DoA Estimation

Zike,Xu Zhiyi,Wang Dinghao,Cheng

January 2, 2021

Contents

1 Narrowband Estimation

We have obtained a stimulation data with $J = 4$ sensors in *Observations_nb.mat*, and we want to estimate angles of the source with MUSIC algorithm.

In MUSIC, there're some basic arguments, J represents the number of sensors, and P represents the number of sources ($P < J$). The signal is given in the form of

$$s(t) = A(t)e^{j\omega_c t+\phi(t)} \tag{1}$$

when $A(t)$ is the envelop, $\phi(t)$ is the phase. In narrowband, they varies slow than $e^{j\omega_c t}$,

$$s(t-\tau) = A(t-\tau)e^{j\omega_c(t-\tau)}e^{j\phi(t-\tau)} \approx A(t)e^{j\omega_c(t-\tau)\phi(t)} = e^{-j\omega_c\tau} \cdot s(t) \tag{2}$$

thus, in narrowband, Time delay $\approx$ Phase shift.

Position J sensors on x-y axis, their relative position can be $p_i = (x, y)^t$. Consider just one far field signal passes the system with angle θ(with y axis), which can be expressed as $\underline{v} = (\sin\theta, \cos\theta)^t$. The time delay of every sensors is pv. Convert it into phase shift, its $e^{-j2\pi\omega_c\Delta t}$, denote it as $a(\omega, \theta)$. Thus, we can construct a matrix with $a(\omega, \theta)$ which can shift the original signal as θ changes.

$$s[k, \underline{p_i}] = s[k] \cdot e^{j\omega\tau_i} = s[k] \cdot a_i(\omega, \theta) \tag{3}$$

For narrowband signal, we can acquire its f_c(center frequency) in advance, so $a_i(\omega, \theta)$ can be written as $a_i(\theta)$.

We model the system as

$$\begin{aligned}
\underline{x}[k] &= (x_1[k], x_2[k], x_3[k], \cdots, x_J[k])^t \\
\underline{n}[k] &= (n_1[k], n_2[k], n_3[k], \cdots, n_J[k])^t \\
\underline{a}(\theta) &= (a_1(\theta), a_2(\theta), a_3(\theta), \cdots, a_J(\theta))^t
\end{aligned}$$

in which

$$\underline{x}[k] = \underline{a}(\theta)s[k] + \underline{n}[k] \tag{4}$$

for multiple sound sources, add them together to get

$$x_i[k] = \sum_p a_i(\theta_p)s_p[k] + n_i[k] \Rightarrow \underline{x}[k] = A\underline{s}[k] + \underline{n}[k] \tag{5}$$

A is a $J \times P$ matrix, and $\underline{s}[k]$ is the column vector consists of different sound sources.

Denoted the covariance matrix as R_x,

$$R_x = \mathbb{E}\{\underline{x}[k]\underline{x}^h[k]\} \tag{6}$$

expand it with $\underline{x}[k] = A\underline{s}[k] + \underline{n}[k]$,

$$\mathbb{E}\{\underline{x}[k]\underline{x}^h[k]\} = \mathbb{E}\{(A\underline{s}[k] + \underline{n}[k])(A\underline{s}[k] + \underline{n}[k])^h\} \Rightarrow A\mathbb{E}\{\underline{s}[k]\underline{s}^h[k]\}A^h + \mathbb{E}\{\underline{n}[k]\underline{n}^h[k]\} \tag{7}$$

$R_s = \mathbb{E}\{\underline{s}[k]\underline{s}^h[k]\}$ is a Hermite positive definite matrix, so

$$\text{rank}(R_s) = \text{rank}(A) = \text{rank}(AR_sA^h) = P_{source}$$

The rank of the matrix equals to its number of non-zero eigenvalues, so AR_sA^h contains $J-P$ zero eigenvalues.

Let $\underline{u_i}$ be the eigenvector corresponding to zero eigenvalues, so

$$\underline{u}_i^h AR_sA^h\underline{u}_i = \underline{0} \Rightarrow (A^h\underline{u}_i)^h R_s(A^h\underline{u}_i) = \underline{0} \Rightarrow A^h\underline{u}_i = \underline{0} \tag{8}$$

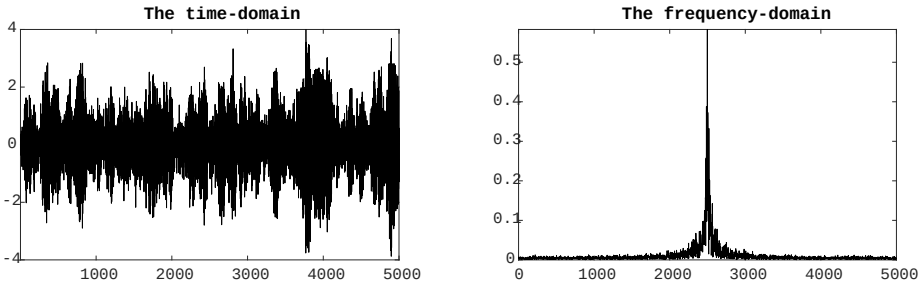
Which claim that the $J-P$ zero eigenvectors is orthogonal to those steering vectors.

We traverse θ to get the minimal $\sum_i \underline{a}^h(\theta_p)\underline{u}_i$, and define

$$P_{music}(\theta) = \frac{1}{\sum_{i=1}^{J-P} \left|\underline{a}^h(\theta_p)\underline{u}_i\right|^2} \tag{9}$$

and the maximum point is the direction of arrival(DoA).

1.1 Find f_c through frequency domain



We can figure out from the graph that $f_c = 2500$Hz.

1.2 Estimate the DoA

The MUSIC algorithm is implemented through MATLAB:

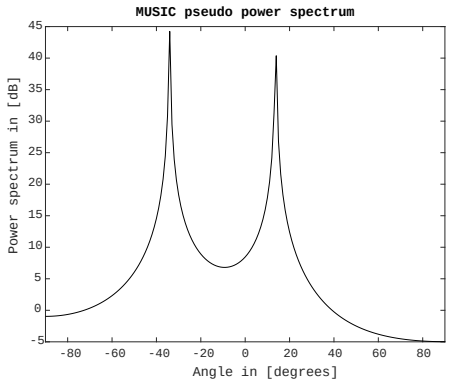
Listing 1: **narrowband.m**

```
clear all; close all;
%% Load data
load("..\data\Observations_nb.mat");
% load data
[Frame,nSensors] = size(X);
f_domain = (-Frame/2:Frame/2-1)*fs/Frame;
%% Plot waveform

figure
subplot(1, 2, 1);
plot(real(X(:, 1)), 'k', 'LineWidth', 0.5);
axis([-inf inf -4 4]);
title("The time-domain");
subplot(1, 2, 2);
plot(f_domain, abs(fftshift(fft(X(:, 1)))/Frame), 'k', 'LineWidth', 0.5);
```

```
16 title("The frequency-domain");
17 axis([0 5000 0 inf]);
18
19 %% Array setup
20 % number of sensors
21 J = nSensors;
22 % inter-sensor distance in x direction (m)
23 dx = 3.4*10^-2;
24 % sensor distance in y direction (m)
25 dy = 0;
26 % sound velocity (m/s)
27 c = 340;
28 % number of sources
29 n_source = 2;
30 Index = linspace(0,J-1,J);
31 % sensor position
32 p = (-(J-1)/2 + Index.') * [dx dy];
33
34 %% Plot sensor positions
35 linspec = {'rx','MarkerSize',12,'LineWidth',2};
36 figure
37 plot(p(:,1),p(:,2),linspec{:});
38 title('Sensor positions');
39 xlabel('x position in meters');
40 ylabel('y position in meters');
41 disp('The four microphones are ready !');
42
43
44 %% DoA estimation (MUSIC)
45 % determine the angular resolution(deg)
46 stride = 1;
47 % grid
48 theta = -90:stride:90;
49 % center frequency (Hz)
50 f_c = 2500;
51 % autocorrelation estimate
52 X = X.';
53 R_x = X*X'/Frame;
54 % direction vector
55 v = [sin(theta*pi/180); -cos(theta*pi/180)];
56 % steer vector
57 a_theta = exp(-1i*2*pi*f_c*(p*v)./c);
58
59 % implement eigen-decomposition
60
61 [V, D] = eig(R_x);
62 eig_val = diag(D);
63 [eig_val, Idx] = sort(eig_val);
64 % noise subspace (columns are eigenvectors), size: J*(J-n_source)
65 Un = V(:, Idx(1:J-n_source));
```

```
% pseudo music power
P_sm = 1./diag(a_theta'*(Un*Un')*a_theta);

%% Plot the MUSIC pseudo power spectrum
figure;
linspec = {'k-','LineWidth', 0.5};
plot(theta, 10*log10(abs(P_sm)), linspec{:});
title('MUSIC pseudo power spectrum')
xlabel('Angle in [degrees]');
ylabel('Power spectrum in [dB]');
xlim([-90,90]);

[source_1, source_2] = find_max(P_sm);

disp(['The desired source DOA with MUSIC is: ',num2str(source_1),' deg']);
disp(['The interfering DOA with MUSIC is: ',num2str(source_2),' deg']);
```

The result of the algorithm is:



Which claims that:

```
The four microphones are ready !
The desired source DOA with MUSIC is: -34 deg
The interfering DOA with MUSIC is: 14 deg
```

2 Broadband Estimation

Since broadband speech signal can be non-stationary, we need STFT(Short-time Fourier Transform) to keep information on frequency domain and time domain at the same time.

The ISSM(Incoherent Signal Subspace Method), which perform STFT on the original signal, and do MUSIC with its result in row. We've read the original paper of ISSM, and ... Whatever, here I'm going to prove the map between frequency and the index in FFT.

In MATLAB, function FFT will perform

$$Y[k] = \sum_{j=1}^{n} X[j] e^{-i2\pi(j-1)(k-1)/n} \tag{10}$$

Thus, in case that the original signal $X[j] = e^{i2\pi f \cdot (j-1)/f_s}$,

$$Y[k] = \sum_{j=1}^{n} e^{i2\pi(j-1)\left[\frac{f}{f_s} - \frac{k-1}{n}\right]}$$

The modulus of the result can reach its peak when k equals to some number. Let $u = \frac{f}{f_s} - \frac{k-1}{n}$, using Euler's formula to transform it into ($u \neq 0$)

$$\mathrm{Re}\{Y[k]\} = \sum_{j=1}^{n} \cos[2\pi u \cdot (j-1)] = \frac{1}{2}(\csc(\pi \cdot u)\sin[(2n-1)\pi \cdot u] + 1)$$

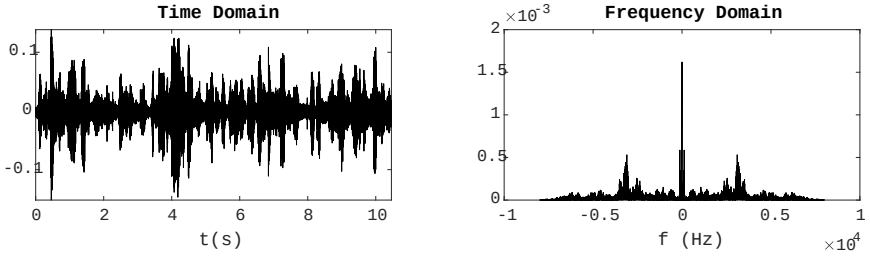
$$\mathrm{Im}\{Y[k]\} = \sum_{j=1}^{n} \sin[2\pi u \cdot (j-1)] = \frac{1}{2}\csc(\pi \cdot u)(\cos(\pi \cdot u) - \cos[(2n-1)\pi \cdot u])$$

Consider the graph of csc function, and other sin or cos part limit it with an approximated 1 value, when u approach to 0, the modulus grows up quickly. And when $u = 0$, as $Y[k] = n$

$$\frac{f}{f_s} = \frac{k-1}{n} \Rightarrow f = \frac{(k-1)f_s}{n} \tag{11}$$

2.1 Plot the wave and frequency response

The figure:



Its easy to figure out that the frequency covers a wide span of frequency and can not find a f_c.

2.2 Perform STFT and MUSIC with spectral density matrix

The processes of ISSM are

1. Perform STFT on the original signal matrix.
2. Perform MUSIC with some of its frequency span.

3. Add them together, get the result.

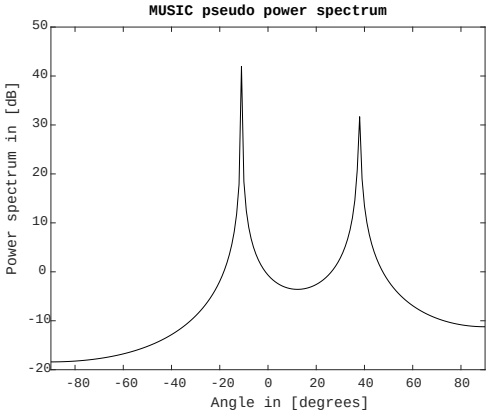
The ISSM algorithm is implemented through MATLAB:

Listing 2: **broadband.m**

```
clear all;
close all;

%% Before STFT
load("..\data\Observation_wb.mat");
% load("data\Observations_nb.mat");
[Frame, ~] = size(X);

%% STFT
len = 512;
inc = 512;
nfft = len; % The smallest 2^n \ge len, to optimize FFT
[st_idx, ed_idx, fn] = separate(len, inc, Frame);

% STFT -> 4 sensors, value after FFT,
STFT = zeros([fn nfft 4]);
window = hann(nfft);
for i=1:fn
    X(st_idx(i):ed_idx(i), :) = diag(window)*X(st_idx(i):ed_idx(i), :);
    STFT(i, :, :) = fft(X(st_idx(i):ed_idx(i), :), nfft);
end

% Perform MUSIC algorithm

% Initialize data
J = 4;
dx = 2.5*10^-2;
dy = 0;
c = 340; % Velocity of sound
Index = linspace(0,J-1,J);
p = (-(J-1)/2 + Index.') * [dx dy]; % Position vector
stride = 1;
theta = -90:stride:90;
v = [sin(theta*pi/180); -cos(theta*pi/180)];

P = zeros([180/stride+1 1]); % -90:stride:90

fr = [40 3000]*nfft/fs+1; % range of frequency (to add weight)

% for i=1:ceil(nfft/2)
for i=floor(fr(1)):ceil(fr(2))

    % P: index -> f
    % $$\frac{(k - 1)f_s}{n}$$
    f_c = (i - 1)*fs/nfft;
    X_ = squeeze(STFT(:, i, :));
```

```
47
48      [Frame_, ~] = size(X_);
49
50      X_ = X_.';
51      R_x = X_*X_'/Frame_;
52      a_theta = exp(-1i*2*pi*f_c*(p*v)./c); % steering vector
53
54      [V, D] = eig(R_x);
55      eig_val = diag(D);
56      [~, Idx] = sort(eig_val);
57      Un = V(:, Idx(1:J-2)); % noise subspace
58      P_sm = diag(a_theta'*(Un*Un')*a_theta);
59      P = P + abs(P_sm);
60  end
61
62  P = 1./P;
63
64  [source_1, source_2] = find_max(P);
65
66  disp(['The first source with MUSIC is: ',num2str(source_1),' deg']);
67  disp(['The second source with MUSIC is: ',num2str(source_2),' deg']);
68
69  figure;
70  linspec = {'k-','LineWidth',0.5};
71  plot(theta, 10*log10(abs(P)), linspec{:});
72  title('MUSIC pseudo power spectrum')
73  xlabel('Angle in [degrees]');
74  ylabel('Power spectrum in [dB]');
75  xlim([-90,90])
76  %% Functions
77
78  function [ st_index, ed_index, fn ] = separate(len, inc, Frame)
79  fn = floor((Frame-len)/inc + 1);
80  st_index = (0:(fn-1))*inc + 1;
81  ed_index = (0:(fn-1))*inc + len;
82  end
```

The result of the algorithm is(STFT Para: [len = 512/inc = 512]):



Which claims that:

```
The first source with MUSIC is: 38 deg
The second source with MUSIC is: -11 deg
```

3 Microphone Array Experiment(with bonus)

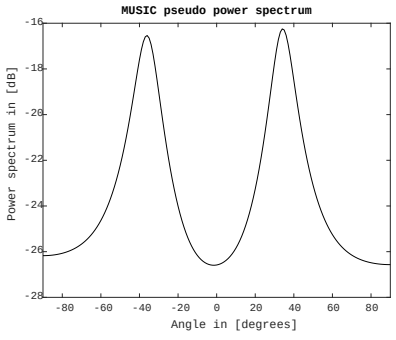
3.1 Adjustment

Real time circumstance is rather complex than the stimulation data, so there's more special cases to deal with. For example, if the result have a long vacant span, there will be a high peak in $deg = 0$ which will interfer our result.

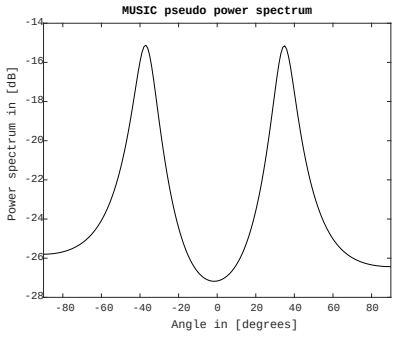
There're multiple solutions to the problem, we can fillter the low frequency band $[0, 20]$ to mitigate the impact of vacant span. Or we can filter the result with its power spectrum. In a word, there's ways through time domain and frequency domain to solve the problem. We chose the former method.

And we apply window function to reduce error. Apply the hanning window function to the segments of STFT, $x[s:e]H^t$ to get the signal.

Here are the figures before and after the optimization:



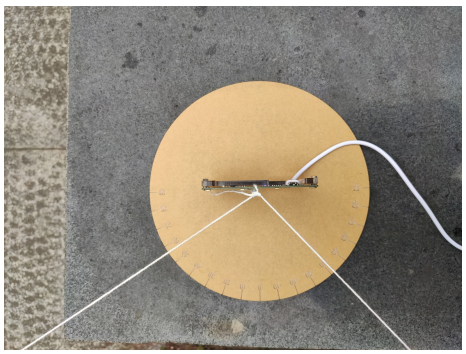
(a) before window



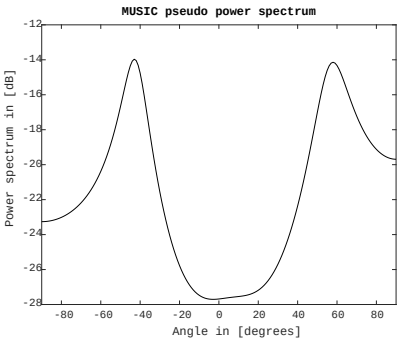
(b) after window

3.2 Process

And we do the experiment with two cotton threads to specify the direction. Two of our members speak at two directions, holding the cotton threads to get precise directions. Using Audacity, we acquired four tracks of sound. Remember to reverse the tracks because of different modeling methods.



(c) Test environment



(d) Output

And the result is rather precise:

```
The first source with MUSIC is: -43 deg
The second source with MUSIC is: 58 deg
```

3.3 Real time

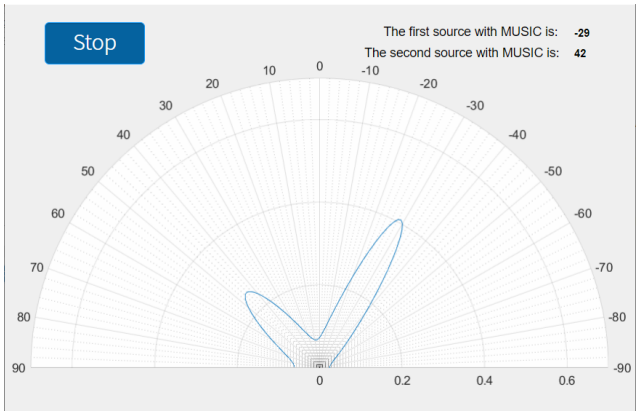
The real time DoA is implemented in *main.m*, and its core is to use *audioDeviceReader()* to get signal from the audio card.

```
devReader = audioDeviceReader( ...
    'Driver', 'DirectSound', ...
    'SamplesPerFrame', fs*samplingTime, ...
    'SampleRate', fs, ...
    'NumChannels', 4, ...
    'BitDepth', '16-bit integer', ...
    'Device', '   (USB YDB01 Audio Effect)', ...
    'ChannelMappingSource', 'Property', ...
    'ChannelMapping', [2 1 4 3] ...
);
setup(devReader);
```

After finishing the initialization, just apply *music* with those segments and display the real-time information.

3.4 GUI

Our GUI finally came out to be:



The GUI part, which focuses on building a user-friendly program doing the real-time DOA estimation, can show the probability of every direction of the sound through a polar diagram, and display the most possible two directions on the interface. The start and stop can be entirely controlled by a button. It is a task finished with lots of difficulties and obstacles mostly caused by the unfamiliarity with MATLAB.

We started with designing an application using MATLAB Guide. But not long we found that it's really hard to use so we turn to MATLAB App designer, a more easy-to-use tool. But what we didn't know then was that it was just the beginning of the tough development process.

Aiming at a GUI which can start and stop the real-time DOA estimation algorithm at any time, the first thought of us was to conform to the multi-threaded procedural framework by using the object *timer* in MATLAB. In fact, it was not only a natural choice but also an efficient way to control all the program work and stop according to every "tic" of the *timer*. But soon we found that it caused a large number of problems, list as follows:

- No graph. Though there're three running modes for *timer*, the period is fixed. That means the program itself will pitilessly start the next period regardless of the graph has not been plot yet, which caused that there will be hard to control the *timer.period* to let the graph be plotted.
- Chaos. The timer will never stop unless the *stop* command is executed. And if you execute the *start* command while the previous *timer* is still running, the command will create another *timer* doing the same thing as the previous one, and you don't have effective means to tell them apart, which causes that all the command become confusing and lead to the total chaos of the program.
- Unreasonable bugs. Though the logic of the program is right, the test of it discovers many unreasonable bugs, such as unfixed update rate of the graph, unexpected crash of MATLAB, not being able to restart after stop, etc.

So it becomes impossible for us to accomplish the GUI through timer. We replace it with another construction, which has a prepare part, a main working part and a stop part. The prepare part does necessary settings like setup the *audioDeviceReader*, change the text of the button; the working part do the MUSIC algorithm to estimate the direction of the sound and update the graph; the stop part help the program stop properly by release the *audioDeviceReader* and change the text of the button back. This model works regularly by the signal produced by the button. Below are the details of how it work.

1. Get started. When the program is first launched, all of the elements are set.
2. Get prepared. The double-state button is the only controller of the whole program. When it turns to "press", which equals its value turn to "1", the prepare part is firstly launched by the callback of the button.
3. Work. Working part runs right after the prepare part. It will continue working until the value of the button element comes to "0".
4. Stop. When the value of the button element comes to "0", the working circulation will be broken and the stop part will do something to recover some properties for next time.

Seems really easy, but the problem is, if a callback is working without special operations, then any other callbacks won't be launched at all. That means, if the button turns to the other state, its corresponding callback will just keep pending, and any of the properties of any elements of the program won't be changed by that. Furthermore, the plot course will also be interrupted by next working part. It really costs some time to find out the solution: *drawnow*.

It surely a great breakthrough in finding this suitable function. The description is "updates figures and processes any pending callbacks". That means it can plot the graph and change the value of the button to "0" to let the working part know it should stop next time. However, this function seems not working every time. It would randomly ignore the callback caused by the button and continue working without any hesitation. Till now, we still don't know why *drawnow* sometimes loses its efficacy, we can only guess that it is caused by the running mode of MATLAB and its unclear multi-threaded procedural framework. This surely leaves some regret for the project, but we have really tried our best.

References

[1] R. Schmidt, "Multiple emitter location and signal parameter estimation," in IEEE Transactions on Antennas and Propagation, vol. 34, no. 3, pp. 276-280, March 1986, doi: 10.1109/TAP.1986.1143830.

[2] Guaning Su and M. Morf, "The signal subspace approach for multiple wide-band emitter location," in IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 31, no. 6, pp. 1502-1522, December 1983, doi: 10.1109/TASSP.1983.1164233.