Posting Date: November 23, 2015.

Problem 1: Repeat Problem 6 of Homework Assignment # 4, but instead of using nested for-loops, use FFT-based correlation to expedite acquisition by exploring all possible code offsets simultaneously, as described in the lecture. Use the document `FFT-based acquisition of GPS signals.pdf` on iLearn as your guide. The FFT implementation in MATLAB, `fft`, is actually a discrete Fourier transform (DFT) and works for any number of input samples, not just a power-of-two number of samples. Thus, you may use the technique described in `FFT-based acquisition of GPS signals.pdf` by assuming N_s samples per C/A code. However, if you wish to make the FFTs in your FFT-based acquisition run faster, you can resample the incoming data so that there are a power-of-two number of samples per nominal C/A code.

Problem 2: In the lecture, we introduced three loop filters that lead to first-, second-, and third-order closed-loop transfer functions for code and carrier phase tracking loops:

- $F(s) = K$ [leads to first-order closed-loop transfer function $H(s)$]
- $F(s) = \frac{K(s+a)}{s}$ [leads to second-order closed-loop transfer function $H(s)$]
- $F(s) = \frac{K(s^2+as+b)}{s^2}$ [leads to third-order closed-loop transfer function $H(s)$]

The form of the loop filters is optimal for the step, ramp, and parabolic input, respectively, in that they minimize the integral square phase error under a bandwidth constraint. Experiment with these loop filters as follows:

1. Derive the closed-loop transfer function $H(s)$ for each loop, assuming the loop is composed of the loop filter $F(s)$ and a voltage-controlled-oscillator modeled as a perfect integrator $1/s$, as described in the lecture.
2. For each $H(s)$, develop an LTI model in MATLAB. You may use `tf` or a related function for this. Set the parameters as discussed in the lecture for a loop noise bandwidth $B_L = 10$ Hz.
3. For each of the LTI models, calculate the response to a step, ramp, and parabolic inputs using the Matlab `lsim` function or equivalent.
4. Using the final value theorem, calculate the steady-state error e_{ss} for each $H(s)$ and for each of step, ramp, and parabolic inputs. Compare your calculations against the plots from your corresponding "MATLAB experiments."
5. Generate a frequency response for each of the three $H(s)$ by invoking MATLAB's `bode` function.

Problem 3: Discretize the continuous-time LTI system models from Problem 2 by invoking the MATLAB `c2d` function. Assume a zero-order hold on the inputs as your discretization method. Let $H[z]$ represent the discretized system. Consider the following set of discretization intervals T, in milliseconds: $1, 10, 20, 40$.

(a) For each $H[z]$ and for each T in this set, plot the frequency response, the step response, and the ramp response.

(b) Given your results, what would be a good "rule of thumb" for the maximum value of the product $B_L T$ for which the behavior of the continuous-time system is approximately preserved, where B_L is the target loop noise bandwidth used in the design of the original continuous-time loop filters $F(s)$.

(c) Calculate the *actual* loop noise bandwidth for each of the discretized loops. The actual loop noise bandwidth tends to diverge from B_L as $B_L T$ increases. The following is some code that performs this calculation, with `H_z` being a MATLAB model of the discrete-time closed-loop LTI system. Explain what this code does and why it accurately estimates the actual loop noise bandwidth.

```
% Calculate BL_act
walias = pi/Ta;
wvec = [0:10000]'*(walias/10000);
[magvec,phsvec] = bode(H_z,wvec);
magvec = magvec(:);
Bn_act = sum(magvec.^2)*mean(diff(wvec))/(2*pi*(magvec(1,1)^2));
```

Problem 4: Write a MATLAB function that builds a discrete-time loop filter for a feedback tracking loop. The function should assume the loop is of the form of the linearized discrete-time Costas loop model that was discussed in the lecture. The function should return a state-space representation of the loop filter $F[z]$ and should also calculate the actual loop bandwidth of the closed-loop discretized system by analyzing the frequency response of the closed-loop transfer function $H[z]$. Your function should adhere to the following interface:

```
function [Ad,Bd,Cd,Dd,BL_act] = configureLoopFilter(BL_target,Tsub,loopOrder)
% configureLoopFilter : Configure a discrete-time loop filter for a feedback
%                       tracking loop.
%
% INPUTS
%
% BL_target ----- Target loop noise bandwidth of the closed-loop system, in Hz.
%
% Tsub ---------- Subaccumulation interval, in seconds.  This is also the loop
%                 update (discretization) interval.
%
% loopOrder ----- The order of the closed-loop system.  Possible choices are
%                 1, 2, or 3.
%
% OUTPUTS
%
% Ad,Bd,Cd,Dd --- Discrete-time state-space model of the loop filter.
%
% BL_act -------- The actual loop noise bandwidth (in Hz) of the closed-loop
%                 tracking loop as determined by taking into account the
%                 discretized loop filter, the implicit integration of the
%                 carrier phase estimate, and the length of the accumulation
%                 interval.
%
%+-----------------------------------------------------------------------------+
% References:
%+=============================================================================+
```

Hint: Here is some example code for the second-order case:

```
% Use is made here of the intermediate variables 'K' and 'a' as in the
% 'Phase-Locked Loops' paper by Gupta, 1975.
% The following calculations assume zeta = 0.707.
% Definitions of intermediate variables in terms of omegaN, zeta, BL:
%
%  zeta^2 = K/(4*a)
%  omegaN^2 = K*a
%  BL = (K+a)/4
%
% Set up open loop from theta[k] to thetahat[k]
K = 8*BL_target/3;
a = K/2;
omegaN = sqrt(K*a);
Fs = K*tf([1 a],[1 0]);
Fz  = c2d(Fs,Tsub,'zoh');  % Conversion to discrete time
NCO = tf([Tsub],[1 -1],Tsub);
PD = 1/2*tf([1 1],[1 0],Tsub);
sysOpenLoop = PD*Fz*NCO;
[Aol,Bol,Col,Dol]=ssdata(ss(sysOpenLoop));

% Convert the loop filter to a discrete-time state-space model
[Ad,Bd,Cd,Dd]=ssdata(Fz);
```

Problem 5: Write a MATLAB function that executes a single update to a phase tracking loop. Assume an arc-tangent phase detector of the form $e_k = \text{atan}(\tilde{Q}_k/\tilde{I}_k)$. The function should form the error e_k, process this error through a state-space realization of the loop filter $F[z]$ and then output the Doppler estimate v_k and the state of the loop filter. The function should adhere to the following interface:

```
function [xkp1,vk] = updatePll(s)
% updatePll : Perform a single update step of a phase tracking loop with an
%             arctangent phase detector.
%
% INPUTS
%
% s ------------- A structure with the following fields:
%
%    Ipk -------- The in-phase prompt accumulation over the interval from
%                 tkm1 to tk.
%
%    Qpk -------- The quadrature prompt accumulation over the interval from
%                 tkm1 to tk.
%
%    xk --------- The phase tracking loop filter's state at time tk.  The
%                 dimension of xk is N-1, where N is the order of the loop's
%                 closed-loop transfer function.
%
%  Ad,Bd,Cd,Dd -- The loop filter's state-space model.
%
% OUTPUTS
%
%    xkp1 -------- The loop filter's state at time tkp1. The dimension of xkp1
%                  is N-1, where N is the order of the loop's closed-loop
%                  transfer function.
```

```
%
%    vk ---------- The Doppler frequency shift that will be used to drive the
%                  receiver's carrier-tracking numerically controlled
%                  oscillator during the time interval from tk to tkp1, in
%                  rad/sec.
%
%+-------------------------------------------------------------------------------+
% References:
%
%+===============================================================================+
```

Write a top-level script to test your functions `configureLoopFilter` and `updatePll`. The top-level script should generate a fictitious phase time history, average this over T_{sub} to generate a time history of prompt in-phase and quadrature measurements $\tilde{I}_{p,k}$ and $\tilde{Q}_{p,k}$, and send these to the `updatePll` function one-by-one for processing. Take the zero vector as the initial state x_0 of the loop filter. Even starting from this initial zero state, the tracking loop will be able to pull in and obtain phase lock as long as the initial time rate of change of the fictitious phase time history is small.

Experiment with your phase tracking loop as follows:

(a) Set $T_{\text{sub}} = 10$ ms and the target B_L to 10 Hz.

(b) Generate a fictitious phase time history that you think will be difficult to track and see how the tracking loop performs.

(c) Modulate your phase time history with instantaneous 180-degree phase transitions, as would be caused by BPSK modulation. How does this affect the loop? Why?

(d) Add zero-mean Gaussian white noise to the $\tilde{I}_{p,k}$ and $\tilde{Q}_{p,k}$ measurements. Start small and increase the noise until you see cycle slipping.

(e) Figure out how to add a level of noise to the $\tilde{I}_{p,k}$ and $\tilde{Q}_{p,k}$ measurements that is commensurate with a given C/N_0 ratio. Then, reduce the C/N_0 (increase the noise variance relative to the amplitude of $\tilde{I}_{p,k}$ and $\tilde{Q}_{p,k}$) until you see cycle slipping. Identify this value of C/N_0 (expressed in dB-Hz) as the tracking threshold of your phase tracking loop.

Hint: Recall from the lecture the expression

$$\frac{C}{N_0} = \frac{E[|\tilde{S}_k|^2] - 2\sigma_{IQ}^2}{2\sigma_{IQ}^2 T_{\text{sub}}}.$$

Note that the numerator of the right-hand side is simply the noise-free squared amplitude of the vector $\tilde{S}_k$. In your simulation, you can set this to any arbitrary value in your calculation of $\tilde{I}_{p,k}$ and $\tilde{Q}_{p,k}$ from the original fictitious phase time history. Then, you can solve for σ_{IQ}^2 and generate noise with this variance. Add independent noise to the $\tilde{I}_{p,k}$ and $\tilde{Q}_{p,k}$ accumulations. Note that the C/N_0 value in the above equation is in units of Hz, not dB-Hz.

(f) How could you improve (decrease) the tracking threshold of your loop? Test your conjectures.

Problem 6: Write a MATLAB function that executes a single update to a first-order carrier-aided code tracking loop. In the lecture, we derived the mean of the error signal e_k for a coherent phase detector of the form $e_k = C \cdot [\tilde{I}_{e,k} - \tilde{I}_{l,k}]$, where C is a normalization constant given by $C = \frac{T_c}{2N_k}$. We showed that $E_c[e_k] = \Delta t_k$.

The coherent code phase detector is simple and has low error variance, but it requires the phase tracking loop to be in lock so that the quadrature components of $\tilde{S}_k$ can be safely ignored.

Consider instead the *dot-product* code phase detector:

$$e_k = C \cdot \left[(\tilde{I}_{e,k} - \tilde{I}_{l,k})\tilde{I}_{p,k} + (\tilde{Q}_{e,k} - \tilde{Q}_{l,k})\tilde{Q}_{p,k} \right],$$

where the normalization constant C is given by

$$C = \frac{T_c}{2} \left[\frac{2}{N_k \overline{A}_k} \right]^2$$

It can be shown that for small code phase errors, the mean value of e_k is approximately equal to Δt_k seconds.

The dot-product phase detector has slightly higher error variance than the coherent detector but does not require phase lock and so is better suited to the initial stages of tracking immediately following acquisition. Implement your code tracking loop with a dot-product phase detector. To obtain the normalization constant C, estimate $E[|\tilde{S}_k|^2]$ by averaging the past values of $|\tilde{S}_k|^2 = \tilde{I}_k^2 + \tilde{Q}_k^2$ (you could set up a moving-window average over the past 100 values, for example, or you could set up a low-pass filter that effectively does the same thing). You will find in your lecture notes (and in the interface below) a relationship between $E[|\tilde{S}_k|^2]$ and N_k, $\overline{A}_k$, and σ_{IQ}. Use this relationship to compute the normalization constant C from your estimate of $E[|\tilde{S}_k|^2]$. Your code tracking update function should adhere to the following interface:

```
function [vTotal] = updateDll(s)
% updateDll : Perform a single update step of a carrier-aided first-order code
%             tracking loop.
%
%
% INPUTS
%
% s ------------- A structure with the following fields:
%
%    BL_target -- Target bandwidth of the closed-loop code tracking loop, in
%                 Hz.
%
%    IsqQsqAvg -- The average value of |Stildek|^2 = Ipk^2 + Qpk^2; also equal to
%                 (Nk*Abark/2)^2 + 2*sigmaIQ^2.
%
%    sigmaIQ ---- The standard deviation of the noise in the prompt in-phase
%                 and quadrature subaccumulations.
%
%    Ipk -------- The in-phase prompt subaccumulation over the interval from
%                 tkm1 to tk.
%
%    Qpk -------- The quadrature prompt subaccumulation over the interval from
%                 tkm1 to tk.
%
%    Iek -------- The in-phase early subaccumulation over the interval from
%                 tkm1 to tk.
%
%    Qek -------- The quadrature early subaccumulation over the interval from
%                 tkm1 to tk.
%
%    Ilk -------- The in-phase late subaccumulation over the interval from
%                 tkm1 to tk.
%
%    Qlk -------- The quadrature late subaccumulation over the interval from tkm1
%                 to tk.
```

```
%
%    vpk -------- The aiding signal from the phase tracking loop, in seconds
%                 per second.  This is equal to the Doppler frequency shift
%                 that will be used to drive the receiver's carrier-tracking
%                 numerically controlled oscillator during the time interval
%                 from tk to tkp1, divided by the carrier frequency and
%                 multiplied by -1 (high-side mixing) or 1 (low-side mixing)
%                 depending on whether the RF front-end peforms high- or
%                 low-side mixing.  Thus, vpk = sMix*fDk/fc, with sMix = +/- 1.
%
%    Tc --------- Spreading code chip interval, in seconds.
%
% OUTPUTS
%
%    vTotalk ---- The code tracking loop's estimate of the code phase rate at
%                  sample time tk, in sec/sec. vTotal is equal to the code
%                  tracking loop's correction term vk plus the carrier aiding
%                  term vpk.
%
%+-------------------------------------------------------------------------------+
% References:
%
%
%+===============================================================================+
```