FSK Error Probability Helper

April 29, 2017

Contents


```
In [1]: %pylab inline
        #%matplotlib qt
        from __future__ import division # use so 1/2 = 0.5, etc.
        import ssd
        import scipy.signal as signal
        from IPython.display import Audio, display
        from IPython.display import Image, SVG
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: pylab.rcParams['savefig.dpi'] = 100 # default 72
        #pylab.rcParams['figure.figsize'] = (6.0, 4.0) # default (6,4)
        #%config InlineBackend.figure_formats=['png'] # default for inline viewing
        %config InlineBackend.figure_formats=['svg'] # SVG inline viewing
        #%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

```
In [5]: import digitalcom as dc
        import lab6
        #reload(lab6)
```

Lab 6 FSK Error Probability

This short notebook provides a checkout of the bit error probability tools found in `lab6.py`. The function `fsk_BEP()` in particular has been update to fix an error in calling the function `dc.BPSK_BEP()` with both the rc and tx data.

```
def fsk_BEP(rx_data,m,flip):
    """
    fsk_BEP(rx_data,m,flip)

    Estimate the BEP of the data bits recovered
    by the RTL-SDR Based FSK Receiver.

    The reference m-sequence generated in Python
    was found to produce sequences running in the opposite
    direction relative to the m-sequences generated by the
    mbed. To allow error detection the reference m-sequence
    is flipped.

    Mark Wickert April 2014
    """
    Nbits = len(rx_data)
    c = dc.m_seq(m)
    if flip == 1:
        # Flip the sequence to compenstate for mbed code difference
        # First make it a 1xN array
        c.shape = (1,len(c))
        c = np.fliplr(c).flatten()
    L = int(np.ceil(Nbits/float(len(c))))
    tx_data = np.dot(c.reshape(len(c),1),np.ones((1,L)))
    tx_data = tx_data.T.reshape((1,len(c)*L)).flatten()
    tx_data = tx_data[:Nbits]
    # Convert to +1/-1 bits
    tx_data = 2*tx_data - 1
    # Note the next line contains a change from the
    # original file as both rx_data and tx_data are input:
    Bit_count,Bit_errors = dc.BPSK_BEP(rx_data,tx_data)
    print('len rx_data = %d, len tx_data = %d' % (len(rx_data),len(tx_data)))
    Pe = Bit_errors/float(Bit_count)
    print('/////////////////////////////////////')
    print('Bit Errors: %d' % Bit_errors)
    print('Bits Total: %d' % Bit_count)
    print('       BEP: %2.2e' % Pe)
    print('/////////////////////////////////////')
```

- Add noise to a ± 1 binary bit sequence, then make hard decisions using the `sign()` function
- Use the hard decision data bits as simulated FSK receive data

```
In [46]: rx_data = 2*ssd.PN_gen(10000,5) - 1
         rx_data = dc.cpx_AWGN(rx_data,4,1).real
         rx_data = sign(rx_data)
```

```
In [47]: lab6.fsk_BEP(rx_data,5,0)
```

```
kmax =  0, taumax = 0
len rx_data = 10000, len tx_data = 10000
/////////////////////////////////////
Bit Errors: 109
Bits Total: 10000
       BEP: 1.09e-02
/////////////////////////////////////
```

- Are the errors counted resaonable?
- The `dc.cpx_AWGN()` function adds Gaussian noise to the input ± 1 data bits, such that the probbility of a bit error is related to the Gaussian PDF tail probability by the complementary error function as:

$$P_e = \frac{1}{2}\text{erfc}\left(\sqrt{\frac{10^{\text{EbN0}_{\text{dB}}/10}}{2}}\right) \tag{1}$$

- Now compare the above experimental result to theory by setting `EbN0 = 4`

```
In [48]: EbN0 = 4
         print('BEP thy = %1.3e' % (dc.erfc(sqrt(10**(EbN0/10)))/2))
```

```
BEP thy = 1.250e-02
```

- The results are reasonable given only ~100 error events are counted

```
In [ ]:
```