MATÉRIEL ET MÉTHODES

Construction d'une interface cerveau machine

Distinction par mesures electroencephalographiques de la focalisation d'un sujet sur sa main gauche ou droite.

Bruno Belucci et Billerey Agathe
Promotion 136

MATERIEL

Le Laboratoire de Plasticité du Cerveau de l'ESPCI nous a prêté le matériel suivant produit par Neuroelectrics:

- Amplificateur Enobio 8 5G référencé NE003WF
- Casque pour fixer les électrodes de différentes tailles (S, M et L)
- Le logiciel NIC (version 2.0.9) et le toolkit pour Matlab MatNIC (version 4.07)

On a également acheté le matériel suivant de la même entreprise :

- 8 électrodes en gel solide référencé NE028 ainsi que leurs embouts NE035
- 110 mL d'une solution de glycérine conductrice

Tout le matériel peut être acheté et la documentation consultée sur le site de l'entreprise (https://www.neuroelectrics.com) et le laboratoire de Plasticité du Cerveau possède déjà un contact direct avec eux. Matnic a été piloté par Matlab (version R2018a) via la librairie Psychtoolbox (version 3) téléchargeable sur le site http://psychtoolbox.org. L'analyse des résultats a aussi été faite à l'aide de Matlab.

METHODES

Nous avons mis en place plusieurs séries d'expériences sur des groupes de sujets. Elles suivaient toutes le même fonctionnement général (voir shema ci contre) mais differaient dans la méthode d'analyse du signal, la présence où non d'un feed-back, l'exercice demandé au sujet. La première série concerne l'optimisation des paramètres expérimentaux et analytiques généraux. La seconde et troisième utilisent ces paramètres pour distinguer deux états chez un sujet.

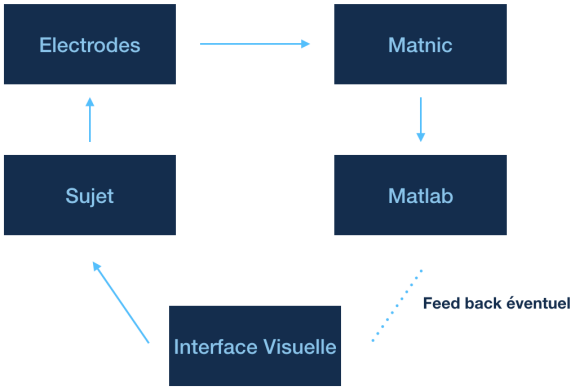


Figure 1: Fonctionnement général.

Montage général

Les expériences doivent être réalisées dans une salle calme et silencieuse pour ne pas perturber le sujet et pour qu'il puisse se concentrer. On place les embouts des électrodes dans les trous du casque adapté à la taille du sujet, puis on trempe les électrodes en gel solide dans la solution de glycérine avant de les placer directement en contact avec le scalp du sujet. Si nécessaire on utilise une tige rigide pour écarter les cheveux du sujet et s'assurer que l'électrode est bien en contact. Ensuite, on connecte les électrodes à l'amplificateur et l'electrode de référence à l'oreille du sujet. L'amplificateur est fixé à l'arrière du casque sur une zone adhesive. La communication entre l'amplificateur et l'ordinateur peut se faire par Wi-Fi ou Bluetooth. Le montage du casque peut être observé dans la Figure 2.

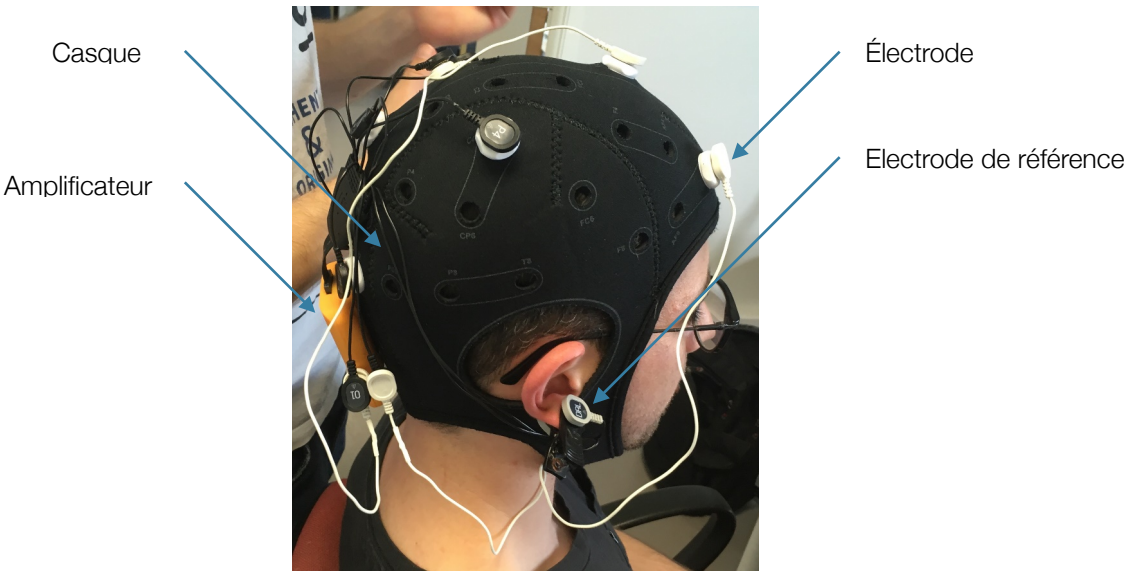


Figure 2 : Casque d'EEG monté.

PSE-INTERFACE CERVEAU MACHINE - MATERIEL ET METHODES

Les sujets étaient tous des volontaires issues de la promotion 136 de l'ESPCI (21- 22 ans). L'expérience débute par l'explication générale au sujet du déroulement de l'exercice. Il lui est ensuite demandé de réaliser deux tâches dans un certain protocole qui sera détaillé pour chaque série d'expérience : « se concentrer » sur sa main gauche ou à sa main droite. Chaque sujet a la liberté de choisir sa stratégie pour réaliser cette tâche, la seule condition est qu'il ne doit pas bouger sa main, pour ne pas polluer le signal avec des artefacts dus au mouvement des muscles. Nous avons fournit aux sujtes les exemples de stratégie suivant : imaginer ta main en mouvement, ressens la température à la surface de la peau de ta main, ressens la texture du matériaux sur lequel est posé ta main. Pendant l'exercice, le sujet est assis devant l'ordinateur. Une série de commandes sont affichées sur l'écran et les mesures sont faites directement par le même ordinateur. A la fin de l'expérience on démonte le montage et on pose une série de questions afin de connaître la startégie du sujet, son resssenti, la pénibilité et comment on pourrait améliorer l'expérience par la suite. Les avis étaient surtout receuillis sur le temps de l'expérience, les stimulus visuels et l'impression de reussite. Notons également que l'amplificateur a été réglé via le logiciel NIC pour qu'il filtre le signal au dessus de 50 Hz.

Pour chaque expérience nous avons extrait du signal de chaque électrode la densité spectrale de puissance (PDS) pour pouvoir l'analyser et l'utiliser dans les jeux. Pour cela nous avons utilisé la fonction « pwelch » de Matlab avec une fenêtre de 500 ms et un recouvrement de 250 ms, les fréquences analysées changent selon l'expérience ainsi que l'intervalle dans lequel la fonction est utilisée.

Nous allons à présent détailler les spécificités de chaque série d'expériences. Les codes utilisés sont disponibles en annexe.

PREMIÈRE SÉRIE

SÉRIE D'EXPERIENCE	OBJECTIF	NOMBRE DE SUJET	BANDE FRÉQUENTIELLE	DISPOSITION DES ÉLECTRODES
1	Recherche du meilleur classificateur ainsi que des zones cérébrales et fréquentielles discriminantes.	15	3 à 30 Hz	C3 C4 FZ FP1 FP2 PZ O1 O2 (voir Figure 3)

La protocole de la première série d'expériences possède 2 phases. Dans la première, dite « d'apprentissage », on demande au sujet de se concentrer sur sa main droite 3 fois pendant 5 secondes, puis de même avec sa main gauche. Ensuite, dans la seconde phase on répète chaque commande aléatoirement 6 fois pendant 5 secondes. L'analyse est faite après l'enregistrement, donc offline, et le sujet n'a aucun feedback en temps réel. Une capture d'écran de l'interface visuelle pendant l'expérience est présentée en Figure 4.

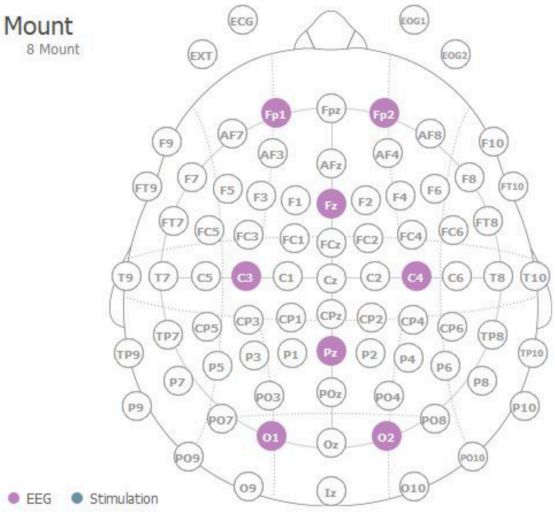


Figure 3 : Disposition des électrodes- Série 1

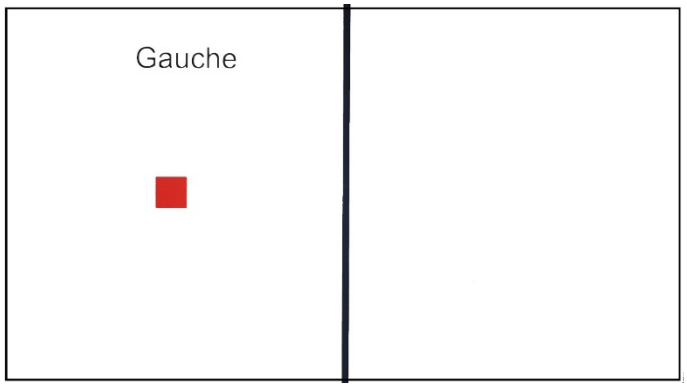
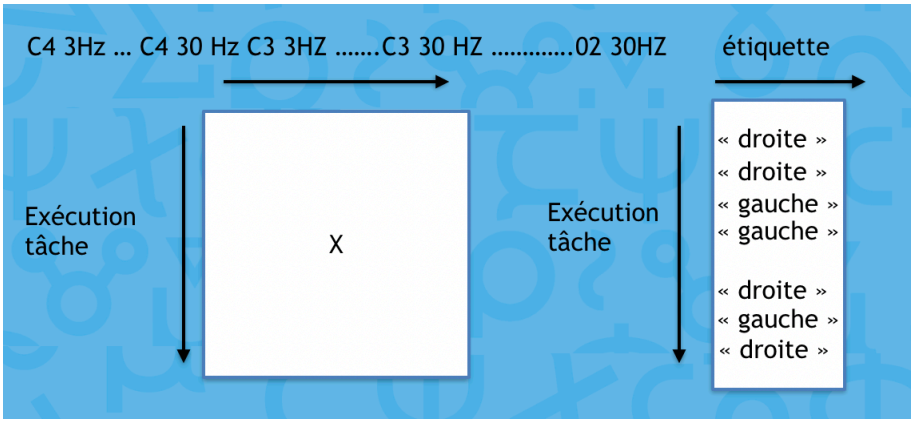


Figure 4 : Capture d'écran lors de la série 1. La commande est affichée en haut à gauche ou à droite et le carré rouge clignote.

Nous avons ensuite analysé les données avec les méthodes suivantes. La densité spectrale est calculée dans l'intervalle de 5 secondes où le sujet est censé réaliser sa tâche, entre les fréquences de 3 à 30 Hz sur chaque électrode. On peut donc construire la matrice de données ainsi que son vecteur étiquette suivante:



Chaque colonne est l'évaluation d'un prédicteur par exemples. On obtient donc 224 prédicteurs (28 fréquences x 8 électrodes). On a alors testé plusieurs classificateurs (voir figure 5) grâce à la Toolbox Classification Learner de Matlab et généré plusieurs modèles de prédictions (classificateur entrainé avce les données d'un sujet). Pour chaque sujet on a sélectionné les modèles de prédiction à la précision (« Score ») la plus élevée pour trouver les prédicteurs et les classificateurs les plus discriminants. Notons que nous avons activé l'option Analyse en Composante principale de la Toolbox pour trouver les prédicteurs les plus discriminants par classificateur.

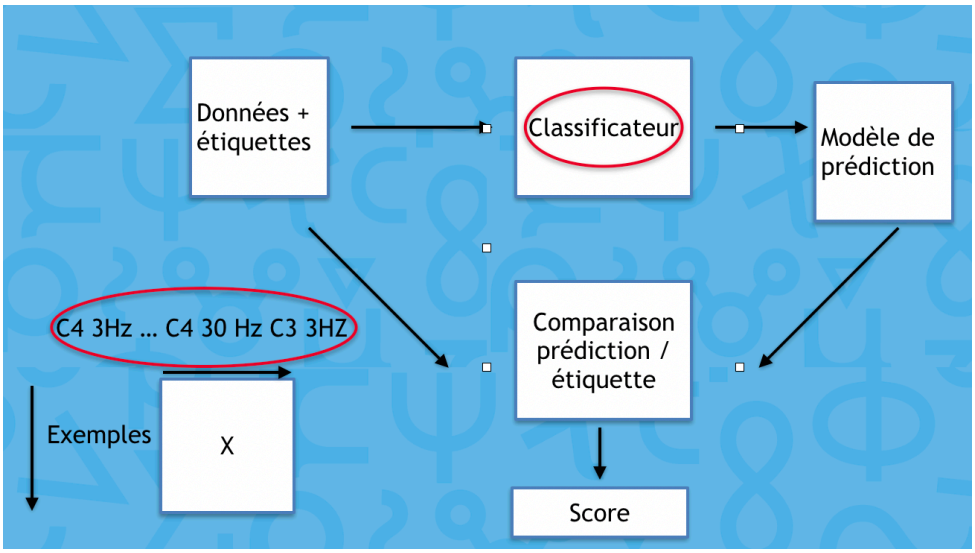


Figure 5: Liste des classificateurs testés

Classificateur Testés	Documentation
Fine Tree Medium Tree Coarse Tree	https://fr.mathworks.com/help/stats/classification-trees.html?s_tid=CRUX_lftnav
Linear Discriminant Quadratic Discriminant	https://fr.mathworks.com/help/stats/classification-discriminant-analysis.html?s_tid=CRUX_lftnav
Logistic Regression	https://fr.mathworks.com/help/stats/generalized-linear-regression-1.html
Linear SVM Cubic SVM Quadratic SVM Fine Gaussian SVM Medium Gaussian SVM Coarse Gaussian	https://fr.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html
Fine KNN Medium KNN Coarse KNN Cosine KNN Weighted KNN Cubic KNN	https://fr.mathworks.com/help/stats/classification-using-nearest-neighbors.html?searchHighlight=nearest%20neighbors&s_tid=doc_srchtitle
Ensemble boosted trees Ensemble bagged trees Ensemble subspace discriminant Ensemble subspace knn Ensemble boosted trees	https://fr.mathworks.com/help/stats/classification-ensembles.html?s_tid=CRUX_lftnav

DEUXIÈME SÉRIE

SÉRIE D'EXPERIENCE	OBJECTIF	NOMBRE DE SUJET	BANDE FRÉQUENTIELLE	DISPOSITION DES ÉLECTRODES
2	Mettre en place le jeu avec « feed_back » en temps réel.	2	12 à 16HZ	C3 P3 CP5 CP1 C4 P4 CP2 CP6 Voir figure 6

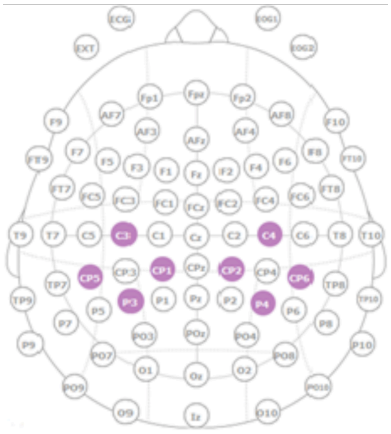


Figure 6 : Disposition des électrodes- Série2

Pour la deuxième expérience on a codé un jeu simple où le sujet a pour but de déplacer une barre à droite ou à gauche de l'écran pour capturer un carré qui tombe. Si la barre est bien placée quand le carré arrive en bas, le score est incrémenté d'un point, sinon il reste inchangé. Pour jouer il faut d'abord initialiser le jeu (le personnaliser au sujet) en enregistrant le signal EEG du sujet via le protocole de la première expérience et fournir les données sauvegardées par le logiciel NIC au script Matlab d'initialisation du jeu. Le script test les meilleurs classificateurs obtenues d'après notre analyse de la première expérience et entraine la meilleur avec les données de l'initialisation. La fonction de prédiction ainsi créée est utilisée dans le code du jeu et le sujet doit utiliser la même stratégie qu'il a employé pendant la première expérience pour déplacer la barre. A chaque chute du carré, le sujet se concentre sur la main du côté où il veut déplacer la barre. En temps réeel le modèle fait une prédiction sur la tâche qu'il réalise et déplace la barre du côté de la prédiction. Une capture d'écran du jeu est montrée dans la Figure 7.

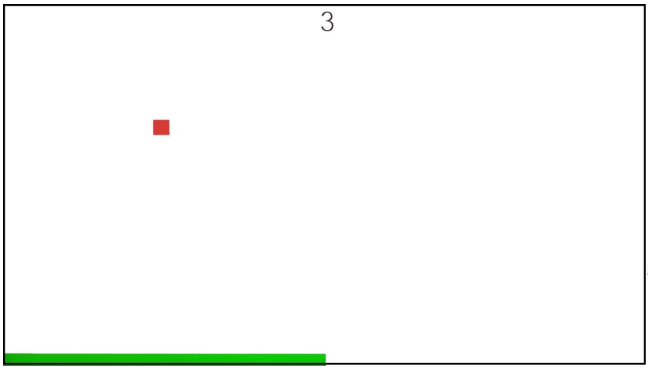


Figure 7 : Jeu vidéo en utilisant la classification. On voit le score en haut, le carré rouge qui tombe et la barre verte qui peut être à gauche ou à droite de l'écran.

Il n'y a pas eu d'analyses spécifiques pour cette expérience car on trouvé que l'expérience a échoué pour les 2 premiers sujets et on a décidé de ne pas l'étendre. En particulier, l'initialisation aurait du être au moins 100 fois plus longue pour que ca fonctionne ce qui n'était pas envisageable. Les autres raisons de l'échec sont évoqués dans les autres livrables du PSE.

TROISIÈME SÉRIE

SÉRIE D'EXPERIENCE	OBJECTIF	NOMBRE DE SUJET	BANDE FRÉQUENTIELLE	DISPOSITION DES ÉLECTRODES
3	Jouer avce le ratio de l'activité à droite et à gauche du cerveau.	13	13 à 15HZ	C3 P3 CP5 CP1 C4 P4 CP2 CP6 Voir figure 6

On change de stratégie pour la troisième expérience et on fait une interface cerveau machine sans passer pour un algorithme de classification. Cette expérience possède aussi deux phases. Pendant la première phase on demande au sujet d'être au repos, cette-à-dire, de ne se concentrer ni à sa main gauche ni à sa main droite. On enregistre alors le ratio entre la moyenne de la densité spectrale de puissance entre les électrodes positionnes à droite du cerveau et les électrodes positionnes à gauche :

$$Ratio = \frac{\sum PDS\,droite}{\sum PDS\,gauche}$$

La densité spectrale est encore une fois calculée à l'aide de la fonction « pwelch » dans un intervalle de 3 secondes et entre les fréquences de 13 à 15 Hz. On calcule un ratio de base en faisant la moyenne des ratios obtenus pendant 30 secondes (10 ratios mesures), on garde aussi l'écart type du ratio de base. Ensuite, on passe à la deuxième phase où on demande au sujet de se concentrer 5 fois sur sa main gauche et 5 fois sur sa main droite, de façon alternée, chaque essai a une durée de 20 secondes. Pendant la deuxième phase le sujet a un feedback à travers un nombre et une barre rouge que se déplace vers la gauche ou vers la droite. Le nombre affiché correspond au rapport entre le ratio du sujet pendant les dernières 3 secondes et le ratio de base enregistré pendant la première phase. La barre se déplace vers la droite ou vers la gauche si le ratio du sujet est plus grand que son ratio de base plus un écart type ou plus petit que son ratio de base moins un écart type, respectivement. Le sujet doit alors essayer de faire monter le nombre au-dessus de 1 quand on lui demande de se concentrer sur sa main droite pour déplacer la barre à droite et il doit faire l'inverse quand on lui demande de se concentrer sur sa main gauche. Une capture d'écran de cette expérience est montrée dans la Figure 8.

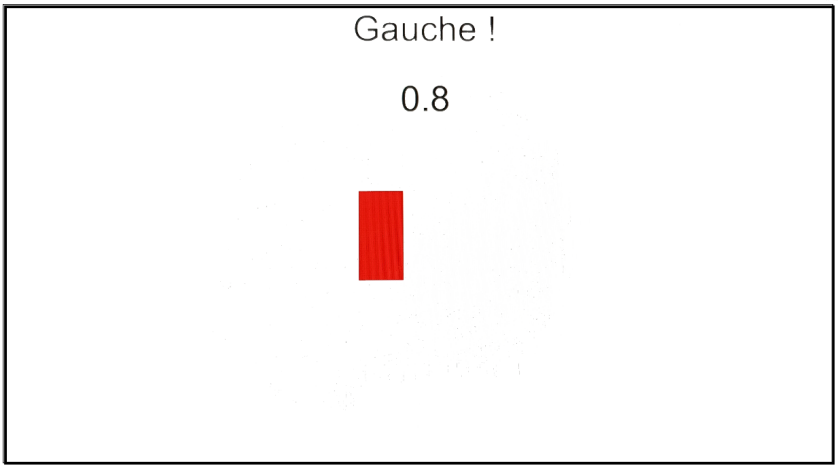


Figure 8 : Capture d'écran de la troisième expérience. On observe la commande en haut, le feedback (ratio/ ratio de base) juste en bas et la barre rouge qui peut se déplacer à droite et à gauche.

On enregistre la matrice « rt » dans Matlab (voir codes en annexe) à la fin de l'expérience qui contient le ratio/ratio de base (ou seulement le ratio de base pendant les 30 premières secondes) dans la première colonne, un marqueur pour identifier chaque intervalle pendant l'expérience dans la deuxième colonne et le temps total de l'expérience dans la troisième colonne.

L'analyse des données a été faite à travers la matrice « rt » obtenue pour chaque sujet. Pour pouvoir rassembler tous les sujets on a réalisé une normalisation des données dans chaque intervalle de réalisation de tâche. Dans l'intervalle où il se concentrait à sa main gauche on soustrait la moyenne du ratio mesuré pendant cet intervalle et on divise par l'écart type, pour obtenir idéalement une distribution normale centré à 0. Dans l'intervalle où la tâche « droite » est réalisée on fait la même normalisation que sur l'intervalle à gauche immédiatement avant.

$$X_{i\ gauche} = \frac{X_{i\ gauche} - \bar{X}_{i\ gauche}}{\sigma_{i\ gauche}} \quad X_{i+1\ droite} = \frac{X_{i+1\ droite} - \bar{X}_{i\ gauche}}{\sigma_{i\ gauche}}$$

Le code qui nous a permit de faire la normalisation et d'afficher les données est aussi dans l'annexe. Pour tester notre hypothèse « le ratio permet de discriminer l'état de gauche de l'état de droite » on a réalisé un test de permutation. On permute de façon aléatoire toutes les étiquettes des données et on calcule la valeur obtenue pour la moyenne des données étiquetées à droite moins la valeur obtenue pour la moyenne des données étiquetées à gauche. On réalise cette opération plusieurs fois, dans notre cas 10000 fois, et on obtient une distribution des valeurs. On construit un histogramme et on compare l'histogramme avec la valeur réelle qu'on obtient avec les étiquettes mesurées. On s'attend à obtenir un histogramme que ressemble à une distribution gaussienne et à ce que notre valeur réelle obtenue avec les bonnes étiquettes soit significative en calculant une p-valeur. L'avantage du test de permutation par rapport à un test t-student est qu'on n'a pas

besoin de supposer que les données ont été obtenues de façon indépendante ce qui n'est pas le cas pour notre expérience. Le code du test de permutation est aussi montré dans l'annexe.

ANNEXE 1 - PROGRAMME SERIE 1

```
% Clear the workspace and the screen
sca;
close all;

[ret,outlet]=MatNICMarkerConnectLSL('mymarker');
%1-start 2-begin right 3-end right 4-begin left 4-end left
6-END

%MatNICfunctions
% [ret,status,socket]=MatNICConnect('localhost');
% [ret,
version,n_channels,deviceSettings]=MatNICSetUp(socket);
% [ret,outlet]=MatNICMarkerConnectLSL('mymarker');
% [ret]=MatNICMarkerSendLSL(0,outlet);

% Here we call some default settings for setting up
Psychtoolbox
PsychDefaultSetup(2);

% Get the screen numbers. This gives us a number for each of
the screens
% attached to our computer.
screens = Screen('Screens');

% Draw we select the maximum of these numbers. So in a
situation where we
% have two screens attached to our monitor we will draw to
the external
% screen. When only one screen is attached to the monitor we
will draw to
% this.
screenNumber = max(screens);

% Define black and white (white will be 1 and black 0).
white = WhiteIndex(screenNumber);
black = BlackIndex(screenNumber);

% Open an on screen window and color it black
```

```
[window, windowRect] = PsychImaging('OpenWindow',
screenNumber, white);

% Get the size of the on screen window in pixels
[screenXpixels, screenYpixels] = Screen('WindowSize',
window);

% Get the centre coordinate of the window in pixels
[xCenter, yCenter] = RectCenter(windowRect);

% Enable alpha blending for anti-aliasing
Screen('BlendFunction', window, GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA);

% Query the frame duration
ifi = Screen('GetFlipInterval', window);
waitframes=1;
% Get the nominal framerate of the monitor. For this simple
timer we are
% going to change the counterdown number every second. This
means we
% present each number for "frameRate" amount of frames. This
is because
% "framerate" amount of frames is equal to one second. Note:
this is only
% for a very simple timer to demonstarte the principle. You
can make more
% accurate sub-second timers based on this.
nominalFrameRate = Screen('NominalFrameRate', window);

%sine wave parameters
amplitude = 1;
frequency = 1;
angFreq = 2 * pi * frequency;
startPhase = 0;

%dotparameters
maxdotsize=100;
dotLeftX=screenXpixels/4;
dotRightX=screenXpixels*3/4;
dotLeftY=screenYpixels/2;
```

```
dotRightY=screenYpixels/2;
dotcolor=[1 0 0];

%divisionparameters
baseRect=[0 0 20 screenYpixels];
divcolor=[0 0 0];
centereddiv = CenterRectOnPointd(baseRect, xCenter,
yCenter);



Screen('TextSize', window, 80);
Screen('TextFont', window, 'Arial');
DrawFormattedText(window, 'Appuyer sur une touche pour
commencer','center','center',black);
Screen('Flip',window);
KbStrokeWait;
[ret]=MatNICMarkerSendLSL(1,outlet);
DrawFormattedText(window, 'Apprentissage !\n
Droite !','center','center',black);
Screen('Flip',window);
WaitSecs(1);

j=0;
while j<3
    i=3;
    while i>0
        number=num2str(i);
        DrawFormattedText(window,
number,'center','center',black);
        vbl=Screen('Flip',window);
        WaitSecs(1);
        i=i-1;
    end
    time=0;
    [ret]=MatNICMarkerSendLSL(2,outlet);
    while time<5
        DrawFormattedText(window,
'Droite',screenXpixels*3/4-100,200,black);
        Screen('FillRect', window, divcolor, centereddiv);
```

```
scalefactor=abs(amplitude*sin(angFreq*time+startPhase));
        dotsize=maxdotsize-30*scalefactor;
        Screen('DrawDots',window,
[dotRightX;dotRightY],dotsize,dotcolor);
        vbl=Screen('Flip',window,vbl+(waitframes-0.5)*ifi);
        time=time+ifi;
    end
    [ret]=MatNICMarkerSendLSL(3,outlet);
    j=j+1;
end


DrawFormattedText(window, 'Appuyer sur une touche pour
continuer...','center','center',black);
Screen('Flip',window);
KbStrokeWait;
DrawFormattedText(window, 'Apprentissage !\n
Gauche !','center','center',black);
Screen('Flip',window);
WaitSecs(1);

j=0;
while j<3
    i=3;
    while i>0
        number=num2str(i);
        DrawFormattedText(window,
number,'center','center',black);
        vbl=Screen('Flip',window);
        WaitSecs(1);
        i=i-1;
    end
    [ret]=MatNICMarkerSendLSL(4,outlet);
    time=0;
    while time<5
        DrawFormattedText(window, 'Gauche',screenXpixels/
4-100,200,black);
        Screen('FillRect', window, divcolor, centereddiv);

scalefactor=abs(amplitude*sin(angFreq*time+startPhase));
```

```
        dotsize=maxdotsize-30*scalefactor;
        Screen('DrawDots',window,
[dotLeftX;dotLeftY],dotsize,dotcolor);
        vbl=Screen('Flip',window,vbl+(waitframes-0.5)*ifi);
        time=time+ifi;
    end
    [ret]=MatNICMarkerSendLSL(5,outlet);
    j=j+1;
end


DrawFormattedText(window, 'Appuyer sur une touche pour
continuer...','center','center',black);
Screen('Flip',window);
KbStrokeWait;
DrawFormattedText(window, 'Test !','center','center',black);
Screen('Flip',window);
WaitSecs(1);

j=0;
while j<6
    i=3;
    while i>0
        number=num2str(i);
        DrawFormattedText(window,
number,'center','center',black);
        vbl=Screen('Flip',window);
        WaitSecs(1);
        i=i-1;
    end
    r=rand;
    if(r<0.5)
        time=0;
        [ret]=MatNICMarkerSendLSL(2,outlet);
        while time<5
            DrawFormattedText(window,
'Droite',screenXpixels*3/4-100,200,black);
            Screen('FillRect', window, divcolor,
centereddiv);

scalefactor=abs(amplitude*sin(angFreq*time+startPhase));
```

```
            dotsize=maxdotsize-30*scalefactor;
            Screen('DrawDots',window,
[dotRightX;dotRightY],dotsize,dotcolor);
            vbl=Screen('Flip',window,vbl+
(waitframes-0.5)*ifi);
            time=time+ifi;
        end
        [ret]=MatNICMarkerSendLSL(3,outlet);
    else
        time=0;
        [ret]=MatNICMarkerSendLSL(4,outlet);
        while time<5
            DrawFormattedText(window,
'Gauche',screenXpixels/4-100,200,black);
            Screen('FillRect', window, divcolor,
centereddiv);

scalefactor=abs(amplitude*sin(angFreq*time+startPhase));
            dotsize=maxdotsize-30*scalefactor;
            Screen('DrawDots',window,
[dotLeftX;dotLeftY],dotsize,dotcolor);
            vbl=Screen('Flip',window,vbl+
(waitframes-0.5)*ifi);
            time=time+ifi;
        end
        [ret]=MatNICMarkerSendLSL(5,outlet);
    end
    j=j+1;
end
DrawFormattedText(window, 'Fin !\nAppuyer sur une touche
pour fermer...','center','center',black);
Screen('Flip',window);
KbStrokeWait;
[ret]=MatNICMarkerSendLSL(6,outlet);

sca;
```

ANNEXE 2 - PROGRAMME SERIE 2

Jeu

```
%%% pwelch config
f_min=3;
f_max=30;
c_welch_window = 500;
c_welch_recover =250;


%Configure Psychotoolbox
[screens,screenNumber,white,black,window,windowRect,screenXp
ixels,screenYpixels,xCenter,yCenter,ifi,waitframes,nominalFr
ameRate] = configpsychotoolbox();

%sine wave parameters
amplitude = 1;
frequency = 1;
angFreq = 2 * pi * frequency;
startPhase = 0;

%scrollbar
scrollformat=[0 0 screenXpixels/2 100];
scrollcolor=[0 1 0];
scrollleft=CenterRectOnPointd(scrollformat,screenXpixels/
4,screenYpixels);
scrollright=CenterRectOnPointd(scrollformat,screenXpixels*3/
4,screenYpixels);

%squares
square=[0 0 50 50];
squarecolor=[1 0 0];

%velocity
v=(screenYpixels-40)/6;

%Init. server
sampling_rate  = 500;   % Sampling rate EEG [SPS]
period = 3;
n_channels = 8;
```

```
bytes_per_double=8;
target_samples = sampling_rate * period;
n_samples = 0;
eeg_set = zeros(target_samples, n_channels);
connect=tcpip('localhost',30000,'NetworkRole','client');
connect.OutputBufferSize=bytes_per_double*numel(eeg_set);
connect.InputBufferSize=bytes_per_double*numel(eeg_set);
timestamp_set  = zeros(target_samples, 1);
fopen(connect);
disp('Connected!');
%%

Screen('TextSize', window, 80);
Screen('TextFont', window, 'Arial');
DrawFormattedText(window, 'Appuyer sur une touche pour
commencer','center','center',black);
Screen('Flip',window);
KbStrokeWait;
position=scrollleft;
pos=1;
Screen('FillRect',window,scrollcolor,position);
Screen('Flip',window);
exit=0;
score=0;
%%
eeg_pwelch=zeros(1,(f_max-f_min)*n_channels);

while(1)
    r=rand;
    y=0;
    time=0;
    time2=0;
    dt=0;
    if(r<0.5)
        while(y<screenYpixels-40)
            y=v*time;
            starttime=now*24*3600;

squareleft=CenterRectOnPointd(square,screenXpixels/4,y);

Screen('FillRect',window,squarecolor,squareleft);
```

```
            Screen('FillRect',window,scrollcolor,position);
            sscore=num2str(score);
            DrawFormattedText(window, sscore,'center',
100,black);
            Screen('Flip',window);
            if(time2>3)
                fwrite(connect,1);
                eeg_set=fread(connect,[sampling_rate*period
n_channels],'double');
                eeg_set=detrend(eeg_set);

eeg_set=pwelch(eeg_set,c_welch_window,c_welch_recover,
f_min:f_max);
                eeg_pwelch=[eeg_set(:,1)' eeg_set(:,2)'
eeg_set(:,3)' eeg_set(:,4)' eeg_set(:,5)' eeg_set(:,6)'
eeg_set(:,7)' eeg_set(:,8)'];
                pos = trainedModel.predictFcn(eeg_pwelch);
                time2=0;
                if pos==1
                    position=scrollright;
                else
                    position=scrollleft;
                end
            end
            dt=(now*24*3600-starttime);
            time=time + dt;
            time2=time2 + dt;
        end

    end
    if(r>0.5)
        while(y<screenYpixels-40)
            y=v*time;
            starttime=now*24*3600;

squareright=CenterRectOnPointd(square,screenXpixels*3/4,y);

Screen('FillRect',window,squarecolor,squareright);
            Screen('FillRect',window,scrollcolor,position);
            sscore=num2str(score);
```

```
            DrawFormattedText(window, sscore,'center',
100,black);
            Screen('Flip',window);
            if(time2>3)
                fwrite(connect,1);
                eeg_set=fread(connect,[sampling_rate*period
n_channels],'double');
                eeg_set=detrend(eeg_set);

eeg_set=pwelch(eeg_set,c_welch_window,c_welch_recover,
f_min:f_max);
                eeg_pwelch=[eeg_set(:,1)' eeg_set(:,2)'
eeg_set(:,3)' eeg_set(:,4)' eeg_set(:,5)' eeg_set(:,6)'
eeg_set(:,7)' eeg_set(:,8)'];
                pos = trainedModel.predictFcn(eeg_pwelch);
                if pos==1
                    position=scrollright;
                else
                    position=scrollleft;
                end
                time2=0;
            end
            dt= (now*24*3600-starttime);
            time=time +dt;
            time2=time2 + dt;
        end

    end
    if exit==1
        break;
    end
    if(r<0.5&&pos==2)
        score=score+1;
    end
    if(r>0.5&&pos==1)
        score=score+1;
    end
end

sca;
```

Initialisation du Jeu

```
clear all

%% Objectif:

% prend data, teste plusieurs modèle et choisit celui qui a
la meilleure
% accuracy avec la pca et retourne une fonction qui permet
de faire des
% prédictions sur nouvelles data

%Pour faire des prédictions avec le modèle adapté au joueur
utiliser
%fonction_prediction (data in shape)

%% Variables à rentrer

load('Bruno2.easy');
 X=Bruno2;
Variance_pca=99
etat1= 2;
etat2= 4;
timeexp=136000;  %%%durée de l'expérience
l= 0 ;   %nombre d'échantillons dans  l'état 1 attention non
dispo dans le fichier info à noter à chaque expérience
v= 0  ;  %nombre d'échantillons dans l'état 2
f_min=3;
f_max=30;
g=8;  %nombre d'electrodes
%X=données de l'expérience étudiée
c_welch_window = 500;
c_welch_recover =250;



%% mise en forme des données


 for i=1:timeexp ;
```

```
      if X(i,12)==etat1
          l=l+1
      end
      if X(i,12)==etat2
          v=v+1
      end
 end



Accuracy=0;
M=zeros(l+v,(f_max-f_min+1)*g);
s=0;
b=0;
  for p=1:timeexp ;
    %timeexpenunittimesample

       if X(p,12)==etat1  && s<l ;
           s=s+1;
           for i=1:g ;
              eeg_trace = detrend(X(p+500:p+1500, i ));
              rythm_power = pwelch(eeg_trace,
c_welch_window,c_welch_recover, f_min:f_max);
              for f=f_min:f_max ;
                  M(s,(f_max-f_min+1)*(i-1)+f-
f_min+1)=rythm_power(f-f_min+1) ;
              end
           end
        end

       if X(p,12)== etat2  && b<v ;  %début de tache etat 2
           b=b+1;
           for i=1:g ;
              eeg_trace = detrend(X(p+500:p+1500, i ));
              rythm_power = pwelch(eeg_trace,
c_welch_window,c_welch_recover, f_min:f_max);
              for f=f_min:f_max;
                    M(l+b ,(f_max-f_min+1)*(i-1)+f-
f_min+1)=rythm_power(f-f_min+1);
              end
           end
```

```
      end


  end

 %préparationpourclassification

 P=zeros(1, l+v);

 for i=1:l;
    P(i)=1;
 end
for i=1:v;
    P(l+i)=2;
end


 T=[M P'];

%% convertit les données et applique une pca

trainingData=T;
inputTable = array2table(trainingData, 'VariableNames',
{'column_1', 'column_2', 'column_3', 'column_4', 'column_5',
'column_6', 'column_7', 'column_8', 'column_9', 'column_10',
'column_11', 'column_12', 'column_13', 'column_14',
'column_15', 'column_16', 'column_17', 'column_18',
'column_19', 'column_20', 'column_21', 'column_22',
'column_23', 'column_24', 'column_25', 'column_26',
'column_27', 'column_28', 'column_29', 'column_30',
'column_31', 'column_32', 'column_33', 'column_34',
'column_35', 'column_36', 'column_37', 'column_38',
'column_39', 'column_40', 'column_41', 'column_42',
'column_43', 'column_44', 'column_45', 'column_46',
'column_47', 'column_48', 'column_49', 'column_50',
'column_51', 'column_52', 'column_53', 'column_54',
'column_55', 'column_56', 'column_57', 'column_58',
'column_59', 'column_60', 'column_61', 'column_62',
'column_63', 'column_64', 'column_65', 'column_66',
'column_67', 'column_68', 'column_69', 'column_70',
'column_71', 'column_72', 'column_73', 'column_74',
```

```
'column_75', 'column_76', 'column_77', 'column_78',
'column_79', 'column_80', 'column_81', 'column_82',
'column_83', 'column_84', 'column_85', 'column_86',
'column_87', 'column_88', 'column_89', 'column_90',
'column_91', 'column_92', 'column_93', 'column_94',
'column_95', 'column_96', 'column_97', 'column_98',
'column_99', 'column_100', 'column_101', 'column_102',
'column_103', 'column_104', 'column_105', 'column_106',
'column_107', 'column_108', 'column_109', 'column_110',
'column_111', 'column_112', 'column_113', 'column_114',
'column_115', 'column_116', 'column_117', 'column_118',
'column_119', 'column_120', 'column_121', 'column_122',
'column_123', 'column_124', 'column_125', 'column_126',
'column_127', 'column_128', 'column_129', 'column_130',
'column_131', 'column_132', 'column_133', 'column_134',
'column_135', 'column_136', 'column_137', 'column_138',
'column_139', 'column_140', 'column_141', 'column_142',
'column_143', 'column_144', 'column_145', 'column_146',
'column_147', 'column_148', 'column_149', 'column_150',
'column_151', 'column_152', 'column_153', 'column_154',
'column_155', 'column_156', 'column_157', 'column_158',
'column_159', 'column_160', 'column_161', 'column_162',
'column_163', 'column_164', 'column_165', 'column_166',
'column_167', 'column_168', 'column_169', 'column_170',
'column_171', 'column_172', 'column_173', 'column_174',
'column_175', 'column_176', 'column_177', 'column_178',
'column_179', 'column_180', 'column_181', 'column_182',
'column_183', 'column_184', 'column_185', 'column_186',
'column_187', 'column_188', 'column_189', 'column_190',
'column_191', 'column_192', 'column_193', 'column_194',
'column_195', 'column_196', 'column_197', 'column_198',
'column_199', 'column_200', 'column_201', 'column_202',
'column_203', 'column_204', 'column_205', 'column_206',
'column_207', 'column_208', 'column_209', 'column_210',
'column_211', 'column_212', 'column_213', 'column_214',
'column_215', 'column_216', 'column_217', 'column_218',
'column_219', 'column_220', 'column_221', 'column_222',
'column_223', 'column_224', 'column_225'});

predictorNames = {'column_1', 'column_2', 'column_3',
'column_4', 'column_5', 'column_6', 'column_7', 'column_8',
```

```
'column_9', 'column_10', 'column_11', 'column_12',
'column_13', 'column_14', 'column_15', 'column_16',
'column_17', 'column_18', 'column_19', 'column_20',
'column_21', 'column_22', 'column_23', 'column_24',
'column_25', 'column_26', 'column_27', 'column_28',
'column_29', 'column_30', 'column_31', 'column_32',
'column_33', 'column_34', 'column_35', 'column_36',
'column_37', 'column_38', 'column_39', 'column_40',
'column_41', 'column_42', 'column_43', 'column_44',
'column_45', 'column_46', 'column_47', 'column_48',
'column_49', 'column_50', 'column_51', 'column_52',
'column_53', 'column_54', 'column_55', 'column_56',
'column_57', 'column_58', 'column_59', 'column_60',
'column_61', 'column_62', 'column_63', 'column_64',
'column_65', 'column_66', 'column_67', 'column_68',
'column_69', 'column_70', 'column_71', 'column_72',
'column_73', 'column_74', 'column_75', 'column_76',
'column_77', 'column_78', 'column_79', 'column_80',
'column_81', 'column_82', 'column_83', 'column_84',
'column_85', 'column_86', 'column_87', 'column_88',
'column_89', 'column_90', 'column_91', 'column_92',
'column_93', 'column_94', 'column_95', 'column_96',
'column_97', 'column_98', 'column_99', 'column_100',
'column_101', 'column_102', 'column_103', 'column_104',
'column_105', 'column_106', 'column_107', 'column_108',
'column_109', 'column_110', 'column_111', 'column_112',
'column_113', 'column_114', 'column_115', 'column_116',
'column_117', 'column_118', 'column_119', 'column_120',
'column_121', 'column_122', 'column_123', 'column_124',
'column_125', 'column_126', 'column_127', 'column_128',
'column_129', 'column_130', 'column_131', 'column_132',
'column_133', 'column_134', 'column_135', 'column_136',
'column_137', 'column_138', 'column_139', 'column_140',
'column_141', 'column_142', 'column_143', 'column_144',
'column_145', 'column_146', 'column_147', 'column_148',
'column_149', 'column_150', 'column_151', 'column_152',
'column_153', 'column_154', 'column_155', 'column_156',
'column_157', 'column_158', 'column_159', 'column_160',
'column_161', 'column_162', 'column_163', 'column_164',
'column_165', 'column_166', 'column_167', 'column_168',
'column_169', 'column_170', 'column_171', 'column_172',
```

```
'column_173', 'column_174', 'column_175', 'column_176',
'column_177', 'column_178', 'column_179', 'column_180',
'column_181', 'column_182', 'column_183', 'column_184',
'column_185', 'column_186', 'column_187', 'column_188',
'column_189', 'column_190', 'column_191', 'column_192',
'column_193', 'column_194', 'column_195', 'column_196',
'column_197', 'column_198', 'column_199', 'column_200',
'column_201', 'column_202', 'column_203', 'column_204',
'column_205', 'column_206', 'column_207', 'column_208',
'column_209', 'column_210', 'column_211', 'column_212',
'column_213', 'column_214', 'column_215', 'column_216',
'column_217', 'column_218', 'column_219', 'column_220',
'column_221', 'column_222', 'column_223', 'column_224'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_225;
isCategoricalPredictor = [false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
```

```
false, false, false, false, false, false, false, false,
false, false, false, false, false, false, false, false,
false, false, false];


isCategoricalPredictorBeforePCA = isCategoricalPredictor;
numericPredictors = predictors(:, ~isCategoricalPredictor);
numericPredictors = table2array(varfun(@double,
numericPredictors));
% 'inf' values have to be treated as missing data for PCA.
numericPredictors(isinf(numericPredictors)) = NaN;
[pcaCoefficients, pcaScores, ~, ~, explained, pcaCenters] =
pca(...
    numericPredictors);
% Keep enough components to explain the desired amount of
variance.
explainedVarianceToKeepAsFraction = Variance_pca/100;
numComponentsToKeep = find(cumsum(explained)/sum(explained)
>= explainedVarianceToKeepAsFraction, 1);
pcaCoefficients = pcaCoefficients(:,1:numComponentsToKeep);
predictors = [array2table(pcaScores(:,
1:numComponentsToKeep)), predictors(:,
isCategoricalPredictor)];
isCategoricalPredictor = [false(1,numComponentsToKeep),
true(1,sum(isCategoricalPredictor))];

%% Recherche le meilleur classifier

%on teste FINE TREE

fineTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'ClassNames', [1; 2]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
```

```
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
finetreePredictFcn = @(x) predict(fineTree, x);
trained_finetree.predictFcn = @(x)
finetreePredictFcn(pcaTransformationFcn(predictorExtractionF
cn(x)));

% Add additional fields to the result struct
trained_finetree.PCACenters = pcaCenters;
trained_finetree.PCACoefficients = pcaCoefficients;
trained_finetree.fineTree = fineTree;
trained_finetree.About = 'This struct is a trained model
exported from Classification Learner R2018a.';
trained_finetree.HowToPredict = sprintf('To make predictions
on a new predictor column matrix, X, use: \n  yfit =
c.predictFcn(X) \nreplacing ''c'' with the name of the
variable that is this struct, e.g. ''trainedModel''. \n \nX
must contain exactly 224 columns because this model was
trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

%calcul de l'accuracy
G=trained_finetree.predictFcn(M);
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end

Accuracy=pos/(l+v)
```

```
fonction_prediction=trained_finetree.predictFcn;
%on teste medium tree

mediumTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 20, ...
    'Surrogate', 'off', ...
    'ClassNames', [1; 2]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
mediumtreePredictFcn = @(x) predict(mediumTree, x);
trained_mediumtree.predictFcn = @(x)
mediumtreePredictFcn(pcaTransformationFcn(predictorExtractio
nFcn(x)));

% Add additional fields to the result struct
trained_mediumtree.PCACenters = pcaCenters;
trained_mediumtree.PCACoefficients = pcaCoefficients;
trained_mediumtree.mediumTree = mediumTree;
trained_mediumtree.About = 'This struct is a trained model
exported from Classification Learner R2018a.';
trained_mediumtree.HowToPredict = sprintf('To make
predictions on a new predictor column matrix, X, use: \n
yfit = c.predictFcn(X) \nreplacing ''c'' with the name of
the variable that is this struct, e.g. ''trainedModel''. \n
\nX must contain exactly 224 columns because this model was
trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
```

```
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

%calcul de l'accuracy
G=trained_mediumtree.predictFcn(M);
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end


if pos/(l+v)>Accuracy;
    Accuracy=pos/(l+v)
    fonction_prediction=trained_mediumtree.predictFcn;
end
%on teste COARSE TREE

coarseTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 4, ...
    'Surrogate', 'off', ...
    'ClassNames', [1; 2]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
coarsetreePredictFcn = @(x) predict(coarseTree, x);
trained_coarsetree.predictFcn = @(x)
coarsetreePredictFcn(pcaTransformationFcn(predictorExtractio
nFcn(x)));

% Add additional fields to the result struct
```

```
trained_coarsetree.PCACenters = pcaCenters;
trained_coarsetree.PCACoefficients = pcaCoefficients;
trained_coarsetree.ClassificationTree = coarseTree;
trained_coarsetree.About = 'This struct is a trained model
exported from Classification Learner R2018a.';
trained_coarsetree.HowToPredict = sprintf('To make
predictions on a new predictor column matrix, X, use: \n
yfit = c.predictFcn(X) \nreplacing ''c'' with the name of
the variable that is this struct, e.g. ''trainedModel''. \n
\nX must contain exactly 224 columns because this model was
trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

%calcul de l'accuracy
G=trained_coarsetree.predictFcn(M);
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end


if pos/(l+v)>Accuracy;
    Accuracy=pos/(l+v)
    fonction_prediction=trained_coarsetree.predictFcn;
end


% on teste logistic regression

% Train a classifier
% This code specifies all the classifier options and trains
the classifier.
```

```
% For logistic regression, the response values must be
converted to zeros
% and ones because the responses are assumed to follow a
binomial
% distribution.
% 1 or true = 'successful' class
% 0 or false = 'failure' class
% NaN - missing response.
successClass = double(2);
failureClass = double(1);
% Compute the majority response class. If there is a NaN-
prediction from
% fitglm, convert NaN to this majority class label.
numSuccess = sum(response == successClass);
numFailure = sum(response == failureClass);
if numSuccess > numFailure
    missingClass = successClass;
else
    missingClass = failureClass;
end
successFailureAndMissingClasses = [successClass;
failureClass; missingClass];
isMissing = isnan(response);
zeroOneResponse = double(ismember(response, successClass));
zeroOneResponse(isMissing) = NaN;
% Prepare input arguments to fitglm.
concatenatedPredictorsAndResponse = [predictors,
table(zeroOneResponse)];
% Train using fitglm.
GeneralizedLinearModel = fitglm(...
    concatenatedPredictorsAndResponse, ...
    'Distribution', 'binomial', ...
    'link', 'logit');

% Convert predicted probabilities to predicted class labels
and scores.
convertSuccessProbsToPredictions = @(p)
successFailureAndMissingClasses( ~isnan(p).*( (p<0.5) + 1 )
+ isnan(p)*3 );
returnMultipleValuesFcn = @(varargin)
varargin{1:max(1,nargout)};
```

```
scoresFcn = @(p) [1-p, p];
predictionsAndScoresFcn = @(p)
returnMultipleValuesFcn( convertSuccessProbsToPredictions(p)
, scoresFcn(p) );

predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
logisticRegressionPredictFcn = @(x) predictionsAndScoresFcn(
predict(GeneralizedLinearModel, x) );
trained_logisticRegression.predictFcn = @(x)
logisticRegressionPredictFcn(pcaTransformationFcn(predictorE
xtractionFcn(x)));
trained_logisticRegression.PCACenters = pcaCenters;
trained_logisticRegression.GeneralizedLinearModel =
GeneralizedLinearModel;
trained_logisticRegression.SuccessClass = successClass;
trained_logisticRegression.FailureClass = failureClass;
trained_logisticRegression.MissingClass = missingClass;
trained_logisticRegression.ClassNames = {successClass;
failureClass};
trained_logisticRegression.About = 'This struct is a trained
model exported from Classification Learner R2018a.';
trained_logisticRegression.HowToPredict = sprintf('To make
predictions on a new predictor column matrix, X, use: \n
yfit = c.predictFcn(X) \nreplacing ''c'' with the name of
the variable that is this struct, e.g. ''trainedModel''. \n
\nX must contain exactly 224 columns because this model was
trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

G=trained_logisticRegression.predictFcn(M);
```

```
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end

if pos/(l+v)>Accuracy;
    Accuracy=pos/(l+v)

fonction_prediction=trained_logisticRegression.predictFcn;
end
%on teste quadratic svm

quadraticSVM = fitcsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 2, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', [1; 2]);

predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
svmPredictFcn = @(x) predict(quadraticSVM, x);
trained_quadraticSVM.predictFcn = @(x)
svmPredictFcn(pcaTransformationFcn(predictorExtractionFcn(x)
));
trained_quadraticSVM.PCACenters = pcaCenters;
trained_quadraticSVM.PCACoefficients = pcaCoefficients;
trained_quadraticSVM.quadraticSVM = quadraticSVM;
trained_quadraticSVM.About = 'This struct is a trained model
exported from Classification Learner R2018a.';
```

```
trained_quadraticSVM.HowToPredict = sprintf('To make
predictions on a new predictor column matrix, X, use: \n
yfit = c.predictFcn(X) \nreplacing ''c'' with the name of
the variable that is this struct, e.g. ''trainedModel''. \n
\nX must contain exactly 224 columns because this model was
trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

G=trained_quadraticSVM.predictFcn(M);
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end

if pos/(l+v)>Accuracy;
    Accuracy=pos/(l+v)
    fonction_prediction=trained_quadraticSVM.predictFcn;
end
%on teste cubic svm


cubicSVM = fitcsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 3, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', [1; 2]);
```

```
predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
svmPredictFcn = @(x) predict(cubicSVM, x);
trained_cubicSVM.predictFcn = @(x)
svmPredictFcn(pcaTransformationFcn(predictorExtractionFcn(x)
));
trained_cubicSVM.PCACenters = pcaCenters;
trained_cubicSVM.PCACoefficients = pcaCoefficients;
trained_cubicSVM.CubicSVM = cubicSVM;
trained_cubicSVM.About = 'This struct is a trained model
exported from Classification Learner R2018a.';
trained_cubicSVM.HowToPredict = sprintf('To make predictions
on a new predictor column matrix, X, use: \n  yfit =
c.predictFcn(X) \nreplacing ''c'' with the name of the
variable that is this struct, e.g. ''trainedModel''. \n \nX
must contain exactly 224 columns because this model was
trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

G=trained_cubicSVM.predictFcn(M);
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end
if pos/(l+v)>Accuracy;
    Accuracy=pos/(l+v)
    fonction_prediction=trained_cubicSVM.predictFcn;
end
```

```
    %on teste ensemble subspace discriminant

subspaceDimension = max(1, min(112, width(predictors) - 1));
subspacediscrEnsemble = fitcensemble(...
    predictors, ...
    response, ...
    'Method', 'Subspace', ...
    'NumLearningCycles', 30, ...
    'Learners', 'discriminant', ...
    'NPredToSample', subspaceDimension, ...
    'ClassNames', [1; 2]);

predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
subspacediscrEnsemblePredictFcn = @(x)
predict(subspacediscrEnsemble, x);
trained_subspacediscrEnsemble.predictFcn = @(x)
subspacediscrEnsemblePredictFcn(pcaTransformationFcn(predict
orExtractionFcn(x)));
trained_subspacediscrEnsemble.PCACenters = pcaCenters;
trained_subspacediscrEnsemble.PCACoefficients =
pcaCoefficients;
trained_subspacediscrEnsemble.ClassificationEnsemble =
subspacediscrEnsemble;
trained_subspacediscrEnsemble.About = 'This struct is a
trained model exported from Classification Learner R2018a.';
trained_subspacediscrEnsemble.HowToPredict = sprintf('To
make predictions on a new predictor column matrix, X, use:
\n  yfit = c.predictFcn(X) \nreplacing ''c'' with the name
of the variable that is this struct, e.g. ''trainedModel''.
\n \nX must contain exactly 224 columns because this model
was trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
```

```
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

G=trained_subspacediscrEnsemble.predictFcn(M);
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end
if pos/(l+v)>Accuracy;
    Accuracy=pos/(l+v)

fonction_prediction=trained_subspacediscrEnsemble.predictFcn
;
end

    % on train subspace knn

subspaceknnEnsemble = fitcensemble(...
    predictors, ...
    response, ...
    'Method', 'Subspace', ...
    'NumLearningCycles', 30, ...
    'Learners', 'knn', ...
    'NPredToSample', subspaceDimension, ...
    'ClassNames', [1; 2]);

predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
pcaTransformationFcn = @(x)
[ array2table((table2array(varfun(@double, x(:,
~isCategoricalPredictorBeforePCA))) - pcaCenters) *
pcaCoefficients), x(:,isCategoricalPredictorBeforePCA) ];
knnensemblePredictFcn = @(x) predict(subspaceknnEnsemble,
x);
trained_subspaceknn.predictFcn = @(x)
knnensemblePredictFcn(pcaTransformationFcn(predictorExtracti
onFcn(x)));
```

```
trained_subspaceknn.PCACenters = pcaCenters;
trained_subspaceknn.PCACoefficients = pcaCoefficients;
trained_subspaceknn.ClassificationEnsemble =
subspaceknnEnsemble;
trained_subspaceknn.About = 'This struct is a trained model
exported from Classification Learner R2018a.';
trained_subspaceknn.HowToPredict = sprintf('To make
predictions on a new predictor column matrix, X, use: \n
yfit = c.predictFcn(X) \nreplacing ''c'' with the name of
the variable that is this struct, e.g. ''trainedModel''. \n
\nX must contain exactly 224 columns because this model was
trained using 224 predictors. \nX must contain only
predictor columns in exactly the same order and format as
your training \ndata. Do not include the response column or
any columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

G=trained_subspaceknn.predictFcn(M);
pos=0;
for i=1:l+v ;
    if G(i)==P(i);
        pos=pos+1;
    end

end

if pos/(l+v)>Accuracy;
    Accuracy=pos/(l+v)
    fonction_prediction=trained_subspaceknn.predictFcn;

end
```

ANNEXE- SERIE 3

```
%% pwelch config
[f_min,f_max,c_welch_window,c_welch_recover]=pwelchconfig();


%% Configure Psychotoolbox
[screens,screenNumber,white,black,window,windowRect,screenXp
ixels,screenYpixels,xCenter,yCenter,ifi,waitframes,nominalFr
ameRate] = configpsychotoolbox();

%% sine wave parameters
amplitude = 1;
frequency = 1;
angFreq = 2 * pi * frequency;
startPhase = 0;
%% squares
    square=[0 0 100 200];
squarecolor=[1 0 0];
barre=CenterRectOnPointd(square,screenXpixels/
2,screenYpixels/2);

%% Init. server
sampling_rate  = 500;    % Sampling rate EEG [SPS]
period = 3;
n_channels = 8;

%% keys
escapeKey = KbName('ESCAPE');
keysOfInterest=zeros(1,256);
keysOfInterest(escapeKey)=1;
KbQueueCreate([],keysOfInterest);
KbQueueStart();

%% init save
rt=[];
rtn=1; %indexacao de rt
%% main ligne de base
STARTEXP=now*24*3600;
Screen('TextSize', window, 80);
```

```
Screen('TextFont', window, 'Arial');
DrawFormattedText(window, 'Appuyer sur une touche pour
commencer','center','center',black);
Screen('Flip',window);
KbStrokeWait;

exit=0;
ratio=0;
tempo=30;
n=0;
eeg_set=[];
direction='Enregistrement de la ligne de base...\n\n';
[ret]=MatNICMarkerSendLSL(3,outlet); %send marker

while(tempo>0)
    start=now*24*3600;
    %just if we want to quit...
    [ pressed, firstPress]=KbQueueCheck();
    if pressed
        if firstPress(escapeKey)
            exit=1;
            break;
        end
    end
    %quit!
    message=[direction num2str(tempo)];
    DrawFormattedText(window,
message,'center','center',black);
    Screen('Flip',window);
    eeg_set=lereeg(eeg_set,inletEEG,3);
    eeg=detrend(eeg_set);
    eeg=pwelch(eeg,c_welch_window,c_welch_recover,
f_min:f_max,500);
    moyD=sum(sum(eeg(:,1:4)))/(f_max-f_min+1)/4;
    moyG=sum(sum(eeg(:,5:8)))/(f_max-f_min+1)/4;
    rt(rtn,:)=[moyD/moyG 3 now*24*3600-STARTEXP];
    rtn=rtn+1;
    tempo=tempo-(now*24*3600-start);
end
[ret]=MatNICMarkerSendLSL(4,outlet); %send marker
ratiobase=mean(rt(:,1));
```

```
dp=std(rt(:,1));
ratio=1;
moyD=0;
moyG=0;

%% main mesures
gd=[1 2 1 2 1 2 1 2 1 2]; %1=gauche 2=droite
n=length(gd);
x=0;
tempo=0;
k=1;
if(exit==0)
while k<n+1
    if gd(k)==1
        direction='Gauche !\n\n';
    end
    if gd(k)==2
        direction='Droite !\n\n';
    end
    message=[direction num2str(ratio/ratiobase,'%2.1f')];
    DrawFormattedText(window,message,'center',100,black);
    Screen('FillRect',window,squarecolor,barre);
    Screen('Flip',window);
    [ret]=MatNICMarkerSendLSL(gd(k),outlet); %send marker
    rt(rtn,:)=[ratio/ratiobase gd(k) now*24*3600-STARTEXP];
    rtn=rtn+1;
    while tempo<20
        starttime=now*24*3600;
        eeg_set=lereeg(eeg_set,inletEEG,3);
        eeg=detrend(eeg_set);
        eeg=pwelch(eeg,c_welch_window,c_welch_recover,
f_min:f_max);
        moyD=sum(sum(eeg(:,1:4)))/(f_max-f_min+1)/4;
        moyG=sum(sum(eeg(:,5:8)))/(f_max-f_min+1)/4;
        ratio=moyD/moyG;
        rt(rtn,:)=[ratio/ratiobase 0 now*24*3600-STARTEXP];
        rtn=rtn+1;
        if ratio>(ratiobase+dp)
            x=x+0.1*screenXpixels/4;
            if(x>screenXpixels/4)
                x=screenXpixels/4;
```

```
            end
            barre=CenterRectOnPointd(square,screenXpixels/
2+x,screenYpixels/2);
        end
        if ratio<(ratiobase-dp)
            x=x-0.1*screenXpixels/4;
            if(x<-screenXpixels/4)
                x=-screenXpixels/4;
            end
            barre=CenterRectOnPointd(square,screenXpixels/
2+x,screenYpixels/2);
        end
        message=[direction num2str(ratio/
ratiobase,'%2.1f')];
        DrawFormattedText(window,message,'center',
100,black);
        Screen('FillRect',window,squarecolor,barre)
        Screen('Flip',window);
        tempo=tempo + (now*24*3600-starttime);
        %just if we want to quit...
        [ pressed, firstPress]=KbQueueCheck();
        if pressed
            if firstPress(escapeKey)
                exit=1;
                break;
            end
        end
        %quit!
    end
    if exit==1
        break;
    end
    tempo=0;
    k=k+1;
    x=0;
    barre=CenterRectOnPointd(square,screenXpixels/
2,screenYpixels/2);
end
end
[ret]=MatNICMarkerSendLSL(5,outlet); %send marker
```

```
Screen('TextSize', window, 80);
Screen('TextFont', window, 'Arial');
DrawFormattedText(window, 'C''est fini !\n(Appuyer sur une
touche pour fermer...)','center','center',black);
Screen('Flip',window);
KbStrokeWait;
sca;
```

ANNEXE - FONCTION COMMUNES

Configuration Psychtoolbox (configpsychtoolbox)

```
function
[screens,screenNumber,white,black,window,windowRect,screenXp
ixels,screenYpixels,xCenter,yCenter,ifi,waitframes,nominalFr
ameRate] = configpsychotoolbox()
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
% Here we call some default settings for setting up
Psychtoolbox
PsychDefaultSetup(2);

% Get the screen numbers. This gives us a number for each of
the screens
% attached to our computer.
screens = Screen('Screens');

% Draw we select the maximum of these numbers. So in a
situation where we
% have two screens attached to our monitor we will draw to
the external
% screen. When only one screen is attached to the monitor we
will draw to
% this.
screenNumber = max(screens);

% Define black and white (white will be 1 and black 0).
white = WhiteIndex(screenNumber);
black = BlackIndex(screenNumber);

% Open an on screen window and color it black
[window, windowRect] = PsychImaging('OpenWindow',
screenNumber, white);

% Get the size of the on screen window in pixels
[screenXpixels, screenYpixels] = Screen('WindowSize',
window);
```

```
% Get the centre coordinate of the window in pixels
[xCenter, yCenter] = RectCenter(windowRect);

% Enable alpha blending for anti-aliasing
Screen('BlendFunction', window, GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA);

% Query the frame duration
ifi = Screen('GetFlipInterval', window);
waitframes=1;
% Get the nominal framerate of the monitor. For this simple
timer we are
% going to change the counterdown number every second. This
means we
% present each number for "frameRate" amount of frames. This
is because
% "framerate" amount of frames is equal to one second. Note:
this is only
% for a very simple timer to demonstarte the principle. You
can make more
% accurate sub-second timers based on this.
nominalFrameRate = Screen('NominalFrameRate', window);
end
```

Initialisation interface MatNIC Matlab

```
%Configure MatNIC (ampli)
[ret,inletEEG]=MatNICEEGConnectLSL('NIC');
[ret,outlet]=MatNICMarkerConnectLSL('Marker');

[ret,status,socket]=MatNICConnect('localhost');
[ret,
version,n_channels,deviceSettings]=MatNICSetUp(socket);
[ret,outlet]=MatNICMarkerConnectLSL('Marker');
```

Configuration pwelch (pwelchconfig)

```
function [f_min,f_max,c_welch_window,c_welch_recover] =
pwelchconfig()
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
f_min=13;
```

```
f_max=15;
c_welch_window = 500;
c_welch_recover =250;
end
```

Test de Permutation

```
% [p, observeddifference, effectsize] =
permutationTest(sample1, sample2, permutations [, varargin])
%
%       Permutation test (aka randomisation test), testing
for a difference
%       in means between two samples.
%
% In:
%       sample1 - vector of measurements representing one
sample
%       sample2 - vector of measurements representing a
second sample
%       permutations - the number of permutations'
%
% Optional (name-value pairs):
%       sidedness - whether to test one- or two-sided:
%           'both' - test two-sided (default)
%           'smaller' - test one-sided, alternative
hypothesis is that
%                       the mean of sample1 is smaller than
the mean of
%                       sample2
%           'larger' - test one-sided, alternative
hypothesis is that
%                      the mean of sample1 is larger than
the mean of
%                      sample2
%       exact - whether or not to run an exact test, in
which all possible
%               combinations are considered. this is only
feasible for
%               relatively small sample sizes. the
'permutations' argument
```

```
%                will be ignored for an exact test. (1|0,
default 0)
%       plotresult - whether or not to plot the distribution
of randomised
%                    differences, along with the observed
difference (1|0,
%                    default: 0)x
%       showprogress - whether or not to show a progress
bar. if 0, no bar
%                      is displayed; if showprogress > 0,
the bar updates
%                      every showprogress-th iteration.
%
% Out:
%       p - the resulting p-value
%       observeddifference - the observed difference between
the two
%                            samples, i.e. mean(sample1) -
mean(sample2)
%       effectsize - the effect size
%
% Usage example:
%       >> permutationTest(rand(1,100), rand(1,100)-.25,
10000, ...
%          'plotresult', 1, 'showprogress', 250)
%
%                    Copyright 2015-2018 Laurens R Krol
%                    Team PhyPA, Biological Psychology and
Neuroergonomics,
%                    Berlin Institute of Technology

% 2019-02-01 lrk
%   - Added short description
%   - Increased the number of bins in the plot
% 2018-03-15 lrk
%   - Suppressed initial MATLAB:nchoosek:LargeCoefficient
warning
% 2018-03-14 lrk
%   - Added exact test
% 2018-01-31 lrk
%   - Replaced calls to mean() with nanmean()
```

```
% 2017-06-15 lrk
%   - Updated waitbar message in first iteration
% 2017-04-04 lrk
%   - Added progress bar
% 2017-01-13 lrk
%   - Switched to inputParser to parse arguments
% 2016-09-13 lrk
%   - Caught potential issue when column vectors were used
%   - Improved plot
% 2016-02-17 toz
%   - Added plot functionality
% 2015-11-26 First version

% This program is free software: you can redistribute it
and/or modify
% it under the terms of the GNU General Public License as
published by
% the Free Software Foundation, either version 3 of the
License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be
useful,
% but WITHOUT ANY WARRANTY; without even the implied
warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public
License
% along with this program.  If not, see <http://www.gnu.org/
licenses/>.


function [p, observeddifference, effectsize] =
permutationTest(sample1, sample2, permutations, varargin)

% parsing input
p = inputParser;
```

```
addRequired(p, 'sample1', @isnumeric);
addRequired(p, 'sample2', @isnumeric);
addRequired(p, 'permutations', @isnumeric);

addParamValue(p, 'sidedness', 'both', @(x)
any(validatestring(x,{'both', 'smaller', 'larger'})));
addParamValue(p, 'exact' , 0, @isnumeric);
addParamValue(p, 'plotresult', 0, @isnumeric);
addParamValue(p, 'showprogress', 0, @isnumeric);

parse(p, sample1, sample2, permutations, varargin{:})

sample1 = p.Results.sample1;
sample2 = p.Results.sample2;
permutations = p.Results.permutations;
sidedness = p.Results.sidedness;
exact = p.Results.exact;
plotresult = p.Results.plotresult;
showprogress = p.Results.showprogress;

% enforcing row vectors
if iscolumn(sample1), sample1 = sample1'; end
if iscolumn(sample2), sample2 = sample2'; end

allobservations = [sample1, sample2];
observeddifference = nanmean(sample1) - nanmean(sample2);
effectsize = observeddifference / nanmean([std(sample1),
std(sample2)]);

w = warning('off', 'MATLAB:nchoosek:LargeCoefficient');
if ~exact && permutations > nchoosek(numel(allobservations),
numel(sample1))
    warning(['the number of permutations (%d) is higher than
the number of possible combinations (%d);\n' ...
             'consider running an exact test using the
''exact'' argument'], ...
             permutations, nchoosek(numel(allobservations),
numel(sample1)));
end
warning(w);
```

```
if showprogress, w = waitbar(0, 'Preparing test...', 'Name',
'permutationTest'); end

if exact
    % getting all possible combinations
    allcombinations = nchoosek(1:numel(allobservations),
numel(sample1));
    permutations = size(allcombinations, 1);
end

% running test
randomdifferences = zeros(1, permutations);
if showprogress, waitbar(0, w, sprintf('Permutation 1 of
%d', permutations), 'Name', 'permutationTest'); end
for n = 1:permutations
    if showprogress && mod(n,showprogress) == 0, waitbar(n/
permutations, w, sprintf('Permutation %d of %d', n,
permutations)); end

    % selecting either next combination, or random
permutation
    if exact, permutation = [allcombinations(n,:),
setdiff(1:numel(allobservations), allcombinations(n,:))];
    else, permutation = randperm(length(allobservations));
end

    % dividing into two samples
    randomSample1 =
allobservations(permutation(1:length(sample1)));
    randomSample2 =
allobservations(permutation(length(sample1)+1:length(permuta
tion)));

    % saving differences between the two samples
    randomdifferences(n) = nanmean(randomSample1) -
nanmean(randomSample2);
end
if showprogress, delete(w); end

% getting probability of finding observed difference from
random permutations
```

```
if strcmp(sidedness, 'both')
    p = (length(find(abs(randomdifferences) >
abs(observeddifference)))+1) / (permutations+1);
elseif strcmp(sidedness, 'smaller')
    p = (length(find(randomdifferences <
observeddifference))+1) / (permutations+1);
elseif strcmp(sidedness, 'larger')
    p = (length(find(randomdifferences >
observeddifference))+1) / (permutations+1);
end

% plotting result
if plotresult
    figure;
    hist(randomdifferences, 20);
    hold on;
    xlabel('Random differences');
    ylabel('Count')
    od = plot(observeddifference, 0, '.r','markersize',20,
'DisplayName', sprintf('Observed difference.\np = %.2f',
p));
    legend(od);
end

end
```

Normalisation des données de l'expérience 3

```
%Pegar todos os valores de esquerda e direita e colocar em
vetores rtg e
%rtd
clear all
%exp={'camille.mat'}
exp={'agathe.mat' 'bruno.mat' 'camille.mat' 'emile.mat'
'ismael.mat' 'laura.mat' 'Mahamane.mat' 'mstthieu2.mat'
'Nicolas.mat' 'rayan.mat' 'remi.mat' 'simone.mat'
'valentine.mat'};
RTD=[];
RTG=[];
p=[];
media=[];
dv=[];
```

```
for j=1:length(exp)
    rtg=[];
    rtd=[];
    rt=[];
    a=load(exp{j});
    rt=a.rt;
    imax=find(rt(:,1)==0,1);
    if isempty(imax)==0
        rt=rt(1:imax-1,:);
    end
    i1=find(rt(:,2)==1);
    i2=find(rt(:,2)==2);
    if(i2(end)>i1(end)) %termina com direita
        k=length(i1);
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            l=l+(i2(i)-i1(i));
        end
        k=length(i2)-1;
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1)=(rt(i2(i)+1:i1(i+1),
1)-media(i))/dv(i);
            l=l+(i1(i+1)-i2(i));
        end
        rtd(l:l+length(rt(:,1))-i2(end)-1)=(rt(i2(end)
+1:length(rt(:,1)))-media(end))/dv(end);
    else %termina com esquerda
        k=length(i1)-1;
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            l=l+(i2(i)-i1(i));
        end
```

```
        media(k+1)=mean(rt(i1(end)+1:length(rt(:,1))));
        dv(k+1)=std(rt(i1(end)+1:length(rt(:,1))));
        rtg(l:l+length(rt(:,1))-i1(end)-1)=(rt(i1(end)
+1:length(rt(:,1)))-media(k+1))/dv(k+1);
        k=length(i2);
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1)=(rt(i2(i)+1:i1(i+1),
1)-media(i))/dv(i);
            l=l+(i1(i+1)-i2(i));
        end
    end
    RTD=[RTD rtd];
    RTG=[RTG rtg];
    p=[p (mean(rtg)-mean(rtd))];
end
```

Renormalisation des données de l'expérience trois (moyenne dans chaque intervalle)

```
%Pegar todos os valores de esquerda e direita e colocar em
vetores rtg e
%rtd
clear all
%exp={'camille.mat'};
exp={'agathe.mat' 'bruno.mat' 'camille.mat' 'emile.mat'
'ismael.mat' 'laura.mat' 'Mahamane.mat' 'mstthieu2.mat'
'Nicolas.mat' 'rayan.mat' 'remi.mat' 'simone.mat'
'valentine.mat'};
RTD=[];
RTG=[];
media=[];
dv=[];
for j=1:length(exp)
    rtg=[];
    rtd=[];
    mrtg=[];
    mrtd=[];
    rt=[];
    a=load(exp{j});
    rt=a.rt;
    imax=find(rt(:,1)==0,1);
```

```
    if isempty(imax)==0
        rt=rt(1:imax-1,:);
    end
    i1=find(rt(:,2)==1);
    i2=find(rt(:,2)==2);
    if(i2(end)>i1(end)) %termina com direita
        k=length(i1);
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            mrtg(i)=mean(rtg(l:l+(i2(i)-i1(i))-1));
            l=l+(i2(i)-i1(i));
        end
        k=length(i2)-1;
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1)=(rt(i2(i)+1:i1(i+1),
1)-media(i))/dv(i);
            mrtd(i)=mean(rtd(l:l+(i1(i+1)-i2(i))-1));
            l=l+(i1(i+1)-i2(i));
        end
        rtd(l:l+length(rt(:,1))-i2(end)-1)=(rt(i2(end)
+1:length(rt(:,1)))-media(end))/dv(end);
        mrtd(k+1)=mean(rtd(l:l+length(rt(:,1))-i2(end)-1));
    else %termina com esquerda
        k=length(i1)-1;
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            mrtg(i)=mean(rtg(l:l+(i2(i)-i1(i))-1));
            l=l+(i2(i)-i1(i));
        end
        media(k+1)=mean(rt(i1(end)+1:length(rt(:,1))));
        dv(k+1)=std(rt(i1(end)+1:length(rt(:,1))));
```

```
        rtg(l:l+length(rt(:,1))-i1(end)-1)=(rt(i1(end)
+1:length(rt(:,1)))-media(k+1))/dv(k+1);
        mrtg(k+1)=mean(rtg(l:l+length(rt(:,1))-i1(end)-1));
        k=length(i2);
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1)=(rt(i2(i)+1:i1(i+1),
1)-media(i))/dv(i);
            mrtd(i)=mean(rtd(l:l+(i1(i+1)-i2(i))-1));
            l=l+(i1(i+1)-i2(i));
        end
    end
    RTD=[RTD mrtd];
    RTG=[RTG mrtg];
end
```

Figure des données brutes pour un sujet

```
clear all
exp={'rayan.mat'};
a=load(exp{1});
rt=a.rt;
figure
plot(rt(:,3),rt(:,1),'.','color','black')
i3=find(rt(:,2)==3);
i3max=max(i3);
i2=find(rt(:,2)==2);
i1=find(rt(:,2)==1);
line([rt(i3max,3),rt(i3max,3)],
[0,5],'color','black','linestyle','--')
for i=i2
    line([rt(i,3),rt(i,3)],
[0,5],'color','red','linestyle','--')
end
for i=i1
    line([rt(i,3),rt(i,3)],
[0,5],'color','blue','linestyle','--')
end
axis([0 rt(end,3) min(rt(:,1))-0.5 max(rt(:,1))+0.5])
xlabel("Temps (s)")
ylabel("Signal/SignalBase")
title("Signal Mesuré")
```

Figure des données normalisées pour un sujet

```
clear all
exp={'Nicolas.mat'}
RTD=[];
RTG=[];
p=[];
media=[];
dv=[];
for j=1:length(exp)
    rtg=[];
    rtd=[];
    rt=[];
    a=load(exp{j});
    rt=a.rt;
    imax=find(rt(:,1)==0,1);
    if isempty(imax)==0
        rt=rt(1:imax-1,:);
    end
    i1=find(rt(:,2)==1);
    i2=find(rt(:,2)==2);
    if(i2(end)>i1(end)) %termina com direita
        k=length(i1);
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1,1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            rtg(l:l+(i2(i)-i1(i))-1,2)=rt(i1(i)+1:i2(i),3);
            l=l+(i2(i)-i1(i));
        end
        k=length(i2)-1;
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1,1)=(rt(i2(i)
+1:i1(i+1),1)-media(i))/dv(i);
            rtd(l:l+(i1(i+1)-i2(i))-1,2)=rt(i2(i)+1:i1(i+1),
3);
            l=l+(i1(i+1)-i2(i));
        end
```

```
        rtd(l:l+length(rt(:,1))-i2(end)-1,1)=(rt(i2(end)
+1:length(rt(:,1)),1)-media(end))/dv(end);
        rtd(l:l+length(rt(:,1))-i2(end)-1,2)=rt(i2(end)
+1:length(rt(:,1)),3);
    else %termina com esquerda
        k=length(i1)-1;
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1,1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            rtg(l:l+(i2(i)-i1(i))-1,2)=rt(i1(i)+1:i2(i),3);
            l=l+(i2(i)-i1(i));
        end
        media(k+1)=mean(rt(i1(end)+1:length(rt(:,1))));
        dv(k+1)=std(rt(i1(end)+1:length(rt(:,1))));
        rtg(l:l+length(rt(:,1))-i1(end)-1,1)=(rt(i1(end)
+1:length(rt(:,1)),1)-media(k+1))/dv(k+1);
        rtg(l:l+length(rt(:,1))-i1(end)-1,1)=rt(i1(end)
+1:length(rt(:,1)),3);
        k=length(i2);
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1,1)=(rt(i2(i)
+1:i1(i+1),1)-media(i))/dv(i);
            rtd(l:l+(i1(i+1)-i2(i))-1,2)=rt(i2(i)+1:i1(i+1),
3);
            l=l+(i1(i+1)-i2(i));
        end
    end
    RTD=[RTD rtd];
    RTG=[RTG rtg];
    p=[p (mean(rtg)-mean(rtd))];
end
figure
rtm=[RTD;RTG];
i3=find(rt(:,2)==3);
i3max=max(i3);
i2=find(rt(:,2)==2);
i1=find(rt(:,2)==1);
```

```
plot(rtm(:,2),rtm(:,1),'.','color','black')
line([rt(i3max,3),rt(i3max,3)],[min(rtm(:,1))-0.5,
max(rtm(:,1))+0.5],'color','black','linestyle','--')
for i=i2
    line([rt(i,3),rt(i,3)],[min(rtm(:,1))-0.5,max(rtm(:,1))
+0.5],'color','red','linestyle','--')
end
for i=i1
    line([rt(i,3),rt(i,3)],[min(rtm(:,1))-0.5,max(rtm(:,1))
+0.5],'color','blue','linestyle','--')
end
axis([0 rt(end,3) min(rtm(:,1))-0.5 max(rtm(:,1))+0.5])
```

Figure Moyenne Brute pour un sujet

```
clear all
rtm=[];
exp={'rayan.mat'};
a=load(exp{1});
rt=a.rt;
figure
i3=find(rt(:,2)==3);
i3max=max(i3);
i2=find(rt(:,2)==2);
i1=find(rt(:,2)==1);
i1=[i1;length(rt)];
for i=1:length(i1)-1
    nl=[mean(rt(i1(i):i2(i),1)) (rt(i1(i),3)+rt(i2(i),3))/
2];
    rtm=[rtm;nl];
end

for i=1:length(i2)
    nl=[mean(rt(i2(i):i1(i+1),1)) (rt(i2(i),3)+rt(i1(i+1),
3))/2];
    rtm=[rtm;nl];
end
plot(rtm(:,2),rtm(:,1),'o','color','black')
i1=i1(1:end-1);
line([rt(i3max,3),rt(i3max,3)],
[0,5],'color','black','linestyle','--')
for i=i2
```

```
    line([rt(i,3),rt(i,3)],
[0,5],'color','red','linestyle','--')
end
for i=i1
    line([rt(i,3),rt(i,3)],
[0,5],'color','blue','linestyle','--')
end
axis([0 rt(end,3) min(rtm(:,1))-0.5 max(rtm(:,1))+0.5])
xlabel("Temps (s)")
ylabel("Moyenne de Signal/SignalBase")
title("Signal Moyenné")
```

Figure moyenne normalise pour un sujet

```
clear all
exp={'rayan.mat'}
RTD=[];
RTG=[];
p=[];
media=[];
dv=[];
for j=1:length(exp)
    rtg=[];
    rtd=[];
    rt=[];
    a=load(exp{j});
    rt=a.rt;
    imax=find(rt(:,1)==0,1);
    if isempty(imax)==0
        rt=rt(1:imax-1,:);
    end
    i1=find(rt(:,2)==1);
    i2=find(rt(:,2)==2);
    if(i2(end)>i1(end)) %termina com direita
        k=length(i1);
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1,1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            mrtg(i,1)=mean(rtg(l:l+(i2(i)-i1(i))-1,1));
```

```
            rtg(l:l+(i2(i)-i1(i))-1,2)=rt(i1(i)+1:i2(i),3);
            mrtg(i,2)=(rt(i1(i)+1,3)+rt(i2(i),3))/2;
            l=l+(i2(i)-i1(i));
        end
        k=length(i2)-1;
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1,1)=(rt(i2(i)
+1:i1(i+1),1)-media(i))/dv(i);
            rtd(l:l+(i1(i+1)-i2(i))-1,2)=rt(i2(i)+1:i1(i+1),
3);
            mrtd(i,1)=mean(rtd(l:l+(i1(i+1)-i2(i))-1,1));
            mrtd(i,2)=(rt(i2(i)+1,3)+rt(i1(i+1),3))/2;
            l=l+(i1(i+1)-i2(i));
        end
        rtd(l:l+length(rt(:,1))-i2(end)-1,1)=(rt(i2(end)
+1:length(rt(:,1)),1)-media(end))/dv(end);
        rtd(l:l+length(rt(:,1))-i2(end)-1,2)=rt(i2(end)
+1:length(rt(:,1)),3);
        mrtd(k+1,1)=mean(rtd(l:l+length(rt(:,1))-
i2(end)-1,1));
        mrtd(k+1,2)=(rt(i2(end)+1,3)+rt(end,3))/2;
    else %termina com esquerda
        k=length(i1)-1;
        l=1;
        for i=1:k
            media(i)=mean(rt(i1(i)+1:i2(i),1));
            dv(i)=std(rt(i1(i)+1:i2(i),1));
            rtg(l:l+(i2(i)-i1(i))-1,1)=(rt(i1(i)+1:i2(i),1)-
media(i))/dv(i);
            rtg(l:l+(i2(i)-i1(i))-1,2)=rt(i1(i)+1:i2(i),3);
            mrtg(i,1)=mean(rtg(l:l+(i2(i)-i1(i))-1,1));
            mrtg(i,2)=(rt(i1(i)+1,3)+rt(i2(i),3))/2;
            l=l+(i2(i)-i1(i));
        end
        media(k+1)=mean(rt(i1(end)+1:length(rt(:,1))));
        dv(k+1)=std(rt(i1(end)+1:length(rt(:,1))));
        rtg(l:l+length(rt(:,1))-i1(end)-1,1)=(rt(i1(end)
+1:length(rt(:,1)),1)-media(k+1))/dv(k+1);
        rtg(l:l+length(rt(:,1))-i1(end)-1,1)=rt(i1(end)
+1:length(rt(:,1)),3);
```

```
        mrtg(k+1,1)=mean(rtg(l:l+length(rt(:,1))-
i1(end)-1,1));
        mrtg(k+1,2)=(rt(i1(end)+1,3)+rt(end,3))/2;
        k=length(i2);
        l=1;
        for i=1:k
            rtd(l:l+(i1(i+1)-i2(i))-1,1)=(rt(i2(i)
+1:i1(i+1),1)-media(i))/dv(i);
            rtd(l:l+(i1(i+1)-i2(i))-1,2)=rt(i2(i)+1:i1(i+1),
3);
            mrtd(i,1)=mean(rtd(l:l+(i1(i+1)-i2(i))-1,1));
            mrtd(i,2)=(rt(i2(i)+1,3)+rt(i1(i+1),3))/2;
            l=l+(i1(i+1)-i2(i));
        end
    end
    RTD=[RTD mrtd];
    RTG=[RTG mrtg];
end
figure
rtm=[RTD;RTG];
i3=find(rt(:,2)==3);
i3max=max(i3);
i2=find(rt(:,2)==2);
i1=find(rt(:,2)==1);
plot(rtm(:,2),rtm(:,1),'o','color','black')
line([rt(i3max,3),rt(i3max,3)],[min(rtm(:,1))-0.5,
max(rtm(:,1))+0.5],'color','black','linestyle','--')
for i=i2
    line([rt(i,3),rt(i,3)],[min(rtm(:,1))-0.5,max(rtm(:,1))
+0.5],'color','red','linestyle','--')
end
for i=i1
    line([rt(i,3),rt(i,3)],[min(rtm(:,1))-0.5,max(rtm(:,1))
+0.5],'color','blue','linestyle','--')
end
axis([0 rt(end,3) min(rtm(:,1))-0.5 max(rtm(:,1))+0.5])
xlabel("Temps (s)")
ylabel("Moyenne de Signal/SignalBase")
title("Signal Renormalisé et Moyenné")
```