SIMULATION OF CORTICAL TRAVELLING WAVES

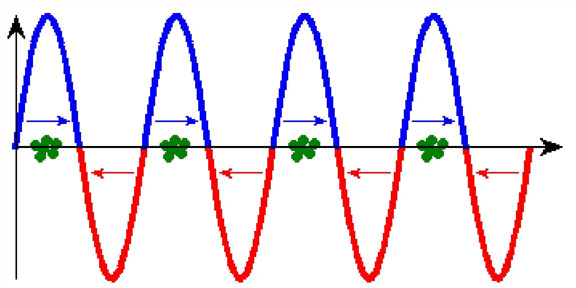
In this project, I tried to simulate an electrophysiological recording dataset (say, EEG) which contains cortical travelling waves (TWs). I spent most of my time on two questions:

1. What exactly is the "travelling wave" in non-periodic data, mathematically?
2. How to make the synthetic data consecutive accross the channels (just like real data) under the requirement of certain TW directions?

THE ESSENCE OF TRAVELLING WAVES

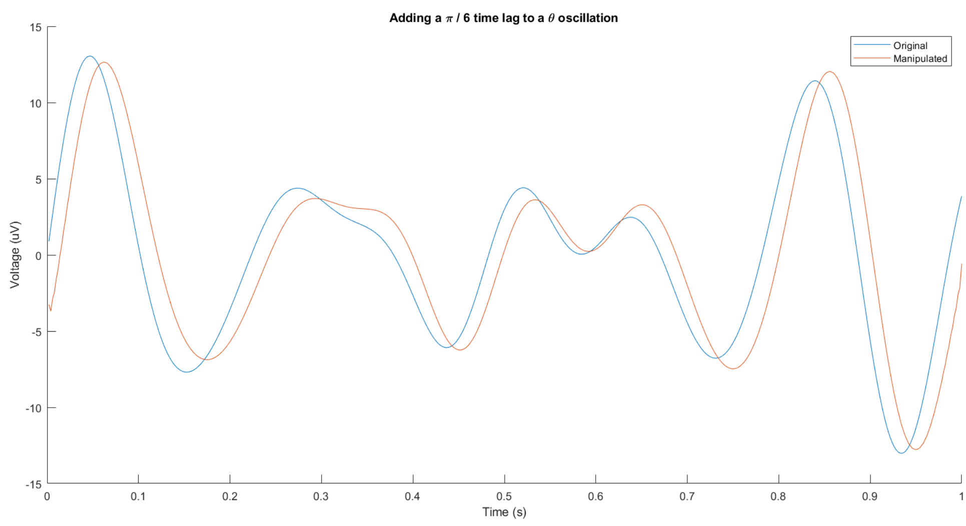
(It's mainly my notes about some problems in signal processing, which may not be meaningful for experts.)

This is what I got in mind about "travelling waves" before I began to really simulate one:



Namely, the activity at each position is a sine wave, and the phase decreases linearly with the distance between the source and the current position: $u(x,t) = A\sin(kx - \omega t + \phi)$. Therefore, if I want to simulate a TW from Cz to Fpz, I just need to add a phase lag like (FCz, -5°), (Fz, -10°), (AFz, -15°), (Fpz, -20°).

However, it becomes less clear when the signal is not a simple sine wave. For example, for a theta-band oscillation, what does it means to describe its "phase" and add "phase lag" to it?



It first reminded me of a famous connectivity criterion called Phase-Lag-Index (PLI), so I reviewed the article and found that the "instantaneous" phase of a narrow-band real-value signal is defined by its analytic representation $s_a(t) = s(t) + j\hat{s}(t) = M(t)e^{j\theta(t)}$. So I googled for concepts like instantaneous phase, analytic representation, Hibert transform and so on. It finally became clear to me that:

(Here are some of my notes)

- the *Hilbert transform* of a signal s(t) is its convolution (in the sense of Cauchy's principle value integral) with the Cauchy kernel $\frac{1}{\pi t}$
  - it can be understood as shifting the phases of the positive frequency components (by Fourier transform) by $\pi/2$ and those of the negative components by $-\pi/2$, which generates the *harmonic conjugate* of the original signal
  - $\mathcal{F}\big(H(u)\big)(\omega) = \big[-i\,\mathrm{sgn}(\omega)\big] \cdot \mathcal{F}(u)(\omega)$
  - some examples: $\mathcal{H}[\sin(\omega t)] = -\cos(\omega t), \mathcal{H}[\cos(\omega t)] = \sin(\omega t), \omega > 0$
  - note that this transform is linear, and the transform of a real function is also real
- the *analytic representation* $s_a(t)$ of signal s(t) is to "cut" the negative frequency components (by Fourier transform) and "copy" their complex conjugate on the positive ones:

$$S(f) = \mathcal{F}[s(t)], S_a(f) = \mathcal{F}[s_a(t)]$$
$$S_a(f) = S(f) + \mathrm{sgn}(f)S(f)$$

- the *analytic representation* can be plotted on the polar plain: $s_a(t) = s(t) + j\hat{s}(t) = s_m(t)e^{j\phi(t)}$.
  - the real part is just the original signal and the imaginary part is the *Hilbert transform* of the signal $\hat{s}(t) = \mathcal{H}[s(t)]$, if s(t) is real-valued.
  - $s_m(t)$ is called *instantaneous amplitude* or *envelope* (波包), $\phi(t)$ is called *instaneous phase* and its derivative is called *instaneous angular frequency*
  - some examples: $\cos(\omega t + \theta) \to e^{j(\omega t+\theta)}, \omega > 0$
- **Note that the MATLAB function `hilbert(x)` computes the analytic representation of x rather than the Hilbert transform!!!**

But still I didn't understand the relationship between the instantaneous phase and the phases of the frequency components. So I tried to derive what will happen if I add a small lag to the instantaneous phase and extract the real part of the new "analytic" signal as $s_1(t)$:

$$
\begin{aligned}
s_a(t) &= M(t)e^{j\theta(t)}, s_{1a}(t) = M(t)e^{j[\theta(t)-\phi_0]} \\
s_1(t) &= \mathrm{Re}[s_{1a}(t)] = M(t)\cos(\theta(t)-\phi_0) \\
&= M(t)\cos\theta(t)\cos\phi_0 + M(t)\sin\theta(t)\sin\phi_0 \\
&= s(t)\cos\phi_0 + \hat{s}(t)\sin\phi_0 \\
S_1(f) &= \mathcal{F}[s_1(t)] = S(f)\cos\phi_0 + \hat{S}(f)\sin\phi_0 \\
&= \begin{cases} S(f)(\cos\phi_0 + j\sin\phi_0) & f<0 \\ S(f)\cos\phi_0 & f=0 \\ S(f)(\cos\phi_0 - j\sin\phi_0) & f>0 \end{cases} \\
&= \begin{cases} S(f)e^{j\phi_0} & f<0 \\ S(f)\cos\phi_0 & f=0 \\ S(f)e^{-j\phi_0} & f>0 \end{cases}
\end{aligned}
$$

And we know that if we want to represent a signal as the summation of a series of cosine waves: $s(t) = d_0 + d(f)\cos(2\pi ft + \phi(f)), f > 0$, then $|d(f)| = 2|S(f)|, \phi(f) = \mathrm{angle}(S(f))$. Therefore, the phase of every cosine components in our $s_1(t)$ is shifted backward for ϕ_0 comparing with s(t), which is consistent with our intuitive understanding about "phase lag". Besides, $s_{1a}(t)$ is indeed the analytic representation of $s_1(t)$.

Therefore, if we want to determine whether there is a consistent phase lag between A and B (namely, whether there is a wave travelling between A and B), we just need to band-pass-filter the signals, compute the analytic representation and extract the instantaneous phase sequence. Or we can compute the derivative of the sequence and see whether they have similar instantaneous angular frequencies (which indicates a consistent phase difference). But definitely the result will not be so straightforward in real-world situations.

SIMULATION OF AN EEG DATASET

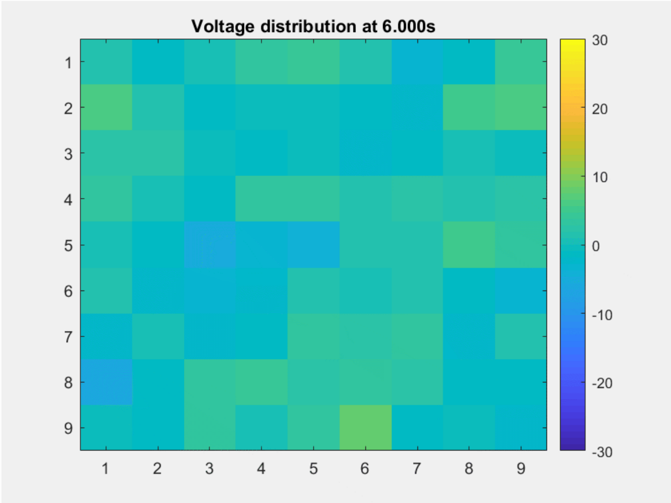
In order to make the synthetic data more similar to real EEG recordings, we decided to construct the signal in frequency domain, with a "reference" power spectrum (computed from a long resting EEG dataset) and a phase spectrum that we can manipulate.

We decided to make the waves travel from center (Cz) to front/back/left/right (Fpz/Oz/T7/T8). Therefore, we:

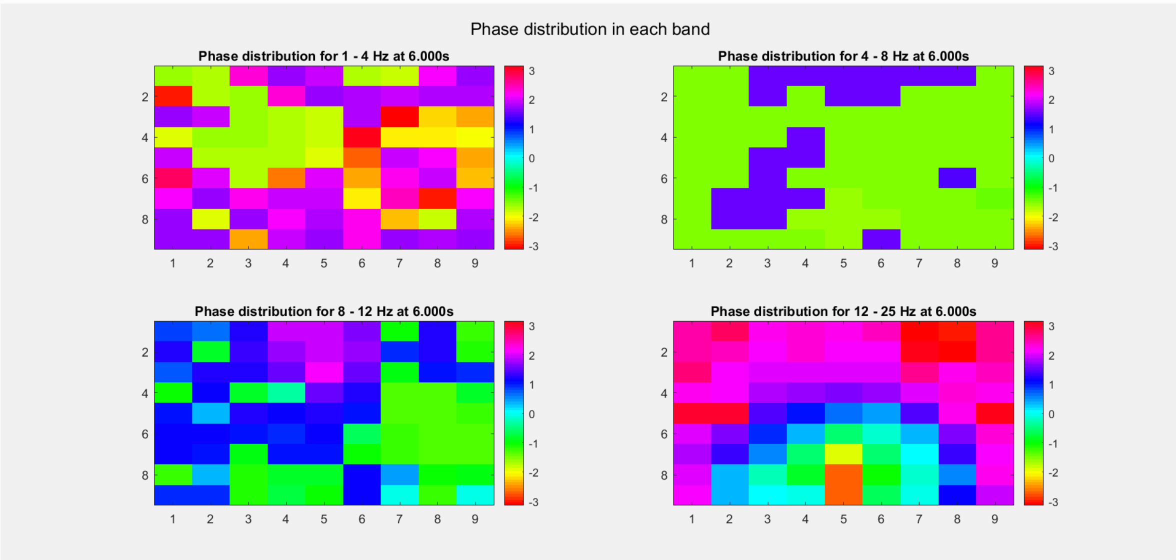
- generate a random phase spectrum for Cz
- use iFFT to reconstruct the signals in each frequency band
- extract the envelope and instantaneous phase in each band
- for each band, select a direction and assign (constant) phase lags to the channels in this direction
- assign (independent, time-variant) random phase lag to other channels by a Wiener Process (Brownian motion)
- spatially "blur" the phase lag by a weighted matrix
- compute the new signal by $s_1(t) = M(t)\cos(\theta(t) - \phi(t))$ where ϕ is the phase pertubation
- add up the signal in each band and scale to 30uV in maximum

Here the constant phase lag is linear to the distance between the channel and Cz, as well as the central frequency in each band (namely, the waves travel at a fixed speed - about 5m/s according to the literature).

Here is what we got:

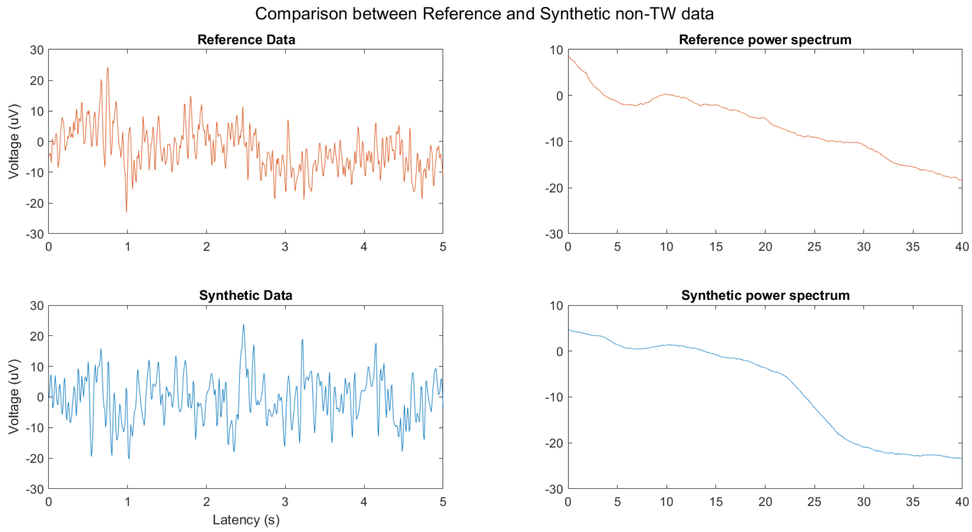


The voltage distribution is mostly consecutive, and it's not very easy to figure out the TW channels at first sight. Here is the phase distribution:



We change the direction of TWs in every two seconds. It's easy to find out the TW channels at the beginning of each segment, but soon it will be hidden in the sea of Brownian phase pertubations.

Finally, let's compare the voltage sequence and oscillatory power of our data and the reference data:



This is a non-TW channel at the corner, and the simulated data looks quite acceptable. Note that we only add up the delta/theta/alpha/beta band (from 1-25Hz) so the high frequency band is weak.

CODES

```
%% syncDat - Create a TW dataset
%
% Ruiqi Chen, 07/04/2020
%
% The synthetic channels are organized into an n*n matrix (n is odd) and
% the waves travel from the center ('Cz') to left ('T7') / right ('T8') /
% front ('Fpz') or back ('Oz').
%
% For each data segments (e.g. 2 seconds):
%
% First, select a random phase spectrum for Cz and use iFFT to reconstruct
% the signal (with a "reference" EEG power spectrum).
%
% (Note that the "reference" spectrum should be interpolated according to
% the Fourier frequencies of the synthetic data (particularly if they have
% different sampling rate), so that we can filter or transform the data by
% simply manipulating the Fourier coefficients without introducing any
% digital-sampling-related artifacts.)
%
% Then for each frequency band (delta, theta, alpha, beta):
%   - "perfectly" filter the signal of Cz (by zero-out Fourier coefficients)
%     and compute the analytic representation
%   - compute the envelope and instantaneous phase
```

```
%   - select one direction (e.g. center to front, from Cz to Fpz)
%   - add a phase lag to each channel on this direction, which is in
%     proportion to the central frequency (i.e. a "constant" time lag) and
%     the distance between two channels (e.g. CPz -5 deg, Pz -10 deg)
%   - add a phase lag (following a Wiener process) to other channels
%   - compute the new band-passed signal for these channels
%
% Then add up these new band-passed signals, and add some noise to every
% channel.
%
% Finally, smooth the intersections between segments if necessary.
%
% Reference: the time lag constant is defined as 7ms per adjacent channel
% according to (Muller et. al., 2018, nrn) that the speed of TW is about
% 5m/s (the distance between channels is calculated according to the
% Biosemi 64 channel system).
%

clear;clc;close all;


%% Parameters

% "Reference" data
REFSET = 'ref/data.mat';
REFCHN = 'ref/chanlocs.mat';
REFSAM = 128;  % Sample rate (Hz)

% Synthetic data
ASIZE = 9;  % Square size, odd number
SAMP = 100;  % Sample rate (Hz)
TLEN = 60;  % Length (second)
TSEG = 2;  % Length of each segment, note that TSEG*SAMP should be integer
PSMOOTH = 0.05;  % Smooth this proportion of data at the beginning of each segment
MAXVOL = 30;  % Maximum voltage value (uV)

% Travelling waves
BANDS = [1 4; 4 8; 8 12; 12 25];  % [HPFREQ LPFREQ] in each line
TWTLAG = 7e-3;  % "Time lag" between two adjacent channels (in seconds)
BMVAR = pi/30;  % variance of the Brownian phase-lag between non-TW channels and Cz
BMDECAY = 0.5; % spatially "blur" the phase lags, see weightMat()

% Visualization
CMPFILE = 'cmpDat.png';  % Compare synthetic data with the reference one
% Illustrate data between [PAHEAD + 1, PAHEAD + PLEN] (indices);
```

```
PLFILE = 'cmpPhaseLag.png';  % Compare TW-channels with Cz
VOLFILE = 'Voltage.gif';  % Show voltage distribution
PHFILE = 'Phase.gif';  % Show phase distribution
FPS = 10;  % Fresh rate for gif
PAHEAD = 600;
PLEN = 300;


%% Preparation

% Get the "reference" data and channel montage
load(REFSET, 'allEEG');
load(REFCHN, 'chanlocs');

% Select Cz
tmpInd = find(strcmp({chanlocs.labels}, 'Cz'));
refDat = allEEG{randi(size(allEEG, 1)), 1}(tmpInd, :);

% Get the "reference" power spectrum
[refP, refF] = periodogram(refDat, [], [], REFSAM);
refP = exp(smoothdata(log(refP)));
% refP = downsample(refP, 10);
% refF = downsample(refF, 10);

% Compute the Fourier frequencies of the synthetic data
newF = 0:1/TSEG:(SAMP - 1/TSEG);
newF = newF(newF < SAMP / 2)';

% Sample the power spectrum
newP = exp(interp1(refF, log(refP), newF));
newA = sqrt(newP);  % The amplitude (omitting a constant)

% Some constants
nFreq = length(newP);
nBand = size(BANDS, 1);
deltaT = 1 / SAMP;
nPSeg = SAMP * TSEG;  % Number of samples in each segment
latency = 0 : deltaT : (TSEG - deltaT);  % latency in each segment
if nPSeg ~= floor(nPSeg)
    error('SAMP * TSEG is not integer')
end

if mod(ASIZE, 2) == 0
    error('Size of the channel array is not odd.');
end
cInd = ceil(ASIZE / 2);  % index of Cz
directions = [-1, 0; 1, 0; 0, -1; 0, 1];  % Front/back/left/right


%% Simulate Cz

Cz = zeros(nBand, SAMP * TLEN);
anasig = complex(Cz);  % Analytic representation for Cz (in each band)

for i = 1:floor(TLEN / TSEG)

```

```
    tmpInd = 1 + (i - 1) * nPSeg : i * nPSeg;  % indices of the data

    initPhases = 2 * pi * rand(length(newP), 1);
    allPhases = initPhases + 2 * pi * newF * latency;

    for j = 1:nBand
        tInd = and(newF >= BANDS(j, 1), newF < BANDS(j, 2)); % "Filtering"
        Cz(j, tmpInd) = newA(tInd)' * cos(allPhases(tInd, :));
        anasig(j, tmpInd) = hilbert(Cz(j, tmpInd));
        % Note that matlab hilbert() computes analytic representation rather than
        % hilbert transform.
    end

end
envl = abs(anasig);  % envelope
iPhases = angle(anasig);  % instant phases


%% Simulate other channels

synDat = zeros(ASIZE, ASIZE, nBand, TLEN * SAMP);
synDat(cInd, cInd, :, :) = Cz;
for i = 1:floor(TLEN / TSEG)

    tmpInd = 1 + (i - 1) * nPSeg : i * nPSeg;  % indices of the data

    for j = 1:size(BANDS, 1)  % Select frequency band

        % Select the TW's direction
        selDind = randi(4);
        if i == 1 + floor(PAHEAD / nPSeg)  % just for visualization
            selDind = min(4, j);
        end
        selD = directions(selDind, :);

        % Compute the random phase lag
        bmSigma = BMVAR * eye(ASIZE*ASIZE);
        bmobj = bm(zeros(ASIZE*ASIZE, 1), bmSigma, 'StartState', 0);
        pLag = simulate(bmobj, nPSeg-1)';

        for k = 1:ASIZE*ASIZE
            ki = 1 + floor((k - 1) / ASIZE);
            kj = 1 + mod(k - 1, ASIZE);
            if ki == cInd && kj == cInd
                pLag(k, :) = 0;
            end
            if all(sign([ki - cInd, kj - cInd]) == selD)  % TW channels
                % Phase lag = c * distance * 2pi * center frequency
                pLag(k, :) = TWTLAG * abs(ki + kj - 2 * cInd) *...
                    2 * pi * mean(BANDS(j,:));
```

```
            end
        end

        % Spatially blur the phase lag
        pLag = weightMat(ASIZE, BMDECAY) * pLag;

        for k = 1:ASIZE*ASIZE
            ki = 1 + floor((k - 1) / ASIZE);
            kj = 1 + mod(k - 1, ASIZE);
            synDat(ki, kj, j, tmpInd) = envl(j, tmpInd) .* ...
                cos(iPhases(j, tmpInd) - pLag(k, :));
        end

    end

end


% Scale the signal
oldSynDat = synDat;  % useful for visualization
synDat = squeeze(sum(synDat, 3));
synDat = synDat - mean(synDat, 3);
synDat = synDat * (MAXVOL / max(synDat, [], 'all'));


% Smooth the intersections

for i = 2:floor(TLEN / TSEG)
    tmpInd = 1 + (i - 1) * nPSeg;
    for j = 1:ASIZE
        for k = 1:ASIZE
            if ~any([j - cInd, k - cInd])  % Skip Cz
                continue;
            end
            synDat(j, k, round(tmpInd - PSMOOTH * nPSeg) : round(tmpInd + PSMOOTH * nPSeg)) = ...
                smoothdata(synDat(j, k, round(tmpInd - PSMOOTH * nPSeg) : round(tmpInd + PSMOOTH * nPSeg)));
        end
    end
end



%% Compare the reference and synthetic data

fig = figure;
fig.WindowState = 'maximized';

subplot(2, 2, 1);
plot(0:(1/REFSAM):5, refDat(1:5*REFSAM + 1), 'Color', [0.8500 0.3250 0.0980]);
title('Reference Data');
xlim([0 5]); ylim([-MAXVOL MAXVOL]);
ylabel('Voltage (uV)');

subplot(2, 2, 2);
plot(refF, 10*log10(refP), 'Color', [0.8500 0.3250 0.0980]);
```

```
title('Reference power spectrum');
xlim([0 40]); ylim([-30 10]);

subplot(2, 2, 3);
plot(0:deltaT:5, squeeze(synDat(1, 1, 1:5*SAMP+1)));
title('Synthetic Data');
xlim([0 5]); ylim([-MAXVOL MAXVOL]);
xlabel('Latency (s)'); ylabel('Voltage (uV)');

subplot(2, 2, 4);
[tmpP, tmpf] = periodogram(squeeze(synDat(1, 1, :)), [], [], SAMP);
plot(tmpf, 10*smoothdata(log10(tmpP)));
title('Synthetic power spectrum');
xlim([0 40]); ylim([-30 10]);

sgtitle('Comparison between Reference and Synthetic non-TW data');
saveas(gcf, CMPFILE);
close(gcf);


%% Visualize the TW

nFig = min(nBand, 4);
tmpInd = PAHEAD:PAHEAD + PLEN - 1;
tmpt = deltaT * tmpInd;
fig = figure; fig.WindowState = 'maximized';
for i = 1:nFig
    subplot(2, 2, i);
    hold on;
    plot(tmpt, squeeze(oldSynDat(cInd, cInd, i, tmpInd)));
    plot(tmpt, squeeze(oldSynDat(cInd + (cInd - 1) * directions(i, 1), ...
        cInd + (cInd - 1) * directions(i, 2), i, tmpInd)));
    legend({'Cz', 'TW channel'});
    title(sprintf('Travelling wave in %.0f - %.0f Hz from %.3fs - %.3fs',...
        BANDS(i, 1), BANDS(i, 2), tmpt(1), tmpt(end)));
end
sgtitle('Comparison between Cz and TW channels');
saveas(gcf, PLFILE);
close(gcf);


%% Visualize the voltage distribution

figure;
fig = imagesc(synDat(:, :, 1), [-30 30]);
colorbar;
for i = 1:PLEN
    fig.CData = synDat(:, :, PAHEAD + i);
    title(sprintf('Voltage distribution at %.3fs', deltaT * (PAHEAD + i - 1)));
    drawnow;
    im = frame2im(getframe(gcf));
    [A,map] = rgb2ind(im,256);
    if i == 1
```

```
        imwrite(A,map,VOLFILE,'gif','LoopCount',Inf,'DelayTime',1/FPS);
    else
        imwrite(A,map,VOLFILE,'gif','WriteMode','append','DelayTime',1/FPS);
    end
end
close(gcf);


%% Visualize the phase

visPhases = zeros(size(oldSynDat));
for i = 1:ASIZE
    for j = 1:ASIZE
        for k = 1:nBand
            visPhases(i, j, k, :) = angle(hilbert(oldSynDat(i, j, k, :)));
        end
    end
end
nFig = min(nBand, 4);

tmp = figure; tmp.WindowState = 'maximized';
colormap(hsv);  % cyclic color mapping
fig = cell(4, 1);
for i = 1:PLEN
    for j = 1:nFig
        subplot(2, 2, j);
        if i == 1
            fig{j} = imagesc(visPhases(:, :, j, i), [-pi pi]);
            colorbar;
        else
            fig{j}.CData = visPhases(:, :, j, i);
        end
        title(sprintf('Phase distribution for %.0f - %.0f Hz at %.3fs',...
            BANDS(j, 1), BANDS(j, 2), deltaT * (PAHEAD + i - 1)));
    end
    if i == 1
        sgtitle('Phase distribution in each band');
    end
    im = frame2im(getframe(gcf));
    [A,map] = rgb2ind(im,256);
    if i == 1
        imwrite(A,map,PHFILE,'gif','LoopCount',Inf,'DelayTime',1/FPS);
    else
        imwrite(A,map,PHFILE,'gif','WriteMode','append','DelayTime',1/FPS);
    end
end
close(tmp);


%% Utility functions
```

```
function w = weightMat(n, decay)
% w = weightMat(n, decay): spatially blur the phase lags
%
% n: size of the square array
% decay: multiply the weight by decay^dis(x, y)
% w: weighting matrix of size (n*n) * (n*n)

p = zeros(n*n, 2);
for i = 1:n*n
    ii = i - 1;
    p(i,:) = [floor(ii / n) + 1, mod(ii, n) + 1];
end
w = squareform(pdist(p));
w = (decay * ones(n*n)) .^ w;
w(w < 0.01) = 0;

end
```