ECGkit Documentation

Release 0.1.1

Mariano Llamedo Soria

Dec 20, 2018

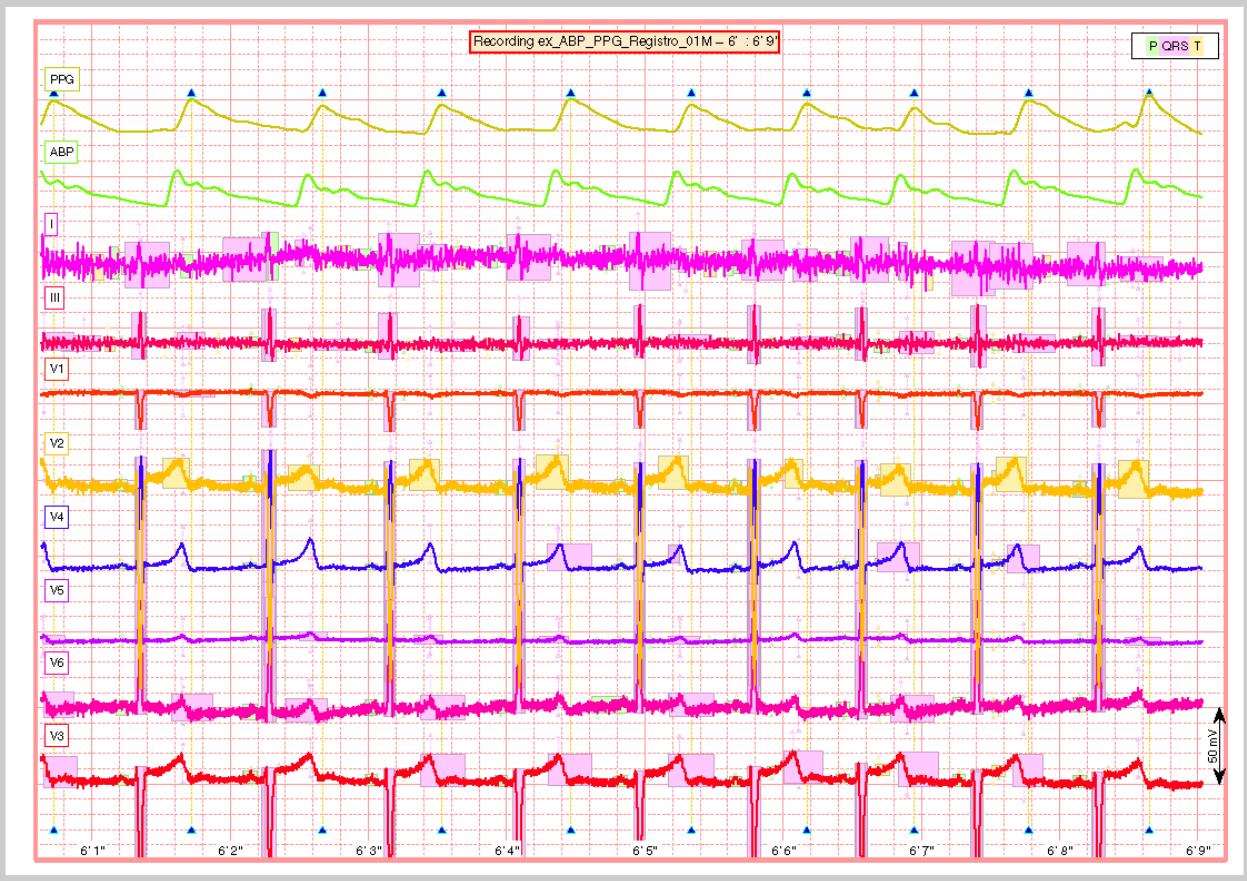
Contents

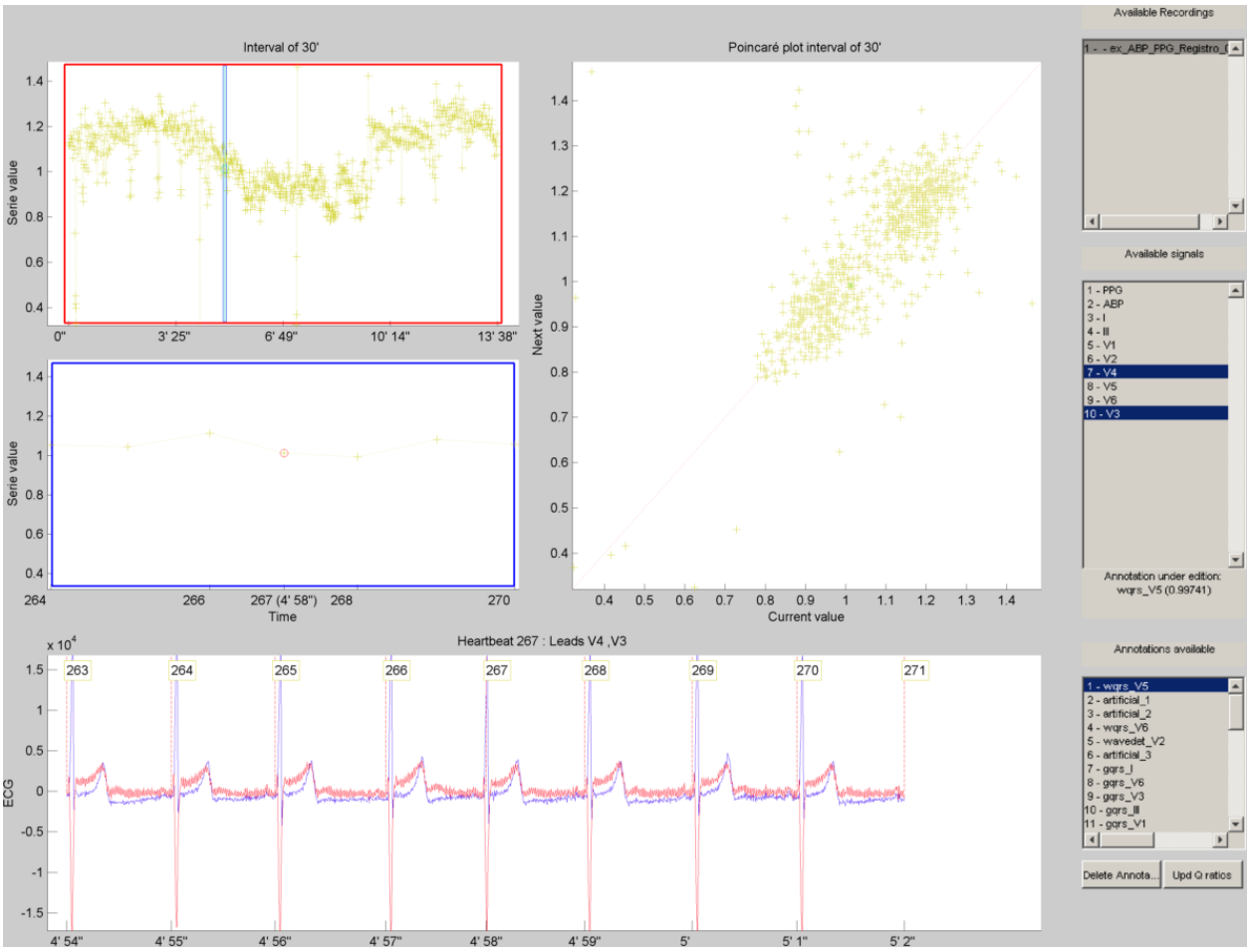
This toolbox is a collection of Matlab tools that I used, adapted or developed during my PhD and post-doc work with the Besicos group at University of Zaragoza, Spain and at the National Technological University of Buenos Aires, Argentina. The ECG-kit has tools for reading, processing and presenting results, as you can see in the documentation or in these demos on Youtube.

The main feature of the this toolbox is the possibility to use several popular algorithms for ECG processing, such as:

- Algorithms from Physionet's WFDB software package
- QRS detectors, such as gqrs, wqrs, wavedet, ecgpuwave, Pan & Tompkins, EP limited
- Wavedet ECG delineator
- Pulse wave detectors as wabp and wavePPG
- a2hbc and EP limited heartbeat classifiers.
- And other scritps for inspecting, correcting and reporting all these results.

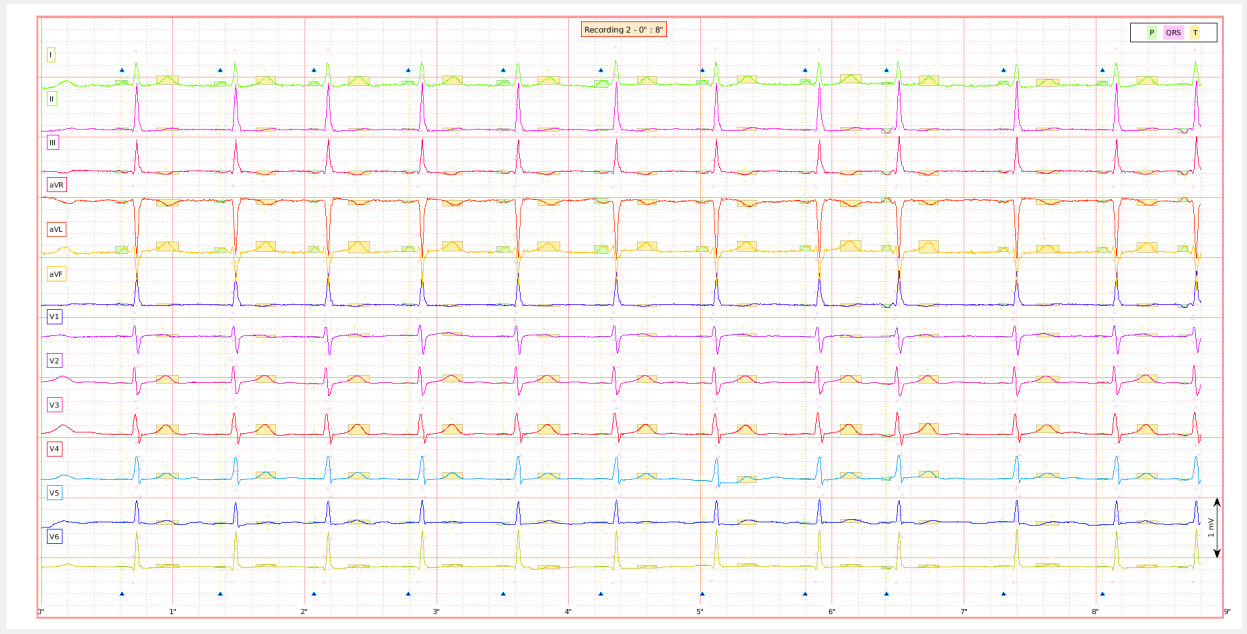
with the same application programmer interface (API) directly in Matlab, under Windows, Linux or Mac. The kit also implements a recording interface which allows processing several ECG formats, such as HL7-aECG, MIT, ISHNE, HES, Mortara, and AHA, of arbitrary recording size (the record so far is a 1 week recording of 3 leads, sampled at 500 Hz).




Interval of 30'
Serie value
0"
3' 25"
6' 49"
10' 14"
13' 38"
Poincaré plot interval of 30'
Next value
Current value
Time
264
266
267 (4' 58")
268
270
Heartbeat 267 : Leads V4 ,V3
x 10^4
ECG
263
264
265
266
267
268
269
270
271
4' 54"
4' 55"
4' 56"
4' 57"
4' 58"
4' 59"
5'
5' 1"
5' 2"
Available Recordings
Available signals
1 - PPG
2 - ABP
3 - I
4 - III
5 - V1
6 - V2
7 - V4
8 - V5
9 - V6
10 - V3
Annotation under edition:
wqrs_V5 (0.99741)
Annotations available
1 - wqrs_V5
2 - artificial_1
3 - artificial_2
4 - wqrs_V6
5 - wavedet_V2
6 - artificial_3
7 - gqrs_I
8 - gqrs_V6
9 - gqrs_V3
10 - gqrs_III
11 - gqrs_V1
Delete Annota...
Upd Q ratios


Recording 208 – 3' 19" : 3' 27"
N S V F U
P QRS T
global
included
gqrs_MLII
MLII
V1
1 mV
3' 19"
3' 20"
3' 21"
3' 22"
3' 23"
3' 24"
3' 25"
3' 26"
3' 27"


Recording 1 - 0" : 12"
P QRS T
I
II
III
aVR
aVL
aVF
V1
V2
V3
V4
V5
V6



This kit also includes many open-source projects such as WFDB Toolbox for MATLAB and Octave from Physionet, PRtools, Libra, export_fig from undocumented Matlab, and other open-source scripts that have their proper references to the original projects or authors.

CHAPTER 1

Voluntary contributions

Many thanks to Andrés Demski from UTN who helped to this project before he learned how to use it. To **all** the friends in Zaragoza, Porto and Lund, but in special to the ones closest to the project:

- Pablo Laguna, Juan Pablo Martínez, Rute Almeida and Juan Bolea, for the wavedet ECG delineator and many parts of the Biosig browser project that were adapted to this project.
- Jesús Lázaro and Eduardo Gil for the PPG / ABP pulse detection code.

CHAPTER 2

Involuntary contributions

The acknowledgements also goes to all these people, important in many ways to the fulfilment of this project

- George Moody, Wei Zong, Ikaro Silva, for all the software of Physionet.
- Reza Sameni, for his Open-Source ECG Toolbox (OSET)
- Bob Duin and all the team behind PRtools
- Yair Altman from undocumented Matlab
- Diego Armando Maradona for this.

2.1 Getting Started

2.1.1 Release notes

The toolbox is in beta testing now, please report any problem you may found as an issue here. The toolbox has been mostly tested in Windows and Linux platforms, if you are a Mac user, we appreciate your feedback for any issue.

It was tested in Matlab versions from R2012a to R2014b. Our recommendation is to avoid using versions 2014x, since the graphical engine was changed and several performance issues were discovered. At the time of writing this document, R2015a was released but not tested yet, so we suggest 2013x for best performance.

Octave 4 was recently released and tested, however the object oriented programming feature is too "green" to make the toolbox fully compatible yet. We hope that the toolbox will be working soon in the next versions of Octave release 4.x.

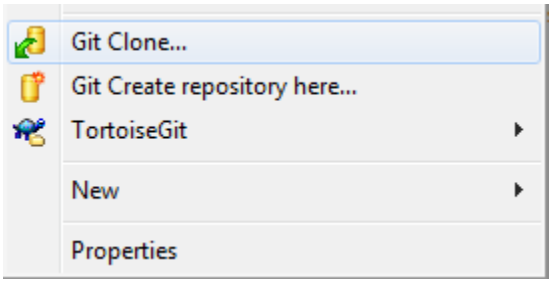
2.1.2 Be in touch

Feel free to join the forum to say hello or ask for help.

If you find any problem, you can open a new issue here, please provide as many details you can about your system, some code and data to reproduce the error.

2.1.3 Installation

The easiest way to install the latest stable version is downloading the project in zip or tgz. If you want the latest development version, clone the master branch with your favorite GIT client. Here you can find one for each supported platform, and a nice GUI for Windows called Tortoise. Then go to an empty folder, right click there and select *Git Clone*



or execute in the shell:

```
git clone --recursive git@github.com:marianux/ecg-kit.git
```

To install the kit, after unpacking or cloning the kit to 'your_installation_folder', then type in Matlab

```
cd 'your_installation_folder'
InstallECGkit()
```

Here you have a Youtube video of a typical installation. You can try then the `examples.m` script in order to check the correct installation in some included recordings:

```
examples()
```

or if you want to try in your own recordings:

```
examples('1/1', 'C:\Your_own_recordings\')
```

The use of this script is explained more in detail in the *Examples* section.

2.1.4 Uninstallation

Uninstallation is as easy as typing:

```
UnInstallECGkit()
```

Here you can check a typical uninstallation video as well

2.2 A first example

2.2.1 Another example

This script exemplifies the use of the ECGkit in a multimodal cardiovascular recording which includes arterial blood pressure (ABP), plethysmographic (PPG) and electrocardiogram signals. The following tasks will be performed in this example:

- *Heartbeat/QRS detection*

- *ABP/PPG pulse detection*
- *ECG wave delineation*
- *Heartbeat classification*
- *Report generation*

Each automatic step is followed by a manual verification step in order to verify the algorithm's results. The script is prepared to run locally without arguments, as well as in a cluster environment by using "pid_str" argument. The pid_str argument is a char with format 'N/M', being N <= M with default value '1/1'. You can partition a big job into M pieces in cluster architecture, by starting M processes with N ranging from 1 to M.

You can watch a typical run of this script for small, local ECG recording on YouTube.

Example of how to run this script

```
examples()
examples('1/1', 'C:\Your_preferred_local_path\', 'arbitrary_string')
examples('1/10', '/Your_preferred_path_in_cluster/', 'arbitrary_string')
```

2.2.1.1 Contents

- *Function prototype*
- *Argument parsing*
- *QRS automatic detection*
- *QRS visual inspection and correction*
- *PPG/ABP pulse detection*
- *PPG/ABP waves visual inspection and correction*
- *ECG automatic delineation*
- *Visual inspection of the detection/delineation*
- *Automatic Heartbeat classification*
- *Visual inspection of the signal*
- *Other user-defined tasks . . .*

2.2.1.2 Function prototype

```
function examples(pid_str, examples_path, user_str)
```

examples accepts three *optional* arguments:

- **pid_str** (optional) string identifier for this work instance in a cluster computing or multitask environment. The identifier follows the form 'N/M', being N a number which identifies this execution instance and M the total amount of instances. `'1/1' (default)`
- **examples_path** (optional) string of the path with ECG recordings. `['.' filesep 'example_recordings' filesep ] (default);`
- **user_str** (optional) string to identify this run or experiment.

2.2.1.3 Argument parsing

Simple and straight forward.

```
    if( nargin < 1 || ~ischar(pid_str) )
        % single PID run
        pid_str = '1/1';
    end
    if( nargin < 2 || ~exist(examples_path, 'dir') )
        % inspect ECG files in rootpath\example_recordings\ folder
        root_path = fileparts(mfilename('fullpath'));
        % default folder to look at
        examples_path = [root_path filesep 'example_recordings' filesep ];
        if(~exist(examples_path, 'dir'))
            disp_string_framed(2, 'Please provide a valid path with ECG recordings');
            return
        end
    else
        if( examples_path(end) ~= filesep )
            examples_path = [examples_path filesep];
        end
    end
    if( nargin < 3  )
        user_str = '';
    end
% Explore the *examples_path* for ECG recordings.
    filenames = dir(examples_path);
    recnames = {filenames(:).name};
% In this case I hardcoded only one recording
    recnames = {'ex_ABP_PPG_Registro_01M'};
% But you can use this to iterate for all of them.
%     [~,recnames] = cellfun(@(a)(fileparts(a)), recnames, 'UniformOutput', false);
%     recnames = unique(recnames);
%     recnames = setdiff(recnames, {'' '.' '..' 'results' 'condor' });
%     recnames = recnames(1)
    lrecnames = length(recnames);
    % In case of running in a user-assisted fashion.
    bUseDesktop = usejava('desktop');
    if( bUseDesktop )
        tmp_path = tempdir;
        output_path = [ examples_path 'results' filesep ];
    else
        % For cluster or distributed environment processing.
        InstallECGkit();
        % this is a local path, usually faster to reach than output_path
        tmp_path = '/scratch/';
        % distributed or cluster-wide accesible path
        output_path = [ examples_path 'results' filesep ];
    end
% just for debugging, keep it commented.
%     bUseDesktop = false
```

2.2.1.4 QRS automatic detection

In this example the first step is the location of each heartbeat, or QRS complexes detection. To achieve this, the kit includes the following algorithms:

- Wavedet
- Pan & Tompkins
- gqrs
- sqrs
- wqrs
- ecgpuwave

The way of performing QRS detection (or almost any other task in this ECGkit) is through an *ECGwrapper* object. The objective of this object is to abstract or separate any algorithm from the particular details of the ECG signal. This object is able to invoke any kind of algorithm through the interface provided of other object, called *ECGtask* objects.

The *ECGtask* objects actually perform specific task on the ECG signal, in this case, the QRS complex detection. Each task have general properties such as *progress_handle* (see *ECGtask* class properties for more details) and other specific for a certain task, such as *detectors*, *only_ECG_leads*, *wavedet_config*, *gqrs_config_filename* (see others in *QRS detection task*).

```
    % go through all files
    ECG_all_wrappers = [];
    jj = 1;
    for ii = 1:lrecnames
        rec_filename = [examples_path recnames{ii}];
        % task name,
%         ECGt_QRSd = 'QRS_detection';
        % or create an specific handle to have more control
        ECGt_QRSd = ECGtask_QRS_detection();
%         % select an specific algorithm. Default: Run all detectors
%         ECGt_QRSd.detectors = 'wavedet'; % Wavedet algorithm based on
%         ECGt_QRSd.detectors = 'pantom';  % Pan-Tompkins alg.
%         ECGt_QRSd.detectors = 'gqrs';    % WFDB gqrs algorithm.
%         % Example of how you can add your own QRS detector.
%         ECGt_QRSd.detectors = 'user:example_worst_ever_QRS_detector';
%         ECGt_QRSd.detectors = 'user:your_QRS_detector_func_name';    %
%         "your_QRS_detector_func_name" can be your own detector.
        ECGt_QRSd.detectors = {'wavedet' 'gqrs' 'wqrs' 'user:example_worst_ever_QRS_
↪detector'};
        % you can individualize each run of the QRS detector with an
        % or group by the config used
%         ECGt_QRSd.only_ECG_leads = false;    % consider all signals ECG
        ECGt_QRSd.only_ECG_leads = true;    % Identify ECG signals based on their␣
↪header description.
        ECG_w = ECGwrapper( 'recording_name', rec_filename, ...
                            'this_pid', pid_str, ...
                            'tmp_path', tmp_path, ...
                            'output_path', output_path, ...
                            'ECGtaskHandle', ECGt_QRSd);
        % external string
        ECG_w.user_string = user_str;
        try
            % process the task
            ECG_w.Run;
            % collect object if were recognized as ECG recordings.
            if( jj == 1)
                ECG_all_wrappers = ECG_w;
            else
                ECG_all_wrappers(jj) = ECG_w;
```

(continues on next page)

(continued from previous page)

```
            end
            jj = jj + 1;
        catch MException
            if( strfind(MException.identifier, 'ECGwrapper:ArgCheck:InvalidFormat') )
                disp_string_framed('*Red', sprintf( 'Could not guess the format of %s
↪', ECG_w.recording_name) );
            else
                % report just in case
                report = getReport(MException);
                fprintf(2, '\n%s\n', report);
            end
        end
    end
    % recognized recordings
    lrecnames = length(ECG_all_wrappers);
    % at the end, report problems if happened.
    for ii = 1:lrecnames
        ECG_all_wrappers(ii).ReportErrors;
    end
```

2.2.1.5 QRS visual inspection and correction

This part of the example uses a graphical user interface (GUI) to allow the user correcting mistakes that the previous automatic algorithm eventually makes.

As can be seen in the following code, the first step is checking that the previous QRS detection task finished without problems. Then if no errors, the corrector will use as starting point the result of this same task, in case the user would like to edit a previously edited result, or if not available the result of the QRS detection task.

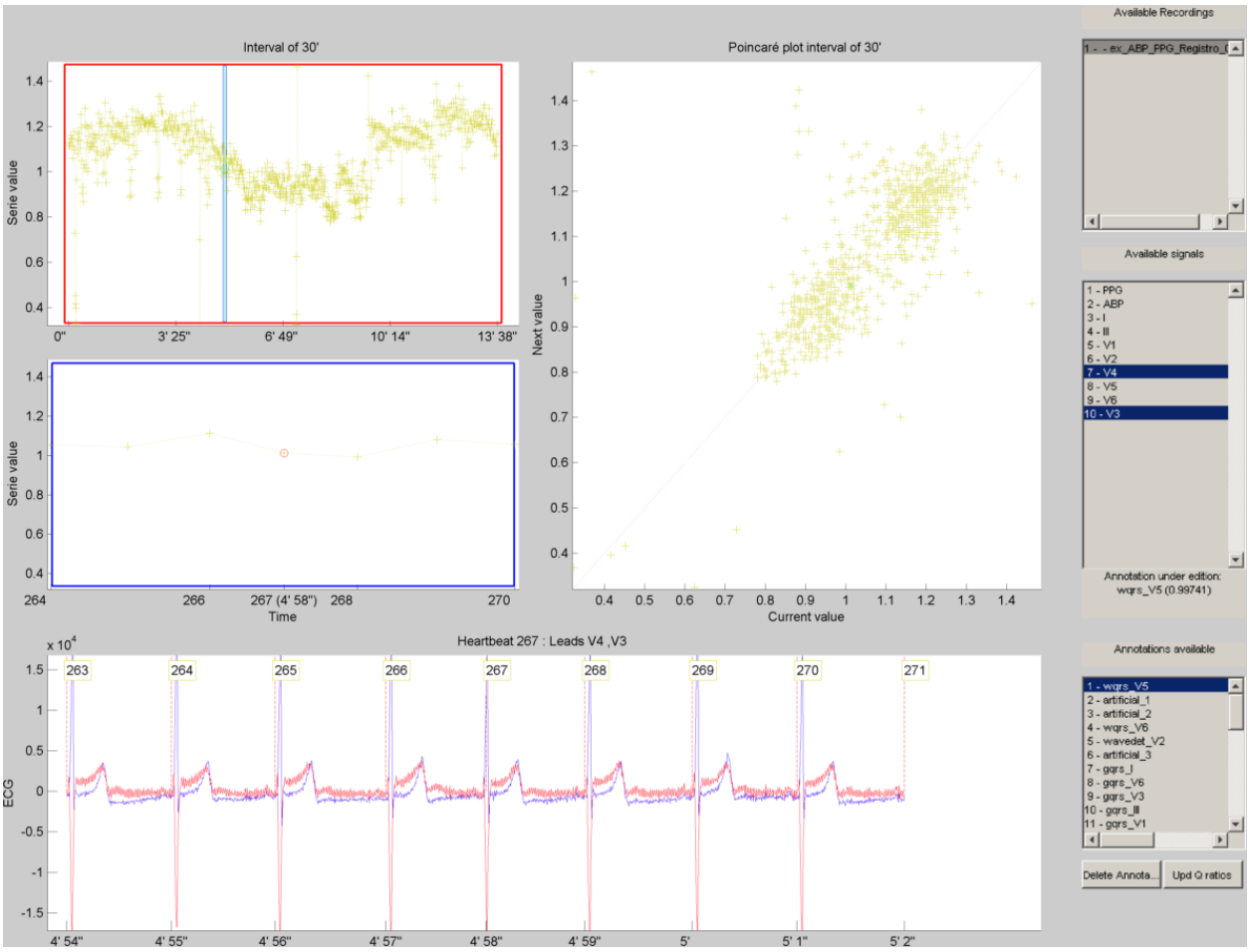
```
if( bUseDesktop )
    % other task can be performed on the same objects
    for ii = 1:lrecnames
        % last worker is the responsible of the visual correction.
        if( ECG_all_wrappers(ii).this_pid == ECG_all_wrappers(ii).cant_pids)
            % if there are not any previous error.
            if( ECG_all_wrappers(ii).Processed && ~ECG_all_wrappers(ii).Error )
                % this is to use previous saved results as starting point,
                % if any available
                cached_filenames = ECG_all_wrappers(ii).GetCahchedFileName({'QRS_
↪corrector' 'QRS_detection'});
                % if no previous correction work, try the automatic
                % detection task
                % if any, do the correction
                if( ~isempty(cached_filenames) )
                    % this is to use previous saved results as starting point,
                    % if any available
                    ECG_all_wrappers(ii).ECGtaskHandle = 'QRS_corrector';
                    % This task is supposed to be supervised, so only one pid is␣
↪enough.
                    ECG_all_wrappers(ii).this_pid = '1/1';
                    % user provided name to individualize each run
                    ECG_all_wrappers(ii).user_string = user_str;
                    % to avoid loading cached results and exit, this flag
                    % allows the re-editing of the current state of the
                    % detections.
```

(continues on next page)

(continued from previous page)

```
                    ECG_all_wrappers(ii).cacheResults = false;
                    % maybe in your application you should run this for
                    % all files.
                    ECG_all_wrappers(ii).ECGtaskHandle.payload = load(cached_filenames
↪{1});
                    % process the task
                    ECG_all_wrappers(ii).Run;
                    % restore the original pids configuration
                    ECG_all_wrappers(ii).this_pid = pid_str;
                    % As we changed for "QRS correction" task, we have to enable this
                    % value again in order to avoid performing the following tasks␣
↪every time.
                    % If you want to recalculate any task, change it to false
                    ECG_all_wrappers(ii).cacheResults = true;
                end
            end
        end
    end
    % at the end, report problems if happened.
    for ii = 1:lrecnames
        ECG_all_wrappers(ii).ReportErrors;
    end
end
```

Then the task invoked by the wrapper object is changed to QRS corrector task and the GUI is presented to the user.



In this example, the GUI have four plots to represent the RR interval series, the two in the top-left show the RR interval versus time at different time windows. The bigger in the top-right, shows a *Poincaré* plot, that is the current RR interval versus the following in the serie. The plot in the bottom shows the selected signal/s versus time. Then the user can interact with the plots according to the QRS corrector documentation

2.2.1.6 PPG/ABP pulse detection

In case the recording includes pulsatile signals, such as plethysmographic (PPG) or arterial blood pressure (ABP), this kit includes the PPG/ABP automatic detector task which allows the use of two algorithms to perform peak detection, WavePPG and Physionet's wabp.

other task can be performed on the same objects

```
for ii = 1:lrecnames
    % set the delineator task name and run again.
    ECG_all_wrappers(ii).ECGtaskHandle = 'PPG_ABP_detector';
    % user provided name to individualize each run
    ECG_all_wrappers(ii).user_string = user_str;
    % process the task
    ECG_all_wrappers(ii).Run;
end
% at the end, report problems if happened.
for ii = 1:lrecnames
```

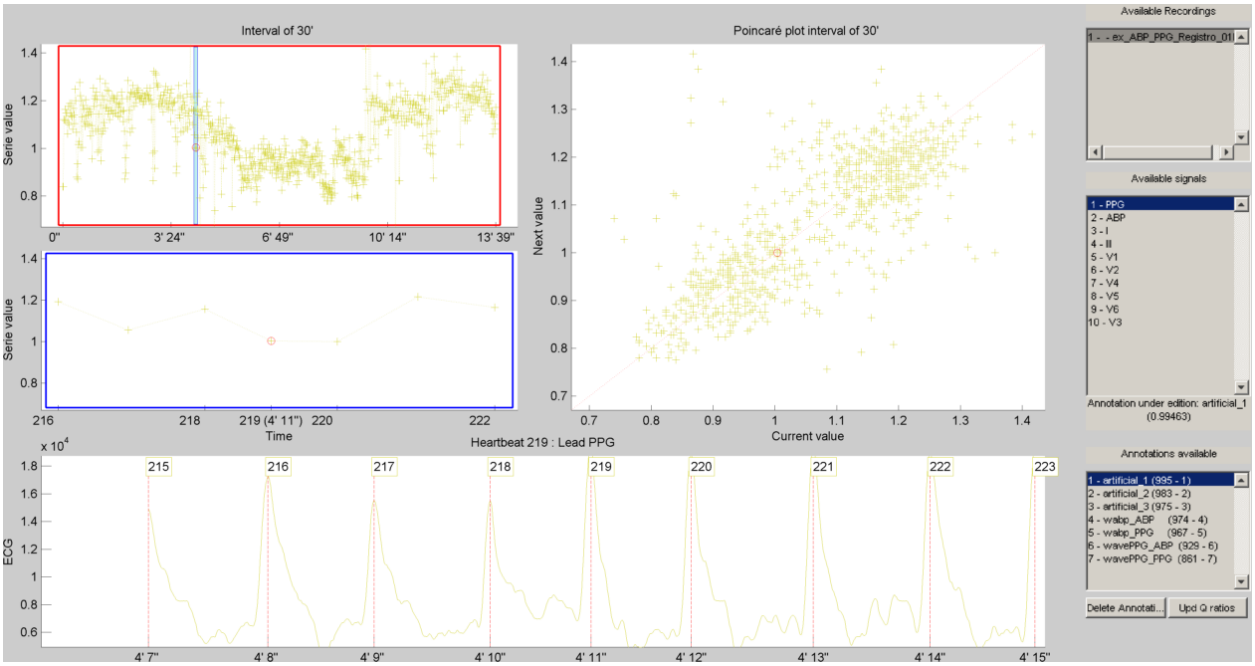
(continues on next page)

(continued from previous page)

```
    ECG_all_wrappers(ii).ReportErrors;
end
```

2.2.1.7 PPG/ABP waves visual inspection and correction

The same manual verification made for automatic QRS detection algorithms can be performed with pulsatile signals. The PPG/ABP corrector task was designed to allow users the verification and correction of automatic detections through the same GUI.



The following code shows how to use this task. As you can note, the interface is almost the same used for the QRS correction task.

```
if( bUseDesktop )
    % other task can be performed on the same objects
    for ii = 1:lrecnames
        % last worker is the responsible of the visual correction.
        if( ECG_all_wrappers(ii).this_pid == ECG_all_wrappers(ii).cant_pids)
            % if there are not any previous error.
            if( ECG_all_wrappers(ii).Processed && ~ECG_all_wrappers(ii).Error )
                % this is to use previous saved results as starting point,
                % if any available
                cached_filenames = ECG_all_wrappers(ii).GetCahchedFileName({'PPG_ABP_
↪corrector' 'PPG_ABP_detector'});
                % if no previous correction work, try the automatic
                % detection task
                % if any, do the correction
                if( ~isempty(cached_filenames) )
                    % this is to use previous saved results as starting point,
                    % if any available
                    ECG_all_wrappers(ii).ECGtaskHandle = 'PPG_ABP_corrector';
                    % This task is supposed to be supervised, so only one pid is␣
↪enough.
```

(continues on next page)

(continued from previous page)

```
                    ECG_all_wrappers(ii).this_pid = '1/1';
                    % user provided name to individualize each run
                    ECG_all_wrappers(ii).user_string = user_str;
                    % to avoid loading cached results and exit, this flag
                    % allows the re-editing of the current state of the
                    % detections.
                    ECG_all_wrappers(ii).cacheResults = false;
                    % maybe in your application you should run this for
                    % all files.
                    ECG_all_wrappers(ii).ECGtaskHandle.payload = load(cached_filenames
↪{1});
                    % process the task
                    ECG_all_wrappers(ii).Run;
                    % restore the original pids configuration
                    ECG_all_wrappers(ii).this_pid = pid_str;
                    % As we changed for "QRS correction" task, we have to enable this
                    % value again in order to avoid performing the following tasks␣
↪every time.
                    % If you want to recalculate any task, change it to false
                    ECG_all_wrappers(ii).cacheResults = true;
                end
            end
        end
    end
    % at the end, report problems if happened.
    for ii = 1:lrecnames
        ECG_all_wrappers(ii).ReportErrors;
    end
end
```

2.2.1.8 ECG automatic delineation

Once the QRS complexes were detected, each heartbeat can be segmented or delineated into P-QRS-T waves. To achieve this the kit includes an ECG delineation task to interface with the wavedet and others user-defined algorithms, as described in the task help. The interface follows the same guidelines described before, as is shown in the following code.

other task can be performed on the same objects

```
    for ii = 1:lrecnames
        % this is to use previous cached results as starting point
        cached_filenames = ECG_all_wrappers(ii).GetCahchedFileName('QRS_corrector');
        % if corrected QRS detections are not available, wavedet
        % performs automatic QRS detection.
        if( ~isempty(cached_filenames) )
            % this is to use previous result from the automatic QRS
            % detection
            ECG_all_wrappers(ii).ECGtaskHandle.payload = load(cached_filenames{1});
        end
        % set the delineator task name and run again.
        ECG_all_wrappers(ii).ECGtaskHandle = 'ECG_delineation';
        % user provided name to individualize each run
        ECG_all_wrappers(ii).user_string = user_str;
        % Identify ECG signals based on their header description and
        % perform delineation in those leads.
```

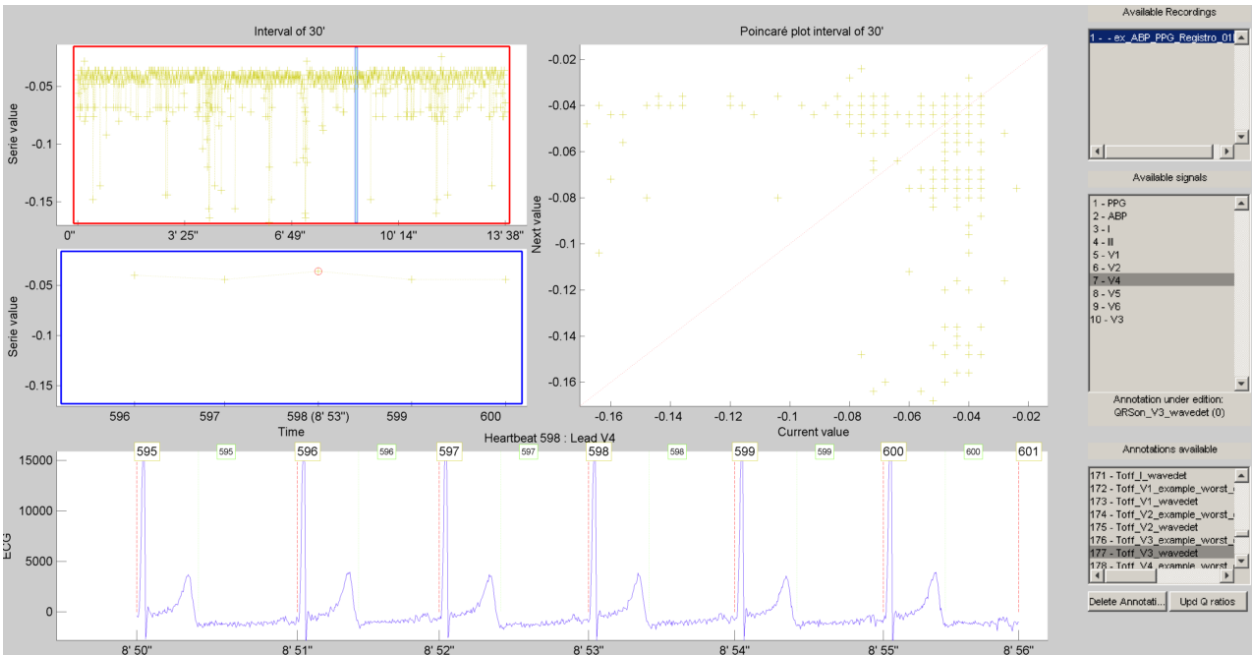
(continues on next page)

(continued from previous page)

```
        ECG_all_wrappers(ii).ECGtaskHandle.only_ECG_leads = true;
%         ECGt_QRSd.detectors = 'wavedet'; % Wavedet algorithm based on
%         ECGt_QRSd.detectors = 'user:example_worst_ever_ECG_delineator';
%         % Example of how you can add your own ECG delineator.
%         ECGt_QRSd.detectors = 'user:your_ECG_delineator_func_name';
%         "your_ECG_delineator_func_name" can be your own delineator.
        ECG_all_wrappers(ii).ECGtaskHandle.delineators = {'wavedet' 'user:example_
↪worst_ever_ECG_delineator'};
        % process the task
        ECG_all_wrappers(ii).Run;
    end
    % at the end, report problems if happened.
    for ii = 1:lrecnames
        ECG_all_wrappers(ii).ReportErrors;
    end
```

2.2.1.9 Visual inspection of the detection/delineation

The same manual verification made for all the previous automatic tasks is repeated for ECG delineation. The ECG delineation corrector task was designed to allow users the verification and correction of automatic delineation through the same GUI. The only difference with respect to the behaviour of the QRS or PPG/ABP correction GUI, is that addition of new events to the P-QRS-T series is not allowed, in order to keep the assosiation of a wave fiducial point to a heartbeat.




```
if( bUseDesktop )
    % other task can be performed on the same objects
    for ii = 1:lrecnames
        % last worker is the responsible of the visual correction.
        if( ECG_all_wrappers(ii).this_pid == ECG_all_wrappers(ii).cant_pids)
            % if there are not any previous error.
            if( ECG_all_wrappers(ii).Processed && ~ECG_all_wrappers(ii).Error )
                % this is to use previous saved results as starting point,
```

(continues on next page)

(continued from previous page)

```
                % if any available
                cached_filenames = ECG_all_wrappers(ii).GetCahchedFileName( {'ECG_
↪delineation_corrector' 'ECG_delineation'} );
                % if no previous correction work, try the automatic
                % detection task
                % if any, do the correction
                if( ~isempty(cached_filenames) )
                    % this is to use previous saved results as starting point,
                    % if any available
                    ECG_all_wrappers(ii).ECGtaskHandle = 'ECG_delineation_corrector';
                    % This task is supposed to be supervised, so only one pid is␣
↪enough.
                    ECG_all_wrappers(ii).this_pid = '1/1';
                    % user provided name to individualize each run
                    ECG_all_wrappers(ii).user_string = user_str;
                    % to avoid loading cached results and exit, this flag
                    % allows the re-editing of the current state of the
                    % detections.
                    ECG_all_wrappers(ii).cacheResults = false;
                    % maybe in your application you should run this for
                    % all files.
                    ECG_all_wrappers(ii).ECGtaskHandle.payload = load(cached_filenames
↪{1});
                    % process the task
                    ECG_all_wrappers(ii).Run;
                    % restore the original pids configuration
                    ECG_all_wrappers(ii).this_pid = pid_str;
                    % As we changed for "QRS correction" task, we have to enable this
                    % value again in order to avoid performing the following tasks␣
↪every time.
                    % If you want to recalculate any task, change it to false
                    ECG_all_wrappers(ii).cacheResults = true;
                end
            end
        end
    end
    % at the end, report problems if happened.
    for ii = 1:lrecnames
        ECG_all_wrappers(ii).ReportErrors;
    end
end
```

2.2.1.10 Automatic Heartbeat classification

The last task described in this example is the classification of heartbeats according to the EC-57 AAMI recommendation. To achieve this task, the kit includes a Heartbeat classification task that interfaces with the Argentino-Aragonés heartbeat classifier (a2hbc) project in order to classify heartbeats into the following classes:

- **N** normal
- **S** supraventricular
- **V** ventricular
- **F** fusion of normal and ventricular

The *a2hbc* algorithm can opperate automatically or assisted by the user, for more details check the a2hbc documentation.

```
    for ii = 1:lrecnames
        % this is to use previous cached results as starting point
        cached_filenames = ECG_all_wrappers(ii).GetCahchedFileName({'QRS_corrector'
↪'QRS_detection'});
        % if corrected QRS detections are not available, wavedet
        % performs automatic QRS detection.
        if( ~isempty(cached_filenames) )
            ECG_all_wrappers(ii).ECGtaskHandle = 'ECG_heartbeat_classifier';
            % the heartbeat classifier uses the QRS detection performed
            % before, if available the task will use the corrected
            % detections.
            ECG_all_wrappers(ii).ECGtaskHandle.payload = load(cached_filenames{1});
            % modes of operation of the a2hbc algorithm
            ECG_all_wrappers(ii).ECGtaskHandle.mode = 'auto';
%             ECG_all_wrappers(ii).ECGtaskHandle.mode = 'slightly-assisted';
%             ECG_all_wrappers(ii).ECGtaskHandle.mode = 'assisted';
            % user provided name to individualize each run
            ECG_all_wrappers(ii).user_string = user_str;
            % process the task
            ECG_all_wrappers(ii).Run;
        end
    end
    % at the end, report problems if happened.
    for ii = 1:lrecnames
        ECG_all_wrappers(ii).ReportErrors;
    end
```

2.2.1.11 Visual inspection of the signal

Finaly a report is generated with the results of the previous tasks, either in a pdf document or several images. The report generated can be customized with the interface described in the documentation. The following are just three examples of a longer report:

A snapshot of the center

And finaly a snapshot of the last part of the recording.

This is the code used to create a PDF report.

```
    filename = []; % default setting. Let the report function decide.
%     filename = 'container_filename'; % to put everything in one big file.
    % other winlengths can be added to the array in order to further
    % explore the recordings, and the algorithm results.
%     winlengths = []; % default setting
    winlengths = [ 7 ]; %seconds
    % go through all files
    for ii = 1:lrecnames
        if( ECG_all_wrappers(ii).this_pid == ECG_all_wrappers(ii).cant_pids)
            % last worker is the responsible of the reporting.
            if( ECG_all_wrappers(ii).this_pid == ECG_all_wrappers(ii).cant_pids)
                try
                    reportECG(ECG_all_wrappers(ii), 'LowDetail', 'full', winlengths,
↪'pdf', filename );
                catch MException
                    report = getReport(MException);
                    fprintf(2, '\n%s\n', report);
                end
            end
        end
    end
```

2.2.1.12 Other user-defined tasks . . .

Maybe the most important and useful aspect of the kit, is that you can add your own algorithms. This can be done by following the interface documented through the several examples included above. The QRS detection and ECG delineation tasks already include a way to interface your own algorithms through the **user:function_name** method. Check the above sections for more details.

```
if( ~bUseDesktop )
    UnInstallECGkit();
end
```

The script `/examples/first_simple_example.m` shows the use of the ECGkit in multimodal-cardiovascular recordings included with this kit in the `recordings` folder. These recordings include arterial blood pressure (ABP), plethysmographic (PPG) and electrocardiogram (ECG) signals. The following tasks will be performed in the first example:

- *Heartbeat/QRS detection*
- *ABP/PPG pulse detection*
- *ECG wave delineation*
- *Heartbeat classification*
- *Report generation*

The script is prepared to perform visual inspection of the automatic algorithms, with the `bGUICorrection` flag, but it is disabled by default. In the following listing, you can see a typical output of the example script.

```
>> first_simple_example()

Description of the process:
 + Recording: \your_path\ecg-kit\recordings\208.hea
 + Task name: QRS_detection

Processing QRS detector gqrs
Processing QRS detector wavedet
No multilead strategy used, instead using the delineation of lead MLII.
Processing QRS detector wqrs

##############
# Work done! #
##############


Results saved in
 + \your_path\ecg-kit\recordings\208_my_experiment_name_QRS_detection.mat
```

The script starts with the QRS complex detection, producing a result file, which can be used later by other tasks, or to access results. Then the script follows with the pulse detection task:

```
##############
# Work done! #
##############


Results saved in
 + \your_path\ecg-kit\recordings\208_my_experiment_name_QRS_detection.mat
```

(continues on next page)

(continued from previous page)

```
Description of the process:
 + Recording: \your_path\ecg-kit\recordings\208.hea
 + Task name: PPG_ABP_detector

Could not find any PPG/ABP signal, check the lead description of the recording:
 +  MLII
 +  V1

Requirements not satisfied in \your_path\ecg-kit\recordings\208.hea for task PPG_ABP_
↪detector.

######################
# Nothing to do here #
######################
```

As the default recording pointed by the script is the 208 from the MIT arrhythmia database, the task exits without finding any pulsatile signal, such as ABP or PPG. After exiting, it starts with the delineation task

```
 Description of the process:
 + Recording: \your_path\ecg-kit\recordings\208.hea
 + Task name: ECG_delineation

Processing ECG delineator wavedet
No multilead strategy used, instead using the delineation of lead MLII.

##############
# Work done! #
##############


Results saved in
 + \your_path\ecg-kit\recordings\208_my_experiment_name_ECG_delineation.mat
```

The result produced is exactly the same as the QRS detection task. This can be convenient for backing up intermediate results and reproducing experiment results. Other interesting aspect to differentiate experiments is the `user_string` property of the ECGwrapper object. Note that the results produced retain the user string suffix. After that, the example performs heartbeat classification and finally produces a report.

```
Description of the process:
 + Recording: \your_path\ecg-kit\recordings\208.hea
 + Task name: ECG_heartbeat_classifier

 + Using gqrs_MLII detections.

15-Apr-2015
Configuration
-------------
+ Recording:  (AHA)
+ Mode: auto (12 clusters, 1 iterations, 75% cluster-presence)

##############
# Work done! #
##############
```

(continues on next page)

(continued from previous page)

```
Results saved in
 + \your_path\ecg-kit\recordings\208_my_experiment_name_ECG_heartbeat_classifier.mat
```

As a result, the report found in `\your_path\ecg-kit\recordings\208_full.pdf` looks like this:

First in the report you will find an overview of the signal.

A more detailed view can be found in the last part of the report, you will find the results of the QRS detection, delineation and heartbeat classification. In the top right of the chart you will find the references for interpreting the results displayed. For example, QRS detection legend indicates a colour-code for the dotted lines with triangles in the extremes, placed around the QRS complexes. As recording 208 does not presents much controversy respect QRS detection, all detections are clustered around each heartbeat, but the legend indicates:

- *global*, the wavedet multilead detection.
- *hb_classifier*, the detection used by the heartbeat classifier.
- *included*, the gold-standard, visually-audited Physionet detections.
- *gqrs_MLII*, the detections of *gqrs* algorithm in lead MLII.

Above the legend of QRS detections, it is the delineation legend, with a colour-code for identifying each wave. For example in pink you can see several QRS complexes correctly segmented, but three of them are wrong. If you pay attention to the extreme widened ventricular beats, they are very underestimated. T-waves in orange, seems quite correctly measured, and P-waves as you can see are not correctly measured at all.

You can experiment with the `/examples/second_simple_example.m` to see how to extend this experiment to a multiprocessor environment.

2.3 ECGwrapper

2.3.1 Matlab file format

The ecg-kit allows users to use Matlab native format to store and access recordings. For using it, ensure that your mat file follows these naming convention:

1. Your signal variable must be named `'sig', 'signal' or 'ECG'`. Remember that the signals or leads must be placed column-wise, that is `signal = [ lead1 lead2 lead3 ... lead_nsig]`.
2. The header or signal information must be stored in a struct named `'header', 'heasig' or 'hea'`. The header variable is a struct with the fields defined *here*.
3. (Optional) In case of including annotations or QRS detections, be sure to be a struct named `'ann', 'annotations' or 'qrs'` and which includes the fields described for the MIT format in Physionet.
    - time: the time within the recording (recorded in the annotation file as the sample number of the sample to which the annotation "points")
    - anntyp [sic]: a numeric annotation code (see ecgcodes.h for definitions)
    - subtyp [sic], chan, num: three small integers (between -128 to 127) that specify context-dependent attributes (see the documentation for each database for details)
    - aux: a free text string

See the `ecg-kit\common\matformat_definitions.m` for more details.

2.3.2 Header Format

The header struct variable includes the technical information that describes an ECG recording. The ECG recording is referred as the ECG_matrix in this document, and is a integer matrix with the raw ADC samples. This structure has a set of mandatory fields that must be included:

- *freq*, is the sampling frequency of ECG_matrix signal, or in other words, the reciprocal of the sampling interval or time (in seconds) elapsed between ADC samples.
- *desc*, description strings about each of the leads/signals. It is a char matrix of [nsig × description_length], where description_length is the length of the longer description string.
- *nsamp* is the number of samples of ECG_matrix, or the size of the first matrix dimension (rows).
- *nsig* is the amount of leads or signals of ECG_matrix, or the size of the second matrix dimension (columns).
- *gain* is a double precision vector of [nsig × 1] with the gain of each lead (ADCsamples / μV).
- *adczero* is a double precision vector of [nsig × 1] with the offset of each lead in ADC samples.

Also other fields described in the Physionet header are accepted by the kit, check the source code for more details.

2.3.3 Custom formats

This document describes the procedure to make other file formats compatible with the toolbox, and in concrete with the ECGwrapper objects. Will be written soon.

This class allows the access to ECG recordings of several *formats* and length.

2.3.4 Syntax

You can create an object without arguments, configuring its properties later. Or you can create it via pairs of PropertyName-PropertyValue pairs.

```
ECGw = ECGwrapper()

ECGw = ECGwrapper( 'PropertyName', PropertyValue )
```

2.3.5 Description

This is the main class of the toolbox since it allows access to cardiovascular signal recordings of several formats (MIT, ISHNE, AHA, HES, *MAT*) and lengths, from seconds to days. The objective of this class is to provide *ECGtask* class, a common interface to access data and perform specific tasks. Briefly, this class sequentially reads data and passes to the Process method of the *ECGtask* plugged in the *ECGtaskHandle* property. Some common tasks, such as QRS detection and ECG delineation, can be easily invoked. Also other predefined tasks or your own code can be adapted as is shown in the *examples*.

A more detailed description of this class, together with an explanation of how you can easily hook your algorithms to this class is *here*.

Finally the results produced by the *ECGtask* are stored in order to ease reproducibility and backup of your experiments, or to be used of subsequent tasks as shown in the examples.

2.3.6 Input Arguments

Specify optional comma-separated pairs of `'Name', Value` arguments. `'Name'` is the argument/property name and `Value` is the corresponding value. `'Name'` must appear inside single quotes `' '`. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example:

```
% specifies to create an ECGwrapper to detect
% heartbeats in recording '/ecg_recordings/rec1.dat'
ECGwrapper('recording_name','/ecg_recordings/rec1.dat', 'ECGtaskHandle', 'QRS_
↪detection')
```

The available arguments for the class constructor are listed below:

`'recording_name'` — The ECG recording full name. `''` `(default)`

The full path filename of the ECG recording.

`'recording_format'` — ECG recording format. `auto-detect format (default)`

The format of the ECG recording. By default or if not specified, the wrapper will attempt to auto-detect the format among the following table:

String	Description
MIT	MIT format
ISHNE	ISHNE format
AHA	American Heart Association ECG Database or AHA in Physionet
HES	Biosigna format
MAT	*Matlab file format*
Mortara	Mortara SuperECG format

`'this_pid'` — Process identification for multiprocess batch jobs. `'1/1' (default)`

In case working in a multiprocess environment, this value will identify the current process. Can be a numeric value, or a string of the form 'N/M'. This pid is N and the total amount of pid's to divide the whole work is M.

`'tmp_path'` — The path to store temporary data. `tempdir() (default)`

Full path to a directory with write privileges.

`'output_path'` — The output path to store results. `fileparts(recording_name) (default)`

Full path to a directory with write privileges. By default will be the same path of the recordings.

`'ECGtaskHandle'` — The task to perform. `'' (default)`

The task to perform, can be the name of the task, or an ECGtask object. Available ECGtasks can be listed with `` `list_all_ECGtask() `` command.

`'partition_mode'` — The way that this object will partition lengthy signals. `'ECG_overlapped' (default)`

The way to do batch partition in lengthy signals:

- 'ECG_contiguous' no overlapp between segments.
- 'ECG_overlapped' overlapp of 'overlapping_time' among segments. This can be useful if your task have a transient period to avoid.
- 'QRS' do the partition based on the annotations provided in ECG_annotations.time property. This option is useful if your task works in the boundaries of a fiducial point (commonly a heartbeat), and not in the whole signal. This partition mode ignores those parts of the recording without annotations.

`'overlapping_time'` — Time in seconds of overlapp among consequtive segments. `30 (default)`

Time in seconds of overlapp among consequtive segments. This segment is useful for ensuring the end of all transients within a task.

`'cacheResults'` — Save intermediate results to recover in case of failure. `true (default)`

Save intermediate results to recover in case of errors. Useful for long jobs or recordings.

`` `'syncSlavesWithMaster' `` — Time in seconds of overlapp among consequtive segments. `false (default)`

In multiprocess environments sometimes it is useful to terminate all pid's together in order to start subsequent tasks synchronously. This value forces all parts of a multipart process to wait until all other parts finish.

`'repetitions'` — Times to repeat the ECGtask. `1 (default)`

In case the ECGtask is not deterministic, the repetition property allows to repeat the task several times.

Other public properties to configure the object, or to access recording's data are:

`Error` — Did the ECGtask produce any Error ? `false (default)`

You can check this value after executing a task.

`Processed` — Was the task already processed ? `false (default)`

You can check if you have already issued a Run method for the current configuration.

`NoWork2Do` — Has this PID work to do ?

In a multi-PID environment, certain tasks are too short to provide work to all configured PIDs. You can check to this flag to deal with it in your code.

`ECG_header` — The gathered information about the ECG recording.

ECG_header, is a struct with the following information:

- *freq*, is the sampling frequency of ECG_matrix signal.
- *desc*, description strings about each of the leads/signals.
- *nsamp* is the number of samples of ECG_matrix.
- *nsig* is the amount of leads or signals of ECG_matrix.
- *gain* is a vector of [nsig × 1] with the gain of each lead (ADCsamples / μV).
- *adczero* is a vector of [nsig × 1] with the offset of each lead in ADC samples.

and others described in the Physionet header.

`ECG_annotations` — Annotations provided with the recording.

Commonly QRS detections, signal quality annotations or other type of measurements included with the recordings. Some documentation about annotations in Physionet.

`class_labeling` — Class conversion for heartbeat annotations.

In case the annotations includes heartbeat types, this property indicates the class-labeling convention used. The EC-57 AAMI recommendation is de default value. The possible values are `'AAMI'` or `AAMI2`. The `AAMI2` is equal to `AAMI` except that only consider three heartbeat classes, *normal* (N), *ventricular* (V) and *supraventricular* (S).

`user_string` — A string to individualize each experiment. `''  (default)`

`Result_files` — The result filenames produced by an ECGtask.

Once the task is completed, this property records the filenames of the results.

2.3.7 Methods

Some useful methods are described below.

`Run` — Execute the ECG task

This method executes the configured ECG task.

`read_signal` — Read signal samples

This method allows to easily reads samples from a recording

```
% function prototype
function ECG = read_signal(ECG_start_idx, ECG_end_idx)
```

where the arguments are:

ECG_start_idx, is the first sample to read. `1 (default).`

ECG_end_idx, is the last sample to read. `ECG\_header.nsamp (default)`

and as a result, it returns:

ECG, which is a matrix of size `[(ECG\_end\_idx - ECG\_start\_idx + 1) ECG\_header.nsig] (default)`

as it is exemplified below

```
% reads ECG 100 samples
ECG = ECG_w.read_signal(1, 99);
```

`ReportErrors` — Display the error report generated during the task execution

This method reports error generated during task execution.

`GetCahchedFileName` — Find the result files of an specific recording/task

This method returns the cached filename for an specific recording, and task, if available.

```
% Get result filename of previous QRS detection.
% The corrected/audited version has precedence if available.
cached_filenames = ECGw.GetCahchedFileName({'QRS_corrector' 'QRS_detection'}
↪);
```

2.3.8 Examples

2.3.8.1 Create the simplest ECG wrapper object

Create the ECGwrapper object.

```
>> ECG_w = ECGwrapper()
ECG_w =
############################
# ECGwrapper object config #
############################
+ECG recording: None selected
+PID: 1/1
+Repetitions: 1
+Partition mode: ECG_overlapped
+Function name: Null task
+Processed: false
```

Then, in your script or in the command window you can type:

```
>> ECG_w.recording_name = 'some_path\100';
>> ECG_w.ECGtaskHandle = 'QRS_detection'
ECG_w =
############################
# ECGwrapper object config #
############################
+ECG recording: some_path\100 (auto)
+PID: 1/1
+Repetitions: 1
+Partition mode: ECG_overlapped
+Function name: QRS_detection
+Processed: false
```

Now, you just want to run the task by executing:

```
>> ECG_w.Run();
```

2.3.8.2 Create an ECGwrapper object for an specific recording and task

In this case, we create the same object of the previous example but using the name-value .

```
>> ECG_w = ECGwrapper( ...
        'recording_name', 'some_path\100', ...
        'recording_format', 'MIT', ...
        'ECGtaskHandle', 'QRS_detection', ...
        )
ECG_w =
############################
# ECGwrapper object config #
############################
+ECG recording: some_path\100 (auto)
+PID: 1/1
+Repetitions: 1
+Partition mode: ECG_overlapped
+Function name: QRS_detection
+Processed: false

>> ECG_w.Run();
```

2.3.8.3 Create an ECGwrapper and access the recording data

In this case, we create an object and access to the `ECG_header` property.

```
>> ECGw = ECGwrapper( 'recording_name', 'd:\mariano\misc\ecg-kit\recordings\208')

ECGw =


############################
# ECGwrapper object config #
############################

+ECG recording: d:\mariano\misc\ecg-kit\recordings\208 (auto)
+PID: 1/1
+Repetitions: 1
+Partition mode: ECG_overlapped
+Function name: Null task
+Processed: false

>> ECGw.ECG_header

ans =

         recname: '208'
            nsig: 2
            freq: 360
           nsamp: 650000
           btime: '00:00:00'
           bdate: '01/01/2000'
             spf: [2x1 double]
        baseline: [2x1 double]
           units: [2x2 char]
           fname: [2x7 char]
           group: [2x1 double]
             fmt: [2x1 double]
            gain: [2x1 double]
          adcres: [2x1 double]
```

(continues on next page)

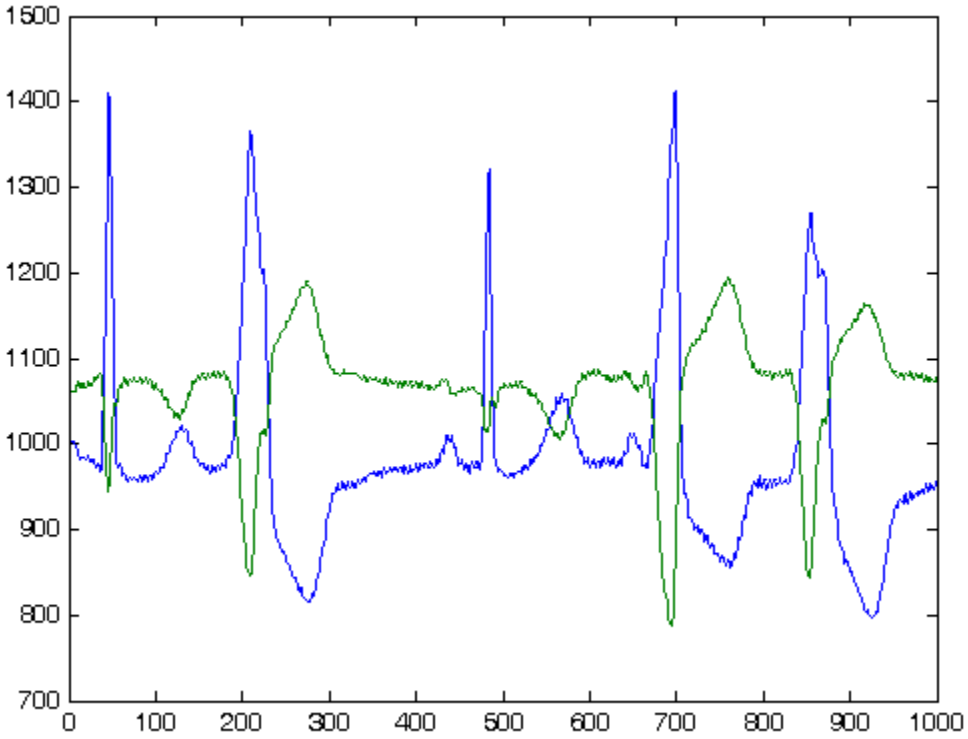
(continued from previous page)

```
          adczero: [2x1 double]
          initval: [2x1 double]
            cksum: [2x1 double]
            bsize: [2x1 double]
             desc: [2x5 char]
```

Then we get the first 1000 signal samples using the `read_signal` method.

```
>> plot(ECGw.read_signal(1,1000))
```

as result we obtain the following plot



2.3.9 Other resources

- Physionet.org
- Telemetric and Holter ECG Warehouse (THEW)
- Pablo Laguna research group at University of Zaragoza
- Computing in Cardiology

2.3.10 See Also

ECGtask | *Examples*

2.4 ECGtask

Perform an specific task to an ECG signal. This document defines the standard interface that tasks must implement.

2.4.1 QRS detection

This document describes how to perform QRS detection.

2.4.1.1 Description

Heartbeat detection is probably one of the first and most important tasks when you process cardiovascular recordings. The ECGkit has several algorithms implemented:

- Wavedet
- gqrs
- wqrs
- Pan and Tompkins
- ECGpuwave
- EP limited
- Aristotle (not distributed with the kit)

You can use any or all the algorithms as you will see below or you can even add your own algorithms if you follow an easy interface, as described *below*. The results are stored in a single file, that you can use to perform other subsequent tasks, such as *ECG delineation* or even the visual *inspection* or *correction* of the algorithms results. For a quick reference about heartbeat detection you may want to check this *example*

2.4.1.2 Input Arguments

The properties that this task uses are the following:

`progress_handle` — Used to track the progress within your function. `[] (default)`

> progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

`tmp_path` — The path to store temporary data. `tempdir() (default)`

> Full path to a directory with write privileges.

`detectors` — The QRS detection algorithms to use. `'all-detectors' (default)`

> This property controls which algorithms are used. A cell string or char array with any of the following names
>
> - all-detectors
> - wavedet
> - pantom
> - aristotle
> - gqrs
> - sqrs

- wqrs
- ecgpuwave
- epltdqrs1 or epltdqrs2

`only_ECG_leads` — Process only ECG signals. `true (default)`

Boolean value. Find out which signals are ECG based on their header description.

`gqrs_config_filename` — A configuration filename for the gqrs algorithm. `[] (default)`

A full filename with the configuration for the gqrs algorithm. See the algorithm web page for details.

`detection_threshold` — A threshold to control the sensitivity of the detector. `1 (default)`

Use higher values to reduce false detections, or lower values to reduce the number of missed beats.

`payload` — An arbitrary format variable to be passed to your user-defined algorithm. `[] (default)`

This variable can be useful for passing data to your own function, in addition to the interface described *below*.

`CalculatePerformance` — Calculate algorithm performances based on gold standard reference detections. `false (default)`

Boolean value. Calculate the algorithm performance based on the reference annotations found by the ECGwrapper object in the same folder where the signals are. This reference annotations are loaded, if detected, in the ECG_annotations property.

`bRecalcQualityAndPerformance` — Recalculate algorithm performances without performing heartbeat detection. `false (default)`

Boolean value. Recalculate algorithm performances without performing heartbeat detection. This feature is useful for BIG jobs where only a small change in performance calculation methodology was changed. . The cached result is passed to the task via the payload property as in the example:

```
% If only recaclulate performance is needed.
ECGw.ECGtaskHandle.bRecalcQualityAndPerformance = true;
% in order to avoid skiping the task
ECGw.cacheResults = false;
%payload has the current detections
cached_filenames = ECGw.GetCahchedFileName('QRS_detection');
ECGw.ECGtaskHandle.payload = load(cached_filenames{1});
```

`bRecalculateNewDetections` — Calculate only heartbeat detections not performed before. `false (default)`

Boolean value. The heartbeat detection is only performed in the algorithms not executed in a previous cached result. The cached result is passed to the task via the payload property as in the example:

```
% this is needed in order to recalculate tasks.
    % Useful if a new detector is added
    ECGw.ECGtaskHandle.bRecalculateNewDetections = true;
    % in order to avoid skiping the task
    ECGw.cacheResults = false;
    %payload has the current detections
    cached_filenames = ECGw.GetCahchedFileName('QRS_detection');
    ECGw.ECGtaskHandle.payload = load(cached_filenames{1});
```

2.4.1.3 Adding a custom detection algorithm

Adding your own QRS detectors to the kit is very simple. Ensure that your function implements this interface:

```
function [positions_single_lead, position_multilead] =

                    your_QRS_detector( ECG_matrix, ECG_header, progress_handle,␣
↪payload_in)
```

where the arguments are:

ECG_matrix, is a matrix size `[ECG\_header.nsamp ECG\_header.nsig]`

ECG_header, is a struct with info about the ECG signal, see *ECG header* for details.

progress_handle, is a handle to a progress_bar object, that can be used to track the progress within your function.

payload_in, is a user variable, of arbitrary format, allowed to be sent to your function. It is sent via the *payload property* of this class, for example:

```
% One variable
this_ECG_wrapper.ECGtaskHandle.payload = your_variable;

% Several variables with a cell container
this_ECG_wrapper.ECGtaskHandle.payload = {your_var1 your_var2};

% Or the result of a previous task, in this case QRS manual correction (if available)
% or the automatic detection if not.
cached_filenames = this_ECG_wrapper.GetCahchedFileName({'QRS_corrector' 'QRS_detection
↪'});
this_ECG_wrapper.ECGtaskHandle.payload = load(cached_filenames);
```

and the output of your function must be:

positions_single_lead, a cell array size ECG_header.nsig with the QRS sample locations found in each lead.

position_multilead, a numeric vector with the QRS locations calculated using multilead rules.

2.4.1.4 Examples

Create the ECGtask_QRS_detection object.

```
% with the task name
ECG_w.ECGtaskHandle = 'QRS_detection';

    % or create an specific handle to have more control
ECGt_QRSd = ECGtask_QRS_detection();
```

and then you are ready to set the algorithms to use. In the following example you have several possible set-ups.

```
% select an specific algorithm. Default: Run all detectors
ECGt_QRSd.detectors = 'wavedet'; % Wavedet algorithm based on
ECGt_QRSd.detectors = 'pantom';  % Pan-Tompkins alg.
ECGt_QRSd.detectors = 'gqrs';    % WFDB gqrs algorithm.
% Example of how you can add your own QRS detector.
ECGt_QRSd.detectors = 'user:example_worst_ever_QRS_detector';
```

(continues on next page)

(continued from previous page)

```
% "your_QRS_detector_func_name" can be your own detector.
ECGt_QRSd.detectors = 'user:your_QRS_detector_func_name';
ECGt_QRSd.detectors = {'wavedet' 'gqrs' 'user:example_worst_ever_QRS_detector'};
```

Finally set the task to the wrapper object, and execute the task.

```
ECG_w.ECGtaskHandle= ECGt_QRSd; % set the ECG task
ECG_w.Run();
```

You can check the result of this task, with either the *detection corrector* or the *visualization functions*.

Also check this *example* for further information.

2.4.1.5 Results format

The result file will have `ECG_header.nsig x algorithms_used` variables, which can later be recovered as a `struct` variable, with fields named according to `[ 'algorithm_name' '_' 'lead_name' ]`. Each of this fields is a `struct` itself with a single field called `time`, where the actual QRS detections are. In addition, another `struct` variable called `series_quality` is stored in order to provide a quality metric of the detections created. This metric is found in the `ratios` field, a higher ratio means better detections. Each ratio corresponds with a name in the `AnnNames` field.

2.4.1.6 More About

Here are some external references about heartbeat detection:

- PhysioNet/Computing in Cardiology Challenge 2014
- Physionet
- A video demo in Youtube

2.4.1.7 See Also

ECGtask | *ECG delineation* | *examples*

2.4.2 QRS correction

This document describes how to perform inspection and correction of automatic heartbeat detection.

2.4.2.1 Description

Automatic heartbeat detection is commonly well performed in those recordings with stable heart rhythms and QRS morphologies. In those cases where these situations are not met, many problems arise and automatic detection is not easy performed. This task provides a graphical user interface (GUI) to ease verification, correction and even manual detection.

2.4.2.2 Input Arguments

The properties that the ECGtask_QRS_corrector class handle are described below. The usage of these properties is restricted to low-level programming, you can use this task through the ECGwrapper as is shown in the example below.

`progress_handle` — Used to track the progress within your function. `[] (default)`

progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

`tmp_path` — The path to store temporary data. `tempdir() (default)`

Full path to a directory with write privileges.

`payload` — An arbitrary format variable to be passed to your user-defined algorithm. `[] (default)`

This propery is typically used to pass the automatic detection results. See the example below.

`caller_variable` — An arbitrary variable name in the caller workspace. `'payload' (default)`

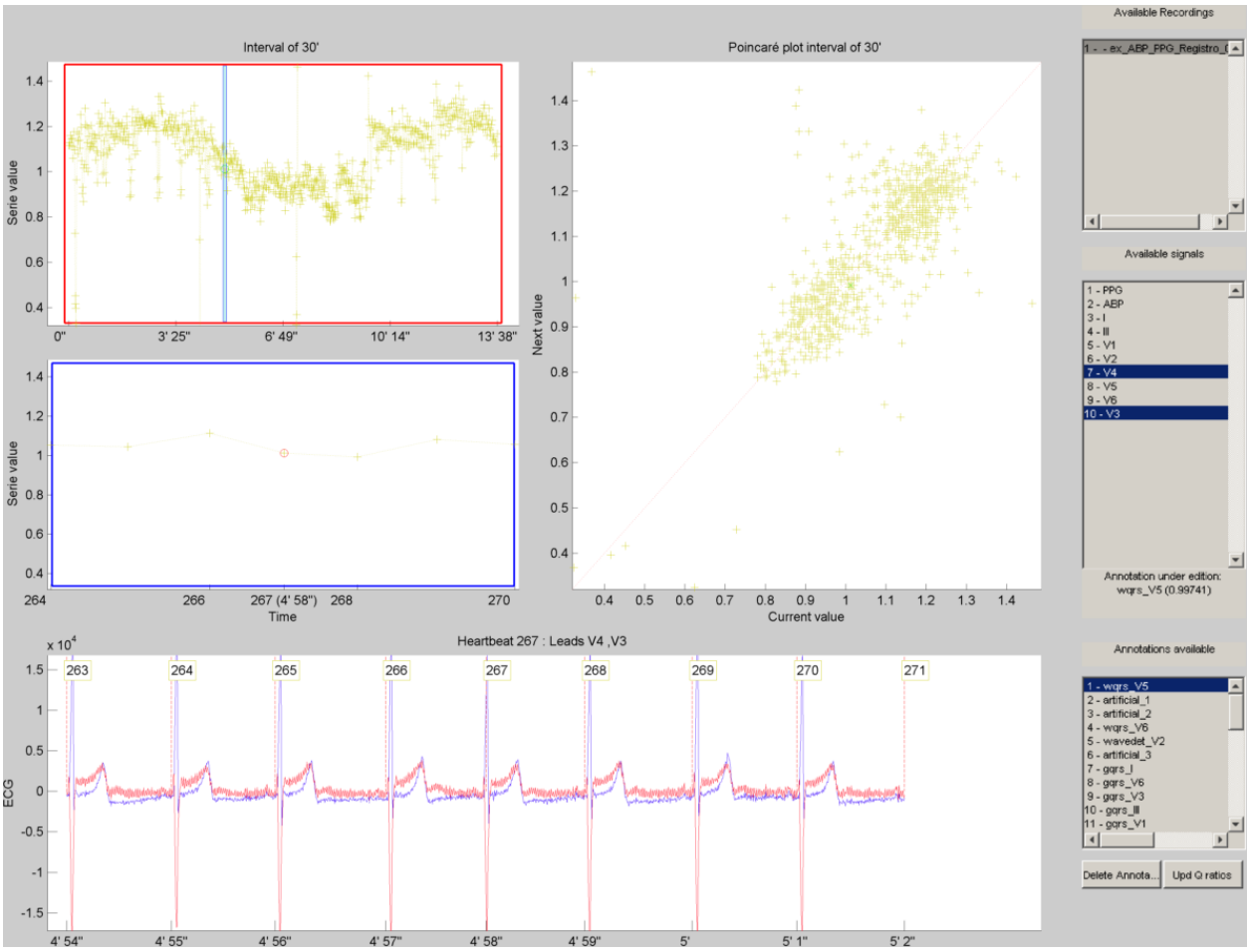
This variable in the caller workspace, will be assigned after user-interaction ends. ECGtask_QRS_corrector uses 'payload' as default variable in order to save the result of edition/verification with the GUI.

2.4.2.3 Examples

Create the ECGtask_QRS_corrector object.

```
ECGw.ECGtaskHandle = 'QRS_corrector';
% this is to use previous saved results as starting point, if any available
cached_filenames = ECG_all_wrappers(ii).GetCahchedFileName({'QRS_corrector' 'QRS_
↪detection'});
ECGw.ECGtaskHandle.payload = load(cached_filenames{1});
ECGw.Run();
```

Then the following GUI appears



and the command window shows the following message:

```
##############################
# User interaction required #
##############################
This ECGtask allow user interaction. Press [CTRL + G] in figure 1 to save results and␣
↪press F5 (Run) to continue.
K>>
```

see the videos in YouTube for a more detailed demo about things you can do with the GUI. The demo shows how to add/remove heartbeats annotation, browse the detections through the whole recording and cut/copy/paste heartbeat detections between different annotations. Other features not described in this video were added in this other. After edition/verification of the automatic delineation, press CTRL+G to save results in the 'payload' variable of the caller workspace. Then press F5 to save the results to disk.

2.4.2.4 Results format

The result file have the same format than *QRS detection task*.

2.4.2.5 More About

Here are some external references about heartbeat detection:

- A video demo in Youtube

2.4.2.6 See Also

ECGtask | *ECG delineation* | *examples*

2.4.3 ABP/PPG peak detection

This document describes how to perform automatic peak detection in pulsatile signals.

2.4.3.1 Description

This task perform peak detection in pulsatile signals such as arterial blood pressure (ABP) or plethysmographic (PPG). The task uses two algorithms to achieve pulse detection, and, as in QRS detection task you can choose to use any of them.

2.4.3.2 Input Arguments

The properties that the ECGtask_PPG_ABP_detector class accepts are described below. The usage of these properties is restricted to low-level programming, you can use this task through the ECGwrapper as is shown in the example below.

`progress_handle` — Used to track the progress within your function. `[] (default)`

progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

`tmp_path` — The path to store temporary data. `tempdir() (default)`

A folder to store temporary data. Full path to a directory with write privileges.

`lead_config` — Select which signals to process. `'PPG-ABP-only' (default)`

This property control on which signals the pulse detection algorithms will be applied. A cell string or char with any of the following names:

- '*all-leads*'. Process all leads.
- '*PPG-ABP-only*'. Detect pulsatile signals based on their `ECG_header.desc` description variable.
- '*User-defined-leads*'. Tell the algorithm (with `PPG_ABP_idx property`) which signal indexes to process, from 1 to `ECG_header.nsig`

`PPG_ABP_idx` — The indexes corresponding to pulsatile signals `[] (default)`

A value from 1 to `ECG_header.nsig` indicating the column indexes (of the signal matrix `ECG`) where pulsatile signals are located. By default this task process all signals.

`detectors` — The PPG/ABP detection algorithms to use `'all-detectors' (default)`

Select which algorithm to use. A cell string or char with any of the following names:

- 'all-detectors'
- 'wavePPG'
- 'wabp'

2.4.3.3 Examples

Create the *ECGtask_PPG_ABP_detector* object.

```
% with the task name
    ECG_w.ECGtaskHandle = 'PPG_ABP_detector';
% or create an specific handle to have more control
    ECGt_PPG = ECGtask_PPG_ABP_detector();
```

and then you are ready to set the algorithms to use. In the following example you have several possible set-ups.

```
% select an specific algorithm. Default: Run all detectors
    ECGt_PPG.detectors = 'wavePPG'; % A J. Lazaro algorithm for peak detection
    ECGt_PPG.detectors = 'wabp';  % Another algorithm from Physionet
```

Finally set the task to the wrapper object, and execute the task.

```
ECG_w.ECGtaskHandle= ECGt_PPG; % set the ECG task
ECG_w.Run();
```

You can check the result of this task, with either the *detection corrector* or the *visualization functions*.

Also check this *example* for further information.

2.4.3.4 Results format

The result file have the same format than *QRS detection task*.

2.4.3.5 More About

Here are some external references about pulse detection:

- **?? Add some**

2.4.3.6 See Also

ECGtask | *QRS detection* | *examples*

2.4.4 ABP/PPG peak correction

This document describes how to perform inspection and correction of automatic peak detection in pulsatile signals.

2.4.4.1 Description

This task is a clone of the *QRS corrector* task, for visual inspection and correction of peaks automatically detected. See the *help* for further reference and also check this *example*.

2.4.4.2 See Also

ECGtask | *Pulse detection* | *examples*

2.4.5 ECG delineation

This document describes how to perform automatic delineation or wave segmentation on ECG signals.

2.4.5.1 Description

Automatic wave segmentation or delineation is exclusively performed by wavedet algorithm.

2.4.5.2 Input Arguments

The properties that this task uses are the following:

`progress_handle` — Used to track the progress within your function. `[] (default)`

progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

`tmp_path` — The path to store temporary data. `tempdir() (default)`

Full path to a directory with write privileges.

`delineators` — The ECG delineation algorithms to use `'all-delineators' (default)`

This property controls which algorithms are used. A cell string or char with any of the following names

- *'all-delineators'*
- 'wavedet'

`only_ECG_leads` — Process only ECG signals `true (default)`

Boolean value. Find out which signals are ECG based on their `ECG_header.desc` description.

`wavedet_config` — A structure for configuring wavedet algorithm. `[] (default)`

Undocumented yet, use it only if you know what you are doing.

`payload` — An arbitrary format variable. `[] (default)`

This variable can be useful for passing data to your own delineation function (described *below*) or to provide visually audited QRS detections to the delineation algorithm.

2.4.5.3 Adding a custom delineation algorithm

Adding your own delineator to the kit is very simple. Ensure that your function implements this interface:

```
function [positions_single_lead, position_multilead] =

            your_ECG_delineation( ECG_matrix, ECG_header, progress_handle, payload_in)
```

where the arguments are:

ECG_matrix, is a matrix size [ECG_header.nsamp ECG_header.nsig]

ECG_header, is a struct with info about the ECG signal, such as:

- *freq*, is the sampling frequency of ECG_matrix signal.
- *desc*, description strings about each of the leads/signals.
- *nsamp* is the number of samples of ECG_matrix.

- *nsig* is the amount of leads or signals of ECG_matrix.
- *gain* is a vector of [nsig × 1] with the gain of each lead (ADCsamples / μV).
- *adczero* is a vector of [nsig × 1] with the offset of each lead in ADC samples.

and others described in the Physionet header.

progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

payload_in, is a user variable, of arbitrary format, allowed to be sent to your function. It is sent via the *payload property* of this class, for example:

```
% One variable
this_ECG_wrapper.ECGtaskHandle.payload = your_variable;

% Several variables with a cell container
this_ECG_wrapper.ECGtaskHandle.payload = {your_var1 your_var2};

% Or the result of a previous task, in this case QRS manual correction (if available)
% or the automatic detection if not.
cached_filenames = this_ECG_wrapper.GetCahchedFileName({'QRS_corrector' 'QRS_detection
↪'});
this_ECG_wrapper.ECGtaskHandle.payload = load(cached_filenames);
```

and the output of your function must be:

positions_single_lead, is an **structure array** of `ECG_header.nsig` elements with *at least* the following wave fiducial points as fields:

- `'Pon'` P wave onset
- `'P'` P wave peak
- `'Poff'` P wave offset
- `'QRSon'` QRS complex onset
- `'qrs'` QRS fiducial point, obtained from QRS detection.
- `'Q'` Q wave peak
- `'R'` R wave peak
- `'S'` S wave peak
- `'QRSoff'` QRS complex offset
- `'Ton'` T wave onset
- `'T'` T wave peak
- `'Toff'` T wave offset

position_multilead, is a single structure with *at least* the wave fiducial points described above. This delineation is commonly calculated from the single lead delineations, in order to obtain a unique wave fiducial point per heartbeat.

2.4.5.4 Examples

Create the *ECGtask_ECG_delineation* object.

```
% with the task name
    ECG_w.ECGtaskHandle = 'ECG_delineation';
% or create an specific handle to have more control
    ECGt = ECGtask_ECG_delineation();
```

and then you are ready to set the algorithms to use. In the following example you have several possible set-ups.

```
% select an specific algorithm. Default: Run all detectors
        ECGt.delineators = 'wavedet'; % Wavedet algorithm based on
        % "your_delineator_func_name" can be your own delineator.
                    ECGt.delineators = 'user:your_delineator_func_name';
        ECGt.delineators = {'wavedet' 'user:your_delineator_func_name'};
```

Finally set the task to the wrapper object, and execute the task.

```
ECG_w.ECGtaskHandle= ECGt; % set the ECG task
ECG_w.Run();
```

You can check the result of this task, with either the *delineator corrector* or the *visualization functions*.

Also check this *example* for further information.

2.4.5.5 Results format

The result file will have a `struct` variable with the name of the algorithm (only *wavedet* at the time of writing this). Inside this, it will contain one *delineation struct* per ECG lead in the `ECG_header.desc` field, plus another called `multilead` which is a delineation accounting with the information present in all leads.

2.4.5.6 More About

This publication describes the wavedet algorithm.

2.4.5.7 See Also

ECGtask | *QRS detection* | *examples*

2.4.6 ECG delineation correction

This document describes how to visualize and correct automatic delineation.

2.4.6.1 Description

This task is a clone of the *QRS corrector* task, for the visualization and inspection of automatic delineator results. See the *help* for further reference and also check this *example*.

2.4.6.2 More About

- **?? Add some**

2.4.6.3 See Also

ECGtask | *ECG delineation* | *examples*

2.4.7 ECG heartbeat classification

This document describes how to classify heartbeats according to its origin.

2.4.7.1 Description

This task implements a heartbeat classifier that follows the EC-57 AAMI recommendation classifying heartbeats into four classes:

- **N** normal
- **S** supraventricular
- **V** ventricular
- **F** fusion of normal and ventricular

Certain background and introduction to this topic is included in my PhD thesis.

2.4.7.2 Input Arguments

`progress_handle` — Used to track the progress within your function. `[] (default)`

progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

`tmp_path` — The path to store temporary data. `tempdir() (default)`

Full path to a directory with write privileges.

`` `payload `` — A structure to provide audited heartbeat detections to the classifier algorithm. `[] (default)`

This variable is useful to pass automatic or corrected QRS detections to the classification task. This can be performed as shown in the following example:

```
cached_filenames = ECGw.GetCahchedFileName({'QRS_corrector' 'QRS_detection'});
ECGw.ECGtaskHandle.payload = load(cached_filenames{1});
```

`mode` — Set the classification mode of operation. `'auto' (default)`

A control string with any of the following names

- 'auto', this mode makes the algorithm operate in automatic mode.
- 'slightly-assisted', this mode requires that an expert labels several representative examples, when the algorithm does not reach a confidence level to do it automatically.
- 'assisted', this mode is completely assisted. An expert must label all the representative heartbeats from each cluster.

2.4.7.3 Examples

The first example shows the simplest setup of the *ECGtask_heartbeat_classifier* object, while at the end of this section a complete example with a real signal is shown.

```
% with the task name
ECG_w.ECGtaskHandle = 'ECG_heartbeat_classifier';
% or create an specific handle to have more control
ECGt = ECGtask_heartbeat_classifier();
```

and then you are ready to setup the task

```
% select a mode, automatic mode does not require assistance
ECGt.mode = 'auto';
% this is to use QRS detection previously calculated
cached_filenames = ECG_all_wrappers(ii).GetCahchedFileName({'QRS_corrector' 'QRS_
↪detection'});
ECGt.payload = load(cached_filenames{1})
```

Finally set the task to the wrapper object, and execute the task.

```
ECG_w.ECGtaskHandle= ECGt; % set the ECG task
ECG_w.Run();
```

This example shows in first place, the previous configuration used in recording 208 from MIT Arrhythmia database.

```
>> ECG_w = ECGwrapper( ...
        'recording_name', 'some_path\208', ...
        'recording_format', 'MIT', ...
        'ECGtaskHandle', 'ECG_heartbeat_classifier', ...
        )ECG_w =
############################
# ECGwrapper object config #
############################
+ECG recording: some_path\208 (auto)
+PID: 1/1
+Repetitions: 1
+Partition mode: ECG_overlapped
+Function name: ECG_heartbeat_classifier
+Processed: false

>> ECG_w.Run();
```

You can follow the evolution in the progress bar, and after a while, it ends and display the classification results

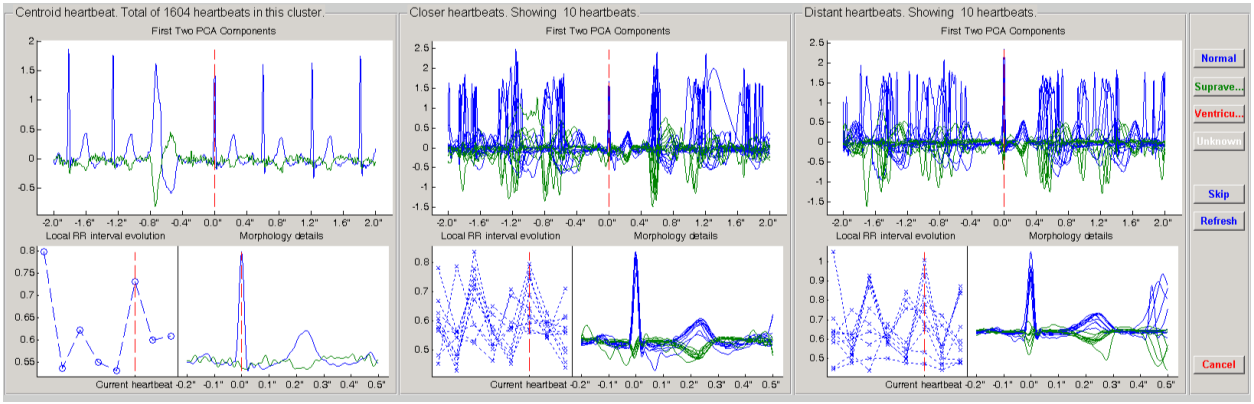
```
Configuration
-------------
+ Recording: ... \example recordings\208.dat (MIT)
+ Mode: auto (12 clusters, 1 iterations, 75% cluster-presence)

  True             | Estimated Labels
  Labels           | Normal Suprav Ventri Unknow| Totals
 ------------------|----------------------------|-------
  Normal           | 1567      6     13      0  | 1586
  Supraventricular|    2      0      0      0  |    2
  Ventricular      |  255      8   1102      0  | 1365
  Unknown          |    2      0      0      0  |    2
 ------------------|----------------------------|-------
```

(continues on next page)

(continued from previous page)

```
  Totals          | 1826     14    1115      0   | 2955

Balanced Results for
---------------------
| Normal     || Supravent || Ventricul ||           TOTALS            |
|  Se    +P  ||  Se    +P  ||  Se    +P  ||   Acc    |   Se     |   +P     |
|  99%   45% ||   0%    0% ||  81%   99% ||   60%    |   60%    |   48%    |

Unbalanced Results for
----------------------
| Normal     || Supravent || Ventricul ||           TOTALS            |
|  Se    +P  ||  Se    +P  ||  Se    +P  ||   Acc    |   Se     |   +P     |
|  99%   86% ||   0%    0% ||  81%   99% ||   90%    |   60%    |   62%    |
```

This is possible because this recording include the expert annotations, or ''ground truth'', for each heartbeat. The manual annotations in MIT format are typically included in ''.atr" files (in this case "208.atr''). Now you can try ''slightly-assisted" mode, where the algorithm may ask you for help in case of cluster heterogeneity. If this happens, a window like this will appear:



In this window the algorithm is asking you to label the centroid of the cluster, that is showed in the left panel. In the top of each panel some information is showed, as the amount of heartbeats in the current cluster. In the middle panel, you have some examples of heartbeats close to the centroid in a likelihood sense. The same is repeated in the right panel, but with examples far from the centroid. This manner you can have an idea of the dispersion of heartbeats within a cluster. Large differences across the panels indicates large cluster dispersion. If you decide to label the cluster, you can use one of the 4 buttons on your right. The unknown class is reserved for the cases where you can not make a confident decision. At the same time, in the command window, a suggestion appears:

```
Configuration
-------------
+ Recording: .\example recordings\208.dat (MIT)
+ Mode: assisted (3 clusters, 1 iterations, 75% cluster-presence)
Suggestion: Normal
```

This means that the centroid heartbeat in the ''.atr" file is labeled as ''Normal''. You will see this suggestion for each cluster analyzed, if there are annotations previously available. You are informed about the percentage of heartbeats already labeled with a progress bar, in the bottom of the control panel window.

In case you believe that a cluster includes several classes of heartbeats, you can decide to ''skip" the classification, and try to re-cluster those heartbeats in the next iteration. You are free to perform as many iterations as you decide, by skipping clusters. The refresh button resamples heartbeats close and far from the centroid, and then redraw the middle and right panels. This feature is useful for large clusters.

You can check the result of this task for every heartbeat in the recording using the *visualization functions*.

Also check this *example* for further information.

2.4.7.4 Results format

The results file includes three variables, the annotation type or classification label `anntyp`, containing a `char` label per heartbeat, which is the initial letter of the heartbeat label. A vector of samples called `time` (in correspondence with `anntyp`), with the occurrence of each heartbeat labeled in this task. The last variable, is a label list called `lablist`, which is a cell array of strings with the full name of each label in `anntyp`.

2.4.7.5 More About

Here are some external references about heartbeat classification:

- EC-57 AAMI recommendation
- EP limited software

2.4.7.6 See Also

ECGtask | *QRS detection* | *examples*

2.4.8 Arbitrary tasks

This document describes how to use arbitrary tasks with the ECGkit.

2.4.8.1 Description

Sometimes the task you need to perform on ECG signals is too simple to develop a new ECGtask, like computing some statistics, or apply a linear filter, or any type of transformation you may need to perform to the signal. For those cases you may found arbitrary tasks useful.

2.4.8.2 Input Arguments

The properties that this task uses are the following:

`progress_handle` — Used to track the progress within your function. `[] (default)`

progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

`tmp_path` — The path to store temporary data. `tempdir() (default)`

Full path to a directory with write privileges.

`only_ECG_leads` — Process only ECG signals `true (default)`

Boolean value. Find out which signals are ECG based on their `ECG_header.desc` description.

`payload` — An arbitrary format variable to be passed to your function. `[] (default)`

This variable can be useful for passing data to your own function.

`signal_payload` — Consider the result of your arbitrary function as a signal. `false (default)`

Boolean value that indicates the ECGwrapper to produce a signal instead of a result payload.

`lead_idx` — The signal indexes that your function will affect. `[] (default)`

A positive integer array with values from 1 to `ECG_header.nsig`.

`function_pointer` — The pointer to your arbitrary function. `[] (default)`

Your function must follow this prototype:

```
function result = your_function( ECG_matrix, ECG_header, progress_handle, payload_in)
```

where the arguments are:

ECG_matrix, is a matrix size `[ECG\_header.nsamp ECG\_header.nsig]`

ECG_header, is a struct with info about the ECG signal, such as:

- *freq*, is the sampling frequency of ECG_matrix signal.
- *desc*, description strings about each of the leads/signals.
- *nsamp* is the number of samples of ECG_matrix.
- *nsig* is the amount of leads or signals of ECG_matrix.
- *gain* is a vector of [nsig × 1] with the gain of each lead (ADCsamples / μV).
- *adczero* is a vector of [nsig × 1] with the offset of each lead in ADC samples.

and others described in the Physionet header.

progress_handle, is a handle to a *progress_bar* object, that can be used to track the progress within your function.

payload_in, is a user variable, of arbitrary format, allowed to be sent to your function. It is sent via the *payload property* of this class, for example:

```
% One variable
this_ECG_wrapper.ECGtaskHandle.payload = your_variable;

% Several variables with a cell container
this_ECG_wrapper.ECGtaskHandle.payload = {your_var1 your_var2};

% Or the result of a previous task, in this case QRS manual correction (if available)
% or the automatic detection if not.
cached_filenames = this_ECG_wrapper.GetCahchedFileName({'QRS_corrector' 'QRS_detection
↪'});
this_ECG_wrapper.ECGtaskHandle.payload = load(cached_filenames);
```

and the output of your function must be a struct variable **result**, or if it is a signal, ensure to make `true` the `signal_payload` property.

`finish_func_pointer` — A pointer to your arbitrary finish function. `@default_finish_function (default)`

A function that will operate over the whole result of your arbitrary function, after the payloads resulting of each iteration were concatenated. This is only used when the result of your `function_pointer` is **not** a signal (`signal_payload = false`). Your function must follow this prototype:

```
payload = your_finish_function(payload, ECG_header)
```

where the arguments are:

payload, is the complete payload.

ECG_header, is a struct with info about the ECG signal, see above for reference.

and this function will change the payload variable as according to your needs and return it to the ECGwrapper object.

`concate_func_pointer` — The pointer to your arbitrary concatenate function. `@default_concatenate_function (default)`

A function that will concatenate or integrate the information produced in each part of your recording, when the result of your `function_pointer` is **not** a signal (`signal_payload = false`). Your function must follow this prototype:

```
payload = your_concatenate_function(plA, plB)
```

where the arguments are:

plA and **plB** are the two payloads to concatenate

and this function will integrate or concatenate both payloads into the resulting payload. This resulting payload, will be plA in the next iteration of concatenation. The `default_concatenate_function` just concatenate payloads:

```
% The default behavior of the concatenate function is to concatenate
% payloads vertically or row-wise.
if( isempty(plA) )
        payload = plB;
else
        payload = [plA; plB];
end
```

2.4.8.3 Examples

1. Arbitrary task producing a **signal** as a result

   This example is used in the QRScorrector function to perform template-matching on an ECGwrapper (arbitrary big recording) object.

   ```
   aux_w = ECGwrapper('recording_name', 'your_path/recname');
   aux_w.ECGtaskHandle = 'arbitrary_function';

   % This is in case you want always to recalculate results, no caching
   aux_w.cacheResults = false;

   % Use first and third columns-signals
   aux_w.ECGtaskHandle.lead_idx = [1 3];

   % Produce a signal as a result
   aux_w.ECGtaskHandle.signal_payload = true;

   % Add a user-string to identify the run
   aux_w.user_string = ['similarity_calc_for_lead_' num2str(sort(lead_idx)) ];

   % add your function pointer
   aux_w.ECGtaskHandle.function_pointer = @similarity_calculation;

   % and any data your function may need.
   aux_w.ECGtaskHandle.payload = pattern2detect;
   % and you are ready to go !
   aux_w.Run
   ```

2. Arbitrary task producing an **arbitrary result**

   This is achieved by defining 3 properties (function handles) that perform:

- The arbitrary task, which produces an arbitrary result `function_pointer`
- The concatenation of these results `concate_func_pointer`
- The final result calculation, when all results are concatenated. `finish_func_pointer`

The configuration of the ECGwrapper object is quite simple:

```
cd your_path\ecg-kit\examples
ECGw = ECGwrapper( 'recording_name', 'your_path\ecg-
↪kit\recordings\208')
% no overlapp needed between signal partitions
ECGw.partition_mode = 'ECG_contiguous';
ECGw.ECGtaskHandle = 'arbitrary_function';
ECGw.ECGtaskHandle.function_pointer = @my_mean;
ECGw.ECGtaskHandle.concate_func_pointer = @my_concatenate_
↪mean;
ECGw.ECGtaskHandle.finish_func_pointer = @my_finish_mean;
ECGw.Run
```

The result is stored in a `mat` file.

```
Description of the process:
 + Recording: d:\mariano\misc\ecg-kit\recordings\208.dat
 + Task name: arbitrary_function


###############
# Work done! #
###############


Results saved in
 + your_path\ecg-kit\recordings\208_arbitrary_function.mat
```

The arbitrary functions used to calculate the mean in an arbitrary large recording are:

- `\ecg-kit\examples\my_mean.m` In this function we only accumulate and count the size of the accumulation.

```
function result = my_mean(x)

result.the_sum = sum(x);
result.the_size = size(x,1);
```

- `\ecg-kit\examples\my_concatenate_mean.m` This function calculate the final accumulation and counting.

```
function payload = my_concatenate_mean(plA, plB)

if( isempty(plA) )
        payload = plB;
else
        payload.the_sum = plA.the_sum + plB.the_sum;
        payload.the_size = plA.the_size + plB.the_size;
end
```

- `\ecg-kit\examples\my_finish_mean.m` In this function the mean calculation is performed.

```
function result_payload = my_finish_mean(payload, ECG_header)

result_payload.mean = payload.the_sum ./ payload.the_size;
```

2.4.8.4 Results format

The format of the results depends on the `signal_payload` property, if it is a signal it will be in MIT format. Otherwise, the results depends on the user-defined output of

2.4.8.5 See Also

ECGtask | *QRS detection* | *ECG delineation* | *examples*

2.4.9 Description

The ECGtask is an abstract class definition where the minimum interface requirements are specified, in order that your own tasks can be safely plugged into *ECGwrapper <ECGwrapper>* objects. As an example of how to use this interface, see the derived classes for *QRS detection* and *ECG delineation*, among others that can be listed with the *list_all_ECGtask* function:

- *QRS detection*
- *QRS correction*
- *ECG delineation*
- *ECG delineation correction*
- *ABP/PPG peak detection*
- *ABP/PPG peak correction*
- *Heartbeat classification*
- *Arbitrary tasks*

These tasks are the core of this kit and you will probably refer to them before you extend the functionality with your own tasks.

2.4.10 Properties

All tasks must implement the following properties with its attributes:

```
properties(GetAccess = public, Constant)
```

`name` — The name of the task.

`target_units` — The signal units required by the task. Possible values are:

ADCu, raw ADC samples.

nV, uV, mV, V, voltage.

`doPayload` — Boolean. Does this task generates a payload to be stored ?

```
properties(GetAccess = public, SetAccess = private)
```

`memory_constant` — A coefficient to indicate the ECGwrapper how big should be a batch processing part. The size of each part is calculated as

```
user = memory;
batch_size = memory_constant * user.MaxPossibleArrayBytes;
```

`started` — Boolean. Did the task executed the Start method ?

```
properties(GetAccess = public, SetAccess = public)
```

`progress_handle` — is a handle to a *progress_bar* object, that can be used to track the progress within your function.

`tmp_path` — The path to store temporary data.

2.4.11 Methods

All tasks must implement the following methods:

`Start` — The task initialization method.

This task initialize specific aspects of the task.

```
Start(obj, ECG_header, ECG_annotations)
```

where the arguments are:

> **ECG_header**, is a struct with info about the ECG signal, See *here* for a description.
>
> **ECG_annotations**, Commonly QRS detections, signal quality annotations or other type of measurements included with the recordings. Some documentation about annotations in Physionet.

`Process` — The task core processing function.

This task is the responsible of do the actual work of the ECGtask. This mehtod is called by an ECGwrapper all the times needed to process the whole recording.

```
payload = Process(ECG,
                  ECG_start_offset,
                  ECG_sample_start_end_idx,
                  ECG_header,
                  ECG_annotations,
                  ECG_annotations_start_end_idx )
```

where the arguments are:

> **ECG**, is a matrix size `[ECG_header.nsamp ECG_header.nsig]`
>
> **ECG_start_offset**, is the location of ECG(1,:) within the whole signal.
>
> **ECG_header**, is a struct with info about the ECG signal, See *here* for a description.
>
> **ECG_annotations**, Commonly QRS detections, signal quality annotations or other type of measurements included with the recordings. Some documentation about annotations in Physionet.
>
> **ECG_annotations_start_end_idx**, are the start and end indexes corresponding to the first and last element of ECG_annotations in the current iteration.

as a result, this method must produce a **payload** variable, that will be handled by the ECGwrapper object.

`Concatenate` — This method is responsible of the payload union after all the processing.

After the execution of all *Process* steps, each payload must be put together with this method. The ECGwrapper object will call this method once for each payload created, building a final payload.

```
payload = Concatenate(plA, plB)
```

where the arguments are:

plA and **plB**, are two payloads created with the *Process* method.

and as a result, this method creates **payload**, the union of **plA** and **plB**.

`Finish` — This task perform the last calculation over the whole payload.

After the concatenation of payloads, the whole payload is sent to this method to perform any final calculation.

```
payload = Finish(obj, payload, ECG_header)
```

where the arguments are:

payload, is the payload created with all the *Concatenate* method invocation.

ECG_header, is a struct with info about the ECG signal, See *here* for a description.

As a result, the final payload is generated, which later will be stored by the ECGwrapper object.

2.4.12 More About

- Physionet.org
- Telemetric and Holter ECG Warehouse (THEW)
- Pablo Laguna research group at University of Zaragoza
- Computing in Cardiology

2.4.13 See Also

ECGwrapper | *ECG_delineation* | *list_all_ECGtask*

2.5 Accessing results

Results are stored in a `mat` file for compatibility reasons. The format depends on the task that generated the results, but a typical procedure to grab data from experiments is:

```
ECGw = ECGwrapper('recording_name', 'your_rec_filename');
result_filename = ECGw.GetCahchedFileName('QRS_detection');
results = load(cached_filenames{1});
```

In this example, the results from the previous QRS detection experiment is loaded in the `results` variable. The format for the specific tasks was described in the following links:

- *QRS detection*
- ABP/PPG pulse detection tasks have the same format of *QRS detection*
- *ECG delineation*
- *Heartbeat classifier*

- *Arbitrary tasks*

2.6 reportECG

2.6.1 Description

This function creates a report of a signal handled by an ECGwrapper object. The report includes several views of the signals at different time scales. In addition, you have the possibility to overprint information from other ECGtask results, such as QRS detections, wave delineation, and heartbeat types. Some aspects of the report can be configured as the detail degree, the length of each time scale and the report format.

2.6.1.1 Plotting signals and task results

Low level function to produce charts and interactively browse signals and annotations produced by other ECGtasks

Some of the relevant features:

- User can interact using mouse shortcuts with several aspects of the visualization, such as zoom, pan and measurements.
- Information of the multilead wave boundaries can be added to the ECG, for example the delineation obtained with wavedet.
- It can "pretty" present the ECG charts for printing to pdf documents
- It can be easily added to your project for debug or result presentation through its versatile interface.

The mouse interaction was adapted from the Dragzoom function, by Evgeny Pr.

Prototype

The function prototype is the following.

```
function ECG_hdl = plot_ecg_strip( ECG, varargin )
```

This function accepts as arguments the arg_name/arg_val method described below.

Arguments

- ECG: [numeric | char | ECGwrapper] REQUIRED. Signal matrix of dimension `[nsamp nsig]` where:
  - nsamp: time length in samples
  - nsig: number of ECG leads or number of signals.

  A recording full-path filename or an ECGwrapper object are also accepted.
- ECG_header: [struct] OPTIONAL. Description of the ECG typically available in the header. See *ECG header* description.
- Start_time: [numeric] OPTIONAL. Start time in seconds. `0 (default)`
- End_time: [numeric] OPTIONAL. Start time in seconds. `end (default)`
- QRS_locations: [numeric] OPTIONAL. QRS complex detection samples. `[] (default)`

- QRS_start_index: [numeric] OPTIONAL. Start at the i-th QRS_start_index heartbeat in QRS_locations, or `QRS_locations(QRS_start_index).1 (default)`
- QRS_complexes: [numeric] OPTIONAL. Display the amount of QRS_start_complexes heartbeats from the QRS_start_index. `QRS_start_index + 10 (default)`
- Lead_offset: [numeric] OPTIONAL. A DC value `[nsig 1]` to be added to each lead. `zeros(nsig,1) (default)`
- Lead_gain: [numeric] OPTIONAL. A value `[nsig 1]` to be multiplied by each lead. `ones(nsig,1) (default)`
- ECG_delineation_single_lead: [struct/ECGwrapper] OPTIONAL. Annotation struct of size `[nsig 1]` described *here*. Each field of size `[1 nhb]`, being `nhb` the amount of heartbeats. If an ECGwrapper was provided, it tries to get results from an `ECG_delineation` task. `[] (default)`
- ECG_delineation_multilead: [struct/ECGwrapper] OPTIONAL. Annotation struct with the same fields of and characteristics of `ECG_delineation_single_lead`. If an ECGwrapper was provided, it tries to get results from an `ECG_delineation` task. `[] (default)`
- Heartbeat_classification: [struct/ECGwrapper] OPTIONAL. Heartbeat labels provided as a struct with the following fields
  - time, an array of `[nhb 1]` with the sample locations of each heartbeat.
  - anntyp, a char array of `[nhb 1]` with the label of each heartbeat, according to Physionet or the EC-57 AAMI recommendation. Commonly N, S, V, F, Q.

  If an ECGwrapper was provided, it tries to get results from an `ECG_heartbeat_classifier` task. `[] (default)`
- Title: [string] OPTIONAL. Description title. `[recname - time interval] (default)`
- DetailLevel: [string] OPTIONAL. The details included in the ECG plot depends on the zoom level and the data provided. Possible values:
  - 'all', overprint all info available.
  - 'single-lead', overprint ECG delineation results.
  - 'multilead', overprint ECG multilead delineation results.
  - `'none' (default)`, display only signals.
- OnlyECG: Display only ECG signals. Filter non-ECG signals based on their descriptions in `ECG_header.desc` field.
- FilterECG: Perform standard noise removal on ECG signals. Low pass @ 35Hz and baseline wander removal by cubic splines or median filtering.
- PrettyPrint: [bool] OPTIONAL. Prepare the plot for printing as a PDF. `false (default)`
- ReportFilename: a string with the full filename to export screen captures. `[signal_path\signal_name.pdf] (default)`
- Figure_handle: [axes handle] OPTIONAL. Choose the figure to display the plot. `gca (default)`

Mouse interaction

- Keyboard hotkeys

  h : Show this help

  + : Zoom plus

- : Zoom minus

d : Toggle the detail level of the annotations

a : Toggle the annotations graph mode.

0 : Set default axes (reset to original view)

c : On/Off pointer in crosshair mode

g : If pressed and holding, change lead gain with scroll

o : If pressed and holding, change lead offset with scroll

x : If pressed and holding, zoom and drag works only for X axis

y : If pressed and holding, zoom and drag works only for Y axis

m : If pressed and holding, Magnifier mode on

p : On/Off paper mode

r : Export format (PDF/PNG)

s : Export current view

- Normal mode

  single-click and holding LB : Activation Drag mode

  single-click and holding RB : Activation rubber band for region zooming

  single-click MB : Activation measuring rubber band mode

  scroll wheel MB : Activation Zoom mode

  double-click LB, RB, MB : Reset to Original View

- Magnifier mode (**m** key)

  single-click LB : Not Used

  single-click RB : Not Used

  single-click MB : Reset Magnifier to Original View

  scroll MB : Change Magnifier Zoom

  double-click LB : Increase Magnifier Size

  double-click RB : Decrease Magnifier Size

Examples

The easiest way of invoking this function is via an ECGwrapper object:

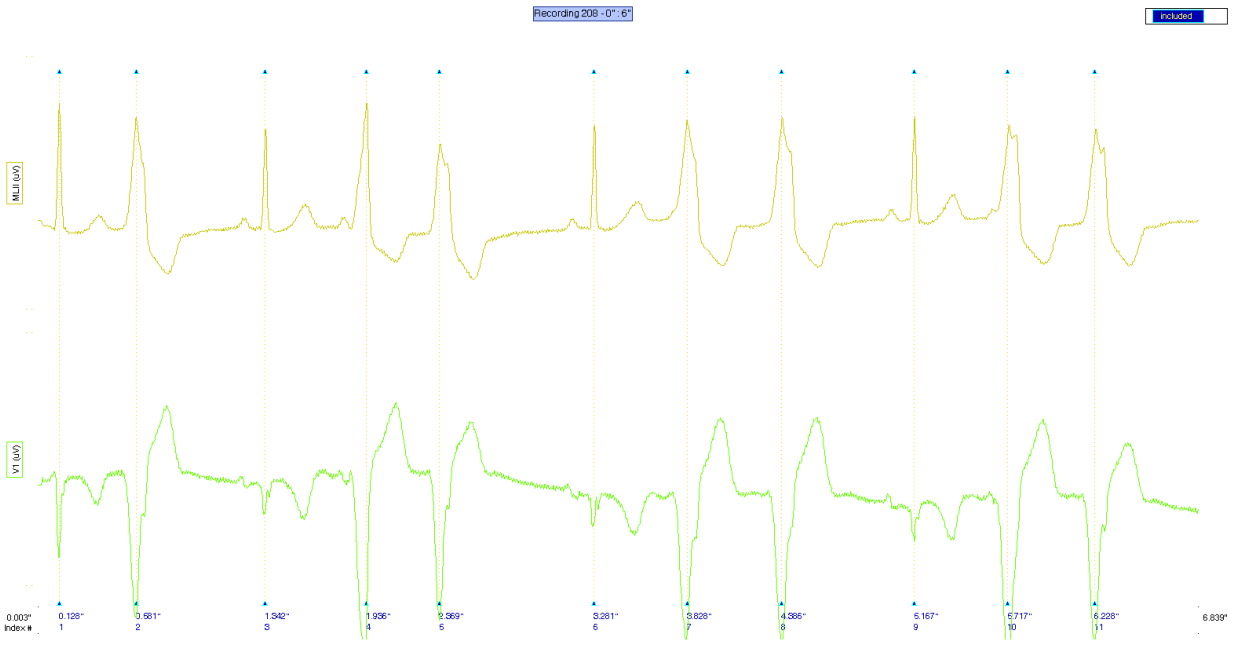
```
>> plot_ecg_strip(ECGw)

########################
# plot_ecg_strip help #
########################

Mouse actions:

  Normal mode:
          single-click and holding LB : Activation Drag mode
```

(continues on next page)

(continued from previous page)

```
          single-click and holding RB : Activation Rubber Band for region zooming
          single-click MB             : Activation 'Extend' Zoom mode
          scroll wheel MB             : Activation Zoom mode
          double-click LB, RB, MB     : Reset to Original View
  Magnifier mode:
          single-click LB             : Not Used
          single-click RB             : Not Used
          single-click MB             : Reset Magnifier to Original View
          scroll MB                   : Change Magnifier Zoom
          double-click LB             : Increase Magnifier Size
          double-click RB             : Decrease Magnifier Size

Hotkeys in 2D mode:

          'h'                         : Show help
          '+'                         : Zoom plus
          '-'                         : Zoom minus
          'd'                         : Toggle the detail level of the annotations
          'a'                         : Toggle the annotations graph mode
          '0'                         : Set default axes (reset to original view)
          'c'                         : On/Off pointer in crosshair mode
          'g'                         : Change lead gain with scroll
          'o'                         : Change lead offset with scroll
          'x'                         : Zoom and drag works only for X axis
          'y'                         : Zoom and drag works only for Y axis
          'm'                         : If pressed and holding, Magnifier mode on
          'p'                         : On/Off paper mode
          'r'                         : Format of the exported file (PDF/PNG)
          's'                         : Export current view
```

As you can see, the basic help is displayed, and this figure is shown as a result:



See Also

Plot ECG mosaic | *ECGwrapper*

2.6.1.2 Plotting signal mosaics

Low level function to produce mosaic charts of signals and annotations. This function plots several subplots in the same figure in order to do a mosaic with the different leads available in ECG. Annotations can be provided individually or for all the mosaics.

Prototype

The function prototype is the following.

```
[ ECG_hdl axes_hdl fig_hdl all_yranges ] = plot_ecg_mosaic( ECG, varargin )
```

This function accepts as arguments the arg_name/arg_val method described below.

Arguments

- ECG: [numeric or cell] REQUIRED
  - [numeric]: signal matrix of dimension `[sig_length sig_size repetitions_size]` where:
    - sig_length: time length in samples
    - sig_size: number of ECG leads or number of signals.
    - repetitions_size: number of repetitions of the same signals. Typically used when time-synchronized events, like heartbeats.
  - [cell]: cell array of length repetitions_size, where each cell is (probably a time alligned event) a signal of dimension `[sig_length sig_size]`
- QRS_locations: [numeric] OPTIONAL. Synchronization sample. In ECG context, this values are the QRS fiducial point. `[] (default)`
- WinSize: [numeric] OPTIONAL. Width of the window around each fiducial point provided in QRS_locations. `[] (default)`
- ECG_header: [struct] OPTIONAL. Description of the ECG typically available in the ECG_header as described *here*.
- MaxECGrange: [numeric or string] OPTIONAL. Force a vertial range in order to ease visual comparison of signals in the mosaic.

  [string]

  'max': force the maximum range to be the range for all mosaics.

  'min', 'mean', 'median': are also available options.

  'none': Each mosaic with a different range. (Default).
- RowsCols: [numeric] OPTIONAL. Number of rows and columns of the mosaic. If ommited or if `rows * cols ~= ECG_header.nsig`, these values are automatically adapted to the best fit mosaic in relation to the aspect ratio of the screen.
- FigureHdl: [figure handle] OPTIONAL. Choose the figure to be produced the mosaic. `gcf (default)`

- ECG_delineation: [struct] OPTIONAL. Annotation struct described *here*.
- ECG_annotations: [cell] OPTIONAL. Annotations to be included in the mosaic. The function accepts 2 type of annotations: points and lines. An example below shows how to define both annotations.

Output:

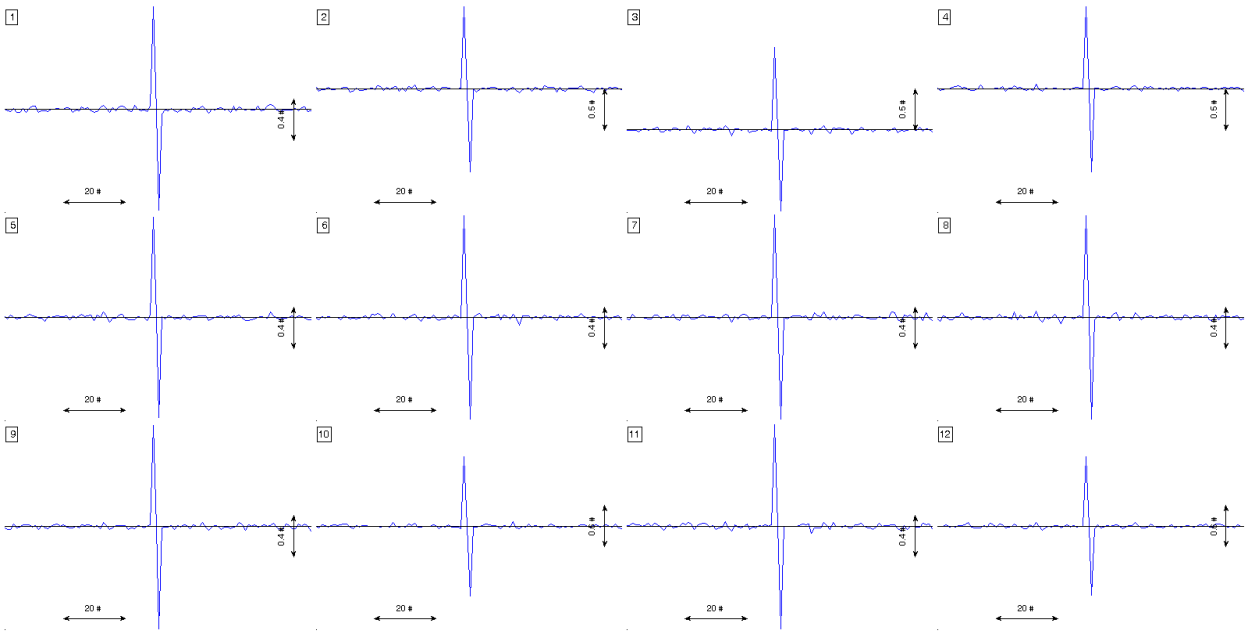
- ECG_hdl: handle to the plotted signals.
- axes_hdl: handle to the axes.
- fig_hdl: handle to fig.
- all_yranges: vertical ranges of the plotted signals.

Example

The following example shows some typical use of the function

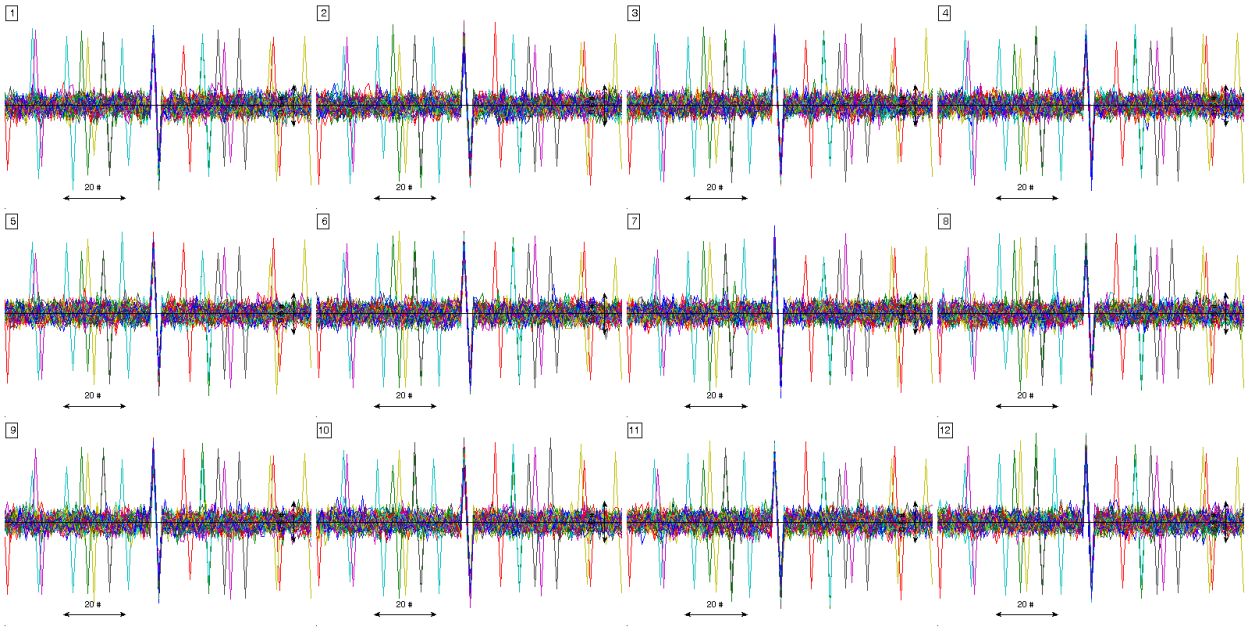
```
win_size = 100;
sig_samp = 10000;
sig_size = 12;
event_size = 50;
x = 0.1*randn(sig_samp,sig_size);
event_locations = randsample(win_size:sig_samp-win_size, event_size);
x(event_locations-1,:) = x(event_locations-1,:) + 1;
x(event_locations+1,:) = x(event_locations+1,:) - 1;
x_packed = pack_signal(x, event_locations, win_size);

figure(1)
% estimation of the signal averaged event
plot_ecg_mosaic( mean(x_packed,3) );
```

Here „„



```
figure(2)
% visualization of all events. In this case previous pack_signal call is
% not needed.
plot_ecg_mosaic(x, 'QRS_locations', event_locations, 'WinSize', win_size);
```

Here „„



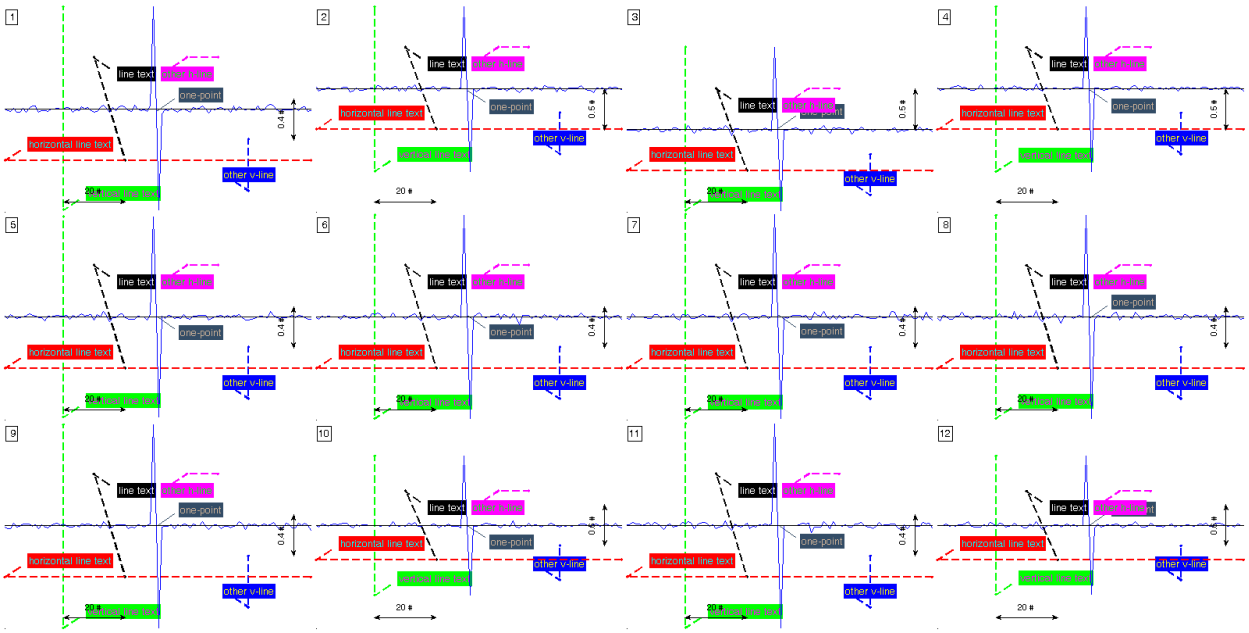
```
figure(3)
% introducing several kind of marks to the plot

h_line = cell(sig_size,7);
h2_line = cell(sig_size,7);
v_line = cell(sig_size,7);
v2_line = cell(sig_size,7);
point = cell(sig_size,7);
a_line = cell(sig_size,7);

h_line(:,1) = {'line'};
h_line(:,2) = { [ { 'String'                    'LineStyle' 'LineWidth' 'Color'
↪'TextColor' }; ...
                                { 'horizontal line text'       '--'         1.5
↪    'r'     'r'         } ]'};
h_line(1:sig_size, [6 7]) = num2cell( repmat(-0.5,sig_size,2) );

h2_line(:,1) = {'line'};
h2_line(:,2) = { [ { 'String'           'LineStyle' 'LineWidth' 'Color' 'TextColor' };
↪ ...
                                { 'other h-line'      '--'         1.5          'm'
↪    'm'         } ]'};
h2_line(1:sig_size, 4:7) = num2cell( [ repmat(60,sig_size,1) repmat(70,sig_size,1)
↪repmat(0.5,sig_size,2) ] );

v_line(:,1) = {'line'};
v_line(:,2) = { [ { 'String'             'LineStyle' 'LineWidth' 'Color' 'TextColor' }
↪; ...
```

(continues on next page)

(continued from previous page)

```
                                    { 'vertical line text' '--'          1.5          'g'␣
↪     'g'          } ]'};
v_line(1:sig_size, [4 5]) = num2cell( repmat(20,sig_size,2) );

v2_line(:,1) = {'line'};
v2_line(:,2) = { [ { 'String'       'LineStyle' 'LineWidth' 'Color' 'TextColor' }; ...
                                    { 'other v-line' '--'          1.5          'b'      'b
↪'          } ]'};
v2_line(1:sig_size, 4:7) = num2cell( [ repmat(80,sig_size,2) repmat(-0.8,sig_size,1)␣
↪repmat(-0.3,sig_size,1) ] );

point(:,1) = {'point'};
point(:,2) = { [ { 'String'     'Color'        'TextColor'    }; ...
                                  { 'one-point' [0.2 0.3 0.4] [0.2 0.3 0.4] } ]'};
point(1:sig_size,4) = num2cell( repmat(50,sig_size,1) );

a_line(:,1) = {'line'};
a_line(:,2) = { [ { 'String'     'LineStyle' 'LineWidth' 'Color' 'TextColor' }; ...
                                   { 'line text' '--'          1.5          'k'      'k' ␣
↪       } ]'};
a_line(1:sig_size,4:7) = num2cell( [ repmat(30,sig_size,1) repmat(40,sig_size,1)␣
↪repmat(0.5,sig_size,1) repmat(-0.5,sig_size,1) ] );

aux_anns = cat(3,h_line,v_line,h2_line,v2_line,point,a_line);

plot_ecg_mosaic(mean(x_packed,3), 'ECG_annotations', aux_anns );
```

Here „„



See Also

Plot ECG strip

2.6.2 Syntax

The function prototype is

```
function reportECG(ECG_w, detailLevel, report_mode, win_lengths, report_format,␣
↪filename)
```

where the arguments are:

- `ECG_w` An ECGwrapper object as the signal handler.
- `detailLevel` The report detail level:
  - 'HighDetail'
  - 'MediumDetail'
  - 'LowDetail'

  A higher detail level means report the whole recording at every time resolution defined in "win_lengths". High resolution also means larger reports. `LowDetail (default).`
- `report_mode` Information from other tasks like QRS detection/delineation/classification added to the signals in case available mode. Possible values are:

  'full'

  `ECG only (default)`

  'QRS detection'

  'Wave delineation'

  'Heartbeat classification'
- `win_lengths` The amount and size (in seconds) of each scale length present in the report. `[60*60 30*60 60 7] (default).` It means 1 hour - 30 min - 1 min and 7 seconds.
- `report_format` The report format of the document. `PDF (default).`
- `filename` The report filename. `rec_folder\rec_name.report_format (default).`

2.6.3 Examples

The example folder has some examples of the use of the reporting functions.

```
reportECG(ECGw, 'LowDetail', 'full');
```

This is an example of an ECG overview

And this with more information overprint

2.6.4 See Also

Plot ECG strip | *Plot ECG mosaic*

2.7 Other functions

Several low-level functions that are located in `your_path\ecg-kit\common\` but are not yet well documented, tested or integrated with other parts of the kit.

2.7.1 General functions

- addpath_if_not_added - (Internal) Add the path only if not was already added.
- colvec - (Internal) Reshape the input into a column vector
- rowvec - (Internal) Reshape a matrix into a row vector
- init_ghostscript - (Internal) Init environment variables for using ghostscript
- init_WFDB_library - (Internal) Init environment variables for using WFDB toolbox
- isMatlab - (Internal) Check if the kit is running on Matlab
- isOctave - (Internal) Check if the kit is running on Octave

- exist_distributed_file - (Internal) Check the existence of a file in a distributed (slow) filesystem
- GetFunctionInvocation - (Internal) Create a string with the invocation of a function
- max_index - (Internal) Index of the maximum element in a vector
- modmax - (Internal) Find modulus maxima in a signal
- myzerocros - (Internal) Detect zero-crosses in a signal
- soft_intersect - (Internal) Intersection of two sets with tolerance
- soft_range_conversion - (Internal) Convert an input range to an output range with a soft function
- soft_set_difference - (Internal) Set difference with tolerance
- parse_pids - (Internal) Identify how many PIDs are in total and which is this PID, based on a string formatted this_pid/cant_pids
- getAnnNames - (Internal) Get names of annotations from annotation structure
- matrix2positions - (Internal) Convert matrix of ECG wave annotations to a struct position format, used in wavedet algorithm
- positions2matrix - (Internal) Convert matrix of ECG wave annotations to a struct position format, used in wavedet algorithm
- pack_signal - (Internal) Example of user-created QRS detector
- *progress_bar* - (Internal) A progress bar class for showing evolution of a process to users
- progress_bar_ex - A progress bar class example
- TaskPartition - (Internal) Generate a PIDs work list
- trim_ECG_header - (Internal) Trim a header info struct to a subset of signals
- WFDB_command_prefix - System commands to initialize the WFDB toolbox
- HasAdminPrivs - Checks administrator privileges
- sys_cmd_separation_string - String to issue multiline system commands
- sys_command_strings - Strings to execute typical I/O commands via system calls

2.7.2 Strings related

- adjust_string - (Internal) Works with strings to center, trim and justify to a certain string width
- calc_btime - (Internal) Creates a string with the base time
- disp_string_framed - (Internal) Display a message framed to a string
- disp_string_title - (Internal) Display a message framed to a string
- disp_option_enumeration - (Internal) Display an enumeration of options to a string
- DisplayConfusionMatrix - (Internal) Pretty display in Screen the confusion matrix.
- DisplayResults - (Internal) Pretty-Display results of a classification experiment
- Seconds2HMS - (Internal) Create a string of hours mins and seconds based on data in seconds

2.7.3 Graphics related

- arrow - (Internal) Creates arrows in grapics
- ds2nfu - Convert data space units into normalized figure units.
- plot_auc - (Internal) Plot the area under the ROC curve
- plot_ecg_heartbeat - (Internal) obsolete, use plot_ecg_strip
- plot_ecg_mosaic - Plots multidimentional signal in mosaic style
- plot_ecg_strip - Plots and interact with ECG signal
- plot_roc - (Internal) Plot the ROC curve
- PlotGlobalWaveMarks - (Internal) Internal function of plot_ecg_strip
- PlotWaveMarks - (Internal) Internal function of plot_ecg_strip
- my_colormap - (Internal) Create a colormap for signal visualization
- maximize - Size a window to fill the entire screen.
- rand_linespec - (Internal) Example of user-created QRS detector
- rotateticklabel - rotates tick labels
- set_a_linespec - (Internal) Set a series of properties to a line handle
- set_rand_linespec - (Internal) Set random properties a line handle
- text_arrow - (Internal) Plot an arrow with text in a graphic
- text_line - (Internal) Plot a line with text in a graphic

2.7.4 Signal processing / statistical methods

- autovec_calculation - (Internal) Calculate eigenvalues and vectors
- autovec_calculation_robust - (Internal) Calculate eigenvalues and vectors using robust covariance estimation
- BaselineWanderRemovalMedian - Remove baseline wandering with the median estimation method
- BaselineWanderRemovalSplines - Remove baseline wandering with the median estimation method
- bxb - (Internal) Compares two heartbeat series and produce the confusion matrix as result
- calc_co_ocurrences - (Internal) Calculate the heartbeats co-ocurrences
- calc_correlation_gain - (Internal) Add the path only if not was already added.
- CalcRRserieQuality - (Obsolete) Estimate the quality of QRS complex detections
- CalcRRserieRatio - Estimate the quality of QRS complex detections
- calculateSeriesQuality - Estimate the quality of QRS complex detections
- MedianFilt - (Internal) Mean/Median filtering
- cluster_data_with_EM_clust - (Internal) Cluster data with expectation-maximization algorithm
- DelayedCovMat - (Internal) Delayed covariance matrix calculation
- deNaN_dataset - (Internal) Replace NaN from PRdatasets
- design_downsample_filter - (Internal) Design a filter to downsample signals prior printing to a report

- bandpass_filter_design - MATLAB Code
- PeakDetection2 - peaks = PeakDetection2(x,fs,wlen,fp1,fp2,th,flag),
- PiCA - [y,W,A] = PiCA(x,peaks1,peaks2)
- PrctileFilt - (Internal) Arbitrary percentile filtering
- logit_function - (Internal) Logit function
- my_ppval - PPVAL Evaluate piecewise polynomial.
- qs_filter_design - (Internal) Design the wavelet decomposition filters for wavedet algorithm
- qs_wt - (Internal) Calculates the wavelet transform
- nanmeda - (Internal) Calculate the median of absolute deviations from the median (MEDA)
- woody_method - (Internal) Woody algorithm for heartbeat allignment
- AUC_calc - Compute area under the ROC curve (AUC).
- ppval - Evaluate piecewise polynomial.
- similarity_calculation - (Internal) Pattern matching function to be used in an arbitrary task
- combine_anns - (Internal) Create new QRS detections based on other lead/algorithms detections

2.7.5 Tasks

- ECGtask - Defines the class interface for the ECGtask derived classes
- ECGtask_do_nothing - Null ECGtask (for Matlab)
- ECGtask_ECG_delineation - ECGtask for ECGwrapper (for Matlab)
- ECGtask_ECG_delineation_corrector - ECGtask for ECGwrapper (for Matlab)
- ECGtask_heartbeat_classifier - ECGtask for ECGwrapper (for Matlab)
- ECGtask_PCA_proj_basis - ECGtask for ECGwrapper (for Matlab)
- ECGtask_PPG_ABP_corrector - ECGtask for ECGwrapper (for Matlab)
- ECGtask_PPG_ABP_detector - ECGtask for ECGwrapper (for Matlab)
- ECGtask_QRS_corrector - ECGtask for ECGwrapper (for Matlab)
- ECGtask_QRS_detection - ECGtask for ECGwrapper (for Matlab)
- ECGtask_QRS_detections_post_process - ECGtask for ECGwrapper (for Matlab)
- ECGtask_Delineation_corrector - ECGtask for ECGwrapper (for Matlab)
- ECGtask_arbitrary_function - ECGtask for ECGwrapper (for Matlab)
- ECGtask_classification_features_calc - ECGtask for ECGwrapper (for Matlab)
- example_worst_ever_ECG_delineator - (Internal) Example of user-created QRS detector
- example_worst_ever_QRS_detector - (Internal) Example of user-created QRS detector
- reportECG - (Internal) function reads the header of signal files
- QRScorrector - (Internal) GUI for correcting QRS detections
- GetBestQRSdetections - Fetch the best QRS detections from an ECGtask_QRSdetections object
- list_all_ECGtask - List al ECGtask availables

2.7.6 Functions from other projects

- GTHTMLtable - GTHTMLtable - Generate an HTML page with a table of a matrix.
- cprintf - displays styled formatted text in the Command Window

2.7.7 I/O signals

- Annotation_process - Convert heartbeat type of annotation from valid ECG formats to EC57 AAMI
- AnnotationFilterConvert - Convert heartbeat type of annotation from valid ECG formats to EC57 AAMI
- ADC2realunits - Convert adimentional sample values to real units
- ADC2units - Convert adimentional sample values to target voltage units
- ECGwrapper - Allow acces to ECG recordings of arbitrary format and length.
- ECGformat - Gets the format of an ECG recording filename
- get_ECG_idx_from_header - (Internal) Guess ECG signals indexes in a multimodal recording
- get_PPG_ABP_idx_from_header - (Internal) Guess PPG/ABP signals indexes in a multimodal recording
- isAHAformat - (Internal) Check if a recording is in ISHNE format.
- isHESformat - (Internal) Check if a recording is in HES format.
- isISHNEformat - (Internal) Check if a recording is in ISHNE format.
- matformat_definitions - (Internal) A definition or header file, for names allowed for signals, header and annotations included in MAT format files
- read_310_format - (Internal) Read the MIT 310 format
- read_311_format - (Internal) Read the MIT 311 format
- read_AHA_ann - Reads ECG annotations in AHA format
- read_AHA_format - Reads ECG recording in AHA format
- read_AHA_header - Reads ECG header in AHA format
- read_ECG - Reads an ECG recording
- read_HES_ann - Reads ECG annotations in HES format
- read_HES_format - Reads ECG recording in HES format
- read_HES_header - Reads ECG header in HES format
- read_ishne - (Internal) Reads ECG recordings in Mortara format
- read_ishne_ann - Reads ECG annotations from ISHNE format
- read_ishne_header - Reads ECG header from ISHNE format
- read_Mortara - (Internal) Reads ECG recordings in Mortara format
- read_Mortara_header - Reads ECG header in Mortara format.
- read_Mortara_format - Reads ECG recordings in Mortara format.
- read_ishne_format - Reads ECG recording in ISHNE format
- readheader - (Internal) function reads the header of signal files
- tablas_y_constantes - (Internal) Constants for the HES format

- writeannot - (Internal) Write annotation files for biomedical signals in MIT Format. (MEX file)
- writeheader - (Internal) Write an ECG header in MIT format
- ConcatenateQRSdetectionPayloads - (Internal) Concatenate two payloads.
- default_concatenate_function - Description:
- default_finish_function - Description:

2.7.7.1 Progress-bar class

This class allows the display of a progress bar to show evolution of lengthy processing loops. Will be documented soon, see examples in the project or in the code itself:

`your_path\ecg-kit\common\progress_bar.m`

2.7.7.2 list_all_ECGtask function

This function display all the installed ECGtask classes. Will be documented soon, see examples in the project or in the code itself:

`your_path\ecg-kit\common\list_all_ECGtask.m`

2.8 ECGkit extensions

This document describes how to create tasks and expand the kit capabilities.

2.8.1 Adding new recording formats

Will be written soon.

2.8.2 Adding new tasks

Arbitrary tasks may help you ?

2.8.3 Tasks that work over the signal

Will be written soon.

2.8.4 Tasks that work beat-by-beat

Will be written soon.