# Package 'ocean'

July 2, 2014

**Version** 0.2-4

**Date** 2014-04-29

**Title** Biophysical Oceanography Tools

**Author** Benjamin Jones [aut, cre]

**Maintainer** Benjamin Jones <btjones16@gmail.com>

**Depends** R (>= 3.0.2), graphics, methods, ncdf4, proj4

**Description** This package contains a set of functions for manipulating and visualizing output from numerical ocean models. Many of the routines are targeted towards Lagrangian particle-tracking models with the data stored in NetCDF files. This package may be useful to spatial marine ecologists and biological or physical oceanographers.

**License** GPL-3

**LazyLoad** yes

**LazyData** yes

**URL** https://github.com/btjones16/ocean

**BugReports** https://github.com/btjones16/ocean/issues

**Collate** 'filter2d.R' 'color_ramps.R' 'fvcom_grid.R'

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2014-05-01 07:44:59

# R topics documented:



---

`bathy.colors` *Generate a sequence of colors for plotting bathymetric data.*

---

## Description

`bathy.colors(n)` generates a sequence of $n$ colors along a linear scale from light grey to pure blue.

## Usage

```
bathy.colors(n, alpha = 1)
```

## Arguments

| | |
|---|---|
| `n` | The number of colors to return. |
| `alpha` | Alpha values to be passed to `rgb()`. |

## Value

A vector of blue scale colors.

## Examples

```
{
# Plot a colorbar using bathy.colors
image(matrix(seq(100), 100), col=bathy.colors(100))
}
```

---

`filter2d` *Applies a 2d Gaussian filter to* `mat` *with a standard deviation of r cells.*

---

## Description

Applies a Gaussian blur to a 2D matrix. The matrix is first convoluted with the filter along rows, then along columns.

## Usage

```
filter2d(mat, r)
```

## Arguments

| | |
|---|---|
| `mat` | The matrix to filter. |
| `r` | The standard deviation of the filter in matrix cells. |

---

`find.elem` *Checks if the points (xy$x, xy$y) are in the* `fvcom.grid` `grid`.

---

## Description

Given a grid and a point or set of points, `find.elem` returns the element number of each point in the grid. NA is returned for any point that does not lie within the grid. `x` and `y` must be the same length.

## Arguments

| | |
|---|---|
| `grid` | A `fvcom.grid` instance. |
| `xy` | A `data.frame` with components `x` and `y` with the `x` and `y` locations of the points. |
| `units` | Either 'll' for latitude and longitude or 'm' for meters. |

## Value

A vector of integer values of length `nrow(xy)`. The values are indexes into the elements of grid.

## Examples

```
{
# Create a regular matrix grid
nodes = get.nodes(ocean.demo.grid)
lattice.grid = expand.grid(x=seq(min(nodes$x), max(nodes$x), len=50),
                           y=seq(min(nodes$y), max(nodes$y), len=50))
elems = find.elem(ocean.demo.grid, lattice.grid, units="m")
# Plot the result
plot(lattice.grid$x, lattice.grid$y, pch=15,
     col=jet.colors(max(elems, na.rm=TRUE) + 2)[elems+2])
}
```

---

`fvcom.grid` *Create a new FVCOM grid instance from a FVCOM NetCDF output file.*

---

## Description

Create a new FVCOM grid instance from a FVCOM NetCDF output file.

## Usage

```
fvcom.grid(filename, proj)
```

## Arguments

| | |
|---|---|
| `filename` | The name of an output NetCDF file from FVCOM. |
| `proj` | A string to passed to proj4::project to convert x,y locations to latitude and longitude. |

## Value

An instance of the `fvcom.grid` class.

---

`fvcom.grid-class` *Finite Volume Community Ocean Model grid*

---

## Description

`fvcom.grid` provides a represenation of the unstructured triangular sigma grid used by the Finite Volume Community Ocean Model (FVCOM). As a disclaimer, please note that the author of this package is a user of FVCOM, but is not affiliated with its development.

### Slots

**nodes.n** The number of nodes in the grid.

**nodes.x** x-coordinates of the nodes (m).

**nodes.y** y-coordinates of the nodes (m).

**nodes.h** z-coordinates (depth) of the nodes (m).

**nodes.lat** Latitude of the nodes.

**nodes.lon** Longitude of the nodes.

**elems.n** Number of elements in the grid

**elems.v1** 1st set of node indices

**elems.v2** 2nd set of node indices

**elems.v3** 3rd set of node indices

**elems.size** The area of each element (m^2).

**proj** A string that could be passed to proj4::project to convert x and y values to latitude and longitude.

### References

http://fvcom.smast.umassd.edu/FVCOM/

---

## `get.depth` *Get the depth at each node in the grid.*

---

### Description

Get the depth at each node in the grid.

### Arguments

| | |
|---|---|
| `grid` | A `fvcom.grid` instance. |

### Value

A vector of length `get.nnodes(grid)` with the depth at each node.

## get.elems — *Get the indices of the element vertices in the grid.*

### Description

Get the indices of the element vertices in the grid.

### Arguments

| | |
|---|---|
| grid | A fvcom.grid instance. |

### Value

A data.frame with v1, v2, and v3 elements that correspond to the indices of the nodes at vertices 1, 2, and 3.

## get.nelems — *Get the number of elements in the grid.*

### Description

Get the number of elements in the grid.

### Arguments

| | |
|---|---|
| grid | A fvcom.grid instance. |

### Value

The number of elements in grid.

## get.nnodes — *Get the number of nodes in the grid.*

### Description

Get the number of nodes in the grid.

### Arguments

| | |
|---|---|
| grid | A fvcom.grid instance. |

### Value

The number of nodes in grid.

## get.nodes — *Get the values of the nodes in the grid.*

### Description

Get the values of the nodes in the grid.

### Arguments

| | |
|---|---|
| grid | A fvcom.grid instance. |

### Value

A data.frame with x, y, and h elements that correspond to the x, y, and depth of each node.

## get.proj — *Get the value of the projection string.*

### Description

Get the value of the projection string.

### Arguments

| | |
|---|---|
| grid | A fvcom.grid instance. |

### Value

The value of the projection string specified when the grid was created.

## image — *Plot a* fvcom.grid *instance as a heatmap.*

### Description

Given a vector of data, this function plots the data as a heatmap on an unstructured grid. The length of the data vector must be as long as either the number nodes in the grid or the number of elements in the grid. The grid is currently stored as a data.frame, but will be converted to an S4 object in the future.

### Arguments

| | |
|---|---|
| x | A fvcom.grid instance. |
| z | A vector to plot as a heatmap. |
| units | Either 'll' for latitude and longitude or 'm' for meters. |
| col | A list of colors, such as that returned by bathy.colors. |
| add | Should the plot be added to the current plot? #' |
| xlim | x-limits for the plot. |
| ylim | y-limits for the plot. |
| zlim | z-limits for the plot. |
| border.col | Color of the element borders. If not provided the borders will be colored to match the adjacent polygons. |
| border.lwd | Line width of the element borders. |
| bg.col | Color of the background. The background is only plotted if add=F, otherwise bg.col is ignored. |

### Examples

```
{
  op = par(ask=TRUE)
  # Plot the grid in a single color
  image(ocean.demo.grid, col='white')
  # Plot the grid in bathy colors
  image(ocean.demo.grid)
  par(op)
}
```

---

## interp *Convert a single scalar or node based quantity to element based.*

---

### Description

The length of x determines how it will be treated. If x has length 1, it is returned as a single color. If the length of x is the number of nodes in the grid, its value for each element is calculated as the average of the value at the adjoining nodes. If the length of x is the number of elements in the grid, it is returned as is. Any other values throw an error.

### Arguments

| | |
|---|---|
| grid | An fvcom.grid instance. |
| x | A vector of length 1 or length get.nnodes(grid) |

### Value

A vector of length get.nelems(grid)

## is.in.grid — *Checks if the points (xy$x, xy$y) are in the* fvcom.grid grid.

### Description

Checks if the points (xy$x, xy$y) are in the fvcom.grid grid.

### Arguments

| | |
|---|---|
| grid | A fvcom.grid instance. |
| xy | A data.frame with components x and y with the x and y locations of the points. |
| units | Either 'll' for latitude and longitude or 'm' for meters. |

### Value

A vector of logical values of length nrow(xy). The ith element is TRUE if (xy$x[i], xy$y[i]) is in grid and FALSE otherwise.

### Examples

```
{
# Create a regular grid of test points
nodes = get.nodes(ocean.demo.grid)
lattice.grid = expand.grid(x=seq(min(nodes$x), max(nodes$x), len=25),
                           y=seq(min(nodes$y), max(nodes$y), len=25))
# Check which points are in the grid
in.grid = is.in.grid(ocean.demo.grid, lattice.grid, units="m")
# Plot the points that are in the grid
with(lattice.grid[in.grid,], plot(x, y))
}
```

## jet.colors — *Generate a sequence of colors alog the jet colormap.*

### Description

jet.colors(n) generates a sequence of $n$ colors from dark blue to cyan to yellow to dark red. It is similar to the default color schemes in Python's matplotlib or MATLAB.

### Usage

```
jet.colors(n, alpha = 1)
```

### Arguments

| | |
|---|---|
| n | The number of colors to return. |
| alpha | The transparency value of the colors. See ?rgb for details. |

## Value

A vector of colors along the jet colorramp.

## Examples

```
{
# Plot a colorbar with jet.colors
image(matrix(seq(100), 100), col=jet.colors(100))
}
```

---

`lines` *Plot an instance of the* `fvcom.grid` *class and overlay the trajectories given by* `xy`.

---

## Description

Plot an instance of the `fvcom.grid` class and overlay the trajectories given by `xy`.

## Arguments

| | |
|---|---|
| `x` | A `fvcom.grid` instance |
| `xy` | A `list` with matrices `x` and `y` components that contain the trajectories to plot. The columns of `xy$x` are plotted against the columns of `xy$y`, so each particle trajectory should be in a column and each time index in a row. |
| `plot.units` | The units for plotting. Either 'm' for meters or 'll' for latitude and longitude. |
| `xy.units` | The units of `xy`. Either 'm' for meters or 'll' for latitude and longitude. |
| `...` | Additional arguments to be passed to `matlines`. |

## Examples

```
{
# Create a set of random trajectories.
nodes = get.nodes(ocean.demo.grid)
set.seed(1)
x = apply(matrix(rnorm(1000, 0, 0.01), 250), 2, cumsum) - 69.5
y = apply(matrix(rnorm(1000, 0, 0.01), 250), 2, cumsum) + 42.5
lines(ocean.demo.grid, list(x=x, y=y), lty=1)
}
```

## loadFVCOMGrid27 *Load an FVCOM grid from a NetCDF output file.*

### Description

Loads enough of an FVCOM grid to use the other methods associated with the `fvcom.grid` class. The (x,y,h) locations of the nodes and their connections to form an unstructured triangular mesh are loaded.

### Usage

```
loadFVCOMGrid27(filename, proj)
```

### Arguments

| | |
|---|---|
| `filename` | The name of an output NetCDF file from FVCOM 2.7. |
| `proj` | A projection string which could be passed to `proj4::project` and converts between the x,y and lat,lon coordinate systems for this grid. |

### Value

An instance of the `fvcom.grid` class.

## ocean.demo.grid *An example instance of the* `fvcom.grid` *class.*

### Description

A dataset containing an example of the `fvcom.grid` class. This example was generated for the purposes of demonstrating the plotting functions of this package and is not an actual grid. The grid uses bathymetry from NOAA's ETOPO1 dataset, which is available at (http://www.ngdc.noaa.gov/mgg/global/global.html).

### Format

A `fvcom.grid` instance.

pdd *Plot the density of x and y on grid.*

## Description

Plots the distribution of x and y on grid. This function follows the method described in Simons et al 2013: first vertically integrating the data, then dividing by the number of particles spawned, and finally applying an Gaussian blur filter. The coordinates should be passed in as x,y in meters, then are inverse projected into lat/long using the PROJ.4 library.

## Arguments

| | |
|---|---|
| `grid` | A `fvcom.grid` instance. |
| `xy` | A `data.frame` with x and y location of points. NAs are not supported. |
| `npoints` | The number of points to scale the density plot by. This defaults to the number of points passed in, but it may be useful to set it to a different value if only a subset of the points are being plotted (e.g. some points are outside of the domain). |
| `res` | The resolution of the plot in the same dimensions as `xy` is given. Square boxes will be plotted with each side of length `res`. |
| `sigma` | The standard deviation of the Gaussian smoothing filter to be applied. If no filter is required, set sigma=0. The units should be the same as for `res`. |
| `log` | Should the density be log10 transformed before plotting? |
| `bg.col` | The background color. |
| `col` | A list of colors, such as that returned by heat.colors. |
| `add` | Should the plot be added to the current plot? |
| `xlim` | x-limits for the plot. |
| `ylim` | y-limits for the plot. |
| `lim.units` | Units for xlim and ylim. One of 'm' (meters) or 'll' (latitude and longitude). |
| `zlim` | z-limits for the plot. |

## References

Simons, R.D. and Siegel, D.A. and Brown K.S. 2013 Model sensitivity and robustness in the estimation of larval transport: A study of particle tracking parameters *J. Marine Systems* 119–120: 19–29.

## Examples

```
{
# Generate artificial data from a Gaussian distribution
nodes = get.nodes(ocean.demo.grid)
set.seed(1)
x = rnorm(50000, mean(nodes$x), sd(nodes$x))
y = rnorm(50000, mean(nodes$y), sd(nodes$y))
```

```
# Plot the density of the mixture
res = (max(nodes$x) - min(nodes$x)) / 50
grd = pdd(ocean.demo.grid, data.frame(x=x, y=y), res=res, sigma=5)
}
```

---

validFVCOMGrid *Check if a* fvcom.grid *instance is valid.*

---

## Description

Check if a fvcom.grid instance is valid.

## Usage

```
validFVCOMGrid(object)
```

## Arguments

| | |
|---|---|
| object | A fvcom.grid instance to check for validity. |

## Value

TRUE if object is well formed, otherwise FALSE.

# Index