Set #3

Due Wednesday March 18, 2020

Problems:

1. Using the complex-baseband PLL for demodulation of the legacy SATCOM standard PCM-PM-NRZ (pulse-code modulation using phase modulation with non-return-to-zero line coding) (http://www.eas.uccs.edu/~mwickert/ece5675/lecture_notes/PCM-PM.pdf). This is a simulation-based problem that first introduces the complex-baseband PLL found in sk_dsp_comm.synchronization. Using this approach we can model a PLL at the full waveform level, not just phase in/phase out, as is done in ‘PLL1’. A sinusoidal phase detector is employed which forms as output

\[ e_d(t) = \text{Im} \left[ x_r(t) \cdot e_o^*(t) \right] \]  

where the now complex signals \( x_r(t) \) and \( e_o(t) \) are of the form

\[ x_r(t) = A_c e^{j[2\pi f_c t + \theta(t)]}, \text{ where } f_c \approx 0 \]  
\[ e_o(t) = e^{j\hat{\theta}(t)} \]

In advanced applications \( x_r(t) \) may also include a noise signal. In what follows \( A_c = 1 \) to insure that the internal gain scaling for the loop parameters remains calibrated.

As an example consider as input a phase step \( \pi/4 \) radians and a frequency step 20 Hz, which introduces cycle slips, but ultimately locks. A Jupyter notebook, PCM_PM_NRZ_sample.ipynb is provided to speed the process of conducting simulation experiments.
From the above plot it should be clear that the complex baseband PLL is easy to work with. Also note that phase error in this case is actually the sine of the phase error, hence the phase wrapping is through this function.

Now we briefly describe three helper functions provided in the notebook PCM_PM_NRZ_sample.ipynb. The first is `PCM_PM_tx()`, that generates a complex baseband PCM-PM waveform using a
non-return to zero (NRZ) pulse shape or a bi-phase/Manchester pulse shape. In mathematical terms the transmitted complex baseband PM signal takes the form

\[
x_{PM}(t) = e^{jK_p x_m(t)} = \exp \left[ jK_p \sum_{n=-\infty}^{\infty} d_n p(t - nT_b) \right]
\] (4)

where \(d_n = \pm 1\) are random data bits, \(T_b = 1/R_b\) is the bit period, \(p(t)\) is the NRZ pulse shape, and \(K_p\) is the phase modulator gain constant. All of this is depicted in the figure below:

The function doc string is listed below:

```python
def PCM_PM_tx(N_bits, Ns, Kp, mode='NRZ'):
    """
    Complex baseband PCM-PM modulator for NRZ and Manchester pulse shaping.
    
    Parameters
    ----------
    N_bits : Number of random bits to produce
    Ns : Number of waveform samples per bit
    Kp : The peak phase deviation per bit
    mode : 'NRZ' or 'Manchester' pulse shaping
    
    Returns
    -------
    x_phase : phase modulated complex sinusoid
    """
```

Page 3 of 16
It may not be obvious, but at complex baseband with the carrier at zero, the spectrum of $x_{PM}(t)$ will contain a spectral line at dc. This is significant as having a residual carrier in a PCM-PM means that a PLL can be used to recover a coherent carrier from the signal and hence recover $x_m(t)$ from $x_{PM}(t)$. To see this consider the PCM-PM phase constellation. Since we constrain $0 < K_p < \pi/2$ and $x_m(t)$ takes on values of $\pm 1$, we can write

$$x_{PM}(t) = \cos(K_p) + j \sin(K_p) \sum_{n=-\infty}^{\infty} d_n p(t - nT_b)$$

For this problem we are only concerned with the NRZ pulse shape, but feel free to try out the Manchester pulse shape on your own. A typical PCM-PM-NRZ test vector generation example is shown below.

```python
fs = 20e6
Ns = 20
# The rate is 1 Mbps
R_b = fs/Ns
N_bits = 100000 # 100000
Kp = pi/4
x_PM, x_mod, b, data = PCM_PM_tx(N_bits,Ns,Kp,'NRZ')
# x_PM, x_mod, b, data = PCM_PM_tx(N_bits,Ns,Kp,'Manchester')
t = arange(len(x_PM),dtype=float64)/fs
Px,f = ss.my_psd(x_PM,2**11,fs/1e6);
plot(f,10*log10(Px))
title(r'PCM-PM Power Complex Baseband Power Spectrum')
ylabel(r'PSD (dB)')
xlabel(r'Frequency (MHz)')
ylim([-60,20])
xlim([-4,4])
grid();
```
PCM-PM-NRZ example, including the transmit spectrum, all at $f_c = 0$ Hz.

The second function, `simple_bit_sync()`, is used behind the scenes to estimate the sampling instant of the demodulated PCM-PM waveform corresponding to the maximum eye opening. When $N_s$ waveform samples are used to represent each bit the optimum sampling instant is needed in the receiver detector, i.e., $N_{opt} \in [0, N_s - 1]$. The function doc string is listed below:

```python
def simple_bit_sync(x, Ns):
    
    Find the maximum eye opening across a bit stream by assuming
    synchronous sampling at $N_s$ samples per bit.

    Parameters
    ----------
    x : Bit stream waveform with $N_s$ samples per bit
    Ns : The number of sample per bit, assumed to be synchronous

    Returns
    -------
    idx_max : The index on the interval $[0, N_s - 1]$ giving the maximum
              eye opening

    Mark Wickert February 2020
```

The third function, `bit_detect_BEP`, takes the demodulated phase modulated waveform, possibly matched filtered, and ... The function doc string is listed below:
```python
def bit_detect_BEP(z_real, data_tx, Ns, N_transient = 1):
    """
    Bit synchronize, bit detect, and compare with the transmit bits

    Parameters
    ----------
    z_real : a real bit stream waveform at Ns samples per bit; for
    a coherent demod this may be the real part after
    appropriate phase rotation
    data_tx : transmitted data bits used in probability of bit
    error (BEP) estimation
    Ns : number of samples per bit
    N_transient : Number of initial bits to discard due to waveform
    transient, which could be due to PLL acquisition
    settling

    Returns
    -------
    N_total : number of bits processed
    N_error : number of bits in error
    opt_samples : max eye opening samples from the detection process

    Mark Wickert March 2020
    """

Note the bit_detect_BEP() function does call the simple_bit_sync() function. An example of bit_detect_BEP() in action is provided below. The scenario is not NRZ-PM, but rather a simple NRZ waveform received with additive white Gaussian noise (AWGN).

Ns = 20
EbN0_dB = 20
x_NRZ, b_NRZ, data_NRZ = dc.NRZ_bits(100,Ns)
y_NRZ = dc.cpx_AWGN(x_NRZ, EbN0_dB, Ns)  # noise is complex
# Apply a rect pulse matched filter, which here is in b_NRZ
z_NRZ = signal.lfilter(b_NRZ,1,y_NRZ)
N_total, N_errors, opt_samples = bit_detect_BEP(z_NRZ.real, data_NRZ, Ns)
print('Bit Count = {}, Bit Errors = {}, BEP_hat {:1.2e}'.format(N_total, N_errors, N_errors/N_total))
```

Estimated mod(Ns) bit synch index is 19 of 20
kmax = 0, taumax = -1
Bit Count = 98, Bit Errors = 0, BEP_hat 0.00e+00

Creating a noisy NRZ waveform with \( N_s = 20 \) samples per bit
Consider an eye plot of the matched filtered waveform and note max eye opening is 19 mod 20

With the helper function descriptions complete, now its time to move into some investigations. Two broad categories are (1) phase demodulation using a 1st-order PLL to directly
recover $x_m[n]$ and its data bits and (2) use a Type 2 PLL to track the residual carrier component of $x_{PM}[n]$ and coherently demodulate $x_m[n]$ to recover the data bits. Note we switch now to discrete-time notation, by assuming that $x_m(t) \rightarrow x_m[n]$ when $t$ is replaced by $n/f_s$.

The channel that the PCM-PM signals passes through consists of frequency error, $\Delta f$ Hz, static phase, $\varphi \in U(0, 2\pi)$, and additive white noise, $w[n]$ having PSD $N_0/2$:

Channel model

(a) Consider a first-order PLL (Type 1) for the recovery of $x_m[n]$ and bit detection to finally obtain the bits $d_n$. In this brief study set $\varphi = 0$. The system block diagram takes the form shown below:

1 Mbps PCM-PM-NRZ

System block diagram with 1st-order PLL

i. Start with $\Delta f = 0.1$ Hz and $f_n$ somewhere in the range $[R_b/4, R_b]$. Follow along in the notebook sample the code cells starting at 1a(i)–1a(iv). Use 100k bits for the test vector and set $E_b/N_0 = 100$ dB. Note the filtering action of the 1st-order PLL in the waveform and eye plots. What is the effective filter frequency response? Verify that there are no bit errors. Note the bit count should be 99998 or so.

ii. Repeat (i) with $\Delta f = 100$ Hz. In particular observe the plot of the Eye Opening Samples, which spans 100 ms. What is happening in mathematical terms. Write a simple expression for the phase $\hat{\theta}[n]$ versus sample index $n$. What does this say about the ability of using a PLL as a phase demodulator when frequency error is present?
iii. Repeat (i) \((\Delta f = 0.1 \text{ Hz})\) and now set \(E_b/N_0 = 13 \text{ dB}\). Observe the BEP estimate. Try tweaking \(f_n\) to see if you can make better or make worse the BEP. By again observing the plot of the \textit{Eye Opening Samples} make sure the samples nominally swing about \(\pm 1\).

iv. Repeat (iii) and now set \(E_b/N_0 = 11 \text{ dB}\). Observe the BEP estimate. From the plot of the \textit{Eye Opening Samples} that the samples will likely be jumping around very large values similar to the figure below. Note with the plot scaling provided the waveform should, without \(\Delta f\) stress, swing between \(\pm 1\) nominally. If you don’t observe this try moving \(f_n\) closer to \(R_b\). This is the PLL slipping cycles. Is this behavior acceptable in your opinion?

![PLL Demodulator Eye Opening Samples](image)

Eye opening samples over 100 ms at low \(E_b/N_0\) and \(\Delta f = 0.1 \text{ Hz}\)

(b) Type 2 PLL locking onto the residual carrier of PCM-PM-NRZ and coherent demodulation to recover the bits, including probability of bit error (BEP) estimation. The system block diagram takes the form shown below:

![System block diagram with 2nd-order Type 2 PLL](image)

i. Complete the code cell under 1b(i) to implement the Type 2 PLL with \(f_n = 10 \text{ Hz}\). Observe from the plot generated in the cell below that the residual carrier phase \textit{randomly walks} over less than 0.1 degrees in 100 ms.
ii. Fill in the details in the code cell under 1b(ii) to implement coherent demodulation including a proper phase shift and the NRZ matched filter. Explain in mathematical terms why coherent demod needs to rotate the signal constellation by $\frac{\pi}{2}$ to make the signal points project the large amplitude swings along the real axis. Observe the signal constellation, the demodulated waveform (real part) and the eye plot. Compare and contrast with the 1st-order PLL equivalents (if they existed). Finally observe the bit errors and the eye opening samples over 100 ms; again compare and contrast with the 1st-order PLL equivalents.

iii. Repeat (ii) except now set $\Delta f = 10$ Hz. This time in your observations focus on the phase constellation plot and the eye opening samples over 100 ms.

iv. Repeat (iii) except now set $f_n = 50$ Hz. Focus exclusively on the phase constellation plot and the eye opening samples over 100 ms. What is significantly different compared with (iii)?

v. Finally consider BEP testing the coherent design and comparing with ideal theory (no phase jitter on recovered coherent carrier). The Jupyter notebook section entitled "Theoretical PCM-PM BEP" provides a function for plotting theoretical BEP curves for different values of $K_p$. With $K_p = \frac{\pi}{2}$ we have ordinary BPSK and the best performance (no residual carrier so the coherent demod carrier has to be obtained in a different fashion). The task here is to overlay some experimental BEP point on the theoretical curve in Section 1b(iv) of the Jupyter notebook sample. Overlay estimated BEP points at $E_b/N_0 = 8$ and 10 dB for both $f_10 = 10$ Hz and 500 Hz. For which $f_n$ value do you expect the BEP points to be closest to ideal theory? Why?

2. In this problem you will compare the transient response of the Type II APLL with the corresponding Type II DPLL. From notes Chapter 3, the closed-loop system functions are given by

$$H(s) = \frac{K}{\tau_1 (\tau_2 s + 1)} \left( \frac{s^2 + K \frac{\tau_2}{\tau_1} s + K}{s^2 + \frac{2\xi \omega_n}{\sqrt{2}} s + \omega_n^2} \right)$$

$$H(z) = \frac{k_1 z^{-1} + (k_2 - k_1) z^{-2}}{1 - (2 - k_1) z^{-1} + (1 - k_1 + k_2) z^{-2}}$$

(6)

(7)

For the continuous-time system

$$\omega_n = \sqrt{\frac{K}{\tau_1}}, \quad \xi = \frac{1}{2\omega_n} K \frac{\tau_2}{\tau_1} = \frac{\tau_2}{2} \sqrt{\frac{K}{\tau_1}}, \quad B_n = \frac{\omega_n}{2} \left( \xi + \frac{1}{4\xi} \right).$$

For the discrete-time system, with noise bandwidth $B_n$ in Hz much less than the sampling
rate $f_s$, $\omega_n$ and $\zeta$ for the continuous and discrete-time systems are related by

$$k_1 = k_c k_d \alpha_1 = \frac{2\zeta \omega_n}{f_s} + \frac{1}{2} \left(\frac{\omega_n}{f_s}\right)^2$$

(8)

$$k_2 = k_c k_d \alpha_2 = \left(\frac{\omega_n}{f_s}\right)^2.$$  

(9)

(a) Choose $B_n = 10$ Hz and $\zeta = 1/\sqrt{2}$ as the point of comparison with $f_s = 1000$ Hz. Plot comparison curves for both the phase step response and the frequency step response. Assume the linear model is valid.

(b) Repeat part (a) with $f_s = 100$ Hz.

(c) Repeat part (a) with $f_s = 10000$ Hz.

3. Construct in Python a baseband DPLL simulation framework making use of the classes DDS and accum described below for both the phase controller and the loop filter, and that uses a selectable linear or sin() phase detector, i.e.,

$$\phi[n] = \sin \left(\theta[n] - \hat{\theta}[n]\right)$$

or

$$\phi_n \approx \theta[n] - \hat{\theta}[n]$$

(10)

(11)

(a) Test the simulation in the linear and nonlinear modes verifying that for a small frequency step the response matches theory as developed in the notes/book/and problem (1) above. Choose $B_n = 10$ Hz and $\zeta = 1/\sqrt{2}$.

(b) Repeat the test above, except now use a frequency the step size that causes one or more cycle slips before the loop pulls back in. Verify also that the cycle slipping only occurs in the nonlinear mode.
(c) Extend the baseband model of (a) to a complex baseband simulation model that takes input $e^{j\phi[n]}$ uses the DDS class as the phase controller since it outputs $e^{-j\hat{\theta}[n]}$. The phase detector will as in the DPLL lab output $\phi[n] = \text{Im}\{e^{j\theta[n]} \cdot e^{-j\hat{\theta}[n]}\}$. Verify that the complex baseband model performs similar to the baseband model, when in the nonlinear mode.

The classes, which can be found in http://www.eas.uccs.edu/~mwickert/ece5675/lecture_notes/ECE5675_Chapter3_DPLL.ipynb.zip, are also listed here:

```python
class DDS(object):
    """
    Implementation of a DDS that outputs exp(-1j*theta_hat)
    Mark Wickert October 2017
    """

def __init__ (self,fcenter,fs,kc = 1,state_init = 0):
    """
    Initialize the DDS with a center frequency in Hz, the sampling rate, the gain kc, and initial theta_hat state in radians.
    """
    self.fcenter = fcenter
    self.fs = fs
    self.delta_w = 2*pi*self.fcenter/self.fs
    self.kc = kc
    self.theta_hat = state_init
```
def update(self, e_in):
    
    Update the DDS phase accumulator
    
    self.theta_hat += self.delta_w + self.kc * e_in
    if self.theta_hat >= 2*pi:
        self.theta_hat -= 2*pi
    elif self.theta_hat < 0:
        self.theta_hat += 2*pi


def output_exp(self):
    
    Output exp(-1j*theta_hat)
    
    return exp(-1j*self.theta_hat)

def set_fcenter(self, fcenter_new):
    
    Set a new center frequency in Hz
    
    self.fcenter = fcenter_new
    self.delta_w = 2*pi*self.fcenter/self.fs

class accum(object):
    
    An accumulator class for use in DPLL loop filters
    
    Mark Wickert October 2017
    
    def __init__(self, state=0):
        
        Initialize the accumulator object by setting the state
        
        self.state = state

    def update(self, x_in):
        
        Update the accumulator
        
        self.state += x_in
Note: In modeling a complex baseband PLL, the NCO quiescent frequency, $f_{center}$ should be set to zero. Non-zero values are also used in the DPLL lab.

4. A PLL with loop filter is designed to track a carrier in additive noise, in a coherent communications system. The operating parameters are chosen such that $\zeta = 1/\sqrt{2}$, $\tau_2/\tau_1 = 0.002$, and $KA = K_p K_v = 6000$ (note this is a passive loop filter with $K_{LF} = 1$, see notes 4–40). Assume that the loop phase error $\phi(t)$ is small so the linear noise model is valid.

(a) Find the time constants $\tau_1$ and $\tau_2$.

(b) Find the single-sided loop noise equivalent bandwidth $B_L$.

(c) If $N_0 = -161$ dBm/Hz, find the received carrier power $P_c$, which yields $\sigma_\phi^2 = 0.1$ rad$^2$ using the linear noise theory.

Appendix: Function Listings for the Problem 1 Python helpers

```python
# PCM-PM Tx Using NRZ or Manchester
def PCM_PM_tx(N_bits,Ns,Kp,mode='NRZ'):
    """
    Complex baseband PCM-PM modulator for NRZ and Manchester pulse shaping.

    Parameters
    ----------
    N_bits : Number of random bits to produce
    Ns : Number of waveform samples per bit
    Kp : The peak phase deviation per bit
    mode : 'NRZ' or 'Manchester' pulse shaping

    Returns
    -------
    x_phase : phase modulated complex sinusoid
    x_mod : random bits PCM waveform
    b : the Tx pulse shape
    data : the underlying random data bits
    """
    if Ns/2 != Ns//2:
        error('Ns must be even')
    x, b, data = ss.NRZ_bits(N_bits,1)
    if mode == 'NRZ':
        p_NRZ = np.ones(Ns)
        b = p_NRZ
```

Mark Wickert February 2020
else:
    p_Manchester = np.hstack((np.ones(Ns//2),-np.ones(Ns//2)))
    b = p_Manchester
    x_mod = signal.lfilter(b,1,ss.upsample(x,Ns))
    x_phase = exp(1j*(Kp*x_mod))
    return x_phase, x_mod, b, data

def simple_bit_sync(x,Ns):
    ""
    Find the maximum eye opening across a bit stream by assuming
    synchronous sampling at Ns samples per bit.
    
    Parameters
    ----------
    x : Bit stream waveform with Ns samples per bit
    Ns : The number of sample per bit, assumed to be synchronous
    
    Returns
    -------
    idx_max : The index on the interval [0,Ns-1] giving the maximum
              eye opening
    ""
    sync_metric = zeros(Ns)
    for k in range(Ns):
        sync_metric[k] = mean(abs(x[k::Ns]))
    idx_max = nonzero(ravel(sync_metric == max(sync_metric)))[0]
    print('Estimated mod(Ns) bit synch index is {} of {}'
          .format(idx_max[0],Ns))
    return idx_max[0], sync_metric

def bit_detect_BEP(z_real, data_tx, Ns, N_transient = 1):
    ""
    Bit synchronize, bit detect, and compare with the transmit bits
    
    Parameters
    ----------
    z_real : a real bit stream waveform at Ns samples per bit; for a
             coherent demod this may be the real part after
             appropriate phase rotation
    data_tx : transmitted data bits used in probability of bit error
              (BEP) estimation
Ns : number of samples per bit
N_transient : Number of initial bits to discard due to waveform transient, which could be due to PLL acquisition settling

Returns
-------
N_total : number of bits processed
N_error : number of bits in error
opt_samples : max eye opening samples from the detection process

Mark Wickert March 2020

```
# Form bit synch estimate and obtain waveform samples at the max
# eye opening
N_opt, sync_metric = simple_bit_sync(z_real, Ns)
opt_samples = z_real[N_opt::Ns]
# Make hard decisions on data sample sign and transform
    # to [0,1] int's
data_hat = (int64(sign(opt_samples)+1)//2)[1:]
N_total, N_errors = dc.bit_errors(data_tx, data_hat,
    Ntransient = N_transient)
return N_total, N_errors, opt_samples
```