Package: BiomassTracking (via r-universe)

November 5, 2026

Version 0.8

Title Biomass tracking package

Depends R (>= 3.3.0)

Description This package is intended to provide an automatic way of estimating and calculating the movement of biomass between different irregularly shaped polygon regions in the ocean on the basis of gridded flowfields. It provides methods or estimating the 'movement tensors' and using them to predict the movement of biomass as well as testing the simulations against a passive-drifter simulation.

License GPL-3

RoxygenNote 6.0.1

Roxygen list(markdown = TRUE)

LazyData yes

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

Date/Publication 2024-05-31 00:07:43 UTC

RemoteUrl https://github.com/AustralianAntarcticDivision/BiomassTracking

RemoteRef HEAD

RemoteSha 2a32af558f59315e6482b03800fc198b71169830

Contents

biomass.compare 2
biomass.tracking 3
BiomassTracking 5
particle.tracking 5
particle.tracking.compare 8
plot.arena 10
prepare.arena 11
quiver 13
randomflowfield 14
Sdata 15

Udata 15
Vdata 16

Index **17**

`biomass.compare` *Compare two different estimations of biomass movement*

Description

This function compares - graphically and analytically - two estimations of biomass movement over the same domain, which will usually be the results of two different algorithms.

Usage

```
biomass.compare(N1, N2, graphics = TRUE, legend = NULL)
```

Arguments

`N1`	An object of class `biomass\_distribution`.
`N2`	An object of class `biomass\_distribution`.
`graphics`	If TRUE, the times series' for the different polygons are plotted.
`legend`	If one of the legend-location keywords (see ?legend) or a location (see ?locator) is given, a legend is drawn at the requested location. If NULL, no legend is drawn.

Details

The objects `N1` and `N2` both contain information about the time points at which they were calculated, to ensure both are comparable. In the rest of this document, `N1` and `N2` refer to the time series only, which are really `N1$bmdist` and `N2$bmdist`.

It is assumed that `N1` and `N2` describe biomass distributions over the same time intervals and over the same domain, as there is no way for the function to check this. `N1` and `N2` have to be matrices of dimension 'number of polygons' x 'number of time steps'.

Value

Two distance measures of the two time series, namely the average squared difference between two corresponding entries in the matrices `N1` and `N2` and the average relative deviation,

`sum((N1-N2)^2)/n`

and

`sum(abs(N1-N2)/(N1+N2)/n`

Author(s)

Thorsten Lenser

Examples

```
## Generate two sample time series with two polygons
N1 = (seq(0,98,length=50)/10)^2
N1 = matrix(c(N1,N1[50:1]), nrow=2, byrow=TRUE)
N2 = N1[1,] + rnorm(50,sd=3)
N2 = matrix(c(N2,N2[50:1]), nrow=2, byrow=TRUE)

## Turn into objects of class "biomass_distribution"
N1 = list(bmdist=N1,times=seq(0,98,length=50))
N2 = list(bmdist=N2,times=seq(0,98,length=50))
class(N1) = "biomass_distribution"
class(N2) = "biomass_distribution"

## Compare them
biomass.compare(N1,N2)
```

`biomass.tracking` *Calculate the movement of biomass from movement tensors*

Description

This function calculates the movement of biomass between given polygons as determined by the movement parameters. It uses the movement tensor calculated by a numerical tracer experiment implemented by the function `particle.tracking`.

Usage

```
biomass.tracking(mk, times, start_setup = NULL, infl_poly = 1,
  infl_time = 10, insert_t_step = NULL, silent = TRUE, S = NULL)
```

Arguments

`mk`	An object of class `movement kernel`, see details.
`times`	A vector giving the time-points at which the output is wanted.
`start_setup`	A vector giving a starting biomass distribution. Specify either this or the next two arguments.
`infl_poly`	The biomass flows in from this polygon. Has to have positive inflow from the boundary.
`infl_time`	The number of time steps over which the biomass is to be inserted. Only used if start_setup==NULL.
`insert_t_step`	The width of the time steps over which the biomass is to be inserted. Only used if start_setup==NULL.
`silent`	If FALSE, the movement is shown in text and graphic and the user has to hit 'enter' after every time step.
`S`	Matrix describing the polygon structure, needed if silent == FALSE.

Details

Two different starting conditions for the flow of biomass can be given: A starting distribution of biomass can be specified or it can flow into the area through the specified inflow polygon over a given number of time steps. In this case, time steps refers to the differences between the first two entries in the `times` vector.

An object of class `movement\_kernel` is a list of the following items:

P: A list of the movement tensors for each polygon, where P[[poly]][i,j,k] describes the probability of a particle leaving polygon `poly` towards polygon `j` in the case that it entered from polygon `i` and is in age class `k`. nk: A list containing the neighbours for each polygon. This is used to translate between the local neighbourhood numbers (used in P) to the global numbers of the polygons. no_age_classes: The number of age classes used in the movement kernel. delta: Width of the age classes used in the movement kernel. retention: Logical to indicate whether retention in the final age class has been enabled or not.

[[poly]: R:[poly

Value

A matrix of dimension 'number of polygons' x 'number of time steps requested'. Each column contains the biomass distribution in the respective time step.

Author(s)

Thorsten Lenser

References

TODO: my report

See Also

`particle.tracking`, `particle.tracking.compare`, `biomass.compare`, `prepare.arena`

Examples

```
data(Udata)
data(Vdata)
data(Sdata)

arena = prepare.arena(Udata,Vdata,Sdata)
plot(arena)

# Use more particles if realistic results are needed
mk = particle.tracking(arena,400,5000,diffusion=0.6,graphics=TRUE)

# Now estimate the biomass movement
N1 = biomass.tracking(mk,seq(0,4900,by=100),infl_poly=2)

# Get a particle tracking result to compare the above to
N2 = particle.tracking.compare(arena,400,100,50,diffusion=0.6,infl_poly=2)
```

```
# Compare the results
biomass.compare(N1,N2)
```

BiomassTracking *BiomassTracking*

Description

BiomassTracking

---

particle.tracking *Estimate a 'movement kernel' describing the movement of biomass.*

Description

A movement kernel is meant to summarise the movement of biomass across polygon-shaped areas in the ocean. It is estimated using a 4-th order Runge-Kutta particle tracking algorithm, which records the moves of passive numerical drifters from polygon to polygon.

Usage

```
particle.tracking(arena, num_particles, no_move_window = 1000,
  end_t_counter = NULL, t_step = NULL, diffusion = 0,
  no_age_classes = "FD", min_nac = 100, max_nac = 500, delta = NULL,
  graphics = FALSE, log = FALSE, from_boundary = TRUE)
```

Arguments

arena	An object of class arena, see details.
num_particles	The number of particles used for the estimation.
no_move_window	Number of time steps in which no move between polygons has to occur in order for the particle tracking to stop. Use 0 to disable.
end_t_counter	The maximal number of time steps used in the Runge-Kutta scheme. Has to be specified if log==TRUE.
t_step	The time step used in the Runge-Kutta scheme. See details for the default value.
diffusion	A diffusion coefficient, see details.
no_age_classes	The number of age classes used in the movement kernel, takes Sturges, FD, scott or an integer number. See details.
min_nac	The minimum number of age classes.
max_nac	The maximum number of age classes.
delta	The width of an age class used in the movement kernel, chosen automatically if NULL.

graphics	If TRUE, the remaining particles at the end of the Runge-Kutta simulation are shown.
log	If TRUE, the path of the particles is saved to a global variable 'log' and displayed if graphics==TRUE. Only use for a small number of particles!
from_boundary	If TRUE, particles are inserted from the boundary, otherwise they are uniformly distributed over the domain. This might be advantageous for plotting purposes if log=TRUE.

Details

The two main bits of work in this function, the Runge-Kutta simulation and the estimation of the movement_ kernel, have been implemented in the C dlls particle_tracking and estimate_P. Even so, the runtime of this function can be quite high when using large numbers of particles and/or a large domain. It is important to note that in the present form, particles can get caught in 'voids' in the flow field, i.e. in places where their velocity is 0. If this happens for too many particles the estimation should be repeated with a higher values of diffusion. A warnings message is given at the end of the simulation, indicating the number of particles left in the domain.

There is considerable space for improvement in replacing the fixed time step Runge-Kutta scheme with a more sophisticated numerical integrator. However, this requires some fiddling with the underlying C functions and has not been tried yet.

An arena object describes the arena in which the function is to be used. It is a list containing elements lat, lon, U, V and S (in that order), where lat and lon are vectors storing the latitude and longitude values of the grid points used, U and V are matrices with the corresponding flow velocities in west-east and south-north direction, and S is a matrix in which each grid points has an integer number, either giving the polygon it belongs to (if > 0) or stating that this grid point lies on land (if == 0).

The default values for the time step in the Runge-Kutta scheme is choosen in such a way that a particle will on average move for 1/10 of the distance between two grid points in each time step. Although a variable time step would be preferable, this yields an accurate estimation in most cases.

Diffusion is included by adding the product of a normal random number and the mean velocity in the either direction to the calculated velocity in that direction. The diffusion coefficient gives the standard variation of the normal random number. Diffusion is needed to get particles out of 'voids', i.e. places in the flow field where the adjacent velocities cancel out and the particles get stuck. It is also physically justifiable as the flow field itself characterises advective motion only.

The main loss of precision in the replication and prediction of movement stems from the management of time, which is realised as a system of 'age classes' in the movement kernel. A particle entering a polygon is in age class 1 and then moves up through the age classes. Each age class has different probabilities of movement attached to it, so that the movement of a biomass depends on the time since it entered its current polygon. Since the choice of the number of age classes and their width delta can have a profound effect, the user can choose whether to trust one of the built-in methods (Sturges, FD, scott) which are usually used to choose the number of classes in a histogram, or specificy his or her own number of age classes. The default uses the Freedman-Diaconis rule (FD), which usually produces the highest number of age-classes and therefore the best precision. Unless the performance of the function biomass_tracking is totally unsatisfactory due to the high number of age-classes, less age-classes should not be used, as the loss of precision can be quite severe.

Value

An object of class `movement\_kernel`, which is a list of the following items:

`P`	A list of the movement tensors for each polygon, where `P[[poly]][i,j,k]` describes the probability of a particle leaving polygon `poly` towards polygon `j` in the case that it entered from polygon `i` and is in age class `k`.
`nk`	A list containing the neighbours for each polygon. This is used to translate between the local neighbourhood numbers (used in `P`) to the global numbers of the polygons.
`no_age_classes`	The number of age classes used in the movement kernel.
`delta`	Width of the age classes used in the movement kernel.

Author(s)

Thorsten Lenser

References

TODO: my report

See Also

`biomass.tracking`, `particle.tracking.compare`, `biomass.compare`, `prepare.arena`

Examples

```
data(Udata)
data(Vdata)
data(Sdata)

arena = prepare.arena(Udata,Vdata,Sdata)
plot(arena)

# Use more particles if realistic results are needed
mk = particle.tracking(arena,400,5000,diffusion=0.6,graphics=TRUE)

# Now estimate the biomass movement
N1 = biomass.tracking(mk,seq(0,4900,by=100),infl_poly=2)

# Get a particle tracking result to compare the above to
N2 = particle.tracking.compare(arena,400,100,50,diffusion=0.6,infl_poly=2)

# Compare the results
biomass.compare(N1,N2)
```

`particle.tracking.compare` *Track the movement of biomass as passive drifters*

Description

This functions tracks the movement of biomass through the given polygons by following the proportionate distribution of a large number of passive drifter (small fluid parcels) trajectories, which are calculated using a 4th-order Runge-Kutta integration scheme.

Usage

```
particle.tracking.compare(arena, num_particles, t_step, end_t_counter,
  start_setup = NULL, infl_poly = 1, infl_time = 10, subdiv = NULL,
  diffusion = 0, silent = TRUE, graphics = FALSE)
```

Arguments

`arena`	An object of class `arena`, see details.
`num_particles`	The number of particles used.
`t_step`	The time step in which the results should be presented.
`end_t_counter`	The number of time steps that should be recorded.
`start_setup`	A vector giving a starting biomass distribution. Specify either this or the next two arguments.
`infl_poly`	The polygon into which the biomass enters. Has to have positive inflow from the outside.
`infl_time`	The number of time steps over which the biomass is to be inserted.
`subdiv`	The number of Runge-Kutta time steps per recorded time step. See details for the default value.
`diffusion`	The coefficient of diffusion, see details.
`silent`	If silent is FALSE, the movement of the biomass will be shown graphically, with the user having to hit 'enter' after every time step
`graphics`	If graphics is TRUE, the remaining particles at the end will be shown. A plot of of the arena has to be done before.

Details

There are two different starting conditions for the flow of biomass: A starting distribution can be given or the biomass can flow into the area through a specified inflow polygon in a certain number of time steps.

A call to the C-function `particle\_tracking\_compare\_inflow` does the bulk of work, which is a lot faster than the same implementation in R due to the greatly increased efficiency of 'for' loops. It is important to note that in the present form, particles can get caught in 'voids' in the flow field, i.e. in places where their velocity is 0. If this happens for too many particles (a warnings

message is given at the end, indicating the number of particles left in the domain), the estimation should be repeated with a higher values of diffusion.

There is considerable space for improvement in replacing the fixed time step Runge-Kutta scheme with a more sophisticated numerical integrator. However, this requires some fiddling with the underlying C functions and has not been tried yet.

An `arena` object describes the arena in which the function is to be used. It is a list containing elements `lat`, `lon`, `U`, `V` and `S` (in that order), where `lat` and `lon` are vectors storing the latitude and longitude values of the grid points used, `U` and `V` are matrices with the corresponding flow velocities in west-east and south-north direction, and `S` is a matrix in which each grid points has an integer number, either giving the polygon it belongs to (if > 0) or stating that this grid point lies on land (if == 0).

The default value for `subdiv` is choosen in such a way that a particle will on average move for 1/10 of the distance between two grid points in each time step. Although a variable time step would be preferable, this yields an accurate estimation in most cases.

Diffusion is included by adding the product of a normal random variable and the mean velocity to the calculated velocity in either direction. The diffusion coefficient gives the standard variation of the normal random number. Diffusion is needed to get particles out of 'voids', i.e. places in the flow field where the adjacent velocities cancel out and the particles get stuck.

Value

A matrix of dimension 'number of polygons' x 'number of time steps', where the last polygon is the 'outside'. Each column holds the biomass distribution in the respective time step.

Author(s)

Thorsten Lenser

References

TODO: My report

See Also

`biomass.tracking`, `particle.tracking.compare`, `biomass.compare`, `prepare.arena`

Examples

```
data(Udata)
data(Vdata)
data(Sdata)

arena = prepare.arena(Udata,Vdata,Sdata)
plot(arena)

# Use more particles if realistic results are needed
mk = particle.tracking(arena,400,5000,diffusion=0.6,graphics=TRUE)

# Now estimate the biomass movement
```

```
N1 = biomass.tracking(mk,seq(0,4900,by=100),infl_poly=2)

# Get a particle tracking result to compare the above to
N2 = particle.tracking.compare(arena,400,100,50,diffusion=0.6,infl_poly=2)

# Compare the results
biomass.compare(N1,N2)
```

---

plot.arena *Show the arena*

Description

Plots a graphical representation of the flow field together with the polygon structure. This is a method for the generic function plot and can therefore be called as plot(arena_object).

Usage

```
## S3 method for class 'arena'
plot(x, arrow_scale = 1, grid = FALSE, ...)
```

Arguments

x	An object of class arena, see details.
arrow_scale	The length of the longest arrow is arrow_scale times the distance between two neighbouring grid points.
grid	If TRUE, a grid is drawn marking the boxes around the grid points.
...	Further arguments to the plot funtion.

Details

An arena object describes the arena in which the function is to be used. It is a list containing elements lat, lon, U, V and S (in that order), where lat and lon are vectors storing the latitude and longitude values of the grid points used, U and V are matrices with the corresponding flow velocities in west-east and south-north direction, and S is a matrix in which each grid points has an integer number, either giving the polygon it belongs to (if > 0) or stating that this grid point lies on land (if == 0).

Author(s)

Thorsten Lenser

References

TODO: my report

See Also

plot, prepare.arena

Examples

```
data(Udata)
data(Vdata)
data(Sdata)

arena = prepare.arena(Udata,Vdata,Sdata)
plot(arena)
```

`prepare.arena` *Prepares the arena for an estimation of movement*

Description

This function prepares the arena, i.e. the flow field and the polygon structure, for use by the other functions in package BiomassTracking.

Usage

```
prepare.arena(file_U = NULL, file_V = NULL, file_S = NULL,
  n = dim(file_U)[2], m = dim(file_U)[1], na.string = "-9999",
  eddy = FALSE, smooth = FALSE, noise_sd = 0)
```

Arguments

`file_U`	Describes the U data, see details.
`file_V`	Describes the V data, see details.
`file_S`	Describes the S data, see details.
`n`	Zonal dimension of the arena, if needed.
`m`	Meridional dimension of the arena, if needed.
`na.string`	The character string denoting NA in the input files.
`eddy`	If TRUE, an eddy is added in the middle of the arena.
`smooth`	If TRUE, the flow field is smoothed with a 3x3 moving average.
`noise_sd`	The standard deviation of Gaussian noise added to the flow field.

Details

The data descriptors `file\_U`, `file\_V` and `file\_S` can come in a number of different formats. If they are filenames, the respective data is loaded from these files. In case the data is already loaded, they can also be data.frames. If they are NULL, a default construction is invoked, where the values `n` and `m` are needed and the flow is straight in west-east direction. They can also be matrices giving the respective data directly.

In the usual case, the data is loaded from files and then prepared to be used. The matrices `U` and `V` are interpolated in every NA position that is not land, while the velocities on land are set to 0. The component perpendicular to land of the sea velocities directly at the land-sea boundary is set to 0 if it points towards land, so that particles wouldn't cross the boundary if an ideal particle tracking algorithm was used.

The data is then rescaled, as the files are supposed to be in m/s format while the particle tracking algorithms act on a longitude/latitude grid. Therefore, the values of 'U' and 'V' are rescaled to degree lon/day and degree lat/day, respectively. Note that this rescaling includes a relative increase of the southern U-velocities compared to the northern U-velocities, because the longitudinal distances are smaller in the South. Also note that the whole algorithm is meant to work in the southern hemisphere and has to be adapted to the Northern Hemisphere if that is wanted.

Value

An object of class `arena`, describing the arena in which the function is too be used. It is a list containing elements `lat`, `lon`, `U`, `V` and `S` (in that order).

`lon`, `lat`	Vectors storing the latitude and longitude values of the grid points used.
`U`, `V`	Matrices with the corresponding flow velocities in west-east and south-north direction.
`S`	A matrix in which each grid points has an integer number, either giving the polygon it belongs to (if > 0) or stating that this grid point lies on land (if == 0).

Warning

This function is written only for the Southern Hemisphere, but can be adapted easily to the Northern Hemisphere.

Author(s)

Thorsten Lenser

References

TODO: my report

See Also

`particle.tracking`, `biomass.tracking`, `particle.tracking.compare`, `biomass.compare`, `plot.arena`

Examples

```
data(Udata)
data(Vdata)
data(Sdata)
arena = prepare.arena(Udata,Vdata,Sdata)
plot(arena)
```

---

quiver *Plot a flow field*

Description

This function plots a flow field given as west-east and south-north velocity values at regularly spaced grid points.

Usage

```
quiver(u, v, xpos = col(u), ypos = row(u), add = FALSE, scale = 1)
```

Arguments

u	The west-east velocities as a matrix.
v	The south-north velocities as a matrix.
xpos	Positions of the arrowbases in x-direction.
ypos	Positions of the arrowbases in y-direction.
add	If TRUE, the flow field is added to the current plot.
scale	The length of the longest arrow is 'scale' * 'distance between grid points'.

Author(s)

Thorsten Lenser

See Also

plot.arena

Examples

```
data(Udata,Vdata,Sdata)
arena = prepare.arena(Udata,Vdata,Sdata)

# First use quiver only
quiver(arena$U,arena$V)

# Now use larger arrows, looks better in this case
quiver(arena$U,arena$V,scale=5)
```

```
# Together with the polygon structure
plot(arena,arrow_scale=5)
```

`randomflowfield` *Generate one direction of a random flow field*

Description

Generate a matrix meant to be used as one direction of a random flow field.

Usage

```
randomflowfield(n, m, std_dev)
```

Arguments

`n`	Zonal (x) extent of the flow field.
`m`	Meridional (y) extent of the flow field.
`std_dev`	Standard deviation used for the random walk, see details.

Details

This function proceeds in the following way: A starting value, either 1 or -1, is drawn at random. This is assigned to the element in the top-left corner of the matrix. The other values are then calculated as the sum of a normal random variable with standard deviaton `std\_dev` and their left neighbour (top row) or the mean of their left and their upper neighbour (rest).

Value

A matrix meant to be used as one direction of a random flow field.

Author(s)

Thorsten Lenser

See Also

`prepare.arena`

Examples

```
U = randomflowfield(30,30,0.3)
V = randomflowfield(30,30,0.3)
quiver(U,V)
```

Sdata *Polygon structure and land positions*

Description

Example data set of a polygon structure over the Kerguelen plateau. 0 denotes land and positive integer values give the number of the polygon the respective grid point belongs to.

Format

A matrix of integer values.

Source

Thorsten Lenser, just an example.

Udata *EW component of a flow field*

Description

Example data set of u-components of the flow field over the Kerguelen plateau. Derived from satellite measurements of sea surface height.

Format

Three columns, containing longitude, latitude and current speed in m/s. Unknown values are given as -9999.

Source

Belinda Ronai, Australian Antarctic Division, unpublished data.

Vdata *NS component of a flow field*

Description

Example data set of v-components of the flow field over the Kerguelen plateau. Derived from satellite measurements of sea surface height.

Format

Three columns, containing longitude, latitude and current speed in m/s. Unknown values are given as -9999.

Source

Belinda Ronai, Australian Antarctic Division, unpublished data.

Index

∗ **datasets**
Sdata, 15
Udata, 15
Vdata, 16
∗ **misc**
biomass.compare, 2
biomass.tracking, 3
particle.tracking, 5
particle.tracking.compare, 8
plot.arena, 10
prepare.arena, 11
quiver, 13
randomflowfield, 14

biomass.compare, 2, *4*, *7*, *9*, *12*
biomass.tracking, 3, *7*, *9*, *12*
BiomassTracking, 5
BiomassTracking-package
*(*BiomassTracking*)*, 5

particle.tracking, *4*, 5, *12*
particle.tracking.compare, *4*, *7*, 8, *9*, *12*
plot, *11*
plot.arena, 10, *12, 13*
prepare.arena, *4*, *7*, *9*, 11, *11*, *14*

quiver, 13

randomflowfield, 14

Sdata, 15

Udata, 15

Vdata, 16