mhrv Documentation

Aviv Rosenberg

Nov 10, 2023

WORKING WITH THE TOOLBOX

`mhrv` is a matlab toolbox for calculating Heart-Rate Variability (HRV) metrics from both ECG signals and RR-interval time series. The toolbox works with ECG data in the PhysioNet WFDB data format.

CHAPTER

ONE

FEATURES

- **WFDB wrappers and helpers**. A small subset of the PhysioNet WFDB tools are wrapped with matlab functions, to allow using them directly from matlab. For example,
  - `mhrv.wfdb.gqrs` - A QRS detection algorithm.
  - `mhrv.wfdb.rdsamp` - For reading PhysioNet signal data into matlab.
  - `mhrv.wfdb.rdann` - For reading PhysioNet annotation data into matlab.
  - `mhrv.wfdb.wrann` - For writing PhysioNet annotation data from matlab datatypes.
  - `mhrv.wfdb.wfdb_header` - Read record metadata from a WFDB header file (`.hea`).
- **ECG signal processing**. Peak detection and RR interval extraction from ECG data in PhysioNet format. For example,
  - `mhrv.wfdb.rqrs` - Detection of R-peaks in ECG signals (based on PhysioNet's `gqrs`). Configurable for use with both human and animal ECGs.
  - `mhrv.ecg.jqrs`/`mhrv.ecg.wjqrs` - An ECG peak-detector based on a modified Pan & Tompkins algorithm and a windowed version.
  - `mhrv.ecg.bpfilt`- Bandpass filtering for removing noise artifacts from ECG signals.
  - `mhrv.wfdb.ecgrr` - Construction of RR intervals from ECG data in PhysioNet format.
  - `mhrv.wfdb.qrs_compare` - Comparison of QRS detections to reference annotations and calculation of quality measures like Sensitivity, PPV.
- **RR-intervals signal processing**. Ectopic beat rejection, frequency filtering, nonlinear dynamic and fractal analysis. For example,
  - `mhrv.rri.filtrr` - Filtering of RR interval time series to detect ectopic (out of place) beats.
  - `mhrv.rri.dfa` - Detrended Fluctuation Analysis, a method of estimating the fractal scaling exponent of a signal [3].
  - `mhrv.rri.mse` - Multiscale Sample Entropy, a measure of the complexity of the signal computed on multiple time scales [4].
  - `mhrv.rri.sample_entropy` - Sample Entropy, a measure of the irregularity of a signal.
- HRV Metrics: Calculating quantitative measures that indicate the activity of the heart based on RR intervals using all standard HRV metrics defined in the literature (see e.g. [2]).
  - `mhrv.hrv.hrv_time` - Time Domain: AVNN, SDNN, RMSSD, pNNx.
  - `mhrv.hrv.hrv_freq` - Frequency Domain:
    - Total and normalized power in (configurable) VLF, LF, HF and custom user-defined bands.

        * Spectral power estimation using Lomb, Auto Regressive, Welch and FFT methods.
        * Additional frequency-domain features: LF/HF ratio, LF and HF peak frequencies, power-law scaling exponent (beta).
    - `mhrv.hrv.hrv_nonlinear` - Nonlinear methods:
        * Short- and long-term scaling exponents (alpha1, alpha2) based on DFA.
        * Sample Entropy and Multiscale sample entropy (MSE).
        * Poincaré plot metrics (SD1, SD2).
    - `mhrv.hrv.hrv_fragmentation` - Time-domain RR interval fragmentation analysis [5].
- Configuration: The toolbox is fully configurable with many user-adjustable parameters.
    - The configuration files are in human-readable YAML format which is easy to edit and extend.
    - The user can create custom configurations files based on the `defatuls.yml` file (only overriding what's required).
    - Custom configuration files can be loaded with a single call which updates the defaults for the entire toolbox. This allows simple, reproducible analysis of different datasets that require different analysis configurations. See the `mhrv.defaults` package.
    - The settings for any of the functions can either be configured globally with configuration `yml` files or on a per-call basis with matlab-style key-value argument pairs.
- Plotting: All toolbox functions support plotting their output for data visualization. The plotting code is separated from the algorithmic code in order to simplify embedding this toolbox in other matlab applications. See the `mhrv.plots` package.
- Top-level analysis functions: These functions work with PhysioNet records and allow streamlined HRV analysis by composing the functions of this toolbox.
    - `mhrv.mhrv` - Analyzes a single PhysioNet record (ECG data or annotations), optionally split into multiple analysis windows. Extracts all supported HRV features and optionally generates plots.
    - `mhrv.mhrv_batch` - Analyzes many PhysioNet records (ECG data or annotations) which can be further separated into user-defined groups (e.g. Control, Test). Automatically computes HRV metrics per group and generates a comparative summary of the HRV features in each group.

CHAPTER

TWO

GETTING STARTED

2.1 Requirements

- Matlab with Signal Processing toolbox. Should work on Matlab R2014b or newer.
- The PhysioNet WFDB tools. The toolbox can install this for you.

2.2 Installation

1. Clone the repo or download the source code.
2. From matlab, run the `mhrv_init` function from the root of the repo. This function will:
   - Check for the presence of the WFDB tools in your system `PATH`.
   - If WFDB tools are not detected, it will attempt to automatically download them for you into the folder `bin/wfdb` under the repository root.
   - Set up your MATLAB path to include the code from this toolbox.

2.2.1 Notes about matlab's `pwd` and `path`

Matlab maintains a PWD, or "present working directory". It's the folder you see at the top of the interface, containing the files you see in the file explorer pane. Type `pwd` at the matlab command prompt to see it's value.

Additionally, matlab maintains a PATH variable, containing a list of folders in which it searches for function definitions (similar to the shell PATH concept). Type `path` at the matlab command prompt to see it's value.

You don't need to change your `pwd` to the root of the repo folder for the toolbox to work. You can simple run the `mhrv_init` function from your current `pwd`, and it will take care of updating matlab's path. For example, if you cloned or downloaded the toolbox in the folder `/Users/myname/mhrv/`, you can run the following command from the matlab prompt:

```
run /Users/myname/mhrv/mhrv_init.m
```

After this the toolbox will be ready to use, regardless of your `pwd`.

2.2.2 Manual WFDB Installation (Optional)

The above steps should be enough to get most users started. In some cases `mhrv_init` may fail to download the correct binaries for you, or you may want to install them yourself.

- On macOS, you can use homebrew. First install homebrew, then install `wfdb` with `brew tap brewsci/science && brew install wfdb`.
- On any OS (including macOS), you can compile the WFDB binaries from source using the instructions on their website.

Once you have the binaries, place them in some folder on your `$PATH` or somewhere under the repo's root folder (`bin/wfdb` would be a good choice as it's `.gitignored`) and they will be found and used automatically. You can replace the binaries that were automatically downloaded with your compiled ones. If you used homebrew, they should already be on your `$PATH`.

If you would like to manually specify a path outside the repo which contains the WFDB binaries (e.g. `/usr/local/bin` for a homebrew install), you can edit `cfg/defaults.yml` and set the `mhrv.paths.wfdb_path` variable to the desired path.

For macOS users it's recommended to install with homebrew, and for linux users it's recommended to install from source, as the binaries provided on the PhysioNet website are very outdated.

CHAPTER

THREE

MHRV

The top level analysis functions in `mhrv` can be used as a command-based user interface to the toolbox. The `mhrv()` and `mhrv_batch()` functions allow analysis of both ECG and R-peak annotation files in WFDB format and return all HRV metrics supported by the toolbox.

`mhrv.`**`mhrv_batch`**(*rec_dir*, *varargin*)

Performs batch processing of multiple records with mhrv.

This function analyzes multiple physionet records with mhrv and outputs tables containting the results. The records to analyze can be subdivided into record types, in which case output tables will be generated for each record type. Optionally, an Excel file can be generated containing the results of the analysis, including a comparison of the results per group.

Parameters

- **`rec_dir`** – Directory to scan for input files.
- **`varargin`** – Optional key-value parameter pairs.
  - rec_types: A cell array containing the names of the type of record to analyze.
  - rec_filenames: A cell array with identical length as 'rec_names', containing patterns to match against the files in 'rec_dir' for each 'rec_type'.
  - rec_transforms: A cell array of transform functions to apply to each file in each record (one transform for each rec_type).
  - ann_ext: Specify an annotation file extention to use instead of loading the record itself (.dat file). If provided, RR intervals will be loaded from the annotation file instead of from the ECG. Can also be a cell-array of strings the same length as rec types. This allows using a different annotator extension for each rec type. Default: empty string (don't use annotation).
  - min_nn: Set a minumum number of NN intervals so that windows with less will be discarded. Default is 0 (don't discard anything).
  - rr_dist_max: Sanity threshold for RR interval distribution. Windows where mean(RR)+2std(RR) > rr_dist_max will be discarded.
  - rr_dist_min: Similar to above, but will discard windows where mean(RR)-2std(RR) < rr_dist_min.
  - mhrv_params: Parameters pass into mhrv when processing each record. Can be either a string specifying the parameters file name, or it can be a cell array where the first entry is the parameters file name and the subsequent entires are key-value pairs that override parameters from the file.
  - skip_plot_data: Whether to skip saving the plot data for each record. This can reduce memory consumption significantly for large batches. Default: false.

- writexls: true/false whether to write the output to an Excel file.
- output_dir: Directory to write output file to.
- output_filename: Desired name of the output file.

Returns

A structure, batch_data, containing the following fields:

- rec_types: A cell of strings of the names of the record types that were analyzed.
- rec_transforms: A cell array of the RR transformation functis used on each record type.
- mhrv_window_minutes: Number of minutes in each analysis windows that each record was split into.
- mhrv_params: A cell array containing the value of the *params* argument passed to mhrv for the analysis (see mhrv documentation).
- hrv_tables: A map from each value in 'rec_types' to the table of HRV values for that type. - stats_tables: A map with keys as above, whose values are summary tables for each type.
- plot_datas: A map with keys as above, whose values are also maps, mapping from an individual record filename to the matching plot data object (which can be used for generating plots).

`mhrv.`**`mhrv`**`(`*rec_name*, *varargin*`)`

Analyzes an ECG signal, detects and filters R-peaks and calculates various heart-rate variability (HRV) metrics on them.

Parameters

- **`rec_name`** – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- **`varargin`** – Pass in name-value pairs to configure advanced options.
  - ecg_channel: The channel number to use (in case the record has more than one). If not provided, mhrv will attempt to use the first channel that has ECG data.
  - ann_ext: Specify an annotation file extention to use instead of loading the record itself (.dat file). If provided, RR intervals will be loaded from the annotation file instead of from the ECG. Default: empty (don't use annotation).
  - window_minutes: Split ECG signal into windows of the specified length (in minutes) and perform the analysis on each window separately.
  - window_index_offset: Number of windows to skip from the beginning.
  - window_index_limit: Maximal number of windows to process. Combined with the above, this allows control of which window to start from and how many windows to process from there.
  - params: Name of mhrv defaults file to use (e.g. 'canine'). Default '', i.e. no parameters file will be loaded. Alternatively, can also be a cell array containing the exact arguments to pass to mhrv_load_params. This allows overriding parameters from a script.
  - transform_fn: A function handle to apply to the NN intervals before calculating metrics. The function handle should accept one argument only, the NN interval lengths.
  - plot: true/false whether to generate plots. Defaults to true if no output arguments were specified.

Returns

- hrv_metrics: A table where each row is a window and each column is an HRV metrics that was calculated in that window.
- hrv_stats: A table containing various statistics about each metric, calculated over all windows.
- plot_datas: Cell array containing the plot_data structs for each window.

CHAPTER

FOUR

MHRV.ECG

Functions in this package work on raw ECG data passed in as matlab vectors.

`mhrv.ecg.ecgsqi`(*ann1*, *ann2*, *ecg*, *fs*, *varargin*)

Computes bsqi [3]_ [4]_ to estimate the signal quality of an ECG signal. The bsqi index is computed over the whole ecg recording with a granularity specified by the user.

Parameters

- **ann1** – vector of QRS annotations from a first peak detector (sample)
- **ann2** – vector of QRS annotations from a secon peak detector (sample)
- **ecg** – electrocardiogram time series (mV)
- **fs** – sampling frequency of the ecg (Hz)
- **varargin** – pass in name-value pairs to configure advanced options:
  - agw: agreement window tolerated between annotations from the two peak detectors in order for the two annotations to be considered in agreement (in seconds)
  - sw: the size of the window on which the signal quality is computed (in seconds). Default is 5 sec.
  - rw: granularity at which the sqi is computed (in seconds). By default the sqi is computed at every second.
  - mw: window for a post-processing median smoothing (in number of samples). By default there is no median post-processing.
  - thrsqi: final sqi threshold. The signal will be considered of good quality for all sqi value above this threshold (output sqi will be one for these). Conversely the signal will be considered of bad quality for all sqi value below this threshold (output sqi will be zero for these).
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- sqi: signal quality time series (nu, in range 0-1)
- tsqi: time vector for the sqi vector (sec)

Example: an ecg sample from the mitdb (physionet.org) is downloaded, preprocessed and two peak detectors are ran (wjqrs and gqrs). The two sets of R-peak annotations are used to compare the signal quality using the ecgsqi function.

```
download_wfdb_records('mitdb', '105', '.');
recordName = 'mitdb/105';
[~,ecg,Fs]=rdsamp(recordName,1);
bpfecg = bpfilt(ecg,Fs,5,45,[],0); % prefilter in range [5 - 45] Hz
anns_jqrs = wjqrs(bpfecg,Fs,0.3,0.250,10); % wjqrs peak detector
anns_gqrs = gqrs(recordName); % gqrs peak detector
anns_gqrs = double(anns_gqrs);

[ sqi, tsqi ] = ecgsqi( anns_jqrs', anns_gqrs, ecg, Fs, 'plot', true, 'mw', 1, 'sw',
↪ 5, 'rw', 1, 'agw', 0.050);
```

mhrv.ecg.**bsqi**(*refqrs*, *testqrs*, *agw*, *fs*)

This algorithm can be used to estimate the signal quality of a single channel electrocardiogram [3]_ [4]_. It compares the agreement between the R-peak annotations (ann1 and ann2) made by two different R-peak detectors. If the two detectors agree locally then the signal quality (sqi) is good and if they disagree then it is likely because of some uderlying artifacts/noise in the signal. The limitation of this method is that when one of the two detectors fails for whatever reason other than the presence of noise then the quality will be zero.

Parameters

- **refqrs** – vector of QRS annotations from a first peak detector (in seconds)
- **testqrs** – vector of QRS annotations from a secon peak detector (in seconds)
- **agw** – agreement window (sec)
- **fs** – sampling frequency (Hz)

Returns

- F1: signal quality measure (nu)
- IndMatch: indices of matching peaks (nu)
- meanDist: mean distance between matching refqrs and testqrs peaks (sec)

Example: Comparing `gqrs` to `wjqrs` on a record from `mitdb`.

```
download_wfdb_records('mitdb', '105', '.');
[tm,ecg,Fs]=rdsamp('mitdb/105',1,'from',450000,'to',480000);
bpfecg = bpfilt(ecg,Fs,5,45,[],0); % prefilter in range [5 - 45] Hz
anns_jqrs = wjqrs(bpfecg,Fs,0.3,0.250,10); % wjqrs peak detector
anns_gqrs = gqrs('mitdb/105','from',450000,'to',480000); % gqrs peak detector
anns_gqrs = double(anns_gqrs);

[F1,~] = bsqi( anns_jqrs', anns_gqrs,0.050,Fs);
plot(tm,ecg);
hold on; plot(tm(anns_jqrs),ecg(anns_jqrs),'+r');
hold on; plot(tm(anns_gqrs),ecg(anns_gqrs),'+k');
legend(['ecg with quality ' num2str(F1)],'wjqrs','gqrs');
```

mhrv.ecg.**wjqrs**(*ecg*, *fs*, *thres*, *rp*, *ws*)

This function [1]_ is used to run the jqrs peak detector using a sliding (non-overlapping) window. This is usefull in the cases where the signal contains important artefacts which could bias the jqrs threshold evaluation or if the amplitude of the ecg is changing substantially over long recordings because of the position of the electrodes move for example. In these instances the adaptation of the energy threshold is useful.

Parameters

- **ecg** – ecg signal (mV)
- **fs** – sampling frequency (Hz)
- **thres** – threshold to be used in the P&T algorithm(nu)
- **rp** – refractory period (sec)
- **ws** – window size (sec)

Returns

- qrs: qrs location in nb samples (ms)

Example: perform peak detection on an ecg recording from the mitdb (physionet.org), A refractory period of 250 ms (a standard value for Human ECG) and a threshold of 0.3 are used.

```
download_wfdb_records('mitdb', '105', '.');
[~,ecg,Fs]=rdsamp('mitdb/105',1);
bpfecg = bpfilt(ecg,Fs,4,45,[],0); % prefilter in range [4-45] Hz

anns_jqrs = wjqrs(bpfecg,Fs,0.3,0.250,10); % jqrs running on each segment of 10 sec␣
↪length
```

mhrv.ecg.**qrs_adjust**(*ecg*, *qrs*, *fs*, *inputsign*, *tol*, *debug*)

This function [1]_ is used to adjust the qrs location by looking for a local min or max around the input qrs points. For example, this is useful when parts of the qrs are located on a positive part of the R-wave and parts on a negative part of the R-wave in order to make them all positive or negative - this can be useful for heart rate variability analysis. It is also useful for template substraction in the context of non-invasive fetal ECG source separation in order to ensure the maternal template is well aligned with each beat.

Parameters

- **ecg** – vector of ecg signal amplitude (mV)
- **qrs** – peak position (sample number)
- **fs** – sampling frequency (Hz)
- **inputsign** – sign of the peak to look for (-1 for negative and 1 for positive)
- **tol** – tolerance window (sec)
- **debug** – (boolean)

Returns

- cqrs: adjusted (corrected) qrs positions (sample number)

Example:

```
download_wfdb_records('mitdb', '105', '.');
[~,ecg,Fs]=rdsamp('mitdb/105',1);
bpfecg = bpfilt(ecg,Fs,4,45,[],0); % prefilter in range [4-45] Hz
anns_jqrs = wjqrs(bpfecg,Fs,0.3,0.250,10); % jqrs running on each segment of 10 sec␣
↪length

cqrs = qrs_adjust(ecg,anns_jqrs,Fs,-1,0.050,1);
```

mhrv.ecg.**bpfilt**(*ecg*, *fs*, *lcf*, *hcf*, *nt*, *debug*)

This function [1]_ is made for prefiltering an ecg time series before it is passed through a peak detector. Of important note: the upper cut off (hcf) and lower cutoff (lcf) of the bandpass filter that are applied, highly depends on the mammal that is being considered. This is particularly true for the hcf which will be higher for a mouse ECG file versus a Human ecg. This is because the QRS is `sharper' for a mouse ecg than for a Human one. Of note, NaN should be represented by the value -32768 in the ecg (WFDB standard).

Parameters

- **ecg** – electrocardiogram (mV)
- **fs** – sampling frequency (Hz)
- **lcf** – low cut-off frequency (Hz)
- **hcf** – high cut-off frequency (Hz)
- **debug** – plot output filtered signal (boolean)
- **nt** – frequency to cut with a Notch filter (Hz). Leave the field empty ('[]') if you do not want the Notch filter to be applied.

Returns

- bpfecg: band pass filtered ecg signal

Example: preprocess an ecg recording from the mitdb (physionet.com) by applying a bandpass filter (0.5-100 Hz)

```
download_wfdb_records('mitdb', '105', '.');
[~,ecg,Fs]=rdsamp('mitdb/105',1);

bpfecg = bpfilt(ecg,Fs,0.5,100,[],1);
```

mhrv.ecg.**jqrs**(*ecg*, *fs*, *thr*, *rp*, *debug*)

Implementation of an energy based qrs detector [1]_. The algorithm is an adaptation of the popular Pan & Tompkins algorithm [2]_. The function assumes the input ecg is already pre-filtered i.e. bandpass filtered and that the power-line interference was removed. Of note, NaN should be represented by the value -32768 in the ecg (WFDB standard).

Parameters

- **ecg** – vector of ecg signal amplitude (mV)
- **fs** – sampling frequency (Hz)
- **thr** – threshold (nu)
- **rp** – refractory period (sec)
- **debug** – plot results (boolean)

Returns

- qrs_pos: position of the qrs (sample)

Example: process an ecg recording from the mitdb (physionet.org). A bandpass filter is used to preprocess the ECG in the range [4 - 45]Hz. Then the peak detector is ran with a refractory period of 250 ms (a standard value for Human ECG analysis) is specified.

```
download_wfdb_records('mitdb', '105', '.');
[~,ecg,Fs]=rdsamp('mitdb/105',1);

bpfecg = bpfilt(ecg,Fs,4,45,[],0); % prefilter in range [4-45] Hz

qrs_pos = jqrs(bpfecg,Fs,0.4,0.250,1); % peak detection and plot
```

CHAPTER FIVE

MHRV.RRI

Functions in this package work on RR-interval data passed in as matlab vectors.

`mhrv.rri.mse`(*sig*, *varargin*)

Calculates the Multiscale Entropy [2]_, MSE, of a signal, a measure of the signals complexity. The algorithms calculates the Sample Entropy of the signal at various 'scales' from 1 to max_scale. At each scale, the signal is downsampled by averaging 'scale' samples and the Sample Entropy is calculated for the downsampled signal.

Parameters

- **`sig`** – Signal to calculate MSE for.
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - mse_max_scale: Maximal scale to calculate up to. Default: 20.
  - sampen_r: Value of 'r' parameter to use when calculating sample entropy (max distance between two points that's considered a match). Default: 0.2.
  - sampen_m: Value of 'm' parameter to use when calculating sample entropy (length of templates to match). Default: 2.
  - normalize_std: Whether or not to normalize the signal to std=1 before calculating the MSE. This affects the meaning of r.
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- mse_result: The sample entropy value at each scale.
- scale_axis: The corresponding scale values that were used.

`mhrv.rri.freqfiltrr`(*rri*, *fc*, *varargin*)

Performs frequency-band filtering of RR intervals. This function can apply a low-pass or high-pass filter to an RR interval time series.

Parameters

- **`rri`** – RR-intervals values in seconds.
- **`fc`** – Filter cutoff frequency, in Hz.
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - resamp_freq: Frequency to resample the RR-intervals at before filtering. Must be at least twich the maximal frequency in the signal. Default: 5 Hz.
  - forder: Order (length in samples) of the filter to use. Default: 100.
  - ftype: A string, either 'low' or 'high', specifying the type of filter to apply.

  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- rri_out: RR intervals after filtering.
- trr_out: Times of filtered RR intervals, in seconds.

`mhrv.rri.sample_entropy`(*sig*, *m*, *r*)

Calculate sample entropy [3]_ (SampEn) of a signal. Sample entropy is a measure of the irregularity of a signal.

Parameters

- **sig** – The input signal.
- **m** – Template length in samples.
- **r** – Threshold for matching sample values.

Returns

The sample entropy value of the input signal.

`mhrv.rri.detrendrr`(*rri*, *lambda*, *Fs*, *varargin*)

A detrending method [4]_ for the RR intervals. This is used when analysing RR interval time series over a long window thus where the stationarity assumption is not valid anymore. Of note: a number of HRV methods such as the fragmentation measures and the DFA measures do not assume the intervals to be stationary. Thus usage of this detrending tool is specific to what HRV measures are being studied. Detrending will also likely affect the low frequency fluctuation information contained in the VLF band.

Parameters

- **rri** – Vector of RR-interval dirations (seconds)
- **lambda** – lambda (lambda=10 for Human)
- **Fs** – the sampling frequency of the original ECG signal (Hz)
- **varargin** – Pass in name-value pairs to configure advanced options:
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified (boolean).

Returns

- rri_detrended: detrended rri interval (seconds)

Example: process an ecg recording sampled at 1000 Hz and specifying a refractory period of 250 ms (a standard value for Human ECG) and with a threshold of 0.5.

```
recordName = 'mitdb/105';
[ecg,fs,~] = rdsamp(recordName,1);
bpfecg = bpfilt(ecg,fs,5,45,[],0); % prefilter in range [5-45] Hz
anns_jqrs = wjqrs(bpfecg,fs,0.3,0.250,10); % jqrs running on each segment of 10 sec␣
↪length
z = diff(anns_jqrs)/fs; % get the RR intervals

nni_detrend = detrend_nn(z',10,'Fs',Fs ,'plot',true); % detrend and plot
```

`mhrv.rri.poincare`(*rri*, *varargin*)

Calculates HRV metrics from a Poincaré plot of the input data.

Parameters

- **rri** – Row vector of RR-interval lengths in seconds.
- **varargin** – Pass in name-value pairs to configure advanced options:
  - sd1_factor: Factor to multiply the standard devation along the perpendicular line (SD1) to get the radius of the ellipse along that axis. Default is 2 (so over 95% of points will be inside the ellipse).
  - sd2_factor: As above, but for the standard deviation along the line of identity (SD2). Default: 3.
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- sd1: Standard deviation of RR intervals along the axis perpendicular to the line of identity.
- sd2: Standard deviation of RR intervals along the line of identity.

`mhrv.rri.`**`dfa`**(*t*, *sig*, *varargin*)

Detrended fluctuation analysis, DFA [1]_. Calculates the DFA of a signal and it's scaling exponents α_1 and α_2.

Parameters

- **t** – time (or x values of signal)
- **sig** – signal data (or y values of signal)
- **varargin** – Pass in name-value pairs to configure advanced options:
  - n_min: Minimal DFA block-size (default 4)
  - n_max: Maximal DFA block-size (default 64)
  - n_incr: Increment value for n (default 2). Can also be less than 1, in which case we interpret it as the ratio of a geometric series on box sizes (n). This should produce box size values identical to the PhysioNet DFA implmentation.
  - alpha1_range: Range of block size values to use for calculating the α_1 scaling exponent. Default: [4, 15].
  - alpha2_range: Range of block size values to use for calculating the α_2 scaling exponent. Default: [16, 64].

Returns

- n: block sizes (x-axis of DFA)
- fn: DFA value for each block size n
- alpha1: Exponential scaling factor
- alpha2: Exponential scaling factor

`mhrv.rri.`**`filtrr`**(*rri*, *trr*, *varargin*)

Calculate an NN-interval time series (filtered RR intervals). Performs outlier detection and removal on RR interval data. This function can perform three types of different outlier detection, based on user input: Range based detection, moving-average filter-based detection and quotient filter based detection.

Parameters

- **rri** – RR-intervals values in seconds.
- **trr** – RR-interval times in seconds.
- **varargin** – Pass in name-value pairs to configure advanced options:

  - filter_range: true/false whether to use range filtering (remove intervals smaller or larger than 'rr_min' and 'rr_max').
  - filter_ma: true/false whether to use a moving-average filter to detect potential outliers in the rr-intervals. If an interval is greater (abs) than 'win_percent' percent of the average in a window of size 'win_samples' around it, excludes the interval.
  - filter_quotient: true/false whether to use the quotient filter. This fliter checks the quotient between an interval and it's predecessor and successors, and only allows a configured change percentage ('rr_max_change') between them.
  - win_samples: Number of samples in the filter window on each side of the current sample (total window size will be 2*win_samples+1). Default: 10.
  - win_percent: The percentage above/below the average to use for filtering. Default: 20.
  - rr_min: Min physiological RR interval, in seconds. Intervals shorter than this will be removed prior to poincare plotting. Default: 0.32 sec.
  - rr_max: Max physiological RR interval, in seconds. Intervals longer than this will be removed prior to poincare plotting. Default: 1.5 sec.
  - rr_max_change: Maximal change, in percent, allowed between adjacent RR intervals. Intervals violating this will be removed prior to poincare plotting. Default: 25.
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- nni: NN-intervals (RR intervals after removing outliers) values in seconds
- tnn: NN-interval times in seconds

`mhrv.rri.`**`ansrr`**(*rri*, *freqs*, *varargin*)

Add Noised Sines to RR interval time series. This function generates a signal of sines embedded in gaussian white noise and adds it to an RR interval time series.

Parameters

- **`rri`** – RR-intervals values in seconds.
- **`freqs`** – Vector containing desired sine frequencies, in Hz.
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - var_r: Desired variance of the RR intervals. Can be used the scale the intervals before adding the noised sines. Leave empty to forgo scaling.
  - mix: Mixture ratio between the given RR intervals and the generated sines. Default: 0.5 i.e. sines will have alf the variance of the RR intervals (after scaling).
  - snr: Signal to Noise ratio of the white gaussian noise that will be added to the sines. Default: 2.

Returns

- rri_out: RR intervals after adding noised sines.

`mhrv.rri.`**`freqband_detect`**(*pxx_dataset*, *varargin*)

FREQBAND_DETECT Detect frequency bands from multiple specrums of RR intervals This function uses an automated clustering algorithm to attempt to detect bands with high frequency power from a dataset containing multiple power spectrums obtained from RR intervals.

CHAPTER

SIX

MHRV.HRV

This package provides high-level HRV analysis functionality.

`mhrv.hrv.`**`hrv_nonlinear`**`(`*nni*, *varargin*`)`

Calcualtes non-linear HRV metrics based on Poincaré plots, detrended fluctuation analysis (DFA) [2]_ and Multiscale Entropy (MSE) [3]_.

Parameters

- **`nni`** – RR/NN intervals, in seconds.
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - mse_max_scale: Maximal scale value that the MSE will be calculated up to.
  - mse_metrics: Whether to output MSE at each scale as a separate metric.
  - sampen_r: `r` value used to calculate Sample Entropy
  - sampen_m: `m` value used to calculate Sample Entropy
  - plot: true/false whether to generate plots. Defaults to true if no output arguments were specified.

Returns

- hrv_nl: Table containing the following HRV metrics:
  - SD1: Poincare plot SD1 descriptor (std. dev. of intervals along the line perpendicular to the line of identity).
  - SD2: Poincare plot SD2 descriptor (std. dev. of intervals along the line of identity).
  - alpha1: Log-log slope of DFA in the low-scale region.
  - alpha2: Log-log slope of DFA in the high-scale region.
  - SampEn: The sample entropy.

`mhrv.hrv.`**`hrv_freq`**`(`*nni*, *varargin*`)`

NN interval spectrum and frequency-domain HRV metrics This function estimates the PSD (power spectral density) of a given nn-interval sequence, and calculates the power in various frequency bands.

Parameters

- **`nni`** – RR/NN intervals, in seconds.
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - methods: A cell array of strings containing names of methods to use to estimate the spectrum. Supported methods are:
    - `lomb`: Lomb-scargle periodogram.

    * `ar`: Yule-Walker autoregressive model. Data will be resampled. No windowing will be performed for this method.
    * `welch`: Welch's method (overlapping windows).
    * `fft`: Simple fft-based periodogram, no overlap (also known as Bartlett's method).

  In all cases, a window will be used on the samples according to the `win_func` parameter. Data will be resampled for all methods except `lomb`. Default value is `{'lomb', 'ar', 'welch'}`.

- time_intervals: specify the time interval vector tnn. If it is not specified then it will be computed from the nni time series.
- power_methods: The method(s) to use for calculating the power in each band. A cell array where each element can be any one of the methods given in 'methods'. This also determines the spectrum that will be returned from this function (pxx). Default: First value in `methods`.
- norm_method: A string, either `total` or `lf_hf`. If `total`, then the power in each band will be normalized by the total power in the entire frequency spectrum. If `lf_hf`, then only for the LF and HF bands, the normalization will be performed by the (LF+HF) power. This is the standard method used in many papers to normalize these bands. In any case, VLF and user-defined custom bands are normalized by total power.
- band_factor: A factor that will be applied to the frequency bands. Useful for shifting them linearly to adapt to non-human data. Default: 1.0 (no shift).
- vlf_band: 2-element vector of frequencies in Hz defining the VLF band. Default: [0.003, 0.04].
- lf_band: 2-element vector of frequencies in Hz defining the LF band. Default: [0.04, 0.15].
- hf_band: 2-element vector of frequencies in Hz defining the HF band. Default: [0.15, 0.4].
- extra_bands: A cell array of frequency pairs, for example `{[f_start,f_end], ...}`. Each pair defines a custom band for which the power and normalized power will be calculated.
- window_minutes: Split intervals into windows of this length, calcualte the spectrum in each window, and average them. A window funciton will be also be applied to each window after breaking the intervals into windows. Set to [] if you want to disable windowing. Default: 5 minutes.
- detrend_order: Order of polynomial to fit to the data for detrending. Default: 1 (i.e. linear detrending).
- ar_order: Order of the autoregressive model to use if `ar` method is specific. Default: 24.
- welch_overlap: Percentage of overlap between windows when using Welch's method. Default: 50 percent.
- win_func: The window function to apply to each segment. Should be a function that accepts one parameter (length in samples) and returns a window of that length. Default: @hamming.
- plot: true/false whether to generate plots. Defaults to true if no output arguments were specified.

Returns

- hrv_fd: Table containing the following HRV metrics:
  - TOTAL_POWER: Total power in all three bands combined.
  - VLF_POWER: Power in the VLF band.
  - LF_POWER: Power in the LF band.
  - HF_POWER: Power in the HF band.
  - VLF_NORM: 100 * Ratio between VLF power and total power.
  - LF_NORM: 100 * Ratio between LF power and total power or the sum of LF and HF power (see 'norm_method'). - HF_NORM: 100 * Ratio between HF power and total power or the sum of LF and HF power (see `norm_method`).
  - LF_TO_HF: Ratio between LF and HF power.
  - LF_PEAK: Frequency of highest peak in the LF band.
  - HF_PEAK: Frequency of highest peak in the HF band.
  - BETA: Slope of log-log frequency plot in the VLF band.

  Note that each of the above metrics will be calculated for each value given in `power_methods`, and their names will be suffixed with the method name (e.g. LF_PEAK_LOMB).
- pxx: Power spectrum. It's type is determined by the first value in `power_methods`.
- f_axis: Frequencies, in Hz, at which pxx was calculated.

`mhrv.hrv.`**`hrv_time`**(*nni*, *varargin*)

Calculates time-domain HRV mertics from NN intervals.

Parameters

- **`nni`** – Vector of NN-interval dirations (in seconds)
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - pnn_thresh_ms: Optional. Threshold NN interval time difference in milliseconds (for the pNNx HRV measure).
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

Table containing the following HRV metrics:

- AVNN: Average NN interval duration.
- SDNN: Standard deviation of NN interval durations.
- RMSSD: Square root of mean summed squares of NN interval differences.
- pNNx: The percentage of NN intervals which differ by at least x (ms) (default 50) from their preceding interval. The value of x in milliseconds can be set with the optional parameter 'pnn_thresh_ms'.
- SEM: Standard error of the mean NN interval length.

`mhrv.hrv.`**`hrv_fragmentation`**(*nni*, *varargin*)

Computes HRV fragmentation indices [1]_ of a NN interval time series.

Parameters
nni – Vector of NN-interval dirations (in seconds)

Returns

Table containing the following fragmentation metrics:

- PIP: Percentage of inflection points.
- IALS: Inverse average length of segments.
- PSS: Percentage of NN intervals that are in short segments.
- PAS: Percentage of NN intervals that are in alternation segments of at least 4 intervals.

CHAPTER

SEVEN

MHRV.WFDB

This package provides wrappers for PhysioNet *wfdb* tools and custom functions which work with data in the PhysioNet format.

`mhrv.wfdb.gqrs`(*rec_name*, *varargin*)

Wrapper for WFDB's 'gqrs' and 'gqpost' tools. Finds the onset of QRS complexes in ECG signals given in PhysioNet format and returns them as a MATLAB vector.

Parameters

- `rec_name` – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- `varargin` – Pass in name-value pairs to configure advanced options:
  - ecg_channel: Number of ecg signal in the record (default [], i.e. auto-detect signal).
  - gqconf: Filename or Path to a gqrs config file to use. This allows adapting the algorithm for different signal and/or animal types (default is '', i.e. no config file). Note that if only a filename is provided, 'gqrs' will attempt to find the gqconf file on the MATLAB path.
  - gqpost: Whether to run the 'gqpost' tool to find erroneous detections (default false).
  - from: Number of first sample to start detecting from (default 1)
  - to: Number of last sample to detect until (default [], i.e. end of signal)

Returns

- qrs: Vector of sample numbers where the an onset of a QRS complex was found.
- outliers: Vector of sample numbers which were marked by gqpost as suspected false detections.

Note: If no output variables are given to the function call, the detected ECG signal and QRS complexes will be plotted in a new figure.

`mhrv.wfdb.get_wfdb_tool_path`(*tool_name*, *base_search_path*)

Returns the path to a wfdb tool, takes OS into account Looks for the given tool recursively under the current MATLAB directory (or a given directory), and then recursively under the folders in the `$PATH` environment variable. In case the tool is found, the containing directory path will be persisted to speed up the next search.

Parameters

- `tool_name` – A string containg the name of the wfdb tool, e.g. 'gqrs', 'wfdb-config', 'rdsamp' etc. Should not include a file extension.

- `base_search_path` – Optional. An absolute path to use as a base for searching for the tool. If not provided defaults to pwd().

Returns

- tool_path - The path of the wfdb tool, including it's os-specific file extension (e.g. .exe). In case the tool wasn't found, an error will be raised.

`mhrv.wfdb.`**`rdann`**(*rec_name*, *ann_ext*, *varargin*)

Wrapper for WFDB's 'rdann' tool. Reads annotation files in PhysioNet format and returns them as a MATLAB vector.

Parameters

- `rec_name` – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- `ann_ext` – Extension of annotation file. E.g. use 'qrs' is the annotation file is mitdb/100.qrs.
- `varargin` – Pass in name-value pairs to configure advanced options:
  - 'ann_types': A double-quoted string of PhysioNet annotation types that should be read, e.g. '"N|"' to read both annotations of type 'N' and type '|'. Default is empty, i.e. return annotations of any type.
  - from: Number of first sample to start detecting from (default 1)
  - to: Number of last sample to detect until (default [], i.e. end of signal)
  - plot: Whether to plot the all the channels and annotations in the file. Useful for debugging.

Returns

- ann: A Nx1 vector with the sample numbers that have annotations.
- ann_types: A Nx1 cell array with annotation types (strings, see PhysioNet documentation).

`mhrv.wfdb.`**`wrann`**(*rec_name*, *ann_ext*, *ann_idx*, *varargin*)

Wrapper for WFDB's 'wrann' tool. Write annotation files in PhysioNet format given a MATLAB vector.

Parameters

- `rec_name` – Path and name of a wfdb record's files e.g. db/mitdb/100. If a header file for the record doesn't exist one will be created (but fs must be specified in varargin).
- `ann_ext` – Extension of annotation file to write. E.g. use 'qrs' to write the annotation file mitdb/100.qrs.
- `ann_idx` – A column vector of integer type containing sample indices of the annotations.
- `varargin` – Pass in name-value pairs to configure %advanced options:
  - fs: The sampling frequency of the signal which is being annotated. Pass this in if writing annotations for a record which doesn't exist (i.e. a header file should be created).
  - comments: A cell array of strings which will bea written to the header file as comments (one per line). Will only be written when a new header file is craeted by this function.
  - type: Either a single character that will be used as the type for all annotations, or a cell array the same size 'ann_idx' containg a different annotation type per sample.
  - sub: Either a single number (-128 ~ 128) that will be used as the subtype attribute for all annotations, or a column vector the same size as 'ann_idx' containg a different subtype per sample.

  - chan: Either a single number (-128 ~ 128) that will be used as the chan attribute for all annotations, or a column vector the same size as 'ann_idx' containg a different chan per sample.
  - num: Either a single number (-128 ~ 128) that will be used as the num attribute for all annotations, or a column vector the same size as 'ann_idx' containg a different num per sample.
  - aux: Either a single string that will be used as the aux attribute for all annotations, or a string cell array the same size as 'ann_idx' containg a different aux per sample.

Returns
A cell array with the paths of files that were created.

`mhrv.wfdb.`**`wfdb_header`**`(`*rec_name*`)`

Returns metadata about a WFDB record based on it's header file.

Parameters
`rec_name` – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.

Returns

A struct with the following fields:

- rec_name: The record name
- Fs: Sampling frequency
- N_samples: Number of samples
- N_channels: Number of channels (different signals) in the record
- channel_info: A cell array of length N_channels with the metadata about each channel
- duration: A struct with the fields h,m,s,ms corresponding to duration fields - hours, miutes, seconds, milliseconds.
- total_seconds: Records total duration in seconds.

Note: If no output arguments are given, prints record and channel info to console.

`mhrv.wfdb.`**`get_signal_channel`**`(`*rec_name*, *varargin*`)`

Find the channel of a signal in the record matching a description. By default, if no description is specified it looks for ECG signal channels.

Parameters

- **`rec_name`** – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - sig_regex: A regular expression that should match the desired signal's description in the header file.
  - comment_regex: A regular expression that matches the comment format in the header file.

Returns

- chan: Number of the first channel in the signal that matches the description regex, or an empty array if no signals match.

- Fs: Sampling frequency
- N: Number of samples

`mhrv.wfdb.signal_duration(N, Fs)`

SIGNAL_DURATION Calculates the duration of a signal. Input

N: Number of samples Fs: Sampling frequency

Output:

t_max: total duration in seconds. h: Hours component m: Minutes component s: Seconds component ms: Milliseconds component

`mhrv.wfdb.qrs_compare_set(set_dir, ann_ext, varargin)`

Compares reference QRS detections to test detections on a set of wfdb records.

Parameters

- `set_dir` – directory path containing the wfdb files and annotations
- `ann_ext` – file extension of the annotation files.
- `varargin` – Pass in name-value pairs to configure advanced options:
  - any parameter supported by qrs_compare() can be passed to this function.
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- sqis: A table containing quality indices for each of the input files.
- stats: A table containing the Mean and Gross values for the quality indices over all files.

`mhrv.wfdb.ecgrr(rec_name, varargin)`

Calculate an RR-interval time series from PhysioNet ECG data. Detects QRS in a given sigal and calculates the RR intervals.

Parameters

- `rec_name` – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- `varargin` – Pass in name-value pairs to configure advanced options:
  - ann_ext: Specify an annotation file extention to use instead of loading the record itself (.dat file). If provided, RR intervals will be loaded from the annotation file instead of from the ECG. Default: empty (don't use annotation).
  - ecg_channel: Number of ecg signal in the record (default [], i.e. auto-detect signal).
  - from: Number of first sample to start detecting from (default 1)
  - to: Number of last sample to detect until (default [], i.e. end of signal)
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- rri: RR-intervals values in seconds.
- trr: RR-interval times in seconds.

`mhrv.wfdb.rqrs`(*rec_name*, *varargin*)

R-peak detection in ECG signals, based on `gqrs` and `gqpost`. `rqrs` Finds R-peaks in PhysioNet-format ECG records. It uses the `gqrs` and `gqpost` programs from the PhysioNet WFDB toolbox, to find the QRS complexes. Then, it searches forward in a small window to find the R-peak.

Parameters

- **rec_name** – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- **varargin** – Pass in name-value pairs to configure advanced options:
  - ecg_channel: Number of ecg signal in the record (default [], i.e. auto-detect signal).
  - gqconf: Path to a gqrs config file to use. This allows adapting the algorithm for different signal and/or animal types (default is '', i.e. no config file).
  - gqpost: Whether to run the 'gqpost' tool to find and remove possibly erroneous detections.
  - from: Number of first sample to start detecting from (default 1)
  - to: Number of last sample to detect until (default [], i.e. end of signal)
  - window_size_sec: Size of the forward-search window, in seconds.
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- qrs: Vector of sample numbers where the an onset of a QRS complex was found.
- tm: Time vector (x-axis) of the input signal.
- sig: The input signal values.
- Fs: The input signals sampling frequency.

`mhrv.wfdb.rdsamp`(*rec_name*, *varargin*)

Wrapper for WFDB's 'rdsamp' tool. Reads channels in PhysioNet data files and returns them in a MATLAB matrix.

Parameters

- **rec_name** – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- **varargin** – Pass in name-value pairs to configure advanced options:
  - chan_list: A list of channel numbers (starting from 1) to read from the record, e.g. to read the first three channels use [1, 2, 3]. Default is [], i.e. read all channels from the record.
  - from: Number of first sample to start detecting from (default 1)
  - to: Number of last sample to detect until (default [], i.e. end of signal)

Returns

- t: A vector with the sample times in seconds.
- sig: A matrix where is column is a different channel from the signal.
- Fs: The sampling frequency of the data.

`mhrv.wfdb.`**`qrs_compare`**`(`*rec_name*, *varargin*`)`

Compare R-peak detection algorithm to annotations. qrs_compare can run a r-peak detector on a given physionet record and compare the detections to an annotation file. The function assumes that the annotation file has the same record name, with a user-configurable file extension. The function supports both wfdb format and matlab's 'mat' format for the annotation files. The comparison of the detected QRS locations to the reference annotations is performed by calculating a joint-accuracy measure (F1), based on the Sensitivity (SE) and Positive-predictivity (PPV) of the test detector's annotations [1]_.

Parameters

- **`rec_name`** – Path and name of a wfdb record's files e.g. db/mitdb/100 if the record files (both 100.dat and 100.hea) are in a folder named 'db/mitdb' relative to MATLABs pwd.
- **`varargin`** – Pass in name-value pairs to configure advanced options:
  - tolerance: Threshold tolerance time, in seconds, for two peak detections to be considered equal.
  - ann_ext: Extension of annotation file.
  - ann_format: Format of annotation file. Can be `wfdb` or `mat`.
  - ecg_channel: Channel number of ecg signal in the record (default [], i.e. auto-detect signal).
  - qrs_detector: Name of qrs detector to use. Can be `rqrs` or `gqrs`.
  - plot: true/false whether to generate a plot. Defaults to true if no output arguments were specified.

Returns

- sqi: Signal quality indices for the comparison between the detector and annotations.

`mhrv.wfdb.`**`download_wfdb`**`(`*dest_base_dir*`)`

Downloads the WFDB binaries for this OS. This function detects the current OS and attempts to download the approprate WFDB binaries. By default they will be downloaded into the folder 'bin/wfdb' under the current MATLAB directory.

Parameters

`dest_base_dir` – Optional. Directory to download into. Will be created if it doesn't exist. A folder name 'wfdb' will be created inside. If this is not provided, defaults to the folder 'bin/' under the current MATLAB directory.

Returns

- bin_path: Path the the directory containing the WFDB binaries that were downloaded.

`mhrv.wfdb.`**`isrecord`**`(`*rec_name*, *data_ext*`)`

Checks if the given WFDB record name exists locally.

Parameters

- **`rec_name`** – Path and name of a wfdb record's files e.g. 'db/mitdb/100'. Can be an absolute or relative path (relative to the MATLAB pwd).
- **`data_ext`** – Optional. The extension of the data file to look for. Defaults to 'dat' if not specfied.

Returns

- isrecord: True if the given record path is valid, otherwise false. Valid means that e.g. both the files db/mitdb/100.dat (or another extension as specified in 'data_ext') and db/mitdb/100.hea exist if rec_name was 'db/mitdb/100').

mhrv.wfdb.**download_wfdb_records**(*db_name*, *rec_names*, *outdir*, *varargin*)

Downloads records from the PhysioBank database on PhysioNet.

Parameters

- **db_name** – Name of database on physiobank, e.g. mitdb.
- **rec_names** – A string or cell of strings containing a regex pattern to match against the record names from the specified database. Matching will be performed against the entire record name (as if regex is delimited by ^$). Empty array or string will match all records. See examples below.
- **outdir** – The root directory to download to. A folder with the specified db_name will be created within it.
- **varargin** – Pass in name-value pairs to configure advanced options:
  - base_url: Specify an alternative physiobank URL to download from.

Returns

- dl_recs: Names of downloaded records.
- dl_ann: Names of annotators for downloaded records.
- dl_files: Paths to all downloaded files.

Examples:

1. Download single record, mitdb/100 to folder db/mitdb:

```
download_wfdb_records('mitdb', '100', 'db');
```

2. Download three specific records from mitdb:

```
download_wfdb_records('mitdb', {'100','200','222'}, 'db');
```

3. Download all records starting with '1' (e.g. 100, 101, 122...) from mitdb:

```
download_wfdb_records('mitdb', '1\d+', 'db');
```

4. Download records '123', '124' and any record ending with 0 from mitdb:

```
download_wfdb_records('mitdb', {'12[3,4]', '\d+0'}, 'db');
```

5. Download all records from mitdb:

```
download_wfdb_records('mitdb', [], 'db');
```

CHAPTER EIGHT

MHRV.UTIL

This package provides utility functions used by the toolbox.

`mhrv.util.`**`fig_print`**`(`*fig_handle*, *out_filename*, *varargin*`)`

FIG_PRINT Prints a figure to file. This function sets various figure properties and then prints it to a specified format. Default format is EPS which can then be imported into a LaTeX document and will maintain it's properties. Note that A4 size is 21.0 x 29.7 cm.

`mhrv.util.`**`vercmp`**`(`*ver1*, *ver2*`)`

VERCMP Compares two version strings Compares two strings of version numbers, e.g. '1.2.3' and '1.23.4'. Returns an integer similar to strcmp:

-1 if ver1 < ver2
0 if ver1 == ver2 1 if ver1 > ver2

`mhrv.util.`**`freqband_power`**`(`*pxx*, *f_axis*, *f_band*`)`

FREQBAND_POWER Calculates the power in a frequency band

`mhrv.util.`**`table_transpose`**`(`*input_table*`)`

TABLE_TRANSPOSE Transposes a table Exchanges between a table's rows and columns, while preserving their names.

input_table: Table to transpose. transposed_table: Transposed table.

`mhrv.util.`**`table_stats`**`(`*input_table*`)`

TABLE_STATS Calcultes statistics of each variable (column) in a table. Calculates Mean, Standard Error and Median values for each variable (column) in a table.

input_table: A table containing numeric data. stats_table: A new table with the same variables as the input, but with a row for each of

the calculated statistics.

CHAPTER

NINE

MHRV.DEFAULTS

This package provides functionality to get and get toolbox default values.

mhrv.defaults.**mhrv_save_defaults**(*output_filename*)

Save the current default values of all parameters defined in the toolbox to a file.

Usage:

```
mhrv_save_defaults <output_filename>
```

This function saves the current default values of all parameters in the toolbox to a specified output file. The file will be in YAML format. If the output_filename parameters doesn't specify a `.yml` extension, it will be added.

mhrv.defaults.**mhrv_get_default**(*param_id*, *meta*)

Returns the default value configured for a parameter. This function attemps to get the value of a parameter as configured by a user config file. If it can't find a user-specified default for the parameter, it throws an error.

Parameters

- **param_id** – Unique id of the parameter This is made up of the keys in the defaults file leading to the parameter (not including the 'value' key), joined by a '.' character (example: 'hrv_freq.methods').
- **meta** – Optional. A fieldname to return from the parameter structure (instead of the structure itself). Can be value/description/name/units.

Returns

A structure containing the following fields:

- value: The user-configured parameter value, if exists, otherwise returns the 'default_value'.
- name: The user-friendly/display name of the parameter.
- description: A description about the parameter.
- units: The units the parameter is specified in.

Note: If a value for 'meta' was specified, only the corresponding field will be returned.

mhrv.defaults.**mhrv_set_default**(*varargin*)

Sets (overrides) current default parameter value(s) with new values.

Usage:

```
mhrv_set_default('param1', value1, 'param2', value2, ...)
```

This function allows overriding specific parameters with custom values given to the function. The input should consist of key-value pairs, where the keys use the '.' character to separate heirarchy levels (e.g. 'rqrs.gqconf').

`mhrv.defaults.`**`mhrv_load_defaults`**`(`*varargin*`)`

Loads an mhrv defaults file, setting it's values as default for all toolbox functions. Optionally, all current parameter defaults will be cleared.

Usage:

```
mhrv_load_defaults [--clear]
mhrv_load_defaults [--clear] <defaults_filename>
mhrv_load_defaults(defaults_filename, 'param1', value1, 'param2', value2, ...)
```

This function loads the parameter values from the default mhrv parameters file and sets them as the default value for the various toolbox functions.

The second usage form loads the parameter values from an arbitrary mhrv parameters file (.yml). The given file can be a name of a file on the matlab path, or, if it's not found there, it will be interpreted as a path (absolute or relative to pwd).

The third usage form also allows overriding or adding specific parameters with custom values given to the function. In this form, the filename is optional; the function will also accept just key-value pairs.

Note: This function clears all current default parameters if the '–clear' option is given. Otherwise, it merges the loaded values with the previously existing parameter defaults.

`mhrv.defaults.`**`mhrv_parameter`**`(`*value*, *description*, *name*, *units*`)`

Creates a parameter structure for use with the mhrv toolbox.

Parameters

- **`value`** – The parameter's value. Can be any matlab object.
- **`description`** – Informative description of the parameter.
- **`name`** – User friendly/display name of the parameter.
- **`units`** – Parameters units.

Note: All inputs are optional and default to an empty string if not provided.

Returns

An object representing the parameter. Can be added to the defaults with the mhrv_set_default or mhrv_load_defaults functions.

`mhrv.defaults.`**`mhrv_get_all_defaults`**`(`*params_struct*`)`

Returns all parameter default values of the mhrv toolbox in a map.

Parameters

`params_struct` – Optional. The parameter structure to work on. If not provided, this function will search traverse the globally defined toolbox parameters.

The returned map contains keys that correspond the the unique id's of parameters, e.g `dfa.n_max`. The map's values are structures containing the value of the parameter and metadata fields.

CHAPTER TEN

MHRV.PLOTS

This package provides plotting capabilities.

mhrv.plots.**plot_ecgrr**(*ax*, *plot_data*, *varargin*)

PLOT_ECGRR Plots filtered RR intervals from ecgrr. ax: axes handle to plot to. plot_data: struct returned from ecgrr.

mhrv.plots.**default_axes_tag**(*func_name*)

DEFAULT_AXES_TAG Default Tag for axes. func_name: Name of the plotting function.

mhrv.plots.**plot_hrv_time_hist**(*ax*, *plot_data*, *varargin*)

PLOT_HRV_TIME_HIST Plots the hrv_time intervals histogram. ax: axes handle to plot to. plot_data: struct returned from hrv_time.

mhrv.plots.**plot_dfa_fn**(*ax*, *plot_data*, *varargin*)

PLOT_DFA_FN Plots the DFA F(n) function. ax: axes handle to plot to. plot_data: struct returned from dfa.

mhrv.plots.**plot_poincare_ellipse**(*ax*, *plot_data*, *varargin*)

PLOT_POINCARE_ELLIPSE Plots a poincare plot and an ellipse fit. ax: axes handle to plot to. plot_data: struct returned from poincare.

mhrv.plots.**plot_filtrr**(*ax*, *plot_data*, *varargin*)

PLOT_FILTRR Plots filtered RR intervals from filtrr. ax: axes handle to plot to. plot_data: struct returned from filtrr.

mhrv.plots.**plot_hrv_freq_beta**(*ax*, *plot_data*, *varargin*)

PLOT_HRV_NL_BETA Plots the slope of the log-log spectrum in the VLF range. ax: axes handle to plot to. plot_data: struct returned from hrv_freq.

mhrv.plots.**plot_mse**(*ax*, *plot_data*, *varargin*)

PLOT_MSE Plots the MSE function. ax: axes handle to plot to. plot_data: struct returned from dfa.

mhrv.plots.**plot_hrv_freq_spectrum**(*ax*, *plot_data*, *varargin*)

PLOT_HRV_FREQ_SPECTRUM Plots the spectrums generated by hrv_freq. ax: axes handle to plot to. plot_data: struct returned from hrv_freq.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

MATLAB MODULE INDEX

INDEX