Matlab Functions

This folder includes the Matlab codes used in the generation of Figure 2,3,5 in the complexity toolbox paper by Li, Fabus, and Sleigh.

The main functions are generate\_Fig2.m, generate\_Fig3.m, generate\_Fig5A\_H.m, generate\_Fig5I\_N.m. The other functions are subfunctions required by one of the main functions.

The EEG data used in these examples, which are not provided here, were from (Mashour et al. eLife 2021;10:e59525) and (Vlisides et al. Anesthesiology 2017, Vol. 127, 58â€“69).

Duan Li, 12/20/2021

function [aec, surro\_aec\_mean]=cal\_aec(x,Nch,fs,Nsurro)

% calculate amplitude envelope correlation (AEC)

% Input:

% x: signal of channels\*samples

% Nch: num of channels

% fs: sampling frequency

% Nsurro: num of surrogate time series

% output:

% aec: raw AEC value

% surro\_wpli\_mean: mean of the AEC values from surrogate time series

% Nov 17, 2021, by Duan Li

aec = zeros(Nch,Nch);

surro\_aec\_mean = zeros(Nch,Nch);

surro\_aec = zeros(Nch,Nch,Nsurro);

epoch=length(x);

% random shifting (adapted from Canolty et al. Science. 2006; 313(5793): 1626–1628)

numpoints = epoch;

minskip=min(fs,numpoints/10); %The min amount to shift amplitude.

maxskip=numpoints-minskip; %The max amount to shift amplitude.

skip = ceil(numpoints.\*rand(Nsurro\*3,1)); %Create a list of amplitude shifts.

skip(find(skip > maxskip))=[]; %Make sure they're not too big.

skip(find(skip < minskip))=[]; %Or too small.

skip = skip(1:Nsurro,1);

x=x';

for cc1=1:Nch

for cc2=cc1+1:Nch

% Leakage Reduction

beta\_leak = pinv(x(:,cc1))\*x(:,cc2);

xc = x(:,cc2) - x(:,cc1)\*beta\_leak;

env = abs(hilbert([x(:,cc1),xc]));

temp=corr(env);

aec(cc1,cc2) = temp(1,2);

for nn=1:Nsurro

surro\_env = abs(hilbert([x(:,cc1),xc([skip(nn):end,1:skip(nn)-1],1)]));

temp=corr(surro\_env);

surro\_aec(cc1,cc2,nn)= temp(1,2);

end

end

end

surro\_aec\_mean = squeeze(mean(surro\_aec,3));

function [wpli, surro\_wpli\_mean]=cal\_wpli(x,Nch,fs,Nsurro)

% calculate debiased weighted phase lag index (wPLI)

% Input:

% x: signal of channels\*samples,

% Nch: num of channels

% fs: sampling frequency

% Nsurro: num of surrogate time series

% Output:

% wpli: raw wPLI

% surro\_wpli\_mean: mean of the wPLI values from surrogate time series

% Nov 17, 2021, by Duan Li

wpli = zeros(Nch,Nch);

surro\_wpli\_mean = zeros(Nch,Nch);

surro\_wpli = zeros(Nch,Nch,Nsurro);

epoch=length(x);

Xspectra=zeros(epoch,Nch,Nch);% CrossSpectra: samples\*channels\*channels

x\_c=hilbert(x').';

for cc1=1:Nch

for cc2=cc1+1:Nch

Xspectra(:,cc1,cc2)=x\_c(cc1,:).\*conj(x\_c(cc2,:));

end

end

wpli=cal\_debiased\_wpli(Xspectra);% connectivity index: channels\*channels

% random shifting (adapted from Canolty et al. Science. 2006; 313(5793): 1626–1628)

numpoints = epoch;

minskip=min(fs,numpoints/10); %The min amount to shift amplitude.

maxskip=numpoints-minskip; %The max amount to shift amplitude.

skip = ceil(numpoints.\*rand(Nsurro\*3,1)); %Create a list of amplitude shifts.

skip(find(skip > maxskip))=[]; %Make sure they're not too big.

skip(find(skip < minskip))=[]; %Or too small.

skip = skip(1:Nsurro,1);

for nn=1:Nsurro

surro\_Xspectra=zeros(epoch,Nch,Nch);

for cc1=1:Nch

for cc2=cc1+1:Nch

surro\_Xspectra(:,cc1,cc2)=x\_c(cc1,:).\*conj(x\_c(cc2,[skip(nn):end,1:skip(nn)-1]));

end

end

surro\_wpli(:,:,nn)=cal\_debiased\_wpli(surro\_Xspectra);% channels\*channels\*surros

clear surro\_Xspectra

end

surro\_wpli\_mean = squeeze(mean(surro\_wpli,3));

function [debiased\_wpli, wpli]=cal\_debiased\_wpli(Xspectra)

% adapted from Fieldtrip toolbox

Im\_Xspectra=imag(Xspectra);

outsum = nansum(Im\_Xspectra,1);

outsumW = nansum(abs(Im\_Xspectra),1);

wpli = squeeze(abs(outsum)./outsumW);

debiasfactor = nansum(Im\_Xspectra.^2,1);

debiased\_wpli= squeeze((outsum.^2 - debiasfactor)./(outsumW.^2 - debiasfactor));

function List=construct\_list(symb\_num,L)

%% construct\_list %%

% input %

% symb\_num: number of symbols (either 2 or 3 is allowed) (ex) 2 --> binary

% L: word length (embedding dim.)

% output %

% List: list of words (patterns)

% 2016. 2. 1. Jisung Wang

%%

List=zeros(1,L);

for i=1:L

nch=nchoosek(1:L,i);

num\_nch=size(nch,1);

org=zeros(num\_nch,L);

l=1;

for k=1:symb\_num-1

if k==1

for j=1:num\_nch

org(j,nch(j,:))=k;

end

List=[List;org];

else

list=zeros(num\_nch\*(2^i-1),L);

for j=1:num\_nch

for z=1:i

nch\_z=nchoosek(1:i,z);

for y=1:size(nch\_z,1)

Org=org(j,:);

Org(1,nch(j,nch\_z(y,:)))=k;

list(l,:)=Org;

l=l+1;

end

end

end

List=[List;list];

end

end

end

function y=embedding(data, m, lag)

% data should be 1-d vector

% v: time lag

% m: dimension

% output is 2-d matrix

MaxEpoch=length(data);

y=zeros(MaxEpoch-lag\*(m-1),m);

for j=1:m

y(:, j)=data(1+(j-1)\*lag:end-(m-j)\*lag);

end

% this function was called by generate\_Fig2.m, and requires two subfunctions:

% embedding.m and construct\_list.m

%

% this code was kindly shared by Jisung Wang and Dr. Heonsoo Lee

% used in (Wang et al, Neuroscience Letters 2017; 653: 320-325)

%

% Duan Li, 12/20/2020

function [complexity,MIG]=fluctuation\_complexity(symb\_data, m, lag, opt\_symb\_num)

%% Fluctuation Compelxity

% input

% symb\_data: symbolized data

% m: embedding dim.

% lag: lag size

% opt\_symb\_num: (optional) # of symbols (default value is 2, binary data)

% output

% complexity: fluctuation complexity

% ref.

% 2016. 2. 2. Jisung Wang

if nargin==3

symb\_num=2;

else

symb\_num=opt\_symb\_num;

end

%% embedding

embedded=embedding(symb\_data, m, lag); % Duan added embedding.m, 10/26/2021

emb\_sz=size(embedded,1);

%% construct the pattern list, List

List=construct\_list(symb\_num, m);

num\_words=size(List, 1); % # of words

%% symbolize the embedded data, symb\_embedded

symb\_embedded=zeros(emb\_sz,1);

for i=1:num\_words

ith=sum(embedded==repmat(List(i,:), emb\_sz,1),2)==m;

symb\_embedded(ith)=i;

end

%% count occurrence of i-th words and transitions ij

transition=[symb\_embedded(1:end-1),symb\_embedded(2:end)]; trans\_sz=size(transition, 1);

p\_i=0; comp=0;

k=1;

for i=1:num\_words

%% count i-th word (probability)

ith=symb\_embedded==i;

p\_i(i)=sum(ith)/emb\_sz;

end

for i=1:num\_words

P\_i=p\_i(i);

if P\_i~=0

ij\_transition=zeros(1,num\_words);

for j=1:num\_words

ij=sum(transition==repmat([i,j], trans\_sz,1),2)==2;

ij\_transition(1,j)=sum(ij);

end

nzero=~(~ij\_transition);

p\_ij=ij\_transition(1,nzero)/emb\_sz;

P\_j=p\_i(nzero);

%%

comp(k)=sum(p\_ij.\*(log2(P\_i./P\_j)).^2);

IG(k)=-sum(p\_ij.\*log2(p\_ij./P\_i));% Duan added, 10/26/2021

k=k+1;

end

end

%%

% complexity=sqrt(sum(comp));

complexity=sum(comp);% Duan removed sqrt, 10/27/2021

MIG=sum(IG); % Duan added, 10/26/2021

% this script was used to generate Fig2 in the complexity toolbox paper by Li, Fabus, and Sleigh

%

% the following functions are required:

% fluctuation\_complexity.m: as provided in the same folder, it was adapted

% from a code kindly shared by Jisung Wang and Dr. Heonsoo Lee

% used in (Wang et al, Neuroscience Letters 2017; 653: 320-325)

%

% calc\_lz\_complexity.m: Quang Thai (2021). calc\_lz\_complexity

% (https://www.mathworks.com/matlabcentral/fileexchange/38211-calc\_lz\_complexity),

% MATLAB Central File Exchange. Retrieved October 12, 2021.

%

% pec.m: Gaoxiang Ouyang (2021). Permutation entropy

% (https://www.mathworks.com/matlabcentral/fileexchange/37289-permutation-entropy),

% MATLAB Central File Exchange. Retrieved October 12, 2021.

%

% Duan Li, 12/20/2021

clear all

close all

clc

% this section is about the EEG data, which should be modified based on your own dataset

dataname\_iso={'UM\_4';'UM\_7';'UM\_8';'UM\_9';'UM\_18';'UM\_21'};% 6 subjects

state\_label\_iso={'baseline';'isoflurane'};% 2 states

dataname\_ketamine={'kk\_01';'kk\_02';'kk\_05';'kk\_06';'kk\_07';'kk\_08';'kk\_09';'kk\_11';'kk\_12';'kk\_13';'kk\_14';'kk\_15'};% 15 subjects

state\_label\_ket={'baseline';'sub-ketamine';'ketamine'};% 3 states

fs=250; % sampling rate

selected\_channel=17;ref\_channel=12; % Fp1 - Fpz: the channel used in the analysis

% parameter setting

fmax=45;

[b,a]=butter(5, [0.5,fmax]/(fs/2));

epoch=10\*fs;% nonoverlapped 10-s epochs

m=4;% fluctuation complexity

lag=1;

dim\_PE=5;% Perumutation entropy

tau\_PE=fix(fs/2/fmax);

for pp=1:length(dataname\_iso)

load([dataname\_iso{pp}])

% data structure:

% eeg{ss}(cc,tt): ss is state, cc is channel, tt is time.

% leng: data length(4 min for each state and subject)

N\_epoch=fix(leng/epoch);

for ss=1:length(state\_label\_iso)

temp\_x\_data=eeg{ss}(selected\_channel,:)-eeg{ss}(ref\_channel,:);

x\_data\_filt=filtfilt(b,a,temp\_x\_data);

amp=abs(hilbert(x\_data\_filt));

parfor j=1:N\_epoch

eeg\_epoch{j}=x\_data\_filt((j-1)\*epoch+1:j\*epoch);

eeg\_binary{j}=double(eeg\_epoch{j}>median(eeg\_epoch{j}));

[FC0(j),MIG0(j)]=fluctuation\_complexity(eeg\_binary{j}, m, lag);

PE0(j)=pec(eeg\_epoch{j},dim\_PE,tau\_PE)/log(factorial(dim\_PE));

amp\_epoch{j}=amp((j-1)\*epoch+1:j\*epoch);

amp\_binary{j}=double(amp\_epoch{j}>mean(amp\_epoch{j}));

LZC0(j)=calc\_lz\_complexity(amp\_binary{j},'exhaustive',1);

end

FC\_iso(pp,ss)=mean(FC0);

MIG\_iso(pp,ss)=mean(MIG0);

PE\_iso(pp,ss)=mean(PE0);

LZC\_iso(pp,ss)=mean(LZC0);

end

end

for pp=1:length(dataname\_ketamine)

load([dataname\_ketamine{pp}])

% data structure:

% eeg{ss}(cc,tt): ss is state, cc is channel, tt is time.

% leng: data length (4 min for each state and subject)

N\_epoch=fix(leng/epoch);

for ss=1:length(state\_label\_ket)

temp\_x\_data=eeg{ss}(selected\_channel,:)-eeg{ss}(ref\_channel,:);

x\_data\_filt=filtfilt(b,a,temp\_x\_data);

amp=abs(hilbert(x\_data\_filt));

parfor j=1:N\_epoch

eeg\_epoch{j}=x\_data\_filt((j-1)\*epoch+1:j\*epoch);

eeg\_binary{j}=double(eeg\_epoch{j}>median(eeg\_epoch{j}));

[FC0(j),MIG0(j)]=fluctuation\_complexity(eeg\_binary{j}, m, lag);

PE0(j)=pec(eeg\_epoch{j},dim\_PE,tau\_PE)/log(factorial(dim\_PE));

amp\_epoch{j}=amp((j-1)\*epoch+1:j\*epoch);

amp\_binary{j}=double(amp\_epoch{j}>mean(amp\_epoch{j}));

LZC0(j)=calc\_lz\_complexity(amp\_binary{j},'exhaustive',1);

end

FC\_ket(pp,ss)=mean(FC0);

MIG\_ket(pp,ss)=mean(MIG0);

PE\_ket(pp,ss)=mean(PE0');

LZC\_ket(pp,ss)=mean(LZC0);

end

end

fig1 = figure('Position',[50 50 1600 400]);set(gcf,'Renderer','zbuffer','color','w');

subplot(1,3,1)

plot(MIG\_iso(:,1),PE\_iso(:,1),'^','MarkerFaceColor','g','MarkerEdgeColor','k','MarkerSize',8);hold on

plot(MIG\_iso(:,2),PE\_iso(:,2),'o','MarkerFaceColor','r','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,2),PE\_ket(:,2),'d','MarkerFaceColor','y','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,3),PE\_ket(:,3),'s','MarkerFaceColor','b','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,1),PE\_ket(:,1),'^','MarkerFaceColor','g','MarkerEdgeColor','k','MarkerSize',8);

xlabel('MIG'),ylabel('PE')

legend('baseline','isoflurane','subanesthetic ketamine','anesthetic ketamine')

subplot(1,3,2)

plot(MIG\_iso(:,1),LZC\_iso(:,1),'^','MarkerFaceColor','g','MarkerEdgeColor','k','MarkerSize',8);hold on

plot(MIG\_iso(:,2),LZC\_iso(:,2),'o','MarkerFaceColor','r','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,2),LZC\_ket(:,2),'d','MarkerFaceColor','y','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,3),LZC\_ket(:,3),'s','MarkerFaceColor','b','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,1),LZC\_ket(:,1),'^','MarkerFaceColor','g','MarkerEdgeColor','k','MarkerSize',8);

xlabel('MIG'),ylabel('LZC')

subplot(1,3,3)

plot(MIG\_iso(:,1),FC\_iso(:,1),'^','MarkerFaceColor','g','MarkerEdgeColor','k','MarkerSize',8);hold on

plot(MIG\_iso(:,2),FC\_iso(:,2),'o','MarkerFaceColor','r','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,2),FC\_ket(:,2),'d','MarkerFaceColor','y','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,3),FC\_ket(:,3),'s','MarkerFaceColor','b','MarkerEdgeColor','k','MarkerSize',8);

plot(MIG\_ket(:,1),FC\_ket(:,1),'^','MarkerFaceColor','g','MarkerEdgeColor','k','MarkerSize',8);

xlabel('MIG'),ylabel('FC')

% this script was used to generate Fig3 in the complexity toolbox paper by Li, Fabus, and Sleigh

%

% the following function is required:

% calc\_lz\_complexity.m: Quang Thai (2021). calc\_lz\_complexity

% (https://www.mathworks.com/matlabcentral/fileexchange/38211-calc\_lz\_complexity),

% MATLAB Central File Exchange. Retrieved October 12, 2021.

%

% Duan Li, 12/20/2021

close all

clear

clc

% simulated signals

fs=100;

f\_signal=10;

t = 0:1/fs:0.2;

x(1,:) = sin(2\*pi\*f\_signal\*t+pi/6);

x(2,:) = awgn(x(1,:),-6,'measured',100)/4;

for i=1:size(x,1)

Thr(i)=mean(x(i,:));

x\_binary(i,:)=double(x(i,:)>Thr(i));

[LZ(i),H{i},~]=calc\_lz\_complexity(x\_binary(i,:),'exhaustive',0);

end

fig1 = figure('Position',[50 50 900 400]);set(gcf,'Renderer','zbuffer','color','w');

for i=1:size(x,1)

subplot(3,2,i),plot(t,x(i,:),'ko-'),ylim([-1.5,1]),axis off

subplot(3,2,i+2),plot(t,x(i,:),'ko-'),hold on,

plot(t,Thr(i)\*ones(1,length(t)),'b'),ylim([-1.5,1]),axis off

for j=1:length(t)

plot(t(j)\*[1,1],[-1.5,x(i,j)],'k:')

end

subplot(3,2,i+4),

plot(t,x\_binary(i,:),'b.'),

xlabel('time(s)'),ylabel('amplitude'),axis off

title([strcat('LZC=',num2str(LZ(i),'%1.3f'))])

end

% this script was used to generate Fig5 A-H in the complexity toolbox paper by Li, Fabus, and Sleigh

%

% Duan Li, 12/20/2021

clear all

close all

clc

% this section is about the EEG data, which should be modified based on your own data

load(['UM\_4\_refLaplacian'])% data structure:

% eeg{ss}(cc,tt): ss is state, cc is channel, tt is time.

fs=250;

selected\_channels=[19,46];% F3, P3

% paramter setting

[b1,a1]=butter(5, 0.5/(fs/2),'high');% used for showing the original signal (remove baseline drift, etc.)

band=[8,13];% alpha band

[b,a]=butter(5, [band(1)/(fs/2),band(2)/(fs/2)]);

for cc=1:2

x0(cc,:)=filtfilt(b1,a1,eeg{1}(selected\_channels(cc),:));

x\_filt0(cc,:)=filtfilt(b,a,x0(cc,:));% band-pass filtering

% extract 30 sec data in the middle for the analysis, avoiding possible edge effect

x(cc,:)=x0(cc,30\*fs+1:60\*fs);

x\_filt(cc,:)=x\_filt0(cc,30\*fs+1:60\*fs);

xc(cc,:)=hilbert(x\_filt(cc,:));

x\_phase(cc,:)=angle(xc(cc,:));

x\_envelope(cc,:) = abs(xc(cc,:));

end

% wPLI

Xspectra=xc(1,:).\*conj(xc(2,:)); % cross spectrum

imag\_Xspectra=imag(Xspectra);

phase\_diff=angle(Xspectra);

N\_phase\_bins=72;

for i=1:N\_phase\_bins

temp\_index=find(phase\_diff>=-pi+2\*pi/N\_phase\_bins\*(i-1) & phase\_diff<-pi+2\*pi/N\_phase\_bins\*i);

phase0(i)=-pi+2\*pi/N\_phase\_bins\*(i-1+0.5);

mean\_mag(i)=mean(abs(imag\_Xspectra(temp\_index)));

end

color\_cc=[0, 0.4470, 0.7410;0.8500, 0.3250, 0.0980];

tt=[1:length(x)]/fs;

fig1 = figure('Position',[50 50 1500 450]);set(gcf,'Renderer','zbuffer','color','w');

for cc=1:2

subplot(3,3,1),

plot(tt,x(cc,:)-100\*(cc-1),'color',color\_cc(cc,:)),hold on

xlim([0,2]),ylim([-200,100]),%axis off

subplot(3,3,4)

plot(tt,x\_filt(cc,:)-100\*(cc-1),'color',color\_cc(cc,:)),hold on

xlim([0,2]),ylim([-200,100]),%axis off

subplot(3,3,7),plot(tt,x\_phase(cc,:)-2\*pi\*(cc-1),'color',color\_cc(cc,:)),hold on

xlim([0,2]),ylim([-3.5\*pi,1.5\*pi]),%axis off

end

subplot(3,3,5),plot(tt,imag\_Xspectra,'k'),xlim([0,2]),ylim([-2000,1100])

hold on,plot([0,2],[0,0],'Color',[150,150,150]/255)

subplot(3,3,8),plot(tt,phase\_diff,'color',[0, 0.5, 0]),xlim([0,2]),ylim([-3.5\*pi,1.5\*pi])

hold on,plot([0,2],[0,0],'Color',[150,150,150]/255)

subplot(1,3,3),polarhistogram(phase\_diff,'BinWidth',2\*pi/N\_phase\_bins,'EdgeColor',[0, 0.5, 0],'FaceColor',[0, 0.5, 0])

hold on,polarplot(phase0,mean\_mag/1.6,'k')

% AEC

fig1 = figure('Position',[50 50 1000 450]);set(gcf,'Renderer','zbuffer','color','w');

for cc=1:2

subplot(3,2,1)

plot(tt,x\_filt(cc,:)-100\*(cc-1),'color',color\_cc(cc,:)),hold on

xlim([0,2]),ylim([-200,100]),%axis off

subplot(3,2,3)

plot(tt,x\_filt(cc,:)-100\*(cc-1),'Color',[150,150,150]/255),hold on

plot(tt,x\_envelope(cc,:)-100\*(cc-1),'color',color\_cc(cc,:)),

xlim([0,2]),ylim([-200,100]),%axis off

end

subplot(1,2,2),plot(x\_envelope(1,:),x\_envelope(2,:),'k.','MarkerSize',3),hold on

mdlr = fitlm(x\_envelope(1,:),x\_envelope(2,:),'RobustOpts','on');

slope=mdlr.Coefficients{2,1};

plot(x\_envelope(1,:),mdlr.Coefficients{1,1}+mdlr.Coefficients{2,1}\*x\_envelope(1,:),'color',[0, 0.5, 0],'LineWidth',2)

% this script was used to generate Fig5 I-N in the complexity toolbox paper by Li, Fabus, and Sleigh

% it requires two subfunctions:cal\_wpli.m and cal\_aec.m, as provided in the same folder

%

% Duan Li, 12/20/2021

clear all

close all

clc

% this section is about the EEG data, which should be modified based on your own data

load(['UM\_4\_refLaplacian'])% data structure:

% eeg{ss}(cc,tt): ss is state, cc is channel, tt is time.

% leng: data length(4 min for each state and subject)

N\_state=2;

state\_label={'baseline';'anesthesia'};

nch=90;% number of channels

fs=250;% sampling rate

% parameter setting

band=[8,13];% alpha

[b,a]=butter(5, [band(1)/(fs/2),band(2)/(fs/2)]);

epoch=10\*fs;% nonovelapped 10s epochs

N\_epoch=fix(leng/epoch);

Nsurro=20; % number of surrogate time series

for ss=1:N\_state

x\_data\_filt=filtfilt(b,a,eeg{ss}')';% band-pass filtering

for j=1:N\_epoch

eeg\_epoch{j}=x\_data\_filt(:,(j-1)\*epoch+1:j\*epoch);

end

parfor j=1:N\_epoch

[temp\_wpli{j},temp\_wpli\_surro\_m{j}]=cal\_wpli(eeg\_epoch{j},nch,fs,Nsurro);

[temp\_aec{j},temp\_aec\_surro\_m{j}]=cal\_aec(eeg\_epoch{j},nch,fs,Nsurro);

end

for j=1:N\_epoch

temp=temp\_wpli{j}-temp\_wpli\_surro\_m{j};% significant wPLI = raw wPLI - mean wPLI of surrogate data

temp(temp<0)=0;

wpli\_sig{ss}(:,:,j)=temp;%channels\*channels\*epochs(time)

clear temp

temp=temp\_aec{j}-temp\_aec\_surro\_m{j};

temp(temp<0)=0;

aec\_sig{ss}(:,:,j)=temp;

clear temp

end

for cc1=1:nch

wpli\_sig{ss}(cc1,cc1,:)=1;% diagonal line

aec\_sig{ss}(cc1,cc1,:)=1;

for cc2=cc1+1:nch

wpli\_sig{ss}(cc2,cc1,:)=wpli\_sig{ss}(cc1,cc2,:,:);

aec\_sig{ss}(cc2,cc1,:)=aec\_sig{ss}(cc1,cc2,:,:);

end

end

clear x\_data\_filt eeg\_epoch temp\_wpli temp\_wpli\_surro\_m temp\_aec temp\_aec\_surro\_m

wpli\_mtx{ss}=squeeze(mean(wpli\_sig{ss},3));% average over time

aec\_mtx{ss}=squeeze(mean(aec\_sig{ss},3));

% SVD

wpli\_eig\_val{ss}=svd(wpli\_mtx{ss});

aec\_eig\_val{ss}=svd(aec\_mtx{ss});

wpli\_eig\_val{ss}=wpli\_eig\_val{ss}/sum(wpli\_eig\_val{ss}); % normalization

aec\_eig\_val{ss}=aec\_eig\_val{ss}/sum(aec\_eig\_val{ss});

wpli\_eig\_max(ss)=max(wpli\_eig\_val{ss}); % max eigenvalue

aec\_eig\_max(ss)=max(aec\_eig\_val{ss});

p=wpli\_eig\_val{ss};wpli\_eig\_En(ss)=1+sum(p.\*log(p))/log(nch);clear p % diversity of the eigenvalues

p=aec\_eig\_val{ss};aec\_eig\_En(ss)=1+sum(p.\*log(p))/log(nch);clear p

% functional complexity

temp=wpli\_mtx{ss};

temp=temp(temp<1);% exclude diagonal elements

[wpli\_pij{ss},edges] = histcounts(temp,[0:0.025:0.5],'Normalization','probability');

m=length(edges)-1;

bins=edges(1:m)+(edges(2)-edges(1))/2;

wpli\_FC(ss)=1-m/(2\*(m-1))\*sum(abs(wpli\_pij{ss}-1/m));

clear temp

temp=aec\_mtx{ss};

temp=temp(temp<1);

[aec\_pij{ss},~] = histcounts(temp,[0:0.025:0.5],'Normalization','probability');

aec\_FC(ss)=1-m/(2\*(m-1))\*sum(abs(aec\_pij{ss}-1/m));

clear temp

end

color\_state={'b';'r'};

fig1 = figure('Position',[50 50 650 600]);set(gcf,'Renderer','zbuffer');

for ss=1:N\_state

subplot(2,2,ss)

imagesc(squeeze(wpli\_mtx{ss}),[0,.6]),axis off,colormap('jet')

xlim([0.5,nch+0.5]),ylim([0.5,nch+0.5]),

subplot(2,2,3)

plot([1:nch],wpli\_eig\_val{ss},color\_state{ss}),hold on

ylim([0,0.165]),xlim([0,nch])

subplot(2,2,4)

plot(bins,wpli\_pij{ss},color\_state{ss}),hold on

ylim([0,0.4]),xlim([bins(1),bins(m)])

end

plot(bins,1/m\*ones(1,m),'k'),set(gcf,'color','w');

fig2 = figure('Position',[50 50 650 600]);set(gcf,'Renderer','zbuffer');

for ss=1:N\_state

subplot(2,2,ss)

imagesc(squeeze(aec\_mtx{ss}),[0,.4]),axis off,colormap('jet'),xlim([0.5,nch+0.5]),ylim([0.5,nch+0.5]),

subplot(2,2,3)

plot([1:nch],aec\_eig\_val{ss},color\_state{ss}),hold on,ylim([0,0.165]),xlim([0,nch])

subplot(2,2,4)

plot(bins,aec\_pij{ss},color\_state{ss}),hold on

ylim([0,0.4]),xlim([bins(1),bins(m)])

end

plot(bins,1/m\*ones(1,m),'k'),set(gcf,'color','w');

**https://gitlab.com/marcoFabus/complexity\_toolbox**

**Time-series complexity**

Complexity metrics are increasingly being used to analyse EEG signals in anaesthesia. This notebook aims to guide the interested reader through properties of signals that affect their complexity. We will see how attributes such as frequency or noise change complexity and what to keep in mind when analysing your own data or reading current literature.

We will utilize the AntroPy toolbox (<https://raphaelvallat.com/antropy/>) to compute different complexity metrics. Other alternatives include NeuroKit (<https://neurokit2.readthedocs.io/en/latest/examples/complexity.html>) in Python and user-contributed Matlab libraries (e.g. RangeEn, <https://www.mathworks.com/matlabcentral/fileexchange/69850-signal-complexity-analysis>).

For more discussion of various complexity metrics and their meaning, please see the publication associated with this notebook:

*Li, Fabus, and Sleigh (2021) - Brain complexities and anesthesia: their meaning and measurement (DOI).*

**1. Basic AntroPy functionality**

Let's start by importing relevant libraries and creating a simulated signal.

In [25]:

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** antropy **as** ant

*# Create a simple oscillatory signal with some noise*

sample\_rate **=** 200 *#Hz*

Nsec **=** 10 *#s*

f0 **=** 10 *#Hz*

time\_vect **=** np**.**linspace(0, Nsec, Nsec**\***sample\_rate)

noise **=** np**.**random**.**normal(0, 0.25, len(time\_vect))

x **=** np**.**cos(2**\***np**.**pi**\***f0**\***time\_vect) **+** noise

*# Plot signal*

plt**.**figure(figsize**=**(4, 3))

plt**.**plot(time\_vect, x)

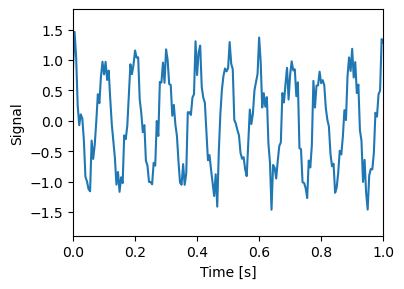
plt**.**xlabel('Time [s]')

plt**.**ylabel('Signal')

plt**.**xlim(0, 1)

Out[25]:

(0.0, 1.0)



We can evaluate different complexity metrics on this simulated signal by calling AntroPy (ant). For example, we can call different entropy metrics:

In [16]:

*# Permutation entropy*

print('PE = ', ant**.**perm\_entropy(x, normalize**=True**))

*# Approximate entropy*

print('AE = ', ant**.**app\_entropy(x))

*# Sample entropy*

print('SE = ', ant**.**sample\_entropy(x))

PE = 2.50844729877499

AE = 1.4394099549757589

SE = 1.5083970579207904

If we are interested in the Lempel-Ziv complexity, we need to binarise the signal before calling the relevant function. Here we threshold the signal by the mean value.

In [18]:

x\_bin **=** x**.**copy()**\***0

thr **=** np**.**mean(x)

x\_bin[np**.**where(x **>=** thr)] **=** 1

*# Plot signal*

plt**.**figure(figsize**=**(4, 3))

plt**.**plot(time\_vect, x, label**=**'Raw')

plt**.**plot(time\_vect, x\_bin, label**=**'Thresholded')

plt**.**xlabel('Time [s]')

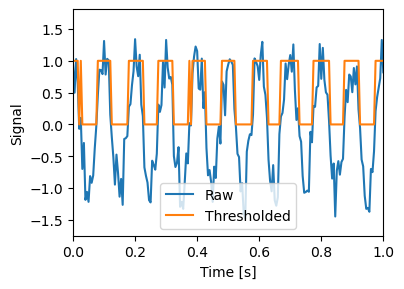
plt**.**ylabel('Signal')

plt**.**xlim(0, 1)

plt**.**legend()

Out[18]:

<matplotlib.legend.Legend at 0x7f7523b8e490>



We can now compute LZC, here normalised to account for sequence length (same as in <https://doi.org/10.1007/s10910-008-9512-2>).

In [19]:

ant**.**lziv\_complexity(x\_bin, normalize**=True**)

Out[19]:

0.3986313713864835

**2. Signal properties affecting complexity**

Now that we've seen how to use example complexity functions, let's explore how changing the signal affects its complexity.

**Signal frequency**

We will take a chirp signal, a sinusoid increasing in frequency, that sweeps between 0.5Hz and 5Hz in 40 seconds. We will take 10 second windows (9s overlap) as we track changes in permutation entropy and Lempel-Ziv complexity.

In [124]:

chirp\_complexity **=** np**.**zeros((2, 31))

freqs **=** np**.**linspace(0.5, 5, 31)

*# Define chirp signal*

Nsec **=** 40

time\_vect **=** np**.**linspace(0, Nsec, Nsec**\***sample\_rate)

f0 **=** 0.5

c **=** (5**-**f0)**/**Nsec

Xchirp **=** np**.**sin(2**\***np**.**pi**\***(f0**+**c**\***time\_vect**/**2)**\***time\_vect)

*# Split signal into overlapping windows*

x\_seg **=** np**.**lib**.**stride\_tricks**.**sliding\_window\_view(Xchirp, 10**\***sample\_rate)[::sample\_rate, :]

*# Loop over frequencies between 0.1Hz and 10Hz*

**for** i **in** range(x\_seg**.**shape[0]):

x **=** x\_seg[i, :]

PE **=** ant**.**perm\_entropy(x, normalize**=True**)

x\_bin **=** x**.**copy()**\***0

thr **=** np**.**mean(x)

x\_bin[np**.**where(x **>=** thr)] **=** 1

LZ **=** ant**.**lziv\_complexity(x\_bin, normalize**=True**)

chirp\_complexity[0, i] **=** PE

chirp\_complexity[1, i] **=** LZ

*# Plot the results*

fig, (ax1, ax3) **=** plt**.**subplots(2, 1, gridspec\_kw**=**{'height\_ratios': [3, 1]})

plt**.**subplots\_adjust(hspace**=**0.4)

color **=** 'tab:red'

ax1**.**plot(freqs, chirp\_complexity[0, :], color**=**color)

ax1**.**set\_ylabel('Permutation Entropy', color**=**color)

ax1**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax1**.**set\_xlabel('Frequency [Hz]')

color **=** 'tab:blue'

ax2 **=** ax1**.**twinx()

ax2**.**plot(freqs, chirp\_complexity[1, :], color**=**color)

ax2**.**set\_ylabel('Lempel-Ziv complexity', color**=**color)

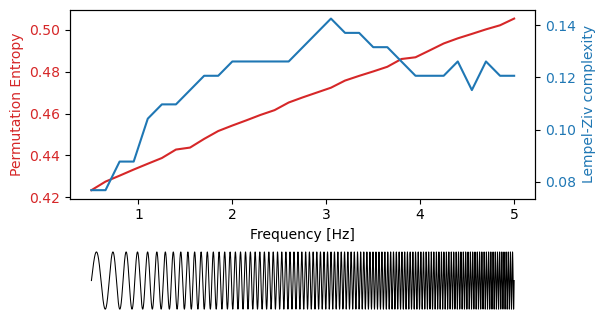
ax2**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax3**.**plot(time\_vect, Xchirp, lw**=**0.75, color**=**'k')

ax3**.**axis('off')

Out[124]:

(-2.0, 42.0, -1.0999998332451333, 1.0999999357359773)



We can see both measures increase as frequency increases, PE almost linearly and LZC in a more complicated fashion.

Let's try the same thing for an oscillatory signal with noise of increasing amplitude (i.e. signal with decreasing signal-to-noise ratio).

In [126]:

snr\_complexity **=** np**.**zeros((2, 31))

noise\_rms **=** np**.**linspace(0., 0.25, 31)

*# Define signal with linearly increasing noise level*

Nsec **=** 40

time\_vect **=** np**.**linspace(0, Nsec, Nsec**\***sample\_rate)

f0 **=** 0.5

Xsnr **=** np**.**sin(2**\***np**.**pi**\***f0**\***time\_vect)

noise **=** np**.**random**.**normal(0, 0.25, Nsec**\***sample\_rate)

noise **\*=** np**.**linspace(0, 1, Nsec**\***sample\_rate)

Xsnr **+=** noise

*# Split signal into overlapping windows*

x\_seg **=** np**.**lib**.**stride\_tricks**.**sliding\_window\_view(Xsnr, 10**\***sample\_rate)[::sample\_rate, :]

*# Loop over noise levels*

**for** i **in** range(x\_seg**.**shape[0]):

x **=** x\_seg[i, :]

PE **=** ant**.**perm\_entropy(x, normalize**=True**)

x\_bin **=** x**.**copy()**\***0

thr **=** np**.**mean(x)

x\_bin[np**.**where(x **>=** thr)] **=** 1

LZ **=** ant**.**lziv\_complexity(x\_bin, normalize**=True**)

snr\_complexity[0, i] **=** PE

snr\_complexity[1, i] **=** LZ

*# Plot the results*

fig, (ax1, ax3) **=** plt**.**subplots(2, 1, gridspec\_kw**=**{'height\_ratios': [3, 1]})

plt**.**subplots\_adjust(hspace**=**0.4)

color **=** 'tab:red'

ax1**.**plot(noise\_rms, snr\_complexity[0, :], color**=**color)

ax1**.**set\_ylabel('Permutation Entropy', color**=**color)

ax1**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax1**.**set\_xlabel('Noise level')

ax1**.**set\_ylim(0.4, 1.05)

color **=** 'tab:blue'

ax2 **=** ax1**.**twinx()

ax2**.**plot(noise\_rms, snr\_complexity[1, :], color**=**color)

ax2**.**set\_ylabel('Lempel-Ziv complexity', color**=**color)

ax2**.**tick\_params(axis**=**'y', labelcolor**=**color)

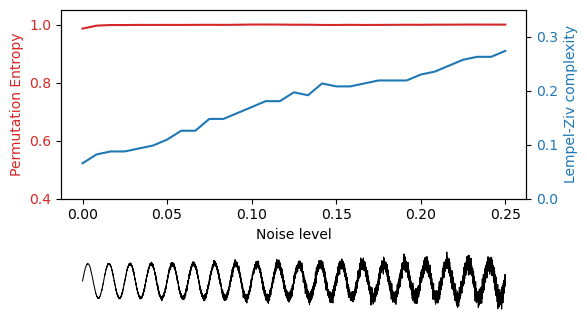
ax2**.**set\_ylim(0, 0.35)

ax3**.**plot(time\_vect, Xsnr, lw**=**0.75, color**=**'k')

ax3**.**axis('off')

Out[126]:

(-2.0, 42.0, -1.8125439851901426, 1.8525520554485144)



LZ increases nearly linearly with increasing noise level (5-fold increase!), whereas permutation entropy is almost unchanged.

**Waveform shape**

Finally, consider a waveform which gets progressively more non-sinusoidal as harmonics are added to it.

In [116]:

wf\_complexity **=** np**.**zeros((2, 31))

nharm **=** np**.**arange(31)

*# Define signal with progressively more harmonics*

Nsec **=** 40

time\_vect **=** np**.**linspace(0, Nsec, Nsec**\***sample\_rate)

f0 **=** 0.5

X0 **=** np**.**sin(2**\***np**.**pi**\***f0**\***time\_vect)

X1 **=** 1**/**3 **\*** np**.**sin(2**\***np**.**pi**\***3**\***f0**\***time\_vect)

X2 **=** 1**/**5 **\*** np**.**sin(2**\***np**.**pi**\***5**\***f0**\***time\_vect)

**def** square\_wave(Nharm, base):

out **=** base**.**copy()

**for** h **in** range(1, Nharm**+**1):

out **+=** 1**/**(2**\***h**+**1) **\*** np**.**sin(2**\***np**.**pi**\***(2**\***h**+**1)**\***f0**\***time\_vect)

**return** out

X **=** X0**.**copy()

**for** i **in** range(0, 40, 2):

X[i**\***sample\_rate:(i**+**2)**\***sample\_rate] **=** square\_wave(i, X0)[i**\***sample\_rate:(i**+**2)**\***sample\_rate]

*# Split signal into overlapping windows*

x\_seg **=** np**.**lib**.**stride\_tricks**.**sliding\_window\_view(X, 10**\***sample\_rate)[::sample\_rate, :]

*# Loop over signal*

**for** i **in** range(x\_seg**.**shape[0]):

x **=** x\_seg[i, :]

PE **=** ant**.**perm\_entropy(x, normalize**=True**)

x\_bin **=** x**.**copy()**\***0

thr **=** np**.**mean(x)

x\_bin[np**.**where(x **>=** thr)] **=** 1

LZ **=** ant**.**lziv\_complexity(x\_bin, normalize**=True**)

wf\_complexity[0, i] **=** PE

wf\_complexity[1, i] **=** LZ

*# Plot the results*

fig, (ax1, ax3) **=** plt**.**subplots(2, 1, gridspec\_kw**=**{'height\_ratios': [3, 1]})

plt**.**subplots\_adjust(hspace**=**0.4)

color **=** 'tab:red'

ax1**.**plot(nharm, wf\_complexity[0, :], color**=**color)

ax1**.**set\_ylabel('Permutation Entropy', color**=**color)

ax1**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax1**.**set\_xlabel('Number of harmonics')

ax1**.**set\_ylim(0.4, 1.05)

color **=** 'tab:blue'

ax2 **=** ax1**.**twinx()

ax2**.**plot(nharm, wf\_complexity[1, :], color**=**color)

ax2**.**set\_ylabel('Lempel-Ziv complexity', color**=**color)

ax2**.**tick\_params(axis**=**'y', labelcolor**=**color)

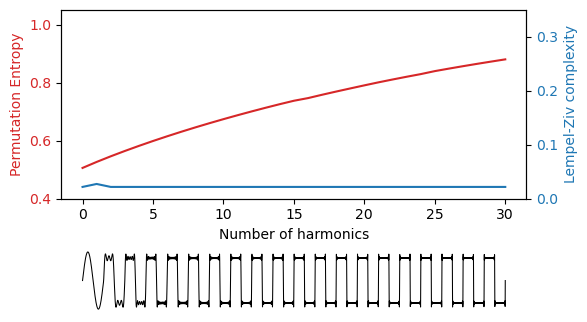
ax2**.**set\_ylim(0, 0.35)

ax3**.**plot(time\_vect, X, lw**=**0.75, color**=**'k')

ax3**.**axis('off')

Out[116]:

(-2.0, 42.0, -1.0999998168267895, 1.0999999710779136)



We can see permutation entropy increases as more frequency content is introduced, whereas LZ is virtually unchanged.

**Summary**

Finally, we plot all the above results together for convenience.

In [129]:

*# Plot the results*

fig, axs **=** plt**.**subplots(2, 3, gridspec\_kw**=**{'height\_ratios': [3, 1]},

figsize**=**(8, 3))

plt**.**subplots\_adjust(hspace**=**0.6, wspace**=**0.5)

*# Frequency*

ax1 **=** axs[0][0]

ax3 **=** axs[1][0]

color **=** 'tab:red'

ax1**.**plot(freqs, chirp\_complexity[0, :], color**=**color)

ax1**.**set\_ylabel('Permutation Entropy', color**=**color)

ax1**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax1**.**set\_xlabel('Frequency [Hz]')

color **=** 'tab:blue'

ax2 **=** ax1**.**twinx()

ax2**.**plot(freqs, chirp\_complexity[1, :], color**=**color)

ax2**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax3**.**plot(time\_vect, Xchirp, lw**=**0.25, color**=**'k')

ax3**.**axis('off')

*# Noise level*

ax1 **=** axs[0][1]

ax3 **=** axs[1][1]

color **=** 'tab:red'

ax1**.**plot(noise\_rms, snr\_complexity[0, :], color**=**color)

ax1**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax1**.**set\_xlabel('Noise level')

ax1**.**set\_ylim(0.4, 1.05)

color **=** 'tab:blue'

ax2 **=** ax1**.**twinx()

ax2**.**plot(noise\_rms, snr\_complexity[1, :], color**=**color)

ax2**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax2**.**set\_ylim(0, 0.35)

ax3**.**plot(time\_vect, Xsnr, lw**=**0.5, color**=**'k')

ax3**.**axis('off')

*# Harmonics*

ax1 **=** axs[0][2]

ax3 **=** axs[1][2]

color **=** 'tab:red'

ax1**.**plot(nharm, wf\_complexity[0, :], color**=**color)

ax1**.**tick\_params(axis**=**'y', labelcolor**=**color)

ax1**.**set\_xlabel('Number of harmonics')

ax1**.**set\_ylim(0.4, 1.05)

color **=** 'tab:blue'

ax2 **=** ax1**.**twinx()

ax2**.**plot(nharm, wf\_complexity[1, :], color**=**color)

ax2**.**set\_ylabel('Lempel-Ziv complexity', color**=**color)

ax2**.**tick\_params(axis**=**'y', labelcolor**=**color)

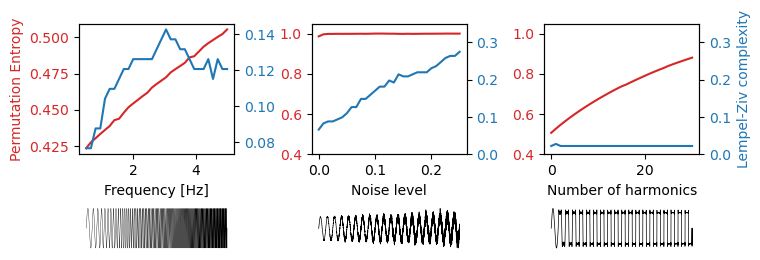
ax2**.**set\_ylim(0, 0.35)

ax3**.**plot(time\_vect, X, lw**=**0.5, color**=**'k')

ax3**.**axis('off')

Out[129]:

(-2.0, 42.0, -1.0999998168267895, 1.0999999710779136)



These are not the only properties that affect complexity metrics. A few more to keep in mind are segment length, noise bandwidth, and non-linear properties such as stochastic variability in the waveform shape. For more details, see our accompanying publication.