signalanalysis

Release 0.1

Philip Gemmell

Oct 12, 2021

CONTENTS:

signalanalysis is a library including various tools for the reading, analysis and plotting of ECG and VCG data. It is designed to be as agnostic as possible for the types of data that it can read. Currently, it can read ECG data from:

1. CARP simulations of whole torso activity, using existing projects from *CARPutil <https://git.opencarp.org/openCARP/carputils>*;
2. .csv and .dat records;
3. wfdb file formats, using *wfdb-python <https://github.com/MIT-LCP/wfdb-python>*

Futher details of how to use these functions are in *Usage*, with the finer points within the files themselves.

CHAPTER

ONE

USAGE

1.1 Installation & Getting Started

To use signalanalysis, it is highly recommended to install using `pipenv`, the virtual environment, to ensure that dependencies are where possible maintained. Clone the repository, then install the requirements as follows:

```
user@home:~$ git clone git@github.com:philip-gemmell/signalanalysis.git
user@home:~$ cd signalanalysis
user@home:~/signalanalysis$ pipenv install
```

Once the repository is cloned, it is currently the case that all work must be done within the Python3 environment. However, it is recommended to use the virtual environment from pipenv rather than the system-wide Python3 (after entering a pipenv shell, it is quit using the `exit` command as shown)

```
user@home:~/signalanalysis$ pipenv shell
(signalanalysis) user@home:~/signalanalysis$ python3
>>> import signalanalysis
>>> quit()
(signalanalysis) user@home:~/signalanalysis$ exit
user@home:~/signalanalysis$
```

The project is arranged into various subdivisions. The required analysis/plotting packages for the ECG/VCG are separated out, and require separate importing. See the next section, and individual help files for further details.

- signalanalysis
  - signalanalysis.general
  - signalanalysis.ecg
  - signalanalysis.vcg
- signalplot
- tools

1.2 Reading ECG/VCG data

Currently, only ECG reading is supported; VCG data is calculated from ECG data using the Kors method (see method). ECG files are read upon the instantiation of the ECG class, though it is possible to re-read data if required for some reason.

```
>>> import signalanalysis.ecg
>>> import signalanalysis.vcg
>>> ecg_example = signalanalysis.ecg.Ecg("filename")
>>> vcg_example = signalanalysis.vcg.Vcg(ecg_example)
```

1.3 Shifting to classes from methods

Previously, all ECG/VCG data was extracted and stored in DataFrames, and most of the modules in this code currently support this format. However, it is planned to shift the main focus of the project to use classes, which allow encapsulation of linked data in one data structure. While the focus of this documentation will be future-facing, and look at using the classes, note that sometimes access to the raw, underlying DataFrames will still be required. To that end, the raw data can be accessed as the .data attribute:

```
>>> ecg_class = signalanalysis.ecg.Ecg("filename") # Returns an Ecg class
>>> vcg_class = signalanalysis.vcg.Vcg(ecg_class)  # Returns a Vcg class
>>> ecg_data = ecg_class.data                      # Returns a Pandas DataFrame of the␣
↪underlying data
>>> vcg_data = vcg_class.data                      # Returns a Pandas DataFrame of the␣
↪underlying data
```

CHAPTER

TWO

DOCUTILS DOCUMENTATION

This is the index of all modules and their associated methods and classes, documenting their use, parameters and return values. See *Usage* for a more step-by-step introduction to the intended use cases.

Broadly speaking, the modules are split thus:

- `signalanalysis` covers the analysis scripts for ECG/VCG analysis (e.g. calculating QRS duration)
- `signalplot` covers plotting methods (e.g. plotting the ECG leads on a single figure with annotation, plotting a 3D plot of VCG (including animation!))
- `tools` covers more general use tools that are not limited to ECG/VCG analysis.

signalanalysis
signalplot
tools

2.1 signalanalysis

signalanalysis.ecg
signalanalysis.general
signalanalysis.vcg

2.1.1 signalanalysis.ecg

Functions

get_ecg_from_electrodes	Converts electrode phi_e data to ECG lead data
get_electrode_phie	Extract phi_e data corresponding to ECG electrode locations
get_qrs_start	Calculates start of QRS complex using method of Hermans et al. (2017).

continues on next page

Table 3 – continued from previous page

read_ecg_from_csv	Extract ECG data from CSV file exported from St Jude Medical ECG recording
read_ecg_from_dat	Read ECG data from .dat file
read_ecg_from_igb	Translate the phie.igb file(s) to 10-lead, 12-trace ECG data

signalanalysis.ecg.get_ecg_from_electrodes

`signalanalysis.ecg.`**`get_ecg_from_electrodes`**(*electrode_data: pandas.core.frame.DataFrame*) → pandas.core.frame.DataFrame

Converts electrode phi_e data to ECG lead data

Takes dictionary of phi_e data for 10-lead ECG, and converts these data to standard ECG trace data

Parameters **`electrode_data`** (*`pd.DataFrame`*) – Dictionary with keys corresponding to lead locations

Returns **ecg** – Dictionary with keys corresponding to the ECG traces

Return type pd.DataFrame

signalanalysis.ecg.get_electrode_phie

`signalanalysis.ecg.`**`get_electrode_phie`**(*phie_data: numpy.ndarray*, *electrode_file: Optional[str] = None*) → pandas.core.frame.DataFrame

Extract phi_e data corresponding to ECG electrode locations

Parameters

- **`phie_data`** (*`np.ndarray`*) – Numpy array that holds all phie data for all nodes in a given mesh
- **`electrode_file`** (*`str, optional`*) – File containing entries corresponding to the nodes of the mesh which determine the location of the 10 leads for the ECG. Will default to very project specific location. The input text file has each node on a separate line (zero-indexed), with the node locations given in order: V1, V2, V3, V4, V5, V6, RA, LA, RL, LL. Will default to '12LeadElectrodes.dat', but this is almost certainly not going to right for an individual project

Returns **electrode_data** – Dataframe of phie data for each node, with the dictionary key labelling which node it is.

Return type pd.DataFrame

signalanalysis.ecg.get_qrs_start

`signalanalysis.ecg.`**`get_qrs_start`**(*ecgs: Union[pandas.core.frame.DataFrame, List[pandas.core.frame.DataFrame]]*, *unipolar_only: bool = True*, *plot_result: bool = False*) → List[float]

Calculates start of QRS complex using method of Hermans et al. (2017)

Calculates the start of the QRS complex by a simplified version of the work presented in[1], wherein the point of maximum second derivative of the ECG RMS signal is used as the start of the QRS complex

[1] Hermans BJM, Vink AS, Bennis FC, Filippini LH, Meijborg VMF, Wilde AAM, Pison L, Postema PG, Delhaas T, "The development and validation of an easy to use automatic QT-interval algorithm," PLoS ONE, 12(9), 1–14 (2017), https://doi.org/10.1371/journal.pone.0184352

Parameters

- **ecgs** (*pd.DataFrame or list of pd.DataFrame*) – ECG data to analyse
- **unipolar_only** (*bool, optional*) – Whether to use only unipolar leads to calculate RMS, default=True
- **plot_result** (*bool, optional*) – Whether to plot the results for error-checking, default=False

Returns **qrs_starts** – QRS start times

Return type list of float

Notes

For further details of the action of unipolar_only, see general_analysis.get_signal_rms

It is faster to use scipy.ndimage.laplace() rather than np.gradient(np.gradient)), but preliminary checks indicated some edge problems that might throw off the results.

References

signalanalysis.ecg.read_ecg_from_csv

signalanalysis.ecg.**read_ecg_from_csv**(*filename: str*, *normalise: bool = False*) $\rightarrow$ pandas.core.frame.DataFrame

Extract ECG data from CSV file exported from St Jude Medical ECG recording

Parameters

- **filename** (*str*) – Name/location of the .dat file to read
- **normalise** (*bool, optional*) – Whether or not to normalise the ECG signals on a per-lead basis, default=True

Returns **ecg** – Extracted data for the 12-lead ECG

Return type list of pd.DataFrame

signalanalysis.ecg.read_ecg_from_dat

signalanalysis.ecg.**read_ecg_from_dat**(*filename: str*, *normalise: bool = False*) $\rightarrow$ pandas.core.frame.DataFrame

Read ECG data from .dat file

Parameters

- **filename** (*str*) – Name/location of the .dat file to read
- **normalise** (*bool, optional*) – Whether or not to normalise the ECG signals on a per-lead basis, default=False

Returns **ecg** – Extracted data for the 12-lead ECG

Return type pd.DataFrame

signalanalysis.ecg.read_ecg_from_igb

`signalanalysis.ecg.read_ecg_from_igb`(*filename: str*, *electrode_file: Optional[str] = None*, *normalise: bool = False*, *dt: float = 0.002*) → pandas.core.frame.DataFrame

Translate the phie.igb file(s) to 10-lead, 12-trace ECG data

Extracts the complete mesh data from the phie.igb file using CARPutils, which contains the data for the body surface potential for an entire human torso, before then extracting only those nodes that are relevant to the 12-lead ECG, before converting to the ECG itself https://carpentry.medunigraz.at/carputils/generated/carputils.carpio.igb.IGBFile.html#carputils.carpio.igb.IGBFile

Parameters

- **filename** (*str*) – Filename for the phie.igb data to extract
- **electrode_file** (*str, optional*) – File which contains the node indices in the mesh that correspond to the placement of the leads for the 10-lead ECG. Default given in get_electrode_phie function.
- **normalise** (*bool, optional*) – Whether or not to normalise the ECG signals on a per-lead basis, default=False
- **dt** (*float, optional*) – Time interval from which to construct the time data to associate with the ECG, default=0.002s (2ms)

Returns **ecgs** – DataFrame with Vm data for each of the labelled leads (the dictionary keys are the names of the leads)

Return type pd.DataFrame

Classes

Ecg	Base ECG class to encapsulate data regarding an ECG recording, inheriting from signalanalysis.general.Signal

signalanalysis.ecg.Ecg

`class signalanalysis.ecg.Ecg`(*filename: str*, ***kwargs*)

Bases: *signalanalysis.general.Signal*

Base ECG class to encapsulate data regarding an ECG recording, inheriting from signalanalysis.general.Signal

data

Raw ECG data for the different leads

Type pd.DataFrame

filename

Filename for the location of the data

Type str

normalised

Whether or not the data for the leads have been normalised

Type bool

n_beats

Number of beats recorded in the trace. Set to 0 if not calculated

Type int

`qrs_start`
Times calculated for the start of the QRS complex

Type list of float

`qrs_end`
Times calculated for the end of the QRS complex

Type end

`data_source`
Source for the data, if known e.g. Staff III database, CARP simulation, etc.

Type str

`comments`
Any further details known about the data, e.g. sex, age, etc.

Type str

`read`(*filename*)
Reads in the data from the original file. Called upon initialisation

`read_ecg_from_wfdb`(*filename*, *normalise=False*)
Reads data from a WFDB type series of files, e.g. from the Lobachevsky ECG database (https://physionet.org/content/ludb/1.0.1/)

`get_n_beats`(*threshold=0.5*, *min_separation=0.2*)
Calculates the number of beats given in the recording

`get_qrs_start`()
Calculates the start of the QRS complex

Methods

get_n_beats	Calculate the number of beats in an ECG trace, and save the individual beats to file for later use
get_qrs_start	Calculates start of QRS complex using method of Hermans et al. (2017).
get_rms	Returns the RMS of the combined signal
read	
read_ecg_from_wfdb	
reset	Reset all properties of the class

`get_n_beats`(*threshold: float = 0.5*, *min_separation: float = 0.2*, *unipolar_only: bool = True*, *plot: bool = False*)
Calculate the number of beats in an ECG trace, and save the individual beats to file for later use

When given the raw data of an ECG trace, will estimate the number of beats recorded in the trace based on the RMS of the ECG signal exceeding a threshold value. The estimated individual beats will then be saved in a list in a lossless manner, i.e. saved as [ECG1, ECG2, …, ECG(n)], where ECG1=[0:peak2], ECG2=[peak1:peak3], …, ECGn=[peak(n-1):end]

Parameters

- **threshold** (*float {0<1}*) – Minimum value to search for for a peak in RMS signal to determine when a beat has occurred, default=0.5
- **min_separation** (*float*) – Minimum time (in s) that should be used to separate separate beats, default=0.2s
- **unipolar_only** (*bool, optional*) – Whether to use only unipolar ECG leads to calculate RMS, default=True
- **plot** (*bool*) – Whether to plot results of beat detection, default=False

Returns **self.n_beats** – Number of beats detected in signal

Return type int

Notes

The scalar RMS is calculated according to

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathrm{ECG}_i^2(t))}$$

for all leads available from the signal (12 for ECG, 3 for VCG). If unipolar_only is set to true, then ECG RMS is calculated using only 'unipolar' leads. This uses V1-6, and the non-augmented limb leads (VF, VL and VR)

$$VF = LL - V_{WCT} = \frac{2}{3}aVF$$

$$VL = LA - V_{WCT} = \frac{2}{3}aVL$$

$$VR = RA - V_{WCT} = \frac{2}{3}aVR$$

get_qrs_start(*unipolar_only: bool = True*, *min_separation: float = 0.05*, *plot_result: bool = False*)

Calculates start of QRS complex using method of Hermans et al. (2017)

Calculates the start of the QRS complex by a simplified version of the work presented in[1], wherein the point of maximum second derivative of the ECG RMS signal is used as the start of the QRS complex

Parameters

- **self** (Ecg) – ECG data to analyse
- **unipolar_only** (*bool, optional*) – Whether to use only unipolar leads to calculate RMS, default=True
- **min_separation** (*float, optional*) – Minimum separation from the peak used to detect various beats, default=0.05s
- **plot_result** (*bool, optional*) – Whether to plot the results for error-checking, default=False

Returns **self.qrs_start** – QRS start times

Return type list of float

[1] Hermans BJM, Vink AS, Bennis FC, Filippini LH, Meijborg VMF, Wilde AAM, Pison L, Postema PG, Delhaas T, "The development and validation of an easy to use automatic QT-interval algorithm," PLoS ONE, 12(9), 1–14 (2017), https://doi.org/10.1371/journal.pone.0184352