

# Package ‘resourcecode’

February 6, 2026

**Title** Access to the 'RESOURCECODE' Hindcast Database

**Version** 0.5.3

**Description** Utility functions to download data from the 'RESOURCECODE' hindcast database of sea-states, time series of sea-state parameters and time series of 1D and 2D wave spectra. See <https://resourcecode.ifremer.fr> for more details about the available data. Also provides facilities to plot and analyse downloaded data, such as computing the sea-state parameters from both the 1D and 2D surface elevation variance spectral density.

**License** GPL (>= 3)

**URL** <https://github.com/Resourcecode-project/r-resourcecode>,  
<https://resourcecode-project.github.io/r-resourcecode/>,  
<https://resourcecode-project.r-universe.dev/resourcecode>

**BugReports** <https://github.com/Resourcecode-project/r-resourcecode/issues>

**Depends** R (>= 4.1)

**Imports** abind, geosphere, ggplot2, grid, gridtext, httr2, lubridate, ncd4, patchwork, pracma, Rcpp, resourcecodedata, rlang, sf, stats, tibble, tidyR

**Suggests** knitr, mockery, rmarkdown, testthat, vcr

**LinkingTo** Rcpp, RcppArmadillo

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/testthat/parallel** true

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**RoxygenNote** 7.3.3

**NeedsCompilation** yes

**Author** Nicolas Raillard [aut, cre] (ORCID:  
<https://orcid.org/0000-0003-3385-5104>)

**Maintainer** Nicolas Raillard <nicolas.raillard@ifremer.fr>  
**Repository** CRAN  
**Date/Publication** 2026-02-06 14:00:02 UTC

Contents

closest_point_field . . . . .	2
closest_point_spec . . . . .	3
compute_orbital_speeds . . . . .	4
compute_sea_state_1d_spectrum . . . . .	5
compute_sea_state_2d_spectrum . . . . .	6
convert_spectrum_2d1d . . . . .	7
cut_directions . . . . .	8
cut_seasons . . . . .	9
dispersion . . . . .	10
fastTrapz . . . . .	10
fractional_day_of_year . . . . .	11
get_1d_spectrum . . . . .	12
get_2d_spectrum . . . . .	13
get_parameters . . . . .	15
jonswap . . . . .	15
mean_direction . . . . .	16
metconv2zmcomp . . . . .	17
plot_1d_specta . . . . .	18
plot_2d_specta . . . . .	19
rscd_data_example . . . . .	19
rscd_dir . . . . .	20
rscd_freq . . . . .	21
rscd_mapplot . . . . .	21
weather_windows . . . . .	22
zmcomp2metconv . . . . .	23
%nin% . . . . .	23
<b>Index</b>	<b>25</b>

---

closest_point_field	<i>Find the closest point of the FIELD grid to the specified position</i>
---------------------	---

---

Description

Find the closest point of the FIELD grid to the specified position

Usage

closest\_point\_field(x, lat = NULL, closest = 1L, ...)

**Arguments**

x	vector of coordinates in the form longitude/latitude data frame
lat	alternatively, x and lat can be vector of the same length
closest	an integer to specify the number of point to output.
...	currently unused

**Value**

a list with two components: the closest point(s) of the grid and the distance (s).

**Examples**

```
semrev_west <- closest_point_field(c(-2.786, 47.239))
semrev_west
```

---

closest_point_spec	<i>Find the closest point of the SPEC grid to the specified position</i>
--------------------	--

---

**Description**

Find the closest point of the SPEC grid to the specified position

**Usage**

```
closest_point_spec(x, lat = NULL, closest = 1L, ...)
```

**Arguments**

x	vector of coordinates in the form longitude/latitude data frame
lat	alternatively, x and lat can be vector of the same length
closest	an integer to specify the number of point to output.
...	currently unused

**Value**

a list with two components: the closest point(s) of the grid and the distance (s).

**Examples**

```
semrev_west <- closest_point_spec(c(-2.786, 47.239))
semrev_west
```

---

code compute\_orbital\_speeds

*Compute the orbital speed at a given depth from the wave elevation 1D spectra*

---

## Description

Compute the orbital speed at a given depth from the wave elevation 1D spectra

## Usage

```
compute_orbital_speeds(spec, freq, z = 0, depth = Inf, output_speeds = FALSE)
```

## Arguments

spec	1D spectral data: TxM matrix
freq	the M frequencies
z	distance above the floor at which we want the orbital speed (single numeric)
depth	depth time series (vector length T. Recycled if a single value is given)
output_speeds	TRUE if the spectral speed are needed. Otherwise, returns the RMS (default)

## Value

depending on spec, a list with the spectral velocities for each component (if output\_speeds==FALSE) or a data.frame with a time series of horizontal and vertical components of (spectral) orbital speed.

## Examples

```
# Compute orbital speed for varying Hs
S <- t(sapply(1:10, function(h) {
  jonswap(h)$spec
}))
orb_speeds <- compute_orbital_speeds(S, rscd_freq, depth = 100, z = 10)
plot(1:10, orb_speeds[, 1],
     type = "l",
     ylim = range(orb_speeds),
     xlab = "Hs (m)",
     ylab = "Orbital speed RMS (m/s)")
lines(1:10, orb_speeds[, 2], type = "l", col = "red")
```

---

`compute_sea_state_1d_spectrum`*Compute sea\_state parameter from wave spectrum*

---

## Description

Compute sea\_state parameter from wave spectrum

## Usage

```
compute_sea_state_1d_spectrum(spec, ...)
```

## Arguments

<code>spec</code>	1D spectrum data, e.g. from <code>get_1Dspectrum</code>
<code>...</code>	currently unused

## Value

a tibble with the sea-state parameters computed from the time series of 1D spectrum

## Examples

```
rscd_params <- get_parameters(  
  node = "134865",  
  start = "1994-01-01",  
  end = "1994-01-31 23:00:00",  
  parameters = c("hs", "tp")  
)  
spec <- resourcecodedata::rscd_1d_spectra  
param_calc <- compute_sea_state_1d_spectrum(spec)  
oldpar <- par(mfcol = c(2, 1))  
plot(param_calc$time, param_calc$hs, type = "l", xlab = "Time", ylab = "Hs (m)")  
lines(rscd_params$time, rscd_params$hs, col = "red")  
plot(param_calc$time, param_calc$tp, type = "l", xlab = "Time", ylab = "Tp (s)")  
lines(rscd_params$time, rscd_params$tp, col = "red")  
par(oldpar)
```

---

`compute_sea_state_2d_spectrum`*Compute sea\_state parameter from wave directional spectrum*

---

## Description

Compute sea\_state parameter from wave directional spectrum

## Usage

```
compute_sea_state_2d_spectrum(spec, ...)
```

## Arguments

<code>spec</code>	2D spectrum data, e.g. from <code>get_2Dspectrum</code>
<code>...</code>	currently unused

## Value

a tibble with the sea-state parameters computed from the time series of 2D spectrum

## Examples

```
rscd_params <- get_parameters(  
  node = "134865",  
  start = "1994-01-01",  
  end = "1994-01-31 23:00:00",  
  parameters = c("hs", "tp")  
)  
spec <- resourcecodedata::rscd_2d_spectra  
param_calc <- compute_sea_state_2d_spectrum(spec)  
oldpar <- par(mfcol = c(2, 1))  
plot(param_calc$time, param_calc$hs, type = "l", xlab = "Time", ylab = "Hs (m)")  
lines(rscd_params$time, rscd_params$hs, col = "red")  
plot(param_calc$time, param_calc$tp, type = "l", xlab = "Time", ylab = "Tp (s)")  
lines(rscd_params$time, rscd_params$tp, col = "red")  
par(oldpar)
```

---

convert\_spectrum\_2d1d *Converts a 2D spectrum time series to a 1D spectrum*

---

## Description

Converts a 2D spectrum time series to a 1D spectrum

## Usage

```
convert_spectrum_2d1d(spec, ...)
```

## Arguments

spec	a structure with the needed fields, as an output from 'get_2Dspectrum' for example
...	unused yet

## Value

a structure comparable to 'get\_1Dspectrum'.

## Examples

```
spec <- resourcecodedata::rscd_2d_spectra
spec1D_RSCD <- resourcecodedata::rscd_1d_spectra
spec1D <- convert_spectrum_2d1d(spec)
# Check the differences, should be low
max(abs(spec1D_RSCD$ef - spec1D$ef))

# Plot the different spectrum
plot(spec1D$freq, spec1D$ef[, 1], type = "l", log = "xy")
lines(spec1D_RSCD$freq, spec1D_RSCD$ef[, 1], col = "red")

# Images
lims <- c(0, 360)
r <- as.POSIXct(round(range(spec1D$forcings$time), "hours"))
oldpar <- par(mfcol = c(2, 1))
image(spec1D$forcings$time, spec1D$freq, t(spec1D$th1m),
      zlim = lims,
      xlab = "Time",
      ylab = "Freq (Hz)",
      xaxt = "n",
      main = "Directionnal spreading"
)
axis.POSIXct(1, spec1D$forcings$time,
  at = seq(r[1], r[2], by = "week"),
  format = "%Y-%m-%d",
  las = 2
)
```

```

image(spec1D_RSCD$forcings$time, spec1D_RSCD$freq, t(spec1D_RSCD$th1m),
      zlim = lims,
      xlab = "Time",
      ylab = "Freq (Hz)",
      xaxt = "n"
)
axis.POSIXct(1, spec1D$forcings$time,
             at = seq(r[1], r[2], by = "week"),
             format = "%Y-%m-%d",
             las = 2
)
par(oldpar)

```

---

cut_directions	<i>Directional binning</i>
----------------	----------------------------

---

## Description

Cuts direction vector into directional bins

## Usage

```
cut_directions(directions, n_bins = 8, labels = NULL)
```

## Arguments

directions	vector of directions to be binned, in degree, 0° being the North.
n_bins	number of bins, default: 8 sectors.
labels	optional character vector giving the sectors names.

## Value

a factor vector the same size as directions with the values binned into sectors.

## Examples

```

# Example usage and demonstration
set.seed(123)
directions <- runif(20, 0, 360)

# Test with different numbers of bins
cat("Original directions:\n")
print(round(directions, 1))

cat("\n8 bins (default):\n")
bins_8 <- cut_directions(directions, n_bins = 8)
print(bins_8)

cat("\n4 bins:\n")

```



```
bins_4 <- cut_directions(directions, n_bins = 4)
print(bins_4)
```

---

cut_seasons	<i>Get season from date time object</i>
-------------	---

---

## Description

Get season from date time object

## Usage

```
cut_seasons(
  datetime,
  definition = "meteorological",
  hemisphere = "northern",
  labels = NULL
)
```

## Arguments

datetime	a POSIXct vector from with the season is constructed
definition	the definition used to compute the season. See details section.
hemisphere	in the Southern hemisphere, seasons are reversed compared to the Northern one.
labels	optional, a character vector of length fours with the seasons' names.

## Details

Available Definitions:

- meteorological: Standard seasons (Dec-Feb = Winter, etc.)
- astronomical: Based on equinoxes/solstices
- djf: Dec-Jan-Feb, Mar-Apr-May, Jun-Jul-Aug, Sep-Oct-Nov
- jfm: Jan-Feb-Mar, Apr-May-Jun, Jul-Aug-Sep, Oct-Nov-Dec
- fma: Feb-Mar-Apr, May-Jun-Jul, Aug-Sep-Oct, Nov-Dec-Jan
- amj, jas, ond: Alternative starting points for quarterly seasons

## Value

a Factor vector with 4 levels depending on the definitions (and labels if provided)

## Examples

```
dates <- seq(
  from = as.POSIXct("2023-01-15"),
  to = as.POSIXct("2023-12-15"),
  by = "month"
)
cut_seasons(dates)
```

---

dispersion	<i>Compute the dispersion relation of waves Find <math>k</math> s.t. <math>(2\pi f)^2 = g.k.\tanh(k.d)</math></i>
------------	---

---

**Description**

Compute the dispersion relation of waves Find  $k$  s.t.  $(2\pi f)^2 = g.k.\tanh(k.d)$

**Usage**

```
dispersion(frequencies, depth, iter_max = 200, tol = 1e-06)
```

**Arguments**

frequencies	frequency vector
depth	depth (m)
iter_max	maximum number of iterations in the solver
tol	tolerance for termination.

**Value**

the wave numbers (same size as frequencies)

**Examples**

```
freq <- seq(from = 0, to = 1, length.out = 100)
k1 <- dispersion(freq, depth = 1)
k10 <- dispersion(freq, depth = 10)
kInf <- dispersion(freq, depth = Inf)
plot(freq, k1, type = "l")
lines(freq, k10, col = "red")
lines(freq, kInf, col = "green")
```

---

fastTrapz	<i>Fast implementation of <code>pracma::trapz</code> from the Armadillo C++ library</i>
-----------	---

---

**Description**

Compute the area of a multivariate function with (matrix) values  $Y$  at the points  $x$ .

**Usage**

```
fastTrapz(x, Y, dim = 1L)
```

**Arguments**

x	x-coordinates of points on the x-axis (vector)
Y	y-coordinates of function values (matrix)
dim	an integer giving the subscripts which the function will be applied over. 1 indicates rows, 2 indicates columns

**Value**

a vector with one dimension less than Y

**Examples**

```
x = 1:10
Y = sin(pi/10*matrix(1:10,ncol=10,nrow=10))
fastTrapz(x*pi/10,Y,2)
```

---

fractional\_day\_of\_year

*Compute Fractional Day of Year from POSIXct*

---

**Description**

Calculates the fractional day of year from a POSIXct datetime object. The fractional day is zero-indexed, starting at 0 for January 1st at midnight and ending at approximately 365.958 for December 31st at 23:00 in a non-leap year (or 366.958 in a leap year).

**Usage**

```
fractional_day_of_year(datetime)
```

**Arguments**

datetime	A POSIXct object or vector of POSIXct objects representing date-time values. Must have a timezone attribute.
----------	--

**Details**

The function computes the time difference in hours between the input datetime and midnight on January 1st of the same year, then divides by 24 to obtain fractional days. The calculation accounts for leap years automatically.

**Value**

A numeric vector of the same length as `datetime`, containing fractional day of year values. January 1st at midnight corresponds to 0, and each hour adds approximately 0.04167 (1/24) to the value.

**Examples**

```

dates <- seq(
  from = as.POSIXct("2024-01-01 00:00:00", tz = "UTC"),
  to = as.POSIXct("2024-01-02 00:00:00", tz = "UTC"),
  by = "6 hours"
)
fractional_day_of_year(dates) # Returns 0.00, 0.25, 0.50, 0.75, 1.00

# End of year
dt_end <- as.POSIXct("2024-12-31 23:00:00", tz = "UTC")
fractional_day_of_year(dt_end) # Returns ~365.958

```

---

get_1d_spectrum	<i>Download the 1D spectrum data from IFREMER ftp</i>
-----------------	---

---

**Description**

Download the 1D spectrum data from IFREMER ftp

**Usage**

```
get_1d_spectrum(point, start = "1994-01-01", end = "1994-02-28")
```

**Arguments**

<b>point</b>	the location name (string) requested. Alternatively, the node number. The consistency is checked internally.
<b>start</b>	the starting date (as a string). The consistency is checked internally.
<b>end</b>	the ending date as a string

**Value**

A list with 12 elements:

**longitude** Longitude

**latitude** Latitude

**frequency1** Lower frequency

**frequency2** Upper frequency

**ef** Surface elevation variance spectral density

**th1m** Mean direction from first spectral moment

**th2m** Mean direction from second spectral moment

**sth1m** Mean directional spreading from first spectral moment

**sth2m** Mean directional spreading from second spectral moment

**freq** Central frequency

**forcings** A data.frame with 14 variables:

- time** Time
- dpt** Depth, positive downward
- wnd** Wind intensity, at 10m above sea level
- wnddir** Wind direction, comes from
- cur** Current intensity, at the surface
- curdir** Current direction, going to
- hs** Significant wave height
- fp** Peak wave frequency
- f02** Mean wave frequency
- f0m1** Mean wave frequency at spectral moment minus one
- th1p** Mean wave direction at spectral peak
- sth1p** Directional spreading at spectral peak
- dir** Mean wave direction
- spr** Mean directional spreading

**station** Station name

## Examples

```
spec1D <- get_1d_spectrum("SEMREV0", start = "1994-01-01", end = "1994-02-28")
if(!is.null(spec1D)){
  r <- as.POSIXct(round(range(spec1D$forcings$time), "month"))
  image(spec1D$forcings$time, spec1D$freq, t(spec1D$ef),
    xaxt = "n", xlab = "Time",
    ylab = "Frequency (Hz)"
  )
  axis.POSIXct(1, spec1D$forcings$time,
    at = seq(r[1], r[2], by = "week"),
    format = "%Y-%m-%d", las = 2
  )
}
```

---

get\_2d\_spectrum

*Download the 2D spectrum data from IFREMER ftp*

---

## Description

Download the 2D spectrum data from IFREMER ftp

## Usage

```
get_2d_spectrum(point, start = "1994-01-01", end = "1994-02-28")
```

**Arguments**

<b>point</b>	the location name (string) requested. Alternatively, the node number. The consistency is checked internally.
<b>start</b>	the starting date (as a string). The consistency is checked internally.
<b>end</b>	the ending date as a string

**Value**

A list with 9 elements:

**longitude** Longitude

**latitude** Latitude

**frequency1** Lower frequency

**frequency2** Upper frequency

**ef** Surface elevation variance spectral density

**th1m** Mean direction from first spectral moment

**th2m** Mean direction from second spectral moment

**sth1m** Mean directional spreading from first spectral moment

**sth2m** Mean directional spreading from second spectral moment

**freq** Central frequency

**dir** Directional bins

**forcings** A data.frame with 6 variables:

**time** Time

**dpt** Depth, positive downward

**wnd** Wind intensity, at 10m above sea level

**wnddir** Wind direction, comes from

**cur** Current intensity, at the surface

**curdir** Current direction, going to

**station** Station name

**Examples**

```
spec2D <- get_2d_spectrum("SEMREVO", start = "1994-01-01", end = "1994-02-28")
if(!is.null(spec2D)){
  image(spec2D$dir, spec2D$freq, spec2D$efth[, , 1],
        xlab = "Direction (°)",
        ylab = "Frequency (Hz)"
  )
}
```

---

get_parameters	<i>Download time series of sea-state parameters from RESOURCE-CODE database</i>
----------------	---

---

### Description

If the remote resource is unavailable or returns an error, the function returns NULL and emits an informative message.

### Usage

```
get_parameters(
  parameters = "hs",
  node = 42,
  start = as.POSIXct("1994-01-01 00:00:00", tz = "UTC"),
  end = as.POSIXct("1994-12-31 23:00:00", tz = "UTC")
)
```

### Arguments

parameters	character vector of sea-state parameters
node	single integer with the node to get
start	starting date (as integer, character or posixct)
end	ending date (as integer, character or posixct)

### Value

a tibble with N-rows and length(parameters) columns.

### Examples

```
rscd_data <- get_parameters(parameters = c("hs", "tp"), node = 42)
if(!is.null(rscd_data)) plot(rscd_data$time, rscd_data$hs, type = "l")
```

---

jonswap	<i>JONWSAP spectrum</i>
---------	-------------------------

---

### Description

Creates a JONWSAP density spectrum (one-sided), defined by its integral parameters.

### Usage

```
jonswap(hs = 5, tp = 15, fmax = rscd_freq, df = NULL, gam = 3.3)
```

Arguments

hs	Hs (default: 5m)
tp	Period (default: 10s)
fmax	higher frequency of the spectrum or vector of frequencies (default to resource-code frequency vector)
df	frequency step (unused if fmax=vector of frequencies)
gam	peak enhancement factor (default: 3.3)

Details

- Reference :
- O.G.Houmb and T.Overvik, "Parametrization of Wave Spectra and Long Term Joint Distribution of Wave Height and Period," in Proceedings, First International Conference on Behaviour of Offshore Structures (BOSS), Trondheim 1976. 23rd International Towing Tank Conference, vol. II, pp. 544-551
  - ITTC Committee, 2002, "The Specialist Committee on Waves - Final Report and Recommendations to the 23rd ITTC", Proc. ITTC, vol. II, pp. 505-736.

Value

Density spectrum with corresponding parameters

Examples

```
S1 <- jonswap(tp = 15)
S2 <- jonswap(tp = 15, fmax = 0.95, df = 0.003)
plot(S1, type = "l", ylim = c(0, 72))
lines(S2, col = "red")
abline(v = 1 / 15)
```

---

mean_direction	<i>Mean Direction</i>
----------------	-----------------------

---

Description

Function for computing the (weighted) arithmetic mean of directional data in **meteorological convention**.

Usage

```
mean_direction(directions, weights = NULL)
```

Arguments

directions	numeric vector of directions, in degree, 0° being the North
weights	numeric vector, usually wind speed or wave height.



**Value**

The (weighted) mean of the values in directions is computed.

**Examples**

```
# Test with some wind directions (unweighted)
wind_directions <- c(10, 20, 350, 5, 15) # Directions mostly around North
mean_dir <- mean_direction(wind_directions)
cat("Mean wind direction (unweighted):", round(mean_dir, 1), "degrees\n")
wind_directions <- c(350, 10, 20, 340, 30) # Directions around North
wind_speeds <- c(15, 5, 2, 12, 3) # Higher speeds for directions closer to North
mean_dir_weighted <- mean_direction(wind_directions, wind_speeds)
cat("Mean wind direction (weighted):", round(mean_dir_weighted, 1), "degrees\n")

# Compare weighted vs unweighted for the same data
mean_dir_unweighted <- mean_direction(wind_directions)
cat("Same data unweighted:", round(mean_dir_unweighted, 1), "degrees\n")
```

---

metconv2zmcomp

---

*Convert meteorological wind speed and direction to u/v components*


---

**Description**

Converts wind speed (magnitude) and direction (in degrees, meteorological convention: direction from which the wind blows, measured clockwise from north) into zonal (u) and meridional (v) components.

**Usage**

```
metconv2zmcomp(speed, direction, names = c("uwnd", "vwnd"))
```

**Arguments**

speed	Numeric vector of wind speeds.
direction	Numeric vector of wind directions in degrees (0° = from north, 90° = from east, 180° = from south, 270° = from west).
names	(optional) names to construct the resulting data.frame.

**Value**

A data.frame with two columns:

- u** Zonal wind component (m/s), positive eastward.
- v** Meridional wind component (m/s), positive northward.

**Examples**

```
# Example 1: North wind of 10 m/s (blowing southward)
metconv2zmcomp(10, 0)

# Example 2: East wind of 5 m/s (blowing westward)
metconv2zmcomp(5, 90)

# Example 3: South wind of 8 m/s (blowing northward)
metconv2zmcomp(8, 180)
```

---

plot_1d_spectra	<i>Plot a wave density 1D spectrum at a given time</i>
-----------------	--

---

**Description**

Plot a wave density 1D spectrum at a given time

**Usage**

```
plot_1d_spectra(spec, time = 1L, print_sea_state = TRUE, ...)
```

**Arguments**

spec	the spectral data, as an output from get_2Dspectrum
time	the time to plot. Either an integer or the date.
print_sea_state	should the sea_states parameters being plot ? Default to TRUE.
...	currently unused

**Value**

a ggplot object

**Examples**

```
plot_1d_spectra(resourcecodedata::rscd_1d_spectra, 1)
```

---

plot_2d_spectra	<i>Plot a wave density 2D spectrum</i>
-----------------	--

---

**Description**

Plot a wave density 2D spectrum

**Usage**

```
plot_2d_spectra(  
  spec,  
  time = 1L,  
  normalize = TRUE,  
  trim = 0.01,  
  cut_off = 0.4,  
  ...  
)
```

**Arguments**

spec	the spectral data, as an output from <code>get_2Dspectrum</code>
time	the time to plot. Either an integer or the date.
normalize	Should the spectrum be normalized to have maximum 1 before plotting
trim	removes the values of the spectral density lower than this value
cut_off	cut-off frequency above which the spectrum is not plotted
...	currently unused

**Value**

a ggplot object

**Examples**

```
plot_2d_spectra(resourcecodedata::rscd_2d_spectra, 1)
```

---

rscd_data_example	<i>Resourcecode data extract</i>
-------------------	----------------------------------

---

**Description**

An extract of the Resourcecode Hindcast database, at node 123456, spanning from 1994-01-01 00:00:00 to 1999-12-31 23:00:00.

**Usage**

```
rscd_data_example
```

**Format**

**A data frame with 87,648 rows and 7 variables::**

**time** POSIXct. Timestamp in UTC (hourly resolution).

**hs** numeric. Significant wave height (m).

**tp** numeric. Peak wave period (s).

**dp** numeric. Mean wave direction (degrees, coming from true North, clockwise).

**uwnd** numeric. Zonal (east–west) component of the 10-m wind (m/s). Positive eastward.

**vwnd** numeric. Meridional (north–south) component of the 10-m wind (m/s). Positive northward.

**dpt** numeric. Depth (m).

**Source**

User Manual of the RESOURCECODE database <https://archimer.ifremer.fr/doc/00751/86306/>

---

rscd_dir	<i>Resourcecode directional bins</i>
----------	--------------------------------------

---

**Description**

(equivalent to a directional resolution of 10°;

**Usage**

```
rscd_dir
```

**Format**

```
rscd_dir:
```

An array of length 36 with the directionnal bins

**Source**

User Manual of the RESOURCECODE database <https://archimer.ifremer.fr/doc/00751/86306/>

---

rscd_freq	<i>Resourcecode frequency vector of 1D and 2D spectra</i>
-----------	---

---

### Description

The wave spectrum discretization considers 36 frequencies, starting from 0.0339 Hz up to 0.9526 Hz with a frequency increment factor of 1.1

### Usage

```
rscd_freq
```

### Format

rscd\_freq:  
An array 36 elements with the frequencies values

### Source

User Manual of the RESOURCECODE database <https://archimer.ifremer.fr/doc/00751/86306/>

---

rscd_mapplot	<i>Create a map of the provided variable on the RESOURCECODE field grid</i>
--------------	---

---

### Description

Create a map of the provided variable on the RESOURCECODE field grid

### Usage

```
rscd_mapplot(
  z,
  name = "Depth (m)",
  zlim = NULL,
  palette = "YlOrRd",
  direction = 1,
  transform = "identity"
)
```

**Arguments**

<code>z</code>	the data to plot: a vector of the same size as the grid (328,030 rows).
<code>name</code>	name of the variable plotted, to be included in the legend.
<code>zlim</code>	limits of the scale. See <a href="#">continuous_scale</a> for details.
<code>palette</code>	If a string, will use that named palette. See <a href="#">scale_colour_brewer</a> for other options.
<code>direction</code>	Sets the order of colours in the scale. See <a href="#">scale_colour_brewer</a> for details.
<code>transform</code>	Transformation to apply to the color scale. See <a href="#">continuous_scale</a> for details.

**Value**

a ggplot2 object

**Examples**

```
rscd_mapplot(resourcecodedata::rscd_field$depth)
```

---

weather_windows	<i>Compute Weather Windows</i>
-----------------	--------------------------------

---

**Description**

Computes and returns start date of each weather window, implemented in C++ for speed.

**Usage**

```
weather_windows(
  valid_periods,
  window_length,
  allow_overlap = TRUE,
  time_step = 3600
)
```

**Arguments**

<code>valid_periods</code>	A data frame with a 'time' column (POSIXct).
<code>window_length</code>	Minimum window duration (hours).
<code>allow_overlap</code>	Logical; If TRUE, the algorithm searches for window, if a window is found, search of next window will start from the end of the previous window. If FALSE, it uses continuous window search: The algorithm searches for window starting from every time step that meets the criteria.
<code>time_step</code>	Expected time step between consecutive timestamps (seconds).

**Value**

POSIXct vector of detected window start times.

---

zmcomp2metconv	<i>Convert u/v to meteorological wind speed and direction</i>
----------------	---

---

**Description**

Converts wind or current zonal and meridional velocity components to magnitude and direction according to meteorological convention.

**Usage**

```
zmcomp2metconv(u, v = NULL, names = c("wspd", "wdir"))
```

**Arguments**

- u                    zonal velocity (1D vector) or matrix with zonal and meridional velocity (Nx2 matrix)
- v                    meridional velocity (1D vector)
- names                names to construct the resulting data.frame

**Value**

a Nx2 data.frame with the norm and direction (meteorological convention)

**Examples**

```
u <- matrix(rnorm(200), nrow = 100, ncol = 2)
vdir <- zmcomp2metconv(u)
```

---

%nin%	<i>Value Matching</i>
-------	-----------------------

---

**Description**

Value Matching

**Usage**

```
x %nin% table
```

**Arguments**

- x                    value to search
- table                table of values

**Value**

the opposite of x *%in%* table

**Examples**

```
1:10 %in% c(1, 3, 5, 9)
1:10 %nin% c(1, 3, 5, 9)
```



# Index

## \* datasets

- rscd\_data\_example, [19](#)
- rscd\_dir, [20](#)
- rscd\_freq, [21](#)
- %nin%, [23](#)

- closest\_point\_field, [2](#)
- closest\_point\_spec, [3](#)
- compute\_orbital\_speeds, [4](#)
- compute\_sea\_state\_1d\_spectrum, [5](#)
- compute\_sea\_state\_2d\_spectrum, [6](#)
- continuous\_scale, [22](#)
- convert\_spectrum\_2d1d, [7](#)
- cut\_directions, [8](#)
- cut\_seasons, [9](#)

- dispersion, [10](#)

- fastTrapz, [10](#)
- fractional\_day\_of\_year, [11](#)

- get\_1d\_spectrum, [12](#)
- get\_2d\_spectrum, [13](#)
- get\_parameters, [15](#)

- jonswap, [15](#)

- mean\_direction, [16](#)
- metconv2zmcomp, [17](#)

- plot\_1d\_spectra, [18](#)
- plot\_2d\_spectra, [19](#)

- rscd\_data\_example, [19](#)
- rscd\_dir, [20](#)
- rscd\_freq, [21](#)
- rscd\_mapplot, [21](#)

- scale\_colour\_brewer, [22](#)

- weather\_windows, [22](#)

- zmcomp2metconv, [23](#)