Package ‘himach’

November 21, 2021

**Type** Package
**Title** High Mach Finds Routes for Supersonic Aircraft
**Version** 0.2.2
**Description** For supersonic aircraft, flying subsonic over land, High Mach finds the best route between airports. Allows for coastal buffer and potentially closed regions. Uses a minimal model of aircraft performance: the focus is on time saved versus subsonic flight, rather than on vertical flight profile. For modelling and forecasting, not for planning your flight!
**License** MIT + file LICENSE
**URL** [https://github.com/david6marsh/himach](https://github.com/david6marsh/himach)
**BugReports** [https://github.com/david6marsh/himach/issues](https://github.com/david6marsh/himach/issues)
**Depends** R (>= 3.5.0)
**Imports** cppRouting, data.table, dplyr (>= 1.0.0), geosphere, ggplot2, lwgeom, purrr, s2, sf (>= 1.0), tidyr
**Suggests** airportr, cowplot, knitr, methods, progress, rmarkdown, naturalearthdata, scales, stringr, testthat, units, utils, viridis, covr
**VignetteBuilder** knitr
**Encoding** UTF-8
**LazyData** true
**Language** en-GB
**RoxygenNote** 7.1.2
**NeedsCompilation** no
**Author** David Marsh [aut, cre], EUROCONTROL [fnd, cph]
**Maintainer** David Marsh <david.marsh@eurocontrol.int>
**Repository** CRAN
**Date/Publication** 2021-11-21 16:40:02 UTC
R topics documented:

crs_120E ............................................................... 2

Description
Coordinate reference system (CRS) for plotting and analysing maps. Centred on East Asia (120E).

Usage

```r
crs_120E
```

Format
CRS

Details
```
"+proj=robin +lon_0=120 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
```
**crs_Atlantic**

**See Also**

crs_Atlantic, crs_Pacific, crs_N, crs_S

---

**crs_Atlantic** \(\text{Atlantic-centred coordinate reference system}\)

**Description**

Coordinate reference system (CRS) for plotting and analysing maps. Atlantic-centred. Works for most analysis, but not recommended for N-region (e.g., New Zealand and Fiji), instead use crs_Pacific.

**Usage**

```r
crs_Atlantic
```

**Format**

CRS

**Details**

```
crs_Atlantic is "+proj=robin +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs"
```

**See Also**

crs_Pacific, crs_120E, crs_N, crs_S

---

**crs_longlat** \(\text{Lat-long coordinate reference system}\)

**Description**

Coordinate reference system (CRS) for creating maps from longitude-latitude coordinates. Used in analysis, but not recommended for plots.

**Usage**

```r
crs_longlat
```

**Format**

CRS
Details

crs_longlat is EPSG4326

See Also

crs_Atlantic, crs_Pacific, crs_S, crs_N

crs_N

Arctic-centred coordinate reference system

Description

Coordinate reference system (CRS) for plotting and analysing maps. WGS 84 / Arctic Polar Stereographic. Used in analysis, but not recommended for plots.

Usage

crs_N

Format

CRS

Details

crs_N is EPSG3995

See Also

crs_Atlantic, crs_Pacific, crs_120E, crs_longlat, crs_S

crs_Pacific

Pacific-centred coordinate reference system

Description

Coordinate reference system (CRS) for plotting and analysing maps. Pacific-centred.

Usage

crs_Pacific

Format

CRS
**crs_S**

**Details**

```
+proj=robin +lon_0=180 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs
```

**See Also**

`crs_Atlantic, crs_120E, crs_N, crs_S`

---

**crs_S**

 Antarctic-centred coordinate reference system

**Description**

Coordinate reference system (CRS) for plotting and analysing maps. WGS 84 / Antarctic Polar Stereographic. Used in analysis, but not recommended for plots.

**Usage**

`crs_S`

**Format**

CRS

**Details**

`crs_N` is EPSG 3031

**See Also**

`crs_Atlantic, crs_Pacific, crs_120E, crs_longlat, crs_N`

---

**find_leg**

Find best non-stop route between 2 airports

**Description**

`find_leg` finds the quickest non-stop route for ac between two airports ap2.
Usage

find_leg(
  ac,
  ap2,
  route_grid,
  fat_map,
  ap_loc,
  avoid = NA,
  enforce_range = TRUE,
  best_by_time = TRUE,
  grace_km = NA,
  shortcuts = TRUE,
  ad_dist_m = 100 * 1000,
  ad_nearest = 12,
  max_leg_circuity = 1.4,
  ...
)

Arguments

ac, ap2, route_grid, fat_map, ap_loc, avoid
See find_route

enforce_range If TRUE (default) then leg is constrained to aircraft range, otherwise routes of excess range can be found.

best_by_time If TRUE (default) then the quickest route is found, else the shortest distance.

grace_km Default NA. Otherwise, if great circle distance is within 3pct of aircraft range, then add grace_km km to the range.

shortcuts If TRUE (default) then path will be checked for great circle shortcuts.

ad_dist_m The length of arrival/departure links, in m. (Default 100,000=100km)

ad_nearest The number of arrival/departure links to create (Default 12)

max_leg_circuity The maximum detour over great circle distance that can be flown to find a quick over-sea route. Default 1.4.

... Other parameters, passed to make_route_envelope

Details

This function finds the quickest non-stop route between two airports. A 'route' is made up of one or two 'legs' (airport to airport without intermediate stop). find_route makes one or more calls to find_leg as required.

It assumes that the routing grid, route_grid, has already been classified as land or sea using the map fat_map. The map is further used when converting the grid-based route to one of great-circle segments.

In fact find_leg finds up to 4 versions of the path:

1. A great circle, direct between the airports
2. A grid path, consisting of segments of the routing grid, plus departure and arrival routes from the airports
3. A simplification of the grid path to great circle segments
4. shortcuts defaults to TRUE. Without this, you see near-raw Dijkstra results, which are _not_ shortest great circle.

Legs are automatically saved in route_cache and retrieved from here if available rather than re-calculated. See vignette on caching for cache management.

**Value**

Dataframe with details of the leg

**Examples**

```r
# need to load some of the built-in data
aircraft <- make_aircraft(warn = FALSE)
airports <- make_airports(crs = crs_Pacific)
# get test datasets
NZ_buffer30 <- hm_get_test("buffer")
NZ_grid <- hm_get_test("grid")

options("quiet" = 4) #for heavy reporting
# from Auckland to Christchurch
ap2 <- make_AP2("NZAA","NZCH",airports)
routes <- find_leg(aircraft[4,],
ap2,
  fat_map = NZ_buffer30,
  route_grid = NZ_grid,
ap_loc = airports)
```

**Description**

find_route finds the quickest route between two airports, refuelling if necessary

**Usage**

```r
find_route(ac,
ap2,
fat_map,
avoid = NA,
route_grid,
cf_subsonic = NA,
```


```
refuel = NA,
refuel_h = 1,
refuel_only_if = TRUE,
refuel_topN = 1,
max_circuity = 2,
ap_loc,
margin_km = 200,
``` ...

Arguments

- **ac**: One aircraft, as from `make_aircraft`
- **ap2**: One airport pair, as from `make_AP2`
- **fat_map**: sf::MULTIPOLYGON map of land, including buffer
- **avoid**: sf::MULTIPOLYGON map of areas not to fly over
- **route_grid**: GridLat routing grid as from `make_route_grid`
- **cf_subsonic**: Further aircraft to use as comparator, default NA. (use is not recommended)
- **refuel**: Airports available for refuelling, dataframe with APICAO, long, lat
- **refuel_h**: Duration of refuelling stop, in hours
- **refuel_only_if**: If TRUE (default) only test refuel options if necessary because the great circle distance is too far for the aircraft range
- **refuel_topN**: Return the best N (default 1) refuelling options
- **max_circuity**: Threshold for excluding refuelling stops (default 2.0)
- **ap_loc**: Airport locations as from `make_airports`
- **margin_km**: Great circle distance between airports must be less than aircraft range minus this operating margin (default 200km), to give a margin for arrival and departure.
- **...**: Other parameters, passed to `find_leg` and thence to to `make_route_envelope`.

Details

This function finds the quickest route between two airports. A 'route' is made up of one or two 'legs' (airport to airport without intermediate stop). `find_route` makes one or more calls to `find_leg` as required.

It assumes that the routing grid, `route_grid`, has already been classified as land or sea using the map `fat_map`. The map is further used when converting the grid-based route to one of great circles segments.

Value

Dataframe with details of the route
Refuelling

If either necessary, because the great circle distance is greater than the aircraft range, or because `refuel_only_if` is FALSE, `find_route` searches through a list of refuelling airports and chooses the quickest one (or `refuel_topN`).

Circuitous refuelling is avoided, tested against total distance < `max_circuity * giant circle distance`. This is separate to the limits placed on circuity of individual legs in `find_leg`.

If no refuel option is found, a message is displayed. The route with ‘NA’ for ‘time_h’ is returned.

Each refueling stop costs `refuel_h` in addition to the time to descend to the airport and then to climb out again.

Examples

```r
# need to load some of the built-in data
aircraft <- make_aircraft(warn = FALSE)
airports <- make_airports(crs = crs_Pacific)
# get test datasets
NZ_buffer30 <- hm_get_test("buffer")
NZ_grid <- hm_get_test("grid")

options("quiet" = 4) # for heavy reporting
# from Auckland to Christchurch
ap2 <- make_AP2("NZAA","NZCH",airports)
routes <- find_route(aircraft[4,],
ap2,
  fat_map = NZ_buffer30,
  route_grid = NZ_grid,
ap_loc = airports)
```

---

**find_routes**

*Find best routes between airport-pair & aircraft combinations*

**Description**

`find_routes` combines an aircraft and airport-pair list and finds the best routes between them, refuelling if necessary

**Usage**

```r
find_routes(ac_ids, ap2_ids, aircraft, airports, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac_ids</td>
<td>A vector of aircraft IDs, as in column 'id' from <code>make_aircraft</code></td>
</tr>
<tr>
<td>ap2_ids</td>
<td>A 2-column matrix or dataframe of airport pair text IDs</td>
</tr>
<tr>
<td>aircraft</td>
<td>Specification of the aircraft, see <code>make_aircraft</code></td>
</tr>
<tr>
<td>airports</td>
<td>Airport locations as from <code>make_airports</code></td>
</tr>
<tr>
<td>...</td>
<td>Other parameters, passed to <code>find_route</code>.</td>
</tr>
</tbody>
</table>
Details

This function finds is a wrapper for the single-case function `find_route`. It takes (text) lists of aircraft and airport codes, combines them, then finds routes for all of these. A 'route' is made up of one or two 'legs' (airport to airport without intermediate stop).

For more details see `find_route`

Value

Dataframe with details of the routes

Examples

```r
# need to load some of the built-in data
aircraft <- make_aircraft(warn = FALSE)
airports <- make_airports(crs = crs_Pacific)
# get test datasets
NZ_buffer30 <- hm_get_test("buffer")
NZ_grid <- hm_get_test("grid")

options("quiet" = 4) #for heavy reporting
# from Auckland to Christchurch
ap2 <- make_AP2("NZAA","NZCH",airports)
routes <- find_route(aircraft[4,],
ap2,
fat_map = NZ_buffer30,
route_grid = NZ_grid,
ap_loc = airports)
```

GridLat-class

A grid and lattice combination

Description

A GridLat keeps together a grid of points and a lattice of links between those points.

It has 3 components:

* A character name, which isn’t used much in anger but might help you remember what’s gone into it. * A dataframe containing the points of the lattice (the vertices), which each have an ID, a longitude and latitude. * A dataframe containing the edges of the lattice, joining the points.
himach

himach: A package for computing supersonic aircraft routes

Description

The himach (high Mach) package finds the quickest route between airports, for supersonic aircraft that fly subsonic over land.

Details

It allows for a coastal buffer and potentially closed regions of airspace. It uses a minimal model of aircraft performance: focus is on time saved versus subsonic flight, rather than a detailed vertical flight profile. Subsonic aircraft can be routed too, for comparison.

Mach cut-off flying is also possible by creating aircraft with supersonic cruise speed over land.

The package essentially combines the functionality of cppRouting for finding routes and sf for handling map 'simple features', with a lot of help from the tidyverse, of course. In the latest version it uses direct spherical geometry, either directly through package s2 or indirectly through support to s2 from package sf.

hm_clean_cache

Clean the route and SID-STAR cache.

Description

Empties the cache.

Usage

hm_clean_cache(cache = c("route", "star"))

Arguments

cache Which caches to clear. Default is both c("route","star").

Value

TRUE silently

See Also

For more details see the cache section in the vignette: vignette("Supersonic_Routes_in_depth",package = "himach"). or Vignette on caching
\textbf{hm_get_test}

\textbf{Examples}
\begin{verbatim}
hm_clean_cache("route")
hm_clean_cache()
\end{verbatim}

\textbf{Description}
Access 5 datasets that are used in vignettes and in testing.

\textbf{Usage}
\begin{verbatim}
hm_get_test(item = c("coast", "buffer", "nofly", "grid", "route"))
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
  \item \textbf{item} Any one of "coast", "buffer", "nofly", "grid", "route". See details.
\end{itemize}

\textbf{Details}
\begin{itemize}
  \item "coast" A dataset containing sf::MULTIPOLYGONS for New Zealand. Simplified version of Stats NZ data, at 1km resolution.
  \item "buffer" As "coast" but with an added 30km buffer to keep supersonic flight away from the coast.
  \item "nofly" As "buffer", but limited to Buller district with a 40km buffer. To test additional no-fly zones.
  \item "grid" Latitude-longitude-based routing grid around New Zealand at 30km target distance, as generated by \texttt{make_route_grid}, so format is GridLat
  \item "route" Some very unlikely supersonic routes around New Zealand using the test aircraft that was given a very short range and slow subsonic cruise to get the example to 'work'. Includes one refuelling stop (!) in Wellington. [Not for operational use!] Returns a dataframe.
\end{itemize}

This is not the normal way to access package test data. But the usual, direct, way fails on some machines that have some older software (a known feature of the ‘sf’ package). This is a least-ugly workaround.

\textbf{Value}
See list above

\textbf{Source}
hm_load_cache

Description

This silently overwrites any existing values in the cache.

Usage

hm_load_cache(file)

Arguments

file Including the path.

Value

Invisible true

See Also

For more details see the cache section in the vignette: vignette("Supersonic_Routes_in_depth",package = "himach"). or Vignette on caching

Examples

# not run
# hm_load_cache(file="") #load from this file

hm_save_cache

Save route and SID/STAR cache to file

Description

Filename is "route_star_cache_id_XXX.rda" where "id" is the id parameter and XXX is made up from the name of the grid (which identifies the map used) and the 'aircraftSet' attribute of the aircraft dataset (which identifies the source). This is because the cache should be for a unique combination of these (and you must have these available, because they were needed to generate the routes).
Usage

hm_save_cache(id, grid, aircraft, path = "data/")

Arguments

id Identifying text, see above. Recommended to use a version number or date.
grid Your route grid dataset. The grid@name will be added to the filename.
aircraft Your aircraft dataset. The attr(aircraft,"aircraftSet") will be added to the filename.
path By default "data/", where the file will be saved.

Value

Invisible true

See Also

For more details see the cache section in the vignette: vignette("Supersonic_Routes_in_depth",package = "himach"). or Vignette on caching

Examples

# not run
# hm_save_cache("v2", grid, ac) #save here

mach_kph

| mach_kph | Speed of sound, for Mach to km conversion |

Description

1 Mach is approximately 1062kph in standard met conditions at the altitude for supersonic flight (approx 50,000 feet).

Usage

mach_kph

Format

double
**make_aircraft**

*Make aircraft data from minimum dataset*

**Description**

`make_aircraft` ensures a minimum set of variables describing aircraft

**Usage**

```r
make_aircraft(ac = NA, sound_kph = himach::mach_kph, warn = TRUE)
```

**Arguments**

- **ac**
  - Dataframe containing the minimum fields, or NA (default)
- **sound_kph**
  - Speed of sound used to convert from Mach to kph, default `mach_kph=1062 at a suitable altitude`
- **warn**
  - Warn if no ac supplied, so default set is used. Default TRUE.

**Details**

This function provides a test set of aircraft if necessary and adds variables to a minimal set of data to give all the information that will be needed.

This minimal set needs to have the following fields:

- **id**, **type**: a very short, and longer text identifier for this aircraft
- **over_sea_M**, **over_land_M**: the eponymous two speeds, given as a Mach number
- **accel_Mpm**: acceleration in Mach per minute between these two
- **arrdep_kph**: the speed on arrival and departure from airports, given in km per hour
- **range_km**: range in km

An attribute is set to help keep track of where the aircraft data came from (and whether a new cache is needed). If the `aircraftSet` attribute of the ac parameter is not set, the set is treated as 'disposable'.

For more details see the help vignette: vignette("SupersonicRouting",package = "himach")

**Value**

Dataframe with at least 11 variables describing the performance of one or more aircraft
Examples

```r
# do minimal version (we know it will use the default so turn off warning)
ac <- make_aircraft(warn = FALSE)

# on-the-fly example
ac <- data.frame(id = "test", type = "test aircraft",
                 over_sea_M = 2.0, over_land_M = 0.9, accel_Mpm = 0.2,
                 arrdep_kph = 300, range_km = 6000, stringsAsFactors=FALSE)
ac <- make_aircraft(ac, warn = FALSE)

## Not run:
# example for your own data
aircraft <- utils::read.csv("data/aircraft.csv", stringsAsFactors = FALSE)
aircraft <- make_aircraft(aircraft)
# strongly recommended to record the file name for later reference
attr(aircraft, "aircraftSet") <- "aircraft.csv"

## End(Not run)
```

make_airports

Make or load airport data

Description

make_airports ensures a minimum set of variables describing airports

Usage

```r
make_airports(ap = NA, crs = himach::crs_longlat, warn = TRUE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ap</td>
<td>Dataframe containing the minimum fields, or NA (default)</td>
</tr>
<tr>
<td>crs</td>
<td>Coordinate reference system for the coded lat-longs. Default 4326.</td>
</tr>
<tr>
<td>warn</td>
<td>warn if default set is used (default = TRUE)</td>
</tr>
</tbody>
</table>

Details

This function provides a test set of airports if necessary from airportr::airports and geocodes the lat-long of this or the dataset provide as ap.

This minimal set needs to have the following fields:

- **APICAO**: the 4-letter ICAO code for the airport (though there is no validity check applied, so 'TEST', or 'ZZZZ' could be used, for example)
- **lat, long**: latitude and longitude in decimal degrees
make_AP2

Value

Dataframe with, in addition, a geocoded lat-long.

Examples

# do minimal version
airports <- make_airports()

# on-the-fly example
airports <- data.frame(APICAO = "TEST", lat = 10, long = 10, stringsAsFactors = FALSE)
airports <- make_airports(airports)

## Not run:
# example for your own data
airports <- utils::read.csv("data/airports.csv", stringsAsFactors = FALSE)
airports <- make_airports(airports)

## End(Not run)

make_AP2  Make airport-pair dataset

Description

make_AP2 creates an airport-pair set from two sets of airports

Usage

make_AP2(adep, ades, ap = make_airports())

Arguments

adep, ades  Identical-length lists of airport codes
ap  List of locations of airports, defaults to the output of make_airports.

Details

This function takes two lists of airports (of the same length), specified as 4-letter codes and combines them, adding the fields:

- from_long, from_lat, to_long, to_lat: the airport lat-longs with adep first
- AP2: a name for the route in a specific order
- gcdist_km: the great circle distance in km

In AP2 European airports (crudely, from starting letter = 'E' or 'L') are listed first, otherwise in alphabetical order. If unidirectional is TRUE, then "->" is the separator, otherwise "<>". (Unidirectional not currently supported)

For more details see the introductory vignette.
Value

Dataframe with additional variables as described above.

Examples

```r
airports <- make_airports() # get a default set of lat-longs
ap2 <- make_AP2("NZAA","NZCH", airports)
```

Description

`make_route_envelope` finds the range envelope for a given route

Usage

```r
make_route_envelope(ac, ap2, envelope_points = 200, fuzz = 0.005)
```

Arguments

- `ac, ap2`: See `find_route`
- `envelope_points`: How many points are used to define the ellipse? Default 200.
- `fuzz`: Add a little margin to the range, to allow the longest range to be flown, rather than be cut off at the boundary. (Default 0.005)

Details

The 'route envelope' is the region within which a route from A to B must remain. This is an ellipse. It differs from the pure 'range envelope' which is the points which an aircraft can reach from a given airport.

Value

sf POLYGON with ad hoc coordinate reference system.

Examples

```r
# Need aircraft and airport datasets
ac <- make_aircraft(warn = FALSE)
ap <- make_airports()
z <- make_route_envelope(ac[1,], make_AP2("EGLL","KJFK", ap))
```
**make_route_grid**  
*Make lat-long grid for route finding*

### Description

make_route_grid creates, and optionally classifies, a lat-long route grid

### Usage

```r
make_route_grid(
  fat_map,
  name,
  target_km = 800,
  lat_min = -60,
  lat_max = 86,
  long_min = -180,
  long_max = 179.95,
  classify = FALSE
)
```

### Arguments

- **fat_map**: MULTIPOLYGON map defining land regions
- **name**: String assigned to the name slot of the result
- **target_km**: Target length. Default 800km only to avoid accidentally starting heavy compute. 30-50km would be more useful.
- **lat_min, lat_max**: Latitude extent of grid
- **long_min, long_max**: Longitude extend of grid. Two allow small grids crossing the 180 boundary, the function accepts values outside [-180,180), then rounds to within this range.
- **classify**: Whether to classify each link. Defaults to FALSE only to avoid accidentally starting heavy compute.

### Details

This function creates a GridLat object that contains a set of point on a lat long grid (ie all the points are on lines of latitude). It also joins these points into a lattice. Optionally, but required later, it classifies each link as land, sea, or transition, with reference to a given map (typically including a coastal buffer).

The definitions are

- **land**: both ends of the link are on land
- **sea**: both ends are on sea, and the link does not intersect the land
- **transition**: otherwise
The length of the links will be around target_km or 50pct longer for the diagonal links. For more details see the help vignette: vignette("Supersonic Routing", package = "himach")

Value

gridLat object containing points and lattice.

Examples

```r
NZ_buffer <- hm_get_test("buffer")
system.time(
  p_grid <- make_route_grid(NZ_buffer, "NZ lat-long at 300km",
  target_km = 300, classify = TRUE,
  lat_min = -49, lat_max = -32,
  long_min = 162, long_max = 182)
)
```

map_routes

Map a set of routes

Description

map_routes plots routes, with many options

Usage

```r
map_routes(
  thin_map,
  routes = NA,
  crs = himach::crs_Atlantic,
  show_route = "time",
  fat_map = NA,
  avoid_map = NA,
  ap_loc = NA,
  ap_col = "darkblue",
  ap_size = 0.4,
  crow = FALSE,
  crow_col = "grey70",
  crow_size = 0.2,
  route_envelope = FALSE,
  bound = TRUE,
  bound_margin_km = 200,
  simplify_km = 8,
  land_f = "grey90",
  buffer_f = "grey60",
  avoid_f = "grey80",
  l_alpha = 0.8,
)```
map_routes

l_size = 0.5,
e_alpha = 0.4,
e_size = 0.6,
e_col = "grey70",
refuel_airports = ap_loc,
rap_col = "red",
rap_size = 0.4,
scale_direction = -1,
title = "",
subtitle = "",
warn = FALSE
)

Arguments

thin_map The minimum is a MULTIPOLYGON map, 'thin' in that it is without buffer, so a normal coastline map.
routes as generated by find_route
crs Coordinate reference system, default crs_Atlantic.
show_route one of "speed", "aircraft", "time", "circuity" to indicate what goes in the legend.
fat_map optional coast + buffer map, default NA.
avoid_map optional map of no-fly zones, default NA.
ap_loc Show used origin and destination airports if this is a set of airports from make_airports, or not if NA (default). This dataset can be all airports, and is filtered to those used by routes.
ap_col, ap_size Colour and size of used airport markers (dark blue, 0.4)
crow, crow_col, crow_size If TRUE, show the 'crow-flies' direct great circle, in colour crow_col and thickness crow_size. Default FALSE, "grey70", 0.2
routeEnvelope show the route envelope (default FALSE).
bound, bound_margin_km If bound=TRUE (default) crop to bounding box of the routes, with additional bound_margin_km in km (default 200)
simplify_km Simplify the two maps to this scale before plotting (default 10).
land_f, buffer_f, avoid_f fill colours for thin, fat and no-fly maps, default grey 90, 70 and 80, respectively
l_alpha, l_size line (route) settings for alpha (transparency) and width, defaults 0.6 and 0.4.
e_col, e_alpha, e_size colour, alpha and width for the range envelope. Default "grey70", 0.4, 0.6
refuel_airports Show the used refuel airports using these locations, or nothing if NA. (Defaults to same as ap_loc.)
**st_window**

rap_col, rap_size

Colour and size of refuel airport markers (red, 0.4)

scale_direction

Passed to scale_colour_viridis, either -1 (default) or or 1.

title, subtitle

Passed to ggplot.

warn

if TRUE show some warnings (when defaults loaded) (default FALSE)

**Details**

This function plots the routes, with options for additional layers. Multiple routes are expected, and they can be coloured by time advantage, by speed along each segment, or by aircraft type.

The option show_route "time" requires 'advantage_h' to have been added to the routes set, from the route summary. If it hasn't then this is done in a local version, then discarded. Running summarise_routes to do this requires an airport dataset; if is.na(ap_loc) then this is not available, so a default set is used. You can turn on warn to see if this is happening, but by default it is silent.

The time to compute the map may not be very different with simplify_km varying between 2km and 20km, but the time to plot on the screen, or ggsave to a file, is longer than the compute time. It is this latter time that’s reduced by simplifying the maps. For single, or short routes, you can probably see the difference between 2km and 10km, so it’s your choice to prefer speed or beauty.

**Value**

A ggplot.

**Examples**

#see introductory vignette

---

**st_window**

*Version of st_transform with view window to avoid dateline*

**Description**

st_window does a st_transform but first cuts the data to an appropriate view window and so avoids problems with objects wrapping around the back of the globe

**Usage**

st_window(m, crs = himach::crs_Atlantic, longit_margin = 0.1)

**Arguments**

m

A map dataframe, ie of class sf and data.frame, or an sfc_MULTIPOLYGON

crs

Destination coordinate reference system, as in st_transform

longit_margin

Amount trimmed off the 'far side' of the projection in degrees.
**Details**

`st_wrap_dateline` _should_ handle the break in a map projections but uses ‘GDAL’ for this. Given persistent issues in installing GDAL, `st_window` achieves the same using s2 instead.

It works for any 'simple' projection, in the sense of one that has a dateline that is a single line of longitude: ie the proj4string contains either "longitude_of_center", so the dateline is that +180; or not, in which case it assumes the "longitude_of_center" is 0.

**Value**

sf dataframe, same as the parameter m

**Examples**

```r
world <- sf::st_as_sf(rnaturalearthdata::coastline110)
w_pacific <- st_window(world, crs_Pacific)
ggplot2::ggplot(w_pacific) + ggpplot2::geom_sf()

# bad - not run - dateline problem example
# ggplot2::ggplot(st_transform(world, crs_Pacific)) +
#  ggpplot2::geom_sf()
```

---

**summarise_routes**

*Summarise a set of routes*

**Description**

Reduce a set of routes to a one-line per route summary

**Usage**

`summarise_routes(routes, ap_loc, arrdep_h = 0.5)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>routes</td>
<td>Each segment in each route, as produced by <code>find_route</code> or <code>find_leg</code></td>
</tr>
<tr>
<td>ap_loc</td>
<td>List of airport locations, output of <code>make_airports</code></td>
</tr>
<tr>
<td>arrdep_h</td>
<td>Total time for the M084 comparator aircraft to arrive &amp; depart in hours. Default 0.5.</td>
</tr>
</tbody>
</table>

**Details**

This function takes the output of `find_route` and summarises to one line per (full) route.

With refuelling, there can be multiple 'full routes' for each 'route'. The best column indicates the best route for each routeID.

The results are rounded to a reasonable number of significant figures. After all this is just an approximate model. The `arrdep_h` has been checked against actual and is reasonable (observed range roughly 0.3-0.5).
summarise_routes

Value

Dataframe with summary of the route, sorted in ascending order of advantage_h so that the best route are plotted on top. The fields are:

- **timestamp**: when the leg was originally generated (it may have been cached)
- **fullRouteID**: including the refuel stop if any
- **routeID**: origin and destination airport, in make_AP2 order
- **refuel_ap**: code for the refuelling airport, or NA
- **acID, acType**: aircraft identifiers taken from the aircraft set
- **M084_h**: flight time for a Mach 0.84 comparator aircraft (including 2*arrdep_h)
- **gcdist_km**: great circle distance between the origin and destination airports
- **sea_time_frac**: Fraction of time_h time spent over sea, hence at supersonic speed, or accelerating to, or decelerating from supersonic speed
- **sea_dist_frac**: as sea_time_frac, but fraction of dist_km
- **dist_km**: total length of the route, in km
- **time_h**: total time, in hours
- **n_phases**: number of distinct phases: arr/dep, transition, land, sea, refuel.
- **advantage_h**: M084_h - time_h
- **circuity**: the route distance extension (1 = perfect) dist_km / gcdist_km
- **best**: for each routeID, the fullrouteID with maximum advantage_h

Examples

```r
# here we use a built-in set of routes
# see vignette for more details of how to obtain it
airports <- make_airports(crs = crs_Pacific)
NZ_routes <- hm_get_test("route")
sumy <- summarise_routes(NZ_routes, airports)
```
Index

* datasets
  crs_120E, 2
  crs_Atlantic, 3
  crs_longlat, 3
  crs_N, 4
  crs_Pacific, 4
  crs_S, 5
  mach_kph, 14

  crs_120E, 2, 3–5
  crs_Atlantic, 3, 3, 4, 5
  crs_longlat, 3, 4, 5
  crs_N, 3, 4, 4, 5
  crs_Pacific, 3, 4, 4, 5
  crs_S, 3–5, 5

  find_leg, 5, 8, 9, 23
  find_route, 6, 7, 9, 10, 18, 21, 23
  find_routes, 9

  GridLat, 12, 19
  GridLat-class, 10

  himach, 11
  hm_clean_cache, 11
  hm_get_test, 12
  hm_load_cache, 13
  hm_save_cache, 13

  mach_kph, 14
  make_aircraft, 8, 9, 15
  make_airports, 8, 9, 16, 17, 21, 23
  make_AP2, 8, 17, 24
  make_route_envelope, 6, 8, 18
  make_route_grid, 8, 12, 19
  map_routes, 20

  st_window, 22
  st_wrap_dateline, 23
  summarise_routes, 23