به نام خدا

علیرضا دری

۹۵۳۱۳۳۳

گزارش کد های نوشته شده در متلب و زبان C برای شبیه‌سازی الگوریتم MUSIC

استاد راهنما : جناب دکتر علی اکبر تدین تفت

Contents

۱- شبیه‌سازی اول

newProject.m , plottest.m , newProjectfortest.m , correlated.m

در این برنامه در مرحله اول اطلاعات شبیه‌سازی سیگنال های ارسالی وارد میشود.

```
C = physconst('lightspeed');
fs =20e6; %sampling frequency
t =0:1/fs:5e-6; %time duration
sample = length(t);
f = [9.9e6 10e6 10.1e6]; %frequence of usres
inAng =[-20 -5 30]; %Direction Of Arrival
doa = inAng/180*pi;%Direction Of Arrival(radian)
nU =length(f);
nE = 6; % number of Element
lambda = C./f;
d = lambda(1)/2; %Elements spaceing
SNR = 10; %SNR
```

در مرحله بعد باید سیگنال ها را با فرکانس های مشخص شده تولید کنیم. در کد زیر مشاهده میشود هر سیگنال با فاز تصادفی تولید میشود.

```
S = zeros(nU,sample);
for index = 1:nU
   S(index,:) =1/2*( exp(2*pi*1j*(f(index)*t + 2*rand-1)));
end
```

اکنون باید سیگنال دریافتی هر المان آرایه آنتن را با توجه به سیگنال های قسمت قبل تولید کنیم. دو راه وجود دارد روش اول که چون سیگنال باریک باند است قابلیت پیاده‌سازی دارد این روش برگرفته از معادله ۱-۲ است یعنی ماتریس A را با کمک رابطه ۲-۲ شبیه سازی کرده و در ماتریس S قسمت قبل ضرب میکنیم. سیگنال x همان سیگنال دریافتی بدون نویز است.

```
A =zeros(nU,nE);
for k=1:nU
    A(k,:)=exp(-1j*2*pi*d*sin(doa(k))/lambda(k)*[0:nE-1]); %phase shift for
each element and signal
end
x = A'*S;
```

روش دوم این است که ابتدا آرایه آنتن را شبیه‌سازی کرده و سپس با دستور phased.Collector سیگنال های دریافتی را با زوایای مشخص شده برای هر المان دریافت کنیم. نتایج در هر صورت هیچ تفاوتی ندارد.

```
array = phased.ULA('NumElements',nE,'ElementSpacing',d,'ArrayAxis','y');
%create the array antenna
collector = phased.Collector('Sensor',array,'OperatingFrequency',10e6);
x = collector(S.',inAng);
x = x.';
```

در برخی شبیه‌سازی ها از روش دوم و در برخی دیگر از روش دوم استفاده شده است.

مرحله بعد اضافه کردن نویز با دستور awgn است.

```
x = awgn(x,SNR);
```

اکنون به مرحله آخر یعنی پیاده‌سازی الگوریتم MUSIC می‌رسیم.

```
Rxx=x*x'; %Data covarivance matrix
[N,V]=eig(Rxx); %Find the eigenvalues and eigenvectors of R
NN=N(:,1:nE-nU); %Estimate noise subspace
theta=-90:0.1:90;
for ii=1:length(theta) %Peak search
    SS=zeros(1,(nE));
    for jj=0:nE-1
        SS(1+jj)=exp(-1j*2*jj*pi*d*sin(theta(ii)/180*pi)/lambda(1));
    end
    PP(ii)=SS*NN*NN'*SS';
    Pmusic(ii)=abs(1/ PP(ii));
end
Pmusic=10*log10(Pmusic/max(Pmusic)); %Spatial spectrum function
figure
plot(theta,Pmusic,'-k')
xlabel('angle \theta/degree')
xlim([-90 90])
ylabel('spectrum function P(\theta) /dB')
title('DOA estimation based on MUSIC algorithm ')
grid on
```

مشاهده میشود که خروجی برنامه یک نمودار است در ادامه الگوریتمی برای جدا سازی نقاط پیک دامنه ارائه خواهد شد.

گفتیم روش capon شبیه به روش MUSIC است. در پایین فقط قسمتی که تغییر می‌کند را نشان داده می‌شود.

```
Rxx=x*x'; %Data covarivance matrix
theta=-90:0.1:90; %Peak search
for ii=1:length(theta)
    SS=zeros(1,(nE));
    for jj=0:nE-1
        SS(1+jj)=exp(-1j*2*jj*pi*d*sin(theta(ii)/180*pi)/lambda(1));
    end
     PP(ii)=SS*inv(Rxx)*SS';    %% Capon method
    Pmusic(ii)=abs(1/ PP(ii));
end
```

قبل و بعد برنامه کاملا شبیه قسمت قبل است.

فایل هایی که به طور کلی در آن ها از این کد ها استفاده شده است فایل های

"newProject.m" : که درواقع برنامه اصلی است.

"plottest.m" : این برنامه شبیه به برنامه قبل است فقط تفاوتش آن است که برنامه قبل در یک حلقه جا داده شده که در این حلقه برنامه به تعدادی مثلا ۳ یا ۴ بار برای شرایطی مثل تعداد المان آنتن متفاوت اجرا می‌شود یا برای SNR های متفاوت و نتایج بر روی یک نمودار چاپ می‌شود.

"newProjectfortest.m" : این برنامه هم شبیه به برنامه اول است فقط برنامه داخل یک حلقه است که ۵۰۰ بار اجرا می‌شود و در هر بار اجرا بدون نمایش نمودار فقط و فقط نقاط زاویه کشف شده را در یک ماتریس دخیره می‌کند و سپس با استفاده از فرمول تعیین خطا مقدار RMSE را برای هر زاویه تعیین می‌کند. روش جدا سازی نقاط پیک نمودار همان روش مشتق چپ و راست هستند که در زبان C استفاده شده است.

در اینجا کد آن قسمت تغییر یافته را نشان می‌هیم.

```
.
.
.
A =zeros(nU,nE);
for k=1:nU
    A(k,:)=exp(-1j*2*pi*d*sin(doa(k))/lambda(k)*[0:nE-1]); %phase shift for
each element
end
x1 = A'*S;
doas = sort(inAng); %در این روش زاویه ها به ترتیب از کوچیک به بزرگ پیدا می‌شود پس باید ترتیب آن هارا داشته باشیم.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
repeat = 500; %number of reapet
for nR = 1:repeat    %start Loop
x = awgn(x1,SNR);
%% sec4
Rxx=x*x'; %Data covarivance matrix
[N,V]=eig(Rxx); %Find the eigenvalues and eigenvectors of R
NN=N(:,1:nE-nU); %Estimate noise subspace
theta=-90:.5:90; %Peak search
for ii=1:length(theta)
    SS=zeros(1,(nE));
    for jj=0:nE-1
        SS(1+jj)=exp(-1j*2*jj*pi*d*sin(theta(ii)/180*pi)/lambda(1));
    end
    PP=SS*NN*NN'*SS';  %MUSIC method
    %PP=SS*inv(Rxx)*SS'; %Capon method

    Pmusic(ii)=abs(1/ PP);
end
Pmusic=10*log10(Pmusic/max(Pmusic)); %Spatial spectrum function
flag1 = 1;   % در اینجا جدا سازی انجام می شود.
flag2 = 1;
zz = 1;
for z = 1:length(Pmusic)-2
     if(Pmusic(z) > 2/3*min(Pmusic))
         if(Pmusic(z) < Pmusic(z+1))
             flag1 = 1;
         else
             flag1 = 0;
         end
         if (Pmusic(z+1) < Pmusic(z+2))
             flag2 = 1;
         else
             flag2 = 0;
         end
         if (flag1 == 1 && flag2 ==0)
             RDOA(zz) = theta(z+1); %هر زاویه که شناسایی شد ذخیره می‌شود.
```

```
            zz= zz+1;
        end
        if (zz == nU+1)
            break;
        end
    end

end
RDOAnR(nR,:) =RDOA; %تا اینجا زاویه هر آزمایش پیدا شده و به ترتیب در یک ماتریس دخیره میشود.
for index = 1:nU %اندازه گیری بخشی از فرمول خطا دخیره اطلاعات
   rms(nR,index) = (doas(index)-RDOA(index))^2;
end

end    %%finish Loop
RMES = sqrt((sum(rms,1))/repeat); %محاسبه خطا برای تمام دفعات تکرار
for index = 1:nU       %چاپ اطلاعات خروجی
formatSpec = 'RMES %3.0f is =>(%2.7f )  \n';
fprintf(formatSpec,doas(index),RMES(index));
end

plot(theta,Pmusic,'-k')
xlabel('angle \theta/degree')
xlim([-90 90])
ylabel('spectrum function P(\theta) /dB')
title('DOA estimation based on MUSIC algorithm ')
grid on
```

نمونه خروجی نوشتاری:

RMES -20 is => (0.4774935)

RMES -5 is => (0.4129165)

RMES 30 is => (0.4555217)

در مورد شبیه‌سازی های انجام شده در باند باریک اطلاعات مهم دیگری وجود ندارد.

یک فایل دیگر نیز به در باند باریک شبیه‌سازی شده است به نام "correlated.m" در این فایل تغییراتی بر روی ماتریس X صورت گرفته که طبق گزارش اصلی است یعنی تبدیل داده های المان آنتن ها به چند زیر آرایه کوچک و میانگین گیری.

در بخش اول یک متغیر به برنامه اضافه شده است به نام nSA که نشان دهنده تعداد زیر آرایه هاست در انتخاب آن باید دقت شود که بر nE بخش پذیر باشد. همچنین sizeSA بیانگرتعداد آنتن های هر زیر آرایه می‌باشد.

```
Rxx = zeros(sizeSA,sizeSA);
for index = 1:nSA
    z = x(1+((index-1)*sizeSA):index*sizeSA,:);
    zxx = z*z';
    Rxx = zxx +Rxx;
end
Rxx = Rxx / nSA;
```

در این قسمت ماتریس X را پس از اعمال نویز به شکل بالا به چند زیر آرایه تبدیل و میانگین گیری می‌کنیم

۲- شبیه‌سازی برنامه با فیلتربانک

"newProjectFilterBank.m" Section 1

```
%% sec1
C = physconst('lightspeed');
fs =20e6; %sampling frequency
t =0:1/fs:1e-5; %time duration
sample = length(t);
%%%%
staFreq = [20e6 2.5e6 5e6 7.5e6 10e6 12.5e6 15e6 17.5e6]; %main frequency
of each channel DO NOT CHANGE THIS VECTOR
staLambda = C./staFreq;  %lambdas that useing in doa estimation
%%%  %frequence of usres
f =  [[19.9e6 20e6   20.1e6]          ...
      [5e6    5.1e6]                  ...
      [2.5e6  2.6e6 ]                 ...
      [7.5e6  7.4e6  7.6e6 ]          ...
      [10e6   10.1e6 9.9e6]           ...
      [12.5e6 12.4e6 12.6e6]          ...
      [15e6]                          ...
      [17.5e6 17.6e6 17.4e6 17.7e6 17.3e6]];

inAng =[[40 30 -40] ...
      [20 60]     ...
      [-20 60]    ...
      [20 60 -40] ...
      [20 60 -40] ...
      [20 60 -40] ...
      [20 ] ...
      [30 60 -30 -20 -70]]; %Direction Of Arrival
doa = inAng/180*pi;
nU =length(f); %number of user

nE = 10; % number of Element
lambda = C./f;
%d = lambda(1)/2;
d  = staLambda(1)/2; %Elements space
SNR = 10; %SNR
```

در اینجا شرایط پیش آزمایش تعیین میشود. دقت شود زاوایا متناظر با هر فرکانس است یعنی فرکانس اول با زاویه اول و فرکانس دوم با زاویه دوم و تا اخر.

Section 2

```
%% sec2 create signals
S = zeros(nU,sample);
for index = 1:nU
   S(index,:) =1/2*( exp(2*pi*1j*(f(index)*t + 2*rand-1)));
end
```

section 3

```
%% sec3 create phase shift matrix (A) and matrix(x)
AF =zeros(nU,nE);
for k=1:nU
    AF(k,:)=exp(-1j*2*pi*d*sin(doa(k))/lambda(k)*[0:nE-1]); %phase shift
for each element
end
x1 = AF'*S; % arrived signal to antenna without noise
x1 = awgn(x1,SNR); % add noise
```

section 4

```
%% sec 4 filter bank
%signal spectrum
X =fft(x1,[],2);
figure
plot(linspace(0,2,sample),(abs(X)),'b')
xlabel('\omega')
ylabel('ampitude')
xticks(0:1/4:2)
xticklabels({'0','\pi/4','2\pi/4','3\pi/4','\pi','5\pi/4','6\pi/4','7\pi/4'
,'2\pi'})
% Filter and Filter Bank designing
L = 8; % length of filter and also L is number of filter bank elements
nnn = 0:L-1; % for create each Filter we need this vector
% FIR Filter
b = [1,-.09,+0.05,-0.07]; %ضرایب صورت و مخرج فیلتر
a = [1,-1.96,.903,.21];

for topindex = 0:L-1   %فرمول محاسبه جملات فیلتر بر اساس ضرایب آن و همچنین اعمال شیفت فرکانسی برای هر فیلتر
    nn = nnn-topindex;
    BF = (b(1)+b(2).*exp(-1j*2*pi*(nn')/L)+b(3).*exp(-
1j*2*2*pi*(nn')/L)+b(4).*exp(-1j*3*2*pi*(nn')/L));
    AF = (a(1)+a(2).*exp(-1j*2*pi*(nn')/L)+a(3).*exp(-
1j*2*2*pi*(nn')/L)+a(4).*exp(-1j*3*2*pi*(nn')/L));
    for index = 1:L%ساخت فیلتر نهایی با تقسیم جملات صورت در مخرج
        H (topindex+1,index)= BF(index)/AF(index);
    end
    H(topindex+1,:) =  H(topindex+1,:)/abs(max( H(topindex+1,:)));
end

% figure
% plot(linspace(0,2,L),10*log10(abs(H)),'b')
% xlabel('\omega')
% ylabel('ampitude')
% xticks(0:1/4:2)
%
xticklabels({'0','\pi/4','2\pi/4','3\pi/4','\pi','5\pi/4','6\pi/4','7\pi/4'
,'2\pi'})

h = ifft(H,[],2);  %Filters in time domain
% Filter specterm
fFilter = linspace(0,fs,1000);
for ii = 1:L
hresp(ii,:) = freqz(h(ii,:),1,fFilter,fs);
end

figure
```

```
plot(fFilter/1000000,10*log10(abs(hresp((1:8),:))));
grid on; xlabel('Frequency (MHz)'); ylabel('Magnitude (dB)');
title('Frequency Response of Filter(s)');
% convolution each entnna data to each Filter (اعمال فیلتر)
for topindex = 1:L
    for index = 1:nE
        x(index+(nE*(topindex-1)),:) = conv(x1(index,:),h(topindex,:));
    end
end
```

دقت شود چون کار با ماتریس سه بعدی سخت است در اینجا دیتا ها را پس از فیلتر کردن در یک ماتریس ۲ بعدی ذخیره کرده ایم به نوعی که تعداد nE سطر اولیه خروجی داده های همه آنتن ها از فیلتر اول و nE سطر دوم خروجی همه آنتن ها از فیلتر دوم و ... می‌باشد.

section 5

```
%% sec 5
nUF = zeros (1,L+1);
lowrenge = -1.25e6;
highrenge = 1.25e6;
 for topindex = 1:L+1
     for index = 1:nU
         if((f(index)>=lowrenge) && (f(index) <= highrenge))
             nUF(topindex) = nUF(topindex) + 1;
         end
     end
     lowrenge = highrenge;
     highrenge = highrenge + 2.5e6;
 end
 nUF (1) = nUF(L+1);
```

در اینجا تعداد سیگنال ها در هر باند مشخص میشود. نکته اینجا این است که فرکانس باند ۲۰ مگاهرتز همان باند ۰ مگاهرتز است دلیل خط آخر همین است.

Section 6

این بخش اهمیتی ندارد صرفاً داده خروجی هر فیلتر را در حوزه فرکانس نشان می‌دهد.

Section 7

```
for index  = 1:L
    if (nUF(index) == 0)
        continue;
    end
Rxx=x(1+(nE*(index-1)):nE*index,:)*(x(1+(nE*(index-1)):nE*index,:))'; %Data
covarivance matrix
[N,V]=eig(Rxx); %Find the eigenvalues and eigenvectors of R
NN=N(:,1:nE-nUF(index)); %Estimate noise subspace
theta=-90:0.1:90; %Peak search
for ii=1:length(theta)
    SS=zeros(1,(nE));
    for jj=0:nE-1
        SS(1+jj)=exp(-1j*2*jj*pi*d*sin(theta(ii)/180*pi)/staLambda(index));
    end
   % Pmusic(:,ii) = abs( SS*SS'./(SS'.*NN*NN'.*SS));
```

```
    PP(ii)=SS*NN*NN'*SS';
    Pmusic(ii)=abs(1/ PP(ii));
end
Pmusic=10*log10(Pmusic/max(Pmusic)); %Spatial spectrum function

figure('Name',['output for Filter :',num2str(index)])
plot(theta,Pmusic,'-k')
xlabel('angle \theta/degree')
xlim([-90 90])
ylabel('spectrum function P(\theta) /dB')
title(['DOA MUSIC,output for channel frequncy: '
num2str(staFreq(index)/1e6), ' MHz'])
grid on
end
```

تغییراتی جزیی در قسمت الگوریتم اجرا شده تا برای هر خروجی هر فیلتر نتایج را نشان دهد.

فایل مرتبط با این برنامه :

"newProjectFilterBank.m"

۳- شبیه‌سازی سیگنال باند وسیع (سیگنال شبیه‌سازی شده)

دو فایل "LTEreal3.m" و "RealSignalMUSIC.m" برای سیگنال شبیه سازی شده است کد این دوفایل به شکل زیر است .

فایل LTEreal3.m یک فانکشن برای تولید سیگنال LTE و جداسازی ساب کریر ها می‌باشد و توسط آقای دکتر واعظی نوشته شده است. کد آن به شکل زیر زیر است:

```
function rxWaveform_Deprecod_direc = LTEreal3(N_Rx,inAng,d,SNR)
% N_Rx = 10 ; % # receive antennas
%% ==
ue = lteRMCUL('A1-1') ;
ue.TotSubframes = 20 ;
 ue.NULRB = 6;
% ue.NTxAnts = 1 ;
%ue.PUSCH.NLayers = 1 ;
% ue.NCellID = 1 ;
% ue.NSubframe =0 ;
%% ================
cec = struct('FreqWindow',7,'TimeWindow',1,'InterpType','cubic') ;
channel.Seed = 3 ;

channel.NormalizeTxAnts = 'On' ;
channel.NormalizePathGains = 'On' ;
channel.DelayProfile = 'EPA' ; %% ETU EPA EVA

channel.NRxAnts = N_Rx ;
channel.DopplerFreq = 0.0;
channel.MIMOCorrelation = 'UplinkMedium' ;
channel.InitPhase = 'Random';
channel.InitTime = 0 ;
channel.ModelType = 'GMEDS' ;
channel.NTerms = 16 ;

% channel.Pa

%% ================
[puschInd, info] = ltePUSCHIndices(ue,ue.PUSCH);
%% == MATLAB channel ===
[txWaveform,txGrid,cfg] = lteRMCULTool(ue,[1;0;0;1]);
channel.SamplingRate = cfg.SamplingRate;
C = physconst('lightspeed');
nU = length(inAng);
fmain = 1900e6;



array = phased.ULA('NumElements',N_Rx,'ElementSpacing',d,'ArrayAxis','y');
collector =
phased.WidebandCollector('Sensor',array,'CarrierFrequency',fmain,'SampleRat
e',2e6,'NumSubbands',26);
 rxWaveform1 = lteFadingChannel(channel,txWaveform);
rxWaveform1 = collector(rxWaveform1(:,1:nU),[inAng]);
rxWaveform = awgn(rxWaveform1,SNR);
rxWaveform_demod = lteSCFDMADemodulate(ue,rxWaveform) ;
%% === SCFDMA Deprecoder ======
for ind_ant = 1 : N_Rx
    rxWaveform_demod1 = rxWaveform_demod(:,:,ind_ant) ;
```

```
    rxPrecoded = rxWaveform_demod1(puschInd) ;
    N_RB = ue.NULRB ;
    num_active_SC = 12*N_RB ;
    data1 =
reshape(rxPrecoded,num_active_SC,size(rxPrecoded,1)/num_active_SC);
    data_deprecode(:,:,ind_ant) = ifft(data1*sqrt(num_active_SC)) ;
    data_deprecode_reshap(:,:,ind_ant) =
reshape(data_deprecode(:,:,ind_ant),size(rxPrecoded));
end
rxWaveform_Deprecod_direc = data_deprecode ;
```

دقت شود کد بالا برای شرایطی که فقط یک سیگنال داشته باشیم طراحی شده است هرچند برای دو یا بیشتر هم جواب قابل قبولی به دست می‌آید ولی بهتر است فقط برای یک سیگنال این آزمایش انجام شود.

فایل بعدی داری کد زیر است که به تعداد هر سیگنال از کد بالا برای تولید سیگنال استفاده می‌کند. در بخش اول اطلاعات شبیه‌سازی شده و ویژگی های سیگنال تولید شده وارد می‌شود.

```
NSC = 72; %number of subcarrier
C = physconst('lightspeed');
f1 = 1800e6;
delta = 15e3;
fsc = zeros(1,NSC);
for index = 0:NSC-1          %frequency of each subcarrier
    fsc (index+1)= f1+(delta*index);
end
fmain = 2e6;
Lmain = C/fmain;

sample =12;
inAng = [30 ];         %Direction Of Arrival length of doa is number of user
doa = inAng/180*pi;
nU =length(doa);            %number of usre:length( doa)
nE = 3;                %number of array element
lambdaSC = C./fsc;
d = min(lambdaSC)/2;     %element space
SNR = 20;
%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
 x1 =LTEreal3(nE,inAng,d,SNR);
```

در بخش دوم الگوریتم به نوعی طراحی شده است که برای هر زیر حامل داده زاویه ای به دست آورد.

```
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %% music algorithm for each sub carrier
for sc = 1:NSC
    x4(:,:) =x1(sc,:,:);
    x =conj(x4');

    Rxx=x*x'; %Data covarivance matrix
    [N,V]=eig(Rxx); %Find the eigenvalues and eigenvectors of R
    NN=N(:,1:nE-nU); %Estimate noise subspace
    theta=-90:0.5:90; %Peak search
    for ii=1:length(theta)
        SS=zeros(1,(nE));
```

```
        for jj=0:nE-1
            SS(1+jj)=exp(-1j*2*jj*pi*d*sin(theta(ii)/180*pi)/lambdaSC(sc));
        end
        PP=SS*NN*NN'*SS';
        Pmusic1(sc,ii)=abs(1/ PP);

    end

end
% summation of MUSIC result of each sub carrier
Pmusic =sum(Pmusic1,1);
Pmusic = Pmusic;
Pmusic=10*log10((Pmusic/max(Pmusic))); %Spatial spectrum function
figure
plot(theta,Pmusic,'-k')
xlabel('angle \theta/degree')
xlim([-90 90])
ylabel('spectrum function P(\theta) /dB')
title('DOA estimation based on MUSIC algorithm ')
grid on
figure
for index = 1:72
    nummax = find(Pmusic1(index,:) == max(Pmusic1(index,:))) ;
    ang (index) = theta(nummax);
Pmusic=10*log10((Pmusic1(index,:)/max(Pmusic1(index,:)))); %Spatial
spectrum function

plot(theta,Pmusic,'-k')
xlabel('angle \theta/degree')
xlim([-90 90])
ylabel('spectrum function P(\theta) /dB')
title('DOA estimation based on MUSIC algorithm ')
grid on
hold on
end
figure
h = histogram(ang,9);
xlim([-90 90])
 xlabel('angle \theta/degree')
ylabel('number of repeat for each theta')
```

چون برنامه برای تخمین یک زاویه یک سیگنال طراحی شده هیستوگرام فقط زاویه های یک سیگنال را بررسی میکند.

۴- تست با سیگنال واقعی

نمونه های یک سیگنال واقعی LTE که زیرحامل های آن جدا شده است در فایل "LTE_UL.mat" توسط اقای دکتر واعظی برای این آزمایش تهیه شده است و برنامه هم همان برنامه قبلی است با این تفاوت که داده ورودی آنتن ها در فایل بالا قرار دارد. نام این برنامه "REAL.m" می‌باشد. چون نکته قابل توجهی در این کد نیست نیازی به توضیح ندارد.

۵- شبیه‌سازی برنامه اول یعنی سیگنال باریک باند با زبان c

در این برنامه ها از کتاب‌خانه علمی Eigen استفاده شده است برای اجرای برنامه ها می‌توانید در کامپایلر c تنظیمات را به نوعی انجام دهید که در داخل پوشه پروژه به دنبال کتاب‌خانه ها بگردد و فایل پوشه کامل کتاب‌خانه را داخل پوشه پروژه کپی کند و یا پوشه کتاب‌خانه Eigen را در مسیر اصلی کتاب‌خانه های کامپایلر اضافه کنید(پوشه Eigen تحویل داده شده است بنا به سیستم خود آن را در یکی از مسیر ها کپی کنید.)

۱

```
#include <iostream>
#include <Eigen/Eigenvalues>
#include "complex"
#include "cmath"
#include <Eigen/Dense>
#include "random"
#define c 299792458
#define p 3                 //number of signal
#define N 200               //number of samples
#define pi 3.1415926535897
#define M 6               //number of elements
```

در این قسمت اطلاعات پایه وارد می‌شود.

۲

```
void deg2rad (float a[]);
void clambda (int f[],float lambda[]);
void cw (float w[],int f[],int fs);
void creatsignal(std::complex<double> S[][N],float w[]);
void creatAmatrix (std::complex<float>  A[][p],float doa[],float
lambda[],float d);
void thehtaMaker(float theta[] ,float degreeStep,int nD);
```

در اینجا اطلاعات مربوط توابع ساخته شده و تعریف آن ها وارد شده است.

۳

```
using namespace std;
int main() {
    int fs = 20000000;
    float doa[p] = {-20, -5,30};
    deg2rad(doa);
    int f[p] = {10000000, 10100000,9900000};
    float lambda[p];
    clambda(f, lambda);
    float d = lambda[0] / 2;
    float SNRdb = 10;
    ///////////////
    float w[p];
    cw(w, f, fs);
    complex<double> S[p][N];
    creatsignal(S, w);
    ///////////////
    complex<float> A[M][p];
    creatAmatrix(A, doa, lambda, d);
    ///////////////
```

در این بخش اطلاعات شبیه سازی و ساخت ماتریس های A و S می‌باشد.

۴

```
// change array to standard matrix for eigen library
Eigen::MatrixXcd SS(p, N);
for (int k = 0; k < p; k++)
    for (int j = 0; j < N; j++)
        SS(k, j) = S[k][j];

Eigen::MatrixXcd AA(M, p);
for (int k = 0; k < M; k++)
    for (int j = 0; j < p; j++)
        AA(k, j) = A[k][j];
// recived signal  without noise
Eigen::MatrixXcd x = AA * SS;
```

در اینجا می‌شد با یک حلقه تو در تو ضرب را انجام داد ولی اطلاعات را از ماتریس معمولی به ماتریس استاندارد Eigen منتقل کرده و از آنجا ضرب را انجام می‌دهیم. در اینجا سیگنال X بدون نویز ساخته می‌شود.

۵

```
//add noise
float Pn = pow(10.0, (-1 * SNRdb / 10));
std::default_random_engine generator;
std::normal_distribution<double> distribution(0.0, 1.0);
Eigen::MatrixXcd randomnumber(M, N);
for (int k = 0; k < M; k++) {
    for (int j = 0; j < N; j++) {
        complex<double> G = distribution(generator) + 1i *
distribution(generator);
        randomnumber(k, j) = G;
    }
    randomnumber = randomnumber * sqrt(.5) * sqrt(Pn);
}

x = x + randomnumber;
```

سیگنال با استفاده از کد بالا دارای نویز گوسی می‌شود اطلاعات نویز در بالا آمده است ابتدا SNR به توان تیدبل شده است.

۶

از اینجا وارد الگوریتم MUSIC می‌شویم.

```
Eigen::MatrixXcd tx = x.adjoint();
Eigen::MatrixXcd Rxx = x * tx;
```

محاسبه R_{xx} با محاسبه کانجوکیت ترنسپوز X

۷

```
Eigen::ComplexEigenSolver<Eigen::MatrixXcd> Eces;
Eces.compute(Rxx);
Eigen::MatrixXcd eigVec = Eces.eigenvectors();

Eigen::MatrixXcd noiseSub(M, M - p);
for (int k = 0; k < M; k++){
    for (int j = 0; j < (M - p); j++)
        noiseSub(k, j) = eigVec(k, j);
}
```

محاسبه ویژه بردار ها و جدا سازی شبه‌فضای نویز از آن ها

۸

```
const float degreeStep =0.5;
const int numberDegree = 180/degreeStep;
float theta [numberDegree];
thehtaMaker(theta,degreeStep,numberDegree);
double Pmusic[numberDegree] ;
double PP ;
double maxPmusic =0;
double minPmusic = 0;

////////////////////
Eigen::MatrixXcd ASS(1,M) ;
for (int k = 0;k<numberDegree;k++) {

    ASS = Eigen::ArrayXXcd::Zero(1, M);
    for (int j = 0; j < M; j++) {
        ASS(0, j) = exp(
                1i * static_cast<complex<double>>((-2) * j * pi * d *
sin(theta[k] / 180 * pi) / lambda[0]));
    }
    Eigen::MatrixXcd PPM = ASS * noiseSub * noiseSub.adjoint() *
ASS.adjoint();

    PP = abs(PPM(0, 0));
    Pmusic[k] = 1 / PP;
    if (Pmusic[k] > maxPmusic)
        maxPmusic = Pmusic[k];
}
```

بخش مهمی از الگوریتم که شامل شبیه‌سازی ماتریس های A و پیاده سازی سایر بخش ها است دقت شود ماکزیمم خروجی الگوریتم در همین جا مشخص می‌شود.

۹

```
for (int k =0 ; k < numberDegree;k++){
    Pmusic[k] =  10*log10 (Pmusic[k]/maxPmusic);
    if (Pmusic[k] < minPmusic)
        minPmusic = Pmusic[k];
}
```

ایجاد اطلاعات به صورت لگاریتمی و جدا سازی مینیمم آن ها(مینیمم برای استفاده در جداسازی نقاط پیک کاربرد دارد.)

۱۰

```
//////////////////////
    bool flag1 = true;
    bool flag2 = true;
    float RDOA[p];
    int jj=0;
    for (int k = 0; k < numberDegree-2; k++) {
        if (Pmusic[k] > (minPmusic*2/3)){
            if (Pmusic[k] < Pmusic[k+1]) {
                flag1 = true;
            } else
                flag1 = false;
            if(Pmusic[k+1] < Pmusic[k+2]){
                flag2 = true;
            } else
                flag2 = false;
```

```
            if (flag1 && !flag2){
                RDOA[jj] = theta[k+1];
                jj++;
            }
        }
        if (jj  == p){
            break;
        }
    }
    cout << "we have (" << p << ") doa.\n";
    for (int k =0;k<p;k++) {

        cout << "[ " << k + 1 << " ] " << " is \"" << RDOA[k] <<
"\""<<endl;
    }


    return 0;
}
```

جدا سازی زاویه های به دست آمده و انتقال به خروجی با همان روش مشتق چپ و راست.

از اینجا به بعد هر کدام از توابع نوشته شده را توضیح می‌دهیم.

۱۱

```
void deg2rad (float a[]){
    for(int  i = 0 ;i<p;i++)
    {
        a[i] = a[i]/180*pi;
    }
}
void clambda (int f[],float lambda[]){
    int i =0 ;
    for (i =0 ;i<p;i++)
    {
        lambda[i] = static_cast<float>(c) / f[i];
    }
}
```

تابع اول زاویه ها را به رادیان تبدیل میکند و تابع دوم طول موج سیگنال هارا محاسبه می‌کند.

۱۲

```
void cw (float w[],int f[],int fs){
    int i ;
    for (i = 0 ;i<p;i++){
        w[i] = (2*pi)*f[i]/fs; }
}
void creatsignal(complex<double> S[][N],float w[]){
    for (int k = 0 ;k<p;k++)
    {
        complex<double> q = w[k];
        for (int j = 0 ;j<N;j++)
        {
            S[k][j] = exp(1i * q * static_cast<complex<double>>( j+1));
        }
    }
}
```

در این برنامه به جای شبیه‌سازی زمان تعداد نمونه ها را شبیه‌سازی کرده ایم تابع CW گام حرکت برای شبیه‌سازی سیگنال را تعیین میکند. که با توجه به فرکانس نمونه برداری و فرکانس سیگنال به دست می‌آید. تابع بعدی سیگنال ها را با توجه به تعداد نمونه ها و گام حرکت تولید میکند.

۱۳

```
void creatAmatrix (complex<float>  A[][p],float doa[],float lambda[],float
d)
{
    for (int k = 0;k<p;k++){
        for(int j =0;j<M;j++)
        {
            A [j][k] = exp(1i  *static_cast<complex<double>> (2*pi * d *
sin(doa[k]) * j / lambda[k]));
        }
    }
}
void thehtaMaker(float theta[] ,float degreeStep,int nD) {
    theta[0] = -90;
    for (int i = 1; i < nD; i++) {
        theta [i] = theta[i-1] + degreeStep;
    }
}
```

تابع اول ماتریس A را تولید می‌کند و تابع دوم ماتریس theta که برای الگوریتم شبیه‌سازی می‌شود.

خروجه برنامه :

we have (3) doa.

[1] is "-20"

[2] is "-5"

[3] is "29.5"

۶- شبیه‌سازی با فیلتر بانک

در اینجا نیز با همان روال قبل برنامه را بخش بخش نشان می‌دهیم.

۱

```
#include <iostream>
#include <Eigen/Eigenvalues>
#include "complex"
#include "cmath"
#include <Eigen/Dense>
#include "random"
#include <unsupported/Eigen/FFT>
#include "Eigen/Core"
#define c 299792458
#define p 27                  //number of signal
#define N 200                 //number of samples
#define pi 3.1415926535897
#define M 10                  //number of elements
#define L 8                   //filterBank length and number
```

تعریف ثوابت و وارد کردن کتابخانه ها

۲

```
void deg2rad (float a[]);
void clambda (int f[],float lambda[],int size);
void cw (float w[],int f[],int fs);
void creatsignal(std::complex<double> S[][N],float w[]);
void creatAmatrix (std::complex<float>  A[][p],float doa[],float
lambda[],float d);
void thehtaMaker(float theta[] ,float degreeStep,int nD);
void cFilterBank(std::complex<double> H[L][L],float a[],float b[]);
void filterpass(std::complex<double> y[L*M][N],Eigen::MatrixXcd
x1,std::complex<double> H[L][L]);
```

توابع مورد نیاز

۳

```
using namespace std;
int main() {
    int fs = 20000000;
    int stdFreq[L] = {20000000, 2500000, 5000000, 7500000, 10000000,
12500000, 15000000, 17500000};
    float stdLambda[L];
    int size = sizeof(stdFreq) / sizeof(stdFreq[0]);
    clambda(stdFreq, stdLambda, size);
    float doa[p] = {60, 20, -20, -40                //DOA
            ,-30,30
            ,40,-40
            ,20,60,-40
            ,20,40,-45
            ,17,77,0,-20
            ,20,45,-70,60,-20
            ,35,-35,0,70};
    deg2rad(doa);
    int f[p] = {10000000, 10100000, 9900000, 10200000       //signal freq
            ,2500000,2600000
            ,5000000,5100000
```

```
            ,7500000,7600000,7400000
            ,12500000,12600000,12400000
            ,15000000,15100000,15200000,14900000
            ,17500000,17600000,17400000,17300000,17700000
            ,20000000,20100000,19900000,20200000};
    float lambda[p];
    clambda(f, lambda, p);
    float d = stdLambda[0] / 2;
    float SNRdb = 10;
```

شرایط شبیه‌سازی

۴

```
float w[p];
cw(w, f, fs);
complex<double> S[p][N];
creatsignal(S, w);
///////////////
complex<float> A[M][p];
creatAmatrix(A, doa, lambda, d);
///////////////
// change array to standard matrix for eigen library
Eigen::MatrixXcd SS(p, N);
for (int k = 0; k < p; k++)
    for (int j = 0; j < N; j++)
        SS(k, j) = S[k][j];

Eigen::MatrixXcd AA(M, p);
for (int k = 0; k < M; k++)
    for (int j = 0; j < p; j++)
        AA(k, j) = A[k][j];
// recived signal  without noise
Eigen::MatrixXcd x1 = AA * SS;
//add noise
float Pn = pow(10.0, (-1 * SNRdb / 10));
std::default_random_engine generator;
std::normal_distribution<double> distribution(0.0, 1.0);
Eigen::MatrixXcd randomnumber(M, N);
for (int k = 0; k < M; k++) {
    for (int j = 0; j < N; j++) {
        complex<double> G = distribution(generator) + 1i *
distribution(generator);
        randomnumber(k, j) = G;
    }
    randomnumber = randomnumber * sqrt(.5) * sqrt(Pn);
}

x1 = x1 + randomnumber;
```

تا به اینجا سیگنال و نویز شبیه سازی شده که مانند قسمت قبل است از اینجا به بعد طراحی فیلتر بانگ است که جدید است

۵.

```
// Filter Bank
float b[] = {1, -0.09, 0.05, -0.07};
float a[] = {1, -1.96, 0.903, 0.21};
complex<double> H[L][L];
cFilterBank(H, a, b);

complex<double> y[L * M][N];
filterpass(y, x1, H);
```

ساخت فیلتر بانک و عبور سیگنال از آن با دو تابع انجام می‌شود که در پایین تر آن ها را بررسی میکنیم.

۶

```
int nUF[L + 1];
int lowrenge = -1250000;
int highrenge = 1250000;


for (int k = 0; k < L + 1; k++) {
    nUF[k] = 0;
    for (int l = 0; l < p; l++) {
        if ((f[l] > lowrenge) && (f[l] < highrenge)) {
            nUF[k] = nUF[k] + 1;
        }
    }
    lowrenge = highrenge;
    highrenge = highrenge + 2500000;
}
nUF[0] = nUF[L];
```

جداسازی تعداد سیگنال های هر باند

۷

```
for(int index=0;index<L;index++) {
    if(nUF[index] == 0){
        continue;
    }
    Eigen::MatrixXcd x (M,N);
    for (int ii = 0 ;ii<M;ii++){
        for(int jj = 0 ; jj<N;jj++){
            x(ii,jj) = y[ii+(index*M)][jj];
        }
    }
```

جداسازی خروجی هر فیلتر از داده ها و آمادگی برای اعمال الگوریتم دقت کنید حلقه for همچنان ادامه دارد تا پایان برنامه.

۸

```
Eigen::MatrixXcd tx = x.adjoint();
    Eigen::MatrixXcd Rxx = x * tx;
    Eigen::ComplexEigenSolver<Eigen::MatrixXcd> Eces;
    Eces.compute(Rxx);
    Eigen::MatrixXcd eigVec = Eces.eigenvectors();
    Eigen::MatrixXcd noiseSub(M, M - nUF[index]);
    for (int k = 0; k < M; k++) {
        for (int j = 0; j < (M - nUF[index]); j++)
            noiseSub(k, j) = eigVec(k, j);
    }
    const float degreeStep = 0.5;
    const int numberDegree = 180 / degreeStep;
    float theta[numberDegree];
    thehtaMaker(theta, degreeStep, numberDegree);
    double Pmusic[numberDegree];
    double PP;
    double maxPmusic = 0;
    double minPmusic = 0;

    ////////////////////
    Eigen::MatrixXcd ASS(1, M);
```

```
    for (int k = 0; k < numberDegree; k++) {

        ASS = Eigen::ArrayXXcd::Zero(1, M);
        for (int j = 0; j < M; j++) {
            ASS(0, j) = exp(
                    1i * static_cast<complex<double>>((-2) * j * pi * d *
sin(theta[k] / 180 * pi) / stdLambda[index]));
        }
        Eigen::MatrixXcd PPM = ASS * noiseSub * noiseSub.adjoint() *
ASS.adjoint();
        PP = abs(PPM(0, 0));
        Pmusic[k] = 1 / PP;
        if (Pmusic[k] > maxPmusic)
            maxPmusic = Pmusic[k];
    }

    for (int k = 0; k < numberDegree; k++) {
        Pmusic[k] = 10 * log10(Pmusic[k] / maxPmusic);
        if (Pmusic[k] < minPmusic)
            minPmusic = Pmusic[k];

    }

    ////////////////////////
    bool flag1 = true;
    bool flag2 = true;
    float RDOA[nUF[index]];
    int jj = 0;
    for (int k = 0; k < numberDegree - 2; k++) {
        if (Pmusic[k] > (minPmusic * 3/ 4)) {
            if (Pmusic[k] < Pmusic[k + 1]) {
                flag1 = true;
            } else
                flag1 = false;
            if (Pmusic[k + 1] < Pmusic[k + 2]) {
                flag2 = true;
            } else
                flag2 = false;
            if (flag1 && !flag2) {
                RDOA[jj] = theta[k + 1];
                jj++;
            }
        }
        if (jj == nUF[index]) {
            break;
        }
    }
    cout << "we have (" << nUF[index] << ") doa in frequence : "<<
static_cast<float>(stdFreq[index])/1000000.0<<" MHz.\n";
    for (int k = 0; k < nUF[index]; k++) {

        cout << "[ " << k + 1 << " ] " << " is \"" << RDOA[k] << "\"" <<
endl;
    }

}
return 0;
}
```

تمام کار هایی که برای یک کانال انجام شد در برنامه قبلی در اینجا برای هر کانال انجام می‌شود.حلقه for که در قسمت قبل شروع شد در این قسمت تمام می‌شود.

چون فقط دوتابع جدید داریم فقط همان توابع در اینجا بررسی می‌شود.

```
void cFilterBank(std::complex<double> H[L][L],float a[],float b[]){
    int nnn[L] = {0,1,2,3,4,5,6,7};
    int nn[L] ;
    complex<double> BF;
    complex<double> AF;
    float maxrow = 0;
    for (int ti = 0;ti < L;ti++){
        for(int k = 0;k<L;k++){
            nn[k] = nnn[k] - ti;
        }
        for (int jj = 0;jj<L;jj++){
            BF = static_cast<complex<double>>(b[0])
                + (static_cast<complex<double>>(b[1])*exp(-1i*
(2*pi*nn[jj]/L)))
                + static_cast<complex<double>> (b[2])* exp(-1i*
(4*pi*nn[jj]/L))
                + static_cast<complex<double>> (b[3])* exp(-1i*
(6*pi*nn[jj]/L));
            AF = static_cast<complex<double>>(a[0])
                + (static_cast<complex<double>>(a[1])*exp(-1i*
(2*pi*nn[jj]/L)))
                + static_cast<complex<double>> (a[2])* exp(-1i*
(4*pi*nn[jj]/L))
                + static_cast<complex<double>> (a[3])* exp(-1i*
(6*pi*nn[jj]/L));
            H[ti][jj] = BF/AF;
            if (abs(H[ti][jj]) > maxrow){
                maxrow = abs(H[ti][jj]);
            }
        }
        for(int ii= 0;ii<L;ii++){
            H[ti][ii] = H[ti][ii]/static_cast<complex<double>>(maxrow);
        }
    }
}
```

در این قسمت فیلتر بانک ساخته می‌شود.

۱۰

```
void filterpass (std::complex<double> y[L*M][N],Eigen::MatrixXcd
x1,std::complex<double> H[L][L]) {
    Eigen::MatrixXcf HH(L, L);
    Eigen::MatrixXcf hh(L, L);
    Eigen::FFT<float> fft;
    for (int ii = 0; ii < L; ii++) {
        for (int jj = 0; jj < L; jj++) {
            HH(ii, jj) = H[ii][jj];
        }
    }
    for (int ii = 0; ii < L; ii++) {
        hh.col(ii) = fft.inv(HH.row(ii));
    }
    for (int ii = 0; ii < (L * M); ii++) {
        for (int jj = 0; jj < N; jj++){
            y[ii][jj] = 0;
        }
```

```
    }
    for (int nl = 0;nl<L;nl++) {
        for (int el = 0; el < M; el++) {
            for (int n = 0; n < N; n++) {
                for (int k = n; k >n-L; k--) {
                    if (k < 0) {
                        break;
                    }
                    y[el+(M*nl)][n] = x1(el, k) *
static_cast<complex<double>>(hh(n-k, nl)) + y[el+(M*nl)][n];
                }
            }
        }
    }

}
```

برای اینکه کد نویسی مانند متلب شود در اینجا نیز ابتدا فیلتر بانک را وارد حوزه زمان کرده و سپس هر سیگنال آرایه آنتن را با هر ۸ فیلتر کانوالو می‌کنیم. آنها را در یک ماتریس ۲ بعدی ذخیره میکنیم به نوعی که M سطر اول مربوط به فیلتر یک سطر دوم مربوط به فیلتر ۲ و . . . می‌باشد.

خروجی برنامه :

we have (4) doa in frequence : 20 MHz.

[1] is "-35"

[2] is "0"

[3] is "35.5"

[4] is "71.5"

we have (2) doa in frequence : 2.5 MHz.

[1] is "-31.5"

[2] is "33"

we have (2) doa in frequence : 5 MHz.

[1] is "-41"

[2] is "38"

we have (3) doa in frequence : 7.5 MHz.

[1] is "-38"

[2] is "21"

[3] is "62.5"

we have (4) doa in frequence : 10 MHz.

[1] is "-42"

[2] is "-20.5"

[3] is "20.5"

[4] is "58.5"

we have (3) doa in frequence : 12.5 MHz.

[1] is "-43.5"

[2] is "19.5"

[3] is "40.5"

we have (4) doa in frequence : 15 MHz.

[1] is "-20"

[2] is "0"

[3] is "17.5"

[4] is "77.5"

we have (5) doa in frequence : 17.5 MHz.

[1] is "-69"

[2] is "-20"

[3] is "19.5"

[4] is "45"

[5] is "58.5"

۷- تست با سیگنال واقعی در زبانc

ابتدا در متلب با کمک فایل "convertMdatatoBinary.m" داده های آنتن واقعی را در دو فایل باینری یکی برای بخش حقیقی یکی برای بخش موهومی ذخیره می‌کنیم و فیل هارا در مسیر پروژه C کپی میکنیم کد زبان متلب به دلیل ساده بود نیازی به توضیح ندارد و در اینجا آورده نشده است.

کدC :

۱

```
#include <iostream>
#include <Eigen/Eigenvalues>
#include "complex"
#include "cmath"
#include <Eigen/Dense>
#include <fstream>

#include "random"
#define c 299792458
#define p 1                 //number of signal
#define N 12                //number of samples
#define pi 3.1415926535897
#define M 2               //number of elements
#define NSC 72               // number of subcarrier
```

۲

```
void thehtaMaker(float theta[] ,float degreeStep,int nD);
```

۳

```
using namespace std;
int main() {
    double freal[NSC][M][N];
    ifstream inr("testReal.bin", ios::in | ios::binary);
    inr.read((char *) &freal, sizeof freal);
    cout << inr.gcount() << " bytes read\n";
    inr.close();
    double fimag[NSC][M][N];
    ifstream in("testimage.bin", ios::in | ios::binary);
    in.read((char *) &fimag, sizeof fimag);
    cout << in.gcount() << " bytes read\n";
    in.close();


    int f1 = 1800000000;
    int delta = 15000;
    int fsc [NSC] ;
    float lambdasc[NSC];
    for (int index = 0 ; index<NSC;index++){
        fsc[index] = f1+(delta*index);
        lambdasc[index] = c/static_cast<float>(fsc[index]);
    }
    float d = 0.075;
```

وارد کردن اطلاعات دریافتی و همچنین مشخصات سیستم.

۴

```
float sumdeg[p] = {0};
complex<double> temp;
const float degreeStep = 0.5;
const int numberDegree = 180 / degreeStep;
float theta[numberDegree];
double Pmusic[NSC][numberDegree];
double maxPmusic = 0;
double minPmusic = 0;
double PP;
thehtaMaker(theta, degreeStep, numberDegree);
```

پیش نیاز های الگوریتم و تغییرات لازم مثلا pmusic در اینجا یک ماتریس ۷۲ سطری است که خروجی هر ساب کریر در یکی از سطر ها ذخیره می‌شود.

۵

```
for (int index = 0;index<NSC;index++) {
    Eigen::MatrixXcd x(M,N);
    for (int k = 0;k<M;k++){
        for (int q = 0; q<N;q++){
            temp = freal[index][k][q]+(1i*fimag[index][k][q]);
            x(k,q) = temp;
        }
    }
```

وارد کردن هر زیرحامل به ورودی الگوریتم دقت شود حلقه for اول تا زمانی که اعلام تکردیم در بخش های بعد بسته نشده است ثانوی بسته نمی‌شود.

۶

```
Eigen::MatrixXcd tx = x.adjoint();
Eigen::MatrixXcd Rxx = x * tx;
Eigen::ComplexEigenSolver<Eigen::MatrixXcd> Eces;
Eces.compute(Rxx);
Eigen::MatrixXcd eigVec = Eces.eigenvectors();
Eigen::MatrixXcd noiseSub(M, M - p);
for (int k = 0; k < M; k++) {
    for (int j = 0; j < (M - p); j++)
        noiseSub(k, j) = eigVec(k, j);
}
```

ساخت شبه‌فضای نویز

۷

```
Eigen::MatrixXcd ASS(1, M);
for (int k = 0; k < numberDegree; k++) {
    ASS = Eigen::ArrayXXcd::Zero(1, M);
    for (int j = 0; j < M; j++) {
        ASS(0, j) = exp(
                1i * static_cast<complex<double>>((-2) * j * pi * d *
sin(theta[k] / 180 * pi) / lambdasc[index]));
    }
    Eigen::MatrixXcd PPM = ASS * noiseSub * noiseSub.adjoint() *
ASS.adjoint();
    PP = abs(PPM(0, 0));

    Pmusic[index][k] = 1 / PP;
    if (Pmusic[index][k] > maxPmusic)
```

```
        maxPmusic = Pmusic[index][k];
}
```

انجام محاسبات و جدا سازی نقطه ماکزیمم، درست مثل قبل

۸

```
for (int k = 0; k < numberDegree; k++) {
    Pmusic[index][k] = 10 * log10(Pmusic[index][k] / maxPmusic);
    if (Pmusic[index][k] < minPmusic)
        minPmusic = Pmusic[index][k];
}
```

لگاریتم گیری و جدا سازی مینیمم

۹

```
    bool flag1 = true;
    bool flag2 = true;
    float RDOA[p];
    int jj = 0;
    for (int k = 0; k < numberDegree-2; k++) {
        if (Pmusic[index][k] > (minPmusic * 2 / 3)) {
            if (Pmusic[index][k] < Pmusic[index][k + 1]) {
                flag1 = true;
            } else
                flag1 = false;
            if (Pmusic[index][k + 1] < Pmusic[index][k + 2]) {
                flag2 = true;
            } else
                flag2 = false;
            if (flag1 && !flag2) {
                RDOA[jj] = theta[k + 1];
                jj++;
            }
        }
        if (jj == p) {
            break;
        }
    }
    cout << "angel SC ("<< index+1 << ") is: ";
    for (int k = 0; k < p; k++) {

        cout << "we have [ " << k + 1 << " ] " << " is \"" << RDOA[k] <<
"\"    ," << endl;
        sumdeg[k] = RDOA[k]+sumdeg[k];
    }

}
```

جدا سازی نقاط پیک نمودار چاپ هر کدام در خروجی

۱۰

```
  float avedeg[p];
     for (int k= 0;k<p;k++){
         avedeg[k]=sumdeg[k] /NSC;
         cout<<endl<<"the average is degree is : (\""<<avedeg[k]<<"\")";
     }
    return 0;
}
```

خروج میانگین زاویه ها و اتمام برنامه

تابع استفاده شده در قسمت اول اورده شده است.

خروجی:

13824 bytes read

13824 bytes read

angel SC (1) is: we have [1] is "-13.5" ,

angel SC (2) is: we have [1] is "-13" ,

angel SC (3) is: we have [1] is "-13" ,

angel SC (4) is: we have [1] is "-13" ,

angel SC (5) is: we have [1] is "-12" ,

angel SC (6) is: we have [1] is "-13" ,

angel SC (7) is: we have [1] is "-12.5" ,

angel SC (8) is: we have [1] is "-12.5" ,

angel SC (9) is: we have [1] is "-13.5" ,

angel SC (10) is: we have [1] is "-12.5" ,

angel SC (11) is: we have [1] is "-12.5" ,

angel SC (12) is: we have [1] is "-13.5" ,

angel SC (13) is: we have [1] is "-13" ,

angel SC (14) is: we have [1] is "-12" ,

angel SC (15) is: we have [1] is "-13" ,

angel SC (16) is: we have [1] is "-13.5" ,

angel SC (17) is: we have [1] is "-12" ,

angel SC (18) is: we have [1] is "-13.5" ,

angel SC (19) is: we have [1] is "-13" ,

angel SC (20) is: we have [1] is "-12" ,

angel SC (21) is: we have [1] is "-12" ,

angel SC (22) is: we have [1] is "-12" ,

angel SC (23) is: we have [1] is "-12.5" ,

angel SC (24) is: we have [1] is "-12" ,

angel SC (25) is: we have [1] is "-14" ,

angel SC (26) is: we have [1] is "-14" ,

angel SC (27) is: we have [1] is "-13" ,

angel SC (28) is: we have [1] is "-13.5" ,

angel SC (29) is: we have [1] is "-12" ,

angel SC (30) is: we have [1] is "-14" ,

angel SC (31) is: we have [1] is "-14" ,

angel SC (32) is: we have [1] is "-12.5" ,

angel SC (33) is: we have [1] is "-12" ,

angel SC (34) is: we have [1] is "-12.5" ,

angel SC (35) is: we have [1] is "-12" ,

angel SC (36) is: we have [1] is "-11.5" ,

angel SC (37) is: we have [1] is "-13.5" ,

angel SC (38) is: we have [1] is "-13" ,

angel SC (39) is: we have [1] is "-13.5" ,

angel SC (40) is: we have [1] is "-13" ,

angel SC (41) is: we have [1] is "-14" ,

angel SC (42) is: we have [1] is "-14" ,

angel SC (43) is: we have [1] is "-12" ,

angel SC (44) is: we have [1] is "-13.5" ,

angel SC (45) is: we have [1] is "-13.5" ,

angel SC (46) is: we have [1] is "-13" ,

angel SC (47) is: we have [1] is "-13" ,

angel SC (48) is: we have [1] is "-13" ,

angel SC (49) is: we have [1] is "-13.5" ,

angel SC (50) is: we have [1] is "-12.5" ,

angel SC (51) is: we have [1] is "-12" ,

angel SC (52) is: we have [1] is "-12.5" ,

angel SC (53) is: we have [1] is "-13.5" ,

angel SC (54) is: we have [1] is "-13.5" ,

angel SC (55) is: we have [1] is "-13" ,

angel SC (56) is: we have [1] is "-13.5" ,

angel SC (57) is: we have [1] is "-13" ,

angel SC (58) is: we have [1] is "-13" ,

angel SC (59) is: we have [1] is "-12" ,

angel SC (60) is: we have [1] is "-14" ,

angel SC (61) is: we have [1] is "-14" ,

angel SC (62) is: we have [1] is "-14.5" ,

angel SC (63) is: we have [1] is "-14" ,

angel SC (64) is: we have [1] is "-12" ,

angel SC (65) is: we have [1] is "-13.5" ,

angel SC (66) is: we have [1] is "-13.5" ,

angel SC (67) is: we have [1] is "-12.5" ,

angel SC (68) is: we have [1] is "-14" ,

angel SC (69) is: we have [1] is "-13" ,

angel SC (70) is: we have [1] is "-12" ,

angel SC (71) is: we have [1] is "-13" ,

angel SC (72) is: we have [1] is "-12" ,

the average is degree is : ("-12.9583")