Linear Processing of an Antenna Array

EEL 6935 - SPRING.2018 Hector Lopez 1-13-2018

Description

Consider the most trivial of all linear array processors that has beamformer weight values `w_i = 1 for every i = 1,2...M` assume that a unit-value signal `(m(t)=1)` on a carier frequency `fc` is arriving at your antenna array with a direction of arrival `theta`. Assume inter-element spacing equal to half the carrier wavelenth , `d=lambda_c/2`.

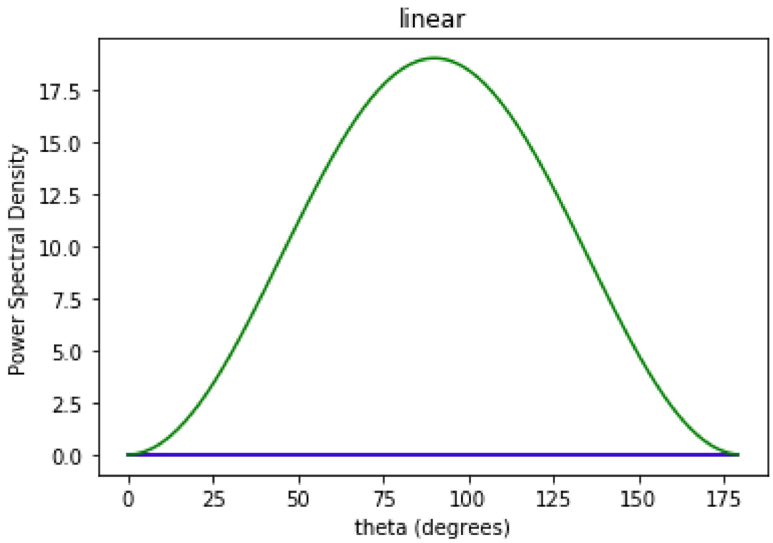
Plot the array output `|y|^2` in plain scale and in dB log-scale as a function of `theta` for `M=3,5,10` (this is known as a power pattern or power beam pattern).

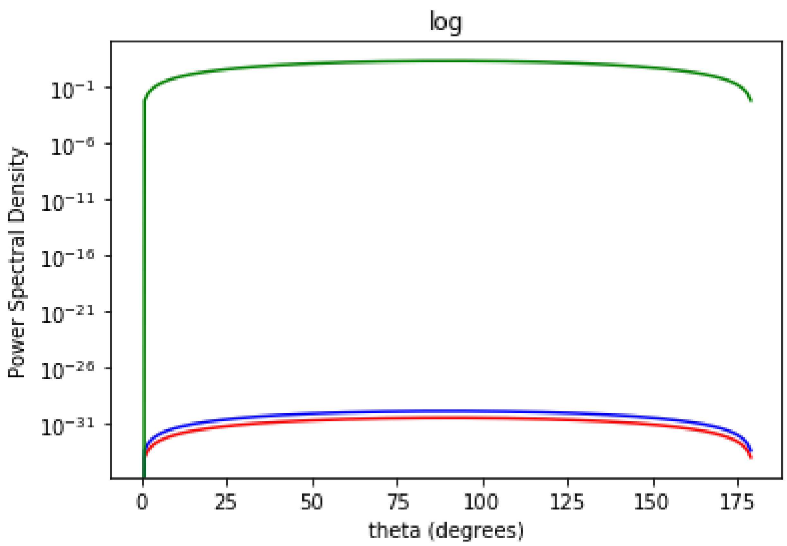
What do you observe? Discuss your findings. Would an array with fixed unit weights like that (plain adder) be useful in some way?

In [143]:

```
import scipy.integrate as integrate
from numpy import *


# create a function that will process different number of elements and r
eturn with a P.vs.theta curve
def Processor(M):
    #number of array elements 'M'
    elements = arange(1,M);

    #weights as unit vector
    weights = ones(M,dtype=complex);

    #unit value signal on carrier frequency fc
    fc=120;

    # create a time vector  with a max time T (seconds)
    max_time = 20;
    time= arange(0,max_time);

    #initialize input signal vector for all elements over time
    x = zeros((M,max_time),dtype=complex);

    Y = zeros((max_time),dtype=complex);
    y = zeros((M),dtype=complex);

    #arriving signal as unit-value at each element over time
    m = ones((M,max_time),dtype=complex);

    #Array Response Vector
    lambda_c = 1;
    spacing = lambda_c / 2;
```

```
    S = zeros(M,dtype=complex);

    #create a range of theta for plotting
    theta_min = 0;
    theta_max = 180;
    theta = arange(theta_min,theta_max,1);

    #Power Array vs theta matrix
    P = zeros(theta_max);

    for th in nditer(theta.T):
        for d in nditer(elements.T):
            #create an arrray response vector for this theta inst.
            S[d]= exp(-1j*2*pi*(((d-1)*spacing)/lambda_c))*sin(th*(pi/18
0));
        for t in nditer(time.T):
            #Step through the time
            for i in nditer(elements.T):
                #create an input signal for all the elements
                x[i,t] = m[i,t]*exp(1j*2*pi*fc*t)*S[i];
                #multiply inputs by the weights
                y[i] =  conjugate(weights[i])*x[i,t];
            #sum all the inputs
            Y[t] = sum(y);
        #compute the power spectral density integral(|y(t)|^2)
        psd = trapz(square(absolute(Y)));
        #print("theta:",th,"psd:",psd );
        P[th]=psd;
    return P;

#Create a plot to show the output with respect to theta
import matplotlib.pyplot as plt
plt.plot(theta, Processor(3), 'r-',Processor(5), 'b-',Processor(10), 'g-
');
plt.ylabel('Power Spectral Density');
plt.yscale('linear');
plt.title('linear');
plt.xlabel('theta (degrees)');
plt.show()

plt.plot(theta, Processor(3), 'r-',Processor(5), 'b-',Processor(10), 'g-
');
plt.ylabel('Power Spectral Density');
plt.yscale('log');
plt.title('log');
plt.xlabel('theta (degrees)');
plt.show()
```


linear
Power Spectral Density
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
25
50
75
100
125
150
175
theta (degrees)


log
Power Spectral Density
10^{-1}
10^{-6}
10^{-11}
10^{-16}
10^{-21}
10^{-26}
10^{-31}
0
25
50
75
100
125
150
175
theta (degrees)