SINR Functional Beamforming

June 1, 2018

```
In [1]: import numpy as np
        from matplotlib import pyplot as plt
        from matplotlib import gridspec
        from matplotlib import animation
        from scipy.stats import norm
        from cmath import sqrt
        from time import time
         = np.pi
        num_points = 500
        tol = 1e-14

In [2]: %matplotlib inline
        plt.rcParams["figure.dpi"] = 300

In [3]: def f(func,C) :
            '''
            If C = UU.T, then f(C) = Uf()U.T = U*diag(func(_1),...,func(_2))*U.T

            Inputs :
                func - Callable scalar function
                C    - Cross Spectral Matrix of a system

            Output :
                f(C) as defined in description
            '''
            eigv, U = np.linalg.eig(C)
            eigv = [0 if val < tol else val for val in eigv]
            f_eigs = [func(eig) for eig in eigv]
            return U@np.diag(f_eigs)@U.T

        def weights(,N,r,) :
            '''
            Calculates the coefficients of the beam equation

            Inputs:
                 (rad) - angle off from orthogonal to array
                N (int) - number of antenna +1
```

```
    Output:
        a_k for the beam form equation _1^N <a_k(), x_k>; x_k is the volatge response
        NOTE: This vector is normalized (why? I don't know)
    '''
     = *np.sin()
    bs = np.zeros(N)
    for k in range(N) :
        bs[k] = np.exp(*k*1j)
    return bs/np.linalg.norm(bs)

def sinr_ber_plp(signal_beam,noise_beams,var_awgn=1,var_ray=1,h_sig=None,pack_len=10,di
    '''
    Calculates the estimated SINR, BER, and PLP (packet loss percentage)
        for certain locations based on transmission beams.
    Assumes a Rayleigh channel with BPSK modulation and CRC.

    Input :
        Beams are of the form [power_levels (n,) array, weights (L,) array]
        signal_beam  - beam  - The signal beam
        noise_beams  - list  - List of noise beams (each beam like signal beam)
        var_awgn     - float - White noise variance
        var_ray      - float - Fading variance of the Rayleigh channel
        h_sig        - array - Fading from the Rayleigh channel [CN(0,var_ray) distrib
        pack_len     - int   - Number of bits per packet (including the checking bit)
        dist_fad_exp - float - Exponent for the (optional) distance attenuation fading
        verbose      - bool  - Whether or not you want an update every 100 iterations

    Output :
        SINR -  (n,n) array  - Signal to Interference and Noise Ratio
        BER  -  (n,n) array  - Bit Error Rate (for BPSK under https://www.unilim.fr/pa
        PLP  -  (n,n) array  - Packet Loss Percentage
    '''
    sig_beam = signal_beam[0]
    sig_weights = signal_beam[1]
    n_beams = [beam[0] for beam in noise_beams]
    n_weights = [beam[1] for beam in noise_beams]
    K = len(n_weights)
    n = len(sig_beam)
    L = sig_weights.shape[0]
    r = np.linspace(tol,max(sig_beam),n)
    SINR = np.zeros((n,n))
    if h_sig is None :
        h_sig = np.random.normal(loc=np.array([0,0]),scale=np.array([var_ray,var_ray])
        h_sig = 1/sqrt(2)*(h_sig[:,0] + 1j*h_sig[:,1])
    sig_const = abs(h_sig @ sig_weights)**2
    noise_const = [abs(h_sig[i] @ weight[i])**2 for i in range(K)]
    for j in range(n) : # Iterating over
```

```
        sig_pow = sig_beam[j]
        noise_pows = [beam[j] for beam in n_beams]
        if dist_fad_exp :
            SINR[j,:] = [((sig_pow/(r[i]**dist_fad_exp))*sig_const)/(var_awgn + sum([(r
        else :
            SINR[j,:] = [(sig_pow*sig_const)/(var_awgn + sum([noise_pows[k]*noise_const
        if verbose and (j+1) % 100 == 0 :
            print(f'{j+1}th iteration complete.')
    mask = SINR >= max(sig_beam)
    SINR[mask] = max(sig_beam)
    BER = 1 - norm.cdf(np.sqrt(2*SINR*abs(h_sig @ h_sig)))
    PLP = 1 - (1 - BER)**pack_len
    SINR = 10*np.log(SINR)/np.log(10)
    return SINR, BER, PLP
```

0.0.1 For a single beam

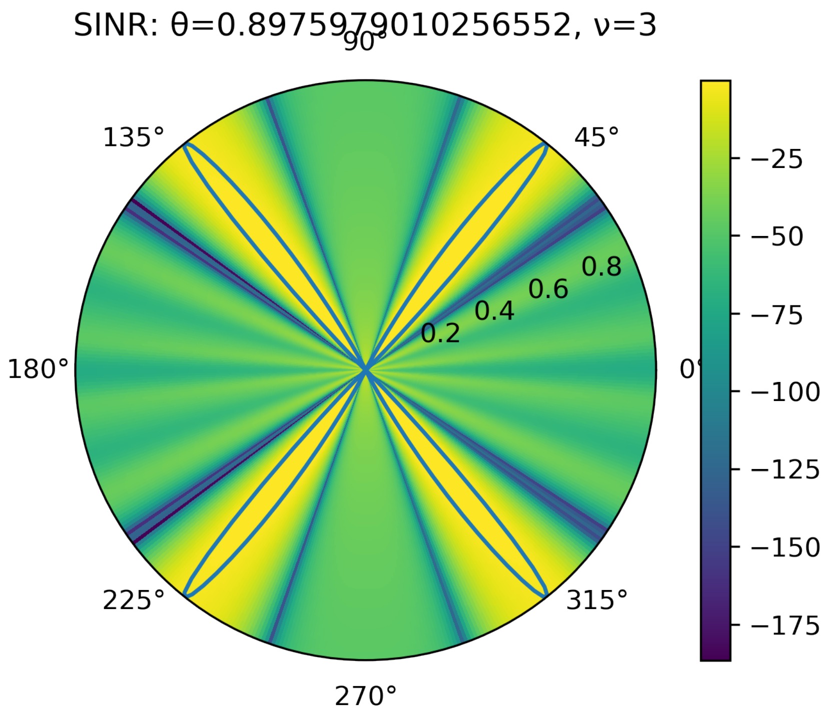
```
In [4]: N = 7                                                  # Number of antenna
         = 1                                                  # Wavelength
         = 3                                                  # Functional Beamforming Exponent
        2_awgn = 1                                            # Noise/AWGN variance
        2_ray = 1                                             # Noise/Fading variance
        beams = 2*/7                                          # Desired azmith angle
        s_k = 10                                               # Alice's signal strength
        packet_len = 9
        s = np.linspace(0,2*,num_points)
        ak = weights(beams,N,s_k,)
        g_k = np.array(ak).reshape(1,N)                        # Direction of Alice's transmitt
        C = (np.dot(g_k.T,g_k))                                # Cross Spectral Matrix
        func = lambda x : x**(1/)
        b = lambda g : (g.T@f(func,C)@g)**
        sig_beam = [b(weights(,N,s_k,)) for  in s]
        signal_beam = [sig_beam,ak]
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:32: ComplexWarning: Casting compl
```

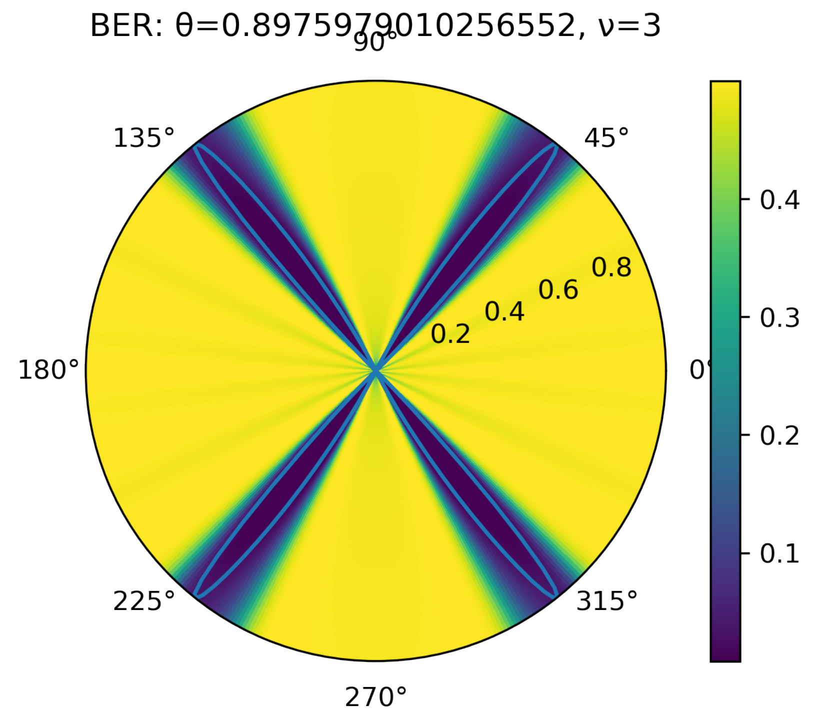
```
In [5]: start = time()
        SINR, BER, PLP = sinr_ber_plp(signal_beam,[],var_awgn=2_awgn,var_ray=2_ray,pack_len=pac
        end = time()
        print(f'Took {(end-start)//3600} hour(s), {((end-start)%3600)//60} minute(s) and {(((en
```

```
100th iteration complete.
200th iteration complete.
300th iteration complete.
400th iteration complete.
500th iteration complete.
Took 0.0 hour(s), 0.0 minute(s) and 0.35666584968566895 seconds.
```

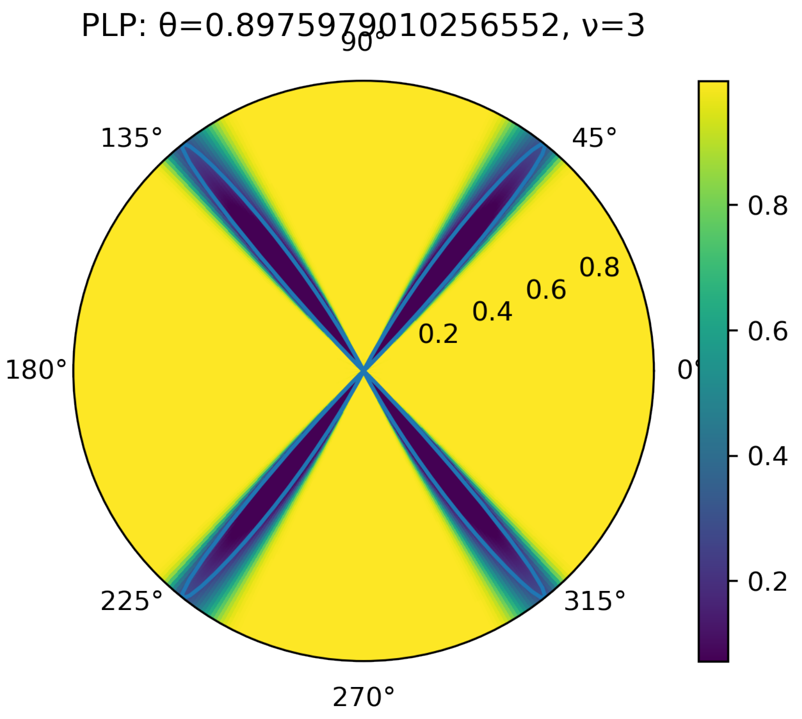
```
In [6]:  = np.linspace(0,2*,num_points)
        r = np.linspace(0,max(sig_beam),num_points)
        R, = np.meshgrid(r,)
        plt.polar(,sig_beam)
        plt.pcolor(,R,SINR)
        plt.colorbar()
        plt.title(f'SINR: ={beams}, ={}')
        plt.show()
```



```
In [7]: plt.polar(,sig_beam)
        plt.pcolor(,R,BER)
        plt.colorbar()
        plt.title(f'BER: ={beams}, ={}')
        plt.show()
```



```
In [8]: plt.polar(,sig_beam)
        plt.pcolor(,R,PLP)
        plt.colorbar()
        plt.title(f'PLP: ={beams}, ={}')
        plt.show()
```



0.0.2 Moving the beam

```
In [9]: N = 7                                                   # Number of antenna
         = 1                                                   # Wavelength
         = 3                                                   # Functional Beamforming Exponent
        2_awgn = 1                                             # Noise/AWGN variance
        2_ray = 1                                              # Noise/Fading variance
        beam_cent = 2*/7                                       # Azmith angle to Bob
        packet_len = 9
        num_packets = 51
        beams = np.linspace(beam_cent-/12,beam_cent+/12,num_packets)
        s_k = 10                                                # Alice's signal strength
        func = lambda x : x**(1/)
        s = np.linspace(0,2*,num_points)
        all_beams = []
        SINRs = np.zeros((num_points,num_points,num_packets))
        BERs = np.zeros((num_points,num_points,num_packets))
        PLPs = np.zeros((num_points,num_points,num_packets))
        hsig = np.random.normal(loc=np.array([0,0]),scale=np.array([2_ray,2_ray]),size=(N,2))
        hsig = 1/sqrt(2)*(hsig[:,0] + 1j*hsig[:,1])
        start = time()
        for i in range(num_packets) :
            ak = weights(beams[i],N,s_k,)
            g_k = np.array(ak).reshape(1,N)                     # Direction of Alice's transmitt
            C = (np.dot(g_k.T,g_k))                                # Cross Spectral Matrix
            b = lambda g : (g.T@f(func,C)@g)**
            sig_beam = [b(weights(,N,s_k,)) for  in s]
            all_beams.append(sig_beam)
            signal_beam = [sig_beam,ak]
            SINRs[:,:,i], BERs[:,:,i], PLPs[:,:,i] = sinr_ber_plp(signal_beam,[],var_awgn=2_awg
        end = time()
        all_beams = np.array(all_beams)
        print(f'Took {(end-start)//3600} hour(s), {((end-start)%3600)//60} minute(s) and {(((en

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:32: ComplexWarning: Casting comple
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:75: ComplexWarning: Casting comple
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:81: ComplexWarning: Casting comple


Took 0.0 hour(s), 0.0 minute(s) and 24.498143434524536 seconds.
```

```
In [10]: np.save('../../../IRES_Files/Beams_fun',all_beams)
         np.save('../../../IRES_Files/SINRs_fun',SINRs)
         np.save('../../../IRES_Files/BERs_fun',BERs)
         np.save('../../../IRES_Files/PLPs_fun',PLPs)
```

```
In [11]: %matplotlib notebook
         %matplotlib notebook
```

```
In [12]: """
         https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/

         Matplotlib Animation Example

         author: Jake Vanderplas
         email: vanderplas@astro.washington.edu
         website: http://jakevdp.github.com
         license: BSD
         """

         # Initialize values
         fig = plt.figure()
         ax = plt.axes(xlim=(0,2*),ylim=(0,1))
         beam, = ax.plot([],[],lw=2)

         # Define constructor fucntion
         def constructor() :
             beam.set_data([],[])
             return beam,

         # Define animation function
         def animating(i) :
              = np.linspace(0,2*,num_points)
             r = all_beams[i,:]
             beam.set_data(,r)
             return beam,

         # call the animator.  blit=True means only re-draw the parts that have changed.
         anim = animation.FuncAnimation(fig, animating, init_func=constructor,frames=num_packe
         # Something's wrong with the save...
         #anim.save('../../../IRES_Files/Animations/Fun_beam.html', fps=30, extra_args=['-vcod
         #plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
In [13]: %matplotlib notebook
         # Initialize values
         fig = plt.figure(figsize=(14,14), facecolor='white')
         gs = gridspec.GridSpec(11,1)

         ax1 = plt.subplot(gs[:2,:])
         ax1.set_xlim(0,2*)
         ax1.set_ylim(0,1)
         ax1.set_xlabel('')
         ax1.set_title(f'Beam from {beam_cent-/12} to {beam_cent+/12}')
         beam, = ax1.plot([],[],lw=2)

         R,  = np.meshgrid(np.linspace(0,1,num_points),np.linspace(0,2*,num_points))
         SNR = SINRs[:,:,0]
         ax2 = plt.subplot(gs[3:5,:])
         ax2.set_xlim(0,2*)
         ax2.set_ylim(0,1)
         ax2.set_xlabel('')
         ax2.set_title(f'SINR')
         snr = ax2.pcolormesh(,R,SNR,shading='gouraud')

         BER = BERs[:,:,0]
         ax3 = plt.subplot(gs[6:8,:])
         ax3.set_xlim(0,2*)
         ax3.set_ylim(0,1)
         ax3.set_xlabel('')
         ax3.set_title(f'BER')
         ber = ax3.pcolormesh(,R,BER,shading='gouraud')

         PLP = PLPs[:,:,0]
         ax4 = plt.subplot(gs[9:11])
```

```
ax4.set_xlim(0,2*)
ax4.set_ylim(0,1)
ax4.set_xlabel('')
ax4.set_title(f'PLP')
plp = ax4.pcolormesh(,R,PLP,shading='gouraud')

# Define constructor fucntion
def constructor() :
    beam.set_data([],[])
    snr.set_array(np.array([]))
    ber.set_array(np.array([]))
    plp.set_array(np.array([]))
    return beam,snr,ber,plp

# Define animation function
def animating(i) :
     = np.linspace(0,2*,num_points)
    r = all_beams[i,:]
    beam.set_data(,r)
    snr.set_array(SINRs[:,:,i].ravel())
    ber.set_array(BERs[:,:,i].ravel())
    plp.set_array(PLPs[:,:,i].ravel())
    return beam,snr,ber

# call the animator.  blit=True means only re-draw the parts that have changed.
anim = animation.FuncAnimation(fig, animating, init_func=constructor,frames=num_packe
# Something's wrong with the save...
#anim.save('Fun_Real_Full.html', writer='ffmpeg', fps=30, extra_args=['-vcodec', 'lib
#plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```