Alex Nguyen
Fall 2020

GPS Software-Defined Radio (SDR) Project

Part 1. GPS L1 Signal Corresponding to PRN 32:

(i) Top-Level Code and Functions

```
% Course: MAE 295 GNSS Signal Processing and SDR Design
% Date: December 14th, 2020
% Author: Alex Nguyen
% Description: Updated Project Part 1, Top-Level Acquisition and Tracking
% Script for PRN 32

clc; clear; close all

%% Load Saved Data:
load('C_N0dBVec.mat')       % C/N0 Data & sigIQ Squared
load('PRNdata.mat')         % Each SV PRN Sequences

%% Setup Parameters
% Speed of Light [m/s]:
c = 299792458;

% GPS L1 C/A Values:
fL1 = 1575.42e6;            % GPS L1 C/A Frequency [Hz]
Rc = 1.02300325e6;          % Chip Rate [chips/sec]
Nc = 1023;                  % Number of Chips per Subaccum
Tc = 1e-3/Nc;               % Chip Sampling Period [s]
fc = 1/Tc;                  % Chip Frequency [Hz]
fsampIQ = 2500/1e-3;        % IQ sampling frequency [Hz]
teml = 0.8*Tc;              % Early Minus Late [Chips]

% Subaccum:
Tsub = 1e-3;                            % Subaccum Sampling Interval
N = floor(fsampIQ*Tsub);                % # of Subaccums per Subaccum
t = linspace(0, Tsub - 1/fsampIQ, N)';  % Subaccum Time Interval [s]

% Load Data:
Tfull = 60;                                    % Time Interval of Data to Load [s]
tDataV = Tfull/Tsub;                           % Full Time Data [s]
Ylen = floor(Tfull*fsampIQ);
fid = fopen('mystery_data_file2.bin','r','l'); % Open File
Y = fread(fid, [2, Ylen], 'int16')';           % Read Binary Data File
Y = Y(:,1) + 1i*Y(:,2);
fclose(fid);                                   % Close File

%% (a) Acquire a PRN Sequence
% Initialize:
sv = 32;                              % PRN Signal to Acquire
SubAccum = 1;                         % Subaccum # for GPS L1 C/A

% Perform Acquisition:
[ts_hat, fDk, C_N0dB, sig, Sk_sq] = acquireSV(sv, SubAccum);

%% (b) Initialize Beat Carrier Phase Estimate
% Carrier Phase:
th_beat = 0;                    % (Arbitary) Initial Beat Carrier Phase

%% (c) Initiailze Moving-Window Average
```

```
% Initial In-Phase and Quadrature Peak Value:
Ip0_sq = real(Sk_sq); Qp0_sq = imag(Sk_sq);
SkAvg = Sk_sq;

%% (d) Initialize Phase Tracking Loop Filter
% Initialize:
BL_target = 10;  % Target Bandwidth [Hz]
loopOrder = 3;   % Loop Order
BL_codeT = 0.2;  % Code Tracking Loop [Hz]

% Phase Tracking Loop Filter:
[Ad, Bd, Cd, Dd, BL_act] = configureLoopFilter(BL_target, Tsub, loopOrder);

% Initial State Matrix:
M = [Ad(1, 1) - 1, Ad(1, 2); ...
       Cd(1)     ,   Cd(2)];

b = [0; 2*pi*fDk];

% Initialize States:
xk = M\b;

%% (f) Tracking Signal
% Preallocate:
ts_hatV = zeros(tDataV, 1);
th_beatV = ts_hatV;
fDV = ts_hatV;
IVec = ts_hatV; QVec = ts_hatV;
ek_p = ts_hatV; ek_d = ts_hatV;
C_N0dBV = ts_hatV;
vTotalkV = ts_hatV;

% Initialize:
jk = 1; jkend = N;       % Input Accumulation Indicies
s.BL_target = BL_codeT;  % Dll Target Bandwidth
s.IsqQsqAvg = 0;         % Average Sk Value

% Gold Code SV Sequence for Subaccum:
Gpm = Gpm_os(sv, :)';

for ii = 1:tDataV
% Carrier Phase Estimate:
th_hat = 2*pi*fDk*t + th_beat;

% Correlator Block:
[Spk, Sek, Slk] = ...
    CorrelatorBlk(Y(jk:jkend), Gpm, fsampIQ, ts_hat, th_hat, teml);

% Update Moving-Window Average:
SkAvg = Spk.*conj(Spk);
C_N0 = (SkAvg - 2*sigIQ_sq)/2/sigIQ_sq/Tsub;
C_N0dB = 10*log10(C_N0);

% Initialize updatePll.m Structure:
s.Ipk = real(Spk); s.Qpk = imag(Spk);
s.xk = xk;
s.Ad = Ad; s.Bd = Bd;
s.Cd = Cd; s.Dd = Dd;

% Phase Tracking Loop:
[xkp1, vk, ek_pll] = updatePll(s);
```

```
% Initialize updateDll.m Structure:
s.IsqQsqAvg = (s.IsqQsqAvg*ii + SkAvg)/ii;
s.sigmaIQ = sqrt(sigIQ_sq);
s.Ipk = real(Spk); s.Qpk = imag(Spk);
s.Iek = real(Sek); s.Qek = imag(Sek);
s.Ilk = real(Slk); s.Qlk = imag(Slk);
s.vpk = vk/2/pi/fL1;
s.Tc = Tc;

% Carrier-Aided Tracking Loop:
[vTotalk, ek_dll] = updateDll(s);

% Save:
fDV(ii) = fDk;
ts_hatV(ii) = ts_hat;
th_beatV(ii) = th_beat;
IVec(ii) = real(Spk);
QVec(ii) = imag(Spk);
ek_d(ii) = ek_dll; ek_p(ii) = ek_pll;
C_N0dBV(ii) = C_N0dB;
vTotalkV(ii) = vTotalk;

% Update:
xk = xkp1;                                 % Loop Filter State
fDk = vk/2/pi;                             % Approximate Doppler
ts_hat = ts_hat + (Nc*Tc)/(1 + vTotalk);   % Code Start Time
th_beat = th_beat + vk*Tsub;               % Beat Carrier Phase Estimate
jk = jk + N; jkend = jkend + N;            % Complex Subaccum Indices
end

%% Estimate of j0 Index
j0 = ts_hatV(1)*fsampIQ;

% Print Results:
fprintf('Part (ii): \n')
fprintf('Sample # After First Complete C/A Code Interval: '); disp(vpa(j0));

%% Plot Results
t = linspace(0, Tfull, length(ts_hatV));

% (a) Code Error [chips]
figure(1);
plot(t, ek_d/Tc, 'Color', '#D95319', 'linewidth', 1)
xlabel('Time [s]');
ylabel('$e_k$', 'interpreter', 'latex');
legend('Arctangent (AT)')
title(sprintf('(a) SV # %d: Code Error [Chips]', sv))

% (b) Phase Error [deg]
figure(2);
plot(t, ek_p*360/2/pi, 'Color', '#A2142F', 'linewidth', 1)
xlabel('Time [s]');
ylabel('$e_k$', 'interpreter', 'latex');
legend('Arctangent (AT) ')
title(sprintf('(b) SV # %d: Phase Tracking Loop Error [deg]', sv))

% (c) Estimated Doppler Frequency [Hz]
figure(3);
plot(t, fDV, 'Color', '#77AC30', 'linewidth', 1.5);
xlabel('Time [s]');
```

```
ylabel('$fD_k$', 'interpreter', 'latex');
title(sprintf('(c) SV # %d: Estimated Doppler Frequency [Hz]', sv));

% (d) Estimated Carrier-to-Noise Ratio [dB-Hz]
figure(4);
plot(t, real(C_N0dBV), '.' , 'Color', '#4DBEEE', 'linewidth', 1)
xlabel('Time [s]');
ylabel('$C/N0$', 'interpreter', 'latex');
title(sprintf('(d) SV # %d: Estimated Carrier-to-Noise Ratio [dB-Hz]', sv))

% (e) Tracked In-Phase and Quadrature Components
figure(5);
plot(t, IVec, 'Color', '#EDB120', 'linewidth', 1.5); hold on;
plot(t, QVec, 'Color', '#7E2F8E', 'linewidth', 1.5);
xlabel('Time [s]');
legend('$\tilde{I}_k$', '$\tilde{Q}_k$', 'interpreter', 'latex', 'location', 'best');
title(sprintf('(e) SV # %d: Tracked In-Phase and Quadrature Subaccum', sv));

% (f) Code Phase Estimate [m]
figure(6)
plot(t, ts_hatV*c, 'Color', '#77AC30', 'linewidth', 1.5);
xlabel('Time [s]');
title(sprintf('(f) SV # %d: Code Phase Estimate [m]', sv));

% (g) Phase Portrait of Sk AFTER Lock
iiPostLock = 2500;  % Sample index achieving PLL Lock

figure(7)
plot(IVec(iiPostLock:end), QVec(iiPostLock:end), '.');
xlabel('$\tilde{I}_k$', 'interpreter', 'latex');
ylabel('$\tilde{Q}_k$', 'interpreter', 'latex');
title(sprintf('(g) SV # %d: Sk Complex Plane after Lock', sv));


function [ts0, fD0, C_N0dB, sig, Pk_t] = acquireSV(sv, SubAccum)
% acquireSV: Acquires the SV signal then outputs the inital Doppler,
%            intial code start time, Carrier-to-Noise Ratio, determine if a
%            signal is present, and Initial peak signal power used in moving
%            window average.
%
% Inputs
%       sv: PRN #
% SubAccum: # of Coherent Sumsubaccums
%
% Outputs
%      ts0: Initial Code Start Time [s]
%      fD0: Initial Doppler Estimate [Hz]
%   C_N0dB: Carrier-to-Noise Ratio [dB]
%      sig: (1) Signal Present or (0) No Signal Present
%
%-----------------------------------------------------------------------%
%% Load Data (Previously Computed):
load('PRNdata.mat')
load('C_N0dBVec.mat')

%% Initialize Parameters:
% GPS L1 C/A Values:
Rc = 1.02300325e6;          % Chip Rate [chips/sec]
Tc = 1e-3/1023;             % Chip Sampling Period [s]
fsampIQ = 2500/1e-3;        % IQ sampling frequency [Hz]
Tfull = 5;                  % Time Interval of Data to Load [s]
```

```
Ylen = floor(Tfull*fsampIQ);

% Subaccum:
Tsub = 1e-3;                % Subaccum Sampling Interval
N = floor(fsampIQ*Tsub);    % # of Subaccums per Subaccum

% Load Data:
fid = fopen('mystery_data_file2.bin','r','l');  % Open File
Y = fread(fid, [2, Ylen], 'int16')';           % Read Binary Data File
Y = Y(:,1) + 1i*Y(:,2);
fclose(fid);                                   % Close File

%% Signal Acquisition:
% Doppler Frequency Time Mesh:
fLow = -6000; fHigh = 6000; df = 150; f = fLow:df:fHigh;

% Code Start Time Mesh:
tLow = 0; tHigh = Tsub - 1/fsampIQ; t = linspace(tLow, tHigh, N)';

% SV Code Delay Chips:
SV = [5 6 7 8 17 18 139 140 141 251 252 254 255 256 257 258 469 470 471 ...
      472 473 474 509 512 513 514 515 516 859 860 861 862 863 950 947 ...
      948 950]';

% Carrier-to-Noise Ratio for NO Signal Present:
C_N0dB_avg = mean(C_N0dB(C_N0dB > 40));
clear C_N0dB

% Initialize:
Sk_sq = 0;
Gpm = Gpm_os(sv, :)';

for M = 1:SubAccum
    % Initialize:
    k0 = 1; kend = N;
    Zk = zeros(length(f), length(t));

    for ff = 1:length(f)
        % Complex Mixing of Complex Sequence:
        xk = Y(k0:kend).*exp(-1i*2*pi*f(ff)*t);

        % Discrete Fourier Transform:
        Xr = fft(xk);       % Mixed Complex Signal
        Cr = fft(Gpm);      % PRN Sequence

        % Circular Correlation Sequence
        Zr = Xr.*conj(Cr);
        zk = ifft(Zr);

        % |Sk_tilde|.^2 Per Subaccum:
        Zk(ff, :) = zk.*conj(zk);

    end
    % Sum Previous Subaccums:
    Sk_sq = Sk_sq + Zk;

    % Update Complex Signal Indices:
    k0 = k0 + N; kend = kend + N;

end
% Average GPS L1 C/A Signal Value:
```

```
Sk_sq = Sk_sq/SubAccum;

% GPS L1 C/A Signal Max Value:
[Pk_t, loc_t] = max(max(Sk_sq));
[~, loc_f] = max(max(Sk_sq'));

% Carrier-to-Noise Ratio [dB-Hz]:
C_N0 = (Pk_t - 2*sigIQ_sq)/2/sigIQ_sq/Tsub;
C_N0dB = 10*log10(C_N0);

% Determine if GPS L1 C/A Signal is Present:
if C_N0dB > C_N0dB_avg
    % Approximate Values:
    ts0 = t(loc_t);      % Code Start Time Offset [s]
    fD0 = f(loc_f);      % Apparent Doppler Frequency [Hz]

    % Determine if Signal Present:
    sig = 1;

else
    sig = 0;
end

end


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.
%
%-----------------------------------------------------------------------------

if loopOrder == 1                 % First-Order Case
% Define Parameter K:
K = 4*BL_target;

% Type 1 Transfer Function:
Fs = K*tf(1, 1);

elseif loopOrder == 2             % Second-Order Case
% Define Parameters K & a:
```

```
K = 8*BL_target/3;
a = K/2;

% Type 2 Transfer Function:
Fs = K*tf([1 a], [1 0]);

elseif loopOrder == 3             % Third-Order Case
% Define Parameters K, a, & b:
a = 1.2*BL_target;
b = a.^2/2;
K = 2*a;

% Type 3 Transfer Function:
Fs = K*tf([1 a b], [1 0 0]);

else                              % Wrong Loop-Order Choice
    error('Choose the Loop Order to be Either 1, 2, or 3!');
end

% Convert Loop Filter to a DT State-Space Model:
Fz = c2d(Fs, Tsub, 'zoh');        % DT Transfer Function
[Ad, Bd, Cd, Dd] = ssdata(Fz);

% Linearized DT Costas Open-Loop System:
NCO = Tsub*tf(1, [1 -1], Tsub);    % Number-Controlled Oscillator (NCO)
PD = 1/2*tf([1 1], [1 0], Tsub); % DT Averager
sysOpenLoop = PD*Fz*NCO;

% Linearized DT Costas Closed-Loop System:
H_z = sysOpenLoop/(1 + sysOpenLoop);

% Calculate Actual Loop Noise Bandwidth:
walias = pi/Tsub;
wvec = (0:10000)'*(walias/10000);
[magvec, ~] = bode(H_z, wvec);
magvec = magvec(:);
BL_act = sum(magvec.^2)*(wvec(2) - wvec(1))/(2*pi); % Approx. Bn

end


function [Spk, Sek, Slk] = CorrelatorBlk(Y, Gpm, fsampIQ, ts_hat, th_hat, teml)
% Inputs
%       Y: Complex Signal
%     Gpm: Oversampled PRN #
% fsampIQ: Signal Sampling Frequency
%  ts_hat: Code Start Time Estimate
%  th_hat: Beat Carrier Phase Estimate
%    teml: Early-Minus-Late Time
%
% Outputs
%     Spk: Prompt Signal
%     Sek: Early Signal
%     Slk: Late Signal
%-----------------------------------------------------------------------%

% Circular Shift PRN:
Tprom   = circshift(Gpm, round(ts_hat*fsampIQ));
TemlAdv = circshift(Gpm, floor((ts_hat - teml/2)*fsampIQ));
TemlDel = circshift(Gpm, floor((ts_hat + teml/2)*fsampIQ));
```

```
% Wipe Real Signal with Doppler and Code Start Estimates:
Sk = Y.*exp(-1i*th_hat);

% Summate Subaccums:
Spk = sum(Sk.*Tprom);     % Prompt
Sek = sum(Sk.*TemlAdv);   % Advance
Slk = sum(Sk.*TemlDel);   % Delay

end


function [lfsrSeq] = generateLfsrSequence(n,ciVec,a0Vec)
%
% Generate a 1/0-valued linear feedback shift register (LFSR) sequence.
%
% INPUTS
%
% n ------ Number of stages in the linear feedback shift register.
%
% ciVec -- Nc-by-1 vector whose elements give the indices of the Nc nonzero
% connection elements.
%
% a0Vec -- n-by-1 1/0-valued initial state of the LFSR. In defining the
% initial LFSR state, a Fibonacci LFSR implementation is assumed.
%
% OUTPUTS
%
% lfsrSeq -- m-by-1 vector whose elements are the 1/0-valued LFSR sequence
% corresponding to n, ciVec, and a0Vec, where m = 2^n - 1. If the
% sequence is a maximal-length sequence, then there is no
% repetition in the m sequence elements.
%
%+------------------------------------------------------------------------------+

m = 2^n - 1;              % LSFR Sequence Period
Nc = length(ciVec);       % Characterisitc Polynomial Length
c(ciVec(:)) = 1;          % Connection Vector
a = a0Vec;                % LSFR State (Assuming Fibonacci Implementation)
lfsrSeq = zeros(m, 1);    % LFSR Sequence Preallocation
b = zeros(Nc - 1, 0);     % Logic Gate Result Preallocation

for k = 1:m-1
    % Logic Gate Operation(s)
    b(1) = xor(a(ciVec(1)), a(ciVec(2)));

    if Nc > 2
        for i = 1:Nc - 2

            b(i + 1) = xor(a(ciVec(i + 2)), b(i));

        end
    end

    % Shifting LSFR Sequence
    j = 1:n-1;
    a(n + 1 - j) = a(n - j);
    a(1) = b(Nc - 1);

    % Save LSFR Sequence Output
    lfsrSeq(k + 1) = a(end);
end
```

```
lfsrSeq(1) = a0Vec(end); % Initial LFSR Sequence Output

end


function [codeOS] = oversampleSpreadingCode(code,delChip,Ns,Np)
%[codeOS] = oversampleSpreadingCode(code,delChip,Ns,Np)
%
% Oversample the +/-1-valued input pseudorandom spreading code.
%
% INPUTS
%
% code ------------ N-by-1 vector containing the +/-1-valued input
%                   pseudorandom spreading code.  The spreading code is
%                   assumed to be periodic with period Np.
%
% delChip --------- Sampling interval, measured in spreading code chip
%                   intervals.  To ensure good auto- and cross-correlation
%                   properties, the sampling rate should not be an integer
%                   multiple of the chipping rate.  Thus, n*delChip should not
%                   equal 1 for any integer n.
%
% Ns -------------- Number of samples required for the output oversampled
%                   code.
%
% Np -------------- Assumed number of chips in one spreading code period.
%
%
% OUTPUTS
%
% codeOS ---------- Ns-by-1 vector containing the +/-1-valued oversampled
%                   spreading code.

if((sum(code == 1) + sum(code == -1))~= length(code))
  error('Spreading code is assumed to be +/-1-valued');
end

if(abs(mod(1/delChip,1)) < 1e-6)
  error(['To ensure good auto- and cross-correlation '...
         'properties, the sampling rate should not be an integer '...
         'multiple of the chipping rate.  Thus, n*delChip should not '...
         'equal 1 for any integer n.']);
end

codeOS = zeros(Ns,1);
for ii=1:Ns
  k = floor((ii-1)*delChip);
  k = mod(k, Np);
  codeOS(ii) = code(k+1);
end


function [xkp1, vk, ek_pll] = 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.
%
%+------------------------------------------------------------------------------+

% Phase Detector (Discriminator):
ek_pll = atan(s.Qpk./s.Ipk);           % Arc-Tangent (AT)

% Next Time Step Calculations (@ tkp1):
xkp1 = s.Ad*s.xk + s.Bd*ek_pll;        % Loop Filter States
vk = s.Cd*s.xk + s.Dd*ek_pll;          % Doppler Frequency Shift

end


function [vTotalk, ek_dll] = 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.
%
%+------------------------------------------------------------------------------+

%----- Dot Product Discriminator (Non-Coherent):
% Normalization Constant:
GainF = 1/(s.IsqQsqAvg - 2*s.sigmaIQ.^2);  % Gain Factor (Derive using E[|Sk|^2] Relation)
C = s.Tc/2*GainF;

% Phase Detector (or Discriminator):
ek_dll = C*((s.Iek - s.Ilk)*s.Ipk + (s.Qek - s.Qlk)*s.Qpk);

%----- Code Tracking Loop Correction:
% Good Choice: 1st Order Loop with Gain K = 4*B_DLL (0.05 <= B_DLL <=0.2 [Hz])
Fz = 4*s.BL_target;
vk = Fz*ek_dll;

%----- Code Tracking Loop's Estimate:
vTotalk = s.vpk + vk;

end
```

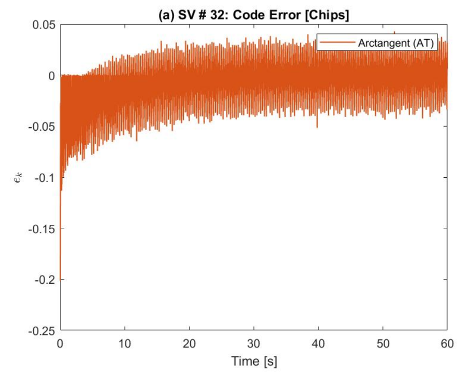
(ii) Estimated Index of First Sample

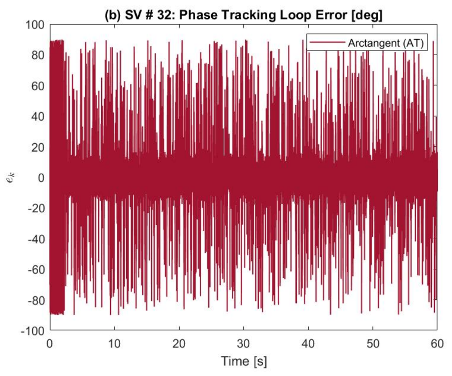
The index of the first sample after the start time of the first complete C/A code interval was determined by the following equation:

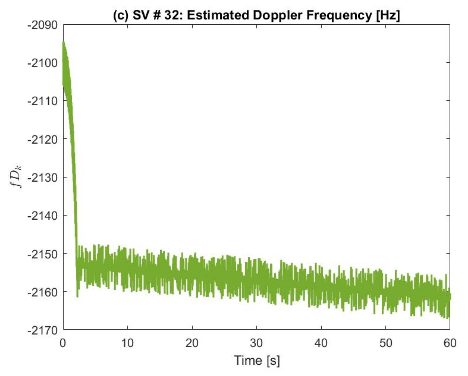
$$j_0 = \hat{t}_s(\tau_j) * F_s$$

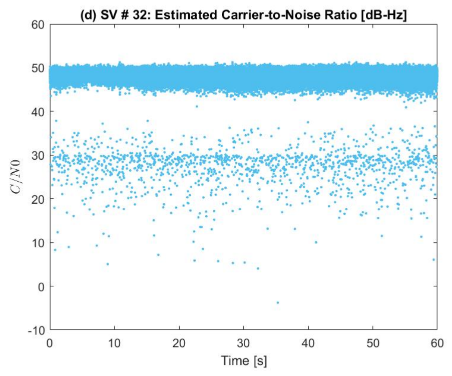
Using this, $j_0 \approx 1160$

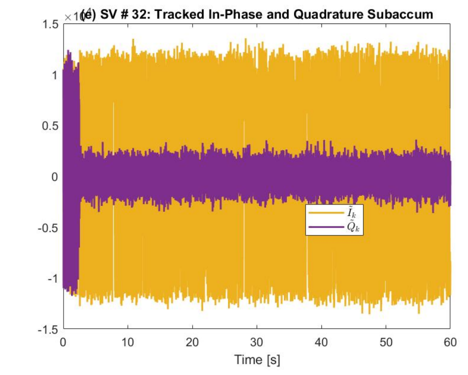
(iii) Time History Plots over Full 60 Seconds

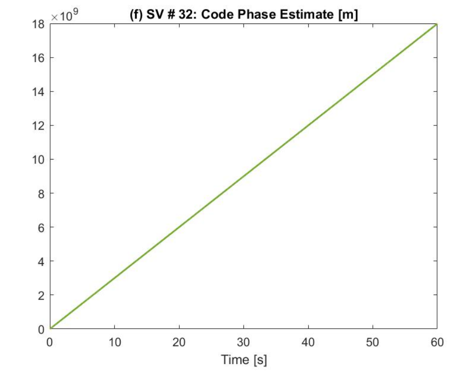


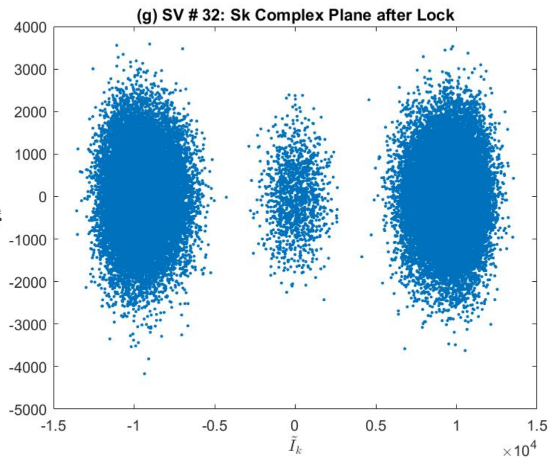












Part 2. Determine Receiver Samples

Here, only track PRNs 10, 11, 14, 31, and 32. We are to correctly determine the start time for the C/A codes about 54 seconds into the data in terms of the first sample within a particular code. We are given all but the last two digits of the start time samples.

The way which I determined these sample values was by using the given table with my calculated start time from the tracking measurements. For each given PRN receiver sample, I assumed the last two digits to be "00" then converted this sample value to time. After, I matched the actual code start time with the approximately calculated one. This gave the following table.

PRN	tRxRaw (samples)	tsV (seconds)
10	135000624.03612601757049560546875	442735.598
11	135002049.17623004317283630371094	442735.601
14	135000189.48001381754875183105469	442735.604
31	135001479.94217893481254577636719	442735.605
32	135001345.63660186529159545898438	442735.601

Now we need to convert the start time samples to seconds then offset them by some amount to bring them near to GPS time. The offset used will be 442681.598 seconds. After performing this conversion we get the following pseudorange values in the units of meters.

$$\rho = \begin{bmatrix} 74832.5365426457 \\ -653646.347244119 \\ -1776032.87001699 \\ -1921077.00329330 \\ -738012.705547532 \end{bmatrix} \text{ meters}$$

Part 3. Navigation Solution:

Finally, the navigation solution will be to find the receiver's location based on the pseudorange values and information from the transmitting SVs. These quantities will be used in a nonlinear least squares estimator to find the receiver's position in an ECEF reference frame.

The pseudorange model used, which accounts for tropospheric and ionospheric effects, is the following:

$$\rho = \Delta r + c[\delta t_R - \delta t_{SV}] + c[\delta t_{Iono} + \delta t_{Tropo}] + w_\rho$$

The $\Delta r, w_\rho$ quantities are given in meters and the $\delta t_R, \delta t_{SV}, \delta t_{Iono}, \delta t_{Tropo}$ quantities are given in seconds. Below is the given values for the transmitting SVs at the instant of transmission.

```
Data for PRN 10
rSvECEF = 9121305.7077300 -23601900.3375000 7954520.8506900
dtIono = 2.273761e-08
dtTropo = 1.622817e-08
dtSV = -0.000100688843623000

Data for PRN 11
rSvECEF = -22226444.3115000 -7263521.9296100 12021436.2202000
dtIono = 1.945468e-08
dtTropo = 1.321500e-08
dtSV = -0.000664972769672000

Data for PRN 14
rSvECEF = -7443729.8892700 -15525794.7538000 20526967.7486000
dtIono = 1.439808e-08
dtTropo = 9.258317e-09
dtSV = -0.000036453122726400

Data for PRN 31
rSvECEF = -6485206.0043400 -24828766.7942000 6135430.7428900
dtIono = 1.545366e-08
dtTropo = 1.003285e-08
dtSV = 0.000246467440852000

Data for PRN 32
rSvECEF = 4189914.7336700 -14908405.1919000 21590991.0269000
dtIono = 1.854755e-08
dtTropo = 1.245384e-08
dtSV = -0.000125803551001000
```

In order to calculate the noise vector, we will calculate the pseudorange variance defined by the following:

$$\sigma^2_{\Delta t} = \frac{dB_{DLL}T_c^2}{2(C/N_0)} \quad ; \text{where } d = \frac{T_{eml}}{T_c}$$

The measurment noise variance depends on the loop noise bandwidth of the DLL, the signal, and the d chosen for your design considerations. Here, the variance is in units of sec^2 but we want units of meters so we multiply by the speed of light (i.e. c $\approx 3 \times 10^8 m/s$). The noise was assumed to be zero-mean white noise with a standard deviation calculated by the above equation.

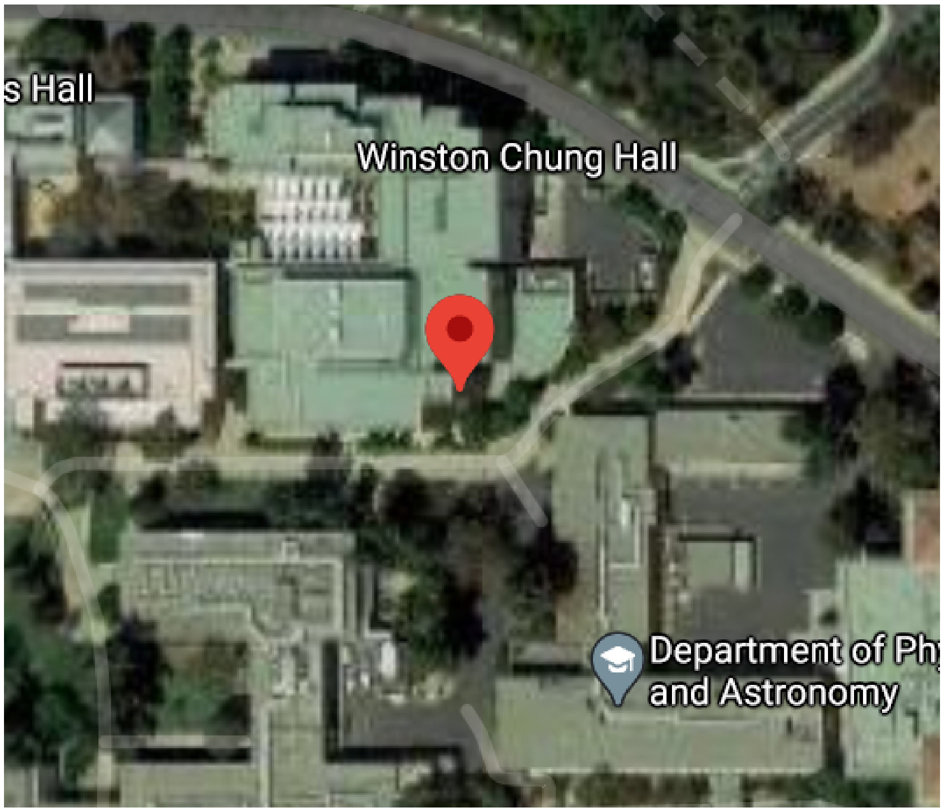
Now, we have everything we need to solve for the state vector $\mathbf{x}_r = [x_r,\, y_r,\, x_r,\, \delta t_r]^\top$. My nonlinear least squares algorithm yields the following state vector values.

$$\mathbf{x}_r = \begin{bmatrix} -2430684.35778553 & \text{meters} \\ -4704170.03733361 & \text{meters} \\ 3544313.78024733 & \text{meters} \\ -2.50780668477888 \times 10^{-10} & \text{secs} \end{bmatrix}$$

Out of curiosity, I converted the ECEF reference frame coordinates to latitude, longitude, and altitude which yielded the following values:

$$\begin{bmatrix} \text{Latitude} \\ \text{Longitude} \\ \text{Altitude} \end{bmatrix} = \begin{bmatrix} 33.9750817176747 \\ -117.325780373809 \\ 284.528689404018 \end{bmatrix}$$

Hence, this receiver was located at **UC Riverside** at the instant of transmission.


s Hall
Winston Chung Hall
Department of Ph
and Astronomy