2 for provincial

capital.

NOTE

Some of the city names may be out of date. Please send any corrections to the package maintainer.

See Also

map.cities

county *United States County Map*

Description

This database produces a map of the counties of the United States mainland generated from US Department of the Census data (see the reference).

Usage

```
data(countyMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (R_MAP_DATA_DIR for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

US Department of Commerce, Census Bureau, *County Boundary File,* computer tape, available from Customer Services, Bureau of the Census, Washingdon DC 20233.

See Also

map.

Examples

```
map('county', 'iowa', fill = TRUE, col = palette())
```

---

county.fips *FIPS county codes for US County Map*

Description

A database matching FIPS codes to maps package county and state names.

Usage

```
data(county.fips)
```

Format

A list with 2 components, namely "fips" and "polyname", containing the FIPS number and respective state or county polygon name. Note that "fips" is represented as an integer, so any leading zero (which is part of the fips code) is not shown by default.

See Also

state.fips

`france` *France Map*

Description

This france database comes from the NUTS III (Tertiary Administrative Units of the European Community) database of the United Nations Environment Programme (UNEP) GRID-Geneva data sets. These were prepared around 1989, and so may be somewhat out of date.

Users of data sets supplied through UNEP/GRID are requested to incorporate in output products and reports acknowledgements to the originator of the data and to the fact that they were acquired through UNEP/GRID. Appropriate wording may be "UNESCO (1987) through UNEP/GRID-Geneva".

Usage

```
data(franceMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (`R_MAP_DATA_DIR` for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

Details

This map database can now easily be replaced by data taken directly from free sources, e.g. Natural Earth (see example).

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

See Also

`map`

Examples

```
map('france', fill = TRUE, col = 1:10)
# replace by a public domain map at higher resolution:
# fr1 <- rnaturalearth::ne_states("france")
# this still includes overseas domains, so we remove those:
# france2 <- map(fr1, xlim=c(-20, 20), ylim=c(30, 60), lforce="e",
#                fill=TRUE, plot=FALSE)
```

`identify.map` *Identify regions on a map*

Description

Identifies the map regions clicked by the user.

Usage

```
## S3 method for class 'map'
identify(x, n = 1, index = FALSE, ...)
```

Arguments

`x`	a map object containing named polygons.
`n`	the number of clicks to wait for.
`index`	If TRUE, returns the index of the polygon, rather than its name.
`...`	additional arguments passed to `identify.default`.

Details

The current algorithm is somewhat crude — selects the region whose centroid is closest to the click. A more sophisticated approach would use `map.where`.

Value

a character vector of length n, naming the selected regions.

Author(s)

Tom Minka

See Also

`identify`, `map.where`

Examples

```
identify(map("state", fill = TRUE, col = 0))
if(require(mapproj))
  identify(map("world", proj = "lagrange", fill = TRUE, col = 0, wrap=c(-180,180,-90)))
```

`iso.expand` *Identify countries by ISO 3166 codes (2 or 3 letters) or by Sovereignty.*

Description

This data set and the simple look-up functions allow to build lists of counrtries for the world map.

Usage

```
iso.expand(a, regex=TRUE)
sov.expand(sov, regex=TRUE)
       iso.alpha(x, n=2)
```

Arguments

`a`	A vector of ISO codes. All elements should have the same length, either 2 or 3 letters. Not case sensitive.
`sov`	A vector of country names. The result is a list of all countries that fall under their sovereignty. Case sensitive, must fit completeley.
`regex`	If TRUE (default), the return vector has the same length as the input (`a` or `sov`), but the entries may be regular expressions. If FALSE, the result is a vector of polygon names. This may be more readable, but the return vector may be longer than the input.
`x`	Vector of country names, may include colons.
`n`	An integer identitying which ISO code is required. Allowed values are 2 and 3.

Details

The ISO 3166-1 standard identifies countries by a 2 and 3 letter codes. `iso.expand` translates these codes into the country names as used by the `world` data base. `iso.alpha` does the reverse. Some countries have different ISO codes for different regions (e.g. China:Hong Kong has ISO code HK). In such cases, `iso.alpha` will return the main code, but `iso.expand` will return a regular expression that excludes some parts.

Value

`iso.expand` returns vector of country names. When used as input for `map` it will plot all the countries as identified either by their sovereignty or by ISO codes. If `regex=FALSE` the length of the vector may be shorter or longer than the input. If `regex=TRUE`, the results are concatenated in regular expressions. This format is less readable, but can be used as input e.g. for `match.map`. `iso.alpha` always returns a vector of the same length as the input, containing the 2- or 3-letter codes.

NOTE

These functions use regular expressions and the results will often not work well with `map(...,exact=TRUE)`.

References

https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

See Also

match.map,map.text,iso3166

Examples

```
# France and all its overseas departments, territories etc.
sov.expand("France") # France and all its overseas departments, territories etc.

# Canary Islands are not included in map("Spain")
iso.expand("ES")
map(regions=sov.expand("Spain"))

# draw a map with ISO codes as labels:
wm <- map("world", fill=TRUE, col=0, xlim=c(-10,40), ylim=c(30,60))
# take out islands, but you loose e.g. UK, New Zealand, small island states
nam <- grep(":", wm$names, inv=TRUE, val=TRUE)
# ad ISO codes as label
map.text(wm, regions=nam, label=iso.alpha(nam), col=2, exact=TRUE, add=TRUE)
```

---

iso3166 *ISO 3166 country codes (2 or 3 letters) and sovereignty.*

Description

This data set lists all ISO3166 country codes and the sovereignty for each country in the list. Some entries are regular expressions.

Format

A data frame with 5 columns: "a2", "a3", "name", "mapname", "sovereignty". These contain the 2- and 3-letter ISO code, the official name, the (possibly shorter) name used in the map data base, and the sovereign country.

Details

The ISO 3166-1 standard identifies countries by a 2 and 3 letter codes. This table listst these for all countries on the world map. This data set also serves as basis for the function iso.expand() and its siblings.

NOTE

Some countries have different ISO codes for some regions. To deal with such particular cases, the "mapname" column may sometimes contain (perl-style) regular expressions rather than simply a country name. For instance, "FI" has mapname "Finland(?!:Aland)", because the Aland islands have a different ISO code. Other codes may appear in two rows if certain parts of countries are not written with the main country as base name. Usually, that is for compatibility with the legacy world data base.

References

https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

See Also

iso.expand

---

italy *Italy Map*

Description

This italy database comes from the NUTS III (Tertiary Administrative Units of the European Community) database of the United Nations Environment Programme (UNEP) GRID-Geneva data sets. These were prepared around 1989, and so may be somewhat out of date.

Users of data sets supplied through UNEP/GRID are requested to incorporate in output products and reports acknowledgements to the originator of the data and to the fact that they were acquired through UNEP/GRID. Appropriate wording may be "UNESCO (1987) through UNEP/GRID-Geneva".

Usage

```
data(italyMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (R_MAP_DATA_DIR for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

See Also

map

Examples

```
map('italy', fill = TRUE, col = 1:10)
```

---

lakes *World lakes database*

Description

This database contains a selection of large lakes (and islands within) taken from the Natural Earth 1:50m map, the same data source as the (v3.0) world map. The lake boundaries are consistent with the 'world' database.

Usage

```
data(lakesMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (R_MAP_DATA_DIR for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

Source

The data in this data base is derived from the public domain GIS project Natural Earth, the file "ne_50m_lakes". The Natural Earth data set is available from https://www.naturalearthdata.com.

References

Natural Earth project https://www.naturalearthdata.com

See Also

map.

Examples

```
map('world')
map('lakes', add=TRUE, fill=TRUE, col='white', boundary='black')
```

map *Draw Geographical Maps*

Description

Draw lines and polygons as specified by a map database.

Usage

```
map(database = "world", regions = ".", exact = FALSE, boundary = TRUE,
  interior = TRUE, projection = "", parameters = NULL, orientation = NULL,
  fill = FALSE, col = 1, plot = TRUE, add = FALSE, namesonly = FALSE,
  xlim = NULL, ylim = NULL, wrap = FALSE, resolution = if (plot) 1 else 0,
  type = "l", bg = par("bg"), mar = c(4.1, 4.1, par("mar")[3], 0.1),
  myborder = 0.01, namefield="name", lforce="n", ...)
```

Arguments

database	character string naming a geographical database, a list of x, y, and names obtained from a previous call to map or a spatial object of class SpatialPolygons or SpatialLines. The string choices include a world map, three USA databases (usa, state, county), and more (type help(package='maps') to see the package index). If the requied database is in a different package that has not been attached, the string may be started with "packagename::". The location of the map databases may be overridden by setting the R_MAP_DATA_DIR environment variable.
regions	character vector that names the polygons to draw. Each database is composed of a collection of polygons, and each polygon has a unique name. When a region is composed of more than one polygon, the individual polygons have the name of the region, followed by a colon and a qualifier, as in michigan:north and michigan:south. Each element of regions is matched against the polygon names in the database and, according to exact, a subset is selected for drawing. The regions may also be defined using (perl) regular expressions. This makes it possible to use 'negative' expressions like "Norway(?!:Svalbard)", which means Norway and all islands except Svalbard. All entries are case insensitive. The default selects all polygons in the database.
exact	If TRUE, only exact matches with regions are selected for drawing. If FALSE, each element of regions is matched as a regular expression against the polygon names in the database and all matches are selected for drawing.
boundary	If FALSE, boundary segments are not drawn. A boundary segment is a line segment of the map that bounds only one of the polygons to be drawn. This argument is ignored if fill is TRUE.
interior	If FALSE, interior segments are not drawn. An interior segment is a line segment of the map that bounds two of the polygons to be drawn. This argument is ignored if fill is TRUE.

`projection` character string that names a map projection to use. See `mapproject` (in the `mapproj` library). The default is to use a rectangular projection with the aspect ratio chosen so that longitude and latitude scales are equivalent at the center of the picture.

`parameters` numeric vector of parameters for use with the `projection` argument. This argument is optional only in the sense that certain projections do not require additional parameters. If a projection does require additional parameters, these must be given in the `parameters` argument.

`orientation` a vector `c(latitude, longitude, rotation)` describing where the map should be centered and a clockwise rotation (in degrees) about this center.

`fill` logical flag that says whether to draw lines or fill areas. If `FALSE`, the lines bounding each region will be drawn (but only once, for interior lines). If `TRUE`, each region will be filled using colors from the `col` = argument, and bounding lines are drawn by default using `par("fg")`. To hide the bounding lines, use `border=NA` (see `...`).

`col` vector of colors. If `fill` is `FALSE`, the first color is used for plotting all lines, and any other colors are ignored. Otherwise, the colors are matched one-one with the polygons that get selected by the `region` argument (and are reused cyclically, if necessary). If `fill` = `TRUE`, the default boundary line colour is given by `par("fg")`. To change this, you can use the `border` argument (see '...'). A color of `NA` causes the corresponding region to be deleted from the list of polygons to be drawn. Polygon colors are assigned *after* polygons are deleted due to values of the `xlim` and `ylim` arguments.

`plot` logical flag that specifies whether plotting should be done. If `plot` is `TRUE` the return value of `map` will not be printed automatically .

`add` logical flag that specifies whether to add to the current plot. If `FALSE`, a new plot is begun, and a new coordinate system is set up.

`namesonly` If `TRUE`, the return value will be a character vector of the names of the selected polygons. See the Value section below.

`xlim` two element numeric vector giving a range of longitudes, expressed in degrees, to which drawing should be restricted. Longitude is measured in degrees east of Greenwich, so that, in particular, locations in the USA have negative longitude. If `fill` = `TRUE`, polygons selected by `region` must be entirely inside the `xlim` range. The default value of this argument spans the entire longitude range of the `database`.

`ylim` two element numeric vector giving a range of latitudes, expressed in degrees, to which drawing should be restricted. Latitude is measured in degrees north of the equator, so that, in particular, locations in the USA have positive latitude. If `fill` = `TRUE`, polygons selected by `region` must be entirely inside the `ylim` range. The default value of this argument spans the entire latitude range of the `database`.

`wrap` Boolean or a numeric vector. If TRUE, lines that cross too far across the map (due to a strange projection) are omitted. If wrap is a vector of length 2 or more, it is interpreted as the longitude range to be used for a global map, e.g. `c(-180,180)` or `c(0,360)`. This wrapping even works when `fill=TRUE` and is performed before any projection (so the range must always be in degrees).

However, the wrapping is performed *after* xlim, ylim are applied, so these options should probably never be combined. If there is a third component, this signifies the latitude at which Antarctica will be "closed". The default value is -89.9. Special values are NA (don't draw Antarctica at all) and 0 (draw the line at the latitude of the extremal points, not at a fixed lower latitude).

resolution — number that specifies the resolution with which to draw the map. Resolution 0 is the full resolution of the database. 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).

type — character string that controls drawing of the map. Aside from the default type = "l", the value type = "n" can be used to set up the coordinate system and projection for a map that will be added to in later calls.

bg — background color.

mar — margins, as in par. Defaults allow for map.axes().

myborder — scalar or vector of length 2 specifying the porportion of the plot to add to the defined or computed limits as borders.

namefield — A vector of column names to be used as region name if database is a SpatialPolygonsDataFrame. Ignored in all other cases.

lforce — Limit enforcement. Only taken into account if xlim and/or ylim are defined and (for "s" and "l") a projection is used. Possible values are "n" (none) "e" (exact),"s" (small),"l" (large). If lforce="e", map data is restricted exactly to the given limits prior to projection. This option affects the output value of the map, not only the plot window. If lforce="s", the four corners defined by xlim and ylim are projected and the plot window is restricted to the largest rectangle inside this region. "l" will result in a larger rectangle that includes the four corners. However, the parts that fall outside the original limits may not be complete (only polygons that partially fall inside the boundaries are included). This will also work with fill=TRUE. In this case the output value of the data is not changed, only the plot window is affected. These options are only useful for some projections.

... — Extra arguments passed to polygon or lines. Of particular interest may be the options border andlty that control the color and line type of the polygon borders when fill = TRUE.

Details

The simplest form of use of this function is:

```
map(mymap)
```

where mymap is the returned value from a previous call to map().

Value

If plot = TRUE, a plot is made where the polygons selected from database, through the regions, xlim, and ylim arguments, are outlined (fill is FALSE) or filled (fill is TRUE) with the colors in col.

The return value is a list with x, y, range, and names components. This object can be used as a database for successive calls to map and functions. If fill is FALSE, the x and y vectors are the coordinates of successive polylines, separated by NAs. If fill is TRUE, the x and y vectors have coordinates of successive polygons, again separated by NAs. Thus the return value can be handed directly to lines or polygon, as appropriate.

When namesonly is TRUE, only the names component is returned.

After a call to map for which the projection argument was specified there will be a global variable .Last.projection containing information about the projection used. This will be consulted in subsequent calls to map which use projection = ''.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.* https://web.archive.org/web/20050825145143/http://www.research.att.com/areas/stat/doc/93.2.ps

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.* https://web.archive.org/web/20050825145143/http://www.research.att.com/areas/stat/doc/95.2.ps

See Also

map.text, map.axes, map.scale, map.grid (in the mapproj library), polygon, SpatialPolygons2map

Examples

```
map() # low resolution map of the world
map(wrap = c(0,360), fill = TRUE, col = 2) # pacific-centered map of the world
map(wrap = c(0, 360, NA), fill = TRUE, col = 2) # idem, without Antarctica
map('usa') # national boundaries
map('county', 'new jersey') # county map of New Jersey
map('state', region = c('new york', 'new jersey', 'penn')) # map of three states
map("state", ".*dakota", myborder = 0) # map of the dakotas
map.axes()    # show the effect of myborder = 0
if(require(mapproj))
  map('state', proj = 'bonne', param = 45) # Bonne equal-area projection of states

# names of the San Juan islands in Washington state
map('county', 'washington,san', names = TRUE, plot = FALSE)

# national boundaries in one linetype, states in another
# (figure 5 in the reference)
map("state", interior = FALSE)
map("state", boundary = FALSE, lty = 2, add = TRUE)

# plot the ozone data on a base map
```

```
# (figure 4 in the reference)
data(ozone)
map("state", xlim = range(ozone$x), ylim = range(ozone$y))
text(ozone$x, ozone$y, ozone$median)
box()
if(require(mapproj)) { # mapproj is used for  projection="polyconic"
  # color US county map by 2009 unemployment rate
  # match counties to map using FIPS county codes
  # Based on J's solution to the "Choropleth Challenge"
  # http://blog.revolutionanalytics.com/2009/11/choropleth-challenge-result.html

  # load data
  # unemp includes data for some counties not on the "lower 48 states" county
  # map, such as those in Alaska, Hawaii, Puerto Rico, and some tiny Virginia
  #  cities
  data(unemp)
  data(county.fips)

  # define color buckets
  colors = c("#F1EEF6", "#D4B9DA", "#C994C7", "#DF65B0", "#DD1C77", "#980043")
  unemp$colorBuckets <- as.numeric(cut(unemp$unemp, c(0, 2, 4, 6, 8, 10, 100)))
  leg.txt <- c("<2%", "2-4%", "4-6%", "6-8%", "8-10%", ">10%")

  # align data with map definitions by (partial) matching state,county
  # names, which include multiple polygons for some counties
  cnty.fips <- county.fips$fips[match(map("county", plot=FALSE)$names,
    county.fips$polyname)]
  colorsmatched <- unemp$colorBuckets [match(cnty.fips, unemp$fips)]

  # draw map
  map("county", col = colors[colorsmatched], fill = TRUE, resolution = 0,
    lty = 0, projection = "polyconic")
  map("state", col = "white", fill = FALSE, add = TRUE, lty = 1, lwd = 0.2,
    projection="polyconic")
  title("unemployment by county, 2009")
  legend("topright", leg.txt, horiz = TRUE, fill = colors)

  # Choropleth Challenge example, based on J's solution, see:
  # http://blog.revolutionanalytics.com/2009/11/choropleth-challenge-result.html
  # To see the faint county boundaries, use RGui menu:  File/SaveAs/PDF
}
```

`map.axes` *Draw Axes on Geographical Maps*

Description

Draws a set of axes on an existing map.

Usage

```
map.axes(...)
```

Arguments

...	Extra arguments passed to axis or box.

Side Effects

x- and y-axes are drawn for the currently displayed map. These will display in longitude and latitude (if no projection= has been specified in the map() call).

Examples

```
map("state")
map.axes(cex.axis=0.8)
```

---

map.cities *Add Cities to Existing Map*

Description

Adds city locations and (optionally) names to an existing map using a specified database.

Usage

```
map.cities(x = world.cities, country = "", label = NULL, minpop = 0,
maxpop = Inf, capitals = 0, cex = par("cex"), projection = FALSE,
parameters = NULL, orientation = NULL, pch = 1, ...)
```

Arguments

x	Name of database. See world.cities to determine the structure of the database.
country	If the string country is specified, limit the displayed cities to be from within the specified country, province or state (depending on how the database has been constructed).
label	If TRUE, label all cities. If NULL, the cities will be labelled unless there are 20 or more.
minpop	The minimum value of population below which a particular city will not be shown.
maxpop	The maximum value of population above which a particular city will not be shown.
capitals	Selection of capitals-only display. Capitals may be 1 (country capital), 2 (provincial, state, or regional capital) or 3 (local capital). See world.cities for further information.
cex	The value of cex acts to override the current value of character size expansion.

projection	Boolean or character value. If FALSE (the default), no projection is assumed, if TRUE, the previous projection is used, otherwise a character string that names a map projection to use. See mapproject (in the mapproj library).
parameters	numeric vector of parameters for use with the projection argument. This argument is optional only in the sense that certain projections do not require additional parameters. If a projection does require additional parameters, these must be given in the parameters argument.
orientation	a vector c(latitude, longitude, rotation) describing where the map should be centered and a clockwise rotation (in degrees) about this center.
pch	plotting character to use for marking city location. See points for options.
...	Further plotting parameters may be specified as for the commands points and text.

Details

The database is searched for all cities matching the specified criteria and fitting within the limits of the plot currently displayed. The default database is of all cities that have a population greater than a certain threshold or which are capital cities of a country or island territory. The threshold varies from country to country, but in general is no higher than about 40,000. The data were originally obtained from Stefan Helders' website (http://www.world-gazetteer.com), which now redirects to http://www.populationmondiale.com. There are no recent updates available.

There are three supplied databases, world.cities (the default), us.cities and canada.cities. The latter two, which need to be made available by using a 'data()' call, include the state or province name with the city name (thanks to John Woodruff <jpwoodruff@irisinternet.net> for the state and province information).

Note that if the underlying map is "Pacific-centric", i.e. longitudes exceed 180 degrees, and a projection is used, then the map.cities data must be transformed appropriately.

Value

No value is returned from map.cities.

Side Effects

All cities within the boundaries of the plot containing the current map are added to the plot. Note that it is possible that the boundaries of the plot exceed the boundaries of the map requested, and so more cities than were expected might be shown.

See Also

world.cities, canada.cities, us.cities

Examples

```
map("world", "China")
map.cities(country = "China", capitals = 2)
map("state", "New Jersey")
data(us.cities)
map.cities(us.cities, country="NJ")
```

`map.scale` *Add Scale to Existing Unprojected Map*

Description

Adds a scale to an existing map, both as a ratio and a distance gauge.

Usage

```
map.scale(x, y, relwidth = 0.15, metric = TRUE, ratio = TRUE, ...)
```

Arguments

`x`	Horizontal location of left end of distance gauge. If not specified, this will be taken to be near the lower left corner of the map.
`y`	Vertical location of left end of distance gauge. If not specified, this will be taken to be near the lower left corner of the map.
`relwidth`	Proportion of width of display to be used for the scale. The default is 0.15 (15%).
`metric`	If `TRUE`, the distance gauge will be in km, otherwise miles.
`ratio`	If `FALSE`, the scale ratio of the map is not displayed.
`...`	Further plotting parameters may be specified as for the command text().

Details

The scale is calculated from the displayed graph's plotting parameters, and the latitude of the location at which the distance gauge will be displayed.

Value

The exact calculated scale is returned.

NOTE

This function is meaningful only if no projection= has been specified in the call to map().

Side Effects

A scale is added to the currently displayed map. This takes the form of an approximate 1:n scale (containing 2-3 significant digits), above a distance gauge which is reasonably accurate for the latitude at which it appears. The circumference at the given latitude is interpolated from a radius of 6356.78 km at the pole and 6378.16 km at the equator.

See Also

`map.axes`

Examples

```
map("world", "China")
map.scale()
```

`map.text` *Draw a map with labeled regions*

Description

Like `map`, but labels the regions.

Usage

```
map.text(database, regions = ".", exact = FALSE, labels, cex = 0.75,
add = FALSE, move = FALSE, ...)
```

Arguments

`database`	character string naming a geographical database, or a list of `x`, `y`, and `names` obtained from a previous call to `map`.
`regions`	character vector that names the polygons to draw.
`exact`	If 'TRUE', only exact matches with 'regions' are selected for drawing.
`labels`	character vector of labels, one for each region selected. Defaults to the names in the database.
`cex`	character expansion factor.
`add`	If `FALSE`, a map is drawn, then labels placed on top. If `TRUE`, labels are added to the existing map.
`move`	If `TRUE`, labels are moved so that they don't overlap. Requires the `mining` library (not in CRAN, contact tpminka@media.mit.edu).
`...`	Other arguments are the same as in `map`.

Value

If `add = FALSE`, a map is drawn by calling `map`. Then the label for each region is placed at the centroid of the region polygon.

The return value is a map object, as from `map`.

Author(s)

Tom Minka

Examples

```
map.text("world", "ira")    # iran and iraq
map.text("state", "penn")
map.text("county", "penn")  # Pennsylvania counties
map.text("county", "new jersey")  # New Jersey counties
```

`map.where` *Locate points on a map*

Description

Returns the region names containing given locations.

Usage

```
map.where(database = "world", x, y, ...)
```

Arguments

`database`	character string naming a geographical database, or a list of `x`, `y`, and `names`. See the documentation for map for more details.
`x`	vector of longitudes.
`y`	vector of latitudes.
`...`	Options for `SpatialPolygons2map`, only used if `database` is of type `SpatialPolygonsDataFrame`
.	

Value

A list of character strings, naming the map region that each (longitude, latitude) pair falls into.

Note

For points close to a border (polygon boundary), the result may be wrong if the resolution of the database is insufficient. This function may also give erroneous results if the database contains enclaves. For instance, a point in San Marino may also be identified as being in Italy.

Author(s)

Tom Minka

See Also

`in.polygon`

Examples

```
# NYC
map.where("state", -73.8, 41)
# Auckland
map.where("nz", 174.6, -36.92)
# find both in the world
map.where(x = c(174.6, -73.8), y = c(-36.92, 41))
# with a map object:
m = map("state", "new york", fill = TRUE, plot = FALSE)
map.where(m, -73.8, 41)
```

`match.map` *Index map regions*

Description

Assigns an index to each map region, useful for map coloring.

Usage

```
match.map(database, regions, exact = FALSE, warn = TRUE)
```

Arguments

`database`	character string naming a geographical database, or a map object. See the documentation for `map` for more details.
`regions`	a vector of names, or more generally regular expressions to match against the map region names.
`exact`	If `TRUE`, only exact matches with `regions` are considered. Otherwise each element of `regions` is assumed to be a regular expression. Matches are always case-insensitive.
`warn`	If `TRUE`, a warning is printed when an element of `regions` matches nothing in the map.

Value

Returns an integer vector giving an index to each region in the database. The index is the index of the string in `regions` which matches the region name. Matching is done as in `map`. More specifically, all regions `r` whose name matches `regions[i]` will have index `i`. Unmatched regions will have index `NA`. Overlapping matches cause an error.

This behavior differs from `pmatch` because a single entry in `regions` may match several entries in the map.

Author(s)

Tom Minka

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report, 1991.*

See Also

`grep`

Examples

```
# filled map showing Republican vote in 1900
# (figure 6 in the reference)
data(state, package = "datasets")
data(votes.repub)
state.to.map <- match.map("state", state.name)
x <- votes.repub[state.to.map, "1900"]
gray.colors <- function(n) gray(rev(0:(n - 1))/n)
color <- gray.colors(100)[floor(x)]
map("state", fill = TRUE, col = color); map("state", add = TRUE)
```

---

nz *New Zealand Basic Map*

Description

This database produce a map of New Zealand at a basic level of detail. The '"nz"' database includes the 3 main Islands and 19 smaller coastal islands.

Usage

```
data(nzMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (`R_MAP_DATA_DIR` for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

See Also

`map`

Examples

```
map('nz')
map('nz', xlim = c(166, 179), ylim = c(-48, -34))
```

ozone *Sample datasets*

Description

Datasets used to illustrate map functions.

ozone contains the median of daily maxima ozone concentration in 41 US cities for June 1974 through August 1974. Concentrations are in parts per billion (ppb).

unemp Has population and unemployment percentage for US counties.

votes.repub contains the percentage republican votes in the 1900 election.

Usage

```
data(ozone)
data(unemp)
data(votes.repub)
```

References

Cleveland, W.S., Kleiner, B., McRae, J.E., Warner, J.L., and Pasceri, P.E. , "The Analysis of Ground-Level Ozone Data from New Jersey, New York, Connecticut, and Massachusetts: Data Quality Assessment and Temporal and Geographical Properties", *Bell Laboratories Memorandum, 1975*.

smooth.map *Smooth out aggregated data*

Description

Increases the resolution of data aggregated over map regions, by either smoothing or interpolation. Also fills in missing values.

Usage

```
smooth.map(m, z, res = 50, span = 1/10, averages = FALSE, type = c("smooth",
"interp"), merge = FALSE)
```

Arguments

m	a map object
z	a named vector
res	a vector of length two, specifying the resolution of the sampling grid in each dimension. If a single number, it is taken as the vertical resolution, with double taken as the horizontal resolution.

span	kernel parameter (larger = smoother). span = Inf is a special case which invokes the cubic spline kernel. span is automatically scaled by the map size, and is independent of res.
averages	If TRUE, the values in z are interpreted as averages over the regions. Otherwise they are interpreted as totals.
type	see details.
merge	If TRUE, a region named in z includes all matching regions in the map (according to match.map). If FALSE, a region named in z is assumed to refer to exactly one region on the map.

Details

For type = "smooth", the region totals are first converted into point measurements on the sampling grid, by dividing the total for a region among all sample points inside it. Then it is a regular kernel smoothing problem. Note that the region totals are not preserved.

The prediction z_o for location x_o (a vector) is the average of z for nearby sample points:

$$z_o = \frac{\sum_x k(x, x_o) z(x)}{\sum_x k(x, x_o)}$$

$$k(x, x_o) = exp(-\lambda ||x - x_o||^2)$$

λ is determined from span. Note that x_o is over the same sampling grid as x, but z_o is not necessarily the same as $z(x_o)$.

For type = "interp", the region totals are preserved by the higher-resolution function. The function is assumed to come from a Gaussian process with kernel k. The measurement z[r] is assumed to be the sum of the function over the discrete sample points inside region r. This leads to a simple formula for the covariance matrix of z and the cross-covariance between zo and z. The prediction is the cross-covariance times the inverse covariance times z. Unlike Tobler's method, the predictions are not constrained to live within the original data range, so there tends to be "ringing" effects.

See the references for more details.

Value

A data frame with columns x, y, and z giving the smoothed value z for locations (x, y). Currently the (x, y) values form a grid, but this is not guaranteed in the future.

Author(s)

Tom Minka

References

W.F. Eddy and A. Mockus. An example of the estimation and display of a smoothly varying function of time and space - the incidence of disease mumps. *Journal of the American Society for Information Science*, 45(9):686-693, 1994. https://web.eecs.utk.edu/~audris/papers/jasis.pdf

W. R. Tobler. Smooth pycnophylactic interpolation for geographical regions. *Journal of the American Statistical Association* 74:519-530, 1979.

Examples

```
# compare to the example for match.map
data(state, package = "datasets")
data(votes.repub)
z = votes.repub[, "1900"]
m = map("state", fill = TRUE, plot = FALSE)
# use a small span to fill in, but not smooth, the data
# increase the resolution to get better results
fit = smooth.map(m, z, span = 1/100, merge = TRUE, ave = TRUE)
mat = tapply(fit$z, fit[1:2], mean)
gray.colors <- function(n) gray(rev(0:(n - 1))/n)
par(bg = "blue")
filled.contour(mat, color.palette = gray.colors, nlev = 32, asp = 1)
# another way to visualize:
image(mat, col = gray.colors(100))

# for a higher degree of smoothing:
# fit = smooth.map(m, z, merge = TRUE, ave = TRUE)
# interpolation, state averages are preserved:
# fit = smooth.map(m, z, merge = TRUE, ave = TRUE, type = "interp")
```

`Spatial2map` *Read SpatialPolygons and SpatialLines objects*

Description

These functions transform some classes provided by the package `sp` into a simple list that can be used by map().

Usage

```
SpatialPolygons2map(database, namefield=NULL)
SpatialLines2map(database, namefield=NULL)
```

Arguments

`database`	A `SpatialPolygons` or `SpatialLines` object.
`namefield`	The name of a data column in `database` to be used for naming the polygons (or lines). If it is a vector of names, these are all used and separated by a colon ':'. Not case sensitive. So if the database contains columns that only differ by case, you get a warning and namefield is not used at all.

Details

The 'map' list object only preserves co-ordinates and polygon names. All other information available in the original data is lost.

The option `namefield` is only taken into account if `database` is class `Spatial[]DataFrame`. `namefield` may be a vector of column names, e.g. to get polygons named as 'country:state'.

Value

A list with four components: `x, y, names, range`, similar to the return value of `map()`. This data can be used as a database for `map()`. The lines and polygons are separated by NA.

See Also

`map`,`SpatialPolygons` (in the sp library).

`state` *United States State Boundaries Map*

Description

This database produces a map of the states of the United States mainland generated from US Department of the Census data (see the reference).

Usage

```
data(stateMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (`R_MAP_DATA_DIR` for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

US Department of Commerce, Census Bureau, *County Boundary File,* computer tape, available from Customer Services, Bureau of the Census, Washingdon DC 20233.

See Also

`map`.

Examples

```
map('state', fill = TRUE, col = palette())
```

`state.carto` *United States State Population Cartogram Map*

Description

This database produces a cartogram of the states of the United States mainland based on CartoDraw, roughly proportional to population (see references).

`state.carto.center` are coordinates of the state centers for annotation purposes.

Usage

```
data(stateMapEnv)
data(state.carto.center)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (`R_MAP_DATA_DIR` for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

CartoDraw, http://www.computer.org/csdl/trans/tg/2004/01/v0095-abs.html

See Also

`map`.

Examples

```
map('state.carto', fill = TRUE, col = palette())
```

state.fips *FIPS state codes for US 48 State Map*

Description

A database matching FIPS codes to maps package state names.

Usage

```
data(state.fips)
```

Format

A list with 6 components, namely "fips", "ssa", "region", "division", "abb" and "polyname", containing the US Census Bureau FIPS, SSA, REGION and DIVISION numbers, the standard state abbreviation and the respective state polygon name. Note that "fips" is represented as an integer, so any leading zero (which is part of the fips code) is not shown by default.

See Also

county.fips

state.vbm *United States State Visibility Base Map*

Description

This database produces a map of the states of the United States mainland. The Visibility Base Map was created by Mark Monmonier to provide simplified state shapes with sufficient areas to allow annotations in even the small states.

`state.vbm.center` are coordinates of the state centers for annotation purposes. The states are alphabetically ordered, in the same order as the map. So state names can be matched via e.g. `map(state.vbm, plot=FALSE)$name`.

Usage

```
data(state.vbmMapEnv)
data(state.vbm.center)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (`R_MAP_DATA_DIR` for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

Mark Monmonier and George Schnell, "The Study of Population", *Elements, Patterns, Processes. Charles E. Merrill. Columbus, OH. 1982.*

See Also

map.

Examples

```
map('state.vbm', fill = TRUE, col = palette())
```

---

us.cities *Database of US cities*

Description

This database is of us cities of population greater than about 40,000. Also included are state capitals of any population size.

Format

A list with 6 components, namely "name", "country.etc", "pop", "lat", "long", and "capital", containing the city name, the state abbreviation, approximate population (as at January 2006), latitude, longitude and capital status indication (0 for non-capital, 1 for capital, 2 for state capital.

NOTE

Some of the city names may be out of date. Please send any corrections to the package maintainer.

See Also

map.cities

usa *United States Coast Map*

Description

This database produces a map of the United States mainland generated from US Department of the Census data (see the reference).

Usage

```
data(usaMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (`R_MAP_DATA_DIR` for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

US Department of Commerce, Census Bureau, *County Boundary File,* computer tape, available from Customer Services, Bureau of the Census, Washingdon DC 20233.

See Also

`map`.

Examples

```
map('usa')
```

world *Low (mid) resolution World Map*

Description

This world map (updated in 2013) is imported from the public domain Natural Earth project (the 1:50m resolution version). It replaces a much older version based on the CIA World Data Bank II data. The old legacy data is still available in the package mapdata (v2.3.0).

Usage

```
data(worldMapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (R_MAP_DATA_DIR_WORLD) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

Details

As of version 3.1, the world database no longer contains any lakes. These have been moved to a separate database called lakes. The legacy world map (dating from around 1990) has been removed from the package and is now available from the mapdata package in two different resolutions (worldHires and worldLores).

Source

The Natural Earth data set is in the public domain and available from https://www.naturalearthdata.com.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

See Also

map,lakes

Examples

```
# notice how some polygons extend beyond the [-180,180] interval:
map('world', fill = TRUE, col = 1:10)
# if you wrap at [-180,180], you also can get a clean closure of Antarctica
map('world', fill = TRUE, col = 1:10, wrap=c(-180,180) )
```

---

world.cities *Database of world cities*

Description

This database is primarily of world cities of population greater than about 40,000. Also included are capital cities of any population size, and many smaller towns.

Usage

```
data(world.cities)
```

Format

A list with 6 components, namely "name", "country.etc", "pop", "lat", "long", and "capital", containing the city name, the country name, approximate population (as at January 2006), latitude, longitude and capital status indication (0 for non-capital, 1 for capital, 2 for China Municipalities, and 3 for China Provincial capitals)

NOTE

Some of the country names and city names may be out of date. Please send any corrections to the package maintainer.

Source

The data were originally obtained from Stefan Helders' website (http://www.world-gazetteer.com), which now redirects to http://www.populationmondiale.com. There are no recent updates available.

See Also

map.cities

world2 *Pacific Centric Low resolution World Map*

Description

This is an alternative version of the world database based on latitudes [0, 360), which then has the Pacific Ocean in the centre of the map.

Usage

```
data(world2MapEnv)
```

Format

The data file is merely a character string which specifies the name of an environment variable which contains the base location of the binary files used by the map drawing functions. This environment variable (R_MAP_DATA_DIR_WORLD for the datasets in the maps package) is set at package load time *if it does not already exist*. Hence setting the environment variable before loading the package can override the default location of the binary datasets.

NOTE

This data set is in fact largely obsolete. Often the same (more general) result can be obtained by using wrapping:

map("world", wrap=c(0,360))

This will also work fine with fill=TRUE or any other appropriate longitude interval (e.g. c(-90,270)).

However, world2 is useful when setting xlim to an interval crossing the 180 meridian.

Source

The public domain Natural Earth data set is available from https://www.naturalearthdata.com.

References

Richard A. Becker, and Allan R. Wilks, "Maps in S", *AT&T Bell Laboratories Statistics Research Report [93.2], 1993.*

Richard A. Becker, and Allan R. Wilks, "Constructing a Geographical Database", *AT&T Bell Laboratories Statistics Research Report [95.2], 1995.*

See Also

map, world

Examples

```
map('world2', xlim = c(100, 300))
map.axes()
# xlim is performed before wrapping:
map('world', wrap=c(0,360), xlim = c(100, 300))
# so to emulate "world2":
ww2 <- map('world', wrap=c(0,360), plot=FALSE, fill=TRUE)
map(ww2, xlim = c(100, 300), fill=TRUE)
```

Index