Package: MatchingPursuit (via r-universe)

June 9, 2026

Type Package

Title Processing Time Series Data Using the Matching Pursuit Algorithm

Version 1.1.0

Author Artur Gramacki [aut, cre] (ORCID: <https://orcid.org/0000-0002-1610-9743>), Jarosław Gramacki [ctb] (ORCID: <https://orcid.org/0000-0001-5032-1353>), Piotr T. Różański [ctb] (ORCID: <https://orcid.org/0000-0002-0457-6731>)

Maintainer Artur Gramacki <a.gramacki@gmail.com>

Description Provides tools for analysing and decomposing time series data using the Matching Pursuit (MP) algorithm, a greedy signal decomposition technique that represents complex signals as a linear combination of simpler functions (called atoms) selected from a redundant dictionary. For more details see Mallat and Zhang (1993) <doi:10.1109/78.258082>, Pati et al. (1993) <doi:10.1109/ACSSC.1993.342465>, Elad (2010) <doi:10.1007/978-1-4419-7011-4> and Różański (2024) <doi:10.1145/3674832>.

SystemRequirements external tool (installed via empi.install() function). The package uses the implementation of the Matching Pursuit algorithm by Piotr T. Różański, available at https://github.com/develancer/empi.

Imports edf, signal, RSQLite, DescTools, imager, raster, graphics, grDevices, utils, digest, EGM

Suggests knitr, rmarkdown, latex2exp, remotes

VignetteBuilder knitr

Depends R (>= 3.5.0)

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Config/pak/sysreqs cmake libfftw3-dev libgdal-dev gdal-bin libgeos-dev libglpk-dev make libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libx11-dev zlib1g-dev

Repository https://artur-gramacki.r-universe.dev

Date/Publication 2026-05-10 22:22:17 UTC

RemoteUrl https://github.com/artur-gramacki/matchingpursuit

RemoteRef HEAD

RemoteSha fa3d33178804e8724508d0a370017072a93c737b

Contents

atom.params . . . 2
clear.cache . . . 3
eeg.montage . . . 4
empi.check . . . 5
empi.execute . . . 6
empi.install . . . 7
empi.locate . . . 8
empi2tf . . . 8
filters.coeff . . . 11
gabor.fun . . . 13
plot.ecg . . . 14
plot.edf . . . 15
plot.empi . . . 17
read.csv.signals . . . 19
read.ecg.signals . . . 20
read.edf.params . . . 21
read.edf.signals . . . 22
read.empi.db.file . . . 23
sig2bin . . . 24

Index **26**

`atom.params` *Read atom parameters from a SQLite database*

Description

Reads atom parameters stored in a SQLite database created by `empi.execute()` function.

Usage

```
atom.params(db.file)
```

Arguments

db.file	A character string giving the path to a SQLite database file.

Value

A data frame containing the atom parameters stored in the database.

Examples

```
# Example database containing data from 18 channels
file <- system.file("extdata", "EEG.db", package = "MatchingPursuit")
out <- atom.params(file)
out[which(out$channel_id == 1), ]
out[which(out$channel_id == 18), ]

# Example database containing data from a single channel
file <- system.file("extdata", "sample1.db", package = "MatchingPursuit")
out <- atom.params(file)
out
```

---

clear.cache *Clear MatchingPursuit Cache*

Description

Deletes all files in the MatchingPursuit cache directory.

Usage

```
clear.cache()
```

Value

Logical scalar. Returns TRUE if all files were successfully removed, and FALSE otherwise. The return value is invisible.

Examples

```
if (interactive()) {
  clear.cache()
}
```

`eeg.montage` *Performs bipolar, reference or average EEG montage*

Description

An EEG montage refers to the arrangement of EEG electrodes and the way their signals are displayed relative to one another during electroencephalogram interpretation. The same EEG recording may appear very different depending on the montage used. This function implements the three montage methods most commonly used in practice: 1) Bipolar Montage, 2) Referential (Monopolar) Montage, and 3) Average Reference Montage.

Usage

```
eeg.montage(
  x,
  montage.type = c("average", "reference", "bipolar"),
  ref.channel = NULL,
  bipolar.pairs = NULL
)
```

Arguments

`x`	Object of class `edf` (from `read.edf.signals()`).
`montage.type`	A character string specifying the montage type. • `"average"` - each electrode is referenced to the average of all electrodes • `"reference"` - each active electrode is compared to a single common reference electrode • `"bipolar"` - each channel compares two adjacent electrodes
`ref.channel`	Name of the reference channel for `"reference"` montage.
`bipolar.pairs`	List of electrodes pairs for `"bipolar"` montage. See example below.

Details

To check the channel names in the analysed EEG recording, use the `read.edf.params()` function.

Value

An object of class `edf`.

Examples

```
file <- system.file("extdata", "EEG.edf", package = "MatchingPursuit")
out <- read.edf.signals(file, resampling = FALSE, from = 0, to = 10)

read.edf.params(file)
```

```
# The classical double banana montage.
pairs <- list(
  c("Fp2", "F4"),
  c("F4",  "C4"),
  c("C4",  "P4"),
  c("P4",  "O2"),
  c("Fp1", "F3"),
  c("F3",  "C3"),
  c("C3",  "P3"),
  c("P3",  "O1"),
  c("Fp2", "F8"),
  c("F8",  "T4"),
  c("T4",  "T6"),
  c("T6",  "O2"),
  c("Fp1", "F7"),
  c("F7",  "T3"),
  c("T3",  "T5"),
  c("T5",  "O1"),
  c("Fz",  "Cz"),
  c("Cz",  "Pz")
)

signal.bip.mont <- eeg.montage(out, montage.type = c("bipolar"), bipolar.pairs = pairs)
signal.ref.mont <- eeg.montage(out, montage.type = c("reference"), ref.channel = "O1")
signal.avg.mont <- eeg.montage(out, montage.type = c("average"))

head(signal.bip.mont$signal)
head(signal.ref.mont$signal)
head(signal.avg.mont$signal)
```

`empi.check` *Checks if EMPI external software is installed*

Description

The EMPI program is installed using the `empi.install()` function and stored in the cache directory. This function checks whether the EMPI program is still available there (users have full access to the cache directory and may remove its contents at any time).

Usage

```
empi.check()
```

Value

If the EMPI program is found, its full path is returned. Otherwise, a message is displayed, prompting the user to install it using the `empi.install()` function.

Examples

```
empi.check()
```

`empi.execute` *Launches the empi program*

Description

Runs the EMPI program for the given data (signal).

Usage

```
empi.execute(
  signal,
  empi.options = NULL,
  write.to.file = FALSE,
  path = NULL,
  file.name = NULL
)
```

Arguments

`signal`	List containing the signal in a data frame together with its sampling frequency. The data frame should have meaningful column names (channel names). The list must contain elements named `"signal"` and `"sampling.rate"`.
`empi.options`	If `NULL`, the EMPI program is run with `"-o local --gabor -i 50 --cpu-workers 8"` parameters. Otherwise, the user may specify any command-line options. See the `README.md` file after downloading the EMPI program using the `empi.install()` function.
`write.to.file`	If `TRUE`, a SQLite database file will be created and saved in the `path` directory or, if `path = NULL`, in the cache directory. This file stores the results of signal decomposition using the MP algorithm
`path`	Directory in which the SQLite database file will be saved. If `NULL`, the file will be saved in the cache directory.
`file.name`	Name of the file to create if `write.to.file = TRUE`.

Details

The EMPI program (source code and binary files for multiple operating systems) can be downloaded from https://github.com/develancer/empi. Details are presented in the journal paper: Różański, P. T. (2024). *empi: GPU-Accelerated Matching Pursuit with Continuous Dictionaries*. ACM Transactions on Mathematical Software, Volume 50, Issue 3, Article No. 17, pp. 1-17, doi:10.1145/3674832.

Value

Results of signal decomposition using the MP algorithm. An object of class `empi` is returned. If `write.to.file = TRUE`, the results are also written to a SQLite file in the `path` directory.

Examples

```
## Not run:
file <- system.file("extdata", "sample1.csv", package = "MatchingPursuit")
out <- read.csv.signals(file)

out.empi <- empi.execute(
  signal = out,
  empi.options = NULL,
  write.to.file = FALSE,
  path = NULL,
  file.name = NULL
)

plot(out.empi)
## End(Not run)
```

`empi.install` *Installs the EMPI external program*

Description

Downloads the **Enhanced Matching Pursuit Implementation** (EMPI) external program compatible with the current operating system and stores it in the package cache directory.

Usage

```
empi.install()
```

Details

The function detects the operating system (Windows, Linux, macOS x64, macOS arm64), downloads the appropriate archive from the official repository, verifies its integrity using a checksum, and extracts it.

Value

The function downloads the EMPI program in a version compatible with the operating system used (Windows, Linux, MacOS-x64, MacOS-arm64) and stores it in the package cache directory.

Examples

```
if (interactive()) {
  empi.install()
}
```

`empi.locate` *Get required external software localization*

Description

Returns **Enhanced Matching Pursuit Implementation** binary locations for the following operation systems: Windows, Linux, MacOS-x64, MacOS-arm64.

Usage

```
empi.locate()
```

Value

List with URL of the EMPI binaries and zip file name.

Examples

```
empi.locate()
```

`empi2tf` *Creates a time-frequency map using atoms from the Matching Pursuit algorithm*

Description

Creates a time-frequency map using atoms from the Matching Pursuit algorithm. The resulting map can be: 1) displayed on the screen, 2) saved as a `.png` file, or 3) saved as an `.RData` object.

Usage

```
empi2tf(
  x = NULL,
  channel,
  mode = "sqrt",
  freq.divide = NULL,
  increase.factor = 1,
  shortening.factor.x = 2,
  shortening.factor.y = 2,
```

```
  display.crosses = TRUE,
  display.atom.numbers = FALSE,
  display.grid = FALSE,
  crosses.color = "white",
  palette = "my custom palette",
  rev = TRUE,
  out.mode = "plot",
  path = NULL,
  file.name = NULL,
  size = c(512, 512),
  draw.ellipses = FALSE,
  plot.signals = TRUE,
  write.atoms = FALSE
)
```

Arguments

x	An object of class empi or a path to a SQLite file created by empi.execute().
channel	Channel from the SQLite file to process.
mode	"sqrt", "log", or "linear". Determines the intensity with which the so-called blobs are displayed on the T-F map.
freq.divide	Specifies how many times the displayed frequency range in the T-F map should be reduced. At high sampling rates, especially when a low-pass filter with a cut-off frequency much lower than the sampling frequency is used, a large part of the T-F map may contain no blobs. If the sampling frequency is f, the maximum frequency displayed in the T-F map will be ceiling(f / 2 / freq.divide) (f / 2 follows the Nyquist rule). If NULL, it is determined from the atom with the highest frequency fmax according to freq.divide = (f / 2) / fmax.
increase.factor	Factor controlling the increase in the number of pixels along the frequency axis. Non-negative integers such as 2, 4, 5, or 8 are usually appropriate.
shortening.factor.x	Usually, a value of 2 provides better atom visualization.
shortening.factor.y	Usually, a value of 2 provides better atom visualization.
display.crosses	Whether small crosses should be displayed at the centres of atoms.
display.atom.numbers	Whether atom numbers should be displayed in the canters of atoms.
display.grid	Whether grid lines should be drawn.
crosses.color	Colour of the small crosses.
palette	PPalette from the list returned by the hcl.pals() function or the string "my custom palette".
rev	Value of the rev argument passed to the hcl.colors() function.
out.mode	One of the following:

- "plot" - draws a T-F map on the screen.
- "file" - saves a T-F map to the file file.name (as a png file).
- "RData" - saves the T-F map of size size to file.name (as an R matrix); resampling is performed using the imager::resize() function.
- "RData2" - saves the T-F map of size size to file.name (as an R matrix); resampling is performed using the raster::resample() function.

path	Path where png, RData, or pdf files will be written. If NULL, files will be written to the cache directory.
file.name	Name of the png file (if out.mode = "file") or name of the RData file (if out.mode = "RData" or out.mode = "RData2").
size	Size of the png file in pixels (if out.mode = "file") or size of the T-F matrix (if out.mode = "RData" or out.mode = "RData2").
draw.ellipses	Intended for testing only. Can be set to TRUE to display the effect. Works correctly only if out.mode = "plot".
plot.signals	Whether the original and reconstructed signals should also be displayed.
write.atoms	If TRUE, writes all atom plots to the Atoms.pdf file (in the cache directory or in a user-specified directory, depending on path).

Value

Depending on the out.mode parameter, the function:

- displays the time-frequency map on the screen
- saves the time-frequency map as a .png file
- saves the time-frequency map as a .RData file

Regardless of the output mode, the function also returns:

- all Gabor functions
- reconstructed signal
- original signal
- sampling frequency
- grid size along the time axis
- grid size along the frequency axis
- epoch size in samples
- signal length in seconds
- time-frequency map
- resampled time-frequency map (if out.mode = "RData" or out.mode = "RData2"; otherwise NULL)
- processed channel number
- frequency division factor

Examples

```
file <- system.file("extdata", "sample1.db", package = "MatchingPursuit")
empi.class <- read.empi.db.file(file)

# 'freq.divide' is set arbitrarily
out <- empi2tf(
  x = empi.class,
  channel = 1,
  mode = "sqrt",
  freq.divide = 4,
  increase.factor= 4,
  display.crosses = TRUE,
  display.atom.numbers = FALSE,
  out.mode = "plot",
)

# 'freq.divide' is determined based on the atom with the highest frequency
out <- empi2tf(
  x = empi.class,
  channel = 1,
  mode = "sqrt",
  increase.factor= 4,
  display.crosses = TRUE,
  display.atom.numbers = FALSE,
  out.mode = "plot",
)
```

`filters.coeff` *A wrapper function for* `signal::butter()` *function*

Description

Implements notch, low-pass, high-pass, band-pass, and band-stop filters with specified frequency ranges and Butterworth filter order.

Usage

```
filters.coeff(
  fs = 256,
  notch = c(49, 51),
  notch.order = 2,
  lowpass = 30,
  lowpass.order = 4,
  highpass = 1,
  highpass.order = 4,
  bandpass = c(0.5, 40),
  bandpass.order = 4,
  bandstop = c(0.5, 40),
```

```
  bandstop.order = 4
)
```

Arguments

fs	Sampling rate.
notch	Vector of two frequencies for notch filter.
notch.order	Notch filter order.
lowpass	Low-pass filter frequency.
lowpass.order	Low-pass filter order.
highpass	High-pass filter frequency.
highpass.order	High-pass filter order.
bandpass	Vector of two frequencies for band-pass filter.
bandpass.order	Band-pass filter order.
bandstop	Vector of two frequencies for band-stop filter.
bandstop.order	Band-stop filter order.

Value

List with parameters of individual filters.

Examples

```
file <- system.file("extdata", "EEG.edf", package = "MatchingPursuit")
out <- read.edf.signals(file, resampling = FALSE)
signal <- out$signal
sampling.rate <- out$sampling.rate

fc <- filters.coeff(
  fs = sampling.rate,
  notch = c(49, 51),
  lowpass = 40,
  highpass = 1,
  bandpass = c(0.5, 40),
  bandstop = c(10, 50)
)

print(fc)

signal::freqz(fc$notch, Fs = sampling.rate)
signal::freqz(fc$lowpass, Fs = sampling.rate)
signal::freqz(fc$highpass, Fs = sampling.rate)
signal::freqz(fc$bandpass, Fs = sampling.rate)
signal::freqz(fc$bandstop, Fs = sampling.rate)

plot(signal[, 1], type = "l", panel.first = grid())

signal.filt <- signal
```

```
for (m in 1:ncol(signal)) {
  signal.filt[, m] = signal::filtfilt(fc$notch, signal.filt[, m]); # 50Hz notch filter
 signal.filt[, m] = signal::filtfilt(fc$lowpass, signal.filt[, m]); # Low pass IIR Butterworth
 signal.filt[, m] = signal::filtfilt(fc$highpass, signal.filt[, m]); # High pass IIR Butterwoth
}

plot(signal.filt[, 1], type = "l", panel.first = grid())
```

`gabor.fun` *Gabor function implementation*

Description

A Gabor function is a sinusoidal wave localized by a Gaussian envelope. In signal processing, it is widely used as a basic building block for representing signals localized in both time and frequency. The Matching Pursuit algorithm uses a redundant dictionary of so-called *Gabor atoms*. These atoms are particularly suitable because they: 1) provide optimal time–frequency localization, 2) represent oscillatory signals well, 3) enable adaptive time-frequency decomposition.

Usage

```
gabor.fun(
  number.of.samples,
  sampling.frequency,
  mean,
  phase,
  sigma,
  frequency,
  normalization = TRUE
)
```

Arguments

`number.of.samples`	Number of samples in the generated atom.
`sampling.frequency`	Sampling frequency.
`mean`	Time position of the Gaussian envelope.
`phase`	Phase of the sinusoidal component.
`sigma`	Scale parameter controlling the width of the Gaussian window.
`frequency`	Frequency of the sinusoidal component.
`normalization`	If `TRUE`, the resulting atom is normalized to have unit norm.

Value

A list containing four numeric vectors of length `number.of.samples`: cosine, Gaussian envelope, Gabor function, and time axis.

Examples

```
number.of.samples <- 512
sampling.frequency <- 256.0
mean <- 1
phase <- pi
sigma <- 0.5
frequency <- 5.0
normalization = TRUE

out <- gabor.fun(
  number.of.samples,
  sampling.frequency,
  mean,
  phase,
  sigma,
  frequency,
  normalization
)

# If normalization = TRUE, norm of atom = 1, we can check it
crossprod(out$gabor)

plot(out$t, out$gabor, type = "l", xlab = "t", ylab = "gabor", panel.first = grid())
```

`plot.ecg` *The function displays ECG signals in a layout corresponding to standard paper ECG printouts*

Description

A typical ECG paper layout was used, with a small grid of 0.04 s × 0.1 mV and a large grid of 0.20 s × 0.5 mV.

Usage

```
## S3 method for class 'ecg'
plot(
  x,
  begin,
  end,
  panel.height = 3,
  small.squares = TRUE,
  zero.line = FALSE,
```

```
  ...
)
```

Arguments

x	Object of class ecg (from read.ecg.signals()).
begin	Time point (in seconds) at which to start plotting.
end	Time point (in seconds) at which to stop plotting.
panel.height	Number of large squares to display (according to standard ECG paper): • small grid: 0.04 sec. x 0.1 mV • large grid: 0.20 sec. x 0.5 mV
small.squares	If TRUE, the small grid is also displayed.
zero.line	If TRUE, a horizontal line representing 0 mV is displayed.
...	Currently ignored. Required for compatibility with the generic plot().

Value

No return value, called to visualize an ECG graph.

Examples

```
# ECG data comes from https://physionet.org/content/ptb-xl/1.0.3/
file <- system.file("extdata", "00001_lr.hea", package = "MatchingPursuit")
dir <- dirname(file)
name <- tools::file_path_sans_ext(basename(file))

out <- read.ecg.signals(file)

plot(
  x = out,
  begin = 0,
  end = 10,
  panel.height = 1,
  zero.line = FALSE,
  small.squares = TRUE
)
```

---

plot.edf *The function displays EEG signals*

Description

Signals are displayed one below another and may be shown in different colours for improved readability.

Usage

```
## S3 method for class 'edf'
plot(
  x,
  begin,
  end,
  panel.height = NULL,
  rainbow = TRUE,
  bg.colour = "black",
  txt.col = "white",
  zero.line = TRUE,
  main = NULL,
  ...
)
```

Arguments

x	Object of class edf (from read.edf.signals()).
begin	Time point (in seconds) at which to start plotting.
end	Time point (in seconds) at which to stop plotting.
panel.height	Controls the vertical spacing between individual signals. If NULL, the value is chosen automatically so that all signals are clearly visible and do not overlap.
rainbow	If TRUE, individual channels are drawn in different colours.
bg.colour	Background colour.
txt.col	Colour of text elements (axis labels and title).
zero.line	If TRUE, a horizontal line representing 0 mV is displayed.
main	The text shown as the plot title.
...	Currently ignored. Required for compatibility with the generic plot().

Value

No return value, called to visualize an EEG graph.

Examples

```
file <- system.file("extdata", "EEG.edf", package = "MatchingPursuit")
out  <- read.edf.signals(file, resampling = FALSE)

plot(
  x = out,
  begin = 0,
  end = 10,
  panel.height = NULL,
  rainbow = TRUE,
  bg.colour = "black",
  txt.col = "white",
  zero.line = TRUE,
```

```
  main = "EEG signals stored in the EEG.edf file"
)

plot(
  x = out,
  begin = 0,
  end = 10,
  panel.height = NULL,
  rainbow = FALSE,
  bg.colour = "white",
  txt.col = "black",
  zero.line = TRUE,
  main = "EEG signals stored in the EEG.edf file"
)
```

`plot.empi` *Plots a time-frequency (T-F) map to visualize EMPI decomposition*

Description

This function is a wrapper around `empi2tf()` with `out.mode = "plot"`.

Usage

```
## S3 method for class 'empi'
plot(
  x,
  channel = 1,
  mode = "sqrt",
  freq.divide = NULL,
  increase.factor = 8,
  shortening.factor.x = 2,
  shortening.factor.y = 2,
  display.crosses = TRUE,
  display.atom.numbers = FALSE,
  display.grid = FALSE,
  crosses.color = "white",
  palette = "my custom palette",
  plot.signals = TRUE,
  ...
)
```

Arguments

`x`	An object of class `empi` created by `empi.execute()`.
`channel`	Channel from the SQLite file to process.
`mode`	`"sqrt"`, `"log"`, or `"linear"`. Determines the intensity with which the so-called blobs are displayed on the T-F map.

`freq.divide` Specifies how many times the displayed frequency range in the T-F map should be reduced. At high sampling rates, and when a low-pass filter with a cut-off frequency much lower than the sampling frequency is used, a large part of the T-F map may contain no blobs. If the sampling frequency is `f`, the maximum frequency in the T-F map will be `ceiling(f / 2 / freq.divide)` (`f / 2` follows the Nyquist rule). If `NULL`, it is determined from the atom with the highest frequency `fmax` according to `freq.divide = (f / 2) / fmax`.

`increase.factor`
Factor controlling the increase in the number of pixels along the frequency axis. Non-negative integers such as 2, 4, 5, or 8 are typically appropriate.

`shortening.factor.x`
Usually, a value of 2 provides better visualization of atoms.

`shortening.factor.y`
Usually, a value of 2 provides better visualization of atoms.

`display.crosses`
Whether small crosses should be displayed at the centres of atoms.

`display.atom.numbers`
Whether atom numbers should be displayed at the centres of atoms.

`display.grid` Whether grid lines should be drawn.

`crosses.color` Colour of the small crosses.

`palette` Palette from the list returned by `hcl.pals()` or the string `"my custom palette"`.

`plot.signals` Whether the original and reconstructed signals should also be displayed.

`...` Currently ignored. Required for compatibility with the generic `plot()`.

Value

No return value, called to visualize the empi decomposition.

Examples

```
file <- system.file("extdata", "sample1.csv", package = "MatchingPursuit")
signal <- read.csv.signals(file, col.names = "ch1")

# Execute the MP algorithm.
empi.class <- empi.execute(signal = signal)

# Plot a time-frequency map based on MP atoms.
plot(empi.class)
```

`read.csv.signals` *Reads and validates a CSV file structure*

Description

Reads and validates a CSV file structure

Usage

```
read.csv.signals(file, col.names = NULL, col.names.in.csv = FALSE)
```

Arguments

`file`	File to be read and checked. The first line of the file must contain two numbers: the sampling rate in Hz (`freq`) and the signal length in seconds (`sec`). The function verifies whether the file contains exactly `round(freq * sec)` samples. The two numbers must be separated by one or more whitespace characters.
`col.names`	Optional character vector of column names. If not specified, default names are created.
`col.names.in.csv`	iLogical value. If `TRUE`, the second line of the file is assumed to contain column names.

Value

A list containing:

- a data frame (rows = samples, columns = channels),
- sampling rate.

Examples

```
file <- system.file("extdata", "sample1.csv", package = "MatchingPursuit")

# The first line of the file must contain two numbers:
# a) the sampling rate in Hz
# b) the signal length in seconds
out <- read.csv(file, header = FALSE)
head(out)

signal <- read.csv.signals(file, col.names = "signal_1")
head(signal$signal)
signal$sampling.rate

file <- system.file("extdata", "sample3.csv", package = "MatchingPursuit")
signal <- read.csv.signals(file, col.names = c("signal_1", "signal_2", "signal_3"))
head(signal$signal)
signal$sampling.rate
```

```
# Now, the csv file contains signal names in the second line
file <- system.file("extdata", "EEG.csv", package = "MatchingPursuit")
signal <- read.csv.signals(file, col.names.in.csv = TRUE)
head(signal$signal)
signal$sampling.rate

# Now, the csv file contains signal names in the second line
# The data here is the same as in the EEG.csv file, but after performing
# 'double banana' montages and after applying filtering.
file <- system.file("extdata", "EEG_bipolar_filtered.csv", package = "MatchingPursuit")
signal <- read.csv.signals(file, col.names.in.csv = TRUE)
head(signal$signal)
signal$sampling.rate
```

`read.ecg.signals` *Reads WFDB-compatible signal and header files*

Description

WFDB (WaveForm DataBase) is a standard file format for storing, reading, and analyzing physiological time-series signals.It is widely used for signals such as: ECG, EEG, blood pressure, respiration and other biomedical waveforms. It was developed by PhysioNet and is common in research datasets. WFDB (WaveForm DataBase) is a standard file format for storing, reading, and analyzing physiological time-series signals. It is widely used for signals such as ECG, EEG, blood pressure, respiration, and other biomedical waveforms. It was developed by PhysioNet and is commonly used in research datasets.

Usage

```
read.ecg.signals(file)
```

Arguments

`file`	Path to the ECG record to be read.

Details

A WFDB record typically consists of two main files: `.dat` - binary signal samples (waveform values),and `.hea` - a header file describing how to interpret the data. In some cases, additional annotation files such as `.atr` may be present, containing beat labels or rhythm annotations.

Value

An object of class `ecg`. The returned value is a list containing: 1) a matrix of signals stored in the ECG file, 2) the sampling rate, 3) time stamps, 4) lead names, 5) record name.

Examples

```
# ECG data comes from https://physionet.org/content/ptb-xl/1.0.3/
file <- system.file("extdata", "00001_lr.hea", package = "MatchingPursuit")
dir <- dirname(file)
name <- tools::file_path_sans_ext(basename(file))

out <- read.ecg.signals(file)
head(out$signal)
out$sampling.rate
out$lead.names

plot(out, begin = 0, end = 10, panel.height = 1.5)
```

`read.edf.params` *Reads a selected EDF or EDF+ file and returns signal parameters*

Description

Reads a selected EDF or EDF+ file and returns basic signal parameters (channel names, sampling frequency of each channel, number of samples per channel, and signal duration in seconds). Additional information stored in EDF+ files (such as interrupted recordings or time-stamped annotations) is not used by the package and is therefore not read.

Usage

```
read.edf.params(file)
```

Arguments

`file`	Path to the EDF / EDF+ file to be read.

Value

A data frame containing the basic parameters of the EDF / EDF+ file.

Examples

```
file <- system.file("extdata", "EEG.edf", package = "MatchingPursuit")
read.edf.params(file)
```

`read.edf.signals` *Reads a selected EDF or EDF+ file and returns signal data*

Description

The function reads a selected EDF or EDF+ file. Optionally, resampling can be performed (upsampling or downsampling).

Usage

```
read.edf.signals(
  file,
  resampling = FALSE,
  f.new = NULL,
  from = NULL,
  to = NULL,
  verbose = FALSE
)
```

Arguments

`file`	Path to the EDF / EDF+ file to be read.
`resampling`	If `TRUE`, all signals are resampled (either upsampled or downsampled), depending on the original sampling rates of the channels.
`f.new`	Target sampling frequency used for upsampling or downsampling.
`from`	Starting time of the signal to be loaded (in seconds).
`to`	Ending time of the signal to be loaded (in seconds).
`verbose`	Logical flag indicating whether progress information should be printed.

Details

If `resampling = TRUE`, signals are resampled according to the target frequency specified by `f.new`. Since the EDF standard allows different sampling rates per channel, some channels may be upsampled while others are downsampled. The function does not support independent resampling of individual channels.

Value

An object of class `edf`, a list containing: 1) a data frame with all signals stored in the EDF file, 2) sampling rate after optional resampling, 3) time stamps after optional resampling, 4) signal names.

Examples

```
file <- system.file("extdata", "EEG.edf", package = "MatchingPursuit")
out1  <- read.edf.signals(file, resampling = FALSE)

lapply(out1, class)
out1$sampling.rate

out2 <- read.edf.signals(file, resampling = TRUE, f.new = 128, verbose = TRUE)

lapply(out2, class)
out2$sampling.rate
```

`read.empi.db.file` *Reads data from a SQLite file created by the Matching Pursuit algorithm*

Description

Reads data from a SQLite file (`.db`) created by the Matching Pursuit algorithm. The reconstructed signal(s) and Gabor function(s) are also returned.

Usage

```
read.empi.db.file(db.file)
```

Arguments

`db.file`	SQLite file.

Value

Object of class `empi` is returned with the following items:

- parameters of all generated atoms,
- original input signal(s),
- reconstructed signal(s) obtained as the sum of generated atoms,
- generated Gabor atoms,
- time stamps,
- sampling rate.

Examples

```
## Not run:
file <- system.file("extdata", "EEG.db", package = "MatchingPursuit")
out <- read.empi.db.file(file)

n.channels <- ncol(out$original.signal)
original.signal <- out$original.signal
reconstruction <- out$reconstruction
t <- out$t
f <- out$f

old.par <- par("mfrow", "pty", "mai")

par(mfrow = c(2, 1))
par(pty = "m")
par(mai = c(0.9, 0.5, 0.3, 0.4))

plot(
  original.signal[,1], type = "l", col = "blue",
  main = paste("channel: ", 1, " / " , n.channels, " (original signal)",  sep = ""),
  xaxt = "n", ylab = "", xlab = "time [sec]"
)

len <- length(original.signal[, 1])
lab <- seq(t[1], t[len] + 1 / f, length.out = 11)
axis(side = 1, las = 1, cex.axis = 0.9, at = seq(0, len, length.out = 11), labels = lab)

plot(
  reconstruction[,1], type = "l", col = "blue",
  main = paste("channel: ", 1, " / " , n.channels, " (reconstructed signal)",  sep = ""),
  xaxt = "n", ylab = "", xlab = "time [sec]"
)

axis(side = 1, las = 1, cex.axis = 0.9, at = seq(0, len, length.out = 11), labels = lab)

par(old.par)

## End(Not run)
```

`sig2bin` *Reads input signal(s) from a data frame and returns them in binary format*

Description

Saves the given data (signals) in binary form. The input signal(s) must be a data frame: rows correspond to samples for all channels, and columns correspond to channels. The function is used internally by `empi.execute()`. The binary data consist of floating-point values in the byte order of the current machine (no byte-order conversion is performed).

For multichannel signals, samples are written in time order: first all channels at t = 0, then all channels at t=Δt, and so on. In other words, the signal is stored in column-major order (rows = channels, columns = samples).

Usage

```
sig2bin(data, write.to.file = FALSE)
```

Arguments

data	Data frame containing the input signal(s).
write.to.file	If TRUE, a .bin file is created and saved in the cache directory.

Value

Input signal returned as raw. If write.to.file = TRUE, a .bin file is additionally created and saved in the current directory.

Note

Users do not work directly with .bin files. Binary files are used only in empi.execute(). The external program *Enhanced Matching Pursuit Implementation* (EMPI), executed inside this function, requires binary input data. This conversion utility may also be useful for users who wish to run EMPI outside of the R environment.

Examples

```
file <- system.file("extdata", "sample3.csv", package = "MatchingPursuit")
out <- read.csv.signals(file)

signal.bin <- sig2bin(data = out$signal, write.to.file = FALSE)

# We have 3 channels. The first 4 time points.
head(out$signal, 4)

# The same elements of the signal in binary (floats are stored in 4 bytes).
head(signal.bin, 48)

# After decoding to numeric.
# Of course we get the same values as in out$signal.
readBin(signal.bin[1:4], what = "numeric", size = 4, endian = "little")
readBin(signal.bin[5:8], what = "numeric", size = 4, endian = "little")
readBin(signal.bin[41:44], what = "numeric", size = 4, endian = "little")
readBin(signal.bin[45:48], what = "numeric", size = 4, endian = "little")
```

Index

atom.params, 2

clear.cache, 3

eeg.montage, 4
empi.check, 5
empi.execute, 6
empi.install, 7
empi.locate, 8
empi2tf, 8

filters.coeff, 11

gabor.fun, 13

plot.ecg, 14
plot.edf, 15
plot.empi, 17

read.csv.signals, 19
read.ecg.signals, 20
read.edf.params, 21
read.edf.signals, 22
read.empi.db.file, 23

sig2bin, 24