EDFanalyzer - user guide

1.Building and running

Prerequisites: Python 2.7, OpenJDK 1.8.0, Swig 3.0.12

Required Python libraries: Jpype 0.6.3, NumPy 1.16.3, Matplotlib 2.2.4 , Pylab, Biosig, pyeeg, xml; (might or might not work with other versions)

In the best case the application should be built by running src/buildit.sh.

If the build was successful, the application can be run in visual mode by:

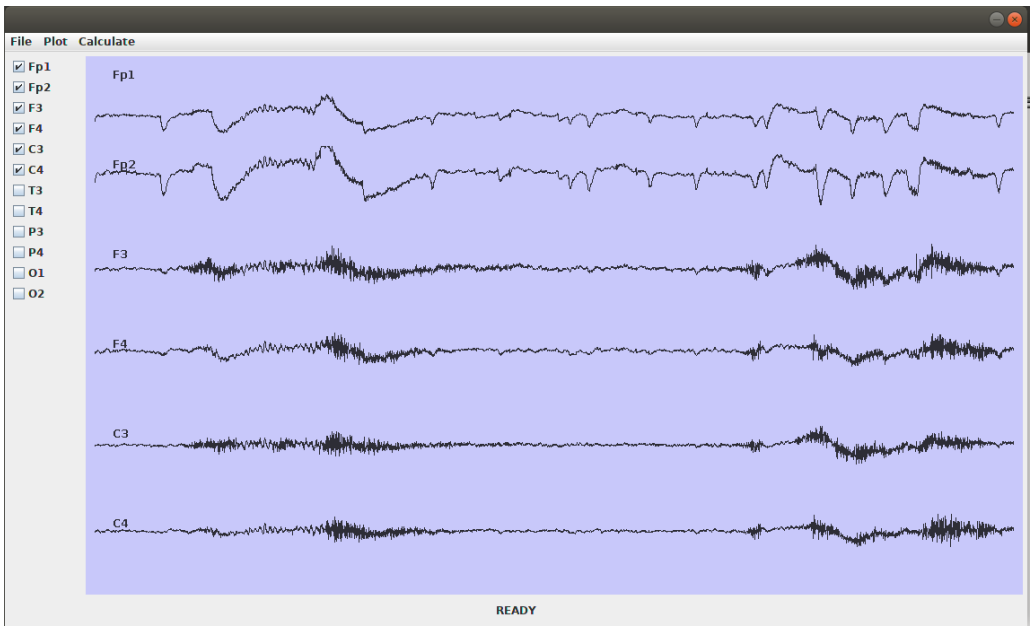
```
python EDFanalyzer.py <edf-file>
```

or can be imported as a module in a python interpreter:

```
import EDFanalyzer
```

2.How to use in visual mode

After starting the application as described in 1. the graphical user interface should appear, something similar to the following :

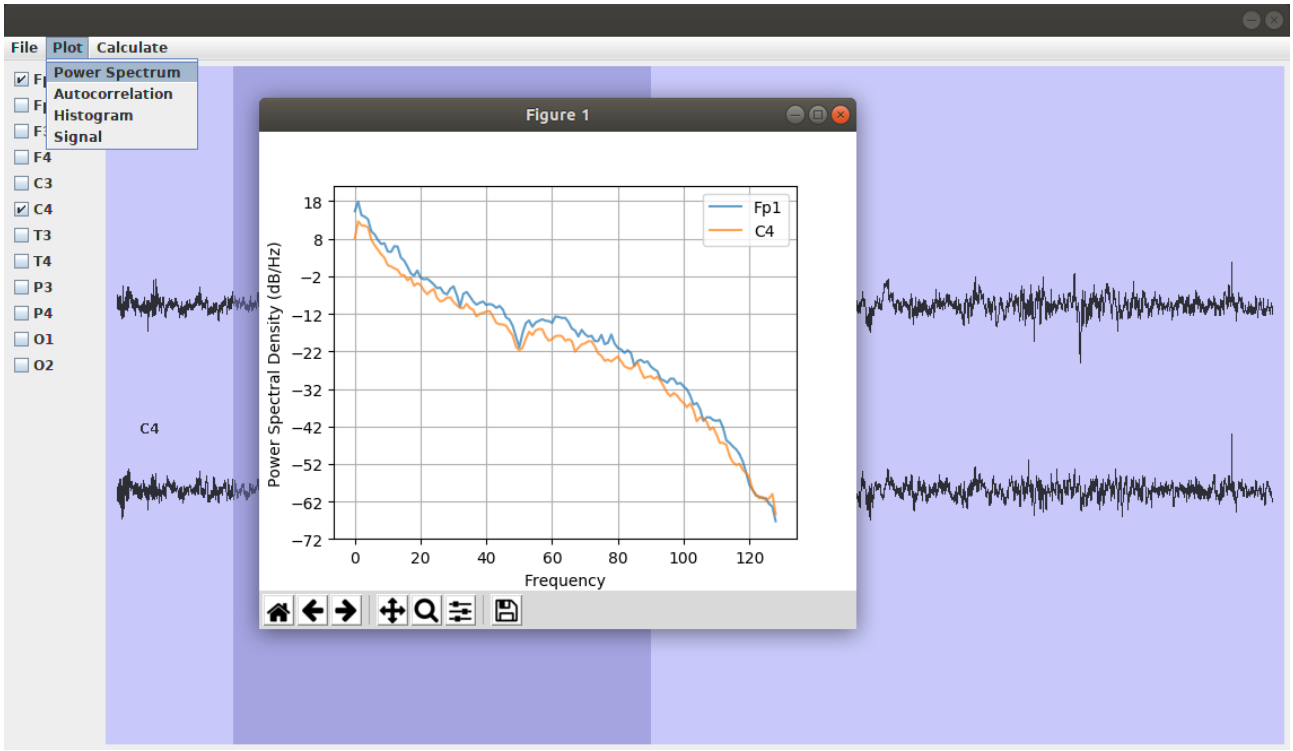


Navigation controls:

- **Moving forward/backward in time – Left-click + Drag**
- **Increasing/decreasing the amplitude of the signals – Mouse Scroll**
- **Zooming in time – Up/Down arrow key**
- **Setting time interval selection – Right-click + Drag**

Creating plots/ calculating measures:

- **Select the channels we are interested in from the left check-list**
- **Select a time interval**
- **Select the desired action from the upper menu**
  (the plot should pop up in a matplotlib window, the calculated measures are printed to the terminal for each selected channel)



In the file menu we can find options to save/load the state of the project into/from an xml file with .proj extension, or another EDF file can be loaded.

3.How to use as python module

In this mode compiling the java files is not necessary, but the swig interfacing is required.

- **Loading data :**

  `Class EEGData`

  `instance methods:`

  `__init__(self, filepath)` - initializes data object given the path

  Parameters : filepath - type: String

  `getSignal(self)` - returns signal data as a list

  `getPath(self)` - returns the path of the data file

  `getLabels(self)` - returns the name of the channels

  `getSampFreq(self)` - returns the sampling frequency

  `reloadData(self, newfile)` - reloads data from another EDF file

  Parameters : newfile - type: String

- **Plotting :**

  Class Plot

  instance methods:

  `__init__(self)` - initializes new plot

  `setData(self,data)` - sets the data to be plotted

  Parameters : data - type: EEGData

  `plotSignal(self,fr,to,activeChannels)` - plots part of the signal

  Parameters : fr,to - type: int, from where to where in samples

  activeChannels - type: list of booleans - channels to plot

  `plotHist(self,fr,to,activeChannels)` - plots histogram

  `plotPsd(self,fr,to,activeChannels)` - plots power spectrum

  `plotAutocorr(self,fr,to,activeChannels)` - plots autocorrelation

- **Calculation of measures :**

  Class Calc

  instance methods:

  `__init__(self)` - initializes new calculation

  `setData(self,data)` - sets the data to be used

  Parameters : data - type: EEGData

  `std(self,fr,to,activeChannels)` - calculates standard deviation

  Parameters : fr,to - type: int, from where to where in samples

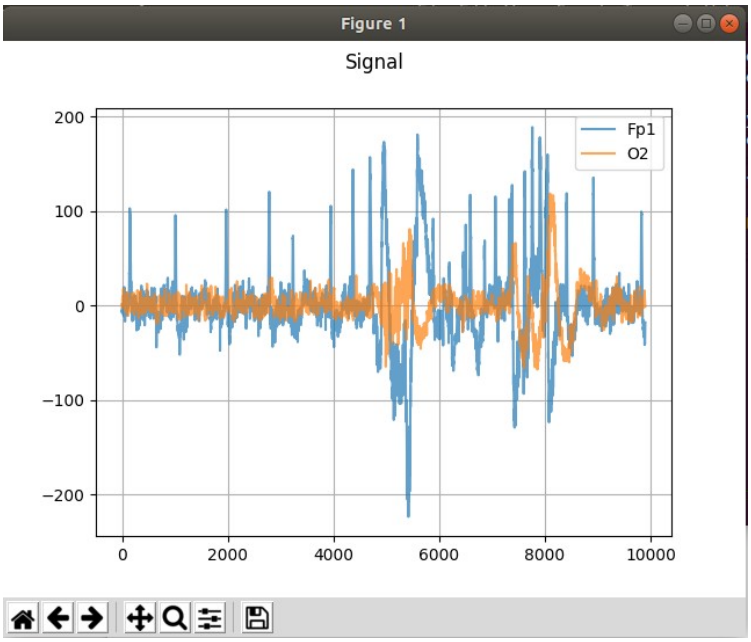
  activeChannels - type: list of booleans - channels to use

  `entrop(self,fr,to,activeChannels)` - plots histogram

- **Example using Ipython interpreter :**

```
IPython 5.5.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from EDFanalyzer import *

In [2]: dat = EEGData('../data/BB0008_SP2.edf')
None

In [3]: dat.getLabels()
Out[3]: ['Fp1', 'Fp2', 'F3', 'F4', 'C3', 'C4', 'T3', 'T4', 'P3', 'P4', 'O1', 'O2']

In [4]: dat.getSampFreq()
Out[4]: 256.0

In [5]: p = Plot()

In [6]: p.setData(dat)

In [7]: p.plotSignal(100,10000,[1,0,0,0,0,0,0,0,0,0,0,1])
```

In this demonstration we loaded an EDF file, then printed the names of the channels and the sampling frequency, then we created a new plot object, set the data which we wanted to plot and then called the plotSignal method, to plot a segment from sample nr. 10 to 10000, and selected the first and the last channels with the list, which produced the following plot:



Credits

Elekes Gyopár – GUI, xml management

Kelemen Szabolcs – Swig interfacing, Python classes

Schneider Bence – Interactive graph plotting, Jpype interfacing, connecting the parts