Package 'eseis'

December 17, 2019

Type Package

Title Environmental Seismology Toolbox

Version 0.5.0

Date 2019-12-16

Maintainer Michael Dietze `<mdietze@gfz-potsdam.de>`

Description Environmental seismology is a scientific field that studies the seismic signals, emitted by Earth surface processes. This package provides all relevant functions to read/write seismic data files, prepare, analyse and visualise seismic data, and generate reports of the processing history.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.6.0)

LinkingTo Rcpp (>= 0.12.5)

Imports sp, multitaper, raster, rgdal, caTools, signal, fftw, matrixStats, methods, IRISSeismic, XML, shiny, rmarkdown, reticulate, Rcpp (>= 0.12.5)

Suggests plot3D, rgl

SystemRequirements gipptools dataselect

RoxygenNote 7.0.2

NeedsCompilation yes

Author Michael Dietze [cre, aut, trl],
Christoph Burow [ctb],
Sophie Lagarde [ctb, trl]

Repository CRAN

Date/Publication 2019-12-17 11:30:02 UTC

R topics documented:

`aux_commondt` *Identify highest common sampling interval*

Description

The function compares the sampling intervals of a list of `eseis` objects and identifies the highest common sampling interval (dt) as well as the aggregation factors for each `eseis` object needed to reach this common sampling interval.

Usage

```
aux_commondt(data, dt)
```

Arguments

`data`	`list` of `eseis` objects or vector of sampling intervals to be checked for highest common sampling interval
`dt`	`Numeric` vector of length one, user-defined common sampling frequency for which aggregation factors shall be computed.

Value

`list` object with elements `dt` (highest common sampling interval) and `agg` (aggregation factors for each of the input data sets to reach the common sampling interval)

Author(s)

Michael Dietze

Examples

```
## Not run:

## TO BE WRITTEN

## End(Not run)
```

`aux_eseisobspy` *Convert eseis object to ObsPy stream object*

Description

The function converts an eseis object to an ObsPy stream object. The functionality is mainly useful when running ObsPy through R using the package 'reticulate'. Currently, only single traces (i.e., single eseis objects) can be converted. Thus, to convert multiple traces, these need to be converted individually and added to the first trace using ObsPy functionalities.

Usage

```
aux_eseisobspy(data)
```

Arguments

`data`	`eseis` object, `list` element.

Value

`ObsPy` stream object as defined by the architecture of package 'reticulate'.

Author(s)

Michael Dietze

Examples

```
## Not run:

## load ObsPy library with package 'reticulate'
## (requires ObsPy to be installed on the computer)
obspy <- reticulate::import("obspy")

## load example data set
data(rockfall)

## convert example eseis object to ObsPy stream object
x <- aux_eseisobspy(data = rockfall_eseis)

## filter data set using ObsPy
x_filter <- obspy$traces[[1]]$filter(type = "bandpass",
                                     freqmin = 0.5,
                                     freqmax = 1.0)

## plot filtered trace using ObsPy plotting routine
x$traces[[1]]$plot()


## End(Not run)
```

`aux_fixmseed` *Fix corrupt miniseed files*

Description

This function is a wrapper for the library 'dataselect' from IRIS. It reads a corrupt mseed file and saves it in fixed state. Therefore, the function requires dataselect being installed (see details).

Usage

```
aux_fixmseed(file, input_dir, output_dir, software)
```

Arguments

`file`	`Character` vector, seismic file to process.
`input_dir`	`Character` value, path to input directory, i.e., the directory where the files to process are located.
`output_dir`	`Character` value, path to output directory, i.e., the directory where the processed files are written to. This must be different from `input_dir`.
`software`	`Character` value, path to the 'dataselect' library, required unless the path to the library is made gobally visible.

Details

The library 'dataselect' can be downloaded at https://github.com/iris-edu/dataselect and requires compilation (see README file in dataselect directory). The function goes back to an email discussion with Gillian Sharer (IRIS team), many thanks for pointing me at this option to process corrupt mseed files.

Value

a set of mseed files written to disk.

Author(s)

Michael Dietze

Examples

```
## Not run:

aux_fixmseed(file = list.files(path = "~/data/mseed",
                               pattern = "miniseed"),
                    input_dir = "~/data/mseed",
                    software = "~/software/dataselect-3.17")


## End(Not run)
```

---

aux_getevent *Load seismic data of a user-defined event*

Description

The function loads seismic data from a data directory structure (see `aux_organisecubefiles()`) based on the event start time, duration, component and station ID.

Usage

```
aux_getevent(
  start,
  duration,
  station,
  component = "BHZ",
  format = "sac",
  dir,
  simplify = TRUE,
  eseis = TRUE,
  try = FALSE
)
```

Arguments

`start`	`POSIXct` value, start time of the data to import.
`duration`	`Numeric` value, duration of the data to import, in seconds.
`station`	`Character` value, seismic station ID, which must correspond to the ID in the file name of the data directory structure (cf. `aux_organisecubefiles`).
`component`	`Character` value, seismic component, which must correspond to the component name in the file name of the data directory structure (cf. `aux_organisecubefiles`). Default is `"BHZ"` (vertical component of a sac file).
`format`	`Character` value, seismic data format. One out of `"sac"` and `"mseed"`. Default is `"sac"`.
`dir`	`Character` value, path to the seismic data directory.
`simplify`	`Logical` value, option to simplify output when possible. This basically means that if only data from one station is loaded, the list object will have one level less. Default is `TRUE`.
`eseis`	`Logical` value, option to read data to an `eseis` object (recommended, see documentation of `aux_initiateeseis`), default is `TRUE`
`try`	`Logical` value, option to run the fuction in try-mode, i.e., to let it return `NA` in case an error occurs during data import. Default is `FALSE`.

Details

The function assumes complete data sets, i.e., not a single hourly data set must be missing. The time vector is loaded only once, from the first station and its first component. Thus, it is assumed that all loaded seismic signals are of the same sampling frequency and length.

Value

A `list` object containing either a set of `eseis` objects or a data set with the time vector (`$time`) and a list of seismic stations (`$station_ID`) with their seismic signals as data frame (`$signal`). If `simplify = TRUE` (the default option) and only one seismic station is provided, the output object containseither just one eseis object or the vectors for `$time` and `$signal`.

Author(s)

Michael Dietze

Examples

```
## set seismic data directory
dir_data <- paste0(system.file("extdata", package="eseis"), "/")

## load the z component data from a station
data <- aux_getevent(start = as.POSIXct(x = "2017-04-09 01:20:00",
                                        tz = "UTC"),
                      duration = 120,
                      station = "RUEG1",
                      component = "BHZ",
                      dir = dir_data)
## plot signal
plot_signal(data = data)

## load data from two stations
data <- aux_getevent(start = as.POSIXct(x = "2017-04-09 01:20:00",
                                        tz = "UTC"),
                     duration = 120,
                     station = c("RUEG1", "RUEG2"),
                     component = "BHZ",
                     dir = dir_data)

## simplify data structure
data <- lapply(X = data, FUN = function(data) {data[[1]]})

## plot both signals
par(mfcol = c(2, 1))
lapply(X = data, FUN = plot_signal)
```

aux_getFDSNdata *Download seismic data from FDSN data base*

Description

The function accesses the specified FDSN internet data base(s) and downloads seismic data based on the network and station IDs and time constraints.

Usage

```
aux_getFDSNdata(
  start,
  duration,
  channel = "BHZ",
  network,
  station,
  url,
  link_only = FALSE,
  eseis = TRUE
)
```

Arguments

`start`	`POSIXct` value, start time of the data to query.
`duration`	`Numeric` value, length of the data to query, in seconds.
`channel`	`Character` value, seismic channel to get. Default is `"BHZ"`.
`network`	`Character` vector, two-character FDSN network ID.

`station`	`Character` vector, FDSN station ID.
`url`	`Character` vector, FDSN URL.
`link_only`	`Logical` vector, return only FDSN link instead of downloading and importing the data.
`eseis`	`Logical` scalar, option to read data to an `eseis` object (recommended, see documentation of `aux_initiateeseis`), default is `TRUE`

Details

A convenient way to get all the required input data is using the function `aux_getFDSNstation` before. It will return all the information in a structured way.

It is possible to use the function to process more than one data set. In this case, the arguments `network`, `station` and `url` must match pairwise. The arguments `start`, `duration` and `channel` will be treated as constants if not also provided as vectors.

Value

`List` object with imported seismic data for each provided set of input arguments.

Author(s)

Michael Dietze

See Also

aux_get_FDSNstation, read_mseed

Examples

```
## Not run:

## get stations < 0.6 degrees away from Piz Chengalo collapse
x <- aux_getFDSNstation(centre = c(46.3, 9.6),
                        radius = 0.6,
                        access = TRUE)

## sort statiions by distance
x <- x[order(x$distance),]

## download available data
d <- aux_getFDSNdata(start = as.POSIXct(x = "2017-08-23 07:30:00",
                                        tz = "UTC"),
                     duration = 180,
                     network = x$network_ID,
                     station = x$station_code,
                     url = x$network_url)

## remove stations without available data
x <- x[!unlist(lapply(d, is.null)),]
```

```
d <- d[!unlist(lapply(d, is.null))]

## generate plots of the three nearest stations
par(mfcol = c(3, 1))

for(i in 1:3) {

  plot_signal(data = d[[i]],
              main = paste(x$ID[i],
                           " | ",
                           round(x$distance[i], 2),
                           "distance (DD)"))
}

## End(Not run)
```

aux_getFDSNstation *Query FDSN data base for stations*

Description

This function queries as series of data bases for seismic stations that match a set of criteria for seismic data. The criteria include signal time stamp and location, and component. The returned data can be used to download data using the function `aux_FDSNdata`.

Usage

```
aux_getFDSNstation(centre, radius, start, access, url)
```

Arguments

`centre`	`Numeric` vector of length two, center coordinates of the location to search data for (`c(latitude,longitude)`). Units must be decimal degrees.
`radius`	`Numeric` value, radius within which to search for seismic stations. Unit must be decimal degrees.
`start`	`POSIXct` value, start time of the data to query. If omitted, stations are queried for the full time available.
`access`	`Logical` value, access type of the data. If omitted, all data sets are returned, if set `TRUE`, only data with access flag `"open"` are returned.
`url`	`Character` vector, optional other FDSN base web addresses to search for stations. See details for default addresses and their format.

Details

The function requires a working internet connection to perform the query. It uses the following FDSN data bases by default:

- `orfeus` `"http://www.orfeus-eu.org"`
- `geofon` `"http://geofon.gfz-potsdam.de/"`
- `bgr` `"http://eida.bgr.de"`
- `sss` `"http://eida.ethz.ch"`

Other FDSN data base addresses can be provided in the same way as the addresses in the above list. They need to be provided as character vector. For a list of addresses see `"http://www.fdsn.org/webservices/datacenter` and `"http://docs.obspy.org/packages/obspy.clients.fdsn.html#module-obspy.clients.fdsn"`.

Value

`Data frame` with query results. The data frame contains information for all seismic stations fulfilling the defined criteria.

Author(s)

Michael Dietze

See Also

aux_get_FDSNdata, aux_getIRISstation

Examples

```
## Not run:

x <- aux_getFDSNstation(start = as.POSIXct(x = "2010-01-01 22:22:22",
                                           tz = "UTC"),
                        centre = c(45, 10),
                        radius = 1)

## optionally plot station locations on a map (requires RgoogleMaps)
center <- c(mean(x$station_latitude),
            mean(x$station_longitude))

zoom <- min(RgoogleMaps::MaxZoom(range(x$station_latitude),
                                 range(x$station_longitude)))

Map <- RgoogleMaps::GetMap(center = center,
                           zoom = zoom,
                           maptype = "terrain")

RgoogleMaps::PlotOnStaticMap(MyMap = Map,
                             lat = x$station_latitude,
                             lon = x$station_longitude,
                             pch = 15,
                             col = 4)

## End(Not run)
```

aux_getIRISdata *Download seismic data from IRIS data base*

Description

This function accesses the IRIS internet data base of seismic signals and downloads seismic data based on the provided SNCL string and time information. The downloaded data is converted to the same structure as would be expected from read_sac or read_mseed.

Usage

```
aux_getIRISdata(
  start,
  duration,
  sncl,
  quality = "D",
  ID_iris = "IrisClient",
  eseis = TRUE
)
```

Arguments

start	POSIXct value, start time of the data to query.
duration	Numeric value, length of the data to query, in seconds.
sncl	Character vector, SNCL string used to identify station and component of interest. These strings should match the time criteria. Typically, the SNCL string can be taken from the output of the function aux_getirisstations.
quality	Character value, quality level of the data. One out of "D" (The state of quality control of the data is indeterminate), "R" (Raw Waveform Data with no Quality Control), "Q" (Quality Controlled Data, some processes have been applied to the data), "M" (Data center modified, time-series values have not been changed), "B". Default is "D".
ID_iris	Character value, IRIS ID. Default is "IrisClient".
eseis	Logical scalar, option to read data to an eseis object (recommended, see documentation of aux_initiateeseis), default is TRUE

Details

The function makes use of the package 'IRISSeismic'. It requires a working internet connection to perform the download.

Value

`List` with downloaded seismic data. For each element in `sncl`, a list element is created, which in turn contains a list with the typical seismic data organisation as, for example, created by `read_sac`.

Author(s)

Michael Dietze

Examples

```
## Not run:

sncl <- aux_getIRISstation(start = as.POSIXct("2010-01-01 22:22:22",
                                              tz = "UTC"),
                            duration = 120,
                            location = c(53, 13),
                            radius = 0.7,
                            component = "BHZ")

s <- aux_getIRISdata(start = as.POSIXct("2010-01-01 22:22:22",
                                        tz = "UTC"),
                        duration = 120,
                        sncl = sncl$sncl[1])

plot_signal(data = s[[1]])

## End(Not run)
```

`aux_getIRISstation` *Query IRIS data base for stations*

Description

This function queries the IRIS data base for seismic stations that match a set of criteria for seismic data. The criteria include signal time stamp and location, component and a search radius. The returned SNCL strings can be used to download data using the function `aux_getIRISdata`.

Usage

```
aux_getIRISstation(
  start,
  duration,
  location,
  radius = 10,
  component = "BHZ",
  ID_iris = "IrisClient"
)
```

Arguments

`start`	`POSIXct` value, start time of the data to query.
`duration`	`Numeric` value, length of the data to query, in seconds.
`location`	`Numeric` vector of length two, coordinates of the seismic source, in decimal degrees (i.e., latitude and longitude).
`radius`	`Numeric` value, search radius for the query, in decimal degrees. Default is `10` (about 1100 km).
`component`	`Character` value, signal component to check for. One out of `"BHE"`, `"BHN"` and `"BHZ"`. Currently, only one component can be defined per search. Default is `"BHZ"`.
`ID_iris`	`Character` value, IRIS ID. Default is `"IrisClient"`.

Details

The function makes use of the package IRISSeismic. It requires a working internet connection to perform the query.

Value

`Data frame` with query results. The data frame contains information for all seismic stations fulfilling the defined criteria.

Author(s)

Michael Dietze

Examples

```
## Not run:

x <- aux_getIRISstation(start = as.POSIXct("2010-01-01 22:22:22",
                          tz = "UTC"),
                          duration = 3 * 3600,
                          location = c(53, 13),
                          radius = 1,
                          component = "BHZ")

## End(Not run)
```

aux_gettemperature *Extract temperature data from cube files.*

Description

This function reads auxiliary information stored in Omnirecs/Digos Datacube files and extracts the temperature data that is stored along with each GPS tag. Optionally, the data is interpolated to equal intervals.

Usage

```
aux_gettemperature(input_dir, logger_ID, interval, cpu, gipptools)
```

Arguments

`input_dir`	`Character` value, path to directory where all cube files to be processed as stored. Each set of files from one logger must be stored in a separate sub-directory named after the cube ID.
`logger_ID`	`Character` vector, logger ID.
`interval`	`Numeric` value, time interval (minutes) to which temperature data is interpolated. No interpolation is performed if this argument is omitted.
`cpu`	`Numeric` value, fraction of CPUs to use for parallel processing. If omitted, one CPU is used.
`gipptools`	`Character` value, path to gipptools or cubetools directory.

Details

This feature is ony available for Omnirecs/Digos Datacube that were produced since 2015, i.e., whose GPS output files also record the temperature inside the logger. Generating an ACSII GPS tag file using the gipptools software requires a few minutes time per daily file.

Value

A `list` of `data frames` with time and temperature values for each cube data logger.

Author(s)

Michael Dietze

Examples

```
## uncomment to use
# t <- aux_gettemperature(input_dir = "input",
#                         logger_ID = c("ANN", "ABT"),
#                         interval = 15,
#                         gipptools = "~/software/gipptools-2015.225/")
```

aux_hvanalysis *Perform H-V-spectral ratio analysis of a seismic data set*

Description

This function cuts a three component seismic data set into time windows that may or may not overlap and calculates the spectral ratio for each of these windows. It returns a matrix with the ratios for each time slice. Thus, it is a wrapper for the function `signal_hvratio`. For further information about the technique and function arguments see the description of `signal_hvratio`.

Usage

```
aux_hvanalysis(
  data,
  time,
  window,
  overlap = 0,
  dt,
  method = "periodogram",
  kernel,
  order = "xyz",
  plot = FALSE,
  ...
)
```

Arguments

data	`List`, `data frame` or `matrix`, seismic componenents to be processed. If `data` is a matrix, the components must be organised as columns. Also, `data` can be a list of `eseis` objects.
time	`POSIXct` vector with time values. If omitted, an synthetic time vector will be created, based on `dt`.
window	`Numeric` scalar, time window length in seconds used to calculate individual spectral ratios. Set to 10 percent of the time series length by default.
overlap	`Numeric` value, fraction of window overlap.
dt	`Numeric` value, sampling period.
method	`Character` value, method for calculating the spectra. One out of `"periodogram"` , `"autoregressive"` and `"multitaper"`, default is `"periodogram"`.
kernel	`Numeric` value, window size (defined by number of samples) of the moving window used for smoothing the spectra. By default no smoothing is performed.
order	`Character` value, order of the seismic components. Describtion must contain the letters `"x"`,`"y"` and `"z"` in the order according to the input data set. Default is `"xyz"` (NW-SE-vertical).
plot	`Logical` value, toggle plot output. Default is `FALSE`.
...	Additional arguments passed to the plot function.

Value

A `matrix` with the h-v-frequency ratios for each time slice.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(earthquake)

## ATTENTION, THIS EXAMPLE DATA SET IS FAR FROM IDEAL FOR THIS PURPOSE

## detrend data
s <- signal_detrend(data = s)

## calculate the HV ratios straightforward
HV <- aux_hvanalysis(data = s,
                     dt = 1 / 200,
                     kernel = 100)

## calculate the HV ratios with plot output, userdefined palette
plot_col <- colorRampPalette(colors = c("grey", "darkblue", "blue", "orange"))
HV <- aux_hvanalysis(data = s,
                     dt = 1 / 200,
                     kernel = 100,
                     plot = TRUE,
                     col = plot_col(100))

## calculate the HV ratios with optimised method settings
HV <- aux_hvanalysis(data = s,
                     time = t,
                     dt = 1 / 200,
                     window = 10,
                     overlap = 0.9,
                     method = "autoregressive",
                     plot = TRUE,
                     col = plot_col(100),
                     xlab = "Time (UTC)",
                     ylab = "f (Hz)")

## calculate and plot stack (mean and sd) of all spectral ratios
HV_mean <- apply(X = HV, MARGIN = 1, FUN = mean)
HV_sd <- apply(X = HV, MARGIN = 1, FUN = sd)
HV_f <- as.numeric(rownames(HV))

plot(x = HV_f, y = HV_mean, type = "l", ylim = c(0, 50))
lines(x = HV_f, y = HV_mean + HV_sd, col = 2)
lines(x = HV_f, y = HV_mean - HV_sd, col = 2)
```

`aux_initiateeseis` *Initiate an eseis object*

Description

The function generates an empty eseis object, starting with processing step 0. The object contains no data and the history only contains the system information.

Usage

```
aux_initiateeseis()
```

Details

The S3 object class `eseis` contains the data vector (`$signal`), a meta information list (`$meta`) with all essential seismic meta data - such as sampling interval, station ID, component, start time of the stream or file name of the input file - a list with header data of the seismic source file (`$header`), and a history list (`$history`), which records all data manipulation steps of an (`eseis`) object. The element (`$meta`) will be used by functions of the package to look for essential information to perform data manipulations (e.g., the sampling interval). Thus, working with (`eseis`) objects is convenient and less prone to user related errors/bugs, given that the meta information is correct and does not change during the processing chain; package functions will update the meta information whenever necessary (e.g., `signal_aggregate`). The element `$header` will only be present if a seismic source file has been imported.

The history element is the key feature for transparent and reproducable research using this R package. An `eseis` object records a history of every function it has been subject to, including the time stamp, the function call, all used function arguments and their associated values, and the overall processing duration in seconds. The history is updated whenever an `eseis` object is manipulated with one of the functions of this package (with a few exceptions, mainly from the aux_... category).

Value

`S3` list object of class `eseis`.

Author(s)

Michael Dietze

Examples

```
## initiate eseis object
aux_initiateeseis()
```

`aux_obspyeseis` *Convert ObsPy object to eseis object*

Description

The function converts an ObsPy stream object to an eseis object. The functionality is mainly useful when running ObsPy through R using the package 'reticulate'.

Usage

```
aux_obspyeseis(data, simplify = TRUE)
```

Arguments

`data`	obspy stream object, `list` element, created by running ObsPy through R using the package 'reticulate'.
`simplify`	`Logical` value, option to simplify output when possible. This basically means that if there is only trace object in the ObsPy stream, the list object will have one level less. Default is `TRUE`.

Value

`eseis` object.

Author(s)

Michael Dietze

Examples

```

## Not run:

## load ObsPy library with package 'reticulate'
## (requires ObsPy to be installed on the computer)
obspy <- reticulate::import("obspy")

## set seismic data directory
dir_data <- paste0(system.file("extdata", package="eseis"), "/")

## read miniseed file to stream object via ObsPy
x <- obspy$read(pathname_or_url = "dir_data/2017/99/RUEG1.17.99.00.00.00.BHZ.SAC")

## convert ObsPy stream object to eseis object
y <- aux_obspyeseis(data = x)

## plot eseis object
plot_signal(y)
```

```
## End(Not run)
```

---

aux_organisecentaurfiles

Reorganise seismic files recorded by Nanometrics Centaur loggers

---

Description

This function optionally converts mseed files to sac files and organises these in a coherent directory structure, by year, Julian day, (station, hour and channel). It depends on the cubetools or gipptools software package (see details). The function is at an experimental stage and only used for data processing at the GFZ Geomorphology section, currently.

Usage

```
aux_organisecentaurfiles(
  stationfile,
  input_dir,
  output_dir,
  format = "sac",
  channel_name = "bh",
  cpu,
  gipptools,
  file_key = "miniseed",
  network
)
```

Arguments

`stationfile`	`Character` value, file name of the station info file, with extension. See `aux_stationinfofile`.
`input_dir`	`Character` value, path to directory where all files to be processed as stored. Each set of files from one logger must be stored in a separate sub-directory named after the logger ID (which in turn must be the four digit number of the logger).
`output_dir`	`Character` value, path to directory where output data is written to.
`format`	`Character` value, output file format. One out of `"mseed"` and `"sac"`. Default is `"sac"`.
`channel_name`	`Character` value, output file extension. One out of `"bh"` and `"p"`. Default is `"bh"`.
`cpu`	`Numeric` value, fraction of CPUs to use for parallel processing. If omitted, one CPU is used.
`gipptools`	`Character` value, path to gipptools or cubetools directory.
`file_key`	`Character` value, file name extension of the files to process. Only files with this extension will be processed. Default is `"miniseed"`.
`network`	`Character` value, optional seismic network code.

Details

The function assumes that the Nanometrics Centaur data logger directory contains only hourly mseed files. These hourly files are organised in a coherent directory structure which is organised by year and Julian day. In each Julian day directory the hourly files are placed and named according to the following scheme: STATIONID.YEAR.JULIANDAY.HOUR.MINUTE.SECOND.CHANNEL. The function requires that the software cubetools (`http://www.omnirecs.de/documents.html`) or gipptools (`http://www.gfz-potsdam.de/en/section/geophysical-deep-sounding/infrastructure/geophysical`) are installed.
Specifying an input directory (`input_dir`) is mandatory. This input directory must only contain the subdirectories with mseed data for each Centaur logger. The subdirectory must be named after the four digit Centaur ID and contain only mseed files, regardless if further subdirectories are used (e.g., for calendar days).

In the case a six-channel Centaur is used to record signals from two sensors, in the station info file (cf. `aux_stationinfofile()`) the logger ID field must contain the four digit logger ID and the channel qualifiers, e.g., "AH" (first three channels) or "BH" (last three channels), separated by an underscore.

Value

A set of hourly seismic files written to disk.

Author(s)

Michael Dietze

Examples

```
## Not run:

## basic example with minimum effort
aux_organisecentaurfiles(stationfile = "output/stationinfo.txt",
                         input_dir = "input",
                         gipptools = "software/gipptools-2015.225/")


## End(Not run)
```

`aux_organisecubefiles` *Convert Omnirecs/Digos Datacube files to mseed or sac files and organise in directory structure.*

Description

This function converts Omnirecs/Digos Datacube files to hourly mseed or sac files and organises these in a coherent directory structure, by year, Julian day, (station, hour and channel). It depends on the cubetools or gipptools software package (see details).

Usage

```
aux_organisecubefiles(
  stationfile,
  input_dir,
  output_dir,
  format = "sac",
  channel_name = "bh",
  cpu,
  fringe = "constant",
  verbose = FALSE,
  gipptools,
  heapspace,
  mseed_manual = FALSE,
  mseed_keep = FALSE
)
```

Arguments

`stationfile`	`Character` value, file name of the station info file, with extension. See `aux_stationinfofile`.
`input_dir`	`Character` value, path to directory where all cube files to be processed are stored. Each set of files from one logger must be stored in a separate sub-directory named after the cube ID.
`output_dir`	`Character` value, path to directory where output data is written to.
`format`	`Character` value, output file format. One out of `"mseed"` and `"sac"`. Default is `"sac"`.
`channel_name`	`Character` value, output file extension. One out of `"bh"` and `"p"`. Default is `"bh"`.
`cpu`	`Numeric` value, fraction of CPUs to use for parallel processing. If omitted, one CPU is used.
`fringe`	`Character` value, option to handle data outside the GPS-tagged time span. One out of `"skip"`, `"nominal"` or `"constant"`. Default is `"constant"`.
`verbose`	`Logical` value, option to enable extended screen output of cubetools operations. Default is `FALSE`.
`gipptools`	`Character` value, path to gipptools or cubetools directory.
`heapspace`	`Numeric` value, heap space assigned to the Jave Runtime Environment, e.g., `4096`. Should be increased if the cube to mseed conversion fails (announced if `verbose = TRUE`).
`mseed_manual`	`Logical` value, option to convert mseed files manually. See details. Default is `FALSE`, i.e., the function converts cube files to mseed files using the GIPP tools.
`mseed_keep`	`Logical` value, option to keep mseed files instead of deleting them. Default is `FALSE`.

Details

The function converts seismic data from the cube file format to either mseed (cf. `read_mseed`) or sac (cf. `read_sac`) and cuts the daily cube files to hourly files. These hourly files are organised in a coherent directory structure which is organised by year and Julian day. In each Julian day directory the hourly files are placed and named after the following scheme: STATIONID.YEAR.JULIANDAY.HOUR.MINUTE.SECOND.CHANNEL.
The function requires that the software cubetools (`http://www.omnirecs.de/documents.html`) or gipptools (`http://www.gfz-potsdam.de/en/section/geophysical-deep-sounding/infrastructure/geophysical`
are installed.
Specifying an input directory (`input_dir`) is mandatory. This input directory must only contain the subdirectories with the cube files to process, each set of cube files must be located in a separate subdirectory and these subdiretories must have the same name as specified by the logger IDs (`logger_ID`). An appropriate structure would be something like:

1. input
   (a) A1A
      i. file1.A1A
      ii. file2.A1A
   (b) A1B
      i. file1.A1B
      ii. file2.A1B

With one of the latest updates of either R or Java the cache size for converting cube files to mseed files has been reduced. Thus, in several cases the conversion stops due to buffer overruns. This effect has been particularly observed when trying to convert more than about 20 consecutive days of cube files at once. In such a case, it is appropriate to set the function argument mseed_manual to `TRUE`. This will stop the function just at the point where the function would call the GIPPtools function cube2mseed. The user will see a confirmation command line in the R console, which asks to first copy all manually converted mseed files to the directory `mseed_raw` before confirming to continue with the R function. To convert all cube files to mseed files it is advised to open a terminal and run the function `GIPPtools/bin/cube2mseed` with the following parameters: `GIPPtools/bin/cube2mseed --verbose --output-dir=./mseed_raw/ ./input_dir/` without further adjustments, except for the fringe sample option, as specified in `aux_organisecubefiles`. Please also see the documentation of the cube2mseed program from the gipptools for further information.

Alternatively, increasing the heap space of the Java Runtime Environment, required for converting the cube files, can solve the above mentioned issue. To increase the heap space, use the argument `heapspace`. By default, this argument is set to 4096.

Value

A set of hourly seismic files written to disk.

Author(s)

Michael Dietze

Examples

```
## Not run:

## basic example with minimum effort
aux_organisecubefiles(stationfile = "output/stationinfo.txt",
                      input_dir = "input",
                      gipptools = "software/gipptools-2015.225/")


## End(Not run)
```

---

aux_psdpanels *Generate spectrogram panels for a seismic network*

Description

The function generates a set of spectrogram (PSD) panels on single to several hours basis. It depends on seismic files being organised in a coherent structure as, for example generated by `aux_organisecubefiles`. The function is similar to `aux_psdsummary` but arranges PSDs of all stations by time, rather than creating individual PSDs by time and station.

Usage

```
aux_psdpanels(
  station,
  component = "BHZ",
  period,
  span = 1,
  input_dir,
  output_dir,
  cpu,
  aggregate = c(1, 5),
  n_dates = 2000,
  jpg_dim = c(4444, 2500, 300, 90),
  verbose = FALSE,
  ...
)
```

Arguments

`station`	`Character` value, seismic station ID, which must correspond to the ID in the file name of the data directory structure (cf. `aux_organisecubefiles`).
`component`	`Character` value, seismic component, which must correspond to the component name in the file name of the data directory structure (cf. `aux_organisecubefiles`). Default is `"BHZ"` (vertical component).

`period`	`POSIXct` vector of length two, time period to be processed.
`span`	`Numeric` vector, time span per PSD in hours. Value can range between 1 and 24. For each time span a separate jpeg-file will be produced. Default is `1` hour.
`input_dir`	`Character` value, path to directory where the seismic files are stored.
`output_dir`	`Character` value, path to directory where PSD image files are saved to.
`cpu`	`Numeric` value, fraction of CPUs to use for parallel processing. If omitted, one CPU is used.
`aggregate`	`Numeric` vector of length two, aggregation factors for the processed PSD matrics. First entry denotes time aggregation, second entry frequency aggregation. Default is `c(1,5)`.
`n_dates`	`Numeric` value, final number of spectra per output PSD. Default is `2000`.
`jpg_dim`	`Numeric` vector of length four, JPEG image properties in the form `c(width,height,resolution,qualit` Default is `c(4444,2500,300,90)`.
`verbose`	`Logical` value, optional screen output of processing progress. Default is `FALSE`.
`...`	Additional arguments passed to different functions. See details section for default values.

Details

The function calls a series of other functions, partly with modified default values, which can be changed by the ...-argument. By default, the seismic files are imported as eseis objects using `aux_getevent(...,eseis = TRUE)`. The signals are deconvolved with `signal_deconvolve()` using the default options, i.e., `sensor = "TC120s"` and `logger = "Cube3extBOB"`. Then, the signals are bandpass filtered with `signal_filter`, using `f = c(1,90)`. The PSDs are calculated with `signal_spectrogram` using `Welch = TRUE`, `window = 30` and `window_sub = 15`.

This and all other aux-functions are primarily written for internal use in the GFZ Geomorphology Section group members and their usual data handling scheme. Thus, they may be of limited use when adopted for other scopes. However, many of these functions are internally consistent in usage.

Value

A set of JPEG images wirtten to disk

Author(s)

Michael Dietze

Examples

```
## Not run:

## PSD generation with minimum input arguments
aux_psdpanels(station = stations$ID,
              input_dir = "input/")
```

```
## End(Not run)
```

`aux_psdsummary` *Generate spectrograms for seismic stations at different time periods*

Description

The function generates a set of spectrograms (PSDs) for all seismic stations provided, for daily, weekly, monthly and total time periods. It depends on seismic files being organised in a coherent structure as, for example, generated by `aux_Organisecubefiles`.

Usage

```
aux_psdsummary(
  station,
  component = "BHZ",
  period,
  output = c("daily", "weekly", "monthly", "total"),
  input_dir,
  output_dir,
  aggregate = c(1, 5),
  n_dates = 2000,
  jpg_dim = c(4444, 2500, 300, 90),
  verbose = FALSE,
  ...
)
```

Arguments

`station`	`Character` value, seismic station ID, which must correspond to the ID in the file name of the data directory structure (cf. `aux_organisecubefiles()`).
`component`	`Character` value, seismic component, which must correspond to the component name in the file name of the data directory structure (cf. `aux_organisecubefiles()`). Default is `"BHZ"` (vertical component of a sac file).
`period`	`POSIXct` vector of length two, time period to be processed.
`output`	`Character` vector, output PSD types. One or more out of `"daily"`, `"weekly"`, `"monthly"`, `"total"`. Default is `c("daily","weekly","monthly","total")`.
`input_dir`	`Character` value, path to directory where the seismic files are stored.
`output_dir`	`Character` value, path to directory where PSD image files are saved to.
`aggregate`	`Numeric` vector of length two, aggregation factors for the processed PSD matrics. First entry denotes time aggregation, second entry frequency aggregation. Default is `c(1,5)`.
`n_dates`	`Numeric` value, final number of spectra per output PSD. Default is `2000`.

`jpg_dim`	`Numeric` vector of length four, JPEG image properties in the form `c(width,height,resolution,qualit` Default is `c(4444,2500,300,90)`.
`verbose`	`Logical` value, optional screen output of processing progress. Default is `FALSE`.
`...`	Additional arguments passed to different functions. See details section for default values.

Details

The function calls a series of other functions, partly with modified default values, which can be changed by the ...-argument. By default, the seismic files are imported as eseis objects using `aux_getevent(...,eseis = TRUE)`. The signals are deconvolved with `signal_deconvolve()` using the default options, i.e., `sensor = "TC120s"` and `logger = "Cube3extBOB"`. Then, the signals are bandpass filtered with `signal_filter`, using `f = c(1,90)`. The PSDs are calculated with `signal_spectrogram` using `Welch = TRUE`, `window = 90` and `window_sub = 30`.

This and all other aux-functions are primarily written for internal use amongst the GFZ Geomorphology Section group members and their usual data handling scheme. Thus, they may be of limited use when adopted for other scopes. However, many of these functions are internally consistent in usage.

Value

A set of JPEG images wirtten to disk

Author(s)

Michael Dietze

Examples

```
## Not run:

## PSD generation with minimum input arguments
aux_psdsummary(station = c("STA01", "STA02"),
               period = as.POSIXct(x = c("2017-04-01",
                                          "2017-04-03"),
                                    tz = "UTC"),
               input_dir = "~/data/seismic/sac/")

## PSD generation with some more arguments
aux_psdsummary(station = c("STA01", "STA02"),
               component = "BHZ",
               period = as.POSIXct(x = c("2017-04-01",
                                          "2017-04-03"),
                                    tz = "UTC"),
               output = c("daily", "monthly"),
               input_dir = "~/data/seismic/sac/",
               aggregate = c(2, 10),
               n_dates = 1000,
               jpg_dim = c(1600, 900, 300, 50),
                 verbose = TRUE)

## End(Not run)
```

`aux_stationinfofile` *Create station info file from cube files.*

Description

This function reads GPS tags from Omnirecs/Digos Datacube files and creates a station info file from additional input data. It depends on the cubetools or gipptools software package (see details).

Usage

```
aux_stationinfofile(
  name,
  input_dir,
  output_dir,
  station_ID,
  station_name,
  station_z,
  station_d,
  sensor_type,
  logger_type,
  sensor_ID,
  logger_ID,
  unit = "dd",
  n,
  quantile = 0.95,
  random = TRUE,
  cpu,
  gipptools,
  write_file = TRUE,
  write_raw = FALSE,
  write_data = FALSE
)
```

Arguments

`name`	`Character` value, file name of the output station info file, without extention (will be added as *.txt).
`input_dir`	`Character` value, path to directory where all cube files to be processed as stored. Each set of files from one logger must be stored in a separate sub-directory named after the cube ID.
`output_dir`	`Character` value, path to directory where output data is written to.

`station_ID`	`Character` vector, seismic station ID. Each value must not contain more than 5 characters. Longer entries will be clipped. If omitted, a default ID will be created.
`station_name`	`Character` vector, seismic station name. If omitted, the station ID is used as name.
`station_z`	`Numeric` vector, elevation of the seismic stations.
`station_d`	`Numeric` vector, deployment depth of the seismic sensor.
`sensor_type`	`Character` vector, sensor type.
`logger_type`	`Character` vector, logger type.
`sensor_ID`	`Character` vector, sensor ID.
`logger_ID`	`Character` vector, logger ID.
`unit`	`Character` value, coordinates unit of the location. One out of `"dd"` (decimal degrees) and `"utm"` (metric in UTM zone). Default is `"dd"`.
`n`	`Numeric` value, number of cube file to process for GPS coordinate extraction. If omitted, all files are processed.
`quantile`	`Numeric` value, quantile size to which the extracted coordinate sample size is restricted. This is mainly used to remove coordinate outliers, due to spurious GPS signals. Default is `0.95`. Set to `1` to omit any sample rejection.
`random`	`Logical` value, option to draw `n` cube files randomly instead of ordered by date. Default is `TRUE`.
`cpu`	`Numeric` value, fraction of CPUs to use for parallel processing. If omitted, one CPU is used.
`gipptools`	`Character` value, path to gipptools or cubetools directory.
`write_file`	`Logical` value, option to write station info file to disk. Default is `TRUE`.
`write_raw`	`Logical` value, option to write (keep) raw ASCII GPS data. Default is `FALSE`.
`write_data`	`Logical` value, option to write gps raw data as rda-file. File name will be the same as for `file`. Default is `FALSE`.

Details

A station info file is an ASCII file that contains all relevant information about the individual stations of a seismic network. The variables contain a station ID (containing not more than 5 characters), station name, latitude, longitude, elevation, deployment depth, sensor type, logger type, sensor ID and logger ID.
The function requires that the software cubetools (`http://www.omnirecs.de/documents.html`) or gipptools (`http://www.gfz-potsdam.de/en/section/geophysical-deep-sounding/infrastructure/geophysical`
are installed. Note that GPS tag extraction may take several minutes per cube file. Hence, depending on the number of files and utilised CPUs the processing may take a while.
Specifying an input directory (`input_dir`) is mandatory. This input directory must only contain the subdirectories with the cube files to process, each set of cube files must be located in a separate subdirectory and these subdiretories must have the same name as specified by the logger IDs (`logger_ID`). An appropriate structure would be something like:

1. input
   (a) A1A
      i. file1.A1A
      ii. file2.A1A
   (b) A1B
      i. file1.A1B
      ii. file2.A1B

Value

A set of files written to disk and a data frame with seismic station information.

Author(s)

Michael Dietze

Examples

```
## Not run:

## basic example with minimum effort
aux_stationinfofile(name = "stationinfo",
                    input_dir = "input",
                    logger_ID = c("864", "876", "AB1"),
                    gipptools = "software/gipptools-2015.225")

## example with more adjustments
aux_stationinfofile(name = "stationinfo",
                    input_dir = "input",
                    logger_ID = c("864", "876", "AB1"),
                    station_name = c("KTZ01", "KTZ02", "KTZ03"),
                    station_z = c(30, 28, 29),
                    station_d = rep(0.5, 3),
                    sensor_type = rep("TC120s", 3),
                    logger_type = rep("Cube3ext", 3),
                    unit = "utm",
                    n = 1,
                    cpu = 0.9,
                    gipptools = "software/gipptools-2015.225",
                    write_raw = TRUE,
                    write_data = TRUE)


## End(Not run)
```

earthquake *Seismic traces of a small earthquake*

Description

The dataset comprises the seismic signal (all three components) of a small earthquake. The data have been recorded at 200 Hz sampling frequency with an Omnirecs Cube ext 3 data logger.

The dataset comprises the time vector associated with the data set `earthquake`.

Usage

```
s

t
```

Format

The format is: List of 3 $ BHE: num [1:8001] -3.95e-07 ... $ BHN: num [1:8001] -2.02e-07 ... $ BHZ: num [1:8001] -1.65e-07 ...

Examples

```
## load example data set
data(earthquake)

## plot signal vector
plot(x = t, y = s$BHZ, type = "l")


## load example data set
data(earthquake)

## show range of time vector
range(t)
```

eseis *eseis: Environmental Seismology Toolbox*

Description

Environmental seismoloy is a scientific field that studies the seismic signals, emitted by Earth surface processes. This package eseis provides all relevant functions to read/write seismic data files, prepare, analyse and visualise seismic data, and generate reports of the processing history.

Details

Package:	eseis
Type:	Package
Version:	0.4.0
Date:	2018-04-25
License:	GPL-3

Author(s)

Michael Dietze

`fmi_inversion` *Invert fluvial data set based on reference spectra catalogue*

Description

The fluvial model inversion (FMI) routine uses a predefined look-up table with reference spectra to identify those spectra that fit the empirical data set best, and returns the corresponding target variables and fit errors.

Usage

```
fmi_inversion(reference, data, n_cores = 1)
```

Arguments

`reference`	`List` containing lists with precalculated model spectra.
`data`	`eseis` object or `numeric` matrix (spectra organised by columns), empiric spectra which are used to identify the best matching target parameters of the reference data set.
`n_cores`	`Numeric` value, number of CPU cores to use. Disabled by setting to 1. Default is 1.

Details

Note that the frequencies of the empiric and modelled data sets must match.

Value

`List` object containing the inversion results.

Author(s)

Michael Dietze

Examples

```
## NOTE THAT THE EXAMPLE IS OF BAD QUALITY BECAUSE ONLY 10 REFERENCE
## PARAMETER SETS AND SPECTRA ARE CALCULATED, DUE TO COMPUATATION TIME
## CONSTRAINTS. SET THE VALUE TO 1000 FOR MORE MEANINGFUL RESULTS.

## create 100 example reference parameter sets
ref_pars <- fmi_parameters(n = 10,
                           h_w = c(0.02, 1.20),
                           q_s = c(0.001, 8.000) / 2650,
                           d_s = 0.01,
                           s_s = 1.35,
                           r_s = 2650,
                           w_w = 6,
                           a_w = 0.0075,
                           f_min = 5,
                           f_max = 80,
                           r_0 = 6,
                           f_0 = 1,
                           q_0 = 10,
                           v_0 = 350,
                           p_0 = 0.55,
                           e_0 = 0.09,
                           n_0_a = 0.6,
                           n_0_b = 0.8,
                           res = 100)

## create corresponding reference spectra
ref_spectra <- fmi_spectra(parameters = ref_pars)

## define water level and bedload flux time series
h <- c(0.01, 1.00, 0.84, 0.60, 0.43, 0.32, 0.24, 0.18, 0.14, 0.11)
q <- c(0.05, 5.00, 4.18, 3.01, 2.16, 1.58, 1.18, 0.89, 0.69, 0.54) / 2650
hq <- as.list(as.data.frame(rbind(h, q)))

## calculate synthetic spectrogram
psd <- do.call(cbind, lapply(hq, function(hq) {

  psd_turbulence <- eseis::model_turbulence(h_w = hq[1],
                                            d_s = 0.01,
                                            s_s = 1.35,
                                            r_s = 2650,
                                            w_w = 6,
                                            a_w = 0.0075,
                                            f = c(10, 70),
                                            r_0 = 5.5,
                                            f_0 = 1,
                                            q_0 = 18,
                                            v_0 = 450,
                                            p_0 = 0.34,
                                            e_0 = 0.0,
                                            n_0 = c(0.5, 0.8),
                                            res = 100,
                                            eseis = FALSE)$spectrum

      psd_bedload <- eseis::model_bedload(h_w = hq[1],
                                      q_s = hq[2],
                                      d_s = 0.01,
                                      s_s = 1.35,
                                      r_s = 2650,
                                      w_w = 6,
                                      a_w = 0.0075,
                                      f = c(10, 70),
                                      r_0 = 5.5,
                                      f_0 = 1,
                                      q_0 = 18,
                                      v_0 = 450,
                                      x_0 = 0.34,
                                      e_0 = 0.0,
                                      n_0 = 0.5,
                                      res = 100,
                                      eseis = FALSE)$spectrum

      ## combine spectra
      psd_sum <- psd_turbulence + psd_bedload

      ## return output
      return(10 * log10(psd_sum))
    }))

    graphics::image(t(psd))

    ## invert empiric data set
    X <- fmi_inversion(reference = ref_spectra,
                   data = psd)

    ## plot model results
    plot(X$parameters$q_s * 2650,
         type = "l")
    plot(X$parameters$h_w,
         type = "l")
```

fmi_parameters — *Create reference model reference parameter catalogue*

Description

In order to run the fluvial model inversion (FMI) routine, a set of randomised target parameter combinations needs to be created. This function does this job.

Usage

```
fmi_parameters(
  n,
  d_s,
  s_s,
  r_s,
  q_s,
  h_w,
  w_w,
  a_w,
  f_min,
  f_max,
  r_0,
  f_0,
  q_0,
  v_0,
  p_0,
  e_0,
  n_0_a,
  n_0_b,
  res
)
```

Arguments

n	Numeric value, number of output reference spectra.
d_s	Numeric value, mean sediment grain diameter (m). Alternative to gsd.
s_s	Numeric value, standard deviation of sediment grain diameter (m). Alternative to gsd.
r_s	Numeric value, specific sediment density (kg / m^3)
q_s	Numeric value, unit sediment flux (m^2 / s)
h_w	Numeric value, fluid flow depth (m)
w_w	Numeric value, fluid flow width (m)
a_w	Numeric value, fluid flow inclination angle (radians)
f_min	Numeric value, lower boundary of the frequency range to be modelled.
f_max	Numeric value, upper boundary of the frequency range to be modelled.
r_0	Numeric value, distance of seismic station to source
f_0	Numeric value, reference frequency (Hz)
q_0	Numeric value, ground quality factor at f_0. "Reasonable value may be 20" (Tsai et al. 2012).
v_0	Numeric value, phase speed of the Rayleigh wave at f_0 (m/s). Assuming a shear wave velocity of about 2200 m/s, Tsai et al. (2012) yield a value of 1295 m/s for this parameter.

p_0	Numeric value, variation exponent of Rayleigh wave velocities with frequency (dimensionless)
e_0	Numeric value, exponent characterizing quality factor increase with frequency (dimensionless). "Reasonable value may be 0" (Tsai et al. 2012).
n_0_a	Numeric value, lower Greens function displacement amplitude coefficients. Cf. N_ij in eq. 36 in Gimbert et al. (2014)
n_0_b	Numeric value, lower Greens function displacement amplitude coefficients. Cf. N_ij in eq. 36 in Gimbert et al. (2014)
res	Numeric value, output resolution, i.e. length of the spectrum vector.

Details

All parameters must be provided as single values, except for those parameters that shall be randomised, which must be provided as a vector of length two. This vector defines the range within which uniformly distributed random values will be generated and assigned.

Value

List object with model reference parameters.

Author(s)

Michael Dietze

Examples

```
## create two parameter sets where h_w (water level) and q_s (sediment
## flux) are randomly varied.

ref_pars <- fmi_parameters(n = 2,
                           h_w = c(0.02, 2.00),
                           q_s = c(0.001, 50.000) / 2650,
                           d_s = 0.01,
                           s_s = 1.35,
                           r_s = 2650,
                           w_w = 6,
                           a_w = 0.0075,
                           f_min = 5,
                           f_max = 80,
                           r_0 = 6,
                           f_0 = 1,
                           q_0 = 10,
                           v_0 = 350,
                           p_0 = 0.55,
                           e_0 = 0.09,
                           n_0_a = 0.6,
                           n_0_b = 0.8,
                           res = 100)
```

fmi_spectra *Create reference model spectra catalogue*

Description

In order to run the fluvial model inversion (FMI) routine, a look-up table with reference spectra needs to be created (based on predefined model parameters). This function does this job.

Usage

```
fmi_spectra(parameters, n_cores = 1)
```

Arguments

parameters	`List` containing lists with model parameters for which the spectra shall be calculated.
n_cores	`Numeric` value, number of CPU cores to use. Disabled by setting to 1. Default is 1.

Value

`List` object containing the calculated reference spectra and the corresponding input parameters.

Author(s)

Michael Dietze

Examples

```
## create 2 example reference parameter sets
ref_pars <- fmi_parameters(n = 2,
                           h_w = c(0.02, 2.00),
                           q_s = c(0.001, 50.000) / 2650,
                           d_s = 0.01,
                           s_s = 1.35,
                           r_s = 2650,
                           w_w = 6,
                           a_w = 0.0075,
                           f_min = 5,
                           f_max = 80,
                           r_0 = 6,
                           f_0 = 1,
                           q_0 = 10,
                           v_0 = 350,
                           p_0 = 0.55,
                           e_0 = 0.09,
                           n_0_a = 0.6,
                           n_0_b = 0.8,
                              res = 100)

## create corresponding reference spectra
ref_spectra <- fmi_spectra(parameters = ref_pars)
```

gui_models *Start GUI with seismic models*

Description

This function starts a browser-based graphic user interface to explore the parameter space of seismic models that predict the spectra of turbulent water flow and bedload flux.

Usage

```
gui_models(...)
```

Arguments

`...`	further arguments to pass to `runApp`

Author(s)

Michael Dietze

See Also

`runApp`

Examples

```
## Not run:
# Start the GUI
gui_models()

## End(Not run)
```

`list_logger` *List library with data logger information.*

Description

The function returns the list of supported data loggers to extract signal deconvolution parameters.

Usage

```
list_logger()
```

Details

The value AD is the analogue-digital conversion factor.

Value

`List` object, supported loggers with their parameters.

Author(s)

Michael Dietze

Examples

```
## show documented loggers
list_logger()

## show names of loggers in list
names(list_logger())
```

`list_sacparameters` *List all header parameters of a sac file.*

Description

The function returns a data frame with all header values of a sac file. It may be used for advanced modifications by the user.

Usage

```
list_sacparameters()
```

Value

`List` object, parameters supported by a sac file.

Author(s)

Michael Dietze

Examples

```
## show sac parameters
list_sacparameters()
```

`list_sensor` *List sensor library.*

Description

The function returns the list of supported sensors to extract signal deconvolution parameters.

Usage

```
list_sensor()
```

Details

Poles and zeros must be given in rad/s. Characteristics of further sensors can be added manually. See examples of `signal_deconvolve` for further information. The value s is the generator constant (sensitivity) given in Vs/m. The value k is the normalisation factor of the sensor.

Value

`List` object, supported sensors with their parameters.

Author(s)

Michael Dietze

Examples

```
## show sensors
list_sensor()
```

`model_bedload` *Model the seismic spectrum due to bedload transport in rivers*

Description

The function calculates a seismic spectrum as predicted by the model of Tsai et al. (2012) for river bedload transport. The code was written to R by Sophie Lagarde and integrated to the R package 'eseis' by Michael Dietze.

Usage

```
model_bedload(
  gsd,
  d_s,
  s_s,
  r_s,
  q_s,
  h_w,
  w_w,
  a_w,
  f = c(1, 100),
  r_0,
  f_0,
  q_0,
  e_0,
  v_0,
  x_0,
  n_0,
  n_c,
  res = 100,
  eseis = FALSE,
  ...
)
```

Arguments

`gsd`	`data frame` grain-size distribution function. Must be provided as data frame with two variables: grain-size class (first column) and vol- or wgt.-percentage per class (second column). See examples for details.
`d_s`	`Numeric` value, mean sediment grain diameter (m). Alternative to `gsd`.
`s_s`	`Numeric` value, standard deviation of sediment grain diameter (m). Alternative to `gsd`.
`r_s`	`Numeric` value, specific sediment density (kg / m^3)
`q_s`	`Numeric` value, unit sediment flux (m^2 / s)
`h_w`	`Numeric` value, fluid flow depth (m)

`w_w`	`Numeric` value, fluid flow width (m)
`a_w`	`Numeric` value, fluid flow inclination angle (radians)
`f`	`Numeric` vector, frequency range to be modelled. If of length two the argument is interpreted as representing the lower and upper limit and the final length of the frequency vector is set by the argument `res`. If `f` contains more than two values it is interpreted as the actual frequency vector and the value of `res` is ignored.
`r_0`	`Numeric` value, distance of seismic station to source
`f_0`	`Numeric` value, reference frequency (Hz)
`q_0`	`Numeric` value, ground quality factor at `f_0`. "Reasonable value may be `20`" (Tsai et al. 2012).
`e_0`	`Numeric` value, exponent characterizing quality factor increase with frequency (dimensionless). "Reasonable value may be `0`" (Tsai et al. 2012).
`v_0`	`Numeric` value, phase speed of the Rayleigh wave at `f_0` (m/s). Assuming a shear wave velocity of about 2200 m/s, Tsai et al. (2012) yield a value of 1295 m/s for this parameter.
`x_0`	`Numeric` value, exponent of the power law variation of Rayleigh wave velocities with frequency
`n_0`	`Numeric` vector of length two, Greens function displacement amplitude coefficients. Cf. N_ij in eq. 36 in Gimbert et al. (2014)
`n_c`	`Numeric` value, option to include single particle hops coherent in time, causing spectrum modulation due to secondary effects. Omitted is no value is specified, here. Usual values may be between 2 and 4.
`res`	`Numeric` value, output resolution, i.e. length of the spectrum vector. Default is 1000.
`eseis`	`Character` value, option to return an eseis object instead of a data frame. Default is `FALSE`.
`...`	Further arguments passed to the function.

Details

The model uses a set of predefined constants. These can also be changed by the user, using the `...` argument:

- `g = 9.81`, gravitational acceleration (m/s^2)
- `r_w = 1000`, fluid specific density (kg/m^3)
- `k_s = 3 * d_50`, roughness length (m)
- `log_lim = c(0.0001,100),limits of grain-size distribution function template (m)`
- `log_length = 10000,length of grain-size distribution function template`
- `nu = 10^(-6)`, specific density of the fluid (kg/m^3)
- `power_d = 3`, grain-size power exponent
- `gamma = 0.9`, gamma parameter, after Parker (1990)
- `s_c = 0.8`, drag coefficient parameter

- s_p = 3.5, drag coefficient parameter
- c_1 = 2 / 3, inter-impact time scaling, after Sklar & Dietrich (2004)

When no user defined grain-size distribution function is provided,the function calculates the raised cosine distribution function as defined in Tsai et al. (2012) using the default range and resolution as specified by log_lim and log_length (see additional arguments list above). These default values are appropriate for mean sediment sizes between 0.001 and 10 m and log standard deivations between 0.05 and 1. When more extreme distributions are to be used, it is necessary to either adjust the arguments log_lim and log_length or use a user defined distribution function.

Value

eseis object containing the modelled spectrum.

Author(s)

Sophie Lagarde, Michael Dietze

References

Tsai, V. C., B. Minchew, M. P. Lamb, and J.-P. Ampuero (2012), A physical model for seismic noise generation from sediment transport in rivers, Geophys. Res. Lett., 39, L02404, doi:10.1029/2011GL050255.

Examples

```
## calculate spectrum (i.e., fig. 1b in Tsai et al., 2012)
p_bedload <- model_bedload(d_s = 0.7,
                           s_s = 0.1,
                           r_s = 2650,
                           q_s = 0.001,
                           h_w = 4,
                           w_w = 50,
                           a_w = 0.005,
                           f = c(0.1, 20),
                           r_0 = 600,
                           f_0 = 1,
                           q_0 = 20,
                           e_0 = 0,
                           v_0 = 1295,
                           x_0 = 0.374,
                           n_0 = 1,
                           res = 100,
                           eseis = TRUE)

## plot spectrum
plot_spectrum(data = p_bedload,
              ylim = c(-170, -110))

## define empiric grain-size distribution
gsd_empiric <- data.frame(d = c(0.70, 0.82, 0.94, 1.06, 1.18, 1.30),
                          p = c(0.02, 0.25, 0.45, 0.23, 0.04, 0.00))
```

```
## calculate spectrum
p_bedload <- model_bedload(gsd = gsd_empiric,
                           r_s = 2650,
                           q_s = 0.001,
                           h_w = 4,
                           w_w = 50,
                           a_w = 0.005,
                           f = c(0.1, 20),
                           r_0 = 600,
                           f_0 = 1,
                           q_0 = 20,
                           e_0 = 0,
                           v_0 = 1295,
                           x_0 = 0.374,
                           n_0 = 1,
                           res = 100,
                           eseis = TRUE)

## plot spectrum
plot_spectrum(data = p_bedload,
              ylim = c(-170, -110))

## define mean and sigma for parametric distribution function
d_50 <- 1
sigma <- 0.1

## define raised cosine distribution function following Tsai et al. (2012)
d_1 <- 10^seq(log10(d_50 - 5 * sigma),
              log10(d_50 + 5 * sigma),
              length.out = 20)

sigma_star <- sigma / sqrt(1 / 3 - 2 / pi^2)

p_1 <- (1 / (2 * sigma_star) *
          (1 + cos(pi * (log(d_1) - log(d_50)) / sigma_star))) / d_1
p_1[log(d_1) - log(d_50) > sigma_star] <- 0
p_1[log(d_1) - log(d_50) < -sigma_star] <- 0
p_1 <- p_1 / sum(p_1)

gsd_raised_cos <- data.frame(d = d_1,
                             p = p_1)
```

model_turbulence *Model the seismic spectrum due to hydraulic turbulence*

Description

The function calculates the seismic spectrum as predicted by the model of Gimbert et al. (2014) for hydraulic turbulence. The code was written to R by Sophie Lagarde and integrated to the R package 'eseis' by Michael Dietze.

Usage

```
model_turbulence(
  d_s,
  s_s,
  r_s = 2650,
  h_w,
  w_w,
  a_w,
  f = c(1, 100),
  r_0,
  f_0,
  q_0,
  v_0,
  p_0,
  n_0,
  res = 1000,
  eseis = FALSE,
  ...
)
```

Arguments

`d_s`	`Numeric` value, mean sediment grain diameter (m)
`s_s`	`Numeric` value, standard deviation of sediment grain diameter (m)
`r_s`	`Numeric` value, specific sediment density (kg / m^3)
`h_w`	`Numeric` value, fluid flow depth (m)
`w_w`	`Numeric` value, fluid flow width (m)
`a_w`	`Numeric` value, fluid flow inclination angle (radians)
`f`	`Numeric` vector, frequency range to be modelled. If of length two the argument is interpreted as representing the lower and upper limit and the final length of the frequency vector is set by the argument `res`. If `f` contains more than two values it is interpreted as the actual frequency vector and the value of `res` is ignored.
`r_0`	`Numeric` value, distance of seismic station to source
`f_0`	`Numeric` value, reference frequency (Hz)
`q_0`	`Numeric` value, ground quality factor at `f_0`
`v_0`	`Numeric` value, phase speed of the Rayleigh wave at `f_0` (m/s)
`p_0`	`Numeric` value, variation exponent of Rayleigh wave velocities with frequency (dimensionless)
`n_0`	`Numeric` vector of length two, Greens function displacement amplitude coefficients. Cf. N_ij in eq. 36 in Gimbert et al. (2014)
`res`	`Numeric` value, output resolution, i.e. length of the spectrum vector. Default is 1000.

eseis	Character value, option to return an eseis object instead of a data frame. Default is FALSE.
...	Further arguments passed to the function.

Details

The model uses a set of predefined constants. These can also be changed by the user, using the ... argument:

- c = 0.5, instantaneous fluid-grain friction coefficient (dimensionless)
- g = 9.81, gravitational acceleration (m/s^2)
- k = 0.5, Kolmogrov constant (dimensionless)
- k_s = 3 * d_s, roughness length (m)
- h = k_s / 2, reference height of the measurement (m)
- e_0 = 0, exponent of Q increase with frequency (dimensionless)
- r_w = 1000, specific density of the fluid (kg/m^3)
- c_w = 0.5, instantaneous fluid-grain friction coefficient (dimensionless)

Value

eseis object containing the modelled spectrum.

Author(s)

Sophie Lagarde, Michael Dietze

Examples

```
## model the turbulence-related power spectrum
P <- model_turbulence(d_s = 0.03, # 3 cm mean grain-size
                      s_s = 1.35, # 1.35 log standard deviation
                      r_s = 2650, # 2.65 g/cm^3 sediment density
                      h_w = 0.8, # 80 cm water level
                      w_w = 40, # 40 m river width
                      a_w = 0.0075, # 0.0075 rad river inclination
                      f = c(1, 200), # 1-200 Hz frequency range
                      r_0 = 10, # 10 m distance to the river
                      f_0 = 1, # 1 Hz Null frequency
                      q_0 = 10, # 10 quality factor at f = 1 Hz
                      v_0 = 2175, # 2175 m/s phase velocity
                      p_0 = 0.48, # 0.48 power law variation coefficient
                      n_0 = c(0.6, 0.8), # Greens function estimates
                      res = 1000) # 1000 values build the output resolution

## plot the power spectrum
plot_spectrum(data = P)
```

plot_components *Plot three seismic components against each other*

Description

The function visualises the time evolution of three seismic components of the same signal against each other as line graphs. There are three different visualisation types available: `2D` (a panel of three 2D plots), `3D` (a perspective threedimensional plot) and `scene` (an interactive threedimensional plot, mainly for exploratory purpose).

Usage

```
plot_components(data, type = "2D", order = "xyz", ...)
```

Arguments

`data`	`List`, `data frame` or `matrix`, seismic componenents to be processed. If `data` is a matrix, the components must be organised as columns. Also, `data` can be a list of `eseis` objects.
`type`	`Character` value, plot type. One out of `"2D"` (panel of three 2-dimensional plots), `"3D"` (perspective 3D plot) and `"scene"` (interactive 3D plot). Default is `"2D"`.
`order`	`Caracter` value, order of the seismic components. Describtion must contain the letters `"x"`,`"y"` and `"z"` in the order according to the input data set. Default is `"xyz"` (NW-SE-vertical).
`...`	Further arguments passed to the plot function.

Details

The plot type `type = "3D"` requires the package `plot3D` being installed. The plot type `type = "scene"` requires the package `rgl` being installed.

Value

A plot

Author(s)

Michael Dietze

Examples

```
## load example data set
data(earthquake)

## filter seismic signals
```

```
s <- eseis::signal_filter(data = s,
                          dt = 1/200,
                          f = c(0.05, 0.1))

## integrate signals to get displacement
s_d <- eseis::signal_integrate(data = s, dt = 1/200)

## plot components in 2D
plot_components(data = s_d,
                type = "2D")

## plot components with time colour-coded
plot_components(data = s_d,
                type = "2D",
                col = rainbow(n = length(s$BHE)))

## plot components with used defined coulour ramp
col_user <- colorRampPalette(colors = c("grey20", "darkblue", "blue",
                                        "green", "red", "orange"))

plot_components(data = s_d,
                type = "2D",
                col = col_user(n = length(s$BHE)))

## plot components as 3D plot, uncomment to use
#plot_components(data = s_d,
#                 type = "3D",
#                 col = rainbow(n = length(s$BHE)))
```

plot_ppsd *Plot a probabilistic power spectral density estimate (PPSD)*

Description

The function uses the output of `signal_spectrogram()` to plot a probabilistic power spectral density estimate.

Usage

```
plot_ppsd(data, res = c(500, 500), n, ...)
```

Arguments

data	`List` object, spectrogram to be plotted. Must be output of `signal_spectrogram()` or of equivalent structure.
res	`Integer` vector of length two, factors of image resolution in pixels, i.e. in time and frequency dimension. Default is `c(100,100)`.

n	Integer vector of length two, factors by which the image will be smoothend by a running average. n sets the filter window size, in x and y direction, respectively. By default, the window sizes are set to one percent of the input data set dimension.
...	Additional arguments passed to the plot function.

Value

Graphic output of a spectrogram.

Author(s)

Michael Dietze

See Also

signal_spectrogram

Examples

```
## load example data set
data(rockfall)

## deconvolve data set
r <- signal_deconvolve(data = rockfall_eseis)

## calculate PSD
p <- signal_spectrogram(data = r)

## plot PPSD
plot_ppsd(data = p$PSD)

## plot PPSD with lower resolution, more smoothing and other colour
ppsd_color <- colorRampPalette(c("white", "black", "red"))

plot_ppsd(data = p$PSD,
          res = c(200, 200),
          n = c(15, 20),
          col = ppsd_color(200))
```

plot_signal *Plot a seismic signal*

Description

This function plots a line graph of a seismic signal. To avoid long plot preparation times the signal is reduced to a given number of points.

Usage

```
plot_signal(data, time, n = 10000, ...)
```

Arguments

`data`	`eseis` object or `numeric` vector, data set to be plotted.
`time`	`POSIXct` vector, corresponding time vector.
`n`	`Numeric` value, number of values to which the dataset is reduced. Default is `10000`.
`...`	Further arguments passed to the plot function.

Details

The `format` argument is based on hints provided by Sebastian Kreutzer and Christoph Burow. It allows plotting time axis units in user defined formats. The time format must be provided as character string using the POSIX standard (see documentation of `strptime` for a list of available keywords), e.g., " "

Value

A line plot of a seismic wave form.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## plot data set straightforward
plot_signal(data = rockfall_eseis)

## plot data set with lower resolution
plot_signal(data = rockfall_eseis, n = 100)

## plot data set but not as an eseis object
plot_signal(data = rockfall_z, time = rockfall_t)

## load earthquake data set
data(earthquake)

## plot all three components (after changing plot options)
pars <- par(no.readonly = TRUE)
par(mfcol = c(3, 1))

plt <- lapply(s, plot_signal, t = t)
```

```
par(pars)
```

`plot_spectrogram` *Plot spectrograms (power spectral density estimates)*

Description

This function plots spectrograms of seismic signals. It uses the output of `signal_spectrogram`.

Usage

```
plot_spectrogram(data, legend = FALSE, keep_par = FALSE, agg = c(1, 1), ...)
```

Arguments

`data`	`List` object, spectrogram to be plotted. Must be output of `signal_spectrogram` or of equivalent structure.
`legend`	`Logical` value, option to add colour bar legend. Legend label can be changed by `zlab`.
`keep_par`	`Logical` value, option to omit resetting plot parameters after function execution. Useful for adding further data to the PSD plot. Default is `FALSE` (parameters are reset to original values).
`agg`	`Integer` vector of length two, factors of image aggregation, i.e. in time and frequency dimension. Useful to decrease image size. Default is `c(1,1)` (no aggregation).
`...`	Additional arguments passed to the plot function.

Value

Graphic output of a spectrogram.

Author(s)

Michael Dietze

See Also

`signal_spectrogram`

Examples

```
## load example data set
data(rockfall)

## deconvolve signal
rockfall <- signal_deconvolve(data = rockfall_eseis)

## calculate spectrogram
PSD <- signal_spectrogram(data = rockfall)

## plot spectrogram
plot_spectrogram(data = PSD)

## plot spectrogram with legend and labels in rainbow colours
plot_spectrogram(data = PSD,
                 xlab = "Time (min)",
                 ylab = "f (Hz)",
                 main = "Power spectral density estimate",
                 legend = TRUE,
                 zlim = c(-220, -70),
                 col = rainbow(100))
```

---

plot_spectrum *Plot a spectrum of a seismic signal*

Description

This function plots a line graph of the spectrum of a seismic signal.

Usage

```
plot_spectrum(data, unit = "dB", n = 10000, ...)
```

Arguments

`data`	`eseis` object or `data frame` with two elements, `frequency` vector and `spectrum` vector.
`unit`	`Character` value. One out of `"linear"`, `"log"`, `"dB"`. Default is `"dB"`.
`n`	`Numeric` value, number of values to which the dataset is reduced. Default is `10000`.
`...`	Further arguments passed to the plot function.

Value

A line plot.

Author(s)

Michael Dietze

See Also

signal_spectrum

Examples

```
## load example data set
data(rockfall)

## calculate spectrum
spectrum_rockfall <- signal_spectrum(data = rockfall_eseis)

## plot data set with lower resolution
plot_spectrum(data = spectrum_rockfall)
```

---

read_mseed *Read mseed files.*

Description

This function reads mseed files. If `append = TRUE`, all files will be appended to the first one in the order as they are provided. In the append-case the function returns a either a list with the elements `signal`, `time`, `meta` and `header` or a list of the class `eseis` (see documentation of `aux_initiateeseis()`). If `append = FALSE` and more than one file is provided, the function returns a list of the length of the input files, each containing the above elements.

The mseed data format is read using the function `readMiniseedFile` from the package `IRISSeismic`.

Usage

```
read_mseed(
  file,
  append = TRUE,
  signal = TRUE,
  time = TRUE,
  meta = TRUE,
  header = TRUE,
  eseis = TRUE,
  type = "waveform"
)
```

Arguments

`file`	`Character` vector, input file name(s), with extension.
`append`	`Logical` value, option to append single files to one continuous file, keeping only the hedaer information of the first file, default is `TRUE`.
`signal`	`Logical` value, option to import the signal vector, default is `TRUE`.
`time`	`Logical` value, option to create the time vector. The timezone is automatically set to `"UTC"`, default is `TRUE`.
`meta`	`Logical` value, option to append the meta data part, default is `TRUE`.
`header`	`Logical` value, option to append the header part, default is `TRUE`.
`eseis`	`Logical` value, option to read data to an `eseis` object (recommended, see documentation of `aux_initiateeseis`), default is `TRUE`
`type`	`Character` value, type keyword of the data. One out of `"waveform"`, `"envelope"`, `"fft"`, `"spectrum"`, `"spectrogram"`, `"other"`, `hilbert`, `hvratio`. Default is `"waveform"`.

Value

`List` object, optionally of class `eseis`

Author(s)

Michael Dietze

Examples

```
## Not run:
## read mseed file with default options
x <- read_mseed(file = "input.miniseed")

## read mseed file, only signal trace, not as eseis object
x <- read_mseed(file = "input.miniseed",
                time = FALSE,
                meta = FALSE,
                header = FALSE,
                eseis = FALSE)

## read more than one mseed files and append traces
x <- read_mseed(file = c("input_1.miniseed", "input_2.miniseed"))

## End(Not run)
```

read_sac — *Read sac files.*

Description

This function reads sac files.

Usage

```
read_sac(
  file,
  append = TRUE,
  signal = TRUE,
  time = TRUE,
  meta = TRUE,
  header = TRUE,
  eseis = TRUE,
  get_instrumentdata = FALSE,
  endianness = "little",
  biglong = FALSE,
  type = "waveform"
)
```

Arguments

`file`	`Character` vector, input file name(s), with extension.
`append`	`Logical` value, option append single files to one continuous file, keeping only the header information of the first file, default is `TRUE`.
`signal`	`Logical` value, option to import the signal vector, default is `TRUE`.
`time`	`Logical` value, option to create the time vector. The timezone is automatically set to `"UTC"`, default is `TRUE`.
`meta`	`Logical` value, option to append the meta data part, default is `TRUE`.
`header`	`Logical` value, option to append the header part, default is `TRUE`.
`eseis`	`Logical` value, option to read data to an `eseis` object (recommended, see documentation of `aux_initiateeseis`), default is `TRUE`
`get_instrumentdata`	`Logical` value, option to fill meta information (sensor name, logger name, logger gain) from SAC user fields (field 127-129, KUSER0-KUSER2). Default is `FALSE`.
`endianness`	`Logical` value, endianness of the sac file. One out of `"little"`, `"big"` and `"swap"`. Default is `"little"`.
`biglong`	`Logical` value, number coding format. Default is `FALSE`.
`type`	`Character` value, type keyword of the data. One out of `"waveform"`, `"envelope"`, `"fft"`, `"spectrum"`, `"spectrogram"`, `"other"`, `hilbert`, `hvratio`. Default is `"waveform"`.

Details

The function reads one or more sac-files. If append = TRUE, all files will be appended to the first one in the order as they are provided. In the append-case the function returns a either a list with the elements signal, time, meta and header or a list of the class eseis (see documentation of aux_initiateeseis). If append = FALSE and more than one file is provided, the function returns a list of the length of the input files, each containing the above elements.

The sac data format is implemented as descibed on the IRIS website (https://ds.iris.edu/files/sac-manual/manual/file_format.html).

Value

List object, optionally of class eseis.

Author(s)

Michael Dietze

Examples

```
## Not run:
## read one file
file1 <- "~/Data/sac/EXMP01.14.213.01.00.00.BHE.SAC"

sac1 <- read_sac(file = file1)

## read two (or more files) without meta and header parts
file2 <- c("~/Data/sac/EXMP01.14.213.01.00.00.BHE.SAC",
           "~/Data/sac/EXMP01.14.213.02.00.00.BHE.SAC")

sac2 <- read_sac(file = file2,
                 meta = FALSE,
                 header = FALSE,
                 eseis = FALSE)

## End(Not run)
```

---

rockfall *Seismic trace of a rockfall event.*

Description

The dataset comprises the seismic signal (vertical component) of a rockfall event, preceeded by an earthquake. The data have been recorded at 200 Hz sampling frequency with an Omnirecs Cube ext 3 data logger.

The dataset comprises the time vector corresponding the to seismic signal of the rockfall event from the example data set "rockfall".

The dataset comprises the seismic signal (vertical component) of a rockfall event, preceeded by an earthquake. The data have been recorded at 200 Hz sampling frequency with an Omnirecs Cube ext 3 data logger.

Usage

```
rockfall_z

rockfall_t

rockfall_eseis
```

Format

The format is: num [1:98400] 65158 65176 65206 65194 65155 ...

Examples

```

## load example data set
data(rockfall)

## plot signal vector using base functionality
plot(x = rockfall_t, y = rockfall_z, type = "l")

## plot signal vector using the package plot function
plot_signal(data = rockfall_z, time = rockfall_t)


## load example data set
data(rockfall)


## load example data set
data(rockfall)
```

`signal_aggregate` *Aggregate a signal vector*

Description

The signal vector `data` is aggregated by an integer factor `n`. If an `eseis` object is provided, the meta data is updated. The function is a wrapper for the funcion `decimate` of the package `signal`.

Usage

```
signal_aggregate(data, n = 2, type = "iir")
```

Arguments

data	`eseis` object, `numeric` vector or list of objects, data set to be processed.
n	`Numeric` value, number of samples to be aggregated to one new data value. Must be an integer value greater than 1. Default is `2`.
type	`Character` value, filter type used for aggregation. For details see documentation of `signal::decimate`. Default is `"iir"`.

Value

Aggregated data set.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## aggregate signal by factor 4 (i.e., dt goes from 1/200 to 1/50)
rockfall_agg <- signal_aggregate(data = rockfall_z,
                                 n = 4)

## create example data set
s <- 1:10

## aggregate x by factor 2
s_agg_2 <- signal_aggregate(data = s,
                            n = 2)

## aggregate x by factor 3
s_agg_3 <- signal_aggregate(data = s,
                            n = 3)

## plot results
plot(x = s,
     y = rep(x = 1, times = length(s)),
     ylim = c(1, 3))

points(x = s_agg_2,
       y = rep(x = 2, times = length(s_agg_2)),
       col = 2)

points(x = s_agg_3,
       y = rep(x = 3, times = length(s_agg_3)),
       col = 3)

abline(v = s_agg_2,
       col = 2)
```

```
abline(v = s_agg_3,
       col = 3)

## create signal matrix
X <- rbind(1:100, 1001:1100, 10001:10100)

## aggregate signal matrix by factor 4
X_agg <- signal_aggregate(data = X,
n = 4)
```

signal_clip   *Clip signal based on time vector.*

Description

The function clips a seismic signal based on the corresponding time vector.

Usage

```
signal_clip(data, time, limits)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`time`	`POSIXct` vector, corresponding time vector. Only needed if `data` is no `eseis` object.
`limits`	`POSIXct` vector of length two, time limits for clipping.

Value

`Numeric` data set clipped to provided time interval.

Author(s)

Michael Dietze

Examples

```
## load example data
data(rockfall)

## define limits (second 10 to 20 of the signal)
limits <- c(rockfall_t[1] + 10, rockfall_t[1] + 20)

## clip signal
rockfall_clip <- signal_clip(data = rockfall_z,
                              time = rockfall_t,
                              limits = limits)

## clip signal using the eseis object
rockfall_clip <- signal_clip(data = rockfall_eseis,
                             limits = limits)
```

signal_cut *Cut signal amplitude at standard deviation-defined level.*

Description

This function cuts the amplitude of signal parts that exceede a user defined threshold set by k times the standard deviation of the signal.

Usage

```
signal_cut(data, k = 1)
```

Arguments

data	`eseis` object, `numeric` vector or list of objects, data set to be processed.
k	`Numeric` value, multiplier of the standard deviation threshold used to cut the signal amplitude. Default is `1` (1 sd).

Value

`Numeric` vector or list of vectors, cut signal.

Author(s)

Michael Dietze

Examples

```
## load example data
data(rockfall)

## cut signal
rockfall_cut <- signal_cut(data = rockfall_eseis)
```

signal_deconvolve *Deconvolve a signal vector.*

Description

The function removes the instrument response from a signal vector.

Usage

```
signal_deconvolve(
  data,
  sensor = "TC120s",
  logger = "Cube3BOB",
  gain = 1,
  use_metadata = FALSE,
  dt,
  p = 10^-6,
  waterlevel = 10^-6,
  na.replace = FALSE
)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`sensor`	`Character` value or `list` object, seismic sensor name. Must be present in the sensor library (`list_sensor`) or parameters must be added manually (see examples). Default is `"TC120s"`.
`logger`	`Character` value, seismic logger name. Must be present in the logger library (`list_logger`) or parameters must be added manually. Default is `"Cube3extBOB"`.
`gain`	`Numeric` value, signal gain level of the logger. Default is `1`.
`use_metadata`	`Logical` value, option to take keywords for `sensor`, `logger` and `gain` from eseis object meta data element instead of using explicitly provided arguments. Default is `FALSE`.
`dt`	`Numeric` value, sampling rate. Only needed if `data` is not an `eseis` object
`p`	`Numeric` value, proportion of signal to be tapered. Default is`10^-6`.
`waterlevel`	`Numeric` value, waterlevel value for frequency division, default is `10^-6`.
`na.replace`	`Logical` value, option to replace NA values in the data set by zeros. Default is `FALSE`. Attention, the zeros will create artifacts in the deconvolved data set. However, `NA` values will result in no deconvolution at all.

Details

The function requires a set of input parameters, apart from the signal vector. These parameters are contained in and read from the function `list_sensor()` and `list_logger()`. Poles and zeros are used to build the transfer function. The value s is the generator constant in Vs/m. The value k is the normalisation factor. AD is the analogue-digital conversion factor. If the signal was recorded with a gain value other than 1, the resulting signal needs to be corrected for this, as well.

Value

`Numeric` vector or list of vectors, deconvolved signal.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## deconvolve signal with minimum effort
rockfall_decon <- signal_deconvolve(data = rockfall_eseis)

## plot time series
plot_signal(data = rockfall_decon,
     main = "Rockfall, deconvolved signal",
     ylab = "m/s")

## add new logger manually
logger_new <- list_logger()[[1]]

## add logger data
logger_new$ID <- "logger_new"
logger_new$name <- "logger_new"
logger_new$AD <- 2.4414e-07

## deconvolve signal with new logger
rockfall_decon <- signal_deconvolve(data = rockfall_eseis,
                                    sensor = "TC120s",
                                    logger = logger_new)

## Change the setup of a logger, here: Centaur AD is changed due to
## other than default Vpp value, according to AD = V / (2^24).

## extract default Centaur logger
Centaur_10V <- list_logger()[[2]]

## replace AD value
Centaur_10V$AD <- 20/(2^24)
```

`signal_demean` *Remove mean of signal vector.*

Description

The function removes the mean from a signal vector.

Usage

```
signal_demean(data)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.

Value

`Numeric` vector or list of vectors, data set with mean subtracted.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## remove mean from data set
rockfall_demean <- signal_demean(data = rockfall_eseis)

## compare data ranges
range(rockfall_eseis$signal)
range(rockfall_demean$signal)

## show mean of initial signal
mean(rockfall_eseis$signal)
```

signal_detrend *Detrend a signal vector.*

Description

The function removes a linear trend from a signal vector.

Usage

```
signal_detrend(data)
```

Arguments

data	eseis object, numeric vector or list of objects, data set to be processed.

Value

Numeric vector or list of vectors, detrended data set.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## remove linear trend from data set
rockfall_detrend <- signal_detrend(data = rockfall_eseis)

## compare data ranges
range(rockfall_eseis$signal)
range(rockfall_detrend$signal)
```

signal_envelope *Calculate signal envelope.*

Description

The function calculates envelopes of the input signals as cosine-tapered envelope of the Hilbert-transformed signal. The signal should be detrended and/or the mean should be removed before processing.

Usage

```
signal_envelope(data, p = 0)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`p`	`Numeric` value, proportion of the signal to be tapered, default is `0`.

Value

`Numeric` vector or list of vectors, signal envelope.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## detrend data set
rockfall_detrend <- signal_detrend(data = rockfall_eseis)

## calculate envelope
rockfall_envelope <- signal_envelope(data = rockfall_detrend)

## plot envelope
plot_signal(data = rockfall_envelope)
```

`signal_filter` *Filter a seismic signal in the time or frequency domain*

Description

The function filters the input signal vector in the time or frequency domain.

Usage

```
signal_filter(
  data,
  f,
  fft = FALSE,
  dt,
  type,
  shape = "butter",
  order = 2,
  p = 0
)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`f`	`Numeric` value or vector of length two, lower and/or upper cutoff frequencies (Hz).
`fft`	`Logical` value, option to filter in the time domain (`fft = FALSE`) or the frequency domain (`fft = TRUE`). Default is (`fft = FALSE`).
`dt`	`Numeric` value, sampling period. If omitted, `dt` is set to 1/200.
`type`	`Character` value, type of filter, one out of `"LP"` (low pass), `"HP"` (high pass), `"BP"` (band pass) and `"BR"` (band rejection). If omitted, the type is interpreted from `f`. If `f` is of length two, `type` is set to `"BP"`. If `f` is of length one, `type` is set to `"HP"`.
`shape`	`Character` value, one out of `"butter"` (Butterworth), default is `"butter"`.
`order`	`Numeric` value, order of the filter, default is `2`. Only needed if `data` is no `eseis` object.
`p`	`Numeric` value, fraction of the signal to be tapered.

Value

`Numeric` vector or list of vectors, filtered signal vector.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## filter data set by bandpass filter between 1 and 90 Hz
rockfall_bp <- signal_filter(data = rockfall_eseis,
                             f = c(1, 90))

## taper signal to account for edge effects
rockfall_bp <- signal_taper(data = rockfall_bp,
                            n = 2000)

## plot filtered signal
plot_signal(data = rockfall_bp)

## compare time domain versus frequency domain filtering
rockfall_td <- signal_filter(data = rockfall_eseis,
                             f = c(10, 40),
                             fft = FALSE)

rockfall_td_sp <- signal_spectrum(data = rockfall_td)

rockfall_fd <- signal_filter(data = rockfall_eseis,
                             f = c(10, 40),
                             fft = TRUE)

rockfall_fd_sp <- signal_spectrum(data = rockfall_fd)

plot_spectrum(data = rockfall_td_sp)
plot_spectrum(data = rockfall_fd_sp)
```

`signal_hilbert` *Calculate Hilbert transform.*

Description

The function calculates the Hilbert transform of the input signal vector.

Usage

```
signal_hilbert(data)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.

Value

`Numeric` vector or list of vectors, Hilbert transform.

Author(s)

Michael Dietze

Examples

```
## load example data
data(rockfall)

## calculate hilbert transform
rockfall_h <- signal_hilbert(data = rockfall_eseis)
```

signal_hvratio *Calculate h-v-ratio of seismic components*

Description

This function uses three components of a seismic signal, evaluates their spectra and builds the ratio of horizontal to vertical power. For details see http://www.geopsy.org/documentation/geopsy/hv.html.

Usage

```
signal_hvratio(data, dt, method = "periodogram", kernel, order = "xyz")
```

Arguments

`data`	`List`, `data frame` or `matrix`, seismic componenents to be processed. If `data` is a matrix, the components must be organised as columns. Also, `data` can be a list of `eseis` objects.
`dt`	`Numeric` value, sampling period.
`method`	`Character` value, method for calculating the spectra. One out of `"periodogram"` , `"autoregressive"` and `"multitaper"`, default is `"periodogram"`.
`kernel`	`Numeric` value, window size (number of samples) of the moving window used for smoothing the spectra. By default no smoothing is performed.
`order`	`Character` value, order of the seismic components. Describtion must contain the letters `"x"`,`"y"` and `"z"` in the order according to the input data set. Default is `"xyz"` (EW-SN-vertical).

Details

The spectra should be smoothed. This can either be done directly during their calculation or before the calculation of the ratio. For the former case set `method = "autoregressive"`. For the latter case provide a value for `"kernel"`, which is the smoothing window size. Smoothing is performed with the logarithms of the spectral power data, using `caTools::runmean()` with the `endrule = "NA"`. After smoothing the data is re-linearised.

Value

A `data frame` with the h-v-frequency ratio.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(earthquake)

## ATTENTION, THIS EXAMPLE DATA SET IS FAR FROM IDEAL FOR THIS PURPOSE

## detrend data
s <- signal_detrend(data = s)

## calculate h-v-ratio, will be very rugged
hv <- signal_hvratio(data = s,
                     dt = 1 / 200)
plot(hv$ratio,
     type = "l",
     log = "y")

## calculate h-v-ratio using the autogressive spectrum method
hv <- signal_hvratio(data = s,
                     dt = 1 / 200,
                     method = "autoregressive")
plot(hv, type = "l")

## calculate h-v-ratio with a smoothing window equivalent to dt
hv <- signal_hvratio(data = s,
                     dt = 1 / 200,
                     kernel = 200)
plot(hv, type = "l")
```

`signal_integrate` *Integrate a seismic signal*

Description

The function integrates a signal vector to convert values from velocity to displacement. Two methods are available

Usage

```
signal_integrate(data, dt, method = "fft", waterlevel = 10^-6)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`dt`	`Numeric` scalar, sampling rate.
`method`	`Character` scalar, method used for integration. One out of `"fft"` (convert in the frequency domain) and `"trapezoid"` (integrate using the trapezoidal rule). Default is `"fft"`.
`waterlevel`	`Numeric` scalar, waterlevel value for frequency division, default is `10^-6`. Only used when `method = "fft"`.

Value

Numeric vector or list of vectors, integrated signal.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## deconvolve signal
rockfall_decon <- signal_deconvolve(data = rockfall_eseis)

## integrate signal
rockfall_int <- signal_integrate(data = rockfall_decon)

## Note that usually the signal should be filtered prior to integration.
```

---

signal_motion *Calculate particle motion parameters*

Description

The function calculates from a data set of three seismic components of the same signal the following particle motion paramters using a moving window approach: horizontal-vertical eigenvalue ratio, azimuth and inclination.

Usage

```
signal_motion(data, time, dt, window, step, order = "xyz")
```

Arguments

data	List, data frame or matrix, seismic componenents to be processed. If data is a matrix, the components must be organised as columns. Also, data can be a list of eseis objects.
time	POSIXct vector, time vector corresponding to the seismic signal components. If omitted, a synthetic time vector will be generated. If omitted, the sampling period (dt) must be provided.
dt	Numeric value, sampling period. Only needed if time is omitted or if data is no eseis object.

`window`	`Numeric` value, time window length (given as number of samples) used to calculate the particle motion parameters. If value is even, it will be set to the next smaller odd value. If omitted, the window size is set to 1 percent of the time series length by default.
`step`	`Numeric` value, step size (given as number of samples), by which the window is shifted over the data set. If omitted, the step size is set to 50 percent of the window size by default.
`order`	`Character` value, order of the seismic components. Describtion must contain the letters `"x"`,`"y"` and `"z"` in the order according to the input data set. Default is `"xyz"` (EW-NS-vertical).

Details

The function code is loosely based on the function GAZI() from the package RSEIS with removal of unnecessary content and updated or rewritten loop functionality.

Value

A List object with eigenvalue ratios (`eigen`), azimuth (`azimuth`) and inclination (`inclination`) as well as the corresponding time vector for these values.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(earthquake)

## filter seismic signals
s <- eseis::signal_filter(data = s,
                          dt = 1/200,
                          f = c(1, 3))

## integrate signals to get displacement
s_d <- eseis::signal_integrate(data = s, dt = 1/200)

## calculate particle motion parameters
pm <- signal_motion(data = s_d,
                    time = t,
                    dt = 1 / 200,
                    window = 100,
                    step = 10)

## plot function output
par_original <- par(no.readonly = TRUE)
par(mfcol = c(2, 1))

plot(x = t, y = s$BHZ, type = "l")
plot(x = pm$time, y = pm$azimuth, type = "l")
```

```
par(par_original)
```

signal_pad *Pad signal with zeros.*

Description

The function adds zeros to the input vector to reach a length, corresponding to the next higher power of two.

Usage

```
signal_pad(data)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.

Value

`Numeric` vector or list of vectors, signal vector with added zeros.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## pad with zeros
rockfall_pad <- signal_pad(data = rockfall_eseis)

## compare lengths
rockfall_eseis$meta$n
rockfall_pad$meta$n
```

signal_rotate *Rotate signal vectors using a 3-D rotation matrix.*

Description

The function rotates the horizontal components of the input data according to the specified angle.

Usage

```
signal_rotate(data, angle)
```

Arguments

`data`	`List`, `data frame` or `matrix`, seismic componenents to be processed. If `data` is a matrix, the components must be organised as rows. Also, `data` can be a list of `eseis` objects. If a matrix, this matrix must contain either two columns (x- and y-component) or three columns (x-, y-, and z-component), in exactly that order of the components.
`angle`	`Numeric` value, rotation angle in degrees.

Value

`Numeric` matrix, the 3-dimensional rotation matrix.

Author(s)

Michael Dietze

Examples

```
## create synthetic data set
data <- rbind(x = sin(seq(0, pi, length.out = 10)),
y = sin(seq(0, pi, length.out = 10)),
z = rep(0, 10))

## rotate the data set
x_rot <- signal_rotate(data = data,
                       angle = 15)

## plot the rotated data set
plot(x_rot[1,], col = 1, ylim = c(-2, 2))
points(x_rot[2,], col = 2)
points(x_rot[3,], col = 3)
```

signal_sign — *Convert amplitude signal to one bit signed signal*

Description

This function assigns 1 for positive values and -1 for negative input values of a signal.

Usage

```
signal_sign(data)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.

Value

`Numeric` vector or list of vectors, sign-cut signal.

Author(s)

Michael Dietze

Examples

```
## load example data
data(rockfall)

## sign-cut signal
rockfall_sign <- signal_sign(data = rockfall_eseis)
```

signal_snr — *Calculate signal-to-noise-ratio.*

Description

The function calculates the signal-to-noise ratio of an input signal vector as the ratio between mean and max.

Usage

```
signal_snr(data, detrend = FALSE)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`detrend`	`Logical` value, optionally detrend data set before calcualting snr.

Value

`Numeric` value, signal-to-noise ratio.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## calculate snr with detrend option off and on
snr <- signal_snr(data = rockfall_eseis)
print(snr$snr)

snr <- signal_snr(data = rockfall_eseis,
                  detrend = TRUE)
print(snr$snr)
```

`signal_spectrogram` *Calculate spectrograms (power spectral density estimates) from time series.*

Description

This function creates spectrograms from seismic signals. It supports the standard spectrogram approach, multitaper, and the Welch method.

Usage

```
signal_spectrogram(
  data,
  time,
  dt,
  Welch = FALSE,
  window,
  overlap = 0.5,
  window_sub,
  overlap_sub = 0.5,
  method = "periodogram",
  nw = 4,
  k = 7,
  n_cores = 1,
  plot = FALSE,
  ...
)
```

Arguments

data	Numeric vector or list of vectors, seismic signal to be processed.
time	POSIX.ct vector with time values. If omitted, an artificial time vector will be created, based on dt. Only needed if data is no eseis object.
dt	Numeric value, sampling period. If omitted, either estimated from time or set to 0.01 s (i.e., f = 100 Hz). Only needed if data is no eseis object.
Welch	Logical value, option to use the Welch method for calculations.
window	Numeric value, time window length in seconds used to calculate individual spectra. Set to 1 percent of the time series length by default.
overlap	Numeric value, fraction of window overlap.
window_sub	Numeric value, length of the sub-window in seconds used to calculate spectra. Only relevant if Welch = TRUE. If omitted, the sub-window length is set to 10 percent of the main window length.
overlap_sub	Numeric value, fraction of sub-window overlap.
method	Character value, method to calculate the spectra. One out of "periodogram", "autoregressive" and "multitaper". Default is "periodogram".
nw	Numeric value, multitaper time-bandwidth parameter, default is 4.0.
k	Numeric value, multitaper number of tapers, default is 7.
n_cores	Numeric value, number of CPU cores to use. Disabled by setting to 1. Default is 1.
plot	Logical value, toggle plot output. Default is FALSE. For more customised plotting see plot_spectrogram.
...	Additional arguments passed to the function.

Details

Data containing NA values is replaced by zeros and set to NA in the output data set.

Value

List with spectrogram matrix, time and frequency vectors.

Author(s)

Michael Dietze

See Also

spectrum, spec.pgram, spec.mtm

Examples

```
## load example data set
data("earthquake")

## calculate and plot PSD straight away
P <- signal_spectrogram(data = s$BHZ,
                               time = t,
                               dt = 1 / 200,
                               plot = TRUE)

## calculate and plot PSD with defined window sizes and the Welch method
P <- signal_spectrogram(data = s$BHZ,
                               time = t,
                               dt = 1 / 200,
                               window = 5,
                               overlap = 0.9,
                               window_sub = 3,
                               overlap_sub = 0.9,
                               Welch = TRUE,
                               plot = TRUE)

## calculate and plot PSD with even smaller window sizes, the Welch
## method and using multitapers, uncomment to use.
# P <- signal_spectrogram(data = s$BHZ,
#                                time = t,
#                                dt = 1 / 200,
#                                window = 2,
#                                overlap = 0.9,
#                                window_sub = 1,
#                                overlap_sub = 0.9,
#                                Welch = TRUE,
#                                method = "multitaper",
#                                plot = TRUE)
```

signal_spectrum *Calculate the spectrum of a time series*

Description

The power spectral density estimate of the time series is calculated using different approaches.

Usage

```
signal_spectrum(data, dt, method = "periodogram", ...)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`dt`	`Numeric` value, sampling period. If omitted, `dt` is set to 1/200. Only needed if `data` is no `eseis` object.
`method`	`Character` value, calculation method. One out of `"periodogram"` , `"autoregressive"` and `"multitaper"`, default is `"periodogram"`.
`...`	Additional arguments passed to the function. See `spec.pgram`, `spec.ar`, `spec.mtm`.

Value

`Data frame` with spectrum and frequency vector

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## calculate spectrum with standard setup
s <- signal_spectrum(data = rockfall_eseis)

## plot spectrum
plot_spectrum(data = s)
```

`signal_stalta` *Calculate stal-lta-ratio.*

Description

The function calculates the ratio of the short-term-average and long-term-average of the input signal.

Usage

```
signal_stalta(data, time, dt, sta, lta, freeze = FALSE, on, off)
```

Arguments

data	eseis object, numeric vector or list of objects, data set to be processed.
time	POSIXct vector, time vector of the signal(s). If not provided, a synthetic time vector will be created.
dt	Numeric value, sampling period. If omitted, either estimated from time or set to 0.01 s (i.e., f = 100 Hz).
sta	Numeric value, number of samples for short-term window.
lta	Numeric value, number of samples for long-term window.
freeze	Logical value, option to freeze lta value at start of an event. Useful to avoid self-adjustment of lta for long-duration events.
on	Numeric value, threshold value for event onset.
off	Numeric value, threshold value for event end.

Value

data frame, detected events (ID, start time, duration in seconds).

Author(s)

Michael Dietze

Examples

```
## load example data
data(rockfall)

## filter signal
rockfall_f <- signal_filter(data = rockfall_eseis,
                            f = c(1, 90),
                            p = 0.05)

## calculate signal envelope
rockfall_e <- signal_envelope(data = rockfall_f)

## pick earthquake and rockfall event
signal_stalta(data = rockfall_e,
              sta = 100,
              lta = 18000,
              freeze = TRUE,
              on = 5,
              off = 3)
```

signal_sum *Calculate signal vector sum.*

Description

The function calculates the vector sum of the input signals.

Usage

```
signal_sum(...)
```

Arguments

...	`Numeric` vectors or `eseis` objects, input signal, that must be of the same length.

Value

`Numeric` vector, signal vector sum.

Author(s)

Michael Dietze

Examples

```
## create random vectors
x <- runif(n = 1000, min = -1, max = 1)
y <- runif(n = 1000, min = -1, max = 1)
z <- runif(n = 1000, min = -1, max = 1)

## calculate vector sums
xyz <- signal_sum(x, y, z)
```

signal_taper *Taper a signal vector.*

Description

The function tapers a signal vector with a cosine bell taper, either of a given proportion or a discrete number of samples.

Usage

```
signal_taper(data, p = 0, n)
```

Arguments

`data`	`eseis` object, `numeric` vector or list of objects, data set to be processed.
`p`	`Numeric` value, proportion of the signal vector to be tapered. Alternative to `n`.
`n`	`Numeric` value, number of samples to be tapered at each end of the signal vector.

Value

`Data frame`, tapered signal vector.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## remove mean from data set
rockfall <- signal_demean(data = rockfall_eseis)

## create artefact at the beginning
rockfall_eseis$signal[1:100] <- runif(n = 100, min = -5000, max = 5000)

## taper signal
rockfall_taper <- signal_taper(data = rockfall, n = 1000)

## plot both data sets
plot_signal(data = rockfall_eseis)
plot_signal(rockfall_taper)
```

`signal_whiten` *Perform spectral whitening of a signal vector*

Description

The function normalises the input signal within a given frequency window. If a time series is provided, it is converted to the spectral domain, whitening is performed, and it is transformed back to the time domain.

Usage

```
signal_whiten(data, f, dt)
```

Arguments

`data`	`eseis` object, or `complex` vector, data set to be processed.
`f`	`Numeric` vector of length two, frequency window within which to normalise. If omitted, the entire bandwidth is normalised.
`dt`	`Numeric` value, sampling period. Only needed if the input object is not an `eseis` object

Value

`Numeric` vector or eseis object, whitened signal vector.

Author(s)

Michael Dietze

Examples

```
## load example data set
data("rockfall")

## whiten data set between 10 and 30 Hz
rockfall_2 <- signal_whiten(data = rockfall_eseis,
                            f = c(10, 30))

## plot whitened data set
plot(rockfall_2)
```

`spatial_amplitude` *Locate the source of a seismic event by modelling amplutide attenuation*

Description

The function fits a model of signal amplitude attenuation for all grid cells of the distance data sets and returns the residual sum as measure of the most likely source location of an event.

Usage

```
spatial_amplitude(data, coupling, d_map, aoi, v, q, f, a_0, normalise = TRUE)
```

Arguments

`data`	`Numeric` matrix or `eseis` object, seismic signals to work with. Since the function will calculate the maxima of the data it is usually the envolopes of the data that should be used here.
`coupling`	`Numeric` vector, coupling efficiency factors for each seismic station. The best coupled station (or the one with the highest amplification) must receive `1`, the others must be scaled relatively to this one.
`d_map`	`List` object, distance maps for each station (i.e., `SpatialGridDataFrame` objects). Output of `distance_map`.
`aoi`	`SpatialGridDataFrame` or `raster` object that defines which pixels are used to locate the source. If omitted, the entire distance map extent is used. `aoi` and `d_map` objects must have the same extents, projections and pixel sizes. The `aoi` map must be of logical values.
`v`	`Numeric` value, mean velocity of seismic waves (m/s).
`q`	`Numeric` value, quality factor of the ground.
`f`	`Numeric` value, frequency for which to model the attenuation.
`a_0`	`Logical` value, start parameter of the source amplitude, if not provided, a best guess is made as 100 times the maximum amplitude value of the data set.
`normalise`	`Logical` value, option to normalise sum of residuals between 0 and 1. Default is `TRUE`.

Value

A raster object with the sums of squared model residuals for each grid cell.

Author(s)

Michael Dietze

Examples

```
## Not run:

L <- spatial_amplitude(data = s_f,
coupling = c(1, 1, 1, 1),
d_map = D_50$maps,
v = 900,
q = 100,
f = 7.5)


## End(Not run)
```

`spatial_clip` *Clip values of spatial data.*

Description

The function replaces raster values based on different thresholds.

Usage

```
spatial_clip(data, quantile, replace = NA, normalise = TRUE)
```

Arguments

`data`	`raster` object, spatial data set to be processed.
`quantile`	`Numeric` value, quantile value below which raster values are clipped.
`replace`	`Numeric` value, replacement value, default is `NA`.
`normalise`	`Logical` value, optionally normalise values above threshold quantile between 0 and 1. Default is `TRUE`.

Value

`raster` object, data set with clipped values.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(volcano)

## convert matrix to raster object
volcano <- raster::raster(volcano)

## clip values to those > quantile 0.5
volcano_clip <- spatial_clip(data = volcano,
                                    quantile = 0.5)

## plot clipped data set
raster::plot(volcano_clip)
```

`spatial_convert` *Convert coordinates between reference systems*

Description

Coordinates are converted between reference systems.

Usage

```
spatial_convert(data, to)
```

Arguments

`data`	`Numeric` vector of length two or data frame, x-, y-coordinates to be converted.
`to`	`Character` value, proj4 string of the output reference system.

Value

`Numeric` data frame with converted coordinates.

Author(s)

Michael Dietze

Examples

```
## create lat lon coordinates
xy <- c(13, 55)

## define output coordinate system
proj_out <- "+proj=utm +zone=32 +datum=WGS84"

## convert coordinate pair
spatial_convert(data = xy,
                to = proj_out)

## define set of coordinates
xy <- data.frame(x = c(10, 11),
                 y = c(54, 55))

## convert set of coordinates
spatial_convert(data = xy,
                to = proj_out)
```

`spatial_crop` *Crop extent of spatial data.*

Description

The function crops the spatial extent of raster objects or other spatial objects based on bounding box coordinates.

Usage

```
spatial_crop(data, bbox)
```

Arguments

`data`	`raster` object, spatial data set to be processed.
`bbox`	`Numeric` vector of length four, bounding box coordinates in the form `c(xmin,xmax,ymin,ymax)`

Value

spatial object, cropped to bounding box

Author(s)

Michael Dietze

Examples

```
## create example data set
x <- raster::raster(nrows = 100,
                    ncols = 100,
                    xmn = 0,
                    xmx = 10,
                    ymn = 0,
                    ymx = 10)
raster::values(x) <- 1:10000

## create crop extent vector
bbox <- c(3, 7, 3, 7)

## crop spatial object
y <- spatial_crop(data = x,
                  bbox = bbox)

## plot both objects
raster::plot(x)
raster::plot(y, add = TRUE)
```

spatial_distance *Calculate topography-corrected distances for seismic waves.*

Description

The function calculates topography-corrected distances either between seismic stations or from seismic stations to pixels of an input raster.

Usage

```
spatial_distance(
  stations,
  dem,
  topography = TRUE,
  cores = 1,
  dmap = TRUE,
  dstation = TRUE,
  aoi
)
```

Arguments

`stations`	`Numeric` matrix of length two, x- and y-coordinates of the seismic stations to be processed (column-wise orgnaised).The coordinates must be in metric units, such as the UTM system and match with the reference system of the `dem`.
`dem`	`raster` object, the digital elevation model (DEM) to be processed. The DEM must be in metric units, such as the UTM system and match with the reference system of the coordinates of `stations`. See `raster` for supported types and how to read these to R.
`topography`	`Logical` scalar, option to enable topography correction, default is `TRUE`.
`cores`	`Numeric` scalar, number of CPU cores to use, only relevant for multicore computers. Default is `1`.
`dmap`	`Logical` scalar, option to enable/disable calculation of distance maps. Default is `TRUE`.
`dstation`	`Logical` scalar, option to enable/disable calculation of interstation distances. Default is `TRUE`.
`aoi`	`Numeric` vector of length four, bounding coordinates of the area of interest to process, in the form `c(x0,x1,y0,y1)`. Only implemented for single core mode (i.e., `cores = 1`).

Details

Topography correction is necessary because seismic waves can only travel on the direct path as long as they are within solid matter. When the direct path is through air, the wave can only travel along the surface of the landscape. The function accounts for this effect and returns the corrected travel distance data set.

Value

`List` object with distance maps list and station distance matrix.

Author(s)

Michael Dietze

Examples

```
## Not run:
## load and aggregate example DEM
data("volcano")
dem <- raster::raster(volcano)
dem <- raster::aggregate(x = dem, 2) * 10
dem@extent <- dem@extent * 1000
dem@extent <- dem@extent + c(510, 510, 510, 510)

## define example stations
stations <- cbind(c(200, 700), c(220, 700))

## plot example data
raster::plot(dem)
points(stations[,1], stations[,2])

## calculate distance matrices and stations distances
D <- spatial_distance(stations = stations,
                      dem = dem,
                      topography = TRUE,
                      cores = 1)

## plot distance map for station 2
raster::plot(D$maps[[2]])

## show station distance matrix
print(D$stations)

## run with small aoi
D <- spatial_distance(stations = stations,
                      dem = dem,
                      topography = TRUE,
                      cores = 1,
                      aoi = c(400, 600, 600, 800))

## End(Not run)
```

`spatial_migrate` *Migrate signals of a seismic event through a grid of locations.*

Description

The function performs signal migration in space in order to determine the location of a seismic signal.

Usage

```
spatial_migrate(data, d_stations, d_map, snr, v, dt, normalise = TRUE)
```

Arguments

`data`	`Numeric` matrix or `eseis` object, seismic signals to cross-correlate.
`d_stations`	`Numeric` matrix, inter-station distances. Output of `distance_stations`.
`d_map`	`List` object, distance maps for each station (i.e., `SpatialGridDataFrame` objects). Output of `distance_map`.
`snr`	`Numeric` vector, optional signal-to-noise-ratios for each signal trace, used for normalisation. If omitted it is calculated from input signals.
`v`	`Numeric` value, mean velocity of seismic waves (m/s).
`dt`	`Numeric` value, sampling period.
`normalise`	`Logical` value, option to normalise stations correlations by signal-to-noise-ratios.

Value

A SpatialGridDataFrame-object with Gaussian probability density function values for each grid cell.

Author(s)

Michael Dietze

Examples

```
## Not run:

## create synthetic DEM
dem <- raster::raster(nrows = 20, ncols = 20,
                      xmn = 0, xmx = 10000,
                      ymn= 0, ymx = 10000,
                      vals = rep(0, 400))

## define station coordinates
sta <- data.frame(x = c(1000, 9000, 5000),
                  y = c(1000, 1000, 9000),
                  ID = c("A", "B", "C"))

## create synthetic signal (source in the center of the DEM)
s <- rbind(dnorm(x = 1:1000, mean = 500, sd = 50),
           dnorm(x = 1:1000, mean = 500, sd = 50),
           dnorm(x = 1:1000, mean = 500, sd = 50))
```

```
## plot DEM and stations
raster::plot(dem)

text(x = sta$x,
     y = sta$y,
     labels = sta$ID)

## calculate spatial distance maps and inter-station distances
D <- eseis::spatial_distance(stations = sta[,1:2],
                             dem = dem)

## locate signal
e <- eseis::spatial_migrate(data = s,
                            d_stations = D$stations,
                            d_map = D$maps,
                            v = 1000,
                            dt = 1/100)

## get most likely location coordinates (example contains two equal points)
xy <- matrix(sp::coordinates(e)[raster::values(e) == max(raster::values(e))],
             ncol = 2)[1,]

## plot location estimate, most likely location estimate and stations
raster::plot(e)
points(xy[1],
       xy[2],
       pch = 20)
points(sta[,1:2])


## End(Not run)
```

`spatial_pmax` *Get most likely source location*

Description

The function identifies the location of a seismic source with the heighest likelihood (P_max).

Usage

```
spatial_pmax(data)
```

Arguments

`data`	`raster` object, spatial data set with source location estimates.

Value

`data.frame`, coordinates (x and y) of the most likely s ource location(s).

Author(s)

Michael Dietze

Examples

```
## create example source location likelihood raster
x <- raster::raster(nrows = 10,
                    ncols = 10,
                    xmn = 0,
                    xmx = 10,
                    ymn = 0,
                    ymx = 10)
raster::values(x) <- runif(n = 100)

## identify location of highest likelihood
p_max <- spatial_pmax(data = x)

## show result
print(p_max)
```

`time_aggregate` *Aggregate a time series*

Description

The time series `x` is aggregated by an integer factor `n`.

Usage

```
time_aggregate(data, n = 2)
```

Arguments

`data`	`POSIXct` vector, time to be processed.
`n`	`Numeric` value, number of samples to be aggregated to one new data value. Must be an integer value greater than 1. Default is `2`.

Value

`POSIXct` vector, aggregated data.

Author(s)

Michael Dietze

Examples

```
## load example data set
data(rockfall)

## aggregate time series
rockfall_t_agg <- time_aggregate(data = rockfall_t,
                                 n = 2)

## compare results
range(rockfall_t)
diff(rockfall_t)

range(rockfall_t_agg)
diff(rockfall_t_agg)
```

---

time_clip *Clip time vector.*

Description

The function clips a time vector based on provided limits.

Usage

```
time_clip(time, limits)
```

Arguments

`time`	`POSIXct` vector, time vector.
`limits`	`POSIXct` vector of length two, time limits for clipping.

Value

`POSIXct` vector, clipped time vector.

Author(s)

Michael Dietze

Examples

```
## load example data
data(rockfall)

## define limits to clip to
limits <- c(min(rockfall_t) + 10,
            max(rockfall_t) - 10)

## clip data set
rockfall_t_clip <- time_clip(time = rockfall_t,
                             limits = limits)

## compare time ranges
range(rockfall_t)
range(rockfall_t_clip)
```

`time_convert` *Convert Julian Day to Date and vice versa*

Description

The function converts a Julian Day value to a date, to `POSIXct` if a year is provided, otherwise to `POSIXlt`.

Usage

```
time_convert(input, output, timezone = "UTC", year)
```

Arguments

`input`	`Numeric` vector, input time Supported formats are `YYYY-MM-DD`, `JD` and `POSIXct`.
`output`	`Numeric` vector, output time. Supported formats are `YYYY-MM-DD`, `JD` and `POSIXct`.
`timezone`	`Character` vector, time zone of the output date. Default is `"UTC"`.
`year`	`Character` vector, year of the date. Only used when `input` is `JD`. If omitted, the current year is used.

Value

`Numeric` vector,

Author(s)

Michael Dietze

Examples

```
## convert Julian Day 18 to POSIXct
time_convert(input = 18, output = "POSIXct")

## convert Julian Day 18 to yyyy-mm-dd
time_convert(input = 18, output = "yyyy-mm-dd")

## convert yyyy-mm-dd to Julian Day
time_convert(input = "2016-01-18", output = "JD")

## convert a vector of Julian Days to yyyy-mm-dd
time_convert(input = 18:21, output = "yyyy-mm-dd")
```

---

write_mseed *Write seismic traces as mseed file to disk.*

Description

This function converts seismic traces to mseed files and writes them to disk. It makes use of the Python library 'ObsPy'. Thus, this software must be installed, to make use of this function.

Usage

```
write_mseed(data, file, time, component, station, location, network, dt)
```

Arguments

`data`	`eseis` object or `numeric` vector, data set to be processed. Most other arguments can be omitted if `data` is an `eseis` object.
`file`	`Character` scalar, mseed file name with extension.
`time`	`POSIXct` vector, time vector corresponding to the seismic trace. Alternatively, the start time stamp can be provided as `POSIXct` value and a value for `dt` must be given.
`component`	`Character` value, component ID, optional.
`station`	`Character` value, station ID, optional.
`location`	`Character` vector of length four, station location data (latitude, longitude, elevation, depth), optional.
`network`	`Character` value, network ID, optional.
`dt`	`Numeric` value, sampling period. Only needed if no time vector is provided.

Details

The ObsPy Python library can be installed following the information provided here: `"https://github.com/obspy/obspy/w`

Since the ObsPy functionality through R is not able to interpret path definitions using the tilde symbol, e.g. `"~/Downloads"`, this Linux type definition must be avoided.

Value

A binary file written to disk.

Author(s)

Michael Dietze

Examples

```
## Not run:
## load example data
data("rockfall")

## write as mseed file
write_mseed(data = rockfall_eseis, file = "rockfall.mseed")


## End(Not run)
```

write_report — *Create a HTML report for (RLum) objects*

Description

This function creates a HTML report for a given eseis object, listing its complete processing history. The report serves both as a convenient way of browsing through objects and as a proper approach to documenting and saving scientific data and workflows.

Usage

```
write_report(object, file, title = "eseis report", browser = FALSE, css)
```

Arguments

`object,`	`eseis` object to be reported on
`file`	`Character` value, name of the output file (without extension)
`title`	`Character` value, title of the report
`browser`	`Logical` value, optionally open the HTML file in the default web browser after it has been rendered.
`css`	`Character` value, path to a CSS file to change the default styling of the HTML document.

Details

The function heavily lends ideas from the function `report_RLum()` written by Christoph Burow, which is contained in the package `Luminescence`. This function here is a truncated, tailored version with minimised availabilities.

Value

HTML and .Rds file.

Author(s)

Michael Dietze

Examples

```
## Not run:
## load example data set
data(rockfall)

## make report for rockfall object
write_report(object = rockfall_eseis,
             browser = TRUE)

## End(Not run)
```

`write_sac` *Write seismic traces as sac file to disk.*

Description

This function converts seismic traces to sac files and writes them to disk.

Usage

```
write_sac(
  data,
  file,
  time,
  component,
  unit,
  station,
  location,
  network,
  dt,
  autoname = FALSE,
  parameters,
  biglong = FALSE
)
```

Arguments

`data`	`eseis` object or `numeric` vector, data set to be processed. Most other arguments can be omitted if `data` is an `eseis` object.
`file`	`Character` scalar, sac file name with extension.
`time`	`POSIXct` vector, time vector corresponding to the seismic trace. Alternatively, the start time stamp can be provided as `POSIXct` value and a value for `dt` must be given.
`component`	`Character` value, component ID, optional.
`unit`	`Character` value, unit of the signal, optional. One out of `"unknown"`, `"displacement"`, `"velocity"`, `"volts"`, `"acceleration"`. Default is `"unknown"`.
`station`	`Character` value, station ID, optinal.
`location`	`Character` vector of length four, station location data (latitude, longitude, elevation, depth), optional.
`network`	`Character` value, network ID, optional.
`dt`	`Numeric` value, sampling period. Only needed if no time vector is provided.
`autoname`	`Logical` value, option to let the function generate the file name automatically. Default is `FALSE`.
`parameters`	`Data frame` sac parameter list, as obtained from `list_sacparameters`. Allows user-specific modifications. If this data frame is provided, it overrides all other arguments.
`biglong`	`Logical` value, biglong option, default is `FALSE`

Details

For description of the sac file format see https://ds.iris.edu/files/sac-manual/manual/file_format.html. Currently the following parameters are not supported when writing the sac file: LAT, LON, ELEVATION, NETWORK.

Value

A binary file written to disk.

Author(s)

Michael Dietze

Examples

```
## Not run:
## load example data
data("rockfall")
```

```
## write as sac file
write_sac(data = rockfall_eseis)


## End(Not run)
```

Index