Direction of Arrival Estimation Algorithms

Nikhil Shetty

April 7, 2004

Contents

Chapter 1

Software report

1.1 The Problem

The aim of the software is to implement the MUSIC algorithm that enables an antenna array to estimate the number of incident signals on the array and their directions of arrival.
Let there be m elements in the array. Let x_i be the input at the ith element of the array (including noise added by the instruments and present in incoming signal) and let $x = \left[x_i\right]$. According to the MUSIC algorithm , the covariance matrix formed as $S = xx*$ (where x^* denotes conjugate transpose) has n repeated minimum eigenvalues (for uncorrelated noise) that represent the variance of the noise. The number of incident signals is given by $d = m - n$. Further , the eigenvectors corresponding to these eigenvalues are orthogonal to the array manifold (the gain and phase change provided by the array elements to the incoming signal directions) at the values of the incoming signal directions. Hence , on forming a matrix E_N of these eigenvectors and plotting

$$PMU(\theta) = \frac{1}{a(\theta)^* E_N E_N^* a(\theta)} \tag{1.1}$$

we get peaks at the values of theta corresponding to the incoming signal directions.

1.2 User's Guide

PART I: This part is a sample calling program designed to validate the working of the music algorithm. A real life case will be considered

later. The sample program is simulating a sine wave input that is arriving at the array in the form of planar wavefronts (the source is assumed sufficiently far away). The array itself is assumed to be an m ($NOOFELEMENT$) element linear array with no directional properties (could be an array of dipoles) in the given plane. The number of elements can be varied. Further the frequency of the input can be changed. $dfactor$ is the division factor of the wavelength that represents the distance between the elements.(for eg: if $dfactor = 2$,distance between elements = $\frac{\lambda}{2}$). Thus we have a handle on both the frequency and the distance between elements in terms of the given frequency.

The *samples* variable represents the number of samples that we wish to take for the given input and fs represents the sampling frequency. These two variables can be adjusted to simulate the processor available , the amount of accuracy and the system refresh rate. For example , if we chose a high value of *samples* , amount of memory utilisation increases and the refresh rate (rate between consequent measurements) decreases however the accuracy of calculations increases (assuming input signal lasts for that long a time).

x is the matrix in which the signal recieved at the aray elements is stored.

$$\begin{bmatrix} x_1(t_1) & x_1(t_2) & \cdots & x_1(t_{samples}) \\ x_2(t_1) & x_2(t_2) & \cdots & x_2(t_{samples}) \\ \vdots & \vdots & \vdots & \vdots \\ x_m(t_1) & x_m(t_2) & \cdots & x_m(t_{samples}) \end{bmatrix}$$

is the format in which the array is stored where $x_i(t_j)$ represents the value of the input function detected at the ith element at the jth time instant(including noise). Thus $f1$ is the required input function and r is the noise matrix. The noise is zero mean with variable variance that can be changed by changing the multiplier.

Once the array manifold function is defined in a different file (refer PART II) all the user has to do is call the *music1d* function to get the estimate of the number of signals and a plot of $PMU(\theta)$ versun θ.

PART II**:** The array manifold is defined as the gain and phase change provided by the array elements to a given signal direction. The array manifold is a function of θ and is saved in the file manifold file. Note that the variables $NOOFELEMENTS$ and $dfactor$ are global and hence can be accessed by the manifold too.

PART III: This part consists of the actual implementation of MUSIC algorithm.The function returns an estimate of the dir3ection of arrival. It takes as input the x matrix. From the x matrix , the number of elements and the samples is obvious. The first step is the calculation of the covariance matrix. The next step involves calculation of the eigenvalues and eigenvectors of S.

On knowing the minimum eigenvalue, the multiplicity of this eigenvalue is obtained. The multiplicity is presently being obtained using a *naive* comparison (refer to the code listing). However , the final aim is to use the MDL criterion to find the number of incident signals. The next step is to obtain the noise eigenvector matrix of these minimum eigenvalues and form the matrix *nev*. Using this nev , we can plot to get direction of arrival in either a single dimension or multiple dimensions. The program presented works only for a planar angle. The final aim is to get direction of arrival in terms of azimuthal angle and elevation angle.

1.3 Testing

Ensure that all the files (main, manifold,music1d are in same directory. Run the *main* program in matlab and enter the number of signals and the angle in degrees for each of the signals. The program will give the number of signals as output along with the plot. Now we just have to obtain the values of the highest d peaks to get the direction of arrivals.

Test Input 1: Enter value for number of signals as 1. Now , change the angle from 10 to 80 and observe the output. We observe sharp peaks for single signal.

Test Input 2: Enter an arbitrary value for number of signals (less than about 3/4th of the number of elements for good results). Enter the values of the angles and observe the output.

1.4 Design features and observed results

There are a number of parameters for which changes can be made and varying results can be obeserved. They are

1. **Number of elements:** Increasing the number of elements increases the resolution for multiple sources. If the number of signals is large , then a higher element array predicts the directions of arrival better.

2. **Noise** With lower noise , the peaks become sharper. The presence of added noise causes a spreading effect on the peaks.

3. **Number of samples** Increasing the number of samples does not bring about a greater improvement in the plots. However , number of samples must be greater than number of incident signals for workable results. 100 samples work well for a 15-element array.

4. **Distance between elements** The distance between elements can be changed. This effectively means that for a fixed frequency of the input wave,the *centre frequency* is changing that is the frequency at which the detection is best. A lower wavelength and hence higher frewquency gives no resolution at all (fig 1). On the other hand , a higher wavelength (lower frequency) gives a lot of spurious peaks. It is observed that for best detection of given frequency the distance between elements must be $\frac{}{2}$.

5. **Kind of input wave** The kind of input wave can be varied . Resluts were verified for sine waves and FM-modulated waves (refer PART IV). The plots obtained match those in theory. However the algorithm could not detect sin^2 waves even on removing the DC offset of the waves.

6. **Obtain 2-d direction** We can also extend the MUSIC algorithm to 2-d with the two dimensions being the plane angle and the elevation angle respectively (refer PARTS V and VI). However the results obtained in 2-d are not so encouraging in terms of the number of peaks obtained.

1.5 Further Work

Further work can proceeed on the following counts

1. Using the MDL criterion to find the number of incident signals to get a better estimate.

2. Delve into why the algorithm does not work satisfactorily for inputs like sin^2 waves.

3. To improve the performance of the 2-d DOA performance.

1.6 Code Listing

PART I:

```
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% PART I - THE CALLING PROGRAM  %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


global  NOOFELEMENTS;%no of elements
NOOFELEMENTS=15;
m=NOOFELEMENTS;

w=1000000000; %frequency (not angular) taken 1 GHz

global dfactor;  % distance between elements= wavelegth / dfactor
dfactor=2;

nos=input('enter no of signals')

angl(nos)=0; % array to store the angles of arrival
for i=1:nos
    angl(i)=input('enter angle in degrees')
    theta(i)=angl(i)*pi/180;
end

samples=100; % no of samples of the input signal taken

fs=(.0001*w-1); % sampling frequency - rate at which samples are taken

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% SIMULATION OF A POSSIBLE INPUT TO THE   %%%%%%%%%%
%%%%%%%%             MUSIC ALGO                  %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f1(m,samples)=0; % array of sampled signal ie signal values at different
   % elements at diff instants
for k=1:samples   % no of time instants
    for i=1:m % no of elements
```

```
        for ctr=1:nos
            f1(i,k)=f1(i,k)+sin(2*pi*w*(k/fs+(1-i)/dfactor*sin(theta(ctr))/w)
        % sine wave sampled - this function can be changed

        end
    end
end

r=0.3162/1.414*randn(m,samples);  % RANDOM VARIABLE REPRESENTING ADDED NOISE

x=f1+r;  % ADDING NOISE TO THE INPUT SIGNAL ARRAY TO GET THE POSSIBLE OUTPUT

noofsignals=music1d(x) % calling the music1d function (1-d music)
```

PART II**:**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%  PART II - ARRAY MANIFOLD  %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function a=manifold(angle)
global NOOFELEMENTS dfactor
no=NOOFELEMENTS;
for k=1:no
a(k,1)=cos((1-k)*2*pi*sin(angle)/dfactor)+sin((1-k)*2*pi*sin(angle)/dfactor)*
end
```

PART III**:**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%  PART III - MUSIC  Algorithm  %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function estimate=music1d(x)

temp=size(x); % returns the size
m=temp(1); % m is the number of elements
samples=temp(2); % no of samples of the incident signals

%finding covariance matrix of the input x
```

```
s(m,m)=0;
for i=1:m
    for k=1:m
        for n=1:samples
            s(i,k)=s(i,k)+x(i,n)*x(k,n);
        end
        s(i,k)=s(i,k)/samples;
    end
end


% eigenvalues of s
[evec eval]=eig(s);  % evec is the set of eigenvectors and eval is the
% set of eigenvalues

minevalue=eval(1,1);% finding the minimum eigenvalue
for i=2:m
    if(minevalue>eval(i,i))
        minevalue=eval(i,i);
    end
end


% finding the multiplicity of the minimum eigenvalue
n=0; %multiplicity
for i=1:m
    if(eval(i,i)<(1+minevalue)) % assuming that all the roots less than
    % 1+minevalue are minimum roots
                                % have to get a better approximator of
    % multiplicity
        n=n+1;
    end
end


'Multiplicity of min root'
n

'no of incident signals'
m-n
```

```
% forming the eigenvector matrix corresponding to the minimum eigenvalues
nev(m,n)=0; % noise eigenvector matrix

for i=1:n
    for l=1:m
        nev(l,i)=evec(l,i);
    end
end


% now to plot using this matrix
ctr=1;
for angle=1:0.25:360 %in degrees
    indep(ctr)=angle;
    rad=angle*pi/180;
    a=manifold(rad); % returns the array behaviour for a signal
% incident from certain direction
    c=a'*nev;
    d=c*c';
    pmu(ctr)=1/abs(d); % pmu(theta) is the function that is to be
  % plotted as a function of theta
    ctr=ctr+1;
end

plot(indep,pmu)  % will plot PMU(theta) versus theta
grid
estimate=m-n % estimate of the number of incident signals
```

PART IV**:**

```
clear
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% PART IV - THE CALLING PROGRAM  %%%%%%%%%%%%%%%%
%%%%%%%%%%     FOR FM-MODULATED INPUT    %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global  NOOFELEMENTS;%no of elements
NOOFELEMENTS=15;
m=NOOFELEMENTS;
```

```
w=1000000000  ; %frequency (not angular)

global dfactor;  % distance between elements= wavelegth / dfactor
dfactor=2;

nos=input('enter no of signals')

angl(nos)=0;



for i=1:nos
    angl(i)=input('enter angle in degrees')
    theta(i)=angl(i)*pi/180;
end

fs=(.0001*w-1); % sampling frequency - rate at which samples are taken

samples=100

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%SIMULATION OF A FM MODULATED INPUT TO THE   %%%%%%%%%
%%%%%%%%              MUSIC ALGO                  %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



for k=1:samples   % no of time instants
    for i=1:m % no of elements
        for ctr=1:nos
            f1(i,k)=f1(i,k)+sin(2*pi*w*((1+.01*(sin(2*pi*w*(k/(.001*w-1)))))
+(1-i)/dfactor*sin(theta(ctr))/w));

        end
    end
end

r=0.3162/1.414*randn(m,samples);  % random variable .
```

```
x=f1+r;

noofsignals=music1d(x)
```

PART V**:**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%  PART V - MUSIC  ALGORITHM    %%%%%%%%%%%%%%%%
%%%%%%%%%%%%       FOR  2-D  DOA           %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function estimate=music1d(x)

temp=size(x); % returns the size
m=temp(1); % m is the number of elements
samples=temp(2); % no of samples of the incident signals

%finding covariance
s(m,m)=0;
for i=1:m
    for k=1:m
        for n=1:samples
            s(i,k)=s(i,k)+x(i,n)*x(k,n);
        end
        s(i,k)=s(i,k)/samples;
    end
end

% eigenvalues of s
[evec eval]=eig(s);  % evec is the set of eigenvectors and eval is the
                        % set of eigenvalues

minevalue=eval(1,1);% finding the minimum eigenvalue
for i=2:m
    if(minevalue>eval(i,i))
        minevalue=eval(i,i);
    end
end

n=0; %multiplicity
for i=1:m
```

```
    if(evalues(i)<(1+minevalue)) % have to get a better approximator of
     % multiplicity
        n=n+1;
    end
end
'Multiplicity of min root'
n

'no of incident signals'
m-n

nev(m,n)=0; % noise eigenvector matrix
%k=1
for i=1:n%m
    %if abs(eval(i,i))<(1+abs(minevalue))
        for l=1:m
            nev(l,i)=evec(l,i);%nev(l,i)=evec2(l,i);
        end
        %k=k+1
        %end
end


% now to plot using this matrix
ctr1=1;
for angle1=1:1:90 %in degrees
    indep1(ctr1)=angle1;
    rad1=angle1*pi/180;
    ctr2=1;
    for angle2=1:1:90 %in degrees
        indep2(ctr2)=angle2;
        rad2=angle2*pi/180;
        for k=1:m
            a(k,1)=manifold3(rad1,rad2);
        end
        c=a'*nev;
        d=c*c';
        pmu(ctr1,ctr2)=1/abs(d);
        ctr2=ctr2+1;
    end
```

```
    ctr1=ctr1+1;
end
'columns of I represent the elevation angles'
[Y,I]=max(pmu)
mesh(indep1,indep2,pmu)
```

PART VI**:**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%  PART VI - ARRAY MANIFOLD  %%%%%
%%%%%       FOR  2-D  DOA        %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function a=manifold(angle1,angle2)
global NOOFELEMENTS dfactor
no=NOOFELEMENTS;
for k=1:no
    a(k,1)=cos((1-k)*2*pi*sin(angle1)/dfactor*cos(angle2))
+sin((1-k)*2*pi*sin(angle1)/dfactor*cos(angle2))*j;
end
```