UwbWaveforms

Unknown Author

December 8, 2013

In [1]:
```
from pylayers.signal.bsignal import *
from pylayers.simul.simulem import *
```

1 Generation of an Impulse of normalized energy

One possible manner to define an energy normalized short UWB impulse is as follows. This UWB waveform is generated with `bsignal.EnImpulse` function.

The default waveform is a gaussian windowing of a sine wave of frequency f_c. The normalization term depends on the exponential scaling factor τ.

$p(t) = \frac{\sqrt{2\sqrt{2}}}{\tau\sqrt{\pi}} \cos(2\pi f_c t) e^{-(\frac{t}{\tau})^2}$

$\tau = \frac{2}{B\pi}\sqrt{\frac{\gamma_{dB} \ln 10}{20}}$

where B is the desired bandwidth defined at γ_{dB} below the spectrum maximum and f_c is the central frequency of the pulse.

In [2]:
```
fc     = 4
band   = 2
thresh = 10
fe     = 100
ip     = EnImpulse([],fc,band,thresh,fe)
```

In [3]:
```
ip.info()
```

1.1 Verification of energy normalization in both domains

In [4]:
```
E1= sum(ip.y*ip.y)*ip.dx()
print "Integration in time",E1
```

In [5]:
```
P = ip.esd()
E2 = sum(P.y)*P.dx()
print "Integration in frequency domain ",E2
```

1.2 Calculation of UWB channel impulse response

In [6]:
```
S = Simul()
S.load('where2.ini')
```

In [7]:
```
st = S.wav.st
sf = S.wav.sf
S.wav.info()
```

In [8]:
```
S.wav
```

In [9]:
```
S.wav.show()
```

Here the time domain waveform is measured and the anticausal part of the signal is artificially set to 0.

To handle properly the time domain wavefom it is required to center the signal in the middle of the array.

- `st` member stands for signal in time domain
- `sf` member stands for signal in frequency domain

The measured waveform has a small offset of 0.7 ns, the following waveform compensate for this bias and is centered on local time origin.

In [10]:
```
S.wav['typ']='W1compensate'
S.wav.eval()
```

The frequency domain version of the signal is embedded in the same object.

`sf` stands for signal in frequency domain.

In [11]:
```
type(S.wav.sf)
```

2 Construction of the propagation channel

The link between Txid = 1 and Rxid =1 is simply loaded as

In [12]:
```
vc = S.VC(1,1)
```

The following representation shows the spatial spreading of the propagation channel. On the left are scattered the intensity of rays wrt to angles of departure (in azimut and elevation). On the right is the intensity of rays wrt to angles of arrival. It misses the application between the 2 planes as well as the delay dimension of the propagation channel.

In [13]:
```
vc.doadod()
```

2.1 Construction of the transmission channel

The transmission channel is obtain from the combination of the propagation channel and the vector antenna pattern at bot side of the radio link

```
In [14]: sc = vc.prop2tran()
```

The ScalChannel object contains all the information about the ray transfer functions. The transmission channel is obtained by applying a vector radiation pattern using an antenna file.

In the presented case, it comes from a real antenna which has been used during the **FP7 project WHERE1** measurement campaign M1.

```
In [15]: sc
```

The antenna radiation pattern is stored in a very compact way thanks to Vector Spherical Harmonics decomposition. The following gives information about the content of the antenna object.

```
In [16]: S.tx.A.info()
```

```
In [17]: fig,ax = sc.plot()
```

The figure below plot on a same graph all the tansfer function in modulus and phase of the ray transfer function.

```
In [ ]: fig,ax = sc.plot(iy=arange(20))
```

If a realistic antenna is applied it gives

```
In [ ]: alpha = 1./sqrt(30)  # scaling constant depends on how are stored the antenna data
        sca = vc.prop2tran(S.tx.A,S.rx.A)
        sca.plot()
```

```
In [ ]: fig,ax = sca.plot(iy=arange(20))
```

2.2 Calculate UWB Channel Impulse Response

```
In [ ]: cir = sc.applywavB(S.wav.sfg)
```

```
In [ ]: figsize(15,15)
        cir.plot()
        xlabel('delay (ns)')
        ylabel('Amplitude (V)')
        title('Received Waveform')
```

3 Hermitian symetry enforcment

If the number of point for the transmission channel and the waveform were the same the mathematical operation is an Hadamrd-Shur product between $\mathbf{Y}$ and $\mathbf{W}$.

$\mathbf{Y} = \mathbf{S} \odot \mathbf{W}$

In practice this is what is done after a resampling of the time base with the greater time step.The process which consists in going from time domain to frequency domain is delegated to a specialized class `Bsignal` which maintain the proper binding between signal samples and their indexation either in time or in frequency domain.

In [ ]:
```
wgam = S.wav.sfg
Y     = sc.apply(wgam)
tau   = Y.tau0
dod = Y.dod
doa = Y.doa
```

The transmission channel has a member data which is the time delay of each path

In [ ]:
```
print 'tau =', tau
```

In [ ]:
```
print "doa = ", doa
```

symHz force the Hermitian symetry of Y with as an argument here a zero padding of 500 points

In [ ]:
```
UH    = Y.symHz(500)
UH.plot(dB=False)
```

In [ ]:
```
uh    = UH.ifft(1)
uh.plot()
```

In [ ]:
```
ips   = Y.ift(500,1)
t     = ips.x
ip0   = TUsignal(t,ips.y[0,:])
```

In [ ]:
```
plot(UH.x,real(UH.y[0,:]),UH.x,imag(UH.y[0,:]))
U0 = FHsignal(UH.x,UH.y[0,:])
u0 = U0.ifft(1)
u1 = ifft(U0.y)
plt.figure()
```

In [ ]:
```
plot(uh.x,uh.y[0,:]*1000+3)
```

In [21]:
```
from IPython.core.display import HTML

def css_styling():
    styles = open("../styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

In [ ]:
```

```