dfa.m : Main Code

```
clc
clear all
close all

%% PRE-PROCESSING
% Loading data and removing noise
load 'eegtrial.mat' % 1.0684
srate = 500; % sampling rate
time = 0:2:25997;
y = bandpass(x,[8 12],srate); % bandpass filter to remove noise
% [cA,cD] = dwt(y,'db4'); % DWT for feature extraction
% xrec = idwt(cA,zeros(size(cA)),'db4');
xrec = y';
% Converting into amplitude time-series
% alpha - 10 ; beta - 26 ; gamma - 40
xfilt = abs(hilbert(filterFGx(xrec,srate,10,5))); % equivalent to wavelet convolution
% filterFGx - converts signal to Gaussian
% Creating random noise data
N = length(xrec);
randnoise = randn(N,1);
% Plotting the two signals obtained
figure(1)
subplot(221)
plot(randnoise)
title('Random white noise');
xlabel('Signal points');
ylabel('Magnitude')
subplot(222)
plot(xfilt)
title('EEG Signal after Hilbert transform');
xlabel('Signal points');
ylabel('Magnitude')

%% STEP-1
% Converting to mean-centered cumulative sum
randnoise = cumsum(randnoise-mean(randnoise)); % for the noise s/g
x4dfa = cumsum(xfilt-mean(xfilt)); % for the main s/g
% Plotting for comparison
subplot(223);
plot(randnoise);
title('Mean-centered cumulative sum of noise');
xlabel('Signal points');
ylabel('Magnitude')
subplot(224);
plot(x4dfa);
title('Mean-centered cumulative sum of EEG');
```

```
xlabel('Signal points');
ylabel('Magnitude')

%% STEP-2
% Defining log-spaced scales
nScales = 20;
ranges = round(N*[0.01 0.2]); % range from 1% of signal to 20%
scales = ceil(logspace(log10(ranges(1)),log10(ranges(2)),nScales));
rmses = zeros(2,nScales); %creating a zero matrix to store the future rms values

%% STEP-3 and STEP-4
% Split signal into epochs,detrend, compute RMS
for scalei = 1:nScales

    % No. of epochs for this scale
    n = floor(N/scales(scalei));

    % RMS for the random noise
    epochs = reshape(randnoise(1:n*scales(scalei)),scales(scalei),n);
    depochs = detrend(epochs);
    rmses(1,scalei) = mean(sqrt(mean(depochs.^2,1)));

    % RMS for the s/g
    epochs = reshape(x4dfa(1:n*scales(scalei)),scales(scalei),n);
    depochs = detrend(epochs);
    rmses(2,scalei) = mean(sqrt(mean(depochs.^2,1)));

end

%% STEP-5
% Compute linear fit between log-scales and log-RMS
A = [ ones(nScales,1) log10(scales)' ]; % linear model
dfa1 = (A'*A) \ (A'*log10(rmses(1,:))'); % fit to noise
dfa2 = (A'*A) \ (A'*log10(rmses(2,:))'); % fit to s/g

%% Plotting linear fit
figure(2);
% For noise
plot(log10(scales),log10(rmses(1,:)),'rs','linew',2,'markerfacecolor','w','markersize',10)
plot(log10(scales),dfa1(1)+dfa1(2)*log10(scales),'r--','linew',2)
hold on;
% For signal
plot(log10(scales),log10(rmses(2,:)),'bs','linew',2,'markerfacecolor','w','markersize',10)
plot(log10(scales),dfa2(1)+dfa2(2)*log10(scales),'b--','linew',2)

legend('Data(signal)');
xlabel('Data scale(log)')
ylabel('RMS (log)')
```

```
title('Linear fit')
axis square

% Viewing Hurst parameter in command window
disp('Hurst exponent of signal:')
disp(dfa2)
disp('Hurst exponent of noise:')
disp(dfa1)
```

Filter code: (to be run first separately)

```
function [filtdat,empVals,fx] = filterFGx(data,srate,f,fwhm,showplot)
% filterFGx   Narrow-band filter via frequency-domain Gaussian
%  [filtdat,empVals] = filterFGx(data,srate,f,fwhm,showplot)
%
%    INPUTS
%       data : 1 X time or chans X time
%      srate : sampling rate in Hz
%          f : peak frequency of filter
%       fhwm : standard deviation of filter,
%              defined as full-width at half-maximum in Hz
%   showplot : set to true to show the frequency-domain filter shape
%
%    OUTPUTS
%    filtdat : filtered data
%    empVals : the empirical frequency and FWHM (in Hz and in ms)
%
% Empirical frequency and FWHM depend on the sampling rate and the
% number of time points, and may thus be slightly different from
% the requested values.

if size(data,1)>size(data,2)
%     help filterFGx
%     error('Check data size')
end

if (f-fwhm)<0
%     help filterFGx
%     error('increase frequency or decrease FWHM')
end

if nargin<4
    help filterFGx
    error('Not enough inputs')
end

if fwhm<=0
```

```
    error('FWHM must be greater than 0')
end

if nargin<5
    showplot=false;
end

%% compute and apply filter

% frequencies
hz = linspace(0,srate,size(data,2));

% create Gaussian
s  = fwhm*(2*pi-1)/(4*pi); % normalized width
x  = hz-f;                 % shifted frequencies
fx = exp(-.5*(x/s).^2);    % gaussian
fx = fx./max(fx);          % gain-normalized

%% filter

filtdat = 2*real( ifft( bsxfun(@times,fft(data,[],2),fx) ,[],2) );

%% compute empirical frequency and standard deviation

idx = dsearchn(hz',f);
empVals(1) = hz(idx);

% find values closest to .5 after MINUS before the peak
empVals(2) = hz(idx-1+dsearchn(fx(idx:end)',.5)) - hz(dsearchn(fx(1:idx)',.5));

% also temporal FWHM
tmp = abs(hilbert(real(fftshift(ifft(fx)))));
tmp = tmp./max(tmp);
tx = (0:length(data)-1)/srate;
[~,idxt] = max(tmp);
empVals(3) = (tx(idxt-1+dsearchn(tmp(idxt:end)',.5)) - tx(dsearchn(tmp(1:idxt)',.5)))*1000;

%% inspect the Gaussian (turned off by default)

if showplot
    figure(10001+showplot),clf
    subplot(211)
    plot(hz,fx,'o-')
    hold on
    plot([hz(dsearchn(fx(1:idx)',.5)) hz(idx-1+dsearchn(fx(idx:end)',.5))],[fx(dsearchn(fx(1:idx)',.5))
fx(idx-1+dsearchn(fx(idx:end)',.5))],'k--')
    set(gca,'xlim',[max(f-10,0) f+10]);
```

```
    title([ 'Requested: ' num2str(f) ', ' num2str(fwhm) ' Hz; Empirical: ' num2str(empVals(1)) ', '
num2str(empVals(2)) ' Hz' ])
    xlabel('Frequency (Hz)'), ylabel('Amplitude gain')

    subplot(212)
    tmp1 = real(fftshift(ifft(fx))); tmp1 = tmp1./max(tmp1);
    tmp2 = abs(hilbert(tmp1));
    plot(tx,tmp1, tx,tmp2), zoom on
    xlabel('Time (s)'), ylabel('Amplitude gain')
end
```