Francisco Javier Sáez Maldonado
franciscojavier.saez@estudiante.uam.es

Online Signature Lab Report

Deep Learning for Biometric Signal Processing

2022-03-17

Contents

Introduction

The objective of this session is to DEVELOP and EVALUATE an online signature recognition algorithm. According to the theory sessions, signature recognition systems can be divided into two categories:

- Off-line: the input is a static image of the signature.
- On-line: the signature is acquired using a specific digital sensor which includes the static image and dynamic signals related with the way the signature was done: x,y coordinates and pressure as a function of time.

Figure 1 shows a block diagram of a typical online signature recognition algorithm where [x,y,p] are the captured signals by the sensor (Cartesian coordinates and pressure), ft is the feature vector of the query signature to be compared with the fc feature vector of the signature stored in the database (claimed identity).
In this session we will assume that the data is available (previously acquired) and we will focus on the development of two modules:

- Feature Extraction Module.
- Matcher.

You must complete the tasks proposed in this document and answer the questions included.

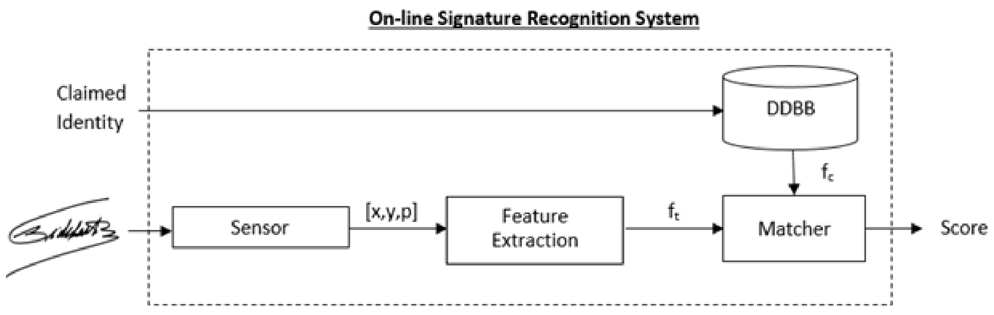


Figure 1: Block Diagram of a typical online signature recognition system

Data

For the practice we will use 50 users from the BiosecurID database. Each of the users have 28 signatures acquired in 4 sessions with a time lapse of 2 months. From the 28 signatures, 16 are genuine (4 per session) and 12 are forgers (3 per session). In this practice we will only consider the genuine signatures.
Each of the signatures is stored in .mat file which contains three vectors of same length with the x, y coordinates and the pressure as functions of time. The formatting of the files is `uXXXXsYYYY_sgZZZZ.mat`:

- XXXX: user number
- YYYY: session number
- ZZZZ: signature number

The GENUINE signatures of each session are those with $ZZZZ = [0001, 0002, 0006, 0007]$.The signatures with $ZZZZ = [0003, 0004, 0005]$ are the FORGERS and they will NOT be used in this practice.

Drawing signals

Choose a signature (from a random user) and show (assuming that the sensor has a 200 samples/second acquisition rate):

- Signal x as a function of signal y.
- Signal x as a function of time.
- Signal y as a function of time.
- Signal p as a function of time.

Firstly, we realise that we can read the data easily iterating through three `for` loops and using a single line to read the file, by using `sprintf(\'%02d\',user)`, which converts the user number to a 2 decimal number. This way, we can avoid the `if` statement presented in the original code:

```
for user=1:n_users
  for session = (1:4)
    for sign_genuine = (1:4)
        %This is how to load the signatures:
        BiosecurID=load(['./DB/u100', sprintf('%02d',user) ,'s000', num2str(session),
                                      '_sg000', num2str(genuine_signs(sign_genuine)), '.mat']);
```

Now, we created the script `plot_signals.m` to load a selected user, session and sign and plot the four asked signals. In the first case, we selected $user = 7$, $session = 2$ and $sign = 1$. The result is shown in Figure 2:

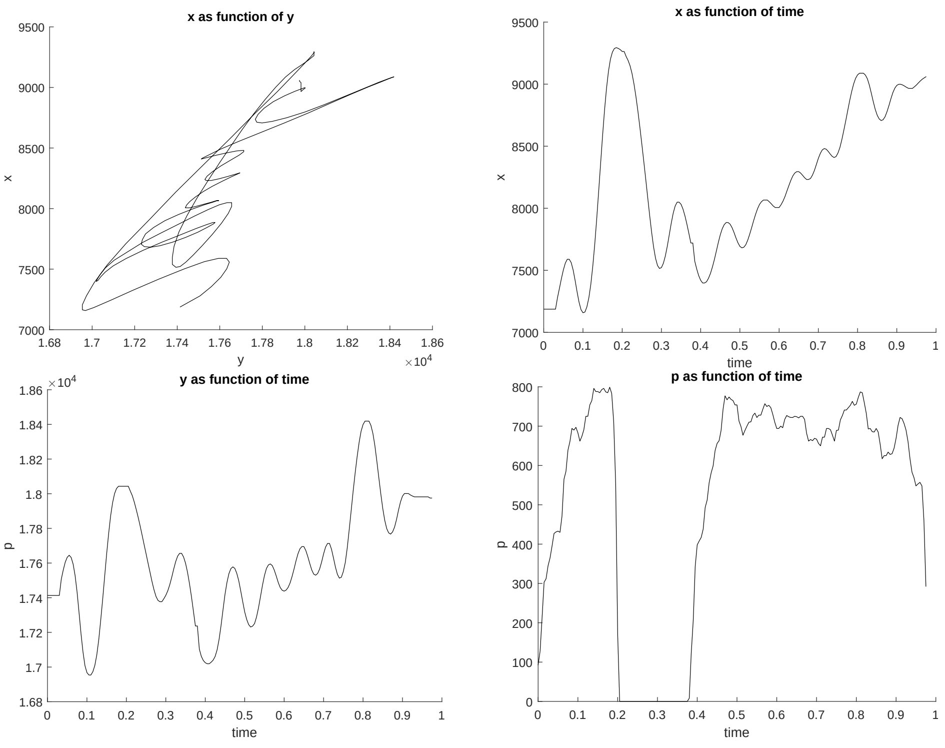


Figure 2: Plot of x as function of y, and then x, y, p as functions of t.

Repeat the process

We repeat the process, using now $session = 3$, $sign = 1$. We changed session to see if we can see any differences between different days.

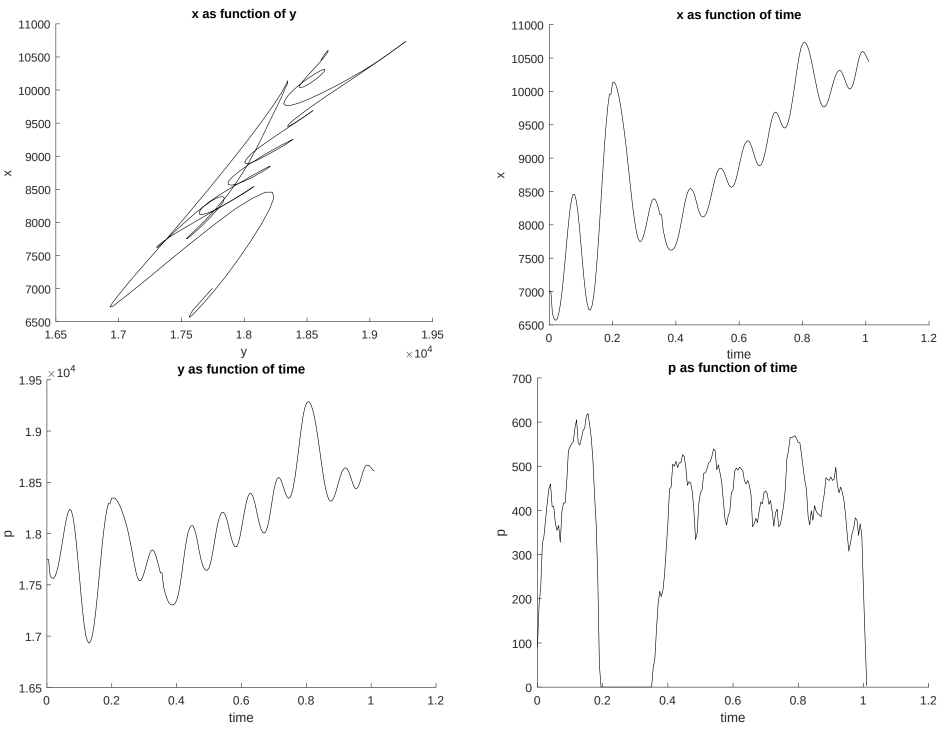


Figure 3: Second sign. Plot of x as function of y, and then x, y, p as functions of t.

Are the different signals reasonable? Do they have the same length? Why?

We can see that the two signs are relatively similar. The second one is shifted to the left, but it has the same shape and similar lines. If we have a look at x, y, p as functions of t, we can see that its shape, trend and *ups and downs* are very similar in both signatures, which tells us that very similar processes have been followed to draw each of them. Hence, we can say that they are **reasonable**.

The signs do not have the same length neither in the x, y axis nor in the t axis. We see that the second sign was drawn quite bigger (approx 1.500 difference) than the second one, and it also took approximately 0.2 seconds longer to be drawn. The **explanation** of this is simple: the time for a person drawing its sign its variable and depends on many factors, and the difference between the times is maybe not noticeable for humans, but it is for the computers that works with exact numbers.

Feature Extraction

The comparison of signals with different lengths is not trivial. Therefore, we will extract 4 global parameters of each of the signatures. So, each signature will be represented by a feature vector with fixed size equal to 4. These parameters are:

1. Total duration of the signature: T
2. Number of pen-up (number of times the pen was lifted). It means the number of times (not the number of samples) that p is equal to 0.
3. Duration of pen-down (signal p is different to 0) Td divided by the total duration $T : Td/T$
4. Average pressure in pen-down (signal p is different to 0).

You have to develop 4 functions to extract each of the parameters:

1. `T=Ttotal(x)`
2. `Npu=Npenups(p)`
3. `Tpd=Tpendown(p)`
4. `Ppd=Ppendown(p)`

According to those functions, we will develop a new function with input data (x, y, p) of a given signature and output data the feature vector containing the 4 parameters (`FeatVect=featureExtractor(x,y,p)`). Based on your function featureExtractor you have to develop a program (`ProcessBiosecurID.m`) to extract all the feature vectors from the database and store it in a matrix with 3 dimensions:

- Dimension 1: number of user (1:50)
- Dimension 2: number of signature (1:16)
- Dimension 3: number of parameter (1:4)

You have to save this matrix into the file `BiosecurIDparameters.mat`.

Once you have the file `BiosecurIDparameters.mat`, you have to plot the distributions normalized between 0 and 1 (dividing by the total number of points of the distribution) for each of the 4 parameters.
HINT: You can use the Matlab functions hist.m and histc.m

Comments about the developed code for the feature extraction.

The four implemented functions were implemented in files with the same name as they are called in the file `featureExtractor.m`. The functions are pretty simple: all of them but one are coded in **1 line**. The one that is a little bit longer is `Npenups`. In this function, we have to take into account that if we find a zero in the *pressure function*, we have to check if the previous position was **not** a zero. To achieve this, we find where the function is equal to zero and we loop over those index and check if the previous position was (or not) a zero.
To be able to code the rest of the functions in one line, *logical indexing* was very useful. This allows us to index the elements of a vector that match a condition. For instance, in the case of `Ppendown`, which computes the mean pressure when the pen is down, it is useful to use `p(p>0)` which selects from p the positions where it is greater than zero. Using this, the complete function is:

```matlab
function Ppd = Ppendown(p)
  % Compute mean pressure using positions
  % Where pen is down (p > 0)
  Ppd = mean(p(p > 0) );
end
```

QUESTION: Plot the 4 distributions.

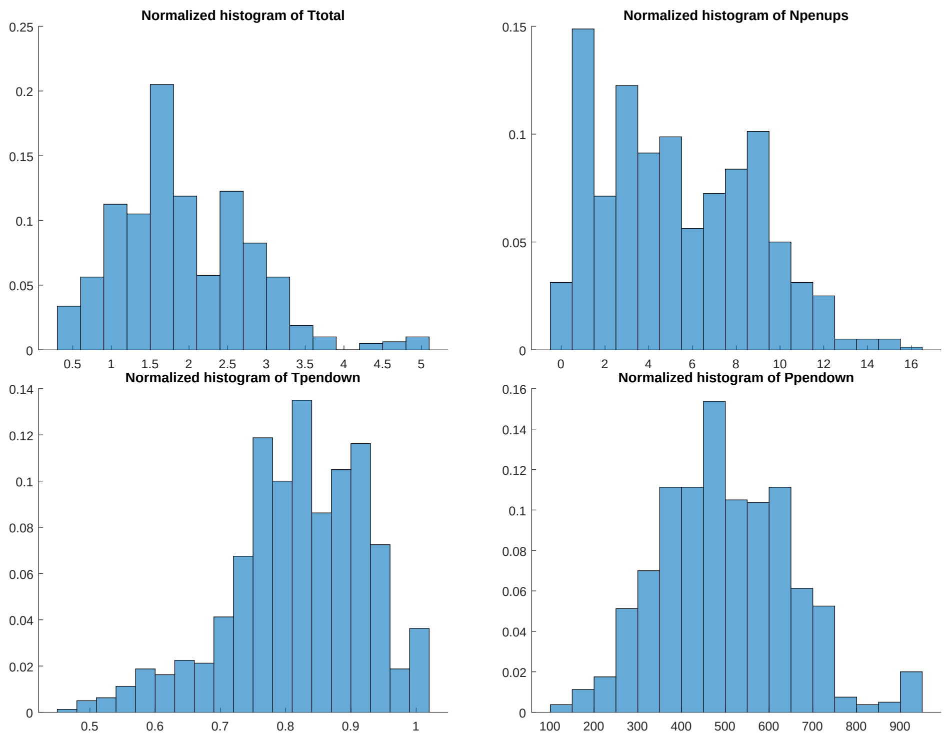


Figure 4: Normalized histograms of each of the features. Represented feature on top of the histogram.

As we can observe, we could say that the distributions of $Ttotal, Tpendown$ and $Ppendown$ are Gaussian distributions for certain parameters μ, σ. The variable $Npenups$ is a little bit different, we could say that it is a *bimodal* distribution without specifying any particular named distribution.

Performance Evaluation

We will evaluate the performance of our system according to the number of signatures N in the enrollment set $(N = 1, N = 4$ and $N = 12)$.

The similarity score between a query/test signature and the enrollment signatures (signatures in the database) will be the Euclidean distance between feature vectors (vectors with 4 parameters). The final score will be the average score of the N comparisons (comparison between the query/test sample and the N enrollment samples).

You have to develop the function `Score=Matcher(test,Model)` where:

- Score: is the final score of the comparison.
- test: is the feature vector of the query/test signature (1×4)
- Model: is a matrix containing the feature vectors of the signatures enrolled in the database. Therefore, this matrix contains Nx4 values in which N is the number of signatures enrolled for the claimed identity.

There are two cases to be analyzed:

Genuine Scores: scores obtained when you compare a signature with his real enrolled identity (claimed identity = enrolled identity). So these users should be accepted by the system. For each user you will use N signatures as enrolled samples and the rest for testing:

1. For $N = 1$ we will have $SG = 15$ genuine scores.
2. For $N = 4$ we will have $SG = 12$ genuine scores.
3. For $N = 12$ we will have $SG = 4$ genuine scores.

For each of the scenarios $(N = 1, 4, 12)$ you have to save all the genuine scores into a matrix (with dimension $50 \times SG$). Each of the three matrixes will be stored into a .mat file with name: `GenuineScores_N.mat`.

Impostor Scores: scores obtained when you compare a signature with the enrolled samples of other users (claimed identity $\neq$ enrolled identity). So these users should be rejected by the system. In this case, we will compare one signature of each user (the first one) with the models of the rest of the users (excluding the genuine case). Therefore, we will obtain SI=49 impostor scores for each user and each scenario $(N = 1, 4, 12)$.
For each scenario $(N = 1, 4, 12)$ these impostor scores will be saved into a matrix with dimensions $50 \times SI$, $(50x49)$. Each of the three matrixes will be stored into a .mat file with name: `ImpostorScores_N.mat`.
Once we obtain the genuine and impostor scores, we will evaluate the performance of our system for each of the three scenarios $(N = 1, 4, 12)$ as a function of: FAR/FRR, EER and DET curves. To obtain these performance metrics you will have available the next functions:
`[EER]=Eval_Det(GenuineScores, ImpostorScores, 'b')`, where the parameters are

1. EER: value of the Equal Error Rate (error when FAR and FRR are equal)
2. `GenuineScores`: the scores from target or genuine comparisons. These scores are obtained after applying the following normalization: $GenuineScores = 1./(GenuineScores_N + 0.00000001)$
3. `ImpostorScores`: the scores form non target or impostor comparisons. These scores are obtained after applying the following normalization: $ImpostorScores = 1./(ImpostorScores_N + 0.00000001)$

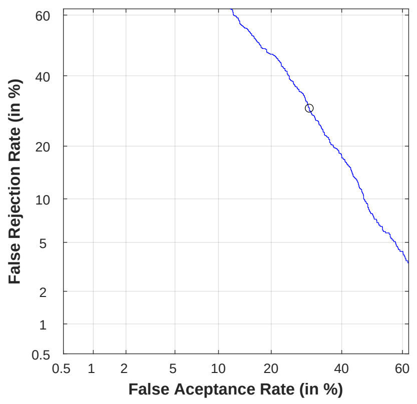
QUESTION. Plot the performance graphics (DET curves) using the genuine and impostors score stored in their respectively matrixes (for each of the scenarios $N = 1, 4, 12$). Indicate the EER value.

Firstly, we code the `Matcher` function. The code of this function is very straigthforward:

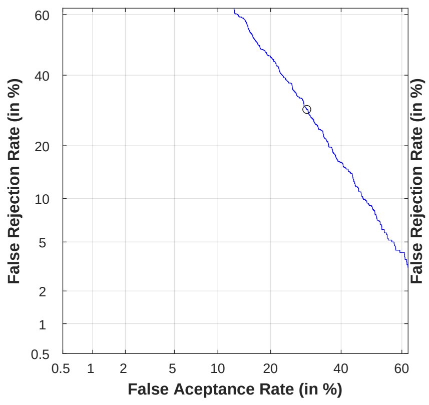
```
function Score=Matcher(test,Model)
  % Computes the euclidean distance between
  % Test and model
  % Equivalently, we can compute the norm of the difference

      N = size(Model,1);
      scores = zeros(1,N);
      for i=1:N
          scores(i) = norm(test-Model(i,:));
      end
      Score = mean(scores);
end
```

As we can see, we only have to check how many examples are we comparing the `test` features against, compute the norm of the difference for each of those examples and then computing the mean of this norm.

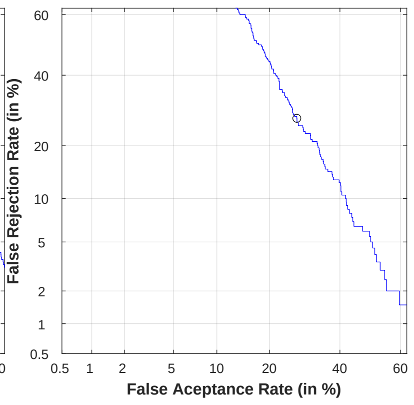
Lastly, we are provided with the `Evaluation.m` scripts that loads the Feature Matrix that we have saved in the `ProcessBiosecurID.m` script and performs an evaluation using this features. This script uses $N = 1, 4, 12$ different signs to compare against, stores and plots the results for all of them. We execute it and obtain the following DET curves and EER values, presented in Figure 5.



(a) $N = 1$



(b) $N = 4$



(c) $N = 12$

Figure 5: DET curves with EER for each of the different $N = 1, 4, 12$ with *handcrafted* features.

We also present the EER values in Table 1.

Feature extraction / N	$N = 1$	$N = 4$	$N = 12$
Handcraft	30	29.5	27.0

Table 1: EER results with *handcrafted* features.

QUESTION. According to the results, are they reasonable? What metrics are more illustrative? When do you obtain the best performance?

Extra Work 2: Changing the features

Previous note

The list of developed files for this exercise are:

- `ProcessBiosecurDTW`
- `EvaluationDTW`
- `Gradient`
- `MatcherDTW` and `SingleMatcherDTW`

Task and introduction

1. Develop an online signature recognition system based on local features (time functions) and Dynamic Time Warping for the Matcher. Repeat the same experimental protocol followed in the practice but using this new signature recognition system.
2. You should use the following local features (time functions): x, y, pressure, (and their corresponding first and second derivative).

3. You can use the DTW matcher available in Matlab. Take into account that it only allows to compare time functions of different lengths 1 to 1, i.e., x1 vs x2, y1 vs y2, etc. Therefore, you should compare time functions 1 to 1 and finally obtain the average between all time functions in order to obtain the final score of the comparison between two signatures.
4. The equation to obtain the score of the 1vs1 time function comparison is $score = e^{-\frac{D}{k}}$, where D is the minimum accumulative distance obtain after using DTW in Matlab, and K is the number of aligned time samples.

Let us revise what we have to do in this section. In previous sections, we **extracted** handcrafted features from our signs and then tested a very simple matcher based on distances. As we were asked for, we completed the *Feature extractor* and the *Matcher* that are represented in Figure 1.

Now, our goal is to change these two parts of the diagram and use different features and adapt our matcher to these new features.

5.1 The new feature extractor

Our new features will be the *raw time series* obtained from the sensor. We will also use its first and second derivatives to increase the amount of available information and they will be useful since they provide extra information about the original time series. To sum up, we will have the following features for each sign:

$$\{x, y, p, dx, dy, dp, d^2x, d^2y, d^2p\}$$

In order to obtain the derivatives, we must note that the discrete derivatives of a time series are defined as

$$\Delta[f](x) = f(x+1) - f(x),$$

that is, the difference between the next and the current values of the time series. Using this, we define the following `Gradient` function in the script that has the same name:

```
function grad=Gradient(a)
% Compute the gradient of a discrete time signal
    grad = a(2:length(a)) - a(1:length(a)-1);
end
```

Having a function to compute the gradient and the features, we are ready to store all the features in a matrix with all the features for each sign and for each user. However, we realised that in this case this is not so easy, since now we are working with features that **do not have the same length** for each sign. Remember that in the previous case, we had 4 features for each sign, which is not the case now.

Our **workaround** is to make use of MatLab `cells` [1]. This data structure allows us to have variable length vectors in the same structure. So, for the new feature extraction, we created a cell

```
BiosecurIDparameters=cell(50,16,9);
```

and then introduced the data as follows (the introduction of the derivatives is not shown to reduce code space in the memory)

```
% Save parameters
sign_idx = (session - 1)*4 + sign_genuine;

BiosecurIDparameters{user}{sign_idx}{1} = BiosecurID.x;
BiosecurIDparameters{user}{sign_idx}{2} = BiosecurID.y;
BiosecurIDparameters{user}{sign_idx}{3} = BiosecurID.p;
```

Lastly, as we did in the previous feature extractor, we save the data into a matlab file named `BiosecurIDparametersDTW`.

The new matcher

We have our time series ready to be matched with other time series. We are asked to make use of **Dynamic Time Warping** in order to make this matching. A few considerations must be taken into account when coding this DTW matcher.

[1]As a note, when introducing a variable to this cells in the $n-th$ position, be sure that the previous positions have values; otherwise MatLab fills them with zeros and this leads to problems with the following parts of the code (a few hours were wasted facing this problem).

1. The distance between the time series is computed as

$$dist(x_t, y_t) = e^{-\frac{D}{k}},$$

   where x_t, y_t are the time series, D is the distance computed by the DTW and k is the number of aligned time samples.

2. `MatLab` has a `dtw` function (see the documentation for this function), that returns the distance D and two lists ix, iy with the index of both signals x_t, y_t taken to compute this distance. We can count the number of coincidences between ix, iy to find the desired k.

3. The comparison must be done for each of the 9 features (time series) that we have chosen. After computing this distance for each feature, we **average** the distances.

4. Lastly, we found during the experimentation that the distance returned from the DTW depends on the scale of the time series. That is, larger values of the time series result in larger distances, which causes larger D values for our score. Hence, we decided to **divide each time series by its maximum** so that all the time series are in the range $[0, 1]$ when they are compared with the DTW.

With these considerations, we created the script `SingleMatcherDTW` that returns, two signs as input s_1, s_2, the average score obtained by the DTW method. The code is simple:

```
function Single_Score_DTW=SingleMatcherDTW(test,Model)
% Computes the score using e^{-D/k} where D is the DTW
% distance and k the number of coincident points in the DTW
    N = size(Model,2);
    scores = zeros(1,N);
    for i=1:N
        [D,ix,iy] = dtw(test{i}/max(test{i}),...
                        Model{i}/max(Model{i}));
        k = sum(ix == iy);
        scores(i) = exp(-D/k);
    end
    Single_Score_DTW = mean(scores);
end
```

Comparing with N signatures

Lastly, as we did in the previous Matcher, we will compare with $N = 1, 4, 12$ signatures. To be able to do this, we will only have to iterate through the N subjects, use the `Single_Score_DTW` function for each of them, and finally compute the average score obtained, which is given by:

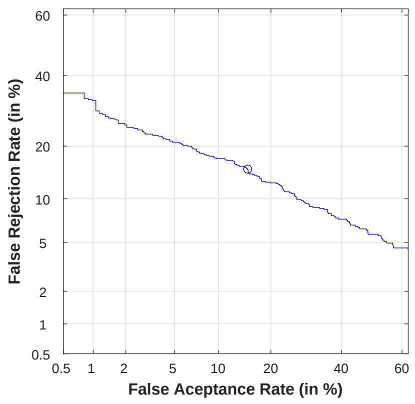
$$\text{Score} = \frac{1}{N} \sum_i \left(\frac{1}{9} \sum_j e^{-\frac{D_{ij}}{k_{ij}}} \right)$$

where subindex i refers to the $i-th$ comparing sign and subindex j refers to the $j-th$ comparing feature. This is done in the script `MatcherDTW`, which code is pretty straigthforward.

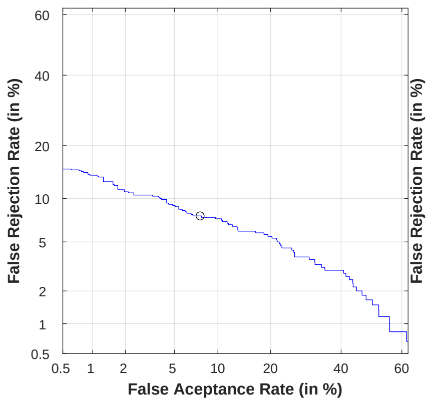
Evaluation

We have our new **features** and **matcher**, now we can use them to determine the DET curve and the EER for our signature recognition system. The previous `Evaluation.m` script is no more valid due to the differences in the data (we now have variable-length time signals instead of 4 features). However, the structure of the new developed script is the same. The code can be found on `EvaluationDTW`, and the main changes with respect to the previous script are in the way of *loading* the signs to be compared.

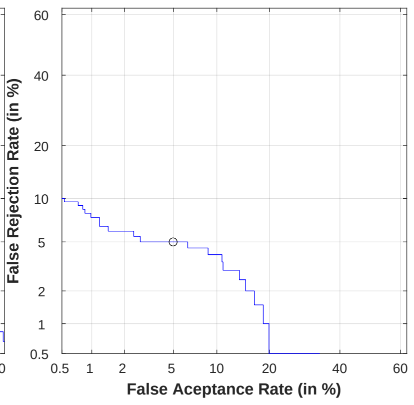
We executed the script to find the DET curve and EER value, with the results obtained shown in Figure 6.



(a) $N = 1$



(b) $N = 4$



(c) $N = 12$

Figure 6: DET curves with EER for each of the different $N = 1, 4, 12$.

We also present the EER results in Table 2.

Feature extraction / N	$N = 1$	$N = 4$	$N = 12$
Time Series	15.06	7.66	5.0

Table 2: EER for the time series features and DTW matcher.

As we can see, the more signs considered in the comparison, the lower EER (and, hence, the better results) we obtain, as it happened in the previous recognition system.