Package ‘OceanView’

December 18, 2019

Version 1.0.5

Title Visualisation of Oceanographic Data and Model Output

Author Karline Soetaert <karline.soetaert@nioz.nl>

Maintainer Karline Soetaert <karline.soetaert@nioz.nl>

Depends plot3D, plot3Drgl, R (>= 3.2)

Imports methods, graphics, grDevices, stats, rgl, shape

Description
Functions for transforming and viewing 2-D and 3-D (oceanographic) data and model output.

License GPL (>= 3.0)

LazyData yes

Repository CRAN

Repository/R-Forge/Project plot3d

Repository/R-Forge/Revision 122

Repository/R-Forge/DateTimeStamp 2019-12-05 15:15:08

Date/Publication 2019-12-18 10:30:02 UTC

NeedsCompilation yes

R topics documented:

`OceanView-package` *Functions for visualising oceanic data sets and model output.*

Description

Visualisation of oceanic data.

Author(s)

Karline Soetaert

References

http://www.rforscience.com/rpackages/visualisation/oceanview/

See Also

db2cross, converts a dataset from database format to cross table.

flowpath, plots velocities as trajectory plot.

remap, transect, extract, mapsigma, transectsigma, mapping and extracting from 2-D or 3-D data.

Mcommon, Mplot, Msplit, functions for plotting matrices.

quiver2D, velocities plotted as arrows.

vectorplot, vector velocity plot.

`Chesapeake data set` *Particle transport in Chesapeake Bay*

Description

`Chesapeake` is a `list` with the bathymetry of Chesapeake Bay, Mid-Atlantic Bight and the initial position of the particles.

`Ltrans` is an `array` with output of the Lagrangian Transport model (Ltrans v.2) from Chesapeake Bay mouth, at 37 dgN in the Mid-Atlantic Bight (Schlag and North, 2012).

Usage

```
data(Chesapeake)
data(Ltrans)
```

Format

- `Chesapeake` is a `list` with the bathymetry of the area. There are 154 x-values, at 77 y-values. It contains:
  - `lon`, the longitude, (154 x 77), dg East.
  - `lat`, the latitude, (154 x 77), dg North.
  - `depth`, the bathymetry (154 x 77), metres.
  - `init`, the initial condition of the particles, a (608 x 4) matrix with (`lon,lat,depth,source`) values.
- `Ltrans` contains output of the Lagrangian particle transport model, in the Chesapeake mouth area. 608 particles were released in two square regions, and their positions followed over 108 output steps. It is an array of dimension (608 x 4 x 108), and which contains for each of the 608 particles, and at each of the 108 output steps the following:
  - `lon`, the longitude of each particle.
  - `lat`, the latitude of each particle.
  - `depth`, the depth of each particle.
  - `source`, the square region of release, either `1` or `2`.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

References

Schlag, Z. R., and E. W. North. 2012. Lagrangian TRANSport model (LTRANS v.2) User's Guide. University of Maryland Center for Environmental Science, Horn Point Laboratory. Cambridge, MD. 183 pp.

North, E. W., E. E. Adams, S. Schlag, C. R. Sherwood, R. He, S. Socolofsky. 2011. Simulating oil droplet dispersal from the Deepwater Horizon spill with a Lagrangian approach. AGU Book Series: Monitoring and Modeling the Deepwater Horizon Oil Spill: A Record Breaking Enterprise.

`http://northweb.hpl.umces.edu/LTRANS.htm`

See Also

Sylt3D for output of a 3-D hydrodynamical model, GETM.

Oxsat for a 3-D data set, package `plot3D`.

tracers2D for plotting time series of tracer distributions in 2D

tracers3D for plotting time series of tracer distributions in 3D

Examples

```
# save plotting parameters
 pm <- par("mfrow")
 mar <- par("mar")

## =============================================================================
## Show bathymetry and initial distribution of particles
```

```
## =============================================================================

 par(mfrow = c(1, 1))

 lon <- Chesapeake$lon
 lat <- Chesapeake$lat
 depth <- Chesapeake$depth
 init  <- Chesapeake$init

 image2D(z = depth, x = lon, y = lat, clab = c("depth", "m"),
   xlab = "lon", ylab = "lat")

# position of particles
 with (init, scatter2D(lon, lat, colvar = source, pch = 16, cex = 0.5,
   col =  c("green", "orange"), add = TRUE, colkey = FALSE))

 par (mar = c(2, 2, 2, 2))
# same, as persp plot
 persp3D(x = lon, y = lat, z = -depth, scale = FALSE,
   expand = 0.02, main = "initial particle distribution",
   plot = FALSE)

 points3D(x = init$lon, y = init$lat, z = -init$depth,
  colvar = init$source, col = c("green", "orange"),
  pch = 16, cex = 0.5,
  add = TRUE, colkey = FALSE, plot = FALSE)

## Not run:
   plotdev(lighting = TRUE, lphi = 45)

## End(Not run)
 plotrgl(lighting = TRUE, smooth = TRUE)

## =============================================================================
## Tracer output in 3D, traditional device
## =============================================================================
## Not run:
 par(mfrow = c(2, 1), mar = c(2, 2, 2, 2))
 for (i in c(50, 100))
   tracers3D(Ltrans[, 1, i], Ltrans[, 2, i], Ltrans[, 3, i],
             colvar = Ltrans[ ,4, i], col = c("green", "orange"),
             pch = 16, cex = 0.5,
             surf = list(x = lon, y = lat, z = -depth, scale = FALSE,
               expand = 0.02, colkey = FALSE, shade = 0.3,
               colvar = depth), colkey = FALSE,
             main = paste("time ", i))

## End(Not run)

## =============================================================================
## Tracer output in 3D, using rgl
## =============================================================================
```

```
 persp3D(x = lon, y = lat, z = -depth, colvar = depth, scale = FALSE,
   expand = 0.02, main = "particle distribution", plot = FALSE)

 plotrgl(lighting = TRUE, smooth = TRUE)

# you may zoom to the relevant region, or cut a region
# cutrgl()
 for (i in seq(1, 108, by = 4)) {
   tracers3Drgl(Ltrans[, 1, i], Ltrans[, 2, i], Ltrans[, 3, i],
             colvar = Ltrans[ ,4, i], col = c("green", "orange"),
             main = paste("time ", i))
# remove # to slow down
#   Sys.sleep(0.1)
 }

# using function moviepoints3D
## Not run:
 persp3Drgl(x = lon, y = lat, z = -depth, colvar = depth, scale = FALSE,
   expand = 0.02, main = "particle distribution",
   lighting = TRUE, smooth = TRUE)

 nt <- dim(Ltrans)[3]  # number of time points
 np <- dim(Ltrans)[1]  # number of particles

 times <- rep(1:nt, each = np)

 moviepoints3D(x = Ltrans[, 1, ], y = Ltrans[, 2, ], z = Ltrans[, 3, ],
               t = times, colvar = Ltrans[ ,4, ], col = c("green", "orange"),
               cex = 5, ask = TRUE)


## End(Not run)
## =============================================================================
## Tracer output in 2D, traditional device
## =============================================================================

 par(mfrow = c(2, 2))
 for (i in seq(10, 106, length.out = 4))
   tracers2D(Ltrans[, 1, i], Ltrans[, 2, i],
             colvar = Ltrans[ ,4, i], col = c("green", "orange"),
             pch = 16, cex = 0.5,
             image = list(x = lon, y = lat, z = depth), colkey = FALSE,
             main = paste("time ", i))

## =============================================================================
## Tracer output in 2D, rgl
## =============================================================================

 image2Drgl (x = lon, y = lat, z = depth)
 for (i in seq(1, 108, by = 3)) {
   tracers2Drgl(Ltrans[, 1, i], Ltrans[, 2, i],
             colvar = Ltrans[ ,4, i], col = c("green", "orange"))
# remove # to slow down
```

```
#   Sys.sleep(0.1)
 }

# reset plotting parameters
 par(mar = mar)
 par(mfrow = pm)
```

Map and extract data *Functions for remapping, changing the resolution, and extracting from 2-D or 3-D data.*

Description

S3 functions remap maps a variable (var) (a matrix or array) with x, y (and z) coordinates to a matrix or array with coordinates given by xto, yto (and zto). x,y,z,xto,yto and zto are all vectors. The functions interpolate to all combinations of xto,yto and zto. Simple 2-D linear interpolation is used. Result is a matrix or array.

Function changeres changes the resolution of a variable (var) (a matrix or array) with x, y (and z) coordinates. If var is a matrix, then x,y can be either a vector or a matrix; if var is an array, then x,y,z should all be vectors. Simple 2-D linear interpolation is used. Result is a matrix or array.

S3-functions extract map a variable (var) from a matrix with (x, y) coordinates or from an array with (x, y, z) coordinates to the xy coordinate *pair* xyto or xyz coordinate *triplets* xyzto by linear interpolation. Result is a vector.

transect takes a cross section across an array (var). Result is a matrix.

mapsigma maps a matrix or array var containing values defined at (x, sigma) (or (x, y, sigma)) coordinates to (x, depth) (or (x, y, depth)) coordinates. The depths corresponding to the sigma values in var are in an input matrix or array called sigma with same dimensions as var. The result is a matrix or array which will contain NAs where the depth-coordinates extend beyond the sigma values.

Usage

```
remap        (var, ...)

## S3 method for class 'matrix'
remap(var, x, y, xto = NULL, yto = NULL,
          na.rm = TRUE, ...)

## S3 method for class 'array'
remap(var, x, y, z, xto = NULL, yto = NULL, zto = NULL,
          na.rm = TRUE, ...)

changeres    (var, ...)

## S3 method for class 'matrix'
changeres(var, x, y, resfac, na.rm = TRUE, ...)
```

```
## S3 method for class 'array'
changeres(var, x, y, z, resfac, na.rm = TRUE, ...)

extract    (var, ...)

## S3 method for class 'matrix'
extract(var, x, y, xyto, ...)

## S3 method for class 'array'
extract(var, x, y, z, xyzto, ...)

transect(var, x, y, z, to, margin = "xy", ...)

mapsigma   (var, ...)

## S3 method for class 'matrix'
mapsigma(var = NULL, sigma, signr = 2, x = NULL,
    depth = NULL, numdepth = NULL, xto = NULL, resfac = 1, ...)

## S3 method for class 'array'
mapsigma(var = NULL, sigma, signr = 3, x = NULL, y = NULL,
    depth = NULL, numdepth = NULL, xto = NULL, yto = NULL,
    resfac = 1, ...)

transectsigma(var = NULL, sigma, x, y, to, depth = NULL,
                numdepth = NULL, resfac = 1, ...)
```

Arguments

`var`	Matrix or array with values to be mapped to other coordinates (`remap`), or to lower or higher resolution (`changeres`), or whose values have to be extracted (`extract`, `transect`), or which has to be mapped from sigma to depth coordinates (`mapsigma`). For `transect` and `transectsigma`, `var` has to be an array.
`x`	Vector with original x-coordinates of the matrix or array `var` to be mapped. Length should be = first dimension of `var`.
`y`	Vector with original y-coordinates of the matrix or array `var` to be mapped. Length should be = second dimension of `var`.
`z`	Vector with original z-coordinates of the array `var` to be mapped. Length should be = third dimension of `var`.
`xto`	Vector with x-coordinates to which `var` should be mapped. The elements in `xto` should be embraced by the elements in `x` (it is not allowed to extrapolate outside of the region). If `NULL` then the range of `x` is covered, with the same number of points.
`yto`	Vector with y-coordinates to which `var` should be mapped. The elements in `yto` should be embraced by the elements in `y` (it is not allowed to extrapolate outside

of the region). If NULL then the range of y is covered, with the same number of points.

zto
: Vector with z-coordinates to which var should be mapped. The elements in zto should be embraced by the elements in z (it is not allowed to extrapolate outside of the region). If NULL then the range of z is covered, with the same number of points.

xyto
: Two-columned matrix, with first and second column specifying the x- respectively y-coordinates to which the matrix var should be mapped. The elements should be embraced by the elements in x (first column) and y (second column) (it is not allowed to extrapolate outside of the region).

xyzto
: Three-columned matrix, specifying the x-, y- and z-coordinates to which the array var should be mapped. The elements should be embraced by the elements in x, y and z (it is not allowed to extrapolate outside of the region).

to
: Two-columned matrix, specifying the values along the margin coordinates of the transect to be taken on the array var. The elements should be embraced by the elements in x, y and z (it is not allowed to extrapolate outside of the region).

margin
: String with the names of the coordinates in the matrix to, and along which the transect is to be taken on the array var. One of "xy","xz","yz". If "xy", then the first and second column in input to represent x and y values respectively, and the transect will select all z values corresponding with these inputs.

sigma
: The sigma coordinates, a matrix or array with the same dimension as var. The sigma coordinates should refer to the column as defined by signr.

signr
: The position of the sigma coordinates, in the matrix or array. The default is the second or third dimension in var for a matrix and array respectively.

depth
: The depth (often referred to as 'z') coordinates to which matrix var has to be mapped. If NULL then seq(min(sigma),max(sigma),length.out = numdepth).

numdepth
: Only used when depth= NULL, the length of the depth vector to which the matrix var has to be mapped. If NULL then the length will be equal to ncol(var) (if var is a matrix), or dim(var)[3] in case var is an array.

resfac
: Resolution factor, one value or a vector of two or three numbers, for the x, y- and z- values respectively. A value > 1 will increase the resolution. For instance, if resfac equals 3 then for each adjacent pair of x- and y- values, var will be interpolated to two intermediary points. This uses simple linear interpolation. If resfac is one number then the resolution will be increased similarly in x, y- and z-direction. In case of mapsigma, resfac is overruled if xto, yto or zto is specified.

na.rm
: How to treat NAs in the matrix or array var. If TRUE, they are ignored while interpolating; this will make the size of NA regions smaller; if FALSE, the size of the NA region will increase.

...
: any other arguments.

Details

S3-function remap can be used to increase or decrease the resolution of a matrix or array var, or to zoom in on a certain area. It returns an object of the same class as var (i.e. a matrix or array).

S3-function `transect` takes a slice from an array; it returns a matrix.

S3-function `extract` returns a vector with one value corresponding to each row in `xyto` or `xyzto`.

`mapsigma` should be used to make images from data that are in sigma coordinates.

Value

`remap.matrix`:

- var The higher or lower resolution matrix with dimension = c(length(xto), length(yto)).
- x The x coordinates, corresponding to first dimension of `var` (input argument `xto`).
- y The y coordinates, corresponding to second dimension of `var` (input argument `yto`).

`remap.array`:

- var The higher or lower resolution array, with dimension = c(length(xto), length(yto), length(zto)).
- x The x coordinates, corresponding to first dimension of `var` (input argument `xto`).
- y The y coordinates, corresponding to second dimension of `var` (input argument `yto`).
- z The z coordinates, corresponding to third dimension of `var` (input argument `zto`).

`extract.matrix`:

- var The higher or lower resolution object, with dimension = c(nrow(xyto), dim(var)[3]).
- xy The pairs of (x,y) coordinates (input argument `xyto`).

`extract.array`:

- var The higher or lower resolution object, with dimension = c(nrow(xyzto), dim(var)[3]).
- xyz The triplets of (x,y,z) coordinates (input argument `xyzto`).

`mapsigma`:

- var A matrix with columns in depth-coordinates.
- depth The depth-coordinates, also known as 'z'-coordinates, referring to the dimension of `var` as specified by `signr`.
- x The 'x'-coordinates referring to the first dimension of `var`, except for the depth.
- y Only if `var` is an array, the 'y'-coordinates referring to the second dimension of `var`, except for the depth.

See Also

Sylt3D for other examples of mapping.

Examples

```
# save plotting parameters
 pm <- par("mfrow")

## =======================================================================
## Simple examples
## =======================================================================
 M <- matrix(nrow = 2, data = 1:4)
 remap(M, x = 1:2, y = 1:2,
   xto = seq(1, 2, length.out = 3), yto = 1:2)

 changeres(M, x = 1:2, y = 1:2, resfac = c(2, 1))
 changeres(M, x = 1:2, y = 1:2, resfac = 2)

# x and or y are a matrix.
 changeres(var = M, x = M, y = 1:2, resfac = c(2, 1))
 changeres(M, x = M, y = 1:2, resfac = 2)


## =======================================================================
## Use remap to add more detail to a slice3D plot
## =======================================================================

 par(mfrow = c(1, 1))
 x <- y <- z <- seq(-4, 4, by = 0.5)
 M <- mesh(x, y, z)

 R <- with (M, sqrt(x^2 + y^2 + z^2))
 p <- sin(2*R) /(R+1e-3)

 slice3D(x, y, z, ys = seq(-4, 4, by = 2), theta = 85,
   colvar = p, pch = ".", clim = range(p))

 xto <- yto <- zto <- seq(-1.2, 1.2, 0.3)
 Res <- remap (p, x, y, z, xto, yto, zto)

# expand grid for scatterplot
 Mt  <- mesh(Res$x, Res$y, Res$z)

 scatter3D(x = Mt$x, y = Mt$y, z = Mt$z, colvar = Res$var,
   pch = ".", add = TRUE, cex = 3, clim = range(p))

# same in rgl:
 ## Not run:
   plotrgl()

## End(Not run)

# extract specific values from 3-D data
 xyzto <- matrix(nrow = 2, data = c(1, 1, 1, 2, 2, 2), byrow = TRUE)
 extract(var = p, x, y, z, xyzto = xyzto)
```

```
# a transect
 to <- cbind(seq(-4, 4, length.out = 20), seq(-4, 4, length.out = 20))
 image2D( transect(p, x, y, z, to = to)$var)

## =======================================================================
## change the resolution of a 2-D image
## =======================================================================

 par(mfrow = c(2, 2))
 nr <- nrow(volcano)
 nc <- ncol(volcano)

 x  <- 1 : nr
 y  <- 1 : nc
 image2D(x = x, y = y, volcano, main = "original")

# increasing the resolution
 x2 <- seq(from = 1, to = nr, by = 0.5)
 y2 <- seq(from = 1, to = nc, by = 0.5)

 VOLC1 <- remap(volcano, x = x, y = y, xto = x2, yto = y2)$var
 image2D(x = x2, y = y2, z = VOLC1, main = "high resolution")

# low resolution
 xb <- seq(from = 1, to = nr, by = 2)
 yb <- seq(from = 1, to = nc, by = 3)
 VOLC2 <- remap(volcano, x, y, xb, yb)$var
 image2D(VOLC2, main = "low resolution")

# zooming in high resolution
 xc <- seq(10, 40, 0.1)
 yc <- seq(10, 40, 0.1)

 VOLC3 <- remap(volcano,x, y, xc, yc)$var
 image2D(VOLC3, main = "zoom")

# Get one value or a grid of values
 remap(volcano, x, y, xto = 2.5,     yto = 5)
 remap(volcano, x, y, xto = c(2, 5), yto = c(5, 10))

# Specific values
 extract(volcano, x, y, xyto = cbind(c(2, 5), c(5, 10)))

## =======================================================================
## take a cross section or transect of volcano
## =======================================================================

 par(mfrow = c(2, 1))
 image2D(volcano, x = 1:nr, y = 1:nc)
 xyto <- cbind(seq(from = 1,  to = nr, length.out = 20),
               seq(from = 20, to = nc, length.out = 20))
 points(xyto[,1], xyto[,2], pch = 16)
```

```
 (Crossection <- extract (volcano, x = 1:nr, y = 1:nc,
                             xyto = xyto))

 scatter2D(xyto[, 1], Crossection$var, colvar = Crossection$var,
   type = "b", cex = 2, pch = 16)

## =======================================================================
##  mapsigma: changing from sigma coordinates into depth-coordinates
## =======================================================================

 par(mfrow = c(2, 2))
 var <- t(matrix (nrow = 10, ncol = 10, data = 1:10))
 image2D(var, ylab = "sigma", main = "values in sigma coordinates",
       clab = "var")

 # The depth at each 'column'
 Depth <- approx(x = 1:5, y = c(10, 4, 5, 6, 4),
                 xout = seq(1,5, length.out = 10))$y

# Sigma coordinates
 sigma <- t(matrix(nrow = 10, ncol = 10, data = Depth, byrow = TRUE) *
                seq(from = 0, to = 1, length = 10))
 matplot(sigma, type = "l", main = "sigma coordinates",
         xlab = "sigma", ylab = "depth", ylim = c(10, 0))

# Mapping to the default depth coordinates
 varz <- mapsigma(var = var, sigma = sigma)
 image2D(varz$var, y = varz$depth, NAcol = "black", ylim = c(10, 0),
       clab = "var", ylab = "depth",
       main = "depth-coord, low resolution")

# Mapping at higher resolution of depth coordinates
 varz <- mapsigma(var, sigma = sigma, resfac = 10)
 image2D(varz$var, y = varz$depth, NAcol = "black", ylim = c(10, 0),
       clab = "var", ylab = "depth",
       main = "depth-coord, high resolution")

## =======================================================================
##  mapsigma: mapping to depth for data Sylttran (x, sigma, time)
## =======================================================================

# depth values
 D <- seq(-1, 20, by = 0.5)
 dim(Sylttran$visc)

# sigma coordinates are the second dimension (signr)
# resolution is increased for 'x' and decreased for 'time'

 visc <- mapsigma(Sylttran$visc, x = Sylttran$x, y = Sylttran$time,
  sigma = Sylttran$sigma, signr = 2, depth = D, resfac = c(2, 1, 0.4))

# changed dimensions
 dim(visc$var)
```

```
 image2D(visc$var, x = visc$x, y = -visc$depth, ylim = c(-20, 1),
  main = paste("eddy visc,", format(visc$y, digits = 2), " hr"),
  ylab = "m", xlab = "x", clab = c("","m2/s"),
  clim = range(visc$var, na.rm = TRUE))

 par(mfrow = c(1, 1))
# make depth the last dimension
 cv <- aperm(visc$var, c(1, 3, 2))

# visualise as slices
 slice3D(colvar = cv, x = visc$x, y = visc$y, z = -visc$depth,
   phi = 10, theta = 60, ylab = "time",
   xs = NULL, zs = NULL, ys = visc$y, NAcol = "transparent")

# restore plotting parameters
 par(mfrow = pm)
```

`Matrix plotting` *Functions for plotting matrices, or for splitting them and for maing suitable summaries*

Description

`Mplot` plots data from (a list of) matrices.

`Msplit` splits a matrix in a list according to factors (or unique values).

`Mcommon` creates a list of matrices that have only common variables.

`Msummary` and `Mdescribe` create suitable summaries of all columns of a matrix or list.

Usage

```
Mplot (M, ..., x = 1, select = NULL, which = select,
       subset = NULL, ask = NULL,
       legend = list(x = "center"), pos.legend = NULL,
       xyswap = FALSE, rev = "")

Msummary (M, ...,
       select = NULL, which = select,
       subset = NULL)

Mdescribe (M, ...,
       select = NULL, which = select,
       subset = NULL)

Msplit (M, split = 1, subset = NULL)

Mcommon (M, ..., verbose = FALSE)
```

Arguments

`M`	Matrix or data.frame to be plotted, or treated. For `Mplot`, `M` can be a list with matrices or data.frames.
`x`	Name or number of the column to be used as the x-values.
`select`	Which variable/columns to be selected. This is added for consistency with the R-function `subset`.
`which`	The name(s) or the index to the variables that should be plotted or selected. Default = all variables, except `time`.
`subset`	Logical expression indicating elements or rows to keep in `select`: missing values are taken as `FALSE`
`ask`	Logical; if `TRUE`, the user is *ask*ed before each plot, if `NULL` the user is only asked if more than one page of plots is necessary and the current graphics device is set interactive, see `par(ask)` and `dev.interactive`.
`legend`	A `list` with parameters for the legend to be added. If `FALSE`, then no legend will be drawn.
`pos.legend`	The position of the legend, a number. The default is to put the legend in the last figure. Also allowed is `pos.legend = 0`, which will create a new figure with only the legend.
`xyswap`	If `TRUE`, then the x- and y-values will be swapped.
`rev`	a character string which contains "x" if the x axis is to be reversed, "y" if the y axis is to be reversed and "xy" or "yx" if both axes are to be reversed.
`split`	The name or number of the column with the factor according to which the matrix will be split.
`verbose`	If `TRUE` will write output to the screen.
`...`	Additional arguments passed to the methods. For `Mplot`: can also be extra matrices to plot. The arguments after ... must be matched exactly.

Value

Function `Msplit` returns a list with the matrices, split according to the factors; the names of the elements is set by the factor's name. It is similar to the R-function split.

Function `Mcommon` returns a list with the matrices, which only have the common variables.

Function `Msummary` returns a data.frame with summary values (minimum, first quantile, median, mean, 3rd quantile, maximum) for each column of the input (variable). If there are more than one object to be summarised, or if M is a list of objects, the name of the object is in the second column.

Function `Mdescribe` returns a data.frame with summary values (number of data, number of missing values, number of unique values, mean value, the standard deviation, the minimum, the p = 0.05, 0.1, 0.5, 0.9, 0.95 quantiles, and the maximum) for each column of the input (variable). If there are more than one object to be summarised, or if M is a list of objects, the name of the object is in the second column.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

Examples

```
# save plotting parameters
 pm <- par("mfrow")

## =======================================================================
## Create three dummy matrices
## =======================================================================

 M1 <- matrix(nrow = 10, ncol = 5, data = 1:50)
 colnames(M1) <- LETTERS[1:5]

 M2 <- M1[, c(1, 3, 4, 5, 2)]
 M2[ ,-1] <- M2[,-1] /2
 colnames(M2)[3] <- "CC"  # Different name

 M3 <- matrix(nrow = 5, ncol = 4, data = runif(20)*10)
 M3[,1] <- sort(M3[,1])
 colnames(M3) <- colnames(M1)[-3]

# show them
 head(M1); head(M2); head(M3)
 Msummary(M1)
 Msummary(M1, M2, M3)

# plot all columns of M3 -  will change mfrow
 Mplot(M3, type = "b", pch = 18, col = "red")

# plot results of all three data sets
 Mplot(M1, M2, M3, lwd = 2, mtext = "All variables versus 1st column",
       legend = list(x = "top", legend = c("M1", "M2", "M3")))


## =======================================================================
## Plot a selection or only common elements
## =======================================================================

 Mplot(M1, M2, M3, x = "B", select = c("A", "E"), pch = c(NA, 16, 1),
       type = c("l", "p", "b"), col = c("black", "red", "blue"),
       legend = list(x = "right", legend = c("M1", "M2", "M3")))

 Mplot(Mcommon(M1, M2, M3), lwd = 2, mtext = "common variables",
       legend = list(x = "top", legend = c("M1", "M2", "M3")))

 Mdescribe(Mcommon(M1, M2, M3))

## =======================================================================
## The iris and Orange data set
## =======================================================================

# Split the matrix according to the species
 Irislist <- Msplit(iris, split = "Species")
 names(Irislist)
```

```
 Mdescribe(Irislist, which = "Sepal.Length")
 Mdescribe(iris, which = "Sepal.Length", subset = Species == "setosa")

# legend in a separate plot
 Mplot(Irislist, type = "p", pos.legend = 0,
       legend = list(x = "center", title = "species"))

 Mplot(Msplit(Orange,1), lwd = 2,
       legend = list(x = "topleft", title = "tree nr"))
 Msummary(Msplit(Orange,1))

# reset plotting parameters
 par(mfrow = pm)
```

`Moving slices in 3D` *Plotting volumetric data as moving slices in 3D using rgl*

Description

`movieslice3D` plots 3D volumetric data as slices moving in one direction in open-GL graphics.

It is based on the `plot3Drgl` function slice3Drgl.

Usage

```
movieslice3D (x, y, z, colvar = NULL, xs = NULL,
  ys = NULL, zs = NULL, along = NULL,
  col = jet.col(100), NAcol = "white", breaks = NULL,
  colkey = FALSE, clim = NULL, clab = NULL,
  wait  = NULL, ask = FALSE, add = FALSE, basename = NULL, ...)
```

Arguments

`x, y, z`	Vectors with x, y and z-values. They should be of length equal to the first, second and third dimension of `colvar` respectively.
`colvar`	The variable used for coloring. It should be an array of dimension equal to `c(length(x),length(y),length(z))`. It must be present.
`col`	Colors to be used for coloring the `colvar` variable. If `col` is `NULL` then a red-yellow-blue colorscheme (jet.col) will be used.
`NAcol`	Colors to be used for `colvar` values that are `NA`.
`breaks`	a set of finite numeric breakpoints for the colors; must have one more breakpoint than color and be in increasing order. Unsorted vectors will be sorted, with a warning.
`colkey`	A logical, `NULL` (default), or a `list` with parameters for the color key (legend). If `colkey = NULL` then a color key will be added only if `col` is a vector. Setting `colkey = list(plot = FALSE)` will create room for the color key without drawing it. if `colkey = FALSE`, no color key legend will be added.

`clim`	Only if `colvar` is specified, the range of the color variable values. Values of `colvar` that extend the range will be put to `NA` and colored as specified with `NAcol`.
`clab`	Only if `colkey` is not `NULL` or `FALSE`, the label to be written on top of the color key. The label will be written at the same level as the main title. To lower it, `clab` can be made a vector, with the first values empty strings.
`xs, ys, zs`	Vectors specify the positions in x, y or z where the slices (planes) are to be drawn consecutively. The movie will loop over the slices, each time projecting the values of `colvar` on them. If all `xs,ys,zs` are `NULL`, then `xs` will be taken equal to `x`.
`along`	A number 1, 2, 3 denoting the dimension over which the slices are to be moved. If `NULL`, then the dimension will be the one corresponding to the longest vector `xs,ys,zs`.
`add`	Logical. If `TRUE`, then the slices will be added to the current plot. If `FALSE` a new plot is started.
`ask`	Logical. If `TRUE`, then the new slice will only be drawn after a key has been struck. If `FALSE`, redrawing will depend on `wait`
`wait`	The time interval inbetween drawing of a new slice, in seconds. If `NULL`, the drawing will not be suspended.
`basename`	The base name of a png file to be produced for each movieframe.
`...`	additional arguments passed to slice3D from package `plot3D` or to `plotrgl` from package `plot3Drgl`.

Value

returns nothing

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

See Also

Sylt3D for a data set that can be displayed with `movieslice3D`

moviepoints3D for plotting moving points in 3D

Examples

```
x <- y <- z <- seq(-1, 1, by = 0.1)
grid   <- mesh(x, y, z)
colvar <- with(grid, x*exp(-x^2 - y^2 - z^2))

movieslice3D  (x, y, z, colvar = colvar, ticktype = "detailed")
```

Moving surfaces in 3D *Plotting moving surfaces in 3D using rgl*

Description

`moviepersp3D` plots moving perspective plots of a surface in open-GL.

It is based on the `plot3Drgl` function persp3Drgl.

Usage

```
moviepersp3D (z, x = NULL, y = NULL, t = NULL, colvar = z, tdim = 1,
  col = jet.col(100), NAcol = "white", breaks = NULL,
  colkey = FALSE, clim = NULL, clab = NULL,
  wait  = NULL, ask = FALSE, add = FALSE, basename = NULL, ... )
```

Arguments

`x, y, t`	Vectors with x, y and t-values. Their position in the z-array depends on `tdim`.
`z`	Three-dimensional array with the z-values to be plotted.
`tdim`	Index to where the `time` variable (over which the plot will loop) is to be found in `z` and `colvar`. The default is the first position, so that `z` and `colvar` are of dimension `(length(t),length(x),(length(y)`.
`colvar`	The variable used for coloring. It should be an array of dimension equal to the dimension of `z`. It need not be present.
`col`	Colors to be used for coloring the `colvar` variable. If `col` is `NULL` then a red-yellow-blue colorscheme (jet.col) will be used.
`NAcol`	Colors to be used for `colvar` values that are `NA`.
`breaks`	A set of finite numeric breakpoints for the colors; must have one more breakpoint than color and be in increasing order. Unsorted vectors will be sorted, with a warning.
`colkey`	A logical, `NULL` (default), or a `list` with parameters for the color key (legend). If `colkey = NULL` then a color key will be added only if `col` is a vector. Setting `colkey = list(plot = FALSE)` will create room for the color key without drawing it. if `colkey = FALSE`, no color key legend will be added.
`clim`	Only if `colvar` is specified, the range of the color variable values. Values of `colvar` that extend the range will be put to `NA` and colored as specified with `NAcol`.
`clab`	Only if `colkey` is not `NULL` or `FALSE`, the label to be written on top of the color key. The label will be written at the same level as the main title. To lower it, `clab` can be made a vector, with the first values empty strings.
`add`	Logical. If `TRUE`, then the slices will be added to the current plot. If `FALSE` a new plot is started.
`ask`	Logical. If `TRUE`, then the new slice will only be drawn after a key has been struck. If `FALSE`, redrawing will depend on `wait`

`wait`	The time interval inbetween drawing of a new slice, in seconds. If `NULL`, the drawing will not be suspended.
`basename`	The base name of a png file to be produced for each movieframe.
`...`	additional arguments passed to persp3Drgl from package `plot3Drgl`.

Value

returns nothing

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

See Also

Sylt3D for a data set that can be displayed with `moviepersp3D`

moviepoints3D for plotting moving points in 3D

movieslice3D for plotting moving slices in 3D

Examples

```
 x <- y <- t <- seq(-1, 1, by = 0.1)
 grid   <- mesh(x, y, t)
 z <- with(grid, x*exp(-x^2 - y^2 - z^2))

 moviepersp3D  (x, y, z = z, colvar = z, colkey = TRUE,
   ticktype = "detailed", wait = 0.1, main = "t = ")

## Not run:
 moviepersp3D (x, y, z = z, colvar = z, colkey = TRUE,
   aspect = TRUE, bty = "n", ask = FALSE, main = "t = ")

## End(Not run)
```

`NIOZ Westerschelde monitoring`

NIOZ monitoring data of Westerschelde estuary.

Description

Part of the long-term monitoring data of the Westerschelde estuary, from 1996 till 2004.

A total of 17 stations were monitored on a monthly basis.

The dataset `WSnioz` is in long format and contains the following variables: oxygen, temperature, salinity, nitrate, ammonium, nitrite, phosphate, silicate and chlorophyll.

The dataset `WSnioz.table` is in tabular format.

The full dataset can be downloaded from: `https://www.nioz.nl/monitoring-data-downloads`

Usage

```
data(WSnioz)
data(WSnioz.table)
```

Format

WSnioz is a data.frame with the following columns:

- SamplingDateTime, a string with the date and time of sampling.
- SamplingDateTimeREAL, a numeric value with day as per 1900.
- Station, the station number.
- Latitude, Longitude, the station position.
- VariableName, the variable acronym.
- VariableDesc, description of the variable.
- VariableUnits, units of measurement.
- DataValue, the actual measurement.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

References

Soetaert, K., Middelburg, JJ, Heip, C, Meire, P., Van Damme, S., Maris, T., 2006. Long-term change in dissolved inorganic nutrients in the heterotrophic Scheldt estuary (Belgium, the Netherlands). Limnology and Oceanography 51: 409-423. DOI: 10.4319/lo.2006.51.1_part_2.0409

http://aslo.org/lo/toc/vol_51/issue_1_part_2/0409.pdf

See Also

image2D for plotting images, package plot3D.

ImageOcean for an image of the ocean's bathymetry, package plot3D.

scatter2D for making scatterplots, package plot3D.

Oxsat for a 3-D data set, package plot3D.

Examples

```
# save plotting parameters
 pm <- par("mfrow")
 mar <- par("mar")

## =============================================================================
## Show stations and measured variables
## =============================================================================
 unique(WSnioz[,c("Station", "Latitude", "Longitude")])
 unique(WSnioz[,c("VariableName", "VariableDesc")])
```

```
## =============================================================================
## An image for Nitrate:
## =============================================================================

# 1. use db2cross to make a cross table of the nitrate data
# assume that samples that were taken within 5 days belong to the same
# monitoring campaign (df.row).

 NO3 <- db2cross(WSnioz, row = "SamplingDateTimeREAL",
       col = "Station", val = "DataValue",
       subset = (VariableName == "WNO3"), df.row = 5)

# 2. plot the list using image2D; increase resolution
 image2D(NO3, resfac = 3)

## =============================================================================
## All timeseries for one station
## =============================================================================

 st1 <- db2cross(WSnioz, row = "SamplingDateTimeREAL",
       col = "VariableName", val = "DataValue",
       subset = (WSnioz$Station == 1), df.row = 5)

 Mplot(cbind(st1$x/365+1900,st1$z))

## =============================================================================
## All timeseries for multiple stations
## =============================================================================

 dat <- NULL
 for (st in 1:17) {
   dd <- db2cross(WSnioz, row = "SamplingDateTimeREAL",
          col = "VariableName", val = "DataValue",
          subset = (WSnioz$Station == st), df.row = 5)
   dat <- rbind(dat,  cbind(st, time = dd$x/365+1900, dd$z))
 }

# select data for station 1, 17
 dat2 <- Msplit(dat, split = "st", subset = st %in% c(1, 17))
 names(dat2)

 Mplot(dat2, lty = 1)

## =============================================================================
## tabular format of the same data
## =============================================================================
 head(WSnioz.table)

# plot all data from station 1:
 Mplot(WSnioz.table, select = 3:11, subset = Station == 1, legend = FALSE)

 Mplot(Msplit(WSnioz.table, "Station", subset = Station %in% c(1, 13)) ,
   select = c("WNO3", "WNO2", "WNH4", "WO2"), lty = 1, lwd = 2,
```

```
   xlab = "Daynr", log = c("y", "y", "y", ""),
   legend = list(x = "left", title = "Station"))


# reset plotting parameters
 par(mar = mar)
 par(mfrow = pm)
```

`Profile data set` *Temperature profiles made along a ship track.*

Description

Profiles of temperature made along a ship track, originally made available by US NOAA NODC.

The data were merged from 29 input files named `gtspp_103799_xb_111.nc` till `gtspp_103827_xb_111.nc`.

These data were acquired from the US NOAA National Oceanographic Data Center (NODC) on 9/06/2012 from http://www.nodc.noaa.gov/gtspp/.

Usage

```
data(TrackProf)
```

Format

list with

- `meta`, a `data.frame` with the metadata, containing for each of the 29 profiles the following:
  - `station`, the number of the station (part of the original filename).
  - `filename`, the original name of the NetCDF file.
  - `date`, the date of sampling.
  - `time`, the time of sampling, a number relative to 1-1-1900 0 hours.
  - `longitude`, dg E.
  - `latitutde`, dg N.
- `temp`, the seawater temperature, at the `depth` of the measurement in dg C. A matrix of dimension `(29,93)` for the 29 profiles and (at most) 93 depth values; `NA` means no measurement.
- `depth`, the depth of the measurement in `temp`, in metres, positive downward. A matrix of dimension `(29,93)` for the 29 profiles and (at most) 93 depth values; `NA` means no measurement.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

References

http://www.nodc.noaa.gov/gtspp/

U.S. National Oceanographic Data Center: Global Temperature-Salinity Profile Programme. June 2006. U.S. Department of Commerce, National Oceanic and Atmosphere Administration, National Oceanographic Data Center, Silver Spring, Maryland, 20910. Date of Access: 9/06/2012.

See Also

image2D for plotting images, package `plot3D`.

ImageOcean for an image of the ocean bathymetry, package `plot3D`.

scatter2D for making scatterplots, package `plot3D`.

Oxsat for a 3-D data set, package `plot3D`.

Examples

```
# save plotting parameters
 pm <- par(mfrow = c(2, 2))
 mar <- par("mar")

## =============================================================================
## show the metadata
## =============================================================================
 print(TrackProf$meta)

## =============================================================================
## display the cruisetrack on the Ocean Bathymetry data
## =============================================================================
 # 1. plots the ocean's bathymetry and add sampling positions
 ImageOcean(xlim = c(-50, 50), ylim = c(-50, 50),
            main = "cruise track")
 points(TrackProf$meta$longitude, TrackProf$meta$latitude, pch = "+")

# mark starting point
 points(TrackProf$meta$longitude[1], TrackProf$meta$latitude[1],
        pch = 18, cex = 2, col = "purple")

## =============================================================================
## image plots of raw data
## =============================================================================

 image2D(z = TrackProf$depth, main = "raw depth values",
       xlab = "station nr", ylab = "sample nr", clab = "depth")
 image2D(z = TrackProf$temp, main = "raw temperature values",
       xlab = "station nr", ylab = "sample nr", clab = "dgC")

## =============================================================================
## image plots of temperatures at correct depth
## =============================================================================

# water depths to which data set is interpolated
```

```
 depth <- 0 : 809

# map from "sigma" to "depth" coordinates
 Temp_Depth <- mapsigma (TrackProf$temp, sigma = TrackProf$depth,
   depth = depth)$var

# image with depth increasing downward and increased resolution (resfac)
 image2D(z = Temp_Depth, main = "Temperature-depth",
       ylim = c(809, 0), y = depth, NAcol ="black", resfac = 2,
       xlab = "station nr", ylab = "depth, m", clab = "dgC")

## =============================================================================
## scatterplot of surface values on ocean bathymetry
## =============================================================================

 par(mar = mar + c(0, 0, 0, 2))
 par(mfrow = c(1, 1))

# No colors, but add contours
 ImageOcean(xlim = c(-30, 30), ylim = c(-40, 40),
          main = "cruise track", col = "white", contour = TRUE)

# use data set TrackProf to add measured temperature, with color key
 with (TrackProf,
   scatter2D(colvar = temp[,1], x = meta[ ,"longitude"],
         y = meta[ ,"latitude"], clab = "temp",
         add = TRUE, pch = 18, cex = 2))

# reset plotting parameters
 par(mar = mar)
 par(mfrow = pm)
```

Quiver and flow paths *Plots velocities as arrows or as trajectory plots.*

Description

Function quiver2D displays velocity vectors as arrows, using ordinary graphics.

Function quiver2Drgl displays velocity vectors as arrows using rgl.

Function flowpath displays the flow paths of particles, based on velocity vectors.

Usage

```
quiver2D(u, ...)

## S3 method for class 'matrix'
quiver2D(u, v, x = NULL, y = NULL,
            colvar = NULL, ...,
            scale = 1, arr.max = 0.2, arr.min = 0, speed.max = NULL,
```

```
            by = NULL, type = "triangle", col = NULL, NAcol = "white",
            breaks = NULL, colkey = NULL, mask = NULL,
            image = FALSE, contour = FALSE,
            clim = NULL, clab = NULL,
            add = FALSE, plot = TRUE)

## S3 method for class 'array'
 quiver2D(u, v, margin = c(1, 2), subset, ask = NULL, ...)

quiver2Drgl (u, v, x = NULL, y = NULL, colvar = NULL, ...,
            scale = 1, arr.max = 0.2, arr.min = 0, speed.max = NULL,
            by = NULL, type = "triangle",
            col = NULL, NAcol = "white", breaks = NULL,
            mask = NULL, image = FALSE, contour = FALSE,
            colkey = NULL, clim = NULL, clab = NULL, add = FALSE, plot = TRUE)

flowpath(u, v, x = NULL, y = NULL, startx = NULL, starty = NULL, ...,
            scale = 1, numarr = 0, arr.length = 0.2, maxstep = 1000,
            add = FALSE, plot = TRUE)
```

Arguments

`u`	A matrix (`quiver2D`) or array (`quiver2D.array`) with velocities in x-direction. For `quiver2D` the number of rows should be = Nx or Nx+1 (Nx = length(x), if x given), the number of columns should be = Ny or Ny+1 (Ny = length(y), if y given).
`v`	A matrix (`quiver2D`) or array (`quiver2D.array`) with velocities in y-direction. For `quiver2D` the number of rows should be = Nx or Nx+1, the number of columns should be = Ny or Ny+1.
`x`	Vector with x-coordinates of the velocities. If `NULL`, it is taken to be a sequence between (0, 1), and with length = `nrow(u)`.
`y`	Vector with y-coordinates of the velocities. If `NULL`, it is taken to be a sequence between (0, 1), and with length = `ncol(v)`.
`startx`	Vector with the start position in x-direction of the flow paths. Length > =1. If not specified, then all combinations of `x` and `y` at the outer margins will be used as starting point.
`starty`	Vector with start position in y-direction of flow paths. Length = length of `startx`.
`colvar`	The variable used for coloring. It need not be present, but if specified, it should be a vector of dimension equal to `c(nrow(u),ncol(v))`. Values of `NULL`, `NA`, or `FALSE` will toggle off coloration according to `colvar`.
`col`	Colors to be used for coloring the arrows as specified by the `colvar` variable. If `col` is `NULL` and `colvar` is specified, then a red-yellow-blue colorscheme (jet.col) will be used. If `col` is `NULL` and `colvar` is not specified, then `col` will be "black".
`NAcol`	Colors to be used for `colvar` values that are `NA`.

`breaks` a set of finite numeric breakpoints for the colors; must have one more breakpoint than color and be in increasing order. Unsorted vectors will be sorted, with a warning.

`scale` Scaling factor for the arrows. When `scale = 1`, the longest arrow will fill a grid cell in x- and y- direction. When `scale = 2`, it will be twice as long.

`arr.max` Maximal size of the arrowhead, in cm (approximately). The arrows are scaled according to the velocity (`sqrt(u^2 + v^2)`). `arr.max` is associated with the maximal velocity.

`arr.min` Minimal size of the arrowhead, in cm (approximately). Set `arr.min = arr.max` for constant size.

`speed.max` Speed that corresponds to `arr.max`. Everything with speed larger than `speed.max` will be depicted with size equal to `arr.max`. If unspecified (`max(sqrt(u^2 + v^2))`).

`by` Number increment for plotting the vectors; one value or two (x, y) values. For example, setting `by = 2` will plot every second velocity value in `x` and in `y` direction. Setting `by = c(1,2)` will plot all vectors in `x` and every second vector in `y`. Useful if the vector density is too high.

`colkey` A logical, `NULL` (default), or a `list` with parameters for the color key (legend). List parameters should be one of `side,plot,length,width,dist,shift,addlines,col.clab,cex.cl` and the axis parameters `at,labels,tick,line,pos,outer,font,lty,lwd,lwd.ticks,col.box,col.` The defaults for the parameters are `side = 4,plot = TRUE,length = 1,width = 1,dist = 0,shift = 0,addlines = FALSE,col.clab = NULL,cex.clab = par("cex.lab"),side.clab = NULL,line.clab = NULL,adj.clab = NULL,font.clab = NULL)` See colkey from package `plot3D`.

The default is to draw the color key on side = 4, i.e. in the right margin. If `colkey = NULL` then a color key will be added only if `col` is a vector. Setting `colkey = list(plot = FALSE)` will create room for the color key without drawing it. if `colkey = FALSE`, no color key legend will be added.

`type` The type of the arrow head, one of `"triangle"` (the default) or `"simple"`, which uses R-function arrows.

`contour, image` If present, then a contour2D or image2D plot will be added to the quiver plot. They should be a `list` with arguments for the contour2D or `image2D` function.

`clim` Only if `colvar` is specified, the range of the colors, used for the color key.

`clab` Only if `colkey` is not `NULL` or `FALSE`, the label to be written on top of the color key. The label will be written at the same level as the main title. To lower it, `clab` can be made a vector, with the first values empty strings.

`margin` A vector giving the subscripts which the plotting function will be applied over. The plotting function will loop over the index that is not in `margin`. For instance, `c(1,2)`, indicates to plot rows(x) and columns(y) and to loop over index `3`; `c(2,1)` will do the same but transposed. `margin` should be a vector with two numbers inbetween `1`, and `3`.

`ask` A logical; if `TRUE`, the user is asked before each plot, if `NULL` the user is only asked if more than one page of plots is necessary and the current graphics device is set interactive, see par(ask) and dev.interactive.

`add`	If `TRUE`, will add to current plot. Else will start a new plot. Note: to use this in a consistent way, the previous plot should have been done with one of the `plot3D` functions.
`mask`	A `matrix` or `list` defining the grid cells outside the domain as `NA`. Use a list with argument `NAcol` to specify the color that the masked cells (that are `NA`) should get; the default is `"black"`. The unmasked cells are left `"white"`. If `x` and `y` are a vector, then `mask` can be a `matrix` with dimension equal to `length(x),length(y)`. If either `x` or `y` is itself a `matrix`, then `mask` should be a `list` that contains the `x`, `y`, and `z` values (and that are named 'x', 'y', 'z'). A `mask` cannot be combined with `add = TRUE`.
`plot`	If `FALSE`, will not plot the flow paths, but will return the matrix with path values instead.
`numarr`	The number of arrows added on the flow paths.
`arr.length`	Constant size of the arrowhead, in cm (approximately).
`maxstep`	Maximum number of steps for calculating the flow paths.
`...`	Additional arguments passed to the plotting methods (arrows2D), The arguments after ... must be matched exactly.
`subset`	A logical expression indicating over which elements to loop; missing values are taken as `FALSE`.

Details

S3 function `quiver2D` plots vectors specified by `u,v` at the coordinates `x,y`.

`flowpath` uses the velocities `u,v` at the coordinates `x,y` to create trajectories, starting at points `startx,starty`. It can also be used to return the flow path points by setting `plot` equal to `FALSE`. It uses very simple Euler integration and may not be very accurate.

Value

`flowpath` returns (as `invisible`) a 2-column matrix with the x-y coordinates of the flow paths. Separate flow paths are separated with `NA`.

`quiver2D` returns (as `invisible`) a `list` containing the coordinates of the arrows (`x0`, `x1`, `y0`, `y1`), the color of each arrow (`col`), the length of the arrowhead (`length`) and the maximal speed corresponding to `arr.max` (`speed.max`). This output can be used e.g. with function arrows.

Note

There was a slight error in the scaling of the arrows in versions previous to 1.0.3, which has been corrected. See last example.

See Also

arrows3D for an arrows function from package `plot3D`.

vectorplot for plotting velocity vectors as spikes.

Arrows for the arrow function from package `shape` on which quiver2D is based.

Examples

```
## =======================================================================
##  EXAMPLE 1:
## =======================================================================
 pm <- par("mfrow")
 par(mfrow = c(2, 2))

# generate velocities
 x  <- seq(-1, 1, by = 0.2)
 y  <- seq(-1, 1, by = 0.2)
 dx <- outer(x, y , function(x, y) -y)
 dy <- outer(x, y , function(x, y) x)

# velocity plot, with legend
 F <- quiver2D(u = dx, v = dy, x = x, y = y)
 legend("topright", bg = "white",
   legend = paste("max = ", format(F$speed.max, digits = 2)))

# different color for up/downward pointing arrows
 quiver2D(u = dx, v = dy, x = x, y = y, colvar = dx > 0,
        col = c("red", "blue"), colkey = FALSE,
        arr.max = 0.4, arr.min = 0.1)

# different scale
 quiver2D(u = dx, v = dy, x = x, y = y, by = 2, scale = 2)

# three flow paths
 flowpath(u = dx, v = dy, x = x, y = y, startx = 0.1, starty = 0.1)
 flowpath(u = dx, v = dy, x = x, y = y,
          startx = c(0.9, -0.9), starty = c(0.0, 0.0), col = "red",
          numarr = 2, add = TRUE)

## =======================================================================
##  EXAMPLE 2: note: has changed in version 1.0.3 - uses contour2D!
## =======================================================================
 par(mfrow = c(1, 1))
 x <- seq(-2, 2, by = 0.2)
 y <- seq(-1, 1, by = 0.2)
 z <- outer (x, y, function(x, y) x^3 - 3*x -2*y^2)
 contour2D(x, y, z = z, col = jet.col(10))

# gradients in x- and y-direction (analytical)
 dX <- outer(x, y, function(x,y) 3*x^2 - 3)
 dY <- outer(x, y, function(x,y) -4*y)

 quiver2D(u = dX, v = dY, x = x, y = y, scale = 1, add = TRUE, by = 1)
 flowpath(u = dX, v = dY, x = x, y = y, startx = c(-2, 1.1),
           starty = c(-1, -1), add = TRUE, arr.length = 0.5,
           col = "darkgreen", lwd = 3, numarr = 1)

## =======================================================================
##  EXAMPLE 3:
```

```
## =======================================================================

 x <- y <- 1:20
 u <- outer (x, y, function (x, y) cos(2*pi*y/10))
 v <- outer (x, y, function (x, y) cos(2*pi*x/10))

 quiver2D(x = x, y = y, u = u, v = v, col = "grey")

# flowpaths using all combinations of x and y at edges
 flowpath(x = x, y = y, u = u, v = v, add = TRUE,
          lwd = 2, col = "orange")

## =======================================================================
##  EXAMPLE 4: quiver of an array..
## =======================================================================

 x <- y <- 1:20
 u2 <- outer (x, y, function (x, y) sin(2*pi*y/10))
 v2 <- outer (x, y, function (x, y) sin(2*pi*x/10))

# merge u, u2 and v, v2 to create an "array"
 U <- array(dim = c(dim(u2), 2), data = c(u, u2))
 V <- array(dim = c(dim(v2), 2), data = c(v, v2))

 quiver2D(u = U, v = V, x = x, y = y, main = c("time 1", "time 2"))

# quiver over x and time, for a subset of y-values:
 quiver2D(u = U, v = V, x = x, y = 1:2,
        margin = c(1, 3), main = paste("y ", y),
        subset = y <= 4)

## Not run:
 quiver2D(u = U, v = V, x = x, y = y, ask = TRUE,
        mfrow = c(1, 1))

 quiver2D(u = U, v = V, x = x, y = 1:2, ask = TRUE,
        margin = c(1, 3), main = paste("y ", y),
        mfrow = c(1, 1))


## End(Not run)

## =======================================================================
##  EXAMPLE 5:
## =======================================================================
 par(mfrow = c(1, 1))

 image2D(x = 1:nrow(volcano), y = 1:ncol(volcano),
       z = volcano, contour = TRUE)

# Assume these are streamfunctions, we calculate the velocity field as:
 dx <- dy <- 1
 v <-   (volcano[-1, ] - volcano[-nrow(volcano), ] )/dx
```

```
 u <- - (volcano[, -1] - volcano[ ,-ncol(volcano)] )/dy

 quiver2D(x = 1:nrow(u), y = 1:ncol(v),
        u = u, v = v, add = TRUE, by = 3)

 flowpath(x = 1:nrow(u), y = 1:ncol(v), numarr = 10,
          u = u, v = v, add = TRUE, lwd = 2, col = "grey",
          startx = 20, starty = 30)

## ======================================================================
##  EXAMPLE 6: boundary mask, images, contours
## ======================================================================
 par (mfrow = c(2, 2))

 mask <- volcano; mask[volcano < 120]  <- NA
 quiver2D(by = c(3, 2), u = u, v = v, mask = mask)

 quiver2D(by = c(3, 2), u = u, v = v,
        image = list(z = mask, NAcol = "black"))

 quiver2D(by = c(4, 3), u = u, v = v,
        contour = list(z = volcano, lwd = 2))

 quiver2D(by = c(4, 3), u = u, v = v,
        contour = list(z = volcano, col = "black"),
        image = list(z = volcano, NAcol = "black"))

## ======================================================================
##  Same in rgl
## ======================================================================
## Not run:
 quiver2Drgl(by = c(3, 2), u = u, v = v, mask = mask, NAcol = "black")

 quiver2Drgl(by = c(3, 2), u = u, v = v,
        image = list(z = volcano, NAcol = "black"))

 quiver2Drgl(by = c(4, 3), u = u, v = v, scale = 2,
        contour = list(z = volcano, lwd = 2))

 quiver2Drgl(by = c(4, 3), u = u, v = v,
        contour = list(z = volcano, col = "black"),
        image = list(z = volcano, NAcol = "black"))
 cutrgl()
 uncutrgl()

## End(Not run)

## ================================================================================
## 2-D Data set SyltSurf
## ================================================================================

 par(mfrow = c(1, 1))
 with (Syltsurf,
```

```
    quiver2D(x = x, y = y, u = u[ , ,2], v = v[ , ,2],
      xlim = c(5, 20), ylim = c(10, 25), by = 3,
      main = paste(formatC(time[1]), " hr"), scale = 1.5,
      image = list(z = depth, x = x, y = y, NAcol = "black",
                   colkey = TRUE),
      contour = list(z = depth, x = x, y = y, col = "black",
        drawlabels = FALSE)
      )
  )

## =============================================================================
## 2-D Data set SyltSurf, several time points
## =============================================================================

# now for an array (first and 4th time point only)
 ii <- c(1, 4)
 with (Syltsurf,
   quiver2D(x = x, y = y, u = u[ ,,ii], v = v[ ,,ii],
     xlim = c(5, 20), ylim = c(10, 25), by = 4,
     mask = list(z = depth, x = x, y = y, NAcol = "blue"),
     main = paste(formatC(time[ii]), " hr"), scale = 1.5,
     contour = list(z = depth, x = x, y = y, drawlabels = FALSE)
     )
  )



## =============================================================================
## Adding quivers ...
## =============================================================================
x <- 1:2
y <- 1:3
u <- matrix(1:6,2,3)
v <- matrix(6:1,2,3)
 par(mfrow = c(1, 1))

A <- quiver2D(x = x, y = y, u = u, v = v)
B <- quiver2D(x = x, y = y[-1], u = u[,-1], v = v[,-1], col = 2, add = TRUE)
C <- quiver2D(x = x, y = y[-3], u = u[,-3], v = v[,-3], col = 3, add = TRUE)

# restore parameter settings
 par(mfrow = pm)
```

Reshaping to a crosstable *Converts a dataset from database-format to a cross table*

Description

Reshapes data arranged in 3 columns to a “crosstable” matrix.

Usage

```
db2cross (input, row = 1, col = 2, value = 3, subset = NULL,
  df.row = NA, df.col = NA, out.row = NA, out.col = NA,
  full.out = FALSE)
```

Arguments

`input`	A `matrix` in *database* format, (x,y,z) .
`row`	Number or name of the column in `input`, to be used as rows in the result.
`col`	Number or name of the column in `input`, to be used as columns in the result.
`value`	Number or name of the column in `input`, to be used as values in the result.
`subset`	Logical expression indicating elements or rows to keep; missing values are taken as `FALSE`
`df.row, df.col`	Maximal distance in row and column values that should be considered the same. The default is to use each unique row or column value in `input` as a row or column value in the crosstable. Overrruled when `out.row` or `out.col` are defined.
`out.row, out.col`	Values of rows and columns to be used in the cross table. The default is to use each unique row or column value in `input` as a row or column value in the crosstable. Each value in `input` is mapped to `out.row` and `out.col` to which it is closest. Overrrules `df.row` or `df.col`.
`full.out`	If `TRUE`, will also output how the input values were mapped to the output values. This is only relevant if either of `df.row`, `df.col`, `out.row` or `out.col` is not `NULL`.

Details

Uses a simple fortran function.

rows and columns are generated by the `unique values` in each x- and y-column.

Value

a list containing:

`x`	The values of the *rows*.
`y`	The values of the *columns*.
`z`	The crosstable, a matrix.

and if `full.out` = `TRUE` also

`map`	The mapping of the *x* and *y* values, consisting of `var.input`, `factor`, `var.output`, with the original values, how they are mapped, and the resulting values respectively.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

See Also

reshape, the official (slow) R-function

remap to remap a matrix or array to higher or lower resolution

Examples

```
## =======================================================================
## test the function on a small data set
## =======================================================================

 df3 <- data.frame(school = rep(c("a","b","c"), each = 4),
                   class = rep(9:10, 6),
                   time = rep(c(1,1,2,2), 3),
                   score = rnorm(12))
 head(df3)
 db2cross(df3, val = 4)

## =======================================================================
## Defining the output rows
## =======================================================================
Samples <- data.frame(time = c(1, 1.1, 1.2, 2, 2.1, 2.2, 4, 4.1, 4.2),
                      var = rep(c("O2", "NO3", "NH3"), 3),
                      val = 1:9)
Samples

db2cross(Samples)
db2cross(Samples, df.row = 0.5)
db2cross(Samples, out.row = c(1, 2, 4))
db2cross(Samples, out.row = 1:4)

## =======================================================================
## A larger dataset; requires OceanView.Data
## =======================================================================
## Not run:
 data (pp.aug2009.db)
 crosstab <- db2cross(pp.aug2009.db)
 crosstab$z[crosstab$z>1000] <- 1000
 crosstab$z[crosstab$z<0]    <- NA

 image2D(z = crosstab$z, x = crosstab$x, y = crosstab$y,
       main = "primary production august 2009 mgC/m2/d",
       NAcol = "black")

## End(Not run)
```

Sylt data set   *Hydrodynamic model output (getm) of Sylt-Romo Bight*

Description

3D Sylt-tidal simulation model output generated by the GETM model version 2.2.2.

The Sylt-Romo bight is a Wadden Sea embayment in the North Sea, between the Danish island Romo and the German island Sylt at about 55 dg N and 8 dg E, an area of approximately 300 km^2.

- `Sylttran` contains (x, sigma, time) data from an E-W transect.
- `Syltsurf` contains 2-D surface data, at 5 time intervals.
- `Sylt3D` contains 3-D (x, y, z) data, at 2 time intervals.

Usage

```
data(Sylttran)
data(Syltsurf)
data(Sylt3D)
```

Format

- `Sylttran` is a `data.frame` with (x, sigma, time) data from an E-W transect (8.1 - 17.9 km) taken at km 18.5. There are 50 x-values, 21 sigma levels and 21 model output times.

  It contains:
  - `x,y`, the positions in km, of length 50 and 1 respectively.
  - `time`, the model output time in hours, of length 21.
  - `visc`, the viscosity (getm variable `num`), (50 x 21 x 21), m2/s.
  - `tke`, the turbulent kinetic energy (getm variable `tke`), (50 x 21 x 21), m2/s2.
  - `u,v`, the zonal and meridional velocity, (50 x 21 x 21), m/s.
  - `sigma`, the depth of the sigma coordinates (50 x 21 x 21), metres.
- `Syltsurf` contains 2-D surface data of the entire model domain, at 5 time intervals (hour 24.7 to 37.1). It is a `data.frame` with:
  - `x,y`, the positions in km, of length 135 and 160 respectively.
  - `time`, the output time in hours, of length 5.
  - `u,v`, the vertically averaged zonal and meridional velocity (135 x 160 x 5), m/s.
  - `elev`, tidal elevation (135 x 160 x 5), metres.
  - `depth`, the bathymetry (135 x 160), metres.
- `Sylt3D` contains 3-D (x, y, z) data, at 2 time intervals (hour 0 and 9.94). The box extends from x inbetween [12.1, 14.9] and from y inbetween [12.7 - 16.3]; there are 21 sigma levels.

  It is a `data.frame` with:
  - `x,y`, the positions in km, of length 15 and 19 respectively.
  - `time`, the output time in hours, of length 2.
  - `visc`, the viscosity (getm model variable num), (55 x 19 x 21 x 2), m2/s.
  - `sigma`, the sigma depth levels, (55 x 19 x 21 x 2), m2/s. metres.
  - `depth`, the bathymetry (15 x 19), metres.

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

References

Hans Burchard and Karsten Bolding, 2002. GETM, A General Estuarine Transport Model, Scientific Documentation. EUR 20253 EN.

http://www.getm.eu

See Also

image2D for plotting images, package `plot3D`.

ImageOcean for an image of the ocean's bathymetry, package `plot3D`.

scatter2D for making scatterplots, package `plot3D`.

Oxsat for a 3-D data set, package `plot3D`.

Examples

```
# save plotting parameters
 pm <- par("mfrow")
 mar <- par("mar")

## =============================================================================
## Show position of transect and 3D box in bathymetry
## =============================================================================

 par(mfrow = c(2, 2))
 par(mar = c(4, 4, 4, 4))

 x <- Syltsurf$x ; y <- Syltsurf$y ; depth <- Syltsurf$depth
 image2D(z = depth, x = x, y = y, clab = c("depth", "m"))

# position of transect
 with (Sylttran, points (x, rep(y, length(x)),
         pch = 16, col = "grey"))
# position of 3-D area
 with (Sylt3D, rect(x[1], y[1], x[length(x)], y[length(y)], lwd = 3))

 image2D(z = depth, x = x, y = y, clab = c("depth", "m"), log = "z")

# sigma coordinates of the transect (at time = 10)
 matplot(Sylttran$x, Sylttran$sigma[,,10], type = "l",
         main = "sigma", ylim = c(25, -2), col = "black", lty = 1)

# perspective view - reduce resolution for speed
 ix <- seq(1, length(x), by = 3)
 iy <- seq(1, length(y), by = 3)

 par(mar = c(1, 1, 1, 2))
 persp3D(z = -depth[ix, iy], x = x[ix], y = y[iy],
   scale = FALSE, expand = 0.2, ticktype = "detailed",
   col = "grey", shade = 0.6, bty = "f",
   plot = FALSE)
```

```
# add 3-D region; small amount added to z so that it is visible in rgl
 persp3D(z = -Sylt3D$depth + 1e-3, x = Sylt3D$x, y = Sylt3D$y,
   col = alpha.col("red", alpha = 0.4), add = TRUE,
   plot = FALSE)

# transect
 with (Sylttran, points3D(x = x, y = rep(y, length(x)),
   z = rep(0, length(x)), pch = 16, add = TRUE, colkey = FALSE))

## Not run:
 plotrgl()
 plotrgl(lighting = TRUE, new = FALSE, smooth = TRUE)

## End(Not run)

## =============================================================================
## Data Syltsurf: Surface elevation
## =============================================================================

 par(mfrow = c(2, 2), mar = c(0, 0, 1, 0))
# reduce resolution for speed
 ix <- seq(1, length(x), by = 3)
 iy <- seq(1, length(y), by = 3)

 clim <- range(Syltsurf$elev, na.rm = TRUE)
 for (i in 1:3)
   persp3D(z = -depth[ix, iy], colvar = Syltsurf$elev[ix,iy,i],
     x = x[ix], y = y[iy], clim = clim, inttype = 2,  d = 2,
     scale = FALSE, expand = 0.1, colkey = FALSE, shade = 0.5,
       main = paste(format(Syltsurf$time[i], digits = 3), " hr"))
 par(mar = c(3, 3, 3, 3))
 colkey(clim = clim, clab = c("elevation", "m"))

# can also be done using shaded image2D plots, faster
 par(mfrow = c(2, 2), mar = c(3, 3, 3, 3))
 clim <- range(Syltsurf$elev, na.rm = TRUE)
 for (i in 1:3)
   image2D(z = -depth[ix, iy], colvar = Syltsurf$elev[ix,iy,i],
     x = x[ix], y = y[iy], clim = clim,
     colkey = FALSE, shade = 0.3, resfac = 2,
       main = paste(format(Syltsurf$time[i], digits = 3), " hr"))
 colkey(clim = clim, clab = c("elevation", "m"))

## =============================================================================
## Data Syltsurf: Surface currents
## =============================================================================

 par(mfrow = c(1, 1))
 Speed <- sqrt(Syltsurf$u[,,2]^2 + Syltsurf$v[,,2]^2)

 with (Syltsurf,
   quiver2D(x = x, y = y, u = u[,,2], v = v[,,2], col = gg.col(100),
     xlim = c(5, 20), ylim = c(10, 25), by = 3,
```

```
      colvar = Speed, clab = c("speed", "m/s"),
      main = paste(formatC(time[1]), " hr"), scale = 1.5,
      image = list(z = depth, x = x, y = y, col = "white",  #background
        NAcol = "darkblue"),
      contour = list(z = depth, x = x, y = y, col = "black",#depth
        lwd = 2)
      )
   )

## =============================================================================
## Data Sylttran: plot a transect
## =============================================================================

 par(mfrow = c(1, 1), mar = c(4, 4, 4, 2))
 D   <- seq(-1, 20, by = 0.02)

 visc <- mapsigma (Sylttran$visc [ , ,1], x = Sylttran$x,
     sigma = Sylttran$sigma[ , ,1], depth = D, resfac = 2)

 image2D(visc$var, x = visc$x, y = -visc$depth, ylim = c(-20, 1),
     main = "eddy viscosity", ylab = "m", xlab = "hour",
     clab = "m2/s")

# show position of timeseries in next example
 abline(v = visc$x[45])

## =============================================================================
## Data Sylttran: plot a time-series
## =============================================================================

 par(mfrow = c(1, 1), mar = c(5, 4, 4, 3))
 ix <- 45

 visct <-  Sylttran$visc  [ix, ,]
 sig   <-  Sylttran$sigma [ix, ,]

# sigma coordinates are first dimension (signr)
 visc <- mapsigma(visct, sigma = sig, signr = 1,
   x = Sylttran$time, numdepth = 100, resfac = 3)
 D    <- -visc$depth

 image2D(t(visc$var), x = visc$x, y = D, NAcol = "black",
   ylim = range(D), main = "eddy viscosity",
   ylab = "m", xlab = "hour", clab = "m2/s")

## =============================================================================
## Data Sylt3D: increase resolution and map from sigma to depth
## =============================================================================

# select a time series point
 it <- 1
 par(mfrow = c(1, 1))
 sigma  <- Sylt3D$sigma[,,,it]
```

```
 visc   <- Sylt3D$visc [,,,it]
 (D <- dim(sigma))     # x, y, z

# remap the data from sigma coordinates to depth coordinates
# depth from max in first box to max in last box
 depth <- seq(max(sigma[,,D[3]], na.rm = TRUE),
              max(sigma[,,1   ], na.rm = TRUE), length.out = 20)

# Step-bystep mapping, increasing the resolution
 z    <- 1:21
 x    <- Sylt3D$x
 y    <- Sylt3D$y

 xto <- seq(min(x), max(x), length.out = 30)
 yto <- seq(min(y), max(y), length.out = 30)

# higher resolution
 Sigma <- remap(sigma, x, y, z, xto, yto, zto = z)$var
 Visc  <- remap(visc, x, y, z, xto, yto, zto = z)$var

# viscosity in sigma coordinates
 visc_sig <- mapsigma(Visc, sigma = Sigma, depth = depth)

## =============================================================================
## The 3-D data set - plotted as slices
## =============================================================================

 slice3D(xto, yto, -visc_sig$depth, colvar = visc_sig$var,
   scale = FALSE, expand = 0.1, NAcol = "transparent",
   ys = yto[seq(1, length(yto), length.out = 10)], plot = FALSE,
   colkey = list(side = 1))
 persp3D(x = x, y = y, z = -Sylt3D$depth, add = TRUE,
   border = "black", facets = NA, colkey = FALSE)

# visualise it in rgl window
 plotrgl()

## the same, as a movie

 persp3Drgl(x = x, y = y, z = -Sylt3D$depth, smooth = TRUE,
   col = "grey", lighting = TRUE)

 movieslice3D(xto, yto, -visc_sig$depth, colvar = visc_sig$var,
   add = TRUE, ys = yto)

# in order to wait inbetween slice drawings until a key is hit:
## Not run:
 persp3Drgl(x = x, y = y, z = -Sylt3D$depth, smooth = TRUE,
   col = "grey", lighting = TRUE)
 movieslice3D(xto, yto, -visc_sig$depth, colvar = visc_sig$var, add = TRUE,
   ask = TRUE, ys = yto)

## End(Not run)
```

```
## =============================================================================
## The 3-D data set - plotted as isosurfaces
## =============================================================================

 isosurf3D(xto, yto, -visc_sig$depth, colvar = visc_sig$var,
   level = c(0.005, 0.01, 0.015), col = c("red", "blue", "green"),
   scale = FALSE, expand = 0.1, ticktype = "detailed",
   main = "viscosity", clab = "m2/s",
   plot = FALSE, colkey = list(side = 1))
 persp3D(x = x, y = y, z = -Sylt3D$depth, border = "black",
   col = "white", add = TRUE, plot = FALSE)


## Not run:
 plotdev(alpha = 0.3, phi = 30)         # this is slow

## End(Not run)
 plotrgl(alpha = 0.3)

# reset plotting parameters
 par(mar = mar)
 par(mfrow = pm)
```

Tracers in 2D *Plots tracer distributions in 2-D.*

Description

tracers2D plots a tracer distribution using traditional R graphics. The topography can be defined when calling this function.

tracers2Drgl plots a tracer distribution in open-GL graphics. A suitable topography has to be created before calling this function.

Usage

```
tracers2D(x, y, colvar = NULL, ...,
          col = NULL, NAcol = "white", colkey = NULL,
          mask = NULL, image = FALSE, contour = FALSE,
          clim = NULL, clab = NULL)

tracers2Drgl(x, y, colvar = NULL, ...,
          col = NULL, NAcol = "white", breaks = NULL,
          colkey = FALSE, clim = NULL, clab = NULL)
```

Arguments

x, y Vectors with x- and y-coordinates of the tracers. Should be of equal length.

`colvar` The variable used for coloring. It need not be present, but if specified, it should be a vector of dimension equal to `x`. Values of `NULL`, `NA`, or `FALSE` will toggle off coloration according to `colvar`.

`col` Colors to be used for coloring each individual point (if colvar not specified) or that define the colors as specified by the `colvar` variable. If `col` is `NULL` and `colvar` is specified, then a red-yellow-blue colorscheme (`jet.col`) will be used. If `col` is `NULL` and `colvar` is not specified, then `col` will be "black".

`NAcol` Colors to be used for `colvar` values that are `NA`.

`breaks` a set of finite numeric breakpoints for the colors; must have one more breakpoint than color and be in increasing order. Unsorted vectors will be sorted, with a warning.

`colkey` A logical, `NULL` (default), or a `list` with parameters for the color key (legend). List parameters should be one of `side,plot,length,width,dist,shift,addlines,col.clab,cex.cl` and the axis parameters `at,labels,tick,line,pos,outer,font,lty,lwd,lwd.ticks,col.box,col.` The defaults for the parameters are `side = 4,plot = TRUE,length = 1,width = 1,dist = 0,shift = 0,addlines = FALSE,col.clab = NULL,cex.clab = par("cex.lab"),side.clab = NULL,line.clab = NULL,adj.clab = NULL,font.clab = NULL`) See colkey from package `plot3D`.

The default is to draw the color key on side = 4, i.e. in the right margin. If `colkey = NULL` then a color key will be added only if `col` is a vector. Setting `colkey = list(plot = FALSE)` will create room for the color key without drawing it. if `colkey = FALSE`, no color key legend will be added.

`contour, image` If `TRUE`, then a contour2D or `image2D` plot will be added to the quiver plot. Also allowed is to pass a `list` with arguments for the contour2D or image2D function.

`clim` Only if `colvar` is specified, the range of the colors, used for the color key.

`clab` Only if `colkey` is not `NULL` or `FALSE`, the label to be written on top of the color key. The label will be written at the same level as the main title. To lower it, `clab` can be made a vector, with the first values empty strings.

`mask` A `list` defining the grid cells outside the domain as `NA`. Use a list with argument `NAcol` to specify the color that the masked cells (that are `NA`) should get; the default is `"black"`. The unmasked cells are left `"white"`.

`mask` should be a `list` that contains the `x`, `y`, and `z` values (and that are named 'x', 'y', 'z'). A `mask` cannot be combined with `add = TRUE`.

`...` additional arguments passed to the plotting method scatter2D. The arguments after ... must be matched exactly.

Value

returns nothing

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

See Also

tracers3D for plotting time series of tracer distributions in 3D

Ltrans for the output of a particle tracking model

Examples

```
# save plotting parameters
 pm <- par("mfrow")

## =======================================================================
## Create topography, data
## =======================================================================

# The topographic surface
 x <-  seq(-pi, pi, by = 0.2)
 y <-  seq(0, pi, by = 0.1)
 M <- mesh(x, y)
 z <- with(M, sin(x)*sin(y))

# Initial condition
 xi <- c(0.125 * rnorm(100) - pi/2, 0.125 * rnorm(100) - pi/4)
 yi <- 0.25 * rnorm(200) + pi/2

# the species
 species <- c(rep(1, 100), rep(2, 100))

# set initial conditions
 xp <- xi; yp <- yi

## =======================================================================
## using a mask and contour
## =======================================================================

 Z <- z; Z[abs(Z) < 0.1] <- NA
 par(mfrow = c(2, 2))

 for (i in 1:4) {
  # update tracer distribution
   xp <- xp + 0.25 * rnorm(200)
   yp <- yp + 0.025 * rnorm(200)

  # plot new tracer distribution
   tracers2D(xp, yp, colvar = species, pch = ".", cex = 5,
     main = paste("timestep ", i), col = c("orange", "blue"),
     colkey = list(side = 1, length = 0.5, labels = c("sp1","sp2"),
     at = c(1.25, 1.75), dist = 0.075), NAcol = "black",
     mask = list(x = x, y = y, z = Z),
     contour = list(x = x, y = y, z = Z) )
 }

## =======================================================================
## using image and contour
```

```
## =======================================================================

 for (i in 1:4) {
  # update tracer distribution
   xp <- xp + 0.25 * rnorm(200)
   yp <- yp + 0.025 * rnorm(200)

  # plot new tracer distribution
   tracers2D(xp, yp, colvar = species, pch = ".", cex = 5,
     main = paste("timestep ", i), col = c("orange", "blue"),
     colkey = list(side = 1, length = 0.5, labels = c("sp1","sp2"),
     at = c(1.25, 1.75), dist = 0.075), NAcol = "black",
     contour = list(x = x, y = y, z = z),
     image = list(x = x, y = y, z = z, colkey = TRUE))
 }

## =======================================================================
## rgl tracer plot
## =======================================================================

# here the image has to be drawn first
 image2Drgl(x = x, y = y, z = z)

# set initial conditions
 xp <- xi; yp <- yi
 nstep <- 40
 for (i in 1:nstep) {
  # update tracer distribution
   xp <- xp + 0.25 * rnorm(200)
   yp <- yp + 0.025 * rnorm(200)

  # plot new tracer distribution                                         n
   tracers2Drgl(xp, yp, colvar = species,  cex = 1,
     main = paste("timestep ", i), col = c("orange", "blue"))

 }

# reset plotting parameters
 par(mfrow = pm)
```

`Tracers in 3D` *Plotting tracer distributions in 3D*

Description

`tracers3D` plots 3D tracer distributions in traditional graphics. The topography can be defined when calling this function or created before calling this function.

`tracers3Drgl` plots 3D tracer distributions in open-GL graphics. A suitable topography has to be created before calling this function. It does not create a movie.

moviepoints3D creates a movie of tracer distributions in open-GL graphics.

It is based on the plot3Drgl function points3Drgl.

Usage

```
tracers3D (x, y, z, colvar = NULL, ...,
    col = NULL, NAcol = "white", breaks = NULL,
    colkey = FALSE, clim = NULL, clab = NULL, surf = NULL)

tracers3Drgl (x, y, z, colvar = NULL, ...,
    col = NULL, NAcol = "white", breaks = NULL,
    colkey = FALSE, clim = NULL, clab = NULL)

moviepoints3D (x, y, z, colvar, t, by = 1,
  col = jet.col(100), NAcol = "white", breaks = NULL,
  clim = NULL, wait  = NULL, ask = FALSE, add = FALSE,
  basename = NULL, ...)
```

Arguments

x, y, z	Vectors with (x, y, z) positions of tracers. Should be of equal length.
colvar	The variable used for coloring. It need not be present, but if specified, it should be a vector of dimension equal to x,y,z. Values of NULL, NA, or FALSE will toggle off coloration according to colvar.
t	Vectors with time points of tracers. Should be of length equal to length of x,y ,z,colvar.
by	Number increment of the time sequence.
col	Colors to be used for coloring each individual point (if colvar not specified) or that define the colors as specified by the colvar variable. If col is NULL and colvar is specified, then a red-yellow-blue colorscheme (jet.col) will be used. If col is NULL and colvar is not specified, then col will be "black".
NAcol	Colors to be used for colvar values that are NA.
breaks	a set of finite numeric breakpoints for the colors; must have one more breakpoint than color and be in increasing order. Unsorted vectors will be sorted, with a warning.
colkey	A logical, NULL (default), or a list with parameters for the color key (legend). List parameters should be one of side,plot,length,width,dist,shift,addlines,col.clab,cex.cl and the axis parameters at,labels,tick,line,pos,outer,font,lty,lwd,lwd.ticks,col.box,col. The defaults for the parameters are side = 4,plot = TRUE,length = 1,width = 1,dist = 0,shift = 0,addlines = FALSE,col.clab = NULL,cex.clab = par("cex.lab"),side.clab = NULL,line.clab = NULL,adj.clab = NULL,font.clab = NULL) See colkey from package plot3D. The default is to draw the color key on side = 4, i.e. in the right margin. If colkey = NULL then a color key will be added only if col is a vector. Setting colkey = list(plot = FALSE) will create room for the color key without drawing it. if colkey = FALSE, no color key legend will be added.

`clim`	Only if `colvar` is specified, the range of the colors, used for the color key.
`clab`	Only if `colkey` is not `NULL` or `FALSE`, the label to be written on top of the color key. The label will be written at the same level as the main title. To lower it, `clab` can be made a vector, with the first values empty strings.
`surf`	If not `NULL`, a list specifying a surface to be added on the scatterplot. They should include at least `x`, `y`, `z`, equal sized matrices, and optional: `colvar,col,NAcol,border,facets,lwd,res` Note that the default is that `colvar` is not specified.
`add`	Logical. If `TRUE`, then the points will be added to the current plot. If `FALSE` a new plot is started.
`ask`	Logical. If `TRUE`, then new points will only be drawn after a key has been struck. If `FALSE`, redrawing will depend on `wait`
`wait`	The time interval inbetween drawing of a set of new points, in seconds. If `NULL`, the drawing will not be suspended.
`basename`	The base name of a png file to be produced for each movieframe.
`...`	additional arguments passed to scatter3D from package `plot3D`. Typical arguments are `cex,main` (both functions), and `pch,...` for `tracers3D`.

Value

returns nothing

Author(s)

Karline Soetaert <karline.soetaert@nioz.nl>

See Also

tracers2D for plotting time series of tracer distributions in 2D

moviesslice3D for plotting slices in 3D

Ltrans for 3-D output of a particle tracking model

Examples

```
# save plotting parameters
 pm <- par("mfrow")

## =======================================================================
## Create topography, data
## =======================================================================

# The topographic surface
 x <-  seq(-pi, pi, by = 0.2)
 y <-  seq(0, pi, by = 0.1)
 M <- mesh(x, y)
 z <- with(M, sin(x)*sin(y))

# Initial condition
 xi <- c(0.25 * rnorm(100) - pi/2, 0.25 * rnorm(100) - pi/4)
```

```
 yi <- 0.25 * rnorm(200) + pi/2
 zi <- 0.005*rnorm(200) + 0.5

# the species
 species <- c(rep(1, 100), rep(2, 100))

# set initial conditions
 xp <- xi; yp <- yi; zp <- zi

## =======================================================================
## Traditional graphics
## =======================================================================

 par(mfrow = c(2, 2))

# Topography is defined by argument surf
 for (i in 1:4) {
  # update tracer distribution
   xp <- xp + 0.25 * rnorm(200)
   yp <- yp + 0.025 * rnorm(200)
   zp <- zp + 0.25 *rnorm(200)

  # plot new tracer distribution
   tracers3D(xp, yp, zp, colvar = species, pch = ".", cex = 5,
     main = paste("timestep ", i), col = c("orange", "blue"),
     surf = list(x, y, z = z, theta = 0, facets = FALSE),
     colkey = list(side = 1, length = 0.5, labels = c("sp1","sp2"),
                   at = c(1.25, 1.75), dist = 0.075))
 }

# same, but creating topography first
## Not run:
# create the topography on which to add points
 persp3D(x, y, z = z, theta = 0, facets = FALSE, plot = FALSE)

 for (i in 1:4) {
  # update tracer distribution
   xp <- xp + 0.25 * rnorm(200)
   yp <- yp + 0.025 * rnorm(200)
   zp <- zp + 0.25 *rnorm(200)

  # plot new tracer distribution
   tracers3D(xp, yp, zp, colvar = species, pch = ".", cex = 5,
     main = paste("timestep ", i), col = c("orange", "blue"),
     colkey = list(side = 1, length = 0.5, labels = c("sp1","sp2"),
                    at = c(1.25, 1.75), dist = 0.075))
 }

## End(Not run)

## =======================================================================
## rgl graphics
## =======================================================================
```

```
# pause <- 0.05
# create a suitable topography
 persp3D(x, y, z = z, theta = 0, facets = NA, plot = FALSE)

 plotrgl( )
 xp <- xi; yp <- yi; zp <- zi

 nstep <- 10
 for (i in 1:nstep) {
   xp <- xp + 0.05 * rnorm(200) + 0.05
   yp <- yp + 0.0025 * (rnorm(200) + 0.0025)
   zp <- zp + 0.05 *rnorm(200)

#   tracers3Drgl(xp, yp, zp, col = c(rep("orange", 100), rep("blue", 100)),
#     main = paste("timestep ", i))
# or:
   tracers3Drgl(xp, yp, zp, colvar = species, col = c("orange", "blue"),
     main = paste("timestep ", i))
#   Sys.sleep(pause)
# or:  readline("hit enter for next")
 }

# using function moviepoints3D

## Not run:
# first create the data in matrices
 xp <- matrix(nrow = 200, ncol = nstep, data = xi)
 yp <- matrix(nrow = 200, ncol = nstep,  data = yi)
 zp <- matrix(nrow = 200, ncol = nstep, data = zi)
 tp <- matrix(nrow = 200, ncol = nstep, data = 0)
 cv <- matrix(nrow = 200, ncol = nstep, data = species)
 nstep <- 10
 for (i in 2:nstep) {
   xp[,i] <- xp[,i-1] + 0.05 * rnorm(200) + 0.05
   yp[,i] <- yp[,i-1] + 0.0025 * (rnorm(200) + 0.0025)
   zp[,i] <- zp[,i-1] + 0.05 *rnorm(200)
   tp[,i] <- i
 }
# create the topography
 persp3Drgl(x, y, z = z, theta = 0, lighting = TRUE, smooth = TRUE)

# add moviepoints:
  moviepoints3D (xp, yp, zp, colvar = cv, t = tp,
    wait = 0.05, cex = 10, col = c("red", "orange"))


## End(Not run)

# reset plotting parameters
 par(mfrow = pm)
```

`vector plots` *Vector velocity plot.*

Description

Displays (velocity) vectors as segments.

Usage

```
vectorplot(u, v, x = 0, y = 0, colvar = NULL, ...,
          col = NULL, NAcol = "white", breaks = NULL, colkey = NULL,
          by = 1, arr = FALSE, xfac = NULL,
          clim = NULL, clab = NULL, add = FALSE)
```

Arguments

`u` A vector with quantities (velocities) in x-direction.

`v` A vector with quantities (velocities) in y-direction. Should have the same length as `u`

`x` A vector with x-axis values. If `0`, everything will be radiating from the origin. Usually `x` will be equal to `time`.

`y` The y-axis value. One number, or a vector of length = u.

`colvar` The variable used for coloring. It need not be present, but if specified, it should be a vector of dimension equal to `c(nrow(u),ncol(v))`. Values of `NULL`, `NA`, or `FALSE` will toggle off coloration according to `colvar`.

`col` Colors to be used for coloring the arrows as specified by the `colvar` variable. If `col` is `NULL` and `colvar` is specified, then a red-yellow-blue colorscheme (jet.col) will be used. If `col` is `NULL` and `colvar` is not specified, then `col` will be "black".

`NAcol` Colors to be used for `colvar` values that are `NA`.

`breaks` a set of finite numeric breakpoints for the colors; must have one more breakpoint than color and be in increasing order. Unsorted vectors will be sorted, with a warning.

`colkey` A logical, `NULL` (default), or a `list` with parameters for the color key (legend). List parameters should be one of `side,plot,length,width,dist,shift,addlines,col.clab,cex.cl` and the axis parameters `at,labels,tick,line,pos,outer,font,lty,lwd,lwd.ticks,col.box,col.` The defaults for the parameters are `side = 4,plot = TRUE,length = 1,width = 1,dist = 0,shift = 0,addlines = FALSE,col.clab = NULL,cex.clab = par("cex.lab"),side.clab = NULL,line.clab = NULL,adj.clab = NULL,font.clab = NULL`) See colkey from package `plot3D`.

The default is to draw the color key on side = 4, i.e. in the right margin. If `colkey = NULL` then a color key will be added only if `col` is a vector. Setting `colkey = list(plot = FALSE)` will create room for the color key without drawing it. if `colkey = FALSE`, no color key legend will be added.

clim	Only if colvar is specified, the range of the colors, used for the color key.
clab	Only if colkey is not NULL or FALSE, the label to be written on top of the color key. The label will be written at the same level as the main title. To lower it, clab can be made a vector, with the first values empty strings.
by	Number increment for plotting vectors. Set this to an integer > 1 if the vector density is too high.
xfac	Only for x not NULL, the proportionality factor with which the vectors on the x-axis must be drawn. A value of 1 means that the distance of one will be drawn as one x-unit on the x-axis. For a value of 2 a distance of 1 will appear as 2 x-units on the x-axis. if NULL, the range on the y-axis is used. In that case, it may be necessary to manually set the xlim of the figure.
arr	If TRUE, then Arrows will be drawn; if FALSE, segments will be drawn.
add	If TRUE, will add to the current plot.
...	additional arguments passed to the plotting methods.

Value

none

See Also

quiver2D, flowpath, for other functions to plot velocities.

Examples

```
# save plotting parameters
 mf <- par("mfrow")

## =======================================================================
##  EXAMPLE 1:
## =======================================================================

 par(mfrow = c(2, 2))

 u <- cos(seq(0, 2*pi, 0.1))
 v <- sin(seq(0, 2*pi, 0.1)+ 1)

 vectorplot(u = u, v = v)
 vectorplot(u = u, v = v, col = 1:10)

 x <- seq(0, 1, length.out = length(u))
 vectorplot(u = u, v = v, x = x, xfac = 3)
 points(x, rep(0, length(u)), pch = "+", col = "red")

 vectorplot(u = u, v = v, x = 1:length(u), xfac = 10)

## =======================================================================
##  EXAMPLE 2:  adding to a plot
## =======================================================================
```

```
 par(mfrow = c(2, 2))
 x <- 1:length(u)
 plot(x, u)
 vectorplot(u = u, v = v, x = x, xfac = 10,
            add = TRUE, col = "red")

 vectorplot(u = u, v = v, x = x, xfac = 10,
   colvar = sqrt(u^2+v^2), clab = "m/s")

 vectorplot(u = u, v = v, x = x, xfac = 10,
   colvar = sqrt(u^2+v^2), clab = "m/s", log = "c")

# reset plotting parameters
 par(mfrow = mf)
```

Index