Array_Processing2

J.-F. Bercher

January 12, 2014

Author: J.-F. Bercher
date: january 09, 2014
Updates : january 12, 2014

1 Some useful definitions

We begin by some useful definitions

- a simple routine that returns a comumn vector from any array
- two functions which return the steering vectors for a Unifor Linear Array (ULA), in the case of plane waves and of circular waves

In [3]:
```
%config InlineBackend.figure_format = 'svg'
def col(v):
     """ transforme un array en vecteur colonne"""
     v=asmatrix(v.flatten())
     return reshape(v,(size(v),1))


def steeringvec_plane(theta,N,d,lamb):
    """ plane waves --\n
    Returns the steering vector for a plane wave\n
    illuminating an ULA
    """
    k=arange(0,N)
    u=d*sin(theta*pi/180)/lamb
    a=exp(1j*2*pi*k*u)
    return a


def steeringvec_circ(H,D,N,d,lamb):
    """ circular waves
    a([D,H])=[.... 1/Rk*exp(-j*2*pi*f0*Rk/c) ...]
    a([D,H])=[.... 1/Rk*exp(-j*2*pi*Rk/lambda) ...]
    with Rk^2= H^2+(D+(k-1)*d)**2
    """
    k=arange(0,N)
    R= sqrt(H^2+(D+(k-1)*d)**2)
    a=(1/R)*exp(-1j*2*pi*R/lamb)
    return col(a)
```

2 Simulation of data

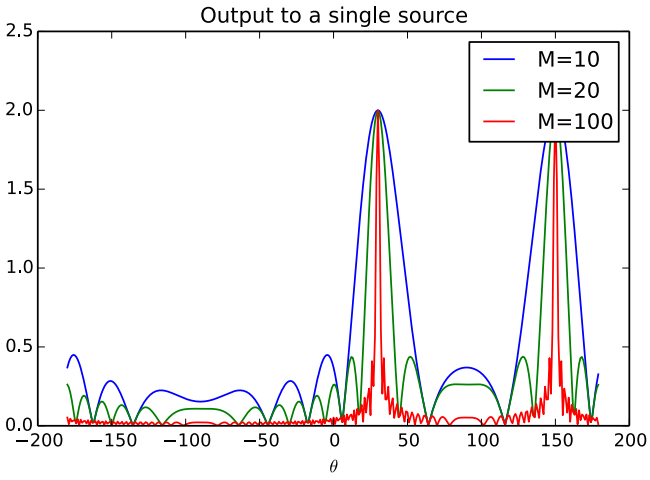
In [4]:
```
# We have a Uniform Linear Array, with

c=346               # sound velocity, in  m/s
f0=1000             # frequency Hz
lamb=c/f0           # wave length
d= lamb/4           # distance between sensors
M=10;               # Number of sensors
mintheta=-180; steptheta=1; maxtheta=180
Nb_snap=500;        # Number of snapshots
```

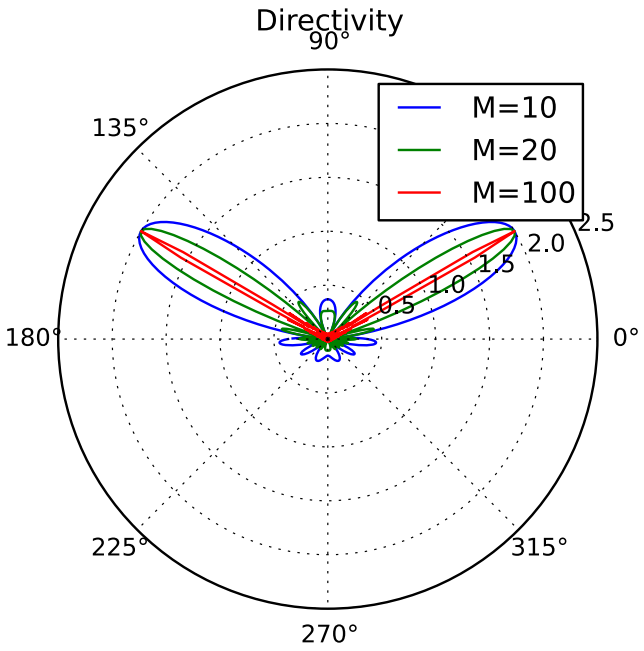
3 Determination of the directivity diagram

We simulate a single source in the direction 30˚,, and plot the directivity diagram for several values of M, the number of sensors. For each θ, we compute the output of the beamformer, ie the spatial filter $w = \frac{a}{a^+ a}$, as $z(\theta) = w^+ x$

In [5]:
```
#########################################
# Directivity diagram
#########################################
# We simulate a single source in the direction 30˚,, and plot the directivity diagram
# values of M, the number of sensors


theta=30


for M in (10,20,100):
    x=2*steeringvec_plane(30,M,d,lamb)             # <-- data
    vtheta=arange(mintheta,maxtheta,steptheta)     # range of values of angles theta to
    m=0
    S=zeros(size(vtheta))
    for theta in vtheta:                            # For each theta, we compute the ou
                                                    # ie the spatial filter $w=\frac{a}
        a=steeringvec_plane(theta,M,d,lamb)
        w= a/vdot(a,a)       # vdot(a,a)=mat(a).H*mat(a) # beamforming : norm 1 vector
        S[m]=abs(vdot(w,x))                         # Output of the beaformer : $z(\the
        m=m+1

    figure(1)
    plot(vtheta, abs(S), label="M={}".format(M))
    figure(2)
    polar(vtheta*pi/180, (abs(S)), label="M={}".format(M))

figure(1)
xlabel('$\\theta$')
title('Output to a single source')
legend()
figure(2)
title('Directivity')
legend()
```

Out [5]:
```
<matplotlib.legend.Legend at 0x7f322dc42050>
```





- We see that the main lobe's width, or beanwidth, decreases with M. Actually, we know that in terms of spatial frequencies, the width is ~ $1/M$.
- The second point is that we can observe a second, phantom source at 180˚,. This source corresponds to the fact that we do not know if the real source is located above or below the array plane. Evidently, from physical reasons, we might know that the source is "above", but this is not included in the model.

4 Fourier Analysis

The beamforming, in the case of plane eves, is directly related to the spatial Fourier transform of the signals recorded at the sensors of the antenna. Actually, we simply have that

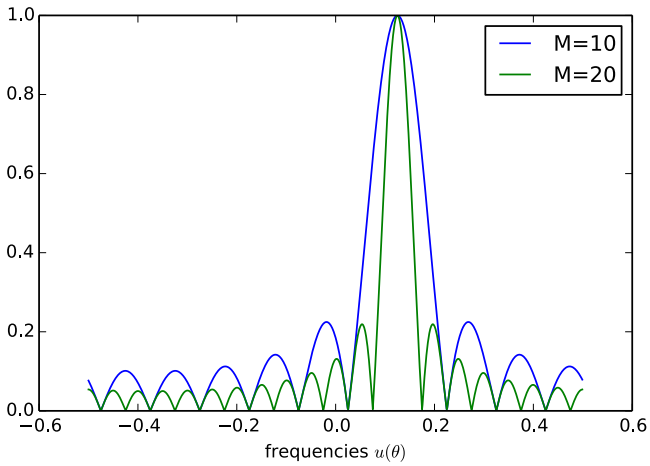
$$S(\theta) = w^{+}x = \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} x(k) \exp(-j2\pi u(\theta)) = \frac{1}{\sqrt{M}} FT\{x\} \tag{1}$$

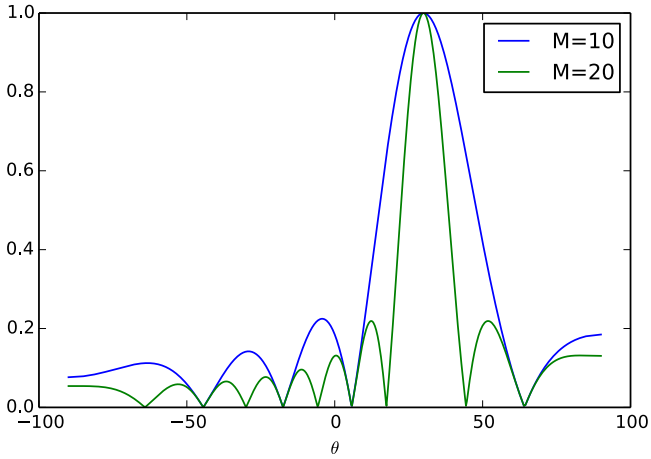
Furthermore,the finite size of the array – the finite number of sensors, induce a convolution by a cardinal sine. As a consequence, for a single source, instead of observing a pure spectral line (a Dirac impulse), we got a cardinal sine, with a main-lobe of width $1/M$, which induces a limitation in resolution.

This is illustrated below.

In [27]:
```
from numpy.fft import fft, ifft

close('all')
for M in (10,20):
    x=steeringvec_plane(30,M,d,lamb)
    N=1024
    x=array(x).flatten()
    S=1/M*abs(fft(x,N));  # with zero padding
    # the prefactor 1/M normalizes the amplitude to 1
    S=fftshift(S);
    f=(arange(N)/N-0.5)
    figure(1)
    plot(f,S,label="M={}".format(M))    # with frequencies between +-1/2
    # The normalized frequencies are actually d sin(theta)/lambda
    # Thus
    figure(2) # Here, we plot with respect to $\theta$ instead of $u(\theta)$
              # There is a simple change of variable in representation which
              # leads to a kind of distorsion of the sidelobes -- the very same observ
              # holds for the output of the beamformer obtained above
    plot( 180/pi*arcsin( lamb/d*f),S,label="M={}".format(M))

figure(1)
xlabel("frequencies $u(\\theta)$")
legend()
figure(2)
xlabel("$\\theta$")
legend()

Smax=max(S)
fmax=(find(S==Smax))[0]/N-0.5
print("The maximum of the FT accurs at $f={}$, that is an \
angle of {:2.2f}˚,".format(fmax,180/pi*arcsin( lamb/d*fmax)))
```

```
The maximum of the FT accurs at $f=0.125$, that is an angle of 30.00˚,
```





5 Spectrum at the beamformer output - resolution

We now consider a mixture of three sources with a Gaussian amplitude (`randn` function), in the directions (-20˚38˚, 40˚) and with variances (1, 4, 4).

- we compute the correlation matrix, using K snapsots x_k, according to

$$R = \frac{1}{K}\sum_{k=1}^{K} x_k x_k^+ \tag{2}$$

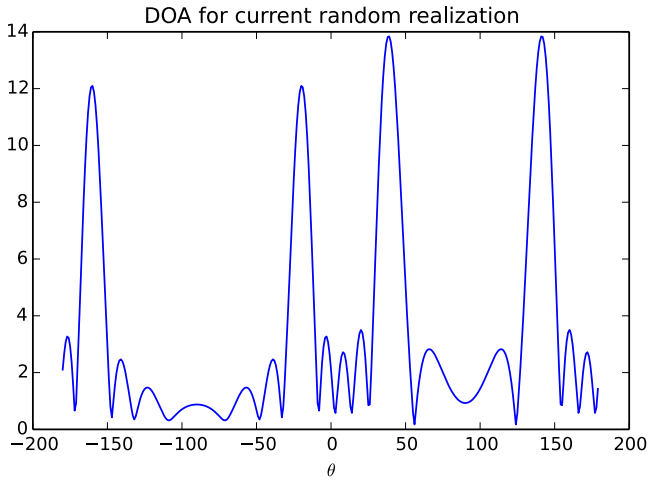
In [28]:
```
"""
"""

# Simulated data
def randn_m(*args):
    return matrix(randn(*args))

x=2*float(randn(1))*steeringvec_plane(-20,M,d,lamb) + \
    2*float(randn(1))*steeringvec_plane(38,M,d,lamb) + \
    float(randn(1))*steeringvec_plane(40,M,d,lamb)

vtheta=arange(mintheta,maxtheta,steptheta)
m=0
S=zeros(size(vtheta))+0j
for theta in vtheta:
    a=steeringvec_plane(theta,M,d,lamb);
    w= a/sqrt(vdot(a,a))
    S[m]=vdot(w,x)
    m=m+1

close('all')
figure(1)
plot(vtheta,abs(array(S)))
xlabel('$\\theta$');
title("DOA for current random realization")

"""
## DOA spectrum
We first compute the correlation matrix of the array, by averaging K
snapshots (i.e. realizations taken over the array sensors)
"""
```

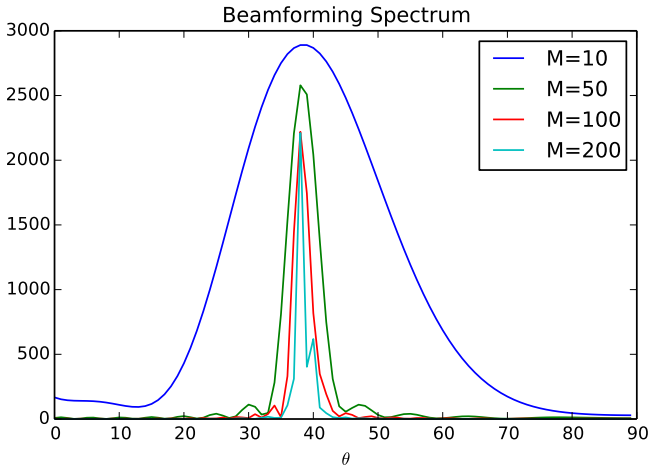
```
M=200
m=0
R=zeros((M,M))
for k in range(Nb_snap):
    x=2*randn(1)*steeringvec_plane(-20,M,d,lamb) + \
    2*randn(1)*steeringvec_plane(38,M,d,lamb) + \
    1*randn(1)*steeringvec_plane(40,M,d,lamb)
    R=R+outer(x,x.conj())

iR=inv(R+0.01*eye(M))

from numpy.linalg import matrix_rank
print("\t ==> The matrix R of size {1} has rank {0}".format(matrix_rank(R),repr(shape(


mintheta=-90; steptheta=1; maxtheta=90
vtheta=arange(mintheta,maxtheta,steptheta)

# Beamforming spectrum
for M in (10, 50, 100, 200):
    m=0;
    S=zeros(size(vtheta))
    C=zeros(size(vtheta))
    for theta in vtheta:
        a=col(steeringvec_plane(theta,M,d,lamb))
        w= a/sqrt(a.H*a)
        S[m]=1/M*abs(w.H*R[:M,:M]*w)    #beamforming spectrum
        # the extra factor M is introduced for normalization purposes
        m=m+1
    figure(2)
    plot(vtheta, abs(S),label="M={}".format(M))
figure(2)
xlabel('$\\theta$')
title("Beamforming Spectrum")
xlim([0, 90])
legend()
```

```
        ==> The matrix R of size (200, 200) has rank 3
```

Out [28]:

```
<matplotlib.legend.Legend at 0x7f322d510cd0>
```





We see that the two sources at 08 and 40°, are resolved for $M = 200$ but not for $M = 100$. Actually, we will have a correctct separation when $u(\theta_2) - u(\theta_1) > 1/M$, which implies $M > 1/(u(\theta_2) - u(\theta_1))$ which, numerically, gives $M > 148$.

```
theta2=40*pi/180; theta1=38/180*pi
1/(d/lamb*(sin(theta2)-sin(theta1)))
Out[24]: 147.45927107728497
```

6 Capon's method - Comparison of Beamforming and Capon spectra for M=20

The idea of Capon's method is to designa a spatial filter which miimizes the overall power in a direction θ, while imposing a unit gain in this direction. This results in a spatial filter

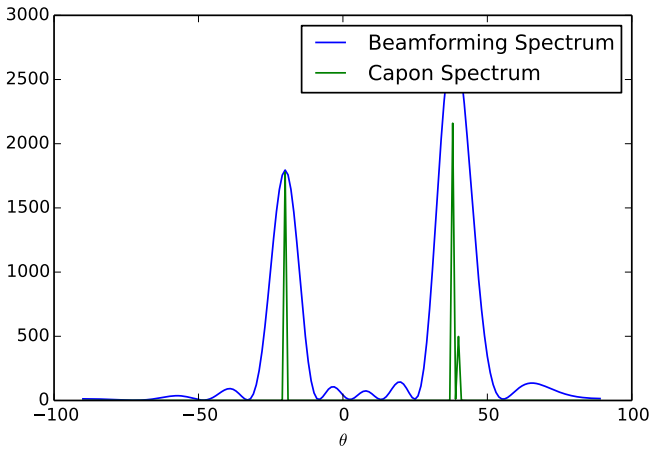
$$w = \frac{R^{-1}a}{a^{+}R^{-1}a} \tag{3}$$

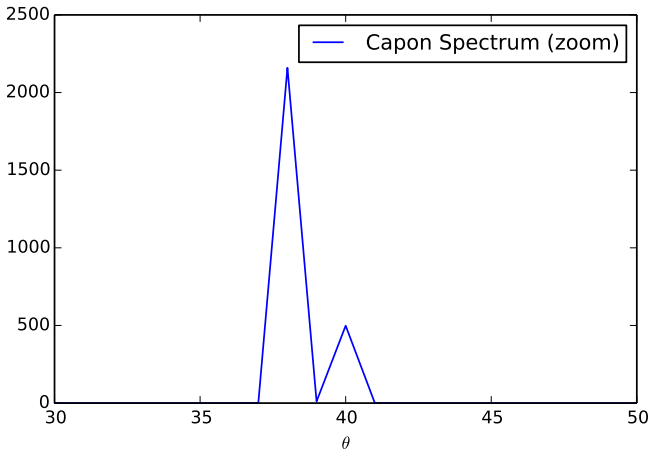
which minimizes the aliasing between sources; the filter depends on the direction scanned, but also on the overall environment, via the matrix R

In [29]:
```
m=0
M=20
iR=inv(R[:M,:M]+0.01*eye(M))
S=zeros(size(vtheta))
C=zeros(size(vtheta))
for theta in vtheta:
    a=col(steeringvec_plane(theta,M,d,lamb))
    w= a/sqrt(a.H*a)
    S[m]=1/M*abs(w.H*R[:M,:M]*w)    #beamforming spectrum
    # the extra factor M is introduced for normalization purposes
    w= iR*a/(a.H*iR*a)  # Capon vector
    C[m]=1/abs(a.H*iR*a)  #Capon spectrum
    m=m+1

figure(2)
plot(vtheta, abs(S),label="Beamforming Spectrum")
plot(vtheta, abs(C),label="Capon Spectrum")
xlabel('$\\theta$')
```

```
legend()

figure(3)
plot(vtheta, abs(C),label="Capon Spectrum (zoom)")
xlim([30, 50])
xlabel('$\\theta$')
legend()
```

Out [29]:
```
<matplotlib.legend.Legend at 0x7f322d1533d0>
```





We see that the results obtained with $M = 20$ with Capon's method are equivalent to the beamforming with $M = 200\ldots$ This shows that it is useful to study advanced concepts and invest in more evolved methods.

7 Spatial filtering

We consider a example of spatial filtering. We have a source of interest in the direction $\theta = 38\Sigma,$, but with with severe jammers in the directions $\theta = -20\Sigma,$ and $\theta = 40\Sigma,$. We show here, that the source of interst can be almost extracted by a spatial filter.

In [30]:
```
def rect_pulse(*arg):
    return (sign(cos(*arg)))

N=1000
```

```
x=2*outer(steeringvec_plane(-20,M,d,lamb),randn(N,1))+ \
    2*outer(steeringvec_plane(38,M,d,lamb),rect_pulse(2*pi*0.01*arange(N)))+ \
    outer(steeringvec_plane(40,M,d,lamb),randn(N,1))

a=col(steeringvec_plane(38,M,d,lamb))
w= iR*a/(a.H*iR*a)   # Capon vector
z=w.H*matrix(x)      # Filtering

# representations
figure(1); clf()
plot(real(x[0,:]))
title("Signal recorded on first sensor")
xlabel("Time")

figure(2); clf()
plot(real(array(z).flatten())) # need to do this cause there is a bug in matplotlib 1.
xlabel("Time")
title("Signal recovered after spatial filtering in the 38°, direction")
```

Out [30]:
```
<matplotlib.text.Text at 0x7f322cd7dd10>
```