Package ‘move2’

July 16, 2023

Title  Processing and Analysing Animal Trajectories
Version  0.2.2
Description  Tools to handle, manipulate and explore trajectory data, with an emphasis on data from tracked animals. The package is designed to support large studies with several million location records and keep track of units where possible. Data import directly from 'movebank' <https://www.movebank.org/cms/movebank-main> and files is facilitated.
License  GPL (>= 3)
Encoding  UTF-8
URL  https://bartk.gitlab.io/move2/
BugReports  https://gitlab.com/bartk/move2/-/issues
RoxygenNote  7.2.3
Depends  R (>= 4.1.0)
Imports  methods, assertthat, sf (>= 1.0.12), rlang, units, tidyselect, dplyr (>= 1.1.0), tibble, vroom (>= 1.6.1), cli, vctrs (>= 0.5.2), bit64
Suggests  knitr, askpass, digest, keyring, xml2, curl, magrittr, purrr, ggplot2, testthat (>= 3.0.0), rmarkdown, lwgeom, s2, move, raster, withr, lubridate, naturalearth, naturalearthdata, circular, tidyr
Config/testthat/edition  3
Config/testthat/parallel  false
VignetteBuilder  knitr
NeedsCompilation  no
Author  Bart Kranstauber [aut, cre] (<https://orcid.org/0000-0001-8303-780X>), Kamran Safi [aut] (<https://orcid.org/0000-0002-8418-6759>), Anne K. Scharf [aut] (<https://orcid.org/0000-0002-3357-8533>)
Maintainer  Bart Kranstauber <b.kranstauber@uva.nl>
Repository  CRAN
Date/Publication  2023-07-16 14:40:02 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filter_track_data</code></td>
<td>2</td>
</tr>
<tr>
<td><code>movebank_download_study</code></td>
<td>4</td>
</tr>
<tr>
<td><code>movebank_get_vocabulary</code></td>
<td>7</td>
</tr>
<tr>
<td><code>movebank_handle</code></td>
<td>9</td>
</tr>
<tr>
<td><code>movebank_store_credentials</code></td>
<td>10</td>
</tr>
<tr>
<td><code>mt_as_move2</code></td>
<td>11</td>
</tr>
<tr>
<td><code>mt_as_track_attribute</code></td>
<td>12</td>
</tr>
<tr>
<td><code>mt_azimuth</code></td>
<td>13</td>
</tr>
<tr>
<td><code>mt_distance</code></td>
<td>14</td>
</tr>
<tr>
<td><code>mt_example</code></td>
<td>15</td>
</tr>
<tr>
<td><code>mt_filter_movebank_visible</code></td>
<td>16</td>
</tr>
<tr>
<td><code>mt_filter_per_interval</code></td>
<td>17</td>
</tr>
<tr>
<td><code>mt_filter_unique</code></td>
<td>18</td>
</tr>
<tr>
<td><code>mt_interpolate</code></td>
<td>20</td>
</tr>
<tr>
<td><code>mt_is_track_id_cleaved</code></td>
<td>21</td>
</tr>
<tr>
<td><code>mt_read</code></td>
<td>23</td>
</tr>
<tr>
<td><code>mt_segments</code></td>
<td>24</td>
</tr>
<tr>
<td><code>mt_sim_brownian_motion</code></td>
<td>25</td>
</tr>
<tr>
<td><code>mt_stack</code></td>
<td>26</td>
</tr>
<tr>
<td><code>mt_time</code></td>
<td>28</td>
</tr>
<tr>
<td><code>mt_time_column</code></td>
<td>29</td>
</tr>
<tr>
<td><code>mt_track_data</code></td>
<td>30</td>
</tr>
<tr>
<td><code>mt_track_id</code></td>
<td>31</td>
</tr>
<tr>
<td><code>mt_track_lines</code></td>
<td>32</td>
</tr>
<tr>
<td><code>to_move</code></td>
<td>33</td>
</tr>
</tbody>
</table>

---

**Index**

<table>
<thead>
<tr>
<th>Filter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filter_track_data</code></td>
<td>35</td>
</tr>
</tbody>
</table>

**Description**

- `filter_track_data` filter data based on a track attribute (e.g. select all juveniles). Based on `filter`.
- `select_track_data` keep or drop attributes in the track data. Based on `select`.
- `mutate_track_data` create or modify attributes in the track data. Based on `mutate`.
- `group_by_track_data` group by one or more attribute of the track data (e.g. group by sex, by taxon, by life stage, etc). Based on `group_by`. 

**dplyr functions to manipulate the track data**

- `filter` (filtering based on a condition)
- `select` (selecting columns)
- `mutate` (creating or modifying columns)
- `group_by` (grouping by one or more columns)
filter_track_data

Usage

filter_track_data(.data, ..., .track_id = NULL)

select_track_data(.data, ...)

mutate_track_data(.data, ...)

group_by_track_data(
  .data,
  ..., 
  .add = FALSE,
  .drop = dplyr::group_by_drop_default(.data)
)

Arguments

.data the move2 object

... The identifiers of one or more tracks to select or selection criteria based on track attributes

.track_id A vector of the ids of the tracks to select

.add see original function docs group_by

.drop see original function docs group_by

Value

a move2 object

Examples

## simulating a move2 object with 4 tracks
data <- mt_sim_brownian_motion(tracks = letters[1:4])

## retaining tracks "b" and "d"
data |> filter_track_data(.track_id = c("b", "d"))

## adding the attribute "sex" to the track data
data <- data |
  mutate_track_data(sex = c("m", "f", "f", "m"))

## retaining tracks of females
data |> filter_track_data(sex == "f")
movebank_download_study

Description

- movebank_download_study downloads a complete study from Movebank by the study id or name.
- movebank_download_deployment downloads all tag, individual and deployment information and merges it into one data.frame
- movebank_download_study_info downloads all study level information, either for all studies, one study with the argument id or a subset, for example, license_type = "CC_0"
- movebank_retrieve is a more flexible function for retrieving information directly from the api.
- movebank_get_study_id using a character string retrieve the associated study id.

Usage

movebank_download_study(
    study_id,
    attributes = "all",
    ..., remove_movebank_outliers = TRUE
)
movebank_download_study_info(...)
movebank_download_deployment(study_id, ...)
movebank_retrieve(
    entity_type = NA,
    ..., handle = movebank_handle(),
    rename_columns = FALSE,
    omit_derived_data = TRUE,
    convert.spatial.columns = TRUE,
    progress = vroom::vroom_progress()
)
movebank_get_study_id(study_id, ...)

Arguments

study_id the study id as a number or a character string can be used to identify a study. This character string needs to be unique enough to identify one and only one study. Argument applicable to all functions.
**movebank_download_study**

attributes a character vector with the event data attributes to download. By default "all" are downloaded, this make it slightly slower, to speed up NULL can be used as it reduces it to the minimal set of required attributes (only for location data). Alternatively a vector of attributes can be provided (the minimal ones are automatically added). Argument applicable to movebank_download_study and movebank_retrieve. See 'Details' for more information.

... arguments added to the movebank api call. See 'Details' for some common arguments.

remove_movebank_outliers if TRUE outliers according to the movebank logic are removed. This should correspond to the visible attribute in movebank. Argument applicable to movebank_download_study and movebank_retrieve.

entity_type the entity type of the data requested from movebank (e.g. "study", "tag", "event"). Alternatively it can be the complete api url for testing purposes. Argument applicable to movebank_retrieve.

handle the curl handle used to perform the api call, generally this is extracted from the system keyring if correctly set up with movebank_store_credentials. Argument applicable to all functions.

rename_columns if TRUE column names of properties that are repeated in the api output (e.g. id, local_identifier and comments) will be appended with the entity_type (e.g. "tag", "individual"). Argument applicable to movebank_download_study, movebank_download_study_info, movebank_retrieve.

omit_derived_data derived data (e.g. timestamp_start, timestamp_end, number_of_events and number_of_deployments) is omitted from the result. The default is TRUE as this data quickly becomes unrepresentative if the results are processed. However in some occasions it might be worth retrieving it, for example if you want to identify deployment periods without downloading all data. Argument applicable to movebank_download_study and movebank_retrieve.

convert_spatial_columns if TRUE column pairs containing spatial data will be converted to an sfc column. Argument applicable to all functions.

progress if TRUE a progress bar will be displayed. More details can be found here [vroom](https://example.com).

Argument applicable to movebank_download_study and movebank_retrieve.

**Details**

Caution, when downloading data with movebank_download_study without specifying the sensor in the argument sensor_type_id (see below), all data of all sensors will be downloaded, but only the attributes of location sensors will be included. We recommend to always specify the sensor(s) to ensure that all associated attributes are downloaded. Use e.g. movebank_download_study_info(study_id=my_study_id)$sensor_type_ids to find out which sensors are available in a given study. attributes = "all" is the default, and it will include only location sensor attributes if no sensor is specified in sensor_type_id. When sensors are specified, it will download all associated attributes of all sensors. attributes = NULL should only be used when downloading location data (by specifying the sensor), as only timestamp, location and track id is downloaded. To specify only a subset of attributes to download, check the list of attributes available for a specific sensor (e.g. GPS) in a given study, use
movebank_retrieve(entity_type = "study_attribute", study_id = myStudyID, sensor_type_id = "gps")$short_name (more details in "Downloading data from movebank" vignette).

The api is quite flexible for adjusting requests. This is elaborately documented in the movebank api documentation. To identify the available arguments, please note that movebank_download_study is based on the entity_type "event", movebank_download_study_info on the entity_type "study" and movebank_download_deployment on the entity_type "deployment", "individual" and "tag".

Here a list of a few arguments that are common for the entity_type "event":

- sensor_type_id can be used to restrict the download to specific sensors. It can be either a character or an integer with the tag_type. For a full list of options see: movebank_retrieve(entity_type='tag_type', values from the id and external_id columns are valid.
- timestamp_start and timestamp_end can be used to limit the temporal range to download. This argument can either be formatted as a POSIXct timestamp or a character string (e.g. "20080604133046000" (yyyyMMddHHmmssSSS))
- event_reduction_profile might be useful to reduce the data downloaded (e.g. daily locations) possible values are character strings (e.g. "EURING_01"). For details see the movebank api documentation. Note that for the time being the required attributes need to be explicitly stated (e.g. attributes = NULL) as "all" does not work with the current movebank api.
- individual_local_identifier for selecting one or more individuals by the local identifier

For more elaborate usage see vignette("movebank", package='move2')

Value

movebank_download_study returns a move2 object.
movebank_retrieve, movebank_download_deployment, movebank_download_study_info return a data.frame/tbl.
movebank_get_study_id returns a big integer.

See Also

Other movebank-download: movebank_handle(), movebank_store_credentials()

Examples

## Not run:
## download entire study (all data of all sensors)
movebank_download_study_info(study_id = myStudyID)$sensor_type_ids
movebank_download_study(2911040, sensor_type_id = c("gps", "acceleration"))

## download data of one individual
movebank_download_study(2911040, individual_local_identifier = "unbanded-160")

## download gps data for multiple individuals
movebank_download_study(2911040, sensor_type_id = "gps", individual_local_identifier = c("1094-1094", "1103-1103"))
movebank_download_study(2911040,
movebank_get_vocabulary

Retrieve information from the movebank vocabulary describing the columns

Description

Retrieve information describing the columns from the 'Movebank Attribute Dictionary'

Usage

movebank_get_vocabulary()
movebank_get_vocabulary

labels,
xml = "http://vocab.nerc.ac.uk/collection/MVB/current/",
omit_deprecated = TRUE,
return_type = c("definition", "list", "xml", "uri")
)

Arguments

labels       Either a character vector with the column names to look up or a move2 object
              from which the column names will be extracted. If no argument is provided all
              movebank terms are returned
xml          Either a connection to the movebank vocabulary xml, a path to the vocabulary
              file or an url where it can be downloaded. The later is the default. By download-
              ing the xml yourself the function will speed up and become independent of an
              internet connection being available.
omit_deprecated
              If concepts are marked deprecated they are omitted from the set of possible
              labels.
return_type  A character scalar identifying the desired return type, see details for more infor-
              mation on the specific types.

Details

This function can return data in several formats (see return_type argument):

• definition A text string with the description of the term.
• list A list with all information for each term.
• xml A xml_node with the definition.
• uri A link to the full definitions page.

Value

A named list of the selected return_type, note that if deprecated are not omitted duplicated names
can occur.

Examples

# list of all terms used in movebank
movebank_get_vocabulary()
# Count the units used in movebank
movebank_get_vocabulary() |> unlist() |> grep(pattern = "Units:", value = TRUE) |> sub(replacement = "", pattern = ".*Units: ") |> sub(replacement = "", pattern = "; .*") |> table() |> sort()
# different return types
movebank_handle

movebank_get_vocabulary("light-level", return_type = "definition")
movebank_get_vocabulary("light-level", return_type = "xml")
movebank_get_vocabulary("light-level", return_type = "uri")
movebank_get_vocabulary("light-level", return_type = "list")

# get definitions of all column names of a move2 object
data <- mt_read(mt_example())
movebank_get_vocabulary(data)

movebank_handle  Create a curl handle for accessing movebank

Description
Create a curl handle for accessing movebank

Usage
movebank_handle(username = NULL, password = NULL)

Arguments

username  Optionally a username as a character string
password  Optionally a password as a character string

Details
If no credentials are provided the function tries to retrieve the username and password from the
system keyring using the keyring package. If a username is provided but no password it is requested
using askpass

Value
A handle that can be added to a request made with curl

See Also
Other movebank-download: movebank_download_study(), movebank_store_credentials()

Examples
movebank_handle("test_user", "test_password")
movebank_store_credentials

Description
The function stores the credentials for accessing movebank, by default it checks if accessing movebank is possible, and fails when either the credentials are invalid or movebank cannot be reached. The force option can be used to override this. Once credentials are stored, these functions are not needed again as all call to movebank can use the credentials from the keyring.
For more details on the usage of the keyring, how passwords are handled and handling multiple accounts see vignette("movebank", package="move2")

Usage
movebank_store_credentials(
  username,
  password,
  key_name = getOption("move2_movebank_key_name"),
  force = FALSE
)

movebank_remove_credentials(key_name = getOption("move2_movebank_key_name"))

Arguments
username A string with the movebank username
password Either a string or missing, if missing then the password is asked for using askpass
key_name The name of the key in the keyring. By default this is stored in getOption("move2_movebank_key_name") this might be useful if you have multiple accounts
force If TRUE, when accessing movebank fails the key is stored anyway.

Value
TRUE invisible if successful

See Also
Other movebank-download: movebank_download_study(), movebank_handle()

Examples
## Not run:
movebank_store_credentials("bart")
## End(Not run)
Create a new move2 object

Create a new move2 object from a data.frame, sf, Move or MoveStack object

Usage

```r
mt_as_move2(x, ...)
```

```r
## S3 method for class 'sf'
mt_as_move2(x, time_column, track_id_column, track_attributes = "", ...)
```

```r
## S3 method for class 'data.frame'
mt_as_move2(x, time_column, track_id_column, track_attributes = "", ...)
```

```r
## S3 method for class '.MoveTrack'
mt_as_move2(x, ...)
```

Arguments

- `x` A data.frame, sf, Move or MoveStack object
- `...` Additional arguments passed to `st_as_sf` if `x` is a data.frame, see the details below for more information
- `time_column` The name of the column in `x` containing timestamps
- `track_id_column` The name of the column in `x` containing the track identities
- `track_attributes` The name(s) of the column(s) that contain track level attributes

Details

Frequently used arguments to `st_as_sf` are:

- `coords` a character vector indicating the columns used as coordinates, the length is generally two, for `x` and `y`, but can also be more if `z` is included
- `sf_column_name` if a geometry column is present the name of the geometry column to use as coordinates as a character scalar
- `crs` the coordinate reference system to use, either as character, number or a `crs` object for more details see `st_crs`
- `na.fail` normally when the coordinate columns are converted to spatial points NA values cause an error, if set to `FALSE` empty points are allowed

Value

A move2 object
### Examples

```r
## create a move2 object from a data.frame and defining projection
n <- 5
data <- data.frame(
  x = cumsum(rnorm(n)), y = cumsum(rnorm(n)),
  time = seq(n), track = "a"
)
mt_as_move2(data,
  coords = c("x", "y"), time_column = "time",
  track_id_column = "track"
) |> sf::st_set_crs(4326L)

## Dealing with empty coordinates:
## If the data frame contains NA coordinates, the coords argument in sf
## will fail. An alternative is to first create an sfc column,
## or to use the na.fail argument
nn <- 3
data <- data.frame(
  x = c(cumsum(rnorm(n)), rep(NA, nn)), y = c(cumsum(rnorm(n)), rep(NA, nn)),
  time = seq(n + nn), track = "a",
  sensor = c(rep("sensor1", n), rep("sensor2", nn)),
  sensor2values = c(rep(NA, n), runif(nn))
)
mt_as_move2(data,
  coords = c("x", "y"),
  na.fail = FALSE,
  time_column = "time",
  track_id_column = "track"
)

## create a move2 object from a sf object
data$geometry <- sf::st_sfc(apply(data[, c("x", "y")], 1, sf::st_point, simplify = FALSE))
mt_as_move2(data,
  sf_column_name = c("geometry"), time_column = "time",
  track_id_column = "track"
)
```

---

**mt_as_track_attribute**  Move one or more columns to track attributes or event attributes

**Description**

- **mt_as_track_attribute**: move a column from the event to the track attributes
- **mt_as_event_attribute**: move a column from the track to the event attributes
**mt_azimuth**

Usage

```
mt_as_track_attribute(x, ...)
mt_as_event_attribute(x, ...)
```

Arguments

- `x`: The move2 object
- `...`: the names of columns to move, it is also possible to use helpers.

Details

When one or more of the selected columns contain more than one unique value per track an error is raised.

Value

An object of the class move2 with the column(s) moved

See Also

- `mt_track_data()` to retrieve the track attribute table
- `mt_set_track_data()` to replace attribute table with new values

Examples

```
sim_data <- mt_sim_brownian_motion()
sim_data$sex <- "female"

# different ways to move column "sex" from event to track attribute
sim_data |> mt_as_track_attribute(sex)
sim_data |> mt_as_track_attribute(starts_with("s"))
sim_data |> mt_as_track_attribute(any_of(c("sex", "age")))
```

---

**mt_azimuth**

*Calculate azimuths or turn angles*

**Description**

- `mt_azimuth`: calculates the heading/azimuth/direction of movement of each segment between consecutive locations of a track.
- `mt_turnangle`: calculates the relative angle between consecutive segments.

Usage

```
mt_azimuth(x)
mt_turnangle(x)
```
`mt_distance`  

**Arguments**  

`x`  

A `move2` object. Timestamps must be ordered within tracks and only contain location data (See 'Details').

**Details**  

`mt_is_time_ordered_non_empty_points` can be used to check if the timestamps are ordered and if the object only contains location data. To omit empty locations, use e.g., `dplyr::filter(x, !sf::st_is_empty(x))`.  

Currently, the calculation of both angles is only implemented for data in a geographic coordinate system and data without coordinates reference system. To reproject the data into long/lat use e.g., `sf::st_transform(x, crs="EPSG:4326")`.  

Azimuths for geographic coordinates are calculated using `lwgeom::st_geod_azimuth()`. The angles are relative to the North pole.

**Value**  

A vector of angles in radians (between -pi and pi).  

In `mt_azimuth` north is represented by 0, positive values are movements towards the east, and negative values towards the west. The last value for each track will be `NA`. In `mt_turnangle` negative values are left turns and positive right turns. The first and the last value for each track will be `NA`.

**See Also**  

Other track-measures: `mt_distance()`, `mt_time()`

**Examples**  

```r  
  x <- mt_read(mt_example())[330:340, ]  
  mt_azimuth(x)  
  mt_turnangle(x)  
```

---

**Description**  

The distance or speed is calculated between consecutive locations

**Usage**  

```r  
  mt_distance(x)  
  mt_speed(x)  
```

**Arguments**  

`x`  

A `move2` object. Timestamps must be ordered within tracks and only contain location data (See 'Details').
Details

`mt_is_time_ordered_non_empty_points` can be used to check if the timestamps are ordered and if the object only contains location data. To omit empty locations use e.g. `dplyr::filter(x, !sf::st_is_empty(x)).` Distances are calculated using `sf::st_distance`.

Value

a vector of the same length as the `move2` object containing the distances/speeds between locations. Each element is the distance/speed to the next location. The last value for each track will be `NA`. Units are included when the data have a coordinate reference system set.

See Also

Other track-measures: `mt_azimuth()`, `mt_time()`

Examples

```
## distance between consecutive locations
mt_sim_brownian_motion() |> mt_distance() |> head()
## When the data has a coordinate reference system set, units are included
dist <- mt_sim_brownian_motion(1:4) |> sf::st_set_crs(4326L) |> mt_distance() 
dist
## transform units of output
units::set_units(dist, km)
## speed between consecutive locations
mt_sim_brownian_motion() |> mt_speed()
## When projections are provided units are included
speed_calc <- mt_read(mt_example())[330:340, ] |> mt_speed()
speed_calc
## transform units of output
units::set_units(speed_calc, m/s)
mt_read(mt_example())[330:340, ] |> sf::st_transform("+proj=aeqd +units=km +lon_0=-73.9 +lat_0=42.7") |> mt_speed()
```
**Description**

The `move2` package comes with an example data file that is directly downloaded from `movebank` and gz compressed for reduction in package size.

**Usage**

```r
mt_example()
```

**Details**

The example dataset is the study "Martes pennanti LaPoint New York" (study id: 69258089), shared under the CC-BY-NC license. For more information on the data see LaPoint et al. (2013) Landscape Ecology. doi: 10.1007/s10980-013-9910-0

**Value**

The path to the example file of the `move2` package

**See Also**

- `mt_read()` to read in the file

**Examples**

```r
# Get path to one example
mt_example()
mt_read(mt_example())
```

---

**mt_filter_movebank_visible**

*Identify records that are not outliers according to the logic used in movebank*

**Description**

- `mt_filter_movebank_visible`: returns a `move2` object with all visible data, i.e., excluding all records marked as outliers according to the logic used in movebank (See Details)
- `mt_movebank_visible`: indicates with TRUE the visible records, and with FALSE those marked as outliers according to the logic used in movebank (See Details)

**Usage**

```r
mt_filter_movebank_visible(x)
mt_movebank_visible(x)
```
Arguments

  x       a move2 object

Details

These functions rely on the columns 'visible', 'algorithm_marked_outlier', 'import_marked_outlier', 'manually_marked_outlier', and/or 'manually_marked_valid'. All of them are expected to be logical. More details can be found in the movebank vocabulary

Value

  mt_movebank_visible returns a logical vector indicating the records that are valid.
  mt_filter_movebank_visible returns a filtered move2 object

See Also

  Other filter: mt_filter_per_interval(), mt_filter_unique()

Examples

  m <- mt_read(mt_example())
  table(mt_movebank_visible(m))
  mt_filter_movebank_visible(m)
Arguments

x  a move2 object
...
additional arguments to `mt_per_interval` and `floor_date`, for example the
day that starts the week
criterion the criterion of what record to select per time interval
unit the time units to select the first record per. This can also be a multiple of units
(e.g. "30 seconds"). For more details see `floor_date`.

Value

`mt_per_interval` returns a logical vector indicating the selected records.
`mt_filter_per_interval` returns a filtered move2 object.

See Also

Other filter: `mt_filter_movebank_visible()`, `mt_filter_unique()`.

Examples

data <- mt_sim_brownian_motion(as.POSIXct("2022-1-1") + 1:10)
data |> mt_filter_per_interval(criterion = "random")
data |> mt_filter_per_interval(unit = "3 secs")
data[mt_per_interval(data, unit = "6 secs")]
Arguments

- **x**: The `move2` object to filter
- ...: Arguments passed on to the `mt_unique` function
- **criterion**: The criterion to decide what records to filter out. For more information see `Details` below.
- **additional_columns**: In some cases different sensors or tracking devices might have the same combination of time and track identifier. It might, for example, be desirable to retain records from an accelerometer and GPS recorded at the same time. This argument can be used to indicate additional column to include in the grouping within which the records should not be duplicated. See the examples below for its usage.

Details

To make an informed choice of how to remove duplicates, we recommend to first try to understand why the data set has duplicates.

Several methods for filtering duplicates are available the options can be controlled through the criterion argument:

- **"subsets"**: Only records that are a subset of other records are omitted. Some tracking devices first transmit an smaller dataset that does not contain all information, therefore some records may be the same as others only containing additional NA values. This strategy only omits those (duplicated) records. As a result duplicates that contain unique information are retained, the dataset is thus not guaranteed to not have unique records afterwards.
- **"sample"**: In this case one record is randomly selected from the duplicated records.
- **"first"**: Select the first location from a set of duplicated locations. Note that reordering the data will affect which record is selected. For movebank data no specific order is enforced, ensure that the order of the locations is like you expect (same goes for "last").
- **"last"**: Select the last location from a set of duplicated locations.

Value

- `mt_unique` returns a logical vector indicating the unique records.
- `mt_filter_unique` returns a filtered `move2` object

See Also

Other filter: `mt_filter_movebank_visible()`, `mt_filter_per_interval()`

Examples

```
m <- mt_sim_brownian_motion(1:2)[rep(1:4, 4), ]
m$sensor_type <- as.character(gl(2, 4))
m$sensor_type_2 <- as.character(gl(2, 8))
table(mt_unique(m, "sample"))
mt_filter_unique(m[, c("time", "track", "geometry")])
mt_filter_unique(m[, c("time", "track", "geometry", "sensor_type")],
```
mt_interpolate = sensor_type
if (requireNamespace("dplyr")) {
  mt_filter_unique(m, additional_columns = across(all_of(c("sensor_type", "sensor_type_2")))))
}  
mt_filter_unique(m, "sample")
mt_filter_unique(m, "first")
m$sensor_type[1:12] <- NA
mt_filter_unique(m[ , c("time", "track", "geometry", "sensor_type")])
# Sometimes it is desirable to not consider specific columns for finding the unique records
# For example the record identifier like 'event_id' in movebank This can be done by reducing
# the data.frame used to identify the unique records e.g.:
if (requireNamespace("dplyr")) {
  m$event_id <- seq_len(nrow(m))
  m[mt_unique(m |> dplyr::select(-event_id, -ends_with("type_2"))), ]
  # Note that because we subset the full original data.frame the columns are not lost
}

mt_interpolate | Linearly interpolate locations

Description

Linear interpolation along the straight line between consecutive locations.

Usage

mt_interpolate(x, time, max_time_lag, omit = FALSE)

Arguments

x A move2 object
time The times to interpolate to, if missing the interpolation is to the empty locations. Alternatively if the timestamps in x are POSIXct then also an interval can be provided. For details on the interval specification see floor_date.
max_time_lag The maximal time lag to interpolate over, if not provided any interval is interpolated
omit If the original location that do not match a value in time should be omitted. This only takes affect when time is not missing.

Details

Each interpolation is done along a straight path from the previous to the next location. Interpolation is done with st_line_sample when there is no CRS provided and s2_interpolate_normalized when the data has a projection.

Value

A move2 object with the interpolated locations
Examples

data <- mt_sim_brownian_motion(t = c(0, 0.6, 3, 3.5))
## interpolating at specific times
mt_interpolate(data, c(.5, 1.5, 2.5))
## interpolating to empty locations
data$geometry[c(1, 3)] <- sf::st_point() ## creating empty locations
mt_interpolate(data)
fishers <- mt_read(mt_example())[1:200, ]
mt_interpolate(fishers, "2 hours")
## omit the original records
mt_interpolate(fishers, "2 hours", omit = TRUE)

mt_is_track_id_cleaved

Functions for asserting properties of a move2 object

Description

- `mt_is_track_id_cleaved()` asserts all tracks are grouped in the data set, they occur consecutively.
- `mt_is_time_ordered()` checks if all tracks are groups and if timestamps within a track are ascending (i.e. the time differences between successive locations are equal or above 0).
- `mt_has_unique_location_time_records()` checks if all records with a location have a unique timestamp (i.e. checks for duplicated timestamps within a track).
- `mt_is_time_ordered_non_empty_points()` this assertion combines the `mt_is_time_ordered()` and `mt_has_no_empty_points()` assertions and thus ensures that each record has a location and timestamps are ordered.
- `mt_has_no_empty_points()` asserts all geometries are points and that there are no empty points.
- `mt_is_move2()` asserts x inherits the class move2

Usage

```
mt_is_track_id_cleaved(x)
mt_is_time_ordered(x, non_zero = FALSE)
mt_has_unique_location_time_records(x)
mt_is_time_ordered_non_empty_points(x, non_zero = FALSE)
mt_has_no_empty_points(x)
mt_is_move2(x)
```
Arguments

- **x**: a `move2` object
- **non_zero**: If `TRUE` only intervals longer than 0 are considered ordered (i.e. no coinciding timestamps), if `FALSE` also 0 intervals are considered ordered

Details

For these functions an `on_failure` error function is defined. This results in meaningful error messages when the function is used in combination with `assert_that`. These functions can also be used in normal logical operations as `TRUE` or `FALSE` is returned.

Value

A logical value if the asserted property is `TRUE` or `FALSE`.

Examples

```r
# examples of what to do if assertion if FALSE
n <- 8
data <- data.frame(x = cumsum(rnorm(n)), y = cumsum(rnorm(n)),
                   time = seq(n), track = sample(c("a", "b"), size=n, replace=TRUE))
data <- rbind(data, data[sample(nrow(data), 2),])  # adding duplicate timestamps
mv <- mt_as_move2(data, coords = c("x", "y"),
                   time_column = "time",
                   track_id_column = "track")
mv$geometry[c(1, 3)] <- sf::st_point()  # adding empty locations
mt_is_track_id_cleaved(mv)
mv <- dplyr::arrange(mv, mt_track_id(mv))
mt_is_time_ordered(mv)
mv <- dplyr::arrange(mv, mt_track_id(mv), mt_time(mv))
mt_has_unique_location_time_records(mv)
mv <- mt_filter_unique(mv)
mt_has_no_empty_points(mv)
mv <- dplyr::filter(mv, !sf::st_is_empty(mv))

# example of using the assertions with assertthat
if (requireNamespace("assertthat")) {
  m <- mt_sim_brownian_motion(t = 1:2, tracks = 2)
  assertthat::see_if(mt_is_track_id_cleaved(m))
  assertthat::see_if(mt_is_track_id_cleaved(m[c(3, 1, 2, 4),]))
  assertthat::see_if(mt_is_time_ordered(m[c(2:1, 3, 4),]))
  assertthat::see_if(mt_has_unique_location_time_records(m[c(1, 1, 2, 3, 4),]))
  assertthat::see_if(mt_is_move2(m$time))
}
```
Description

Reading files downloaded from movebank

Usage

mt_read(file, ...)

Arguments

file The file path to read or a R connection (for details see connections)
...
Arguments passed on to vroom, for example col_select

Details

Files can be gz compressed and if the same columns are present multiple files can be read simultaneously. Using the pipe command in R and some command line tools it is possible to select specific days or months.

When using the col_select argument of vroom it is possible to speed up file reading considerably while reducing memory consumption. Especially columns containing acceleration values can become quite large.

For files that contain both a individual-local-identifier and a tag-local-identifier column a check is performed if individuals have been wearing multiple tags over time. If this is the case tracks are created based on the combination of both id’s. A new column names individual-tag-local-identifier in created, which will correspond to the track ids. This somewhat resembles the movebank logic however the track ids do not necessarily correspond to the deployments in movebank as this information is not contained in exported csv’s.

Value

An object of the class move2

See Also

• mt_example() for the path to an example file.

Examples

path_fishers <- mt_example()
mt_read(path_fishers)
## Reduce the mount of data read this might provide memory advantages
## and speed up reading
mt_read(path_fishers, col_select = c(
  "location-long", "location-lat"),
mt_segments

Create a LINESTRING for each track segment

Description

Creates a LINESTRING for each segment between consecutive points within a track.

Usage

mt_segments(x)

Arguments

x A move2 object.

Details

The last location of each track is formed by a POINT as no segment can be formed.

Value

A sfc object containing LINESTRINGs for each segment of a trajectory.

Examples

track <- mt_sim_brownian_motion()
mt_segments(track)
## adding the segments as an attribute to the move2 object
track$segments <- mt_segments(track)
Description

Creates a move2 object with simulated data following a Brownian motion

Usage

```r
mt_sim_brownian_motion(
  t = 1L:10L,
  sigma = 1L,
  tracks = 2L,
  start_location = c(0L, 0L),
  track_id = NULL
)
```

Arguments

- **t**: a vector of timestamps, numeric values or times to simulate for. If multiple tracks are created this vector will be used for all of them, alternatively a list with a vector per track can be provided.
- **sigma**: The Brownian motion variance movement rate \( \sigma \) either as scalar number or a vector with a number per segment. Not that this argument is the movement rate so the motion variance will be adjusted for the length of the interval. Alternatively a function that is integrated over time in the simulation function, this function needs to be vectorized (if needed see Vectorize). If a list is provided one element of the list is taken per track.
- **tracks**: Either the number of tracks or a vector containing the names of the tracks.
- **start_location**: Either one or a list of start locations, as a vector with two numbers.
- **track_id**: The identifier of the track if a single track is requested.

Details

Note that when lists are provided as in input the names of these lists are ignored. Individuals are simulated by order.

If `t` is numeric the movement rate (`sigma`) is assumed to be expressed per unit `t`, if `t` is a timestamp or a date, `sigma` is assumed to be expressed per second.

Value

a move2 object
Examples

```r
mt_sim_brownian_motion() |> plot()
mt_sim_brownian_motion(list(1:10, 1:100)) |> mt_track_lines() |> plot()
mt_sim_brownian_motion(1:200, sigma = .25, letters[1:4],
                        list(c(0, 0), c(10, 0), c(0, 10), c(10, 10))) |> mt_track_lines() |> plot()
```

---

**mt_stack**  
*Combine multiple move2 objects into one*

**Description**

This function does a similar job to `dplyr::bind_rows()`, when columns are missing of any of the objects, they are added.

**Usage**

```r
mt_stack(
  ..., .track_combine = c("check_unique", "merge", "rename"),
  .track_id_repair = c("unique", "universal", "unique_quiet", "universal_quiet")
)
```

**Arguments**

- `...` Either a list of `move2` objects to combine or the objects to combine as separate arguments
- `.track_combine` A character string indicating the way duplicated tracks should be resolved. By default ("check_unique") an error is raised if different objects contain tracks with the same name. With "merge" tracks with the same name can be merged, and with "rename" non unique tracks can be renamed.
- `.track_id_repair` The way in which names should be repaired when renaming is done, see `vctrs::vec_as_names()` for more details on each option

**Details**

An attempt is made to combine objects that have a different `track_id_column` or `time_column`, however this is only done if it can be done without data loss.

When objects are too different (e.g. different projection or different types of timestamps that cannot be combine) and error is raised. When tracks have the same name in different objects to combine this will results in an error.
When merging several tracks, the track attributes of these tracks are also combined. This is done by creating a lists within each column. See examples in \texttt{mt_set_track_id()}. 

**Value**

An object of the class \texttt{move2}

**See Also**

rbind

**Examples**

```r
a <- mt_sim_brownian_motion(1:2, tracks = c("a", "b"))
b <- mt_sim_brownian_motion(1:2, tracks = c("g", "h"))
mt_stack(a, b)

## having different columns does not cause problems
a$extra_data <- 1:nrow(a)
mt_stack(list(a, b))

## Combining different datasets works
fishers <- mt_read(mt_example(), n_max = 100, col_select = c(
  "eobs:used-time-to-get-fix",
  "location-long", "location-lat", "timestamp", "individual-local-identifier"
))

## Objects to stack need to have the same CRS, use either st_set_crs
## or st_transform depending what is appropriate
random_track <- mt_sim_brownian_motion(
  t = as.POSIXct("1970-1-1") + 1:3,
  tracks = factor(letters[1:2])
) |> sf::st_set_crs(4326)
mt_time(random_track) <- "timestamp"
mt_stack(random_track, fishers)

if (requireNamespace("units")) {
  males <- tail(filter_track_data(fishers, grepl("M", \'individual-local-identifier\' ), 5))
  females <- filter_track_data(fishers, grepl("F", \'individual-local-identifier\' )
  females$\'eobs:used-time-to-get-fix\' <- units::set_units(females$\'eobs:used-time-to-get-fix\', "hours"
```
females <- tail(females, 5)
## combining with different units works correctly (units are unified with correct conversion)
mt_stack(males, females)

---

### mt_time

Retrieve/replace timestamps or get the interval duration between locations

**Description**

- `mt_time()` retrieve timestamps
- `mt_time(x)` <- value and `mt_set_time(x, value)` replace timestamps with new values
- `mt_time_lags()` returns time lags, i.e. duration interval between consecutive locations

**Usage**

```
mt_time(x)
mt_time(x) <- value
mt_set_time(x, value)
mt_time_lags(x)
```

**Arguments**

- `x` a `move2` object
- `value` either the new time values or the name of the new time column as a scalar character. When the column is present then that column is used, otherwise the existing time column is renamed

**Details**

Time lags are calculated as the time difference to the next location.

When calculating time lags between locations NA values are used for the transitions between tracks. This is because the interval between the last location of the previous track and first of the next track do not make sense.

**Value**

- `mt_time()` returns a vector of timestamps, depending on the type of data these can be either POSIXct, date or numeric
- `mt_time_lags()` returns a vector of the time lags as numeric or `units` depending on the type of data.
See Also

Other track-measures: \texttt{mt_azimuth()}, \texttt{mt_distance()}

Examples

```r
## in the simulated track, time is numeric, so the timelags are also numeric
x <- mt_sim_brownian_motion(1:3)
x |> mt_time()
x |> mt_time_lags()

## here the simulated track has timestamps, so the timelags have units
x <- mt_sim_brownian_motion(as.POSIXct((1:3) * 60^2, origin = "1970-1-1"), tracks = 1)
x |> mt_time()
x |> mt_time_lags()

## units of the timelags can also be transformed, e.g. from days to hours
tl <- x |> mt_time_lags()
units::set_units(tl, h)
```

---

\texttt{mt_time_column} \hspace{1cm} \textit{Get or set the name of the column containing the track_id and time}

Description

- \texttt{mt_time_column()} returns the name of the column containing the timestamps
- \texttt{mt_track_id_column()} returns the name of the column containing the track ids
- \texttt{mt_set_time_column()} set the column that should be used as time column
- \texttt{mt_set_track_id_column()} set the column that should be used as track id column

Usage

\begin{verbatim}
mt_time_column(x)
mt_track_id_column(x)
mt_set_time_column(x, value)
mt_set_track_id_column(x, value)
\end{verbatim}

Arguments

- \texttt{x} \hspace{1cm} \texttt{a move2 object}
- \texttt{value} \hspace{1cm} \texttt{a character string of the new column name}
Details

The set functions purely update the attribute containing the column name after checking the minimal requirements.

Value

*mt_time_column* and *mt_track_id_column* return character string of the column name

*mt_set_time_column* and *mt_set_track_id_column* return an updated *move2* object

See Also

*mt_time()* to retrieve or change timestamps from each record.

*mt_track_id()* to retrieve or change the track id from each record.

Examples

```r
## getting the column names
mt_sim_brownian_motion() |> mt_time_column()
mt_sim_brownian_motion() |> mt_track_id_column()

## setting 'time' to a new column
x <- mt_sim_brownian_motion()
x$date <- as.Date("2020-1-1") + x$time * 3
x |> mt_time_lags()
x |> mt_set_time_column("date") |> mt_time_lags()
```

### mt_track_data

Setting and retrieving the track data in *move2* objects

Description

- *mt_track_data()* retrieve track attribute table
- *mt_set_track_data()* replace the attribute table

Usage

```r
mt_track_data(x)

mt_set_track_data(x, data)
```

Arguments

- **x** the *move2* object
- **data** the new track data. This *data.frame* must contain the column with the track ids, the column name must be the same as in the *move2* object.
**Value**

*mt_track_data* returns a data frame containing the track attribute data.  
*mt_set_track_data* returns the move2 object with updated track data

**Examples**

```
mt_sim_brownian_motion() |>
  mutate_track_data(sex = c("f", "m")) |>
  mt_track_data()

x <- mt_sim_brownian_motion(1:2, tracks = letters[1:4])
mt_set_track_data(x, data.frame(track = letters[1:4], age = 2:5))
```

---

### mt_track_id

Retrieves the column with track ids or gets the number of tracks

**Description**

- `mt_track_id()` retrieves track ids
- `mt_track_id(x) <- value` and `mt_set_track_id(x, value)` replace track ids with new values
- `mt_n_tracks(x)` returns the number of tracks

**Usage**

```
mt_track_id(x)

mt_track_id(x) <- value

mt_set_track_id(x, value)

mt_n_tracks(x)
```

**Arguments**

- `x` a move2 object
- `value` either the new track id values or the name of the new track id column as a scalar character. When the column is present then that column is used, otherwise the existing track id column is renamed

**Details**

When changing the track ids with new values that results in the combination of several tracks, the track attributes of these tracks are also combined. This is done by creating a lists within each column. See examples.
mt_track_lines

Description

Converts each track into one line

Usage

mt_track_lines(x, ...)

Arguments

x A move object

... Arguments passed on to the summarise function

Details

Note that all empty points are removed before summarizing. Arguments passed with ... thus only summarize for the non empty locations.

Value

A sf::sf object with a line representing the track as geometry for each track.
Examples

```r
mt_sim_brownian_motion() |> 
  mt_track_lines(
    n = dplyr::n(),
    minTime = min(time),
    maxTime = max(time)
  )

## empty points are not counted in summary statistic
dx <- mt_sim_brownian_motion(1:3)
x$geometry[[2]] <- sf::st_point()
x |> mt_track_lines(
  n = dplyr::n()
)
## plot of the tracks as a line
mt_sim_brownian_motion(
  tracks = letters[1:2],
  start_location = list(c(0, 0), c(10, 0))
) |> 
  mt_track_lines() |> 
  plot()
```

---

**to_move**

*Convert a move2 object to a move object*

**Description**

Convert a move2 object to a move object

**Usage**

```r
to_move(x)
```

**Arguments**

- `x` a move2 object.

**Value**

an object of the class Move/MoveStack

to_move converts back to a objects from the move package. When multiple individuals are provided a `MoveStack-class` is created otherwise a `Move-class` object.

**See Also**

Other move2-convert: `mt_as_move2()`
Examples

```r
if (requireNamespace("move")) {
  data(leroy, package = "move")
  leroy_move2 <- mt_as_move2(leroy)
  to_move(leroy_move2)
}
```
Index

* filter
  mt_filter_movebank_visible, 16
  mt_filter_per_interval, 17
  mt_filter_unique, 18
* move2-convert
  mt_as_move2, 11
  to_move, 33
* movebank-download
  movebank_download_study, 4
  movebank_handle, 9
  movebank_store_credentials, 10
* track-measures
  mt_azimuth, 13
  mt_distance, 14
  mt_time, 28

askpass, 9, 10
assert_that, 22
connections, 23
curl, 9
dplyr::bind_rows(), 26
filter, 2
filter_track_data, 2
floor_date, 18, 20
group_by, 2, 3
group_by_track_data
  (filter_track_data), 2
handle, 9
helpers, 13
lwgeom::st_geod_azimuth(), 14
movebank_download_deployment
  (movebank_download_study), 4
movebank_download_study, 4, 9, 10
movebank_download_study_info
  (movebank_download_study), 4
movebank_get_study_id
  (movebank_download_study), 4
movebank_get_vocabulary, 7
movebank_handle, 6, 9, 10
movebank_remove_credentials
  (movebank_store_credentials), 10
movebank_retrieve
  (movebank_download_study), 4
movebank_store_credentials, 6, 9, 10
mt_as_event_attribute
  (mt_as_track_attribute), 12
mt_as_move2, 11, 33
mt_as_track_attribute, 12
mt_azimuth, 13, 15, 29
mt_distance, 14, 14, 29
mt_example, 15
mt_example(), 23
mt_filter_movebank_visible, 16, 18, 19
mt_filter_per_interval, 17, 17, 19
mt_filter_unique, 17, 18, 18
mt_has_no_empty_points
  (mt_is_track_id_cleaved), 21
mt_has_unique_location_time_records
  (mt_is_track_id_cleaved), 21
mt_interpolate, 20
mt_is_move2 (mt_is_track_id_cleaved), 21
mt_is_time_ordered
  (mt_is_track_id_cleaved), 21
mt_is_time_ordered_non_empty_points
  (mt_is_track_id_cleaved), 21
mt_is_track_id_cleaved, 21
mt_movebank_visible
  (mt_filter_movebank_visible), 16
mt_n_tracks (mt_track_id), 31
mt_per_interval
(mt_filter_per_interval), 17

mt_read, 23
mt_read(), 16
mt_segments, 24
mt_set_time (mt_time), 28
mt_set_time_column (mt_time_column), 29
mt_set_track_data (mt_track_data), 30
mt_set_track_data(), 13
mt_set_track_id (mt_track_id), 31
mt_set_track_id(), 27
mt_set_track_id_column
  (mt_time_column), 29
mt_sim_brownian_motion, 25
mt_speed (mt_distance), 14
mt_stack, 26
mt_time, 14, 15, 28
mt_time(), 30
mt_time<- (mt_time), 28
mt_time_column, 29
mt_time_lags (mt_time), 28
mt_track_data, 30
mt_track_data(), 13
mt_track_id, 31
mt_track_id(), 30
mt_track_id<- (mt_track_id), 31
mt_track_id_column (mt_time_column), 29
mt_track_lines, 32
mt_turnangle (mt_azimuth), 13
mt_unique (mt_filter_unique), 18
mutate, 2
mutate_track_data (filter_track_data), 2

on_failure, 22

s2_interpolate_normalized, 20
select, 2
select_track_data (filter_track_data), 2
sf::sf, 32
sf::st_distance, 15
st_as_sf, 11
st_crs, 11
st_line_sample, 20
summarise, 32

to_move, 12, 33

units, 28

vctrs::vec_as_names(), 26