The value of the ionospheric delay for the parameters given in the exam is `delTauG = 1.8271e-08` seconds.

```
function [delTauG] = getIonoDelay(ionodata,fc,rRx,rSv,tGPS,model)
% getIonoDelay : Return a model-based estimate of the ionospheric delay
%                experienced by a transionospheric GNSS signal as it
%                propagates from a GNSS SV to the antenna of a terrestrial
%                GNSS receiver.
%
% INPUTS
%
% ionodata ------- Structure containing a parameterization of the
%                  ionosphere that is valid at time tGPS.  The structure is
%                  defined differently depending on what ionospheric model
%                  is selected:
%
%                  broadcast --- For the broadcast (Klobuchar) model, ionodata
%                  is a structure containing the following fields:
%
%                     alpha0 ... alpha3 -- power series expansion coefficients
%                     for amplitude of ionospheric TEC
%
%                     beta0 .. beta3 -- power series expansion coefficients
%                     for period of ionospheric plasma density cycle
%
%
%                  Other models TBD ...
%
% fc ------------- Carrier frequency of the GNSS signal, in Hz.
%
% rRx ------------ A 3-by-1 vector representing the receiver antenna position
%                  at the time of receipt of the signal, expressed in meters
%                  in the ECEF reference frame.
%
% rSv ------------ A 3-by-1 vector representing the space vehicle antenna
%                  position at the time of transmission of the signal,
%                  expressed in meters in the ECEF reference frame.
%
% tGPS ----------- A structure containing the true GPS time of receipt of
%                  the signal.  The structure has the following fields:
%
%                  week -- unambiguous GPS week number
%
%                  seconds -- seconds (including fractional seconds) of the
%                  GPS week
%
% model ---------- A string identifying the model to be used in the
%                  computation of the ionospheric delay:
%
%                  broadcast --- The broadcast (Klobuchar) model.
%
%                  Other models TBD ...
```

```
%
% OUTPUTS
%
% delTauG -------- Modeled scalar excess group ionospheric delay experienced
%                  by the transionospheric GNSS signal, in seconds.
%
%+------------------------------------------------------------------------------+
% References: For the broadcast (Klobuchar) model, see IS-GPS-200F
% pp. 128-130.
%
%+==============================================================================+
% Copyright © 2006 Isaac Miller, all rights reserved.  No portion of this code
% may be reproduced, reused, or distributed in any form without prior written
% permission of the author.

if(strcmp(model,'broadcast')~=1)
  error('Unrecognized model.');
end

% extract ionospheric parameters
alpha0 = ionodata.alpha0;
alpha1 = ionodata.alpha1;
alpha2 = ionodata.alpha2;
alpha3 = ionodata.alpha3;
beta0 = ionodata.beta0;
beta1 = ionodata.beta1;
beta2 = ionodata.beta2;
beta3 = ionodata.beta3;
% receiver geodetic latitude and longitude, in radians
[lat,lon,alt] = ecef2lla(rRx);
% satellite elevation and azimuth
[sEL, sAZ] = satelaz(rSv,rRx);
% earth's central angle between receiver and earth projection of ionospheric
% intersection point, in radians
psi = 0.0137*pi/(sEL/pi + 0.11) - 0.022*pi;
% slant factor to account for non-straight path thru ionosphere
Fs = 1 + 16*(0.53 - sEL/pi)^3;
% ionospheric intersection point geodetic latitude (rad.)
phi_i = lat + psi*cos(sAZ);
if (abs(phi_i) > pi*0.416)
    phi_i = pi*0.416*sign(phi_i);
end
% ionospheric intersection point geodetic longitude (rad.)
lamda_i = lon + psi*sin(sAZ)/cos(phi_i);
% geomagnetic latitude (rad.)
phi_m = phi_i + 0.064*pi*cos(lamda_i - 1.617*pi);
% geomagnetic latitude (semicircles) for expansion coefficients
phi_m_ss = phi_m / pi;
% local time of day in seconds (local offset + time at Greenwich)
tlocal = 43200*lamda_i/pi + mod(tGPS.seconds,86400);
if (tlocal < 0)
    tlocal = tlocal + 86400;
```

```
end
if (tlocal >= 86400)
    tlocal = tlocal - 86400;
end
% ionospheric period (sec)
T = beta0 + beta1*phi_m_ss + beta2*phi_m_ss^2 + beta3*phi_m_ss^3;
if (T < 72000)
    T = 72000;
end
% local time angle (rad.)
theta = 2*pi*(tlocal - 50400)/T;
% amplitude of ionospheric fluctuation
C1 = alpha0 + alpha1*phi_m_ss + alpha2*phi_m_ss^2 + alpha3*phi_m_ss^3;
if (C1 < 0)
    C1 = 0;
end
% final ionospheric delay (sec)
if (abs(theta) < 1.57)
    delTauG = Fs * (5e-9 + C1*(1 - theta^2/2 + theta^4/24));
else
    delTauG = Fs * 5e-9;
end

% The above calculations are for the GPS L1 frequency.  Multiply delTauG by
% (fL1^2)/(fc^2) to map to the carrier frequency fc.
fL1 = 1575.42e6;
delTauG = delTauG*((fL1/fc)^2);
```

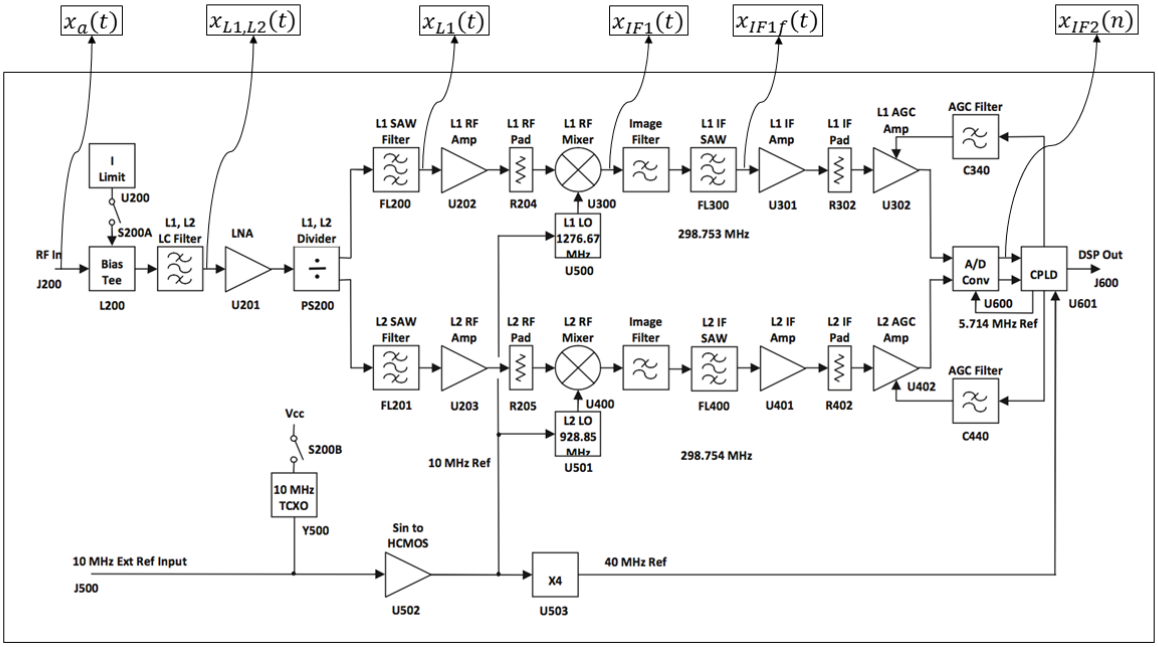

Figure 1: Bobyn front end schematic.

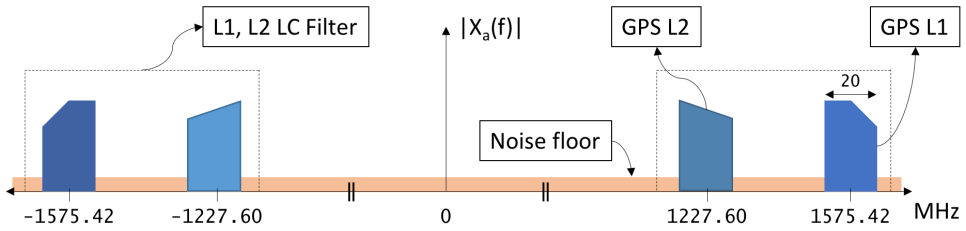

Figure 2: $x_a(t)$ in frequency domain.

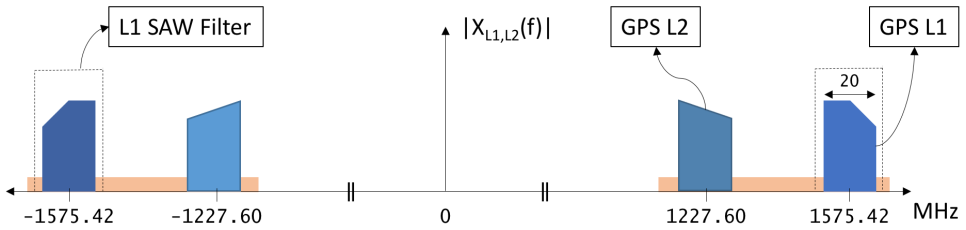

Figure 3: $x_{L1,L2}$ in frequency domain.

Figure 1 shows the front end schematic. Figure 2 shows the input RF signal consisting of both L1 and L2 GPS signals. The GPS L1 signal, centered at 1575.42 MHz, has a bandwidth of approximately 20 MHz. Notice that at this stage the noise floor is spread across all frequencies. This assumes that the antenna has infinite bandwidth, which will not be the case in practice.

The first LC (built with an inductor ("L") and a capacitor ("C")) filter selects a wide band that includes both GPS L1 and L2 bands and restricts the bandwidth of the noise. This is shown in Figure 3. The signal is then

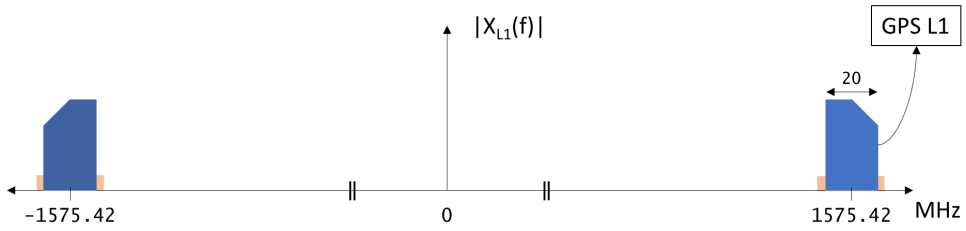

Figure 4: $x_{L1}(t)$ in frequency domain.

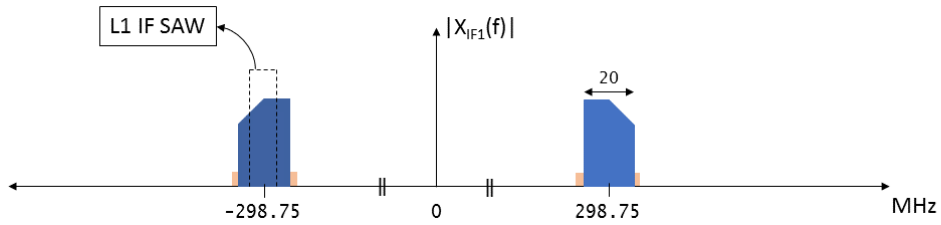

Figure 5: $x_{IF1}(t)$ in frequency domain.

split into two separate RF chains. We focus on the L1 chain in this problem. In the L1 chain, first the GPS L1 signal is isolated. It is assumed that the L1 SAW filter has a one-sided bandwidth of approximately 20 MHz. The output of the L1 SAW filter is $x_{L1}(t)$, and its frequency transform is shown in Figure 4. Subsequently, $x_{L1}(t)$ is mixed with a local carrier at $f_l = 49790/39 = 1276.66666666666$ MHz. After the mixing stage, the signal x_{IF1}(t) is centered at $f_{IF1} = 298.753333333333$ MHz. Note that the frequency indicated under the L1 IF SAW filter is the approximate center frequency of the SAW filter, not that of the signal after mixing.

The frequency transform of $x_{IF1}(t)$ is shown in Figure 5. After mixing, the signal is filtered with the L1 IF SAW filter, resulting in $x_{IF1f}(t)$. The frequency transform of $x_{IF2}(t)$ is shown in Figure 6. Note the reduced bandwidth of the signal. The output of the L1 IF SAW filter is then sampled at 40/7 MHz by an ADC. This sampling operation leads to aliasing of $x_{IF1f}(t)$. When sampled at 40/7 MHz, the frequency domain representation of the resulting signal $x_{IF2}(t)$ is only unique between $-f_s/2 = -\pi = -2.857143$ MHz and $f_s/2 = \pi = 2.857143$ MHz. Note that the final intermediate frequency that applies for the discrete-time signal is

$$f_{IF2} = |f_{IF1} - f_s\lfloor f_{IF1}/f_s \rceil| = 1.610476190476 \quad \text{MHz}$$

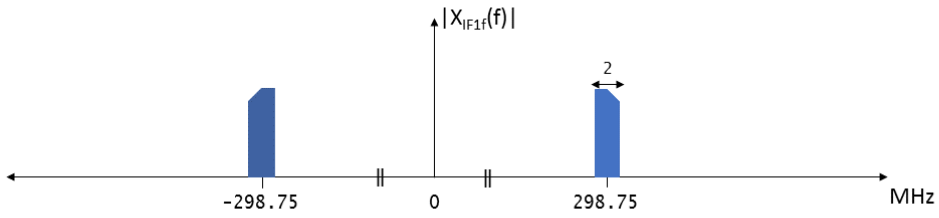

Figure 6: $x_{IF1f}(t)$ in frequency domain.

Also note that, because the argument of $|\cdot|$ is positive, the frequency domain representation of the discrete-time signal in the first Nyquist zone is **not** inverted as compared to the analog input signal $x_a(t)$. This follows from the fact that all mixing and sampling operations in this front-end are low-side.

iq2if.m

```
function [xVec] = iq2if(IVec,QVec,Tl,fIF)
% IQ2IF : Convert baseband I and Q samples to intermediate frequency samples.
%
% Let xl(m*Tl) = I(m*Tl) + j*Q(m*Tl) be a discrete-time baseband
% representation of a bandpass signal.  This function converts xl(n) to a
% discrete-time bandpass signal x(n) = I(n*T)*cos(2*pi*fIF*n*T) -
% Q(n*T)*sin(2*pi*fIF*n*T) centered at the user-specified intermediate
% frequency fIF, where T = Tl/2.
%
%
% INPUTS
%
% IVec -------- N-by-1 vector of in-phase baseband samples.
%
% QVec -------- N-by-1 vector of quadrature baseband samples.
%
% Tl ---------- Sampling interval of baseband samples (complex sampling
%               interval), in seconds.
%
% fIF --------- Intermediate frequency to which the baseband samples will
%               be up-converted, in Hz.
%
%
% OUTPUTS
%
% xVec -------- 2*N-by-1 vector of intermediate frequency samples with
%               sampling interval T = Tl/2.
%
%
%+-------------------------------------------------------------------------+
% References:
%
%
%+=========================================================================+

%----- Setup
N = length(IVec);
if(length(QVec) ~= N)
  error('IVec and QVec must be of same length');
end
T = Tl/2;
tIfVec = [0:2*N-1]'*T;

%----- Interpolate and resample the baseband quadrature signals
IVecInterp = interp(IVec,2);
QVecInterp = interp(QVec,2);

%----- Upconvert
cVec = sqrt(2)*cos(2*pi*fIF*tIfVec);
sVec = sqrt(2)*sin(2*pi*fIF*tIfVec);
xVec = IVecInterp.*cVec - QVecInterp.*sVec;
```

if2iq.m

```
function [IVec,QVec] = if2iq(xVec,T,fIF)
% IF2IQ : Convert intermediate frequency samples to baseband I and Q samples.
%
% Let x(n) = I(n*T)*cos(2*pi*fIF*n*T) - Q(n*T)*sin(2*pi*fIF*n*T) be a
% discrete-time bandpass signal centered at the user-specified intermediate
% frequency fIF, where T is the bandpass sampling interval.  Then this
% function converts the bandpass samples to quadrature samples from a complex
% discrete-time baseband representation of the form xl(m*Tl) = I(m*Tl) +
% j*Q(m*Tl), where Tl = 2*T.
%
%
% INPUTS
%
% xVec -------- N-by-1 vector of intermediate frequency samples with
%               sampling interval T.
%
% T ----------- Sampling interval of intermediate frequency samples, in
%               seconds.
%
% fIF --------- Intermediate frequency of the bandpass signal, in Hz.
%
%
% OUTPUTS
%
% IVec -------- N/2-by-1 vector of in-phase baseband samples.
%
% QVec -------- N/2-by-1 vector of quadrature baseband samples.
%
%
%+-------------------------------------------------------------------------------+
% References:
%
%
%+===============================================================================+

%----- Setup
N = 2*floor(length(xVec)/2);
xVec = xVec(1:N);
tIfVec = [0:N-1]'*T;

%----- Downconvert
cVec = sqrt(2)*cos(2*pi*fIF*tIfVec);
sVec = -sqrt(2)*sin(2*pi*fIF*tIfVec);
bbcVec = xVec.*cVec;
bbsVec = xVec.*sVec;

%----- Filter and resample
IVec = decimate(bbcVec,2);
QVec = decimate(bbsVec,2);
```

Figure 1 shows the PSD estimate of the given baseband IQ data. The baseband IQ samples are converted to IF using the function `iq2if`. The IF frequency f_{IF} was chosen to be 5 MHz, which is one half of the Nyquist rate of the `iq2if`-converted signal. The resulting PSD estimate of the IF data is shown in Figure 2. Observe that the PSD is now symmetrical about zero. This implies that the data is real, as enforced in the last line of `iq2if.m`. Also observe that the PSD is centered at 5 MHz, as desired. Real samples at 20 MHz are needed to sample the IF signal without aliasing. Note that the peak of the IF PSD is 3 dB below the corresponding peak in the IQ PSD. This is because a scaling of $\sqrt{2}$ was used in the function `iq2if`. If no scaling were used, this peak would go down by 6 dB.

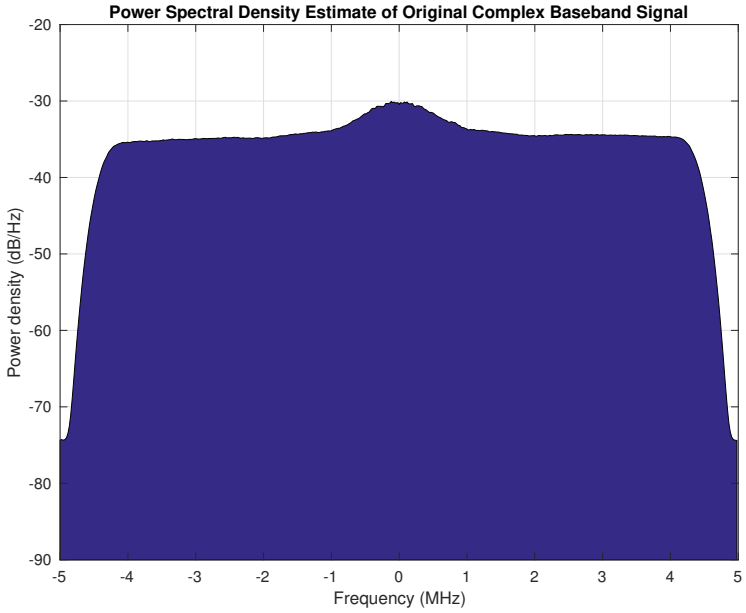

Figure 1: PSD estimate of the original baseband IQ samples in `niData01head.bin`.

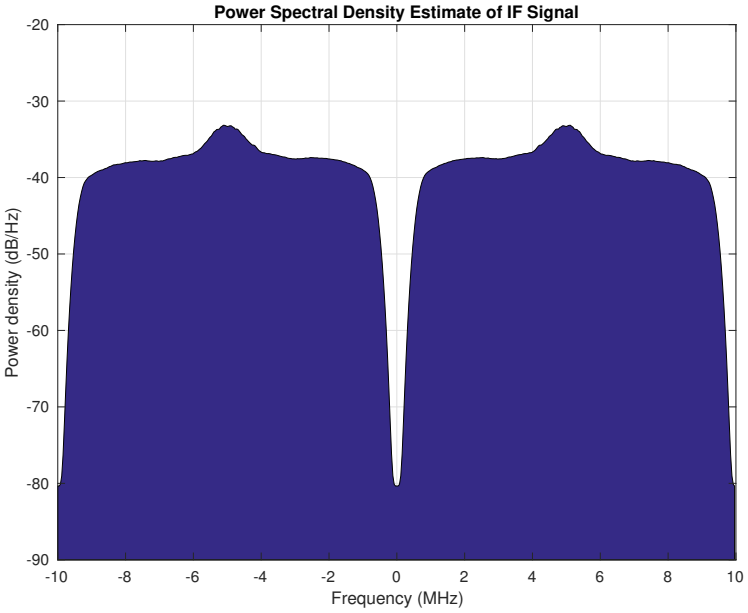

Figure 2: PSD estimate of the IF data generated using `iq2if.m`.

The IF signal at 5 MHz is then converted back to baseband using the function `if2iq`. The resulting PSD is shown in Figure 3. This recreated PSD is very similar to the PSD of the original IQ samples near the center frequency. The edges of the PSD of the recreated samples fall more sharply as compared to the original PSD. This is the effect of the default lowpass Chebyshev Type I IIR filter of order 8 used by the `MATLAB` function `decimate`. For comparison, Figure 4 shows the recreated PSD when an FIR filter of 4th order is used instead of the default IIR filter. The amplitude of the peak in Figure 3 is the same as the corresponding peak in the original PSD and 3 dB higher than the peaks in the IF PSD. Once again, this is due to the $\sqrt{2}$ scaling used in the function `if2iq`.

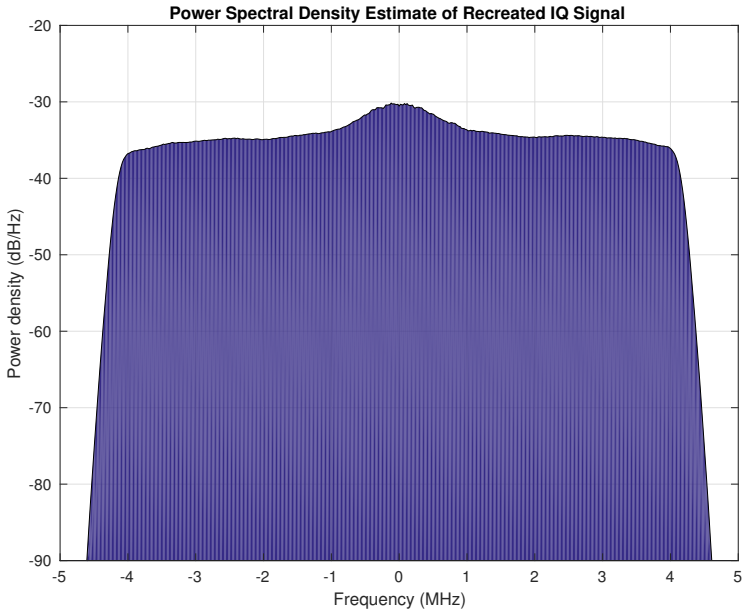

Figure 3: PSD estimate of the IQ data recreated using `if2iq.m`.

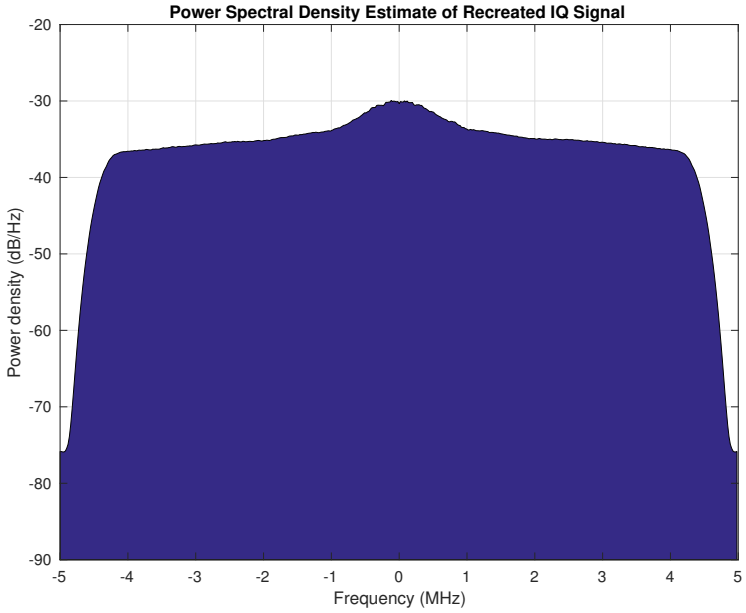

Figure 4: PSD estimate of the IQ data recreated using `if2iq.m` with an FIR filter of order 2.

We have that $f_{L1} = 1575.42$ MHz and $B = 4$ MHz. Thus,

$$f_H = 1577.42 \text{ MHz}$$
$$f_L = 1573.42 \text{ MHz}$$

In lecture, it was claimed that, to avoid aliasing, the sampling frequency f_s must satisfy

$$\frac{2f_H}{k} \leq f_s \leq \frac{2f_L}{k-1}$$

where

$$1 \leq k \leq \left\lfloor \frac{f_H}{B} \right\rfloor, \quad k \in \mathbb{Z}$$

Clearly, the maximum value of k is

$$k_{\max} = 394$$

from which we find the minimum frequency that avoids aliasing:

$$\begin{aligned} f_{s,\min} &= \frac{2f_H}{k_{\max}} \\ &= 8007208.12182741 \text{ Hz} \end{aligned}$$

Sampling at $f_s = f_{s,\min}/0.8 = 10009010.1522843$ Hz is a bad idea because this sampling frequency falls in the forbidden region for every possible integer k. For $k_1 = 316$, we have that

$$\frac{2f_H}{k_1} = 9983670.88607595 \leq f_s \leq 9989968.25396825 = \frac{2f_L}{k_1 - 1}$$

and for $k_2 = 315$, we have that

$$\frac{2f_H}{k_2} = 10015365.0793651 \leq f_s \leq 10021783.4394904 = \frac{2f_L}{k_2 - 1}$$

Thus, the desired sampling frequency 10009010.1522843 Hz does not fall in any permissible region.

Setting $W = B/0.8 = 5$ MHz, and $f'_H = mW$ for $m = 316$ gives

$$f'_H = 1580 \text{ MHz}$$
$$f'_L = 1575 \text{ MHz}$$

This scheme would avoid aliasing provided the original signal were filtered so that no signal remained below f'_L. But this would cause significant distortion and loss of signal power. Even if we accepted this, the lack of a buffer on the low-frequency side would allow a small error in sampling rate to cause aliasing: the scheme is not robust.

Working scheme: Assume that a total of 1 MHz of excess bandwidth is available, as before. Choose a guard band of 0.5 MHz on both sides.

$$f^*_L = f_L - 500000 = 1572.92 \text{ MHz}$$
$$f^*_H = f_H + 500000 = 1577.92 \text{ MHz}$$
$$B^* = B + 0.5 + 0.5 = 5 \text{ MHz}$$

Choosing $k^* = \lfloor f^*_H / B^* \rfloor = 315$, we have a permissible sampling range of

$$\frac{2f^*_H}{k^*} = 10018539.6825397 \leq f_s \leq 10018598.7261146 = \frac{2f^*_L}{k^* - 1}$$

Thus, a nominal sampling frequency of $(10018539.6825397 + 10018598.7261146)/2 = 10018569.2043272$ Hz is a good choice. Recall that it is permitted to go outside these limits of 10018539.6825397 Hz and 10018598.7261146 Hz without causing aliasing because of the spare 500 kHz. More specifically, with $k^* = 315$, the permissible sampling rates for the original signal are

$$\frac{2f_H}{k^*} = 10015365.0793651 \leq f \leq 10021783.4394904 = \frac{2f_L}{k^* - 1}$$

Thus, this scheme provides a protection of 3174.603175 Hz in the negative direction and 3184.713376 Hz in the positive direction.

There are seven signals with $C/N_0 > 35$ dB-Hz present in the data set.

- PRNs 1, 11, and 18 can be acquired with $N \geq 1$ non-coherent summations of 1-ms coherent accumulations.
- PRNs 14 and 31 can be acquired with $N \geq 4$ non-coherent summations of 1-ms coherent accumulations.
- PRN 23 can be acquired with $N \geq 11$ non-coherent summations of 1-ms coherent accumulations. It fades in and out at the beginning of the data set. The successful acquisition of PRN 23 indicated in the table below occurred around 80 ms into the data set. The code start time was extrapolated back to the beginning of the data set using the measured Doppler.
- PRN 27 can be acquired with $N \geq 11$ non-coherent summations of 1-ms coherent accumulations. The successful acquisition of PRN 27 indicated in the table below occurred around 80 ms into the data set. The code start time was extrapolated back to the beginning of the data set using the measured Doppler.

The result of acquisition is shown in Table 1.

PRN	Doppler (Hz)	Code Start (μsec)	C/N$_0$ (dB-Hz)
1	-33800	411.08	48.0
11	-35900	578.55	49.5
14	-20200	35.53	39.6
18	-34200	370.30	50.6
23	-38500	544.0	36
27	-3350	895.5	35.5
31	-25300	804.30	40.8

Table 1: Acquisition Results

`Top-level script for acquisition`

```
clear; clc; close all; format long g;

fIF = 5e6;

Tfull = 0.5;          % Time interval of data to load
fsampIQ = 10.0e6;     % IQ sampling frequency (Hz)
N = floor(fsampIQ*Tfull);
nfft = 2^9;           % Size of FFT used in power spectrum estimation

% ======================================================================== %
%                              Load data                                   %
% ======================================================================== %
fid = fopen('../datafiles/niData03head_10MHz.bin','r','l');
Y = fread(fid, [2,N], 'int16')';
Y = Y(:,1) + (1i)*Y(:,2);
fclose(fid);

% Coherent integration time. CHANGE THIS.
Ta = 1e-3;
% Number of non-coherent integrations. CHANGE THIS.
N = 8;

% ======================================================================== %
```

```
%                               IQ to IF                                 %
% ======================================================================== %
X = iq2if(real(Y), imag(Y), 1/fsampIQ, fIF);

% ======================================================================== %
%                           Generate Codes                               %
% ======================================================================== %
[prnCodeTable] = generatePrnCodeTable(2*fsampIQ, Ta);

% ======================================================================== %
%                             Acquisition                                %
% ======================================================================== %
% Number of samples in an accumulation.
Nk = floor(Ta * 2 * fsampIQ);
% Sample times.
tVec = (0 : 1/(2*fsampIQ) : (Nk-1)/(2*fsampIQ))';
% Search vector for Doppler.
dopplerStep = 1/(4*Ta);
dopplerVec = -3.5e3 : dopplerStep : 3.5e3;
% Search vector for code phase.
codeStep = 3; % Shift by 3 samples
codeVec = 0 : (codeStep / (2 * fsampIQ)) : 1e-3 - (1 / (2 * fsampIQ)); % All possible start times

for prn = 1 : size(prnCodeTable, 1)

    % Initialize to save time.
    M = zeros(length(codeVec), length(dopplerVec));

    disp(['PRN: ' num2str(prn)]);
    ff = 1; % Frequency bin index
    locCode = prnCodeTable(prn,:)'; % Generate code replica once
    for fD = dopplerVec
        disp(['fD = ' num2str(fD) ' Hz']);
        % Generate local carrier replica
        locExp = exp( -(1i) * (2*pi*fIF*tVec + 2*pi*fD*tVec) );
        % Combined code and carrier local replica
        locReplica = locExp .* locCode;
        % Loop over non-coherent integrations
        for nn = 1 : N
            jj = 1; % code phase index
            for jk = 1 : codeStep : (2 * fsampIQ * 1e-3)
                % Dot product for fast performance
                % Add non-coherently
                M(jj, ff) = M(jj, ff) + ...
                            abs( locReplica' * (X(((nn-1)*Nk)+jk : ((nn-1)*Nk)+jk+Nk-1)) )^2;
                jj = jj + 1;
            end
        end
        ff = ff + 1;
    end

    surf(dopplerVec, codeVec, M);
    title(['PRN: ' num2str(prn)]);

    % Ask human to detect acquisition
    acq = input('Is peak detected? (1 = Yes): ');
    if isempty(acq)
        acq = 0;
    end

    if acq
        % Find peak
        [~, maxCol] = max(max(M));
        fDoppler = dopplerVec(maxCol);
        [~, maxRow] = max(max(M'));
```

```
        codeStart = codeVec(maxRow);

        % C/N0 computation
        MCrop = M( [1:(maxRow-40), (maxRow+40):end], ...
                   [1:(maxCol-(1000/dopplerStep)), (maxCol+(1000/dopplerStep)):end] );
        MCropVec = MCrop(:);
        tsigmaIQ_sq = mean(MCropVec); % 2*sigma_{IQ}^2

        C_N0 = 10 * log10( (max(max(M)) - tsigmaIQ_sq) / (tsigmaIQ_sq*Ta) );

        disp(['Doppler = ' num2str(fDoppler)]);
        disp(['Code Start = ' num2str(codeStart*1e6)]);
        disp(['C/N_0 = ' num2str(C_N0)]);
    end
end
```

generatePrnCodeTable.m

```
function [prnCodeTable] = generatePrnCodeTable(fs, Ta)

N = 1023;
delChip = 1.023e6 / fs;
Ns = Ta * fs;

n = 10;
a0Vec = ones(10, 1);

ciVec_G1 = [3, 10];
G1 = generateLfsrSequence(n, ciVec_G1, a0Vec);

ciVec_G2 = [2, 3, 6, 8, 9, 10];
G2 = generateLfsrSequence(n, ciVec_G2, a0Vec);

delayTable = [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];

prnCodeTable = zeros(length(delayTable), Ns);

for ii = 1 : length(delayTable)
    G2i = [G2(end-delayTable(ii)+1 :  end); G2(1 :  end-delayTable(ii))];
    prnCode = bitxor(G1, G2i);
    prnCodeOS = oversampleSpreadingCode(sign(prnCode-0.5), delChip, Ns, N);
    prnCodeTable(ii, :) = prnCodeOS;
end
```

generateLfsrSequence.m

```
function [lfsrSeq] = generateLfsrSequence(n,ciVec,a0Vec)
% [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.  For example, if the characteristic polynomial
%          of an LFSR is f(D) = 1 + D^2 + D^3, then ciVec = [2,3]' or [3,2]'.
%
% a0Vec -- n-by-1 1/0-valued initial state of the LFSR, where a0Vec = [a(-1),
%          a(-2), ..., a(-n)]'.  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;
lfsrSeq = zeros(m, 1);

cVec = zeros(n, 1);
cVec(ciVec) = 1;

aVec = a0Vec;
for ii = 1 : m
    lfsrSeq(ii) = aVec(end);
    axci = aVec .* cVec;
    if mod(sum(axci), 2)
        aVec = [1; aVec(1 :  end-1)];
    else
        aVec = [0; aVec(1 :  end-1)];
    end
end
```

Coherence time of white frequency noise (phase random walk). Letting $\sigma_\omega = 0.01$ radians per sampling interval, with a sampling interval of $T = 1$ ms, the coherence time of white frequency noise (phase random walk) is $T_{\text{coh}} = 50$ seconds.

Coherence time of white frequency rate noise (frequency random walk). Letting $\sigma_\alpha = 0.0001$ radians per sampling interval squared, with a sampling interval of $T = 1$ ms, the coherence time of white frequency rate noise (frequency random walk) is $T_{\text{coh}} = 1.59$ seconds.

Coherence time of white phase noise. As N increases, white phase noise approaches an asymptotic coherence $C_{\text{coh}}(N) \approx 0.73$. Thus, it does not make sense to refer to a coherence time for white phase noise because as N increases the coherence becomes independent of time.

```
% coherenceExperiments
%
% Experiments with the phase noise models and the coherence function

clear; clc;
processTypeVec = {'whitePhase','whiteFrequency','whiteFrequencyRate'};
processType = processTypeVec{3};

% White phase noise
if(strcmp('whitePhase', processType))
  sigmaPhaseNoise = 0.8;  % radians
  N = 10000;
  DeltaThetaVec = sigmaPhaseNoise*randn(N,1);
  figure(1);clf
  plot(DeltaThetaVec);
  title('\Delta \theta(\tau_j)')
  xlabel('Sample');
  ylabel('\Delta \theta(\tau_j)   (rad)');
  Ccoh = computeCoherence(DeltaThetaVec,N);
  Ccoh
end

% White frequency noise (phase random walk)
if(strcmp('whiteFrequency', processType))
  Nt = 10000;
  Ccoh2Vec = zeros(Nt,1);
  for jj = 1:Nt
    sigmaFrequencyNoise = 0.01;  % radians per sampling interval
    N = 50000;
    DeltaFrequencyVec = sigmaFrequencyNoise*randn(N,1);
    DeltaThetaVec = zeros(N,1);
    for ii=2:N
      DeltaThetaVec(ii) = DeltaThetaVec(ii-1) + DeltaFrequencyVec(ii-1);
    end
    % figure(1);clf
    % plot(DeltaThetaVec);
    % title('\Delta \theta(\tau_j)')
```

```
    % xlabel('Sample');
    % ylabel('\Delta \theta(\tau_j)   (rad)');
    Ccoh = computeCoherence(DeltaThetaVec,N);
    Ccoh2Vec(jj) = Ccoh.^2;
  end
  mean(Ccoh2Vec)
end

% White frequency rate noise (frequency random walk)
if(strcmp('whiteFrequencyRate', processType))
  Nt = 100000;
  Ccoh2Vec = zeros(Nt,1);
  for jj = 1:Nt
    sigmaFrequencyRateNoise = 0.0001;  % radians per sampling interval squared
    N = 1590;
    DeltaFrequencyRateVec = sigmaFrequencyRateNoise*randn(N,1);
    DeltaFrequencyVec = zeros(N,1);
    DeltaThetaVec = zeros(N,1);
    for ii=2:N
      DeltaFrequencyVec(ii) = DeltaFrequencyVec(ii-1) + ...
          DeltaFrequencyRateVec(ii-1);
      DeltaThetaVec(ii) = DeltaThetaVec(ii-1) + DeltaFrequencyVec(ii-1);
    end
    % figure(1);clf
    % plot(DeltaThetaVec);
    % title('\Delta \theta(\tau_j)')
    % xlabel('Sample');
    % ylabel('\Delta \theta(\tau_j)   (rad)');
    Ccoh = computeCoherence(DeltaThetaVec,N);
    Ccoh2Vec(jj) = Ccoh.^2;
  end
  mean(Ccoh2Vec)
end
```

```
function [Ccoh] = computeCoherence(DeltaThetaVec,N)
% computeCoherence : Compute the value of the discrete-time coherence function
%                    Ccoh(N).
%
%
% INPUTS
%
% DeltaThetaVec ----- Ns-by-1 vector representing a sampled carrier phase
%                     error time history, in rad.
%
% N ----------------- The number of samples that will be used to evaluate
%                     the coherence Ccoh(N).
%
%
% OUTPUTS
%
```

```
% Ccoh -------------- The value of the discrete-time coherence function for
%                     the first N samples of DeltaThetaVec.
%
%+-------------------------------------------------------------------------+
% References: Thompson, Moran, and Swenson, "Interferometry and Synthesis in
% Radio Astronomy," p. 277.
%
%
%+=========================================================================+

if(length(DeltaThetaVec) < N)
  error('The length of DeltaThetaVec must not be less than N');
end
Ccoh = abs((1/N)*sum(exp(j*DeltaThetaVec(1:N))));
```

In this problem, the beat carrier phase $\theta(\tau_j)$ is unknown. We are told that $\theta(\tau_j) = \tilde{\theta}_2 \sim \mathcal{U}(0, 2\pi)$ and that $\tilde{\theta}_2$ is independent of the noise samples $\{n(j) \mid j \in \mathcal{J}_k\}$.

In this case, the parameter vector can be modeled as $\boldsymbol{\theta} = [\tilde{\theta}_1, \tilde{\theta}_2]^\mathsf{T}$, with $\tilde{\theta}_1 \in \{0, A\}$, and $\tilde{\theta}_2 \sim \mathcal{U}(0, 2\pi)$. The hypothesis test can be written as

$$\begin{aligned} H_0 : \boldsymbol{\theta} \in \mathcal{X}_0 &\triangleq \left\{ \boldsymbol{\theta} \in \mathcal{X} | \tilde{\theta}_1 = 0 \right\} \\ H_1 : \boldsymbol{\theta} \in \mathcal{X}_1 &\triangleq \left\{ \boldsymbol{\theta} \in \mathcal{X} | \tilde{\theta}_1 = A \right\} \end{aligned}$$

The distribution of $\boldsymbol{z}$ under H_0 is

$$\begin{aligned} p(\boldsymbol{z} | \boldsymbol{\theta} \in \mathcal{X}_0) &= \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \sigma_n^2 I) \\ &= \frac{1}{(2\pi)^{N_k/2} \sigma_n^{N_k}} \exp\left(-\frac{\boldsymbol{z}^\mathsf{T} \boldsymbol{z}}{2\sigma_n^2} \right) \end{aligned}$$

Define $\boldsymbol{s}(\xi)$ as

$$\boldsymbol{s}(\xi) \triangleq [s(j_k, \xi), s(j_k + 1, \xi), \ldots, s(j_k + N_k - 1, \xi)]^\mathsf{T}$$

where

$$s(j, \xi) \triangleq D[\tau_j - t_d(\tau_j)] C[\tau_j - t_s(\tau_j)] \cos\left[2\pi f_{\mathrm{IF}} \tau_j + \xi\right]$$

Then the distribution of $\boldsymbol{z}$ under H_1 can be written as

$$\begin{aligned} p(\boldsymbol{z} | \boldsymbol{\theta} \in \mathcal{X}_1) &= \int_0^{2\pi} p(\boldsymbol{z} | \tilde{\theta}_1 = A, \tilde{\theta}_2 = \xi) \underbrace{p_2(\xi)}_{\mathcal{U}[\xi; 0, 2\pi]} \, \mathrm{d}\xi \\ &= \frac{1}{2\pi} \int_0^{2\pi} \frac{1}{(2\pi)^{N_k/2} \sigma_n^{N_k}} \exp\left\{ -\frac{(\boldsymbol{z} - A\boldsymbol{s}(\xi))^\mathsf{T} (\boldsymbol{z} - A\boldsymbol{s}(\xi))}{2\sigma_n^2} \right\} \mathrm{d}\xi \end{aligned}$$

Now form the likelihood ratio

$$\begin{aligned} \Lambda(\boldsymbol{z}) &= \frac{1}{2\pi} \frac{\int_0^{2\pi} \exp\left\{ -\frac{(\boldsymbol{z} - A\boldsymbol{s}(\xi))^\mathsf{T} (\boldsymbol{z} - A\boldsymbol{s}(\xi))}{2\sigma_n^2} \right\} \mathrm{d}\xi}{\exp\left\{ -\frac{\boldsymbol{z}^\mathsf{T} \boldsymbol{z}}{2\sigma_n^2} \right\}} \\ &= \frac{1}{2\pi} \frac{\cancel{\exp\left\{ -\frac{\boldsymbol{z}^\mathsf{T} \boldsymbol{z}}{2\sigma_n^2} \right\}} \int_0^{2\pi} \exp\left\{ \frac{2A\boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{z} - A^2 \boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{s}(\xi)}{2\sigma_n^2} \right\} \mathrm{d}\xi}{\cancel{\exp\left\{ -\frac{\boldsymbol{z}^\mathsf{T} \boldsymbol{z}}{2\sigma_n^2} \right\}}} \\ &= \frac{1}{2\pi} \int_0^{2\pi} \exp\left\{ \frac{2A\boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{z} - A^2 \boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{s}(\xi)}{2\sigma_n^2} \right\} \mathrm{d}\xi \\ &= \frac{1}{2\pi} \int_0^{2\pi} \exp\left\{ \frac{A}{\sigma_n^2} \boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{z} \right\} \exp\left\{ -\frac{A^2}{2\sigma_n^2} \boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{s}(\xi) \right\} \mathrm{d}\xi \\ &= \frac{1}{2\pi} \int_0^{2\pi} \exp\left\{ \frac{A}{\sigma_n^2} \boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{z} \right\} \exp\left\{ -\frac{A^2}{2\sigma_n^2} \sum_j s^2(j, \xi) \right\} \mathrm{d}\xi \end{aligned}$$

Note that $s^2(j, \xi) = D^2\left[\tau_j - t_d(\tau_j)\right] C^2\left[\tau_j - t_s(\tau_j)\right] \cos^2\left[2\pi f_{\mathrm{IF}} \tau_j + \xi\right]$. By design, D^2 and C^2 evaluate to 1 at all times. Thus, we have that

$$\Lambda(\boldsymbol{z}) = \frac{1}{2\pi} \int_0^{2\pi} \exp\left\{ \frac{A}{\sigma_n^2} \boldsymbol{s}(\xi)^\mathsf{T} \boldsymbol{z} \right\} \exp\left\{ -\frac{A^2}{2\sigma_n^2} \sum_j \cos^2\left[2\pi f_{\mathrm{IF}} \tau_j + \xi\right] \right\} \mathrm{d}\xi$$

The sum in the above equation involves addition of N_k samples with sampling period T, where $1/N_kT$ is considerably smaller than f_{IF}. As a result, this summation is well approximated by the long-term average of $\cos^2(.)$ which evaluates to $1/2$.

$$\begin{aligned}\Lambda(\boldsymbol{z}) &= \frac{\exp\left\{-\frac{A^2}{4\sigma_n^2}\right\}}{2\pi}\int_0^{2\pi}\exp\left\{\frac{A}{\sigma_n^2}\boldsymbol{s}(\xi)^{\mathsf{T}}\boldsymbol{z}\right\}\mathrm{d}\xi \\ &= \frac{\exp\left\{-\frac{A^2}{4\sigma_n^2}\right\}}{2\pi}\int_0^{2\pi}\exp\left\{\frac{A}{\sigma_n^2}\sum_j x(j)s(j,\xi)\right\}\mathrm{d}\xi\end{aligned}$$

Rewrite $\sum_j x(j)s(j,\xi)$ as

$$\begin{aligned}\sum_j x(j)s(j,\xi) &= \sum_j x(j)D\left[\tau_j - t_d(\tau_j)\right]C\left[\tau_j - t_s(\tau_j)\right]\cos\left[2\pi f_{\text{IF}}\tau_j + \xi\right] \\ &= \sum_j x(j)D\left[\tau_j - t_d(\tau_j)\right]C\left[\tau_j - t_s(\tau_j)\right]\left\{\cos(2\pi f_{\text{IF}}\tau_j)\cos(\xi) - \sin(2\pi f_{\text{IF}}\tau_j)\sin(\xi)\right\}\end{aligned}$$

Distributing the sum

$$\begin{aligned}\sum_j x(j)s(j,\xi) = &\underbrace{\sum_j x(j)D\left[\tau_j - t_d(\tau_j)\right]C\left[\tau_j - t_s(\tau_j)\right]\cos(2\pi f_{\text{IF}}\tau_j)}_{s_1}\cos(\xi) \\ &\underbrace{-\sum_j x(j)D\left[\tau_j - t_d(\tau_j)\right]C\left[\tau_j - t_s(\tau_j)\right]\sin(2\pi f_{\text{IF}}\tau_j)}_{-s_2}\sin(\xi)\end{aligned}$$

Thus, we have that

$$\begin{aligned}\Lambda(\boldsymbol{z}) &= \frac{\exp\left\{-\frac{A^2}{4\sigma_n^2}\right\}}{2\pi}\int_0^{2\pi}\exp\left\{\frac{A}{\sigma_n^2}(s_1\cos(\xi) + s_2\sin(\xi))\right\}\mathrm{d}\xi \\ \Lambda(\boldsymbol{z}) &= \exp\left\{-\frac{A^2}{4\sigma_n^2}\right\}\cdot\underbrace{\frac{1}{2\pi}\int_0^{2\pi}\exp\left\{\frac{A}{\sigma_n^2}s_1\cos(\xi) + \frac{A}{\sigma_n^2}s_2\sin(\xi)\right\}\mathrm{d}\xi}_{I_0\left(\frac{A}{\sigma_n^2}\sqrt{s_1^2+s_2^2}\right)}\end{aligned}$$

The hypothesis test has simplified to

$$\Lambda'(\boldsymbol{z}) = I_0\left(\frac{A}{\sigma_n^2}\sqrt{s_1^2+s_2^2}\right)\underset{H_0}{\overset{H_1}{\gtrless}}\nu'$$

But since $I_0(x)$ is monotonically increasing in x, the test is equivalent to

$$\Lambda''(\boldsymbol{z}) = \sqrt{s_1^2+s_2^2}\underset{H_0}{\overset{H_1}{\gtrless}}\nu''$$

The hypothesis test is non-trivial only when the right-hand side is greater than zero. In this case, it is safe to square both sides while maintaining the inequality

$$\Lambda^*(\boldsymbol{z}) = s_1^2 + s_2^2\underset{H_0}{\overset{H_1}{\gtrless}}\nu^*$$

where

$$s_1^2 = \left[\sum_j x(j) \underbrace{D\left[\tau_j - t_d(\tau_j)\right] C\left[\tau_j - t_s(\tau_j)\right] \cos(2\pi f_{\mathrm{IF}} \tau_j)}_{s_c(j)} \right]^2$$

$$s_2^2 = \left[\sum_j x(j) \underbrace{D\left[\tau_j - t_d(\tau_j)\right] C\left[\tau_j - t_s(\tau_j)\right] \sin(2\pi f_{\mathrm{IF}} \tau_j)}_{s_s(j)} \right]^2$$

It is clear that the detection statistic implies a quadrature correlation architecture wherein the incoming signal is correlated against both $s_c(j)$ and $s_s(j)$. Note that the value of A is not needed to form the detection statistic.

As a part of the struct `s`, we are given the number of code offsets n_{code}, and that $n_{\text{freq}} = 2f_{\max}T_a$. From these we have that

$$n_{\text{cells}} = 2n_{\text{code}}f_{\max}T_a$$

Next, we must evaluate ρ_k. Recall that

$$\frac{C}{N_0} = \frac{N_k^2\bar{A}_k^2}{4N_k^2\sigma_n^2T}$$

where N_k is the number of samples in kth accumulation, T is the sampling interval, $\bar{A}_k$ is the average amplitude over the kth accumulation, and σ_n^2 is the noise variance. Also, recall that

$$\sigma_{IQ}^2 = \frac{N_k\sigma_n^2}{2}$$
$$T_a = N_kT$$
$$\rho_k = \frac{\bar{A}_kN_k}{2\sigma_{IQ}}$$

From these equations, we get that

$$\frac{C}{N_0} = \frac{\rho_k^2}{2T_a}$$
$$\Rightarrow \rho_k = \sqrt{\frac{C}{N_0} \times 2T_a}$$

As mentioned in lecture,

$$p(Z|\theta_0) = \chi_{2N}^2$$
$$p(Z|\theta_1) = \chi_{2N}^2(\lambda)$$
$$\lambda = N\rho_k^2$$

The total acquisition false alarm probability is

$$P_F = 1 - [P(Z < \lambda_0|\theta_0)]^{n_{\text{cells}}}$$

Given the values of P_F and n_{cells}, we derive λ_0 as

$$\lambda_0 = \texttt{chi2inv}\left((1-P_F)^{1/n_{\text{cells}}}, N\right)$$

The probability of detection is then computed as

$$P_d = P(Z > \lambda_0|\theta_1)$$
$$= 1 - \texttt{ncx2cdf}\left(\lambda_0, N, N\rho_k^2\right)$$

Figure 1 shows an example output of the `performAcqHypothesisCalcs` function with $P_F = 0.0001$, 1 ms coherent integration and no noncoherent summations. For C/N_0 of 43 dB-Hz, this threshold gives $P_F = 0.50411$.

Figure 2 shows the N required to achieve a probability of detection $P_D = 0.95$ or more for each of the C/N_0 values in the set 27, 30, 33, 36, 39, 42, 45, 48, 51.

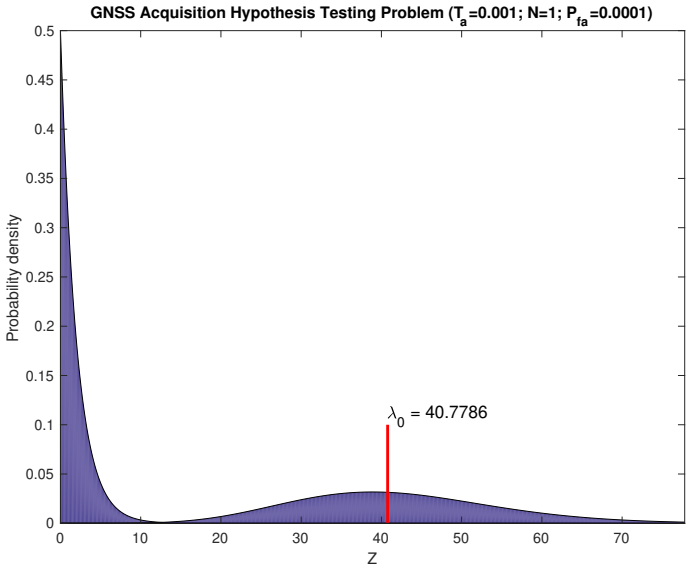

FIGURE 1. Example of acquisition detection threshold with $P_F = 0.0001$, 1 ms coherent integration and no noncoherent summations. The probability of detection, P_F, is 0.50411 for C/N_0 of 43 dB-Hz.

Figure 3 shows the T_a required to achieve a probability of detection $P_D = 0.95$ or more for each of the C/N_0 values in the set 5, 10, 15, ..., 45, 50 without any noncoherent summations.

The comparison in Figures 2 and 3 shows the clear superiority of coherent integration per sample. For example, noncoherent integration requires about 0.1 seconds of data to acquire a 30 dB-Hz signal with the required P_F and P_D. For the same C/N_0, coherent integration requires only 0.037 seconds of data to meet the required probabilities.

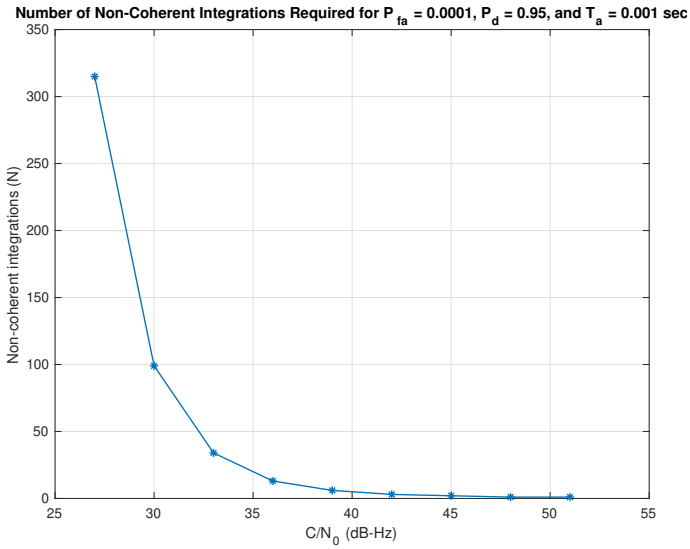

FIGURE 2. Number of non-coherent integrations required for $P_F = 0.0001$, $P_d = 0.95$, and $T_a = 0.001$ seconds.

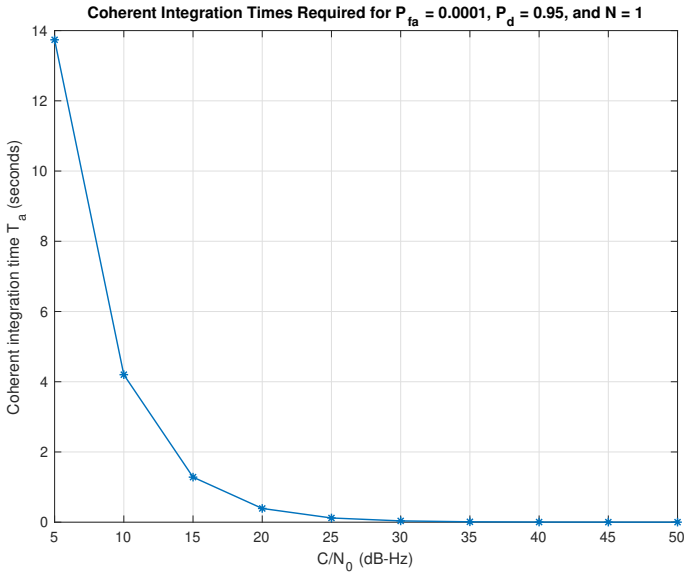

FIGURE 3. Coherent integration times required for $P_F = 0.0001$, $P_d = 0.95$, and $N = 1$ non-coherent integrations.

`topPerformAcqHypothesisCalcs.m`

```
% topPerformAcqHypothesisCalcs
%
% Top-level script for performing acquisition calculations

%----- Setup
clear; clc; close all; format long g;
s.PfaAcq = 0.0001;
s.Ta = 0.001;
s.fMax = 7000;
s.nCodeOffsets = 1023*5;
s.ZMax = 1000;
s.delZ = 0.1;

%----- Part (a)
C_N0 = 27 : 3 : 51;
Nreq = [];
for ii = C_N0
    Pd = 0;
    s.N = 1;
    disp(['C/N0 = ' num2str(ii)]);
    s.C_N0dBHz = ii;
    while Pd < 0.95
        [~,~,~,Pd,~] = performAcqHypothesisCalcs(s);
        s.N = s.N + 1;
    end
    Nreq = [Nreq; s.N-1];
end
```

```
plot(C_N0, Nreq, '*-', 'LineWidth', 1);
grid on;
xlabel('C/N_0 (dB-Hz)');
ylabel('Non-coherent integrations (N)');
title(['Number of Non-Coherent Integrations Required for P_{fa} = 0.0001, ' ...
                                        'P_{d} = 0.95, and T_a = 0.001 sec']);
print('../figs/Nreq', '-depsc');

%----- Part (b)
s.N = 1;
C_N0 = 5 : 5 : 50;
Tareq = [];
for ii = C_N0
Pd = 0;
    disp(['C/N0 = ' num2str(ii)]);
    s.C_N0dBHz = ii;
    while Pd < 0.95
        [~,~,~,Pd,~] = performAcqHypothesisCalcs(s);
        s.Ta = s.Ta + 0.001;
    end
    Tareq = [Tareq; s.Ta - 0.001];
    s.Ta = 0.001;
end

figure;
plot(C_N0, Tareq, '*-', 'LineWidth', 1);
grid on;
xlabel('C/N_0 (dB-Hz)');
ylabel('Coherent integration time T_a (seconds)');
title('Coherent Integration Times Required for P_{fa} = 0.0001, P_{d} = 0.95, and N = 1');
print('../figs/Tareq', '-depsc');

%----- Visualize an example
s.C_N0dBHz = 43;
s.N = 1;
s.PfaAcq = 0.0001;
s.Ta = 0.001;
s.fMax = 7000;
s.nCodeOffsets = 1023*5;
s.ZMax = 1000;
s.delZ = 0.1;

[pZ_H0,pZ_H1,lambda0,Pd,ZVec] = performAcqHypothesisCalcs(s);

figure(3);
[pmax,iimax] = max(pZ_H1);
Zmax = ZVec(iimax);
clf;
ash = area(ZVec,pZ_H0);
set(get(ash,'children'), 'facecolor', 'g', 'linewidth', 2, 'facealpha', 0.5);
hold on;
```

```
ash = area(ZVec,pZ_H1);
set(get(ash,'children'), 'facecolor', 'b', 'linewidth', 2, 'facealpha', 0.5);
linemax = 1/5*max([pZ_H0;pZ_H1]);
line([lambda0,lambda0],[0,linemax], 'linewidth', 2, 'color', 'r');
xlim([0 max(Zmax*2,lambda0*1.5)]);
ylabel('Probability density');
xlabel('Z');
fs = 12;
title('GNSS Acquisition Hypothesis Testing Problem (T_a=0.001; N=1; P_{fa}=0.0001)');
disp(['Probability of acquisition false alarm (PfaAcq): ' ...
      num2str(s.PfaAcq)]);
disp(['Probability of detection (Pd): ' num2str(Pd)]);
text(lambda0,linemax*1.1, ['\lambda_0 = ' num2str(lambda0) ], ...
     'fontsize',fs);
print('../figs/acqHypothesis', '-depsc')
```

performAcqHypothesisCalcs.m

```
function [pZ_H0,pZ_H1,lambda0,Pd,ZVec] = performAcqHypothesisCalcs(s)
% performAcqHypothesisCalcs : Calculate the null-hypothesis and alternative
%                             hypothesis probability density functions and the
%                             decision threshold corresponding to GNSS signal
%                             acquisition with the given inputs.
%
% Z is the acquisition statistic:
%
%          N  |    |^2
% Z =     sum | Sk |
%         k=1 |    |
%
%
%               -           -
%          N  |               |
%   =     sum | Ik^2 + Qk^2   |
%         k=1 |_            _|
%
%
%
% where Sk = rhok + nk
%          = Ik + j*Qk
%
% and   nk = nIk + j*nQk
%
% with nIk ~ N(0,1), nQk ~ N(0,1), and E[nIk nQi] = 1 for k = i and 0 for k !=
% i. The amplitude rhok is related to familiar parameters Nk, Abark, and
% sigma_IQ by rhok = (Nk*Abark)/(2*sigma_IQ), i.e., it is the magnitude of the
% usual complex baseband phasor normalized by sigma_IQ.
%
% Under H0, the statistic Z is distributed as a chi square distribution with
% 2*N degrees of freedom; under H1, it is distributed as a noncentral chi
```

```
% square distribution with lambda = N*rhok^2 and 2*N degrees of freedom.
%
% The total number of cells in the search grid is assumed to be nCells =
% nCodeOffsets*nFreqOffsets, where nFreqOffsets = 2*fMax*Ta and Ta = Na*T is
% the total coherent accumulation time. Here, Na is the average value of the
% number of samples in each accumulation, Nk.
%
% INPUTS
%
% s -------- A structure containing the following fields:
%
%           C_N0dBHz ------- Carrier to noise ratio in dB-Hz.
%
%           Ta ------------- Coherent accumulation interval, in seconds.
%
%
%           N -------------- The number of accumulations summed noncoherently to
%                            get Z.
%
%           fMax ----------- Frequency search range delimiter. The total
%                            frequency search range is +/- fMax.
%
%           nCodeOffsets --- Number of statistically independent code offsets in
%                            the search range.
%
%           PfaAcq --------- The total acquisition false alarm probability.
%                            This is the probability that the statistic Z
%                            exceeds the threshold lambda in any one of the
%                            search cells under the hypothesis H0. One can
%                            derive the false alarm probability for each search
%                            cell from PfaAcq.
%
%           ZMax ----------- The maximum value of Z that will be considered.
%
%           delZ ----------- The discretization interval used for the
%                            independent variable Z. The full vector of Z
%                            values considered is thus ZVec = [0:delZ:ZMax].
%
%
% OUTPUTS
%
% pZ_H0 ---------- The probability density of Z under hypothesis H0.
%
% pZ_H1 ---------- The probability density of Z under hypothesis H1.
%
% lambda0 -------- The detection threshold.
%
% Pd ------------- The probability of detection.
%
% Zvec ----------- The vector of Z values considered.
%
%+-------------------------------------------------------------------------+
```

```
% References:
%
%
%+========================================================================+

ZVec = 0 : s.delZ : s.ZMax;

nFreqOffsets = 2 * s.fMax * s.Ta;
nCells = s.nCodeOffsets * nFreqOffsets;

rho_k = sqrt( 10^(s.C_N0dBHz/10) * 2 * s.Ta );

pZ_H0 = chi2pdf(ZVec, 2*s.N)';
pZ_H1 = ncx2pdf(ZVec, 2*s.N, s.N*(rho_k^2))';

lambda0 = chi2inv((1 - s.PfaAcq)^(1/nCells), 2*s.N);

Pd = 1 - ncx2cdf(lambda0, 2*s.N, s.N*(rho_k^2))';
```