RADAR IMAGING – HOMEWORK 1

Kotsaba Mykhaylo

TASK 1
Model of the received signal and system design

a.

For the 3D model we make the same assumption we have made for the 2D. The emitting point is far away, so it's possible to assume that the rays are all parallel and using 3D geometry we can find the equation of the received signal at (nx,ny) coordinate of the antenna.

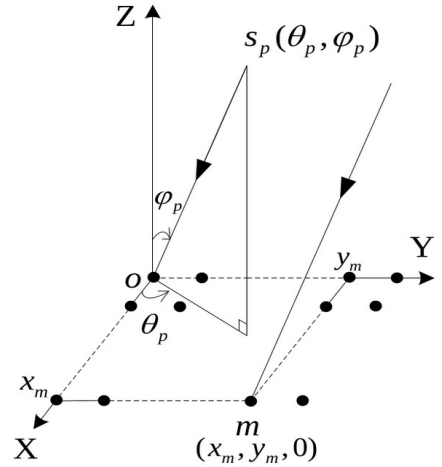


Figure 1.
From Spherical coordinates to Cartesian (x,y,z)

$$S_{RX}^{n,m} = ae^{-j\frac{2\pi}{\lambda}(r+ndx\sin\varphi\cos\theta+mdy\sin\varphi\sin\theta)}$$

$$k = \frac{2\pi}{\lambda}$$

M is the m-th point on the y axis and n is the n-th point on the x axis. Dx and dy are the spaces between the two near antennas points that are going to be selected to be able to reveal the frequency of our wave. If we want a more compact way to write it, we can use the scalar product of the vectors K and B.

$$\mathbf{K} = k \cdot \sin\varphi \cdot \cos\theta \cdot \mathbf{i_x} + k \cdot \sin\varphi \cdot \sin\theta \cdot \mathbf{i_y} + k \cdot \cos\varphi \cdot \mathbf{i_z}$$

$$\mathbf{B} = n \cdot dx \cdot \mathbf{i_x} + m \cdot dy \cdot \mathbf{i_y} + 0 \cdot \mathbf{i_z}$$

So we can write:

$$S_{RX}^{n,m} = ae^{-j\frac{2\pi}{\lambda}r}e^{-j\mathbf{B}(n,m)\cdot\mathbf{K}}$$

b.

The frequency is 9.6 GHz and the velocity in the medium (air) is 300.000.000m/s. We obtain that the wavelength is 31.25 mm. Now we need to find a relation with the dx and dy.
The spacial frequency fx and fy can be obtained from the k relations.

$$f_x = \frac{\sin\varphi\cos\theta}{\lambda}, f_y = \frac{\sin\varphi\sin\theta}{\lambda}$$

And from the sampling theorem we know that

$$f_{sx} = \frac{1}{dx} > 2f_x, f_{sy} = \frac{1}{dy} > 2f_y$$

So we get this

$$dx < \frac{\lambda}{2sin(\varphi)\cdot\cos(\theta)}; dy < \frac{\lambda}{2sin(\varphi)\cdot sin(\theta)}$$

That in case of cos()*sin()=1 or sin()*sin()=1, equals to

$$dx = dy < \frac{\lambda}{2}$$

The number of points depends on the angle of arrival. Even if it's not accurate we can set as 2x2 the minimum array. Otherwise it's the maximum between Lambdax/dx and Lambday/dx.

TASK 2
Implementation of DoA estimation by FFT

a.
Using the mesh-grid function we create the space of antennas array in Matlab.

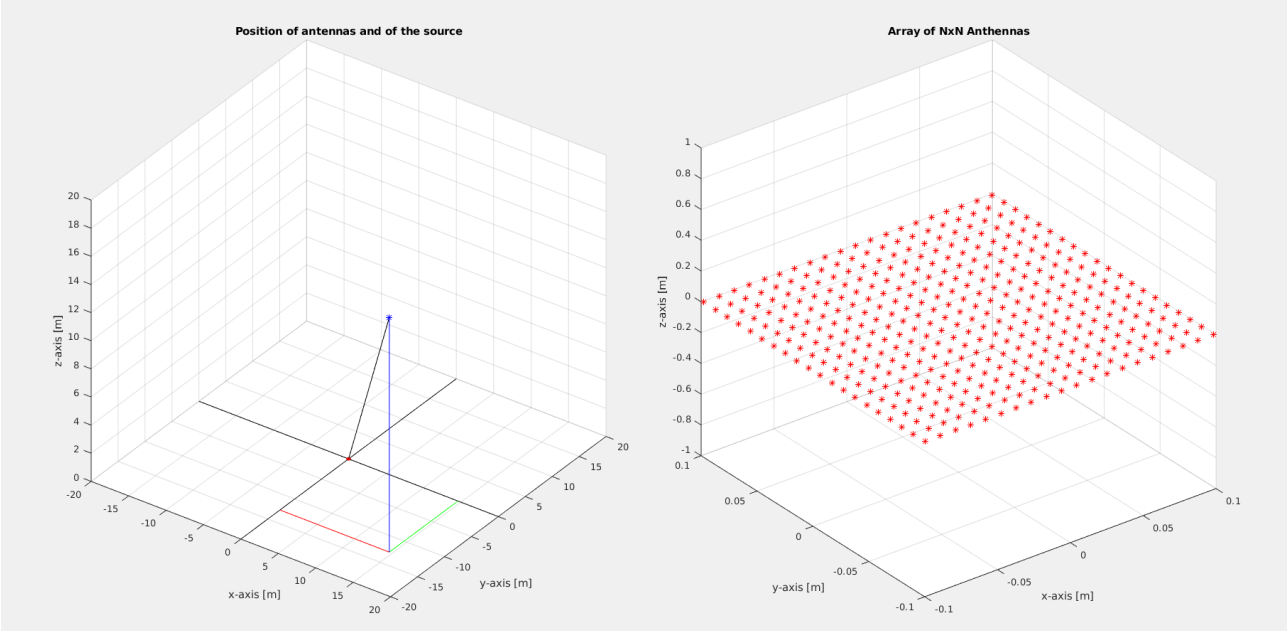
```
%  TASK 2 - Implementation of DOA estimation by 2D-FFT
%% A.
N=20;                                   % NxN Array of antennas
f0=9.6e9;
lambda = 3e8/f0;
ds=lambda/3;
[X,Y]=meshgrid(((-(N-1)/2):(N-1)/2)*ds,((-(N-1)/2):(N-1)/2)*ds);
```

We also create a vector P to save the coordinates of the source and we use a random function to create the coordinates inside the defined limits by the homework.

b.

```
%% B.
limit=[10, 20];
p=[0,0,0];

% We create a source point in our X,Y,Z limits coordinations.
while ((-limit(1)<=p(1) && p(1)<limit(1) ) || (-limit(1)<=p(2) && p(2)<limit(1)) || (-limit(1)<=p(3) && p(3)<limit(1) ))
    p=[2*limit(2)*rand-limit(2), 2*limit(2)*rand-limit(2), limit(2)*rand];
end
real_distance=norm(p);
real_phi=acos(p(3)/real_distance);
real_teta=atan(p(1)/p(2));
```

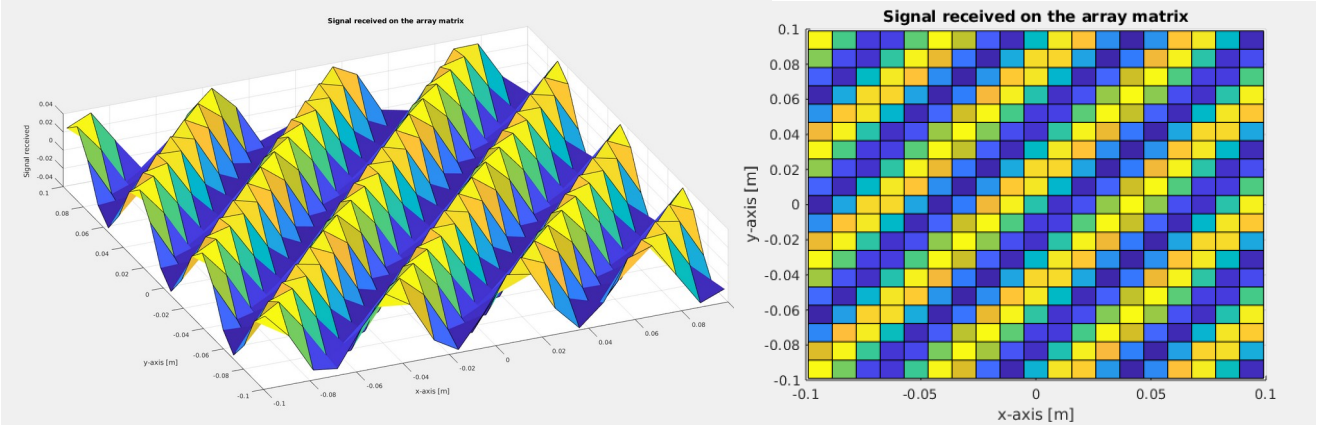
After we have defined the source point we can plot the graphs. We have added some lines to make more clear the point orientation in the 3D space. On the left we can observe the source point orientation respect the array. On the right instead we can observe the antennas array NxN.



c.
Using the formula of the transmitted signal, we implement in Matlab the algorithm.
First of all we define the source amplitude of the signal. We calculate the distance from the (n,m) point to the source and in the end we calculate the received signal and store it in a matrix.

```
Vamp=1;
distance_m_n=sqrt((p(1)-X).^2 +(p(2)-Y).^2 +(p(3)).^2);
TSignal = Vamp./distance_m_n.*exp(-1j.*(2*pi/lambda).*distance_m_n);
```

We can plot the real part of the received signal

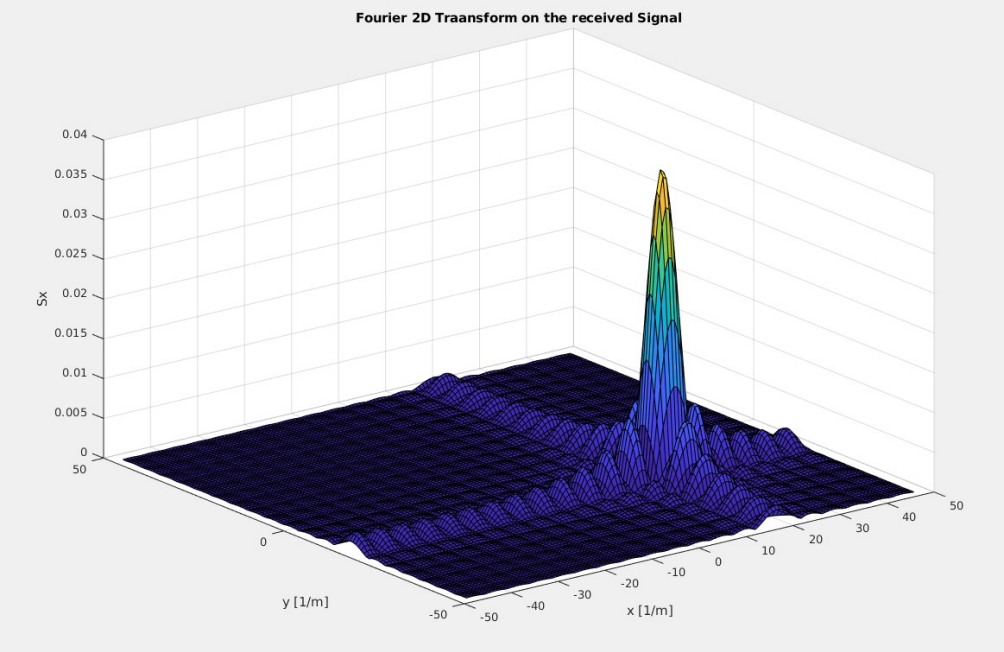


On the left we see the signal sampled by the array from the side view. On the left we observe the signal from the top view. It's easy to notice the change of the lambda on the x axis and on the y axis.

d.

Now it's the time to use the 2D DFT and find the space frequency Fx and Fy

```
Ns = 128;
df=(1/ds)/Ns;
FSignal=fftshift(fft2(TSignal,Ns,Ns));
[frx, fry]=meshgrid(((-(Ns)/2):(Ns-1)/2)*df, ((-(Ns)/2):(Ns-1)/2)*df);

MaxVal = max(max(abs(FSignal)));                    % Max value of the dft
[xm,ym]=find(abs(FSignal)==MaxVal);                 % Index of the max value
Fx=frx(1,xm);                                       % Fx value
Fy=fry(ym,1);                                       % Fy value
```



Ns is the size of the matrix after the dft transformation, df is the space interval in the frequency domain. The Fsignal must be normalized after the fft2 according the Matlab help fft2 command example. In the end we find the value of Fx and Fy.

At this point it's easy to find the two desired value

```
Real paramaeters:
Phi: 49.27 gradi
Teta -48.90 gradi
----------------------
Calculated :
Phi: 48.37 gradi
Teta -48.81 gradi
```

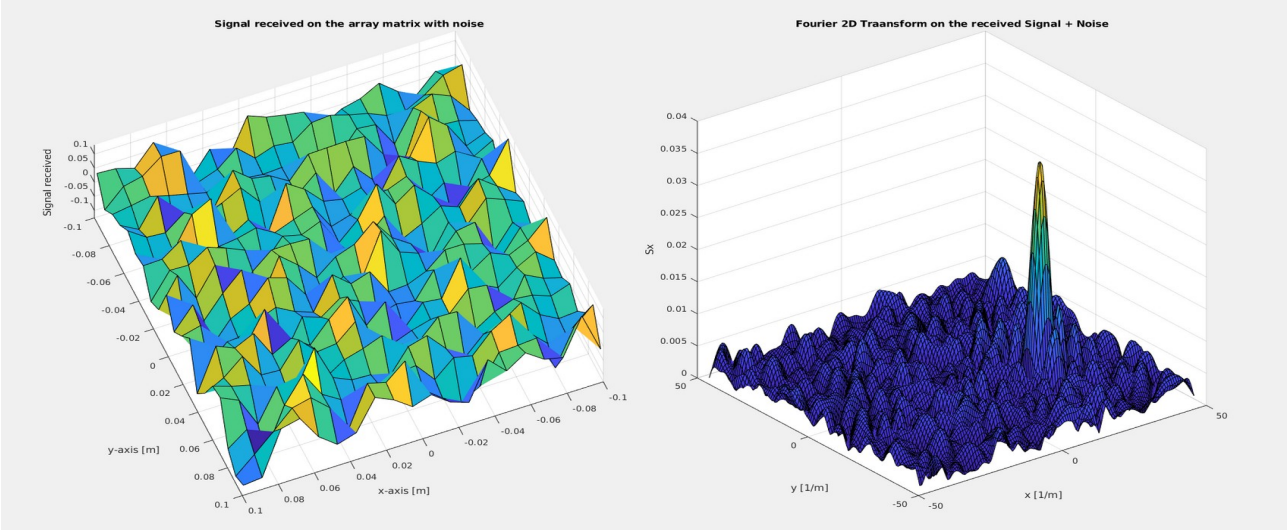
We can now calculate the real parameters from the P point and the calculated ones theoretically. As we can see on the left table the results are almost perfect.

e.

Now it's time to add some noise.

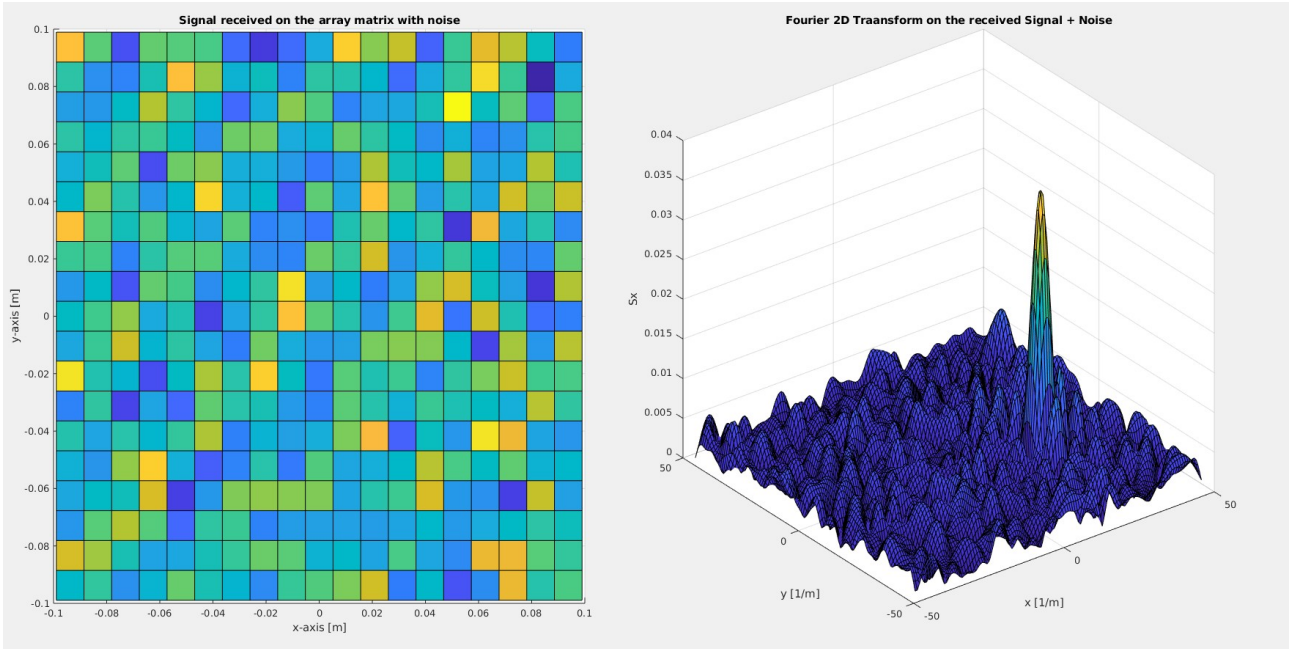
```
SNR=100;

Amp=sqrt(1/SNR)*rand(N);
phase=2*pi*rand(N);
TSignalNoise=TSignal+Amp.*exp(-1j.*phase);
FSignalNoise=fftshift(fft2(TSignalNoise,Ns,Ns))/N/N;

MaxValNoise = max(max(abs(FSignalNoise)));                  % Max value of the dft with noise
[xmnoise,ymnoise]=find(abs(FSignalNoise)==MaxValNoise);     % Index of the max value noise
FxNoise=frx(1,xmnoise);                                     % Fx value noise
FyNoise=fry(ymnoise,1);                                     % Fy value Noise

tetaNoise=atan(FyNoise/FxNoise);
phiNoise=asin(FxNoise*lambda/cos(tetaNoise));
```

SNR is the Signal Noise Ratio. We can generate some complex white noise by creating amplitude proportional to the square root of the 1/SNR and than the phase. After that we create the Noisy Signal Matrix by summing the noise matrix to the signal.




```
Calculated with Noise:
Phi: 49.74 gradi
Teta -47.49 gradi
```

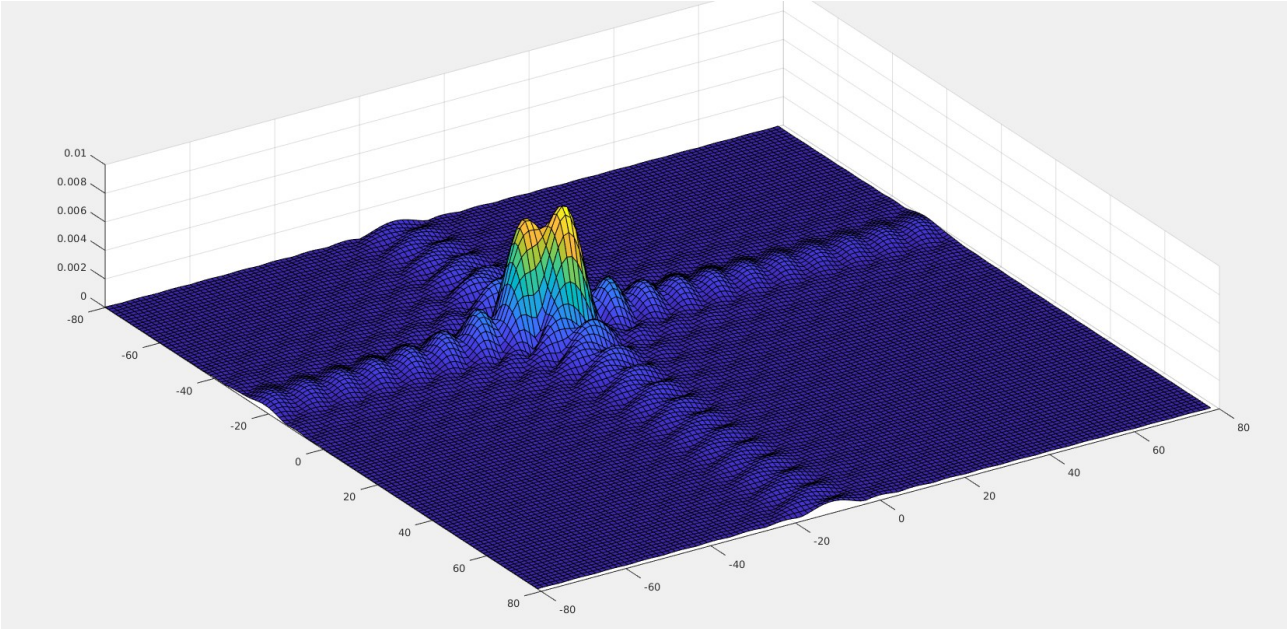
Even if from the time domain it's almost impossible to distinguish the signal due to the low SNR, from the frequency domain it's very easy and we can again calculate our parameters.



f.

Now we try to add another point in the space and calculate the new Signal received from two points

```
p2=[0,0,0];

while ((-limit(1)<=p2(1) && p2(1)<limit(1) ) || (-limit(1)<=p2(2) && p2(2)<limit(1)) || (-limit(1)<=p2(3) && p2(3)<limit(1) ))
    p2=[2*limit(2)*rand-limit(2), 2*limit(2)*rand-limit(2), limit(2)*rand];
end

real_distance2=norm(p2);                                    % Real distance from the [0,0,0]
real_phi2=acos(p2(3)/real_distance2);                       % Real elevation angle of the point
real_teta2=atan(p2(2)/p2(1));

% Signal of 2 sources
distance_m_n2 = sqrt((p2(1)-X).^2 +(p2(2)-Y).^2 +(p2(3)).^2);
TSignal2 = Vamp./distance_m_n2.*exp(-1j.*(2*pi/lambda).*distance_m_n2)+Vamp./distance_m_n.*exp(-1j.*(2*pi/lambda).*distance_m_n);

Distancepp=sqrt( (p(1)-p2(1)).^2+(p(2)-p2(2)).^2+(p(3)-p2(3)).^2);
```



I’ve tried a lot of parameters settings and the two signals in the frequency domain are separated till the distance is above 1 m. Otherwise It’s not possible to separate them.

TASK 3
Implementation of DoA by back-projection

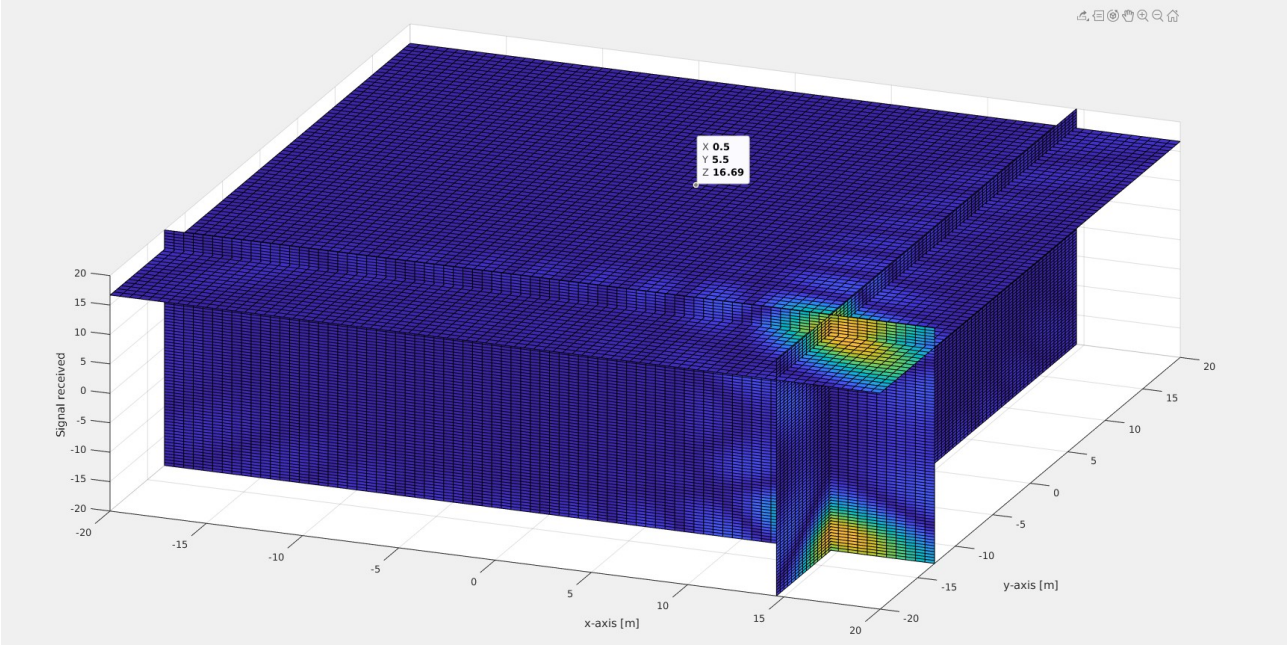
a.
We set a dx to discrete our 3D space, if we use a narrower dx, the calculation's time will be too long. We will use dx = 0.5, that empirically should be fine.

```
dx=0.5;
space_dx=-limit(2):dx:limit(2);
dim=size(space_dx,2);

[XX,YY,ZZ]=meshgrid(space_dx);
BackSignal=zeros(dim, dim, dim);
```

b.
We use three for cycles to calculate the backed signal at each point of the space, given by the contribution of all the (n, m) antennas of the array.

```
for ii=1:dim
    for jj=1:dim
        for hh=1:dim
            DistPositive=sqrt((X-space_dx(ii)).^2 +(Y-space_dx(jj)).^2 +(space_dx(hh)).^2);
            BackSignal(jj,ii,hh)=sum(TSignal.*DistPositive.*exp(1j*(2*pi/lambda)*DistPositive),'all');
        end
    end
end

xslice=[p(1)];  %% Slicing in the real p point
yslice=[p(2)];
zslice=[p(3)];

figure
slice(XX,YY,ZZ,abs(BackSignal),xslice,yslice,zslice);
xlabel('x-axis [m]');ylabel('y-axis [m]');zlabel('Signal received ');
```

In the end we plot three planes that pass per P-source-point of the source and we can see that the signal is more intense near the intersection, because it's a composition, the signal has zero delta-phase in that point. So the result is right. Obviously we have a symmetry on the z positive and negative.



TASK 4
Implementation of 3D target localization by default-array (OPTIONAL)

The problem is to find the distance of the object. To solve it I've tried to put 2 arrays, one at -D and the other at D from the origin (0,0,0) moving along the x-axis.

```
N=100;                                  % NxN Array of antennas
f0=9.6e9;
lambda = 3e8/f0;
ds=lambda/10;
D=5;
Vamp=1;

[X1,Y1]=meshgrid(((-(N-1)/2-D/ds):(N-1)/2-D/ds)*ds,((-(N-1)/2):(N-1)/2)*ds);
[X2,Y2]=meshgrid(((-(N-1)/2+D/ds):(N-1)/2+D/ds)*ds,((-(N-1)/2):(N-1)/2)*ds);

limit=[10, 20];
p=[0,0,0];
```

After creating the point P and the Signal generated on both arrays, I've done the fft2 on both signals and find the Fx and Fy for both arrays.

```
TSignal1 = Vamp./distance_m_n1.*exp(-1j.*(2*pi/lambda).*distance_m_n1);
TSignal2 = Vamp./distance_m_n2.*exp(-1j.*(2*pi/lambda).*distance_m_n2);

Ns = 128;
df=(1/ds)/Ns;
FSignal1=fftshift(fft2(TSignal1,Ns,Ns));
FSignal2=fftshift(fft2(TSignal2,Ns,Ns));

[frx, fry]=meshgrid(((-(Ns)/2):(Ns-1)/2)*df, ((-(Ns)/2):(Ns-1)/2)*df);

MaxVal1 = max(max(abs(FSignal1)));                  % Max value of the dft
[xm1,ym1]=find(abs(FSignal1)==MaxVal1);               % Index of the max value
Fx1=frx(1,xm1);                                     % Fx value
Fy1=fry(ym1,1);

MaxVal2 = max(max(abs(FSignal2)));                  % Max value of the dft
[xm2,ym2]=find(abs(FSignal2)==MaxVal2);               % Index of the max value
Fx2=frx(1,xm2);                                     % Fx value
Fy2=fry(ym2,1);
```

```
teta1=atan(Fy1/Fx1);
phi1=deg2rad(abs(rad2deg(asin(Fx1*lambda/cos(teta1)))));

teta2=atan(Fy2/Fx2);
phi2=deg2rad(abs(rad2deg(asin(Fx2*lambda/cos(teta2)))));

z1=[sin(phi1)*cos(teta1),sin(phi1)*sin(teta1), cos(phi1);];
z2=[sin(phi2)*cos(teta2),sin(phi2)*sin(teta2), cos(phi2)];

A=[z1(1), -z2(1); z1(2), -z2(2)];
b=[2*D ; 0];
K=inv(A)*b;

PUNTO_DIST=[D+z2(1)*K(2), z2(2)*K(2), z2(3)*K(2)];
PUNTO_DIST_2=[-D+z1(1)*K(1), z1(2)*K(1), z1(3)*K(1)];
```

Now that I have all the parameters I need, Phi and Theta for both array, I can write a system to solve the intersection between two lines. Using the parametric formulas and writing the system Ak=b.

After some launches seems not working correctly and I can't figure why. The system seems right and the parameters too.

It was just a try TASK 4, unfortunately it doesn't work or it works but it's very inaccurate.