**SSWT**

%EEG SSWT single file run

% previous edit Original

% last edit 8/4 Mingda

%% import files

% use load for .set and .mat files

% use readtable for .txt or .csv files

% use sload from biosig for .bdf files

% use bload from buzsaki for buzsaki data

folderPath = "/Users/Mingda/Library/Mobile Documents/com~apple~CloudDocs/Documents/UCLA/Research/EEG/Data";

fileName = "/eyesclosed30.csv";

filePath = folderPath + fileName;

filePath = "C:\Users\ATSUSHI\Desktop\Arisaka2022\_8\8.30.2022\Acharya\_2015\_fig1\_stage4\_deep\_slow\_wave.csv";

%eegTable = LoadBinary(,[1,2,3,4,5,6,7,8],8);

eegTable = readtable(filePath);

%% turn data into vertical arrays

%eegTable2 = eegTable.SIGNAL(:,17);

%eegTimeData = eegTable.SIGNAL(:,1);

%eegData = transpose(eegTable2);

%eegData = eegTable2;

%eegData = table2array(eegTable(1:10000,2));

eegArray = table2array(eegTable(:,2));

%eegData = eegTable(1:8400,11);

eegData = eegArray.';

%% sets time data

eegTimeData = linspace(0, length(eegData)-1, length(eegData));

%eegTimeData = table2array(eegTable(:,1));

Hz =1000;

%% highpass filter to get rid of <0.5hz data

%eegData = highpass(eegData, 0.5, Hz);

% defaultWSSTPlotNew( x, y, Fs, subject, experiment, date)

% x is an nx1 time vector

% y is an nx1 data vector, where n is the number of points in the 1D data time-series

% Fs is the sampling rate

defaultWSSTPlot(eegTimeData, eegData, Hz);

**defaultWSSTPlot.m**

% DEFAULTWSSTPLOT Makes a wsst plot with the most common parameters in the

% Arisaka lab

% previous edit Original

% last edit 8/4 People

function [cfs] = defaultWSSTPlot( x, y, Fs)

% x is an nx1 time vector, and y is an nx1 data vector, where n is the

% number of points in the 1D data time-series. Fs is the sampling rate

% plot raw trace and various band-pass filters of the raw trace in upper

% plot

FigH = figure('Position', get(0, 'Screensize'));

y = y - mean(y);

%% Upper plot

% set up axes

plot1 = subplot(2, 1, 1);

set(plot1, 'OuterPosition', [0, 0.7, 1, 0.3]);

% pass bands is a nx2 numeric array specifying the bandpass frequencies

% that will be displayed on the top plot in addition to the raw trace, where

% n is the number of different passbands.

% Each additional passband range is specified in a separate row. A value of

% 0 in the first column indicates a lowpass, and 0 in the second column

% indicates a highpass

%legend

if Fs < 160\*2

passBands = [1, 3.5; 3.5, 7; 7, 14; 14, 28; 28, 40; 70, 100];

legendLabels = {'Delta', 'Theta', 'Alpha', 'Beta', 'Slow Gamma', 'Mid Gamma', 'Raw Filtered'};

else

passBands = [1, 3.5; 3.5, 7; 7, 14; 14, 28; 28, 40; 70, 100; 140, 160];

legendLabels = {'Delta', 'Theta', 'Alpha', 'Beta', 'Slow Gamma', 'Mid Gamma', 'Fast Gamma', 'Raw Filtered'};

end

% colors is a nx3 numeric array of colors

colors = [0, 0.4470, 0.7410; 0.8500, 0.3250, 0.0980; 0.9290, 0.6940, 0.1250;

0.4940, 0.1840, 0.5560; 0.4660, 0.6740, 0.1880; 0.3010, 0.7450, 0.9330;

0.6350, 0.0780, 0.1840];

hold(plot1, 'on');

% set up variables to hold bandpass filtered traces

y = double(y);

filteredData = zeros(size(passBands, 1), length(y));

filterCoefficients = {};

% calculate third-order butterworth filter coefficients for each pass band specified

% change 0.5\*Fs higher values for higher sensitivity

for i = 1:size(passBands, 1)

if (passBands(i, 1) == 0 && passBands(i, 2) == 0)|| (passBands(i,1) < 0 && passBands(i,2) < 0)

continue

elseif passBands(i,1) == 0

[filterCoefficients{i,1}, filterCoefficients{i,2}] = butter(3, passBands(i, 2)/(0.5\*Fs), 'low');

elseif passBands(i, 2) == 0

[filterCoefficients{i,1}, filterCoefficients{i,2}] = butter(3, passBands(i, 1)/(0.5\*Fs), 'high');

else

[filterCoefficients{i,1}, filterCoefficients{i,2}] = butter(3, [passBands(i,1), passBands(i, 2)]/(0.5\*Fs));

end

end

% apply each zero phase filter to the raw data

for i = 1:size(passBands, 1)

filteredData(i, :) =filtfilt(filterCoefficients{i,1},filterCoefficients{i,2},y);

end

% plot each pass band

for i = 1:size(passBands, 1)

try

plot(plot1, x/Fs, filteredData(i,:), 'Color', colors(i,:));

catch

continue

end

end

y2 = sum(filteredData, 1);

% plot the raw trace

desFilt = designfilt('bandstopiir','FilterOrder',2, ...

'HalfPowerFrequency1',55,'HalfPowerFrequency2',65, ...

'DesignMethod','butter','SampleRate',Fs);

yFilt = filtfilt(desFilt,y);

plot(plot1, x/Fs, yFilt, 'Color', 'black');

%changed 8/29/2022

%legend(legendLabels);

ylabel('Potential (ƒÊV)')

%% Lower plot

% set up axes

plot2 = subplot(2, 1, 2);

set(plot2, 'OuterPosition', [0, 0, 1, 0.70]);

% set up 100 log-spaced scales to calculate wsst over

scales = logspace(0, log10(Fs), 100);

% compute cfs, magnitude of the potential

[cfs,frequencies] = wsst(y2,Fs);

% take the real part

c=real(cfs);

% convert color to log scale

% sensitivity default 100

% increase sensitivity to 1000 for low sensitivity data

sensitivityMag = 100;

Vmax=max(max(abs(real(c))));

c1=c/Vmax;

negIndex=find(c1<-1/sensitivityMag);

posIndex=find(c1> 1/sensitivityMag);

zerIndex=find(-1/sensitivityMag <= c1 & c1 <= 1/sensitivityMag);

c1(negIndex)=-log10(sensitivityMag\*abs(c1(negIndex)));

c1(posIndex)=log10(sensitivityMag\*abs(c1(posIndex)));

c1(zerIndex)=0;

% make color plot

pc = pcolor(plot2, x/Fs,frequencies,c1/2);

grid on; shading interp;

set(pc, 'EdgeColor', 'none');

os = get(gca, 'Position');

% set up jet edit colorscale (center green changed to white)

ncol=128;

%reduce the green

deleteCols = 2;

shiftCol = 1;

ColorInit=jet(ncol);

%reducewhite[2,1]

jetEdit=[ColorInit(1:ncol/2-deleteCols+shiftCol,:);repmat([1 1 1], [2,1]);ColorInit(ncol/2+deleteCols+shiftCol:ncol,:)];

% set color scale to jetedit

colormap(jetEdit);

% add colorbar

cbh=colorbar;

set(gca, 'Position', os);

xlabel(cbh,sprintf('\\muV'));

% create array of y-tick labels

ytickArray = 1.25;

ylimits = ylim(gca);

while ytickArray(end) <= ylimits(2)/2

ytickArray(end+1) = ytickArray(end)\*2;

end

plot2.YTick = ytickArray;

set(gca, 'YScale', 'log');

ylim([1.0, 120]);

%fit the graph above and below

xl = xlim;

xlim(plot1, xl)

% change labels

xlabel('Time (s)', 'FontSize', 12, 'FontWeight', 'bold');

ylabel('Frequency (Hz)', 'FontSize', 12, 'FontWeight', 'bold');

end

**wsst.m**

function [sst,f] = wsst(x,varargin)

%Wavelet Synchrosqueezed Transform

% SST = wsst(X) returns the wavelet synchrosqueezed transform for the 1-D

% real-valued signal, X. X must have at least 4 samples. The

% synchrosqueezed transform uses 32 voices per octave and the number of

% octaves is floor(log2(numel(X)))-1. The transform uses the analytic

% Morlet wavelet by default. SST is a Na-by-N matrix where Na is the

% number of scales, 32\*(floor(log2(numel(X)))-1), and N is the number of

% samples in X.

%

% [SST,F] = wsst(X) returns a vector of frequencies, F, in cycles/sample

% corresponding to the rows of SST.

%

% [...] = wsst(X,Fs) specifies the sampling frequency, Fs, in hertz as a

% positive scalar. If you specify the sampling frequency, WSST returns

% the frequencies in hertz.

%

% [...] = wsst(X,Ts) uses the positive <a href="matlab:help duration">duration</a>, Ts, to compute the

% scale-to-frequency conversion, F. Ts is the time between samples of X.

% F has units of cycles/unit time where the unit of time is the same time

% unit as the duration.

%

% [...] = wsst(...,WAV) uses the analytic wavelet specified by WAV to

% compute the synchrosqueezed transform. Valid choices for WAV are

% 'amor' and 'bump' for the analytic Morlet and bump wavelet. If

% unspecified, WAV defaults to 'amor'.

%

% [...] = wsst(...,'VoicesPerOctave',NV) specifies the number of voices

% per octave as a positive even integer between 10 and 48. The number of

% scales is the product of the number of voices per octave and the number

% of octaves. If unspecified, NV defaults to 32 voices per octave. You

% can specify the 'VoicesPerOctave' name-value pair anywhere in the input

% argument list after the signal X.

%

% [...] = wsst(...,'ExtendSignal',EXTENDFLAG) specifies whether to

% symmetrically extend the signal by reflection to mitigate boundary

% effects. EXTENDFLAG can be one of the following options [ true |

% {false}]. If unspecified, EXTENDSIGNAL defaults to false. You can

% specify the 'ExtendSignal' name-value pair anywhere in the input

% argument list after the signal X.

%

% wsst(...) with no output arguments plots the wavelet synchrosqueezed

% transform as a function of time and frequency. If you do not specify a

% sampling frequency or interval, the synchrosqueezed transform is

% plotted in cycles/sample. If you supply a sampling frequency, Fs, the

% synchrosqueezed transform is plotted in hertz. If you supply a <a href="matlab:help duration">duration</a>

% as a sampling interval, the synchrosqueezed transform is plotted

% in cycles/unit time where the time unit is the same as the duration.

%

% % Example 1:

% % Obtain the wavelet synchrosqueezed transform of a quadratic chirp.

% % The chirp is sampled at 1000 Hz.

% load quadchirp;

% [sst,f] = wsst(quadchirp,1000);

% hp = pcolor(tquad,f,abs(sst));

% hp.EdgeColor = 'none';

% title('Wavelet Synchrosqueezed Transform');

% xlabel('Time'); ylabel('Hz');

%

% % Example 2:

% % Obtain the wavelet synchrosqueezed transform of the sunspot

% % data. Specify the sampling interval to be 1 for one sample per

% % year.

% load sunspot.dat;

% wsst(sunspot(:,2),years(1))

%

% See also iwsst, wsstridge, duration

% Copyright 2016-2018 The MathWorks, Inc.

narginchk(1,8);

nbSamp = numel(x);

x = x(:)';

validateattributes(x,{'double'},{'row','finite','real'},'wsst','X');

if numel(x)<4

error(message('Wavelet:synchrosqueezed:NumInputSamples'));

end

params = parseinputs(nbSamp,varargin{:});

nv = params.nv;

noct = params.noct;

% Create scale vector

na = noct\*params.nv;

% If sampling frequency is specified, dt = 1/fs

if (isempty(params.fs) && isempty(params.Ts))

% The default is 1 for normalized frequency

dt = params.dt;

Units = '';

elseif (~isempty(params.fs) && isempty(params.Ts))

% Accept the sampling frequency in hertz

fs = params.fs;

dt = 1/fs;

Units = '';

elseif (isempty(params.fs) && ~isempty(params.Ts))

% Get the dt and Units from the duration object

[dt,Units] = getDurationandUnits(params.Ts);

end

a0 = 2^(1/nv);

scales = a0.^(1:na);

NbSc = numel(scales);

% Construct time series to analyze, pad if necessary

meanSIG = mean(x);

x = x - meanSIG;

NumExten = 0;

if params.pad

%Pad the time series symmetrically

np2 = nextpow2(nbSamp);

NumExten = 2^np2-nbSamp;

x = wextend('1d','symw',x,NumExten,'b');

end

%Record data length plus any extension

N = numel(x);

%Create frequency vector for CWT computation

omega = (1:fix(N/2));

omega = omega.\*((2.\*pi)/N);

omega = [0., omega, -omega(fix((N-1)/2):-1:1)];

% Compute FFT of the (padded) time series

xdft = fft(x);

[psift,dpsift] = sstwaveft(params.WAV,omega,scales,params.wavparam);

%Obtain CWT coefficients and derivative

cwtcfs = ifft(repmat(xdft,NbSc,1).\*psift,[],2);

dcwtcfs = ifft(repmat(xdft,NbSc,1).\*dpsift,[],2);

%Remove padding if any

cwtcfs = cwtcfs(:,NumExten+1:end-NumExten);

dcwtcfs = dcwtcfs(:,NumExten+1:end-NumExten);

%Compute the phase transform

phasetf = imag(dcwtcfs./cwtcfs)./(2\*pi);

% Threshold for synchrosqueezing

phasetf(abs(phasetf)<params.thr) = NaN;

% Create frequency vector for output

log2Nyquist = log2(1/(2\*dt));

log2Fund = log2(1/(nbSamp\*dt));

freq = 2.^linspace(log2Fund,log2Nyquist,na);

Tx = 1/nv\*sstalgo(cwtcfs,phasetf,params.thr);

if (nargout == 0)

plotsst(Tx,freq,dt,params.engunitflag,params.normalizedfreq,Units);

else

sst = Tx;

f = freq;

end

%-------------------------------------------------------------------------

function [wft,dwft] = sstwaveft(WAV,omega,scales,wavparam)

% Admissible wavelets are:

% - MORLET wavelet (A) - 'morl':

% PSI\_HAT(s) = exp(-(s-s0).^2/2) \* (s>0)

% Parameter: s0, default s0 = 6.

% - Bump wavelet: 'bump':

% PSI\_HAT(s) = exp(1-(1/((s-mu)^2./sigma^2))).\*(abs((s-mu)/sigma)<1)

% Parameters: mu,sigma.

% default: mu=5, sigma = 0.6.

% Normalized to have unit magnitude at the peak frequency of the wavelet

NbSc = numel(scales);

NbFrq = numel(omega);

wft = zeros(NbSc,NbFrq);

switch WAV

case 'amor'

cf = wavparam;

for jj = 1:NbSc

expnt = -(scales(jj).\*omega - cf).^2/2.\*(omega > 0);

wft(jj,:) = exp(expnt).\*(omega > 0);

end

case 'bump'

mu = wavparam(1);

sigma = wavparam(2);

for jj = 1:NbSc

w = (scales(jj)\*omega-mu)./sigma;

expnt = -1./(1-w.^2);

daughter = exp(1)\*exp(expnt).\*(abs(w)<1-eps(1));

daughter(isnan(daughter)) = 0;

wft(jj,:) = daughter;

end

end

%Compute derivative

omegaMatrix = repmat(omega,NbSc,1);

dwft = 1j\*omegaMatrix.\*wft;

%-------------------------------------------------------------------------

function plotsst(Tx,F,dt,engunitflag,isfreqnormalized,Units)

if ~isempty(Units)

freqUnits = Units(1:end-1);

end

t = 0:dt:(size(Tx,2)\*dt)-dt;

if engunitflag && isfreqnormalized

frequnitstrs = wgetfrequnitstrs;

freqlbl = frequnitstrs{1};

xlbl = 'Samples';

elseif engunitflag && ~isfreqnormalized

[F,~,uf] = engunits(F,'unicode');

freqlbl = wgetfreqlbl([uf 'Hz']);

[t,~,ut] = engunits(t,'unicode','time');

xlbl = [getString(message('Wavelet:getfrequnitstrs:Time')) ' (' ut ')'];

else

freqlbl = getString(message('Wavelet:synchrosqueezed:FreqLabel'));

freqlbl = ...

[freqlbl '/' freqUnits ')'];

xlbl = getString(message('Wavelet:synchrosqueezed:Time'));

xlbl = [xlbl ' (' Units ')'];

end

h = pcolor(t,F,abs(Tx));

h.EdgeColor = 'none';

shading interp;

ylabel(freqlbl); xlabel(xlbl);

title(getString(message('Wavelet:synchrosqueezed:SynchrosqueezedTitle')));

%-------------------------------------------------------------------------

function params = parseinputs(nbSamp,varargin)

% Set defaults.

params.fs = [];

params.dt = 1;

params.Ts = [];

params.sampinterval = false;

params.engunitflag = true;

params.WAV = 'amor';

params.wavparam = 6;

params.thr = 1e-8;

params.nv = 32;

params.noct = floor(log2(nbSamp))-1;

params.pad = false;

params.normalizedfreq = true;

[varargin{:}] = convertStringsToChars(varargin{:});

% Error out if there are any calendar duration objects

tfcalendarDuration = cellfun(@iscalendarduration,varargin);

if any(tfcalendarDuration)

error(message('Wavelet:FunctionInput:CalendarDurationSupport'));

end

tfsampinterval = cellfun(@isduration,varargin);

if (any(tfsampinterval) && nnz(tfsampinterval) == 1)

params.sampinterval = true;

params.Ts = varargin{tfsampinterval>0};

if (numel(params.Ts) ~= 1 ) || params.Ts <= 0 || isempty(params.Ts)

error(message('Wavelet:FunctionInput:PositiveScalarDuration'));

end

params.engunitflag = false;

params.normalizedfreq = false;

varargin(tfsampinterval) = [];

end

%Look for Name-Value pairs

numvoices = find(strncmpi('voicesperoctave',varargin,1));

if any(numvoices)

params.nv = varargin{numvoices+1};

%validate the value is logical

validateattributes(params.nv,{'numeric'},{'positive','scalar',...

'even','>=',10,'<=',48},'wsst','VoicesPerOctave');

varargin(numvoices:numvoices+1) = [];

if isempty(varargin)

return;

end

end

extendsignal = find(strncmpi('extendsignal',varargin,1));

if any(extendsignal)

params.pad = varargin{extendsignal+1};

if ~isequal(params.pad,logical(params.pad))

error(message('Wavelet:FunctionInput:Logical'));

end

varargin(extendsignal:extendsignal+1) = [];

if isempty(varargin)

return;

end

end

% Only scalar left must be sampling frequency or sampling interval

% Only scalar left must be sampling frequency

tfsampfreq = cellfun(@(x) (isscalar(x) && isnumeric(x)),varargin);

if (any(tfsampfreq) && (nnz(tfsampfreq) == 1) && ~params.sampinterval)

params.fs = varargin{tfsampfreq};

validateattributes(params.fs,{'numeric'},{'positive'},'wsst','Fs');

params.normalizedfreq = false;

params.engunits = true;

elseif any(tfsampfreq) && params.sampinterval

error(message('Wavelet:FunctionInput:SamplingIntervalOrDuration'));

elseif nnz(tfsampfreq)>1

error(message('Wavelet:FunctionInput:Invalid\_ScalNum'));

end

%Only char variable left must be wavelet

tfwav = cellfun(@(x)ischar(x),varargin);

if (nnz(tfwav) == 1)

params.WAV = varargin{tfwav>0};

params.WAV = validatestring(params.WAV,{'bump','amor'},'wsst','WAV');

elseif nnz(tfwav)>1

error(message('Wavelet:FunctionInput:InvalidChar'));

end

if strncmpi(params.WAV,'bump',1)

params.wavparam = [5 1];

end

%------------------------------------------------------------------------

function Tx = sstalgo(cwtcfs,phasetf,gamma)

M = size(cwtcfs,1);

N = size(cwtcfs,2);

log2Fund = log2(1/N);

log2Nyquist = log2(1/2);

iRow = real(1 + floor(M/(log2Nyquist-log2Fund)\*(log2(phasetf)-log2Fund)));

idxphasetf = find(iRow>0 & iRow<=M & ~isnan(iRow));

idxcwtcfs = find(abs(cwtcfs)>gamma);

idx = intersect(idxphasetf,idxcwtcfs);

iCol = repmat(1:N,M,1);

Tx = accumarray([iRow(idx) iCol(idx)],cwtcfs(idx),size(cwtcfs));