seizureDetection Matlab Tool User Guide

Author: Paul Meyer-Rachner, paul@meyer-rachner.email
Date: May 2018

Contents

1 Introduction

These MATLAB classes and functions were developed for the Berkeley MEng Capstone Project *On-Chip Biosignal Computation for Health Monitoring* at the University of California, Berkeley Department of Electrical Engineering and Computer Science during the 2017-2018 academic year. The advisor was professor Rikky Muller.

The goal of the project was to develop a implantable microchip that monitors the brain's electrical activity (electroencephalogram or EEG) to detect seizures. This tool was developed while coming up with a detection algorithm that could reliably detect seizures while minimizing hardware complexity.

> **Note: The final project report should also be available to download at the same location as this tool. Reading it, especially the software part (section 2.1) of it will help understand this tool. Furthermore, all functions should have ample comments and the `main.m` file contains example of how to uses the functions and classes provided by this tool.**
>
> **This documentation is not exhaustive, it is supposed to give an overview of the tool. Many functions in the code are not documented here (but there are ample comments within the code).**

2 Tool overview

The main component of the tool are 4 MATLAB classes:

- The `Measurement` class holds raw EEG data.
- The `FeatureCalculation` class is used to extract different features from the data. It offers several functions to calculate that feature, further process it and visualize it. Every feature has it's own class that is a children of the FeatureCalculation class.
- The `FeatureMatrix` class serves to hold multiple FeatureCalculation objects inside a matrix.
- The `MatrixDetection` class is used to detect seizures. It uses the information of all the features stored inside a `FeatureMatrix` object to calculate a weighted sum that will be used to detect seizures.

The usual workflow is as follows:

1. Create a Measurement object with some EEG data. This can be achieved either by downloading the data from iEEG.org using the function `IEEG_getData` or manually creating a Measurement object with the appropriate data.
2. Filter the data using the `preprocess` function of the Measurement class. Then calculate different features and store those in a `FeatureMatrix` object. This can be done by adapting the script `generateMatrix`.
3. Create a `MatrixDetection` object and use it to detect seizures.

3 The Measurement class

The `Measurement` class holds the EEG data and the location of the seizures.

The function `IEEG_getData` allows to download data from the iEEG.org[1] website (see section 8.1).

3.1 Class Properties

The table below show the properties of the `Measurement` class. Some of these properties have their SetAcces set to protected, meaning it is not possible to modify them directly from outside the class. This is done to protect the integrity of the object by preventing accidental modification of important properties.

Name:	**Description:**
`Data`	holds the EEG data
`Ts`	sampling time in seconds
`Fs`	sampling frequency in Hz (equivalent to 1/Ts)
`Length`	length of the Data property in samples
`NumSeizures`	number of seizures present in the measurment
`SeizureStart`	a double array containing the start of each seizure in terms of samples
`SeizureEnd`	a double array containing the end of each seizure in terms of samples
`SeizureDuration`	the duration of each seizure in seconds
`Duration`	duration of the measurement in seconds
`isNaN`	a logical array that has the same length as the Data property. For some measurement, the original data contains NaN values. This results in issues when filtering the data. Thus the object is created, these NaN values are replaced with their nearest non NaN value, but their position is remembered so that the original NaN value can be set again at a later point.
`FilterOrder`	the order of the bandpass filter used to preprocess the data
`FilterFc1`	the lower cutoff frequency of the bandpass filter, in Hz
`FilterFc2`	the higher cutoff frequency of the bandpass filter, in Hz
`StudyName`	a string containing the original name of the study the data is coming from. This is mainly used to identify the data.
`StudyChannelNumber`	a double holding the channel number the data is coming from. This is mainly used to identify the data.
`StudyChannelName`	a string holding the identifier of the channel. This is mainly used to identify the data.

3.2 Class methods

3.2.1 Constructor

An instance of the Measurement class is created by calling the class constructor:

```
measurement = Measurement(data, Fs, seizureStart, seizureEnd, 'Name', 'Value')
```

It has 4 required input arguments. `data` is the raw EEG data and `Fs` the sampling frequency. `seizureStart` and `seizureEnd` are two vectors containing the start and end location in samples of each seizure.

[1] `https://www.ieeg.org`

Possible Name-Value pairs for customization are:

Name:	Type:
`StudyName`	char
`StudyChannelNumber`	scalar double
`StudyChannelName`	char

The `FilterOrder`, `FilterFc1` and `FilterFc2` are initialized with default values 6, 1 Hz and 70 Hz respectively and can be changed once the object is created.

3.2.2 Preprocess function

The `preprocess()` function filters the data using a Butterworth bandpass filter with order and cutoff frequencies specified in the object properties. This function has no input arguments.

The filter object is created using MATLAB's builtin `fdesign.bandpass` and `design` functions. The resulting feature object is a second-order section implementation, meaning that several second order Butterworth filters are cascaded. For example, if original order is 6, then 3 second order sessions will be cascaded. This implementation reduces the hardware requirement of the filter and prevents the filter from becoming unstable due to the numerical instability introduced by the bilinear transformation in the filter design process.

3.2.3 Downsample function

The `downsample(n)` function downsample the measurement by an integer factor. For example, if the original sampling frequency is 512 Hz, then after calling `downsample(2)`, the sampling frequency will be 256 Hz.

The function takes one input, `n`, which is the downsampling factor. It has to be an integer, and the resulting frequency also has to be an integer. For example, a 512 Hz measurement cannot be downsampled by 3.

Internally, the MATLAB function `decimate` is called, which applied an anti-aliasing lowpass filter to the data before downsampling it to avoid aliasing.

All object properties like `seizureStart` are adjusted to the new sampling frequency.

3.2.4 Cut function

The function `cut(startIndex, endIndex)` trims the measurement to the indices given as input parameters. The function issues a warning if seizures are removed from the data and throws an error if the start or end index are within a seizure. All object properties like `seizureStart` and `duration` are adjusted to the start and end.

3.2.5 Other functions

The function `plot()` plots the current EEG data. Seizures are plotted in red. If the measurement is very large, it will take a while to run and the resulting plot will not be very responsive when interacted with.

The function `isSeizure(queryIndices)` return a logical array of the same size as the input argument `queryIndices`. For every index inside `queryIndices`, the output is `True` if that index corresponds to a seizure in the measurement and `False` otherwise.

4 The FeatureCalculation class and it's children

The `FeatureCalculation` class is an abstract class used to calculate one feature from the EEG data. It has 4 children that can be instantiated:

- The `LineLength` class to calculate the line length feature
- The `NonlinearEnergy` class to calculate the nonlinear energy feature
- The `Power` class to calculate power in the whole spectrum
- The `SpectralBandPower` class to calculate the power in one spectral band

See appendix A.1 for the mathematical formulas of how those features are implemented.

4.1 Class Properties

The table below show the properties of the `FeatureCalculation` class. As for the `Measurement` class, some of these properties have their SetAcces set to protected. The term 'feature sample' refers to the update frequency or step size of the sliding window used to calculate the feature. For example, it is 0.2 seconds by default, meaning one feature sample is equivalent to 0.2 seconds and one new feature value will be calculated every 0.2 seconds.

Name:	**Description:**
`FeatureName`	the name of the feature
`Value`	the value of the feature
`CharacteristicCurve`	holds an `OperatingCharacteristicCurve object` (see section 7)
`StudyName`	study name of original measurement, copied from the linked measurement
`StudyChannelNumber`	channel number of original measurement, copied from the linked measurement
`StudyChannelName`	identifier of the channel, copied from the linked measurement
`BaselineWindowLength`	the length of the window over which the baseline is calculated, in term of feature samples
`BaselineShift`	the amount of time the window over which the baseline is calculated is delayed, in term of feature samples
`BaselineRefreshrate`	how often the baseline value is updated, in term of feature samples
`ThresholdBaselineFactor`	factor by which the baseline is multiplied to obtain the threshold
`LinkedMeasurement`	a handle (reference/link) to the `Measurement` object that is being analysed
`PlotHandle`	a handle to the plot of the feature (if there is one)
`AxesHandle`	a handle array containing the axes handles of the plot (if there is one)
`CostSensitivity`	factor for calculating cost of not detecting seizures when finding the best threshold
`CostFalseAlarmRate`	factor for calculating cost of false alarms when finding the best threshold
`SeizureHoldTime`	duration of a seizure for evaluation purpose in feature samples (see section 4.2.3)
`Baseline`	feature value baseline, average of feature value over a longer window, used to calculate the threshold
`StepSize`	update frequency of the sliding window, in terms of measurement samples (for example, if the sampling frequency of the measurement is 500Hz and the update frequency is 0.2s, then `StepSize` will be 100)

`WindowLength`	the length of the window used to calculate the feature value, in terms of measurement samples (for example, if the sampling frequency of the measurement is 500Hz and the update frequency is 1s, then `WindowLength` will be 500)
`Length`	total number of calculated values
`SeizureStart`	start index of the seizures, in terms of feature samples
`SeizureEnd`	end index of the seizures, in terms of feature samples
`NumSeizures`	total number of seizures
`MeasurementFs`	sampling rate of linked measurement, useful is case the link in the `LinkedMeasurement` property is deleted
`Duration`	duration of linked measurement, useful is case the link in the `LinkedMeasurement` property is deleted
`Sensitivity`	percentage of detected seizures
`Specificity`	percentage of time above threshold outside of seizure areas
`FalseAlarms`	number of false alarms
`ThresholdCrossings`	number of times the feature value exceeded the threshold
`FalseAlarmRate`	number of false alarms per hour
`MissedSeizures`	locations of seizure that were not detected, in terms of feature samples
`BitSeizureDetected`	Logical array, index `BitSeizureDetected[i]` is `True` when the ith seizure is detected by this feature and `False` otherwise
`ThresholdCrossedLocation`	indices where the feature value crossed the threshold, in terms of feature samples
`FalseAlarmLocation`	indices of false alarms, in terms of feature samples

Additionally, the `SpectralBandPower` has two additional properties:

- `FrequencyBounds` contains the upper and lower frequency limit in Hz of the band. It is also used as cutoff frequencies when filtering the data during the feature value calculation.
- `FilterOrder` is the order of the Butterworth bandpass filter used to filter the data

4.2 Class Methods

4.2.1 Constructors

The `FeatureCalculation` is an abstract class and can therefore not be instanciated.

The objects of the classes `LineLength`, `NonlinearEnergy` and `Power` can be created by calling

```
linelength = LineLength(measurement, 'Name', 'Value')
nonlinearEnergy = NonlinearEnergy(measurement, 'Name', 'Value')
power = Power(measurement, 'Name', 'Value')
```

where measurement is a `Measurement` object.

Possible Name-Value pairs are:

Name:	Type:	Default:	Description
StepSize_seconds	double	0.2	defines by how much the sliding window is moved in seconds (equivalent to how often the sliding window is updated). This can be thought of as the feature sampling frequency.
WindowLength_seconds	double	1	the length of the sliding window in seconds
BaselineAverageTime_seconds	double	180	over what period of time the baseline is calculated, is seconds
AverageWindowShift_seconds	double	120	by how much the window over which the baseline is calculated is delayed, in seconds
BaselineRefreshRate_seconds	double	30	how often the baseline is recalculated/refreshed, in seconds
SeizureHoldTime_seconds	double	60	duration of a seizure for evaluation purpose in feature samples (see section 4.2.3)
CostSensitivity	double	50	factor for calculating cost of not detecting seizures when finding the best threshold
CostFalseAlarmRate	double	1	factor for calculating cost of having many false alarms when finding the best threshold
ThresholdBaselineFactor	double	5	factor by which the baseline is multiplied to obtain the threshold

Instances of the `SpectralBandPower` class are created by calling

```
thetaBand = SpectralBandPower(measurement, bandName, frequencyBounds)
```

where `bandName` is a char containing the name of that band and `frequencyBounds` an double array of length 2 containing the upper and lower limit of the band in Hz. Additionally, the order of the filter used to filter the data can be passed as a Name-Value pair, for example

```
thetaBand = SpectralBandPower(measurement, 'ThetaBand', [4 8], 'FilterOrder', 6)
```

The constructors of all 4 children class convert the parameters specified by the Name-Value pairs (or the default values) from duration in seconds to length in feature samples.

4.2.2 Calculate function

The `calculate()` function calculates the feature value using the filtered data in the `Measurement` object linked to it. It does not take any input arguments. It calls the `calculateFeatureValue` that has to be implemented by the children classes. The `calculateFeatureValue` used the MATLAB builtin function `movmean` since it allows for a fast calculation of the feature value.

The `calculate` function also calculates the baseline. The baseline is the average of the feature value over a longer window (180 seconds by default). This window is however delayed by a certain amount (120 seconds by default). For example, the baseline corresponding to time t will be calculated by averaging the feature value from time $t - 300$ to time $t - 120$.

4.2.3 Evaluate function

The `evaluate()` function calculates the seizure detection metrics using the current value of the `ThresholdBaselineFactor` property. See appendix A.2 for a definition of some of the metrics calculated. It does not take any input arguments.

One seizure counts as detected when the feature value crosses the threshold at least once during the seizure time.

One false alarm is defined as the feature value crossing the threshold outside of seizure areas. However, if the threshold is crossed repeatedly within a short timeframe (specified by the `SeizureHoldTime`) property, only the first threshold crossing is marked as a false alarm and all others are ignored. This is to avoid generating too many false alarms when the feature values oscillates around the threshold. This also mirrors the behavior of the actual seizure detection chip, where a response is trigger when a seizure is detected. This response cannot be trigged in too short time intervals. Thus, internally, the function `calculateFalseAlarmLocation` is called to identify and delete such redundant false alarms.

4.2.4 CalculateOperatingCharacteristicCurve function

The `calculateOperatingCharacteristicCurve()` calculates the sensitivity and false alarm rate for different `ThresholdBaselineFactor` and uses the result to create a `OperatingCharacteristicCurve` object (see section 7) that is then stored in the `CharacteristicCurve` property. It does not take any input arguments.

The sensitivities and false alarm rates are calculated by calling the `evaluate` function repeatedly for different values of `ThresholdBaselineFactor`.

4.2.5 FindBestThreshold function

The `findBestThreshold()` function finds the value of `ThresholdBaselineFactor` that minimizes a cost function and stores it inside that property. It does not take any input arguments. The cost function is currently defined by

$$\text{Cost} = \text{CostSensitivity} \times (1 - \text{Sensitivity})^2 + \text{CostSensitivity} \times \text{FalseAlarmRate}.$$

The `findBestThreshold` function uses the sensitivities and false alarm rates stored in the `OperatingCharacteristicCurve` object to calculate the cost, therefore the function `calculateOperatingCharacteristicCurve` must be called before the `findBestThreshold` function.

4.2.6 Plot function

The `plot(flags)` function plots the feature value. Several flags can be passed as chars to the function to modify how the plot looks:

- `'Measurement'`: also plot the linked measurement data. This will results in a plot with two subplots and can be slow if the measurement is large.
- `'Threshold'`: plots the threshold in addition the feature value
- `'Baseline'`: plots the baseline in addition the feature value
- `'Mark'`: mark missed seizures and false alarms on the plot
- `'Bare'`: removes legends and ticks and other annotations from the plot

For example, the plot in figure 1 was created by calling the command

```
linelength.plot('Measurement', 'Threshold', 'Mark').
```

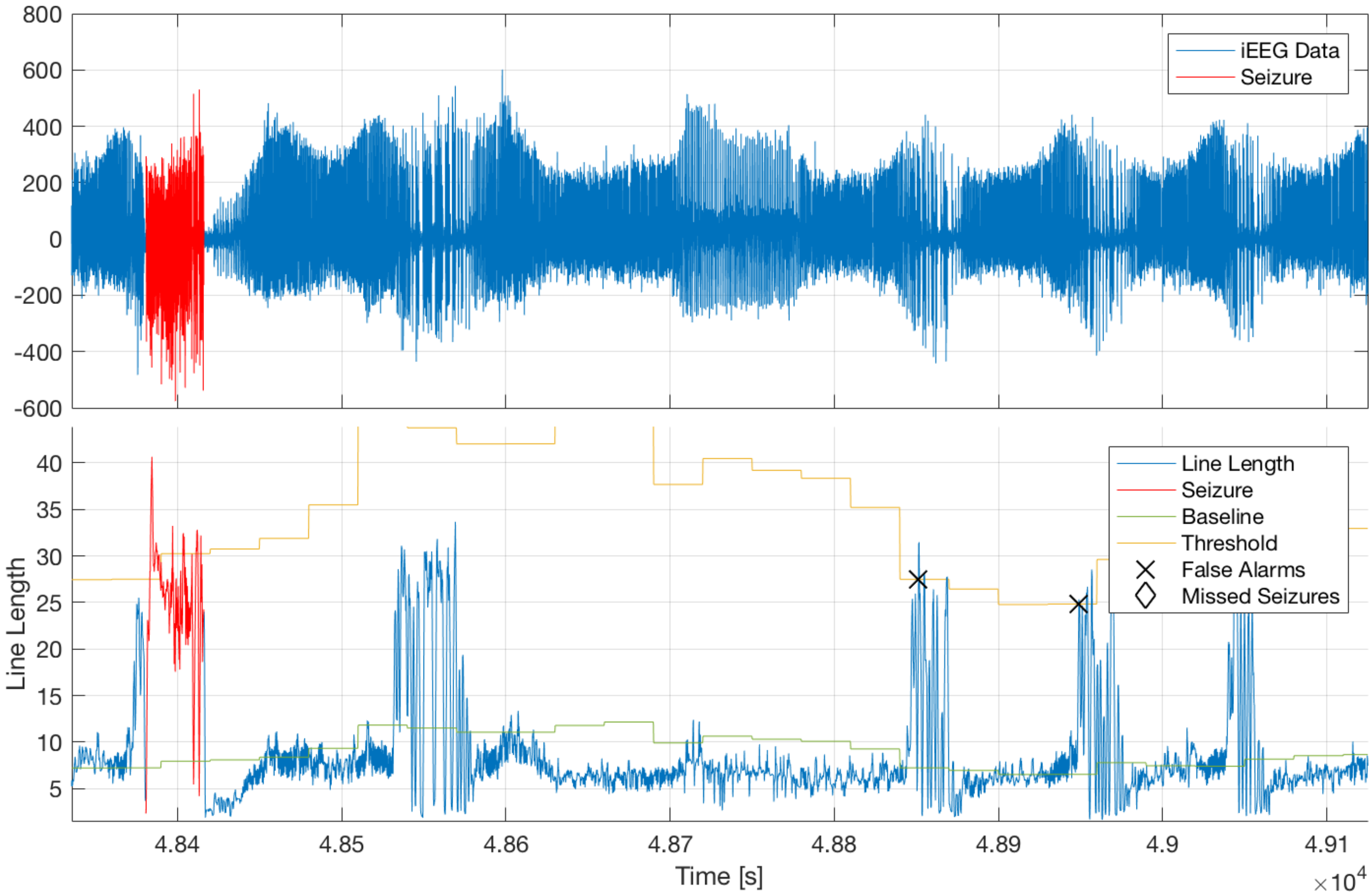

Figure 1: Example of a plot generated with the `plot` function of the `FeatureCalculation` class.

5 The FeatureMatrix class

The `FeatureMatrix` class provides a container to store `FeatureCalculation` objects (more precisely, it's children object) inside a matrix.

A feature matrix object can be generated by adapting the `generateMatrix` script.

5.1 Class Properties

All the properties of the `FeatureMatrix` class are private.

Name:	**Description:**
`FeatureNames`	a cell array of strings containing the names of the features
`Matrix`	the actual feature matrix, it is a cell array of size number of channels $\times$ number of features
`NumChannels`	number of channels (and therefore rows) in the feature matrix
`NumFeatures`	number of different features (and therefore columns) in the feature matrix
`NumElements`	the total number of elements in the matrix (equals to number of rows $\times$ number of columns)
`FeatureLength`	the length in samples of one feature
`NumSeizures`	the number of seizures in one feature
`SensitivityMatrix`	a double matrix of the same size as `Matrix` that holds the sensitivity of each feature
`SpecificityMatrix`	a double matrix of the same size as `Matrix` that holds the specificity of each feature
`FalseAlarmRateMatrix`	a double matrix of the same size as `Matrix` that holds the false alarm rate of each feature
`AreaMatrix`	a double matrix of the same size as `Matrix` that holds the area under curve between a false alarm rate of 0 and 1 of each feature
`FlagMatrix`	a 3-dimensional matrix of logical values. Index $[i, j, k]$ is true when for channel i feature j is above it's threshold at index k. The dimension is [number of channel $\times$ number of features $\times$ length of one feature]
`BitSeizureDetectedMatrix`	a 3-dimensional matrix of logical values. Index $[i, j, k]$ is true when for channel i feature j detected seizure number k. The dimension is [number of channel $\times$ number of features $\times$ number of seizures]

5.2 Class Methods

5.2.1 Constructor

The construct of the `FeatureMatrix` class has two required input arguments, the number of channels `numChannels` which is a double and a cell array of string containing the names of the features that will be included in the feature matrix.

```
matrix = FeatureMatrix(numChannels, featureNames)
```

The feature names have to match exactly with the `FeatureName` properties of the features classes.

5.2.2 Filling the feature matrix entries

An single element can be added to the matrix by the following command:

```
matrix.setElement(indexChannel, feature)
```

where `indexChannel` is a double corresponding to the channel number and `feature` is a feature calculation object. The `setElement` function will compare the `FeatureName` property of the object passed as input argument and compare it with the `FeatureNames` property of the `FeatureMatrix` object to determine the correct insertion column. This is why when instantiating a `FeatureMatrix` object, the feature names passed to the constructor have to match the names of the feature objects.

5.2.3 Filling the other matrices

The matrices `SensitivityMatrix`, `SpecificityMatrix`, `FlagMatrix` and `BitSeizureDetectedMatrix` can be filled by calling the function `fillMatrices()`. These matrices are mainly used for the matrix (weighted sum) seizure detection.

The matrix `AreaMatrix` can be filled by calling the `fillAreaMatrix()` function. This requires all elements inside the feature matrix to have their `CharacteristicCurve` property calculated.

5.2.4 Plotting all features in on channel or all features of one kind

The function `plot(strWhat, index)` plots either all features in one channel if the input argument `strWhat` is set to `'feature'` or one specific feature for all channels if it is set to `'channel'`. The feature or channel to plot is determined by the second input argument `index`. For example, to plot all features of channel 1 call `plot('channel', 1)` and to plot the nonlinear energy feature for all channels call `plot('feature', '2')` (assuming the nonlinear feature corresponds to the seconds column). As with the `plot` function of the `FeatureCalculation` object, it is possible to pass additional flags to the function to customize the appearance of the plot (see section 4.2.6). The possible flags are:

- `'Threshold'`: plots the threshold in addition the feature value
- `'Baseline'`: plots the baseline in addition the feature value
- `'Mark'`: mark missed seizures and false alarms on the plot
- `'Bare'`: removes legends and ticks and other annotations from the plot

For example, the plot in figure 2 was created using the command

```
matrix.plot('feature', 1, 'Threshold', 'Mark', 'Bare').
```

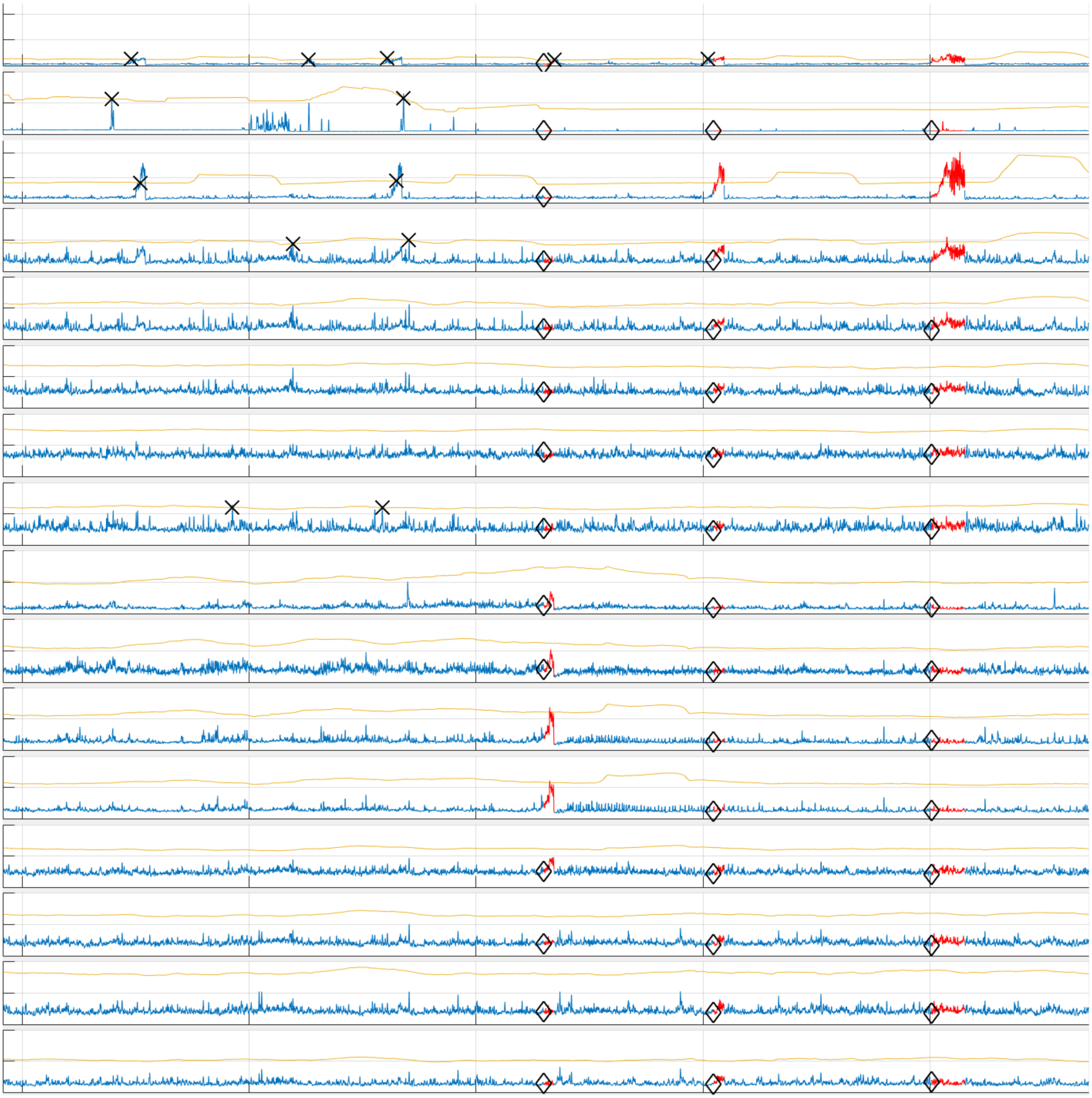

Figure 2: Example of a plot generated with the `plot` function of the `FeatureMatrix` class. The plot is not annotated because the flag `'Bare'` was used.

6 The MatrixDetection class and it's children

The `MatrixDetection` class is a class designed to handle the weighted sum detection of seizures. The idea is to assign a weight to every feature of the feature matrix inside the `FeatureMatrix` class. The weight is determined by how good that feature is at detecting seizures. For more information on how and why this is done, please read the Capstone Project Final Report that is also available with this user guide.

The class can be instantiated on it's own, but usually one of it's children is used. Currently, there is only one children of this class, the `MatrixDetection_LinearRegression` class.

This class is in many aspects similar to the `FeatureCalculation` class.

6.1 Class properties

This class has many of the properties of the `FeatureCalculation` class.

Name:	**Description:**
`WeightMatrix`	a double matrix that holds the weight of every element of the feature matrix
`Name`	name of the object
`Value`	the value of the weighted sum
`CharacteristicCurve`	holds an `OperatingCharacteristicCurve object` (see section 7)
`Threshold`	the threshold used for detection. Unlike in the `FeatureCalculation` class, this is a fixed value as there is no baseline in the `MatrixDetection` class.
`LinkedMatrix`	a handle (reference/link) to the `FeatureMatrix` object that is being analysed
`NumElements`	number of elements in the linked feature matrix
`CostSensitivity`	factor for calculating cost of not detecting seizures when finding the best threshold
`CostFalseAlarmRate`	factor for calculating cost of false alarms when finding the best threshold
`SeizureHoldTime`	duration of a seizure for evaluation purpose in feature samples (see section 4.2.3)
`FeatureLength`	the length of the individual features, also the length of the weighted sum value
`SeizureStart`	start index of the seizures, in terms of feature samples
`SeizureEnd`	end index of the seizures, in terms of feature samples
`NumSeizures`	total number of seizures
`Sensitivity`	percentage of detected seizures
`Specificity`	percentage of time above threshold outside of seizure areas
`FalseAlarms`	number of false alarms
`ThresholdCrossings`	number of times the feature value exceeded the threshold
`FalseAlarmRate`	number of false alarms per hour
`MissedSeizures`	locations of seizure that were not detected, in terms of feature samples
`BitSeizureDetected`	Logical array, index `BitSeizureDetected[i]` is `True` when the ith seizure is detected by this feature and `False` otherwise
`ThresholdCrossedLocation`	indices where the feature value crossed the threshold, in terms of feature samples
`FalseAlarmLocation`	indices of false alarms, in terms of feature samples
`Ranking`	a [number of elements $\times$ 1] struct that ranks all the different features by their weight

`ThresholdFactorValues`	different threshold values to evaluate when calculating the operating characteristic curve. The `calculateOperatingCharacteristicCurve` function will evaluate the detection at all threshold specified in this property.

6.2 Class methods

6.2.1 Constructor

The constructor of the `MatrixDetection` class has one required input argument `matrix` that is an `FeatureMatrix` object.

Possible Name-Value pairs are:

Name:	**Type:**	**Default:**	**Description**
`Name`	char	'Generic Matrix Detection'	name of the object
`CostSensitivity`	double	50	factor for calculating cost of not detecting seizures when finding the best threshold
`CostFalseAlarmRate`	double	1	factor for calculating cost of having many false alarms when finding the best threshold
`ThresholdFactorValues`	double vector	[]	the threshold values at which to evaluate when calculating the operating characteristic curve

Some of the class properties are simply copied from the `FeatureCalculation` object inside the feature matrix.

6.2.2 Calculate function

The `calculate()` function calculates the weighted sum value by using the weights inside the `WeightMatrix` property. After calculating the weighted sum, the operating characteristic curve is calculated and the best threshold is found.

The weight calculation either has to be implemented by the children class (see section 6.3) or the weight matrix has to be set externally.

The way the operating characteristic curve and the best threshold are computed is identical to the way it is done in the `FeatureCalculation` class. The only difference is that the operating characteristic curve is calculated by only evaluating the threshold specified in the `ThresholdFactorValues` property. If it is empty, 100 values linearly spaced between 0 and maximum of the weighted sum are used instead.

The equivalent of the `evaluate` function in the `FeatureCalculation` class it the `weightedDetection` function.

6.2.3 PlotWeightedDetection function

Similar to the `plot` function of the `FeatureCalculation` class (see section 4.2.6) except that the possible input flags are restricted to `'Threshold'` and `'Mark'`.

6.3 The MatrixDetection_LinearRegression class

The `MatrixDetection_LinearRegression` is a children of the `MatrixDetection` detection class. It has three additional properties:

- `CalculationFunctionHandle`: a function handle to the function that is used to calculate the weights
- `WeightSeizure`: the weights given to seizures when calculating the weights
- `WeightInterictal`: the weights given to non-seizure samples when calculating the weights

As such, the constructor for the `MatrixDetection_LinearRegression` class has two more possible Name-Value pairs `'WeightSeizure'` and `'WeightInterictal'`.

6.3.1 CalculateWeights function

This function calculates the weights of each feature inside the feature matrix.

Before calling this function, the `CalculationFunctionHandle` has to be set. This can be the standard ridge regression calculation

$$\hat{w} = (X^\top X + \lambda I)^{-1} X^\top y$$

in which case the handle should be

```
dummyObject.CalculationFunctionHandle = @(X, y) (X'*X + 0.01*eye(size(X'*X)))\X'*y.
```

The matrix X is a modified version of the `FlagMatrix` property of the `FeatureMatrix` class. One columns of X corresponds to one feature. This matrix contains either 0 or 1, depending on whether the individual features are above or below their respective thresholds. Please refer to the Capstone Project Final Report for more information. All samples corresponding to seizures are removed from the flag matrix. Then, a single sample per seizure is added again at the end of the matrix. For every feature, this sample is 1 if that feature detected that seizure and zero otherwise. Therefore, a vector of length equal to the number of seizures is added at the end of every columns. *Note: This implementation was chosen because it worked well on some measurements, but it might not be ideal.*

The y matrix then is vector containing all zeros except for the last n entries which are 1, where n is the number of seizures.

7 The OperatingCharacteristicCurve class

The `OperatingCharacteristicCurve` class holds data used to plot the operating characteristic curve of a feature. The operating characteristic plot is a plot of the sensitivity vs. the false alarm rate. See figure 3 for an example.

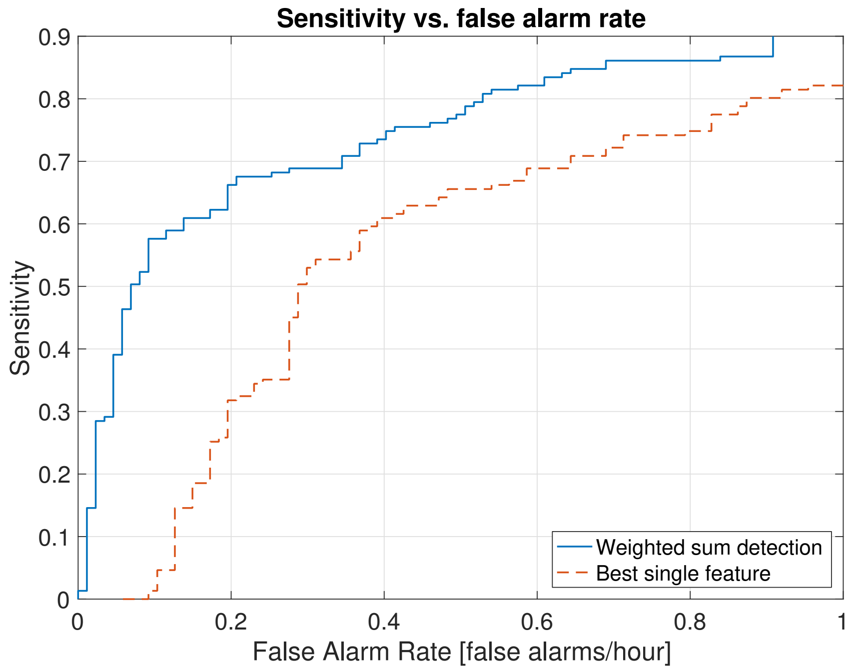

Figure 3: An exemplary operating characteristic plot.

The properties of the `OperatingCharacteristicCurve` class properties are:

Name:	**Description:**
`Name`	name of the object, used to identify the curve once plotted in the plot legend
`Sensitivity`	a vector that holds the sensitivity values for different thresholds
`FalseAlarmRate`	a vector of that holds the false alarm rates for different thresholds
`ThresholdFactor`	a vector that holds the thresholds (or threshold factor values) corresponding the sensitivities and false alarm rates in the above properties
`Area`	area under the curve

The constructor for this class is

```
curve = OperatingCharacteristicCurve(name, sensitivity, falseAlarmRate, threshold)
```

where the first input argument is a char and the last three input arguments are vectors of the same size.

The characteristic curve can be plotted by calling `plot()`. This will create a new figure with the characteristic curve plotted in it. The function `plot()` returns two output arguments. The first is a handle to the newly created figure and the second one is a handle to the axis that contains the curve.

The function `addCurve(h)` can be called to add the curve contained in this object to an already existing plot. The input argument `h` is a handle to figure or to the axes containing the already existing curves.

Both the `plot` and `addCurve` function can take an additional extra input argument that specifies the name under which the curve will appear in the legend. If this is not specified, the name in the legend will be the `Name` property of the object. For example, the plot in figure 3 was created by

```
[h, ax] = dummyObject1.plot('Weighted sum detection');
dummyObject2.addCurve(h, 'Best single feature');
```

Finally, the function `calculateArea(upperBound)` calculates the area under the curve between a false alarm rate of 0 and the false alarm rate specified by input argument `upperBound` and returns it. The input argument can be omitted, in which case it is defaulted to 1.

8 Other functions

Some (but not all) of the other available functions are described in this section.

8.1 iEEG_getData function

The `IEEG_getData` function is used to download data from the iEEG.org website.

```
measurement = IEEG_getData(studyName, channel, loginName, loginBin)
```

It has 4 required input arguments and return an instance of the `Measurement` class that contains all relevant information about the recording (seizure start and end, sampling frequency, ...). The required input arguments are in order:

1. `studyName` is a char that must correspond to the name of the iEEG.org study that is to be downloaded
2. `channel` is a double that corresponds to the number of the channel that is to be downloaded
3. `loginName` is a char that corresponds to the User ID that is used to login to the iEEG.org website
4. `loginBin` is a path to the `.bin` file that holds the password that is used to login to the iEEG.org website. Please see the iEEG.org MATLAB Toolbox documentation for more details about what this is.

8.2 MatrixCrossValidation function

The `MatrixCrossValidation` function performs k-Fold cross-validation on a `MatrixDetection` class.

```
[curves_training, curves_validation] = MatrixCrossValidation(...
    matrix, chunkSize_minutes, numFolds)
```

It has 4 required input parameters:

- `matrix`: a `FeatureMatrix` object
- `chunkSize_minutes`: the length in minutes of one chunk
- `numFolds`: the number of folds the data is divided into

It returns two cell arrays `curves_training` and `curves_training` of length `numFolds`.

The data is first divided into chunks of `chunkSize_minutes` length. Seizures are not being divided, so some chunks might be slightly longer than that duration, since if a seizure is at the edge of one chunk, the chunk is extended so that the whole seizure is inside the chunk. The chunks are then shuffled.

The matrix seizure detection algorithm is then trained on `1-1/numFolds` parts of the data and tested on the `1/numFolds` remaining part. This is repeated `numFolds` times, changing the part of the data that is used for training and testing for every iteration. The data from the original `FeatureMatrix` passed as input argument is split into the training and testing part using the `cut` function of the `FeatureMatrix` class (this function is not documented in the user guide).

Two `MatrixDetection` objects are created, one for training and one for testing. The training consists of calculating all the optimal thresholds for all features inside the feature matrix, their weights and the optimal threshold for the `MatrixDetection` object. This is done only on the training data. All these thresholds and weights are then transferred to the second `MatrixDetection` object and the detection is then evaluated using the testing data. The operating characteristic curves of both the training and the testing `MatrixDetection` objects are saved and returned at the end of the function.

Currently, the detection algorithm being cross-validated is the one implemented in MatrixDetection_LinearRegression. Thus, the training object is actually a MatrixDetection_LinearRegression object. To change the algorithm being cross-validated, the relevant section has to be changed inside the function. The testing object is always a MatrixDetection object because no weights are computed for the testing object, they are just copied over from the training object.

A Appendix

A.1 Feature Calculation Formulas

The different feature are calculated based on the formulas described below. The sliding window has a length of N samples and $x[n]$ represents the nth sample of every window.

- **Line Length**

$$\sum_{i=2}^{N} \Big| x[i] - x[i-1] \Big|$$

- **Nonlinear Energy**

$$\frac{1}{N-2} \sum_{i=2}^{N-1} x[i]^2 - x[i-1]x[i+1]$$

- **Full Spectrum Power**

$$\frac{1}{N} \sum_{i=1}^{N} x[i]^2$$

- **Spectral band power in the theta band**: same as full spectrum power but the data is first filtered using a Butterworth bandpass filter with cutoff frequencies 4 Hz and 8 Hz
- **Spectral band power in the alpha band**: same as full spectrum power, the data was first filtered using a Butterworth bandpass filter with cutoff frequencies 8 Hz and 14 Hz
- **Spectral band power in the beta band**: same as full spectrum power, the data was first filtered using a Butterworth bandpass filter with cutoff frequencies 14 Hz and 32 Hz

A.2 Metrics definition

The sensitivity is defined as

$$\text{sensitivity} = \frac{\text{number of seizures detected}}{\text{total number of seizures}}.$$

The specificity is defined as

$$\text{specificity} = \frac{\text{number of interictal feature samples below threshold}}{\text{total number of interictal feature samples}}$$

where an interictal feature sample is a feature sample that does not correspond to a seizure.