In this repository you will find 4 R files and 2 CSV data files.

**CSV files:**

* **RP\_data.csv**

This file contains 4 seconds of average signal from 64 channels from a participant performing an RP task (350 trials). It is used as a template RP signal in our simulations.

* **EEG\_values.csv**

This file contains median values for mean, variance and beta values of 64 EEG channels from 4 second epochs of a participant NOT performing an RP task (350 trial).

**R files:**

* **EEG\_simulator.R**

This file uses data from EEG\_values.csv to generate EEG simulated baseline data that mimics EEG signal for 64 channels

* **change\_beta.R**

This function will change the beta value of the power spectrum for any given pair of signal and beta value.

* **modelA\_generator.R**

This file will add random RP-events onto an EEG signal according to model A of the RP

* **modelB\_generator.R**

This file will add random RP-events onto an EEG signal according to model B of the RP

* **forecast\_matrix.R**

This file, given a signal and event-labels will generate a forecast map of the probability of future events occurring given current signal

* **data\_length\_finder.R**

This file will find how much data is needed to minimise the distance between forecast maps (i.e. for forecast maps to converge)

* **Caller\_Forecast.R**

This file will call combinations of EEG\_simulator, modelA\_generator, modelB\_generator and forecast\_matrix.R

**EGG\_simulator.R**

DEPENDENCIES:

* abind
* change\_beta.R

INPUT:

* EEG\_values.csv
* numSubject (number of subjects)
* numChannels (number of EEG channels)
* Srate (sampling rate)
* simulation\_duration (number of simulated seconds)

OUPUT :

* data\_subjects : an numSubject-by-numChannels-by-numSamples 3D matrix containing simulated EEG values for each subjects/channels

MAIN COMPUTATIONS :

No main computations. This code will simply call change\_beta for each channel/subject and store the output into a 3D matrix

**change\_beta.R**

DEPENDENCIES:

* pracma

INPUT:

* signal (a vector signal)
* beta (a single numeric beta value)
* EEG\_values.csv

OUPUT :

* Signal\_scaled (the original signal changed to the beta input and scaled to the EEG signal values)

MAIN COMPUTATIONS :

* The first part changes the beta of a signal:
  + By first generating a filter (full\_filter) to alter the beta value in the frequency domain
  + Second, bringing the signal into the frequency domain (Signal\_freq\_domain) and filtering it
  + By third bringing it back to the time domain (Signal\_beta)
* The second part scales the signal by using the following formula:

**modelA\_generator.R**

INPUT:

* RP\_data.csv
* numEvent\_perMin (number of events per minute)
* Srate (sampling rate)
* event\_spacing (minimum spacing between two events in seconds)
* data\_subjects (3D array from EEG\_simulator.R output)

OUPUT :

* averaged\_data\_subjects (3D array with averaged over channels EEG data per participants with events and event labels)

MAIN COMPUTATIONS :

* Firstly, position of the events is selected, according to input event\_spacing, using the randcomb function in R.
* Second, RPs from RP\_data.csv are convolved to the simulated EEG data at event positions. (using convolve() base function)

**modelB\_generator.R**

INPUT:

* numEvent\_perMin (number of events per minute)
* Srate (sampling rate)
* event\_spacing (minimum spacing between two events in seconds)
* data\_subjects (3D array from EEG\_simulator.R output)

OUPUT :

* averaged\_data\_subjects (3D array with averaged over channels EEG data per participants with events and event labels)

MAIN COMPUTATIONS :

* Firstly, data from 64 channels are collapsed together into a single vector using mean() function
* A temporary vector is then generated and a filter is applied to it (one tenth of a second), in order to then apply diff() to get the slope of the noise.
* Then a threshold is found on this temporary vector, such that the slope is positive and the threshold is crossed only 18 times (with sufficient event\_spacings) by the signal. Events are labelled at these threshold crossings.
* The 4 seconds following an event are then multiplied with a linearly decreasing function that brings the signal to the mean of the entire timeseries.

**forecast\_matrix.m**

INPUT:

* timeBins\_perSecond (number of time bins in each second)
* varBins (total number of possible variable value in the forecast map)
* numFuture (total number of future time points forecasted)
* Srate (sampling rate)
* averaged\_data\_subjects (3D array with averaged over channels EEG data per participants with events and event labels)

OUPUT :

* forecast\_subjects (3D array with varBins-by-numFuture forecast maps per subjects)

MAIN COMPUTATIONS :

* First the data is downsampled (using mean() ) to the input time bin size
* Second a matrix is filled with the count of times a future event happened given the current state of the system (e.g. if 3 times the state had a current signal of 5 and was followed by an event 7 time bins later. Then in the forecast matrix, the 5th row, 7th column would display the value “3”)
* Third, for each possible variable value, the count of the number of times the system is in is filled into a vector. Then the entire row in the forecast is divided by the number of times the system was in this state. (e.g. for the above mentioned example, if the system has been 157 times with the signal value 5, then the value in the 5th row, 7th column would be “3/157”)

**data\_length\_finder.R**

INPUT:

* timeBins\_perSecond (number of time bins in each second)
* varBins (total number of possible variable value in the forecast map)
* numFuture (total number of future time points forecasted)
* Srate (sampling rate)
* averaged\_data\_subjects (3D array with averaged over channels EEG data per participants with events and event labels)
* simulation\_duration (length of simulated data added at each iteration)

OUPUT :

* counter (the number of iterations needed for the maps to converge)

MAIN COMPUTATIONS :

* forecast maps are generated for every ‘simulation\_duration’ addition, until the five previous maps are all close enough together (distance<0.01)
* the distance between two maps are calculated as the sum of the squares of the difference of the elements of the maps