Advanced Human-System Interfaces

Second assignment

ANASS NASSIRI

AHSI - Lab 02.2 - Physiological signals analysis pipeline

This lecture aims to define a physiological signal data processing pipeline consisting in: signal denoising, signal normalization and signal segmentation in single tasks.

On these latter, a proper set of features is going to be extracted according to the relative physiological signal. Finally a classification task will be performed to discriminate between high arousal task and low arousal task.

In this lecture you will learn:

- how to denoising, normalize and segmented PPG and GSR physiological signals
- how to extract simple features from the two type of analyzed signals.
- how to use the features extracted to discriminate between two levels of arousal (high and low).

Dataset description

The data used in the following analysis are a subset of the dataset CLAWDAS, described in *"GASPARINI, Francesca, et al. Personalized PPG Normalization Based on Subject Heartbeat in Resting State Condition. Signals, 2022, 3.2: 249-265."*

Two types of physiological signals are collected from a population of 20 elderly subjects: the heartbeat collected through Photopletismography (PPG) and the Galvanic Skin Response (GSR).

The part of the experimental protocol here analyzed consist in a 15 minutes sequence in which the partecipants had to perform six repetition of two different tasks: 2 min of relaxing Audio Listening (AL) followed by 30 seconds of cognitive load induced by mental Math Calculations (MC).

For the purpose of this analysis, we consider AL as low arousal task while MC as high arousal task.

Data reading and preprocessing

In the first part of the pipeline, the raw physiological signals should be properly processed to reduce noise, normalize signals to disentangle subjects dependencies, and segmented into different tasks.

[QUESTION] Why do we apply normalization and denoising before segmented the signal in single the task?

Data loading

```
% [TO DO] Load the short version of the CLAWDAS dataset
load('FEB_CLData_AUCL_AUDIO.mat')
disp("starting")
```

```
starting
```

Data preprocessing

The signals of each subject must be individually preprocessed. For this reason, a *for-loop* control flow statement is used to run the preprocessing procedure (denoising, normalization and segmentation) on the signals collected from each subject separately.

This code perform this 3 steps

1) Signal Denoising

2) Signal amplitude Normalization .

3) Signal segmentation in tasks.

```
Fs = 128; % sampling frequency of 128 Hz
function GSRfilt = filterSegment_GSR(GSRraw)
    % Sampling frequency
    Fs = 128;

    % Low-pass filter design (cut-off frequency 0.5 Hz)
    [b, a] = butter(2, 0.5 / (Fs / 2), 'low');

    % Apply the filter
    GSRfilt = filtfilt(b, a, GSRraw);
end
function PPGfilt = filterSegment_PPG(PPGraw)
    % Sampling frequency
    Fs = 128;

    % Low-pass filter design (cut-off frequency 3 Hz)
    [b, a] = butter(2, 3 / (Fs / 2), 'low');

    % Apply the filter
    PPGfilt = filtfilt(b, a, PPGraw);
end
```

```
% Low-pass filter design for GSR (cut-off frequency 0.5 Hz)
[b_gsr, a_gsr] = butter(2, 0.5 / (Fs / 2), 'low');

% Low-pass filter design for PPG (cut-off frequency 3 Hz)
[b_ppg, a_ppg] = butter(2, 3 / (Fs / 2), 'low');
for sbj=1:length(CLData)
    disp(sbj)

    % [TO DO] Filter the analyzed signals in order to remove noise and
    % artifacts.

    CLData(sbj).GSRfilt = filterSegment_GSR(CLData(sbj).GSRraw);
    % [TO DO] apply a filter to the GSR data (CLData(sbj).GSRraw)

    CLData(sbj).PPGfilt = filterSegment_PPG(CLData(sbj).PPGraw);
    % [TO DO] apply a filter to the PPG data (CLData(sbj).PPGraw)
        % [TO DO] Apply Signal ampliude normalization to the analyzed signal
    % (CLData(sbj).GSRfilt)
    CLData(sbj).GSRnorm = zscore(CLData(sbj).GSRfilt);
    CLData(sbj).PPGnorm = zscore(CLData(sbj).PPGfilt);
        % Find the positions where the pulses occur
    MK=CLData(sbj).GSRPPGmarker;
    iAudioPeaks = find(MK == 17);

    % For each pulse, devide the two task according to what described
    % before
    for it = 1:length(iAudioPeaks)

        if it == 1 %For the first trial, the segment starts at the beginning of
signal
            CognLoad(sbj).GSR.AUDIO{it} =
CLData(sbj).GSRnorm(1:iAudioPeaks(it));
            CognLoad(sbj).PPG.AUDIO{it} =
CLData(sbj).PPGnorm(1:iAudioPeaks(it));
        else
            CognLoad(sbj).GSR.AUDIO{it} = CLData(sbj).GSRnorm(iAudioPeaks(it)-
4*30*128:iAudioPeaks(it));
            CognLoad(sbj).PPG.AUDIO{it} = CLData(sbj).PPGnorm(iAudioPeaks(it)-
4*30*128:iAudioPeaks(it));
        end

        if it == 6 % For the last trial (6th), the end of the segment match
with the end of the signal
            CognLoad(sbj).GSR.AUCL{it} =
CLData(sbj).GSRnorm(iAudioPeaks(it)+1:end);
```

```
            CognLoad(sbj).PPG.AUCL{it} =
CLData(sbj).PPGnorm(iAudioPeaks(it)+1:end);
        else
            CognLoad(sbj).GSR.AUCL{it} =
CLData(sbj).GSRnorm(iAudioPeaks(it)+1:iAudioPeaks(it)+30*128);
            CognLoad(sbj).PPG.AUCL{it} =
CLData(sbj).PPGnorm(iAudioPeaks(it)+1:iAudioPeaks(it)+30*128);
        end

    end

 end
```

```
    20
```

```
disp("finish")
```

```
 finish
```

1) Signal Denoising

Filter the two physiological signals (GSR and PPG) in order to remove noise and artifacts. Make use of low pass or band pass filters as seen in previous lessons.

```
% code is up
```

2) Signal amplitude Normalization

Physiological data are subjective and they depend on the person and the conditions under which they were acquired. In order to reduce this inter and intra subjects heterogeneity, an amplitude normalization is applied using z-score.

```
%code is up
```

3) Signal segmentation in tasks

The denoising and normalization procedures are applied to the whole signals collected from each subject. In the next step of the preprocessing pipeline, each signal is divided into different tasks as defined in the experimental protocol.

In the signal of markers of the proposed dataset, a pulse of value "17" is used to identify the beginning of each high Cognitive Load task (Math calculation). Considering that each task has a fixed length (2 minute for audio listening task and 30 second for math calculation task), the 2 minutes related samples before the pulse are labeled as Audio Listening task while 30 seconds after each pulse peak are defined as Math Calculation task.

The code to perform this segmentation is reported here in following.

```
% the code is up
```

And finally the end of the loop on subjects.

The whole procedure described so far generate a new matlab structure, called in this example CognLoad. In this structure each row is related to a subject while the fields represent the two analyzed physiological signals. In each cell, the data are grouped according to the task performed and the relative trial.

Features extraction

For each signal of the two analyzed tasks, a proper set of features have to be computed. The features extracted change according to the physiological signal type considered.

Let's try to extract a subset of features for PPG and GSR signal.

```
disp('%%% FEATURES EXTRACTION %%%')
```

```
%%% FEATURES EXTRACTION %%%
```

PPG features

Several features can be extracted from PPG signals both in time and frequency domain. In this example, you will learn how to extract some simple time domain features.

```
auCl_feat_PPG = [];
audio_feat_PPG = [];
auCl_sbjFeatIndex_PPG = [];
audio_sbjFeatIndex_PPG = [];
auCl_sbjFeatIndex_GSR = [];
audio_sbjFeatIndex_GSR = [];
Fs = 128;

for sbj= 1 : length(CognLoad)

    % Extract PPG features from the signals collected from the subject "sbj"
    % during the High Cognitive Load task
    aucl = CognLoad(sbj).PPG.AUCL;
    tabTmp = struct2table(computeFeaturesPPG_short(aucl, Fs)); %[TO DO]:
compute PPG features

    auCl_feat_PPG = vertcat(auCl_feat_PPG, tabTmp);
    auCl_sbjFeatIndex_PPG = [auCl_sbjFeatIndex_PPG, ones(1, size(tabTmp,
1))*sbj];
```

```
    % Extract PPG features from the signals collected from the subject "sbj"
    % during the Low Cognitive Load task
    relax = CognLoad(sbj).PPG.AUDIO;
    tabTmp = struct2table(computeFeaturesPPG_short(relax, Fs)); %[TO DO]:
compute PPG features

    audio_feat_PPG = vertcat(audio_feat_PPG, tabTmp);
    audio_sbjFeatIndex_PPG = [audio_sbjFeatIndex_PPG, ones(1, size(tabTmp,
1))*sbj];

 end
```

GSR features

Features from both phasic and tonic component of GSR signal can be computed starting from the analyzed signals.

```
 auCl_feat_GSR = [];
 audio_feat_GSR = [];

 auCl_sbjFeatIndex_GSR = [];
 audio_sbjFeatIndex_GSR = [];
 Fs = 128;

 for sbj= 1 : length(CognLoad)

    % Extract PPG features from the signals collected from the subject "sbj"
    % during the High Cognitive Load task
    aucl = CognLoad(sbj).GSR.AUCL;
    tabTmp = struct2table(computeFeaturesGSR_short(aucl, Fs)); %[TO DO]:
compute PPG features

    auCl_feat_GSR = vertcat(auCl_feat_GSR, tabTmp);
    auCl_sbjFeatIndex_GSR = [auCl_sbjFeatIndex_GSR, ones(1, size(tabTmp,
1))*sbj];


    % Extract PPG features from the signals collected from the subject "sbj"
    % during the Low Cognitive Load task
    relax = CognLoad(sbj).GSR.AUDIO;
    tabTmp = struct2table(computeFeaturesGSR_short(relax, Fs)); %[TO DO]:
compute PPG features
```

```
    audio_feat_GSR = vertcat(audio_feat_GSR, tabTmp);
    audio_sbjFeatIndex_GSR = [audio_sbjFeatIndex_GSR, ones(1, size(tabTmp,
1))*sbj];
 end
```

High and Low cognitive load tasks classification using Classification Learner

The features extracted from the physiological signals can be used to discriminate among two different levels of cognitive load: *low cognitive load* (or relax) representing by audio listening task (*AUDIO*), and *high cognitive load* representing by the Math Calculation task (*AUCL)*.

Classify the two different levels of cognitive load using Classification Learner. Select the proper validation strategy and the classifier (you can test several of them and chose the best one).

```
disp('%%% CLASSIFICATION')
```

```
%%% CLASSIFICATION
```

```
classificationLearner
```

High and Low cognitive load tasks classification using LOSO

The features extracted from the physiological signals can be used to discriminate among two different levels of cognitive load: *low cognitive load* (or relax) representing by audio listening task (*AUDIO*), and *high cognitive load* representing by the Math Calculation task (*AUCL)*.

In order to to generate more robust performances, a *leave one subject out (LOSO)* cross-validation strategy will be employed. In this evaluation methods the analyzed data are divided into subsets according to the subject from which the data were acquired. At each iteration, all the instances of one subject are used to test the model, while the signals of the remaining subjects are used as a training set.

In this part of the lecture, you will learn to define a cubic SVM classifier able to discriminate between two different cognitive load levels (High and Low) and evaluate its performances using a LOSO cross validation strategy.

Heare I use only ppg feature for classification

```
% disp('%%% CLASSIFICATION')
```

```
 %
 % nsbj = size(CognLoad, 2);
 % % Combine features for both tasks
 % feat_total_CL = auCl_feat_PPG;
 % CL_sbjFeatindex = auCl_sbjFeatIndex_PPG';
 %
 % feat_total_relax = audio_feat_PPG;
 % relax_sbjFeatindex = audio_sbjFeatIndex_PPG';
 %
 % % Define a single table with all the features (both high and low CL)
 % feat_total_session = ...
 % [feat_total_CL; ...
 % feat_total_relax];
 %
 % index_Sbj_features = [CL_sbjFeatindex; ...
 % relax_sbjFeatindex];
 %
 % % Definizione delle label
 % labels = repmat({'CognLoad'}, size(feat_total_CL, 1), 1);
 % labels = vertcat(labels, repmat({'relax'}, size(feat_total_relax, 1),1));
 %
 %
 % %% -- Classificazione -- %
 %
 % % Inizializzazione array
 % Y_Pred_total = [];
 % Y_Real_total = [];
 %
 %
 % % TO DO: Implement the LOSO (Leave One Subject Out) cross validation to
 % % evaluate the performance of a classifier (you can select the one you
prefer) that
 % % discriminate between low and high cognitive load.
 %
 % % Compute the evaluation metrics (precision, recall, accuracy, etc...) on the
single confusion matrix
 % % generated joining the confusion matrices resulting from each iteration.
 %
 % % Use the following guidelines as a basis for your code.
 %
 % for numIt = 1 : nsbj
 %     disp(numIt)
 %
 %     % [TO DO]...
 %         disp(['Iteration: ', num2str(numIt)])
 %
```

```
%     % Split data into training and testing sets
%     train_idx = index_Sbj_features ~= numIt;
%     test_idx = index_Sbj_features == numIt;
%
%     train_data = feat_total_session(train_idx, :);
%     test_data = feat_total_session(test_idx, :);
%     train_labels = labels(train_idx);
%     test_labels = labels(test_idx);
%
%     % Train a cubic SVM model
%     model = fitcsvm(train_data, train_labels, 'KernelFunction', 'polynomial',
'PolynomialOrder', 3, 'Standardize', true);
%
%     % Test the model
%     predictions = predict(model, test_data);
%
%     % Collect predictions and real labels
%     Y_Pred_total = [Y_Pred_total; predictions];
%     Y_Real_total = [Y_Real_total; test_labels];
%
% end
%
% % [TO DO] Evaluate performances on the final confusion matrix
%
% %[TO DO...]
% % Evaluate performances on the final confusion matrix
% confMat = confusionmat(Y_Real_total, Y_Pred_total);
% accuracy = sum(diag(confMat)) / sum(confMat(:));
% disp(['Overall Accuracy: ', num2str(accuracy)])
%
% % Compute other evaluation metrics
% precision = diag(confMat) ./ sum(confMat, 2);
% recall = diag(confMat) ./ sum(confMat, 1)';
% F1 = 2 * (precision .* recall) ./ (precision + recall);
%
% disp(['Precision: ', num2str(mean(precision))])
% disp(['Recall: ', num2str(mean(recall))])
% disp(['F1-score: ', num2str(mean(F1))])
% disp("************************ Assignment finished*******************")
```

Here I use both features combined PPG and GSR for Classification

```
disp('%%% CLASSIFICATION')
```

```
%%% CLASSIFICATION
```

```
 nsbj = length(CognLoad);

 % Combine PPG and GSR features for both tasks
 feat_total_CL_PPG = auCl_feat_PPG;
 feat_total_CL_GSR = auCl_feat_GSR;
 feat_total_CL = [feat_total_CL_PPG, feat_total_CL_GSR];
 CL_sbjFeatindex = auCl_sbjFeatIndex_PPG';  % if indexes are the same for GSR
and PPG

 feat_total_relax_PPG = audio_feat_PPG;
 feat_total_relax_GSR = audio_feat_GSR;
 feat_total_relax = [feat_total_relax_PPG, feat_total_relax_GSR];
 relax_sbjFeatindex = audio_sbjFeatIndex_PPG';  % if indexes are the same for
GSR and PPG

 % Defining a single table with all the features (both high and low CL)
 feat_total_session = [feat_total_CL; feat_total_relax];
 index_Sbj_features = [CL_sbjFeatindex; relax_sbjFeatindex];
 labels = [repmat({'CognLoad'}, size(feat_total_CL, 1), 1); repmat({'relax'},
size(feat_total_relax, 1), 1)];

 %% -- Classificazione -- %

 % Initializing arrays for predictions and real labels
 Y_Pred_total = [];
 Y_Real_total = [];

 % Implementing the LOSO (Leave One Subject Out) cross-validation to evaluate
the performance of a classifier
 % that discriminates between low and high cognitive load.

 for numIt = 1:nsbj
     disp(['Iteration: ', num2str(numIt)])

     % Split data into training and testing sets
     train_idx = index_Sbj_features ~= numIt;
     test_idx = index_Sbj_features == numIt;

     train_data = feat_total_session(train_idx, :);
     test_data = feat_total_session(test_idx, :);
     train_labels = labels(train_idx);
     test_labels = labels(test_idx);

     % Train a cubic SVM model
```

```
    model = fitcsvm(train_data, train_labels, 'KernelFunction', 'polynomial',
'PolynomialOrder', 3, 'Standardize', true);

    % Test the model
    predictions = predict(model, test_data);

    % Collect predictions and real labels
    Y_Pred_total = [Y_Pred_total; predictions];
    Y_Real_total = [Y_Real_total; test_labels];
end
```

```
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Iteration: 10
Iteration: 11
Iteration: 12
Iteration: 13
Iteration: 14
Iteration: 15
Iteration: 16
Iteration: 17
Iteration: 18
Iteration: 19
Iteration: 20
```

```
% Evaluate performances on the final confusion matrix
confMat = confusionmat(Y_Real_total, Y_Pred_total);
accuracy = sum(diag(confMat)) / sum(confMat(:));
disp(['Overall Accuracy: ', num2str(accuracy)])
```

```
Overall Accuracy: 0.9
```

```
% Compute other evaluation metrics
precision = diag(confMat) ./ sum(confMat, 2);
recall = diag(confMat) ./ sum(confMat, 1)';
F1 = 2 * (precision .* recall) ./ (precision + recall);

disp(['Precision: ', num2str(mean(precision))])
```

```
Precision: 0.9
```

```
disp(['Recall: ', num2str(mean(recall))])
```

```
Recall: 0.9
```

```
disp(['F1-score: ', num2str(mean(F1))])
```

```
F1-score: 0.9
```

```
disp("************************* Assignment finished*******************")
```

```
************************ Assignment finished******************
```

After I finish from , I export the best model .

I saved to use in third assignment .

```
% Code to save to model to use it in Third Assignment
% save('bestModel_final.mat', 'trainedModel_final');
```

Compute Features from PPG

Different features can be computed starting from PPG signal. The following function will allow you to extract:

- statistical features from PPG (max, min, mean and variance)
- simple Peak features from PPG (peakRate, IBI)

INPUT:

- struct = matlab structure with the physiological signals from which extract features
- Fc = Sampling Frequency

OUTPUT:

- out = table of extracted features.

```
function [out] = computeFeaturesPPG_short(struct, Fc)

    measures_max_ppg = [];
    measures_min_ppg = [];
    measures_mean_ppg = [];
    measures_var_ppg = [];

    measures_rate_pks_ppg = [];

    measures_IBI_mean = [];
    measures_SDNN = [];

    % For each signal in the input structure "stuct"
    for j = 1 : length(struct)

        % Estrazione segmento da analizzare
        temp_ppg = struct{1,j};
```

1) Compute statistical features

```
        % [TO DO]: Evaluate the four statistical features Maximum, Minimum,
        % Mean and Variance from the signal "temp_ppg" and add them to
        % their respective structures (measures_max_ppg, measures_min_ppg,
        % etc...)
        max_ppg = max(temp_ppg);
        measures_max_ppg = [measures_max_ppg; max_ppg];
```

```
        min_ppg = min(temp_ppg);
        measures_min_ppg = [measures_min_ppg; min_ppg];

        mean_ppg = mean(temp_ppg);
        measures_mean_ppg = [measures_mean_ppg; mean_ppg];

        var_ppg = var(temp_ppg);
        measures_var_ppg = [measures_var_ppg; var_ppg];
```

2) Compute Peak Related Features

One of the most used features extracted from PPG is the *Peak Rate* representing the mean number of peaks per second. To evaluate this features, it is necessary finding first the systolic peaks in PPG signal. Keep in mind that not all the peaks in signal are systolic peaks. In particular, two systolic peaks can never be less than 300 milliseconds apart due to the physical characteristics of the heartbeat.

The matlab function "findpeaks" allow to find the local maxima (peaks) of the input signal vector.

```
        % [TO DO] Use the findPeak function and the min distances between two
peaks to
        % compute the PeakRate feature for each of the PPG signal
        % (temp_ppg). Then, add it to the structure "measures_rate_pks_ppg"
                % Use the findPeak function and the min distances between two
peaks to
        % compute the PeakRate feature for each of the PPG signal (temp_ppg).

        [pks, locs] = findpeaks(temp_ppg, 'MinPeakDistance', round(0.3 * Fc));
% 300 ms min distance
        peakRate = length(pks) / (length(temp_ppg) / Fc); % Peaks per second
        measures_rate_pks_ppg = [measures_rate_pks_ppg; peakRate];
```

3) Inter Beat Interval (IBI) or beat-to-beat intervals parameters

Some parametrers usually extracted from PPG are the ones related to *Heart Rate Variability* (HRV), which is the variation in the time interval between heartbeats. The heart rate does not remain constant but varies over time, thus HRV parameters allow to analyze this variability.

To calculate HRV, the beat-to-beat intervals (or Inter Beat Intervals) must be extracted. In particular, the distances between each couple of consecutive peaks need to be evaluated, thus computing the beat-to-beat intervals (or IBI).

Starting from the IBI vector, a proper set of time domain features can be extracted such as the mean distances between consecutive peaks (IBI_mean) or the standard deviation of intervals (SDNN).

```
        % [TO DO] Starting from the peaks detected, evaluate the Inter Beat
Intervals (IBIs)
        % as the distances between consecutive peaks. Then, using IBI,
        % compute the two time domain features IBI_mean and SDNN.
        % ...
        IBI=[];

        if length(locs) > 1
            IBI = diff(locs) / Fc; % Convert sample differences to seconds
            IBI_mean = mean(IBI);
            measures_IBI_mean = [measures_IBI_mean; IBI_mean];

            SDNN = std(IBI);
            measures_SDNN = [measures_SDNN; SDNN];
        else
            % If there are no peaks detected, set features to NaN
            measures_IBI_mean = [measures_IBI_mean; NaN];
            measures_SDNN = [measures_SDNN; NaN];
        end

    end
```

Function output definition

```
    % Costruzione dell'output
    out.max_ppg = measures_max_ppg;
    out.min_ppg = measures_min_ppg;
    out.mean_ppg = measures_mean_ppg;
    out.var_ppg = measures_var_ppg;

    out.rate_peaks_ppg = measures_rate_pks_ppg;

    out.IBI_mean = measures_IBI_mean;
    out.SDNN = measures_SDNN;
```

```
end
```

Compute Features from GSR (or EDA)

Different features can be computed starting from GSR signal. The following function will allow you to extract:

- statistical features from GSR (max, min, mean and variance)
- statistical features from phaisc component of GSR (max, min, mean and variance)
- simple Peak features from phasic component of GSR (peakRate)
- the Regression Coefficient from GSR tonic component

INPUT:

- struct = matlab structure with the physiological signals from which extract features
- Fc = Sampling Frequency

OUTPUT:

- out = table of extracted features.

.

```
function [out] = computeFeaturesGSR_short(struct, Fc)

    % Final structures inizialization
    measures_max_gsr = [];
    measures_min_gsr = [];
    measures_mean_gsr = [];
    measures_var_gsr = [];

    measures_max_gsr_phas = [];
    measures_min_gsr_phas = [];
    measures_mean_gsr_phas = [];
    measures_var_gsr_phas = [];

    measures_n_pks_gsr = [];
    measures_rate_pks_gsr = [];

    measures_reg_coef_gsr = [];


    % For each signal in the input structure "stuct"
    for j = 1 : length(struct)

        temp_gsr = struct{1,j};
```

1) GSR signal decomposition in Phasic and Tonic

The GSR signal could be decomposed into three parts:

- the phasic component that represents rapid changes in skin conduction (skin conductance responses) due to external stimuli or spontaneous responses,
- the tonic component related to the slow change in the signal and representative of the general arousal level and
- the additive white Gaussian noise term incorporating errors and artifacts.

```
        % Modo 1: usato in "Detecting Moments of Stress from Measurements of
        %Wearable Physiological Sensors
        f = 0.05; %Hz
        [ba, aa] = butter(1, f/Fc, 'high');
        phasic_temp = filtfilt(ba, aa, temp_gsr);

        phas_min = min(phasic_temp);
        phasic_temp = phasic_temp - phas_min;

        [ba, aa] = butter(1, f/Fc, 'low');
        tonic_temp =  filtfilt(ba, aa, temp_gsr);


        % Modo 2: usando cxvEDA (cvxEDA: A Convex Optimization Approach to
        % Electrodermal Activity Processing]
        %[phasic_temp, ~, tonic_temp, ~, ~, ~, ~] = ...
        %    cvxEDA(temp_gsr, 1/Fc);
        %phasic_temp = phasic_temp';
        %tonic_temp = tonic_temp';
```

Visualization of the two components:

```
        % Note: Comment this part when you have to extract features from all
the signals.

        % figure('units','normalized','outerposition',[0 0 0.9 0.9]);
sgtitle([testo ' - Trial ' num2str(j)])
```

```
        % subplot(2,2,1); plot(phasic_temp); title('Phasic Component');
        % hold on; plot(temp_gsr); legend('phasic', 'original');
        % subplot(2,2,2); plot(tonic_temp); title('Tonic Component');
        % hold on; plot(temp_gsr); legend('tonic', 'original');
```

1) Compute statistical features from full signal

```
        % [TO DO]: Evaluate the four statistical features Maximum, Minimum,
        % Mean and Variance from the signal "temp_gsr" and add them to
        % their respective structures (measures_max_gsr, measures_min_gsr,
        % etc...)
        max_gsr = max(temp_gsr);
        measures_max_gsr = [measures_max_gsr; max_gsr];

        min_gsr = min(temp_gsr);
        measures_min_gsr = [measures_min_gsr; min_gsr];

        mean_gsr = mean(temp_gsr);
        measures_mean_gsr = [measures_mean_gsr; mean_gsr];

        var_gsr = var(temp_gsr);
        measures_var_gsr = [measures_var_gsr; var_gsr];
```

2) Compute statistical features from phasic component

```
        % [TO DO]: Evaluate the four statistical features Maximum, Minimum,
        % Mean and Variance from the phasic component of "temp_ppg" signal and
add them to
        % their respective structures (measures_max_gsr_phas,
measures_min_gsr_phas,
        % etc...)
        max_gsr_phas = max(phasic_temp);
        measures_max_gsr_phas = [measures_max_gsr_phas; max_gsr_phas];

        min_gsr_phas = min(phasic_temp);
        measures_min_gsr_phas = [measures_min_gsr_phas; min_gsr_phas];

        mean_gsr_phas = mean(phasic_temp);
        measures_mean_gsr_phas = [measures_mean_gsr_phas; mean_gsr_phas];

        var_gsr_phas = var(phasic_temp);
        measures_var_gsr_phas = [measures_var_gsr_phas; var_gsr_phas];
```

3) Compute Peak Related Features

From the phasic component of GSR it is possible to evaluate the *Peak Rate* representing the mean number of peaks per second. To evaluate this features, it is necessary finding first the peaks in phasic component of GSR signal. Keep in mind that not all the peak in the signal represent activation but some of them are due to noise or artifacts. In order to extract only correct peaks, consider a min distances between two peaks of 128 samples and a min peak prominence of 0.02.

The matlab function "findpeaks" allow to find the local maxima (peaks) of the input signal vector.

```
        % [TO DO] Use the findPeak function, the min distances between two
peaks and the min peak prominance to
        % compute the PeakRate feature for each of the GSR signal
        % (phasic_temp). Then, add it to the structure "measures_rate_pks_gsr"
        [pks, locs] = findpeaks(phasic_temp, 'MinPeakDistance', 128,
'MinPeakProminence', 0.02); % Example parameters
        peakRate = length(pks) / (length(phasic_temp) / Fc); % Peaks per second
        measures_rate_pks_gsr = [measures_rate_pks_gsr; peakRate];
```

4) Extract features from tonic components

Finally, we will extract the Regression coefficient of tonic component as signal slope representative.

```
        % Evaluate Regression Coefficient
        y = tonic_temp;
        x = (1:length(y))';
        p = polyfit(x,y,1);

        measures_reg_coef_gsr = [measures_reg_coef_gsr; p(1)];

    end
```

Function output definition

```
    out.max_gsr = measures_max_gsr;
    out.min_gsr = measures_min_gsr;
    out.mean_gsr = measures_mean_gsr;
    out.var_gsr = measures_var_gsr;

    out.max_gsr_phas = measures_max_gsr_phas;
    out.min_gsr_phas = measures_min_gsr_phas;
    out.mean_gsr_phas = measures_mean_gsr_phas;
    out.var_gsr_phas = measures_var_gsr_phas;
```

```
    out.rate_peaks_gsr = measures_rate_pks_gsr;

    out.reg_coef_gsr = measures_reg_coef_gsr;

end
```