Contents


```
clear all;
close all;
clc;
```

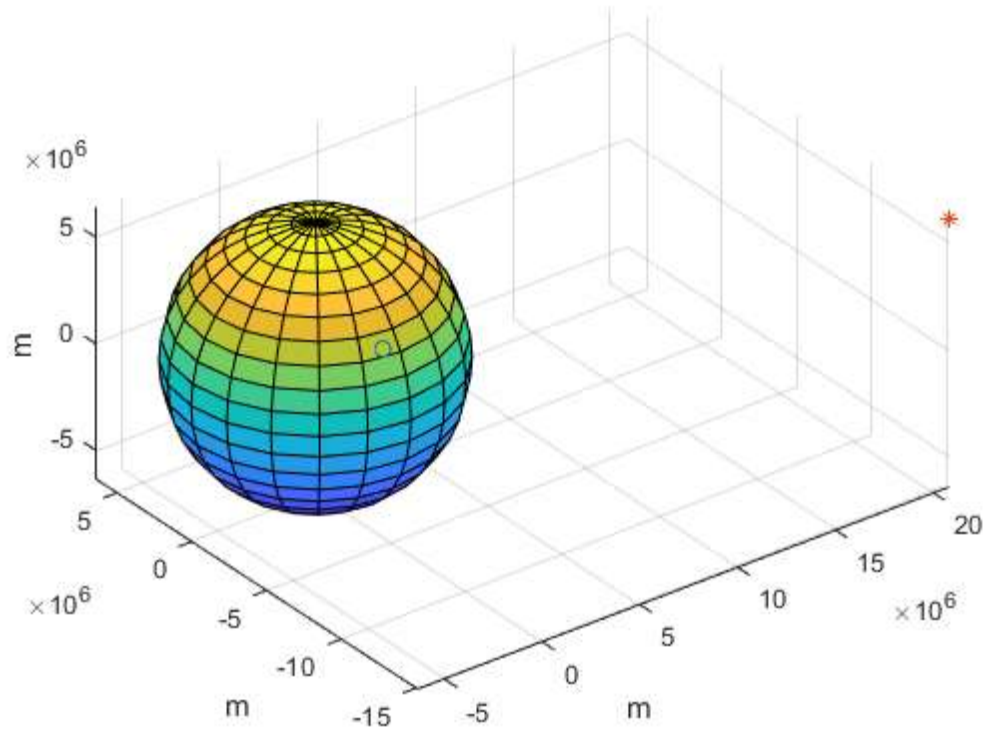
Get Data

```
load("channel.mat")
M= channel';  % transpose to get in column
```

Ionospheric Delay calc from a model

```
model = 'broadcast';
ionodata.broadcast.alpha0 = 1.1176e-008;
ionodata.broadcast.alpha1 = 7.4506e-009;
ionodata.broadcast.alpha2 = -5.9605e-008;
ionodata.broadcast.alpha3 = -6.9605e-008;
ionodata.broadcast.beta0  = 90112;
ionodata.broadcast.beta1  = 0;
ionodata.broadcast.beta2  = -296610;
ionodata.broadcast.beta3  = -75536;
tGPS.week    = 1575;
tGPS.seconds = 518201.501;
 rRx = [-742005.851560607;-5462223.38476596; 3198008.7346792];

% TXID
 rSv = [20847329.7083373;-15185642.4780402; 6205281.68907901];
[delTauG] = getIonoDelay(ionodata,0,rRx,rSv,tGPS,model);
delTaugG  = round (delTauG,4,'significant');
disp(['--------------------Answer----------------------------------------'])
disp([' delTauG from getIonoDelay: ',num2str(delTauG), ' s'])
% find index of I_L1_p_29 where time equals tGPS.seconds.Note: the seconds
% do not exactly match each other, so I take only the integer value of
% GPS.seconds to compare with 4th column of L1_TXID
```

```
--------------------Answer----------------------------------------
 delTauG from getIonoDelay: 1.1388e-08 s
```



Published with MATLAB® R2023a

```
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 delay
%
% 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.
%
%+==============================================================================+
figure(),
```

```
[X,Y,Z]=sphere;
Re= 6378137 ; %Earth Radius in meters
surf(X*Re,Y*Re,Z*Re);
xlabel('m');
ylabel('m');
zlabel('m');
axis equal,
hold on,

plot3(rRx(1),rRx(2),rRx(3),'o');
hold on,
plot3(rSv(1),rSv(2),rSv(3),'*')


% A is left as rad instead of semi-circle unit since they are only used for trig calculations
[E, A, r_lla, s_lla]=findElevationAzimuthAngleANDLLA(rRx,rSv);
E = E/pi; % semi-circle
phi_u    = r_lla(1)/pi; % semi-circles
lambda_u = r_lla(2)/pi; % semi-circles

alpha    = [ionodata.broadcast.alpha0,ionodata.broadcast.alpha1,ionodata.broadcast.alpha2,ionodata.broadcast.alpha3];
beta     = [ionodata.broadcast.beta0,ionodata.broadcast.beta1,ionodata.broadcast.beta2,ionodata.broadcast.beta3];

psi   = 0.0137/(E+0.11)-0.22; % semi-circle
phi_i = phi_u+psi*cos(A); % semi-circle

if phi_i > 0.416
    phi_i = 0.416;
elseif phi_i < -0.416
    phi_i = -0.416;
end

lambda_i = lambda_u + psi*sin(A)/cos(phi_i*pi); % semi-circle
t        = 4.32*10^4*lambda_i +tGPS.seconds;

if t >= 86400
    t = t-86400;
elseif t<= -86000
    t = t+86400;
end

phi_m    = phi_i + 0.064*cos((lambda_i-1.617)*pi); % Multiply by pi to remove semicircle unit
F        = 1 + 16*(0.53 -E)^3;
PER      = sum(beta.*phi_m);

if PER < 72000
    PER = 72000;
end

x      = 2*pi*(t-50400)/PER;
AMP    = sum(alpha.*phi_m);

if AMP < 0
    AMP = 0;
end

if abs(x) < 1.57
    delTauG = F*(5*10^-9+AMP*(1-x^2/2+x^4/24)); % sec
elseif abs(x) >= 1.57
    delTauG = F*5*10^-9; % sec
end
```

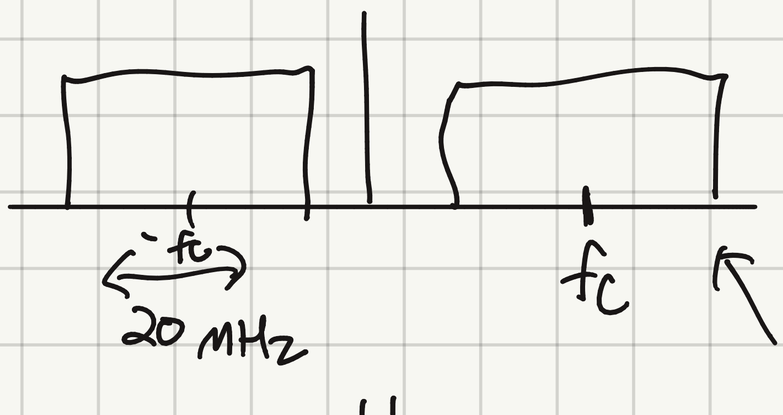
```
Not enough input arguments.

Error in getIonoDelay (line 71)
plot3(rRx(1),rRx(2),rRx(3),'o');
```

Published with MATLAB® R2023a

Exam 2 Problem 2

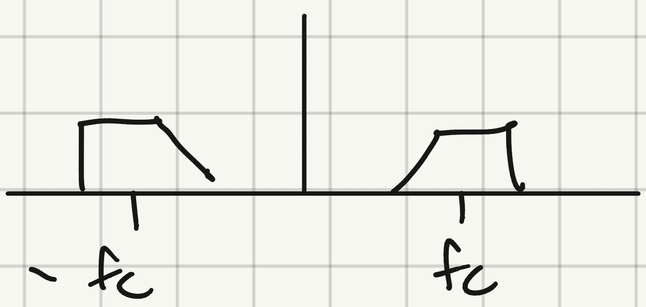
Original signal $f_c = L_1 = 1575.42\,MHz$
after L_1 and L_2 diivder

X

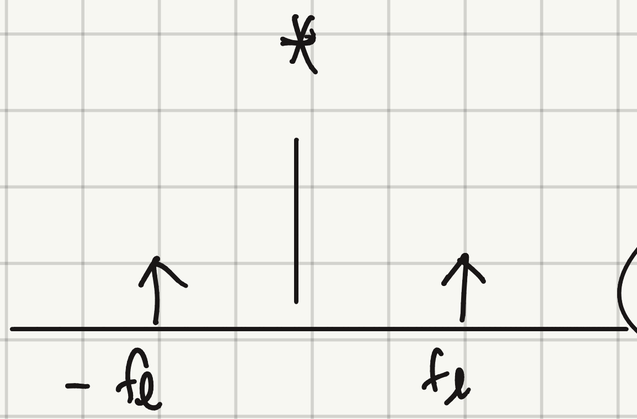


L_1 RF SAW filter

much bigger than Signal bandwidth



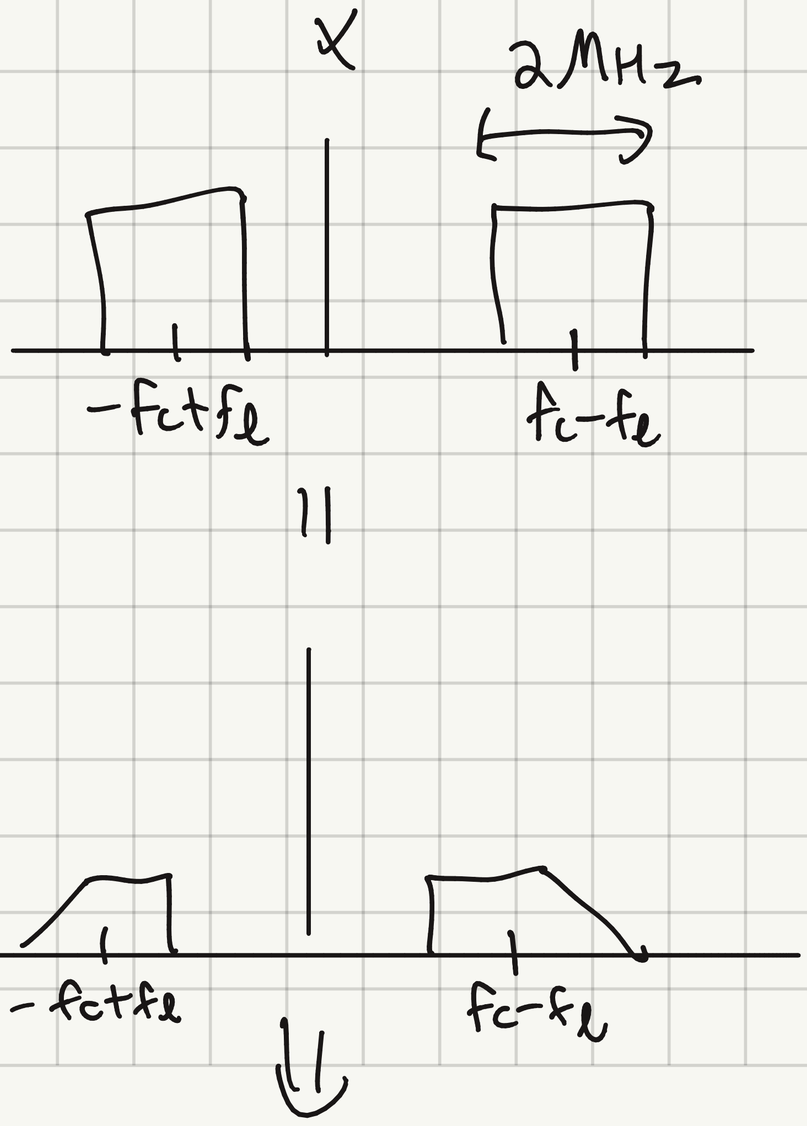
*



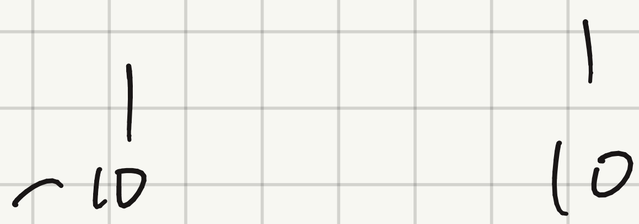
$f_\ell = 1276.\overline{6}\,MHz$
low-side mixy

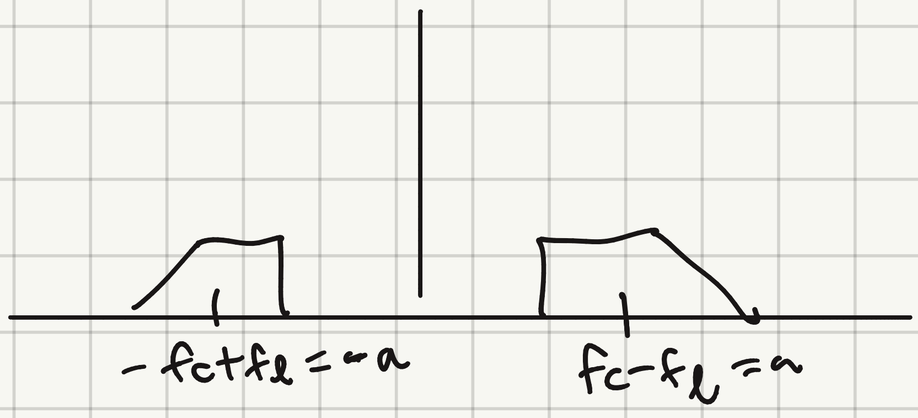
$f_c - f_\ell = 298.75\overline{3}$

X



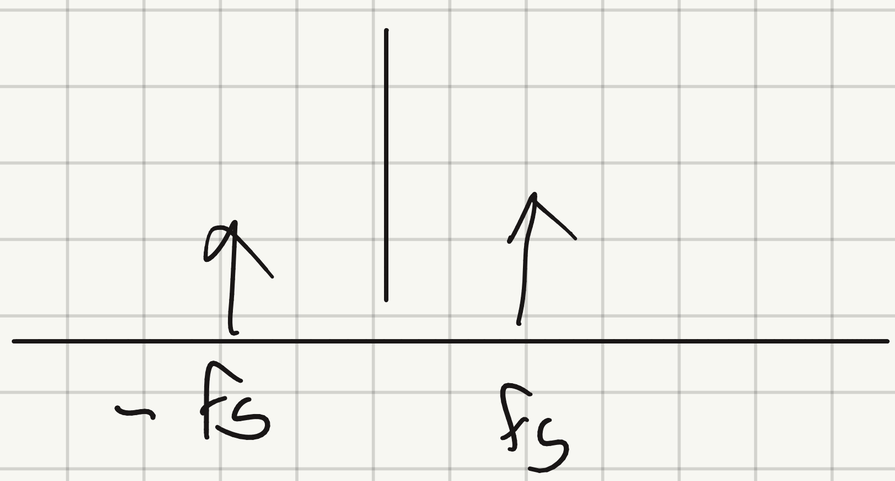
L_1 IF SAW filter





*

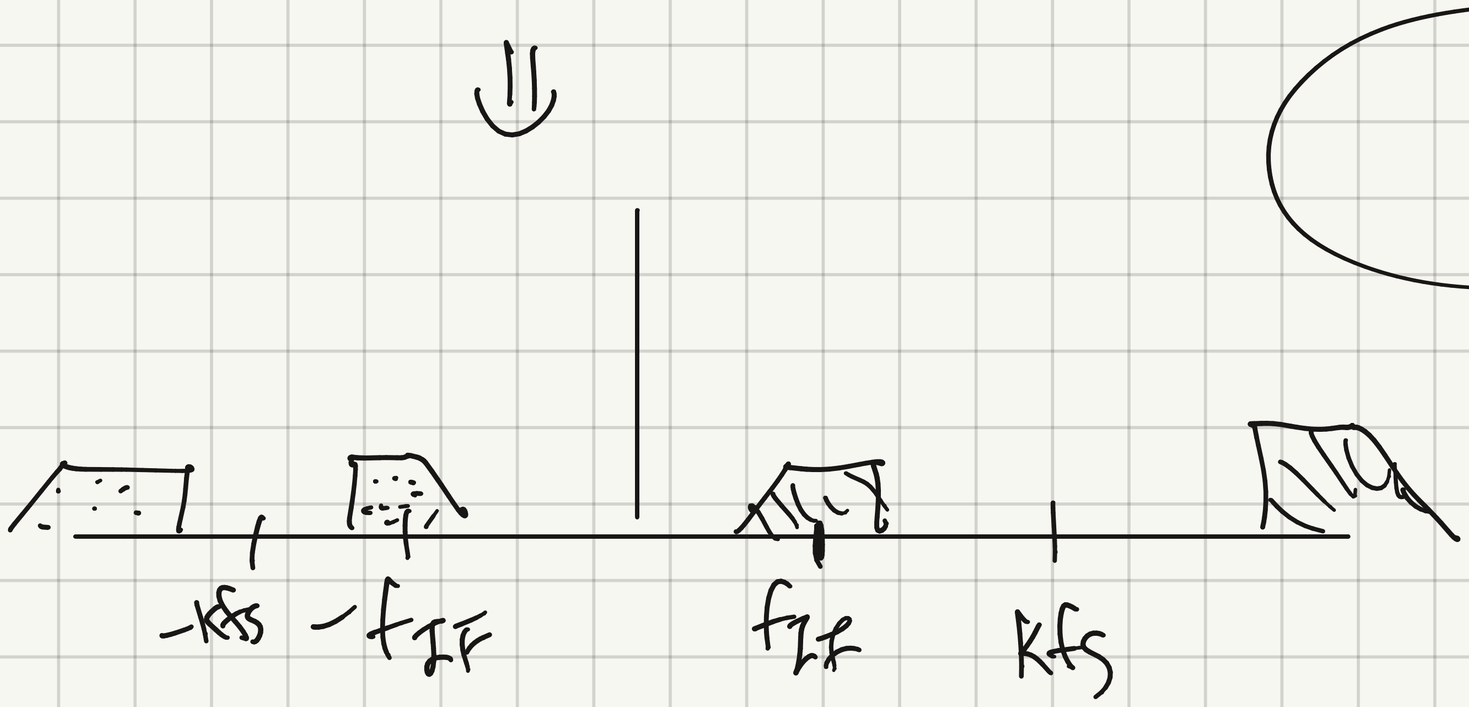
Sample at $\frac{40}{7}$ MHz.



$$f_{IF} = \left| a - f_s \cdot \left\lfloor \frac{a}{f_s} \right\rceil \right|$$

$$= \left| a - \frac{40}{7} \left\lfloor \frac{a}{\frac{40}{7}} \right\rceil \right|$$

$$\simeq 1.610476 \text{ MHz}$$



Contents


```
clear; close all; clc;
```

```
%----- Setup
Tfull   = 0.5;           % Time interval of data to load
fsampIQ = 10.0e6;         % IQ sampling frequency (Hz)
fIF     = 5e6;          % Intermediate frequency (Hz)

N = floor(fsampIQ*Tfull);
nfft = 2^9;             % Size of FFT used in power spectrum estimation
%----- Load data
fid = fopen("C:\Users\gsh04\Desktop\2024-Fall\GPS\exam2\problem 3\niData03head_10MHz.bin",'r','l');
Y = fread(fid, [2,N], 'int16')';
Y = Y(:,1) + 1j*Y(:,2);
fclose(fid);
```

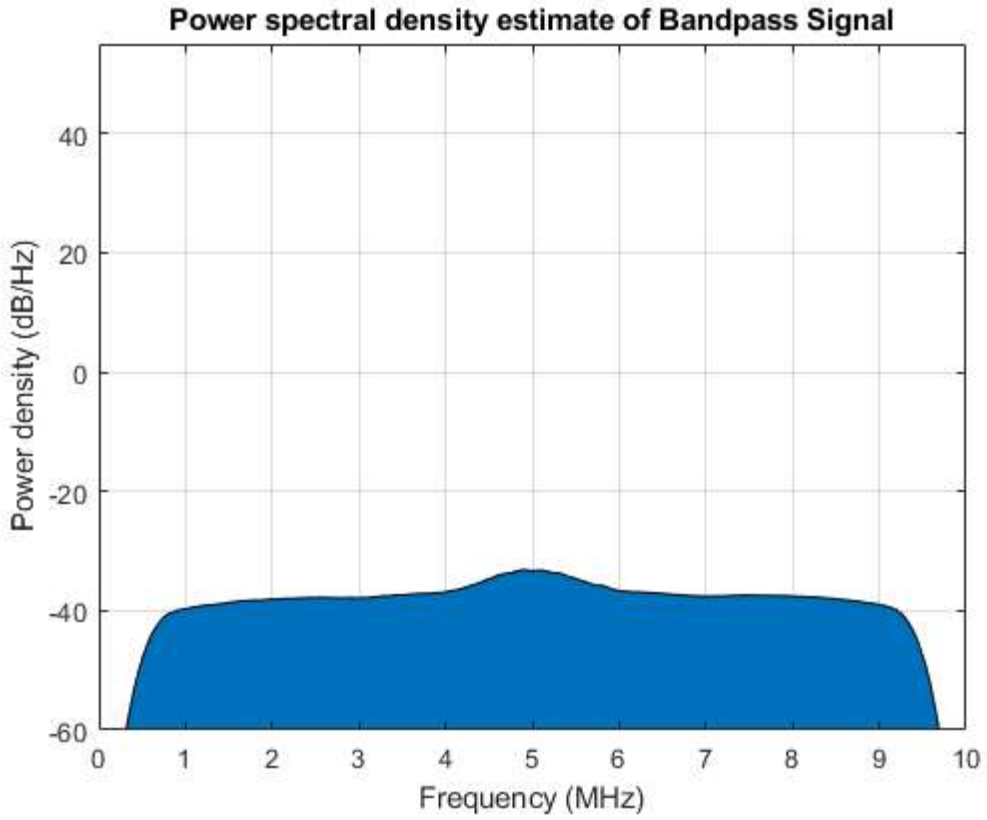
Pwelch from signal

```
%----- Compute power spectrum estimate
[Syy,fVec] = pwelch(Y,hann(nfft),[],nfft,fsampIQ);
%----- Plot results
% figure,
% yLow = -60;
% yHigh = 50;
% T = nfft/fsampIQ;
% delf = 1/T;
% fcenter = (nfft/2)*delf;
% fVec = fVec - fcenter;
% Syy = [Syy(nfft/2 + 1 : end); Syy(1:nfft/2)];
% area(fVec/1e6,10*log10(Syy),yLow);
% ylim([yLow,yHigh]);
% grid on;
% shg;
% xlabel('Frequency (MHz)');
% ylabel('Power density (dB/Hz)');
% title('Power spectral density estimate from complex data');
% shg;
```

Convert to Bandpass Signal

```
Tl = 1/fsampIQ;
[xVec] = iq2if(real(Y),imag(Y),Tl,fIF);
[Syy2,fVec2] = pwelch(xVec,hann(nfft),[],nfft,2*fsampIQ);
%----- Plot results
figure,
yLow2 = -60;
```

```
yHigh2 = 55;
area(fVec2/1e6,10*log10(Syy2),yLow2);
ylim([yLow2,yHigh2]);
grid on;
shg;
xlabel('Frequency (MHz)');
ylabel('Power density (dB/Hz)');
title('Power spectral density estimate of Bandpass Signal');
shg;
```

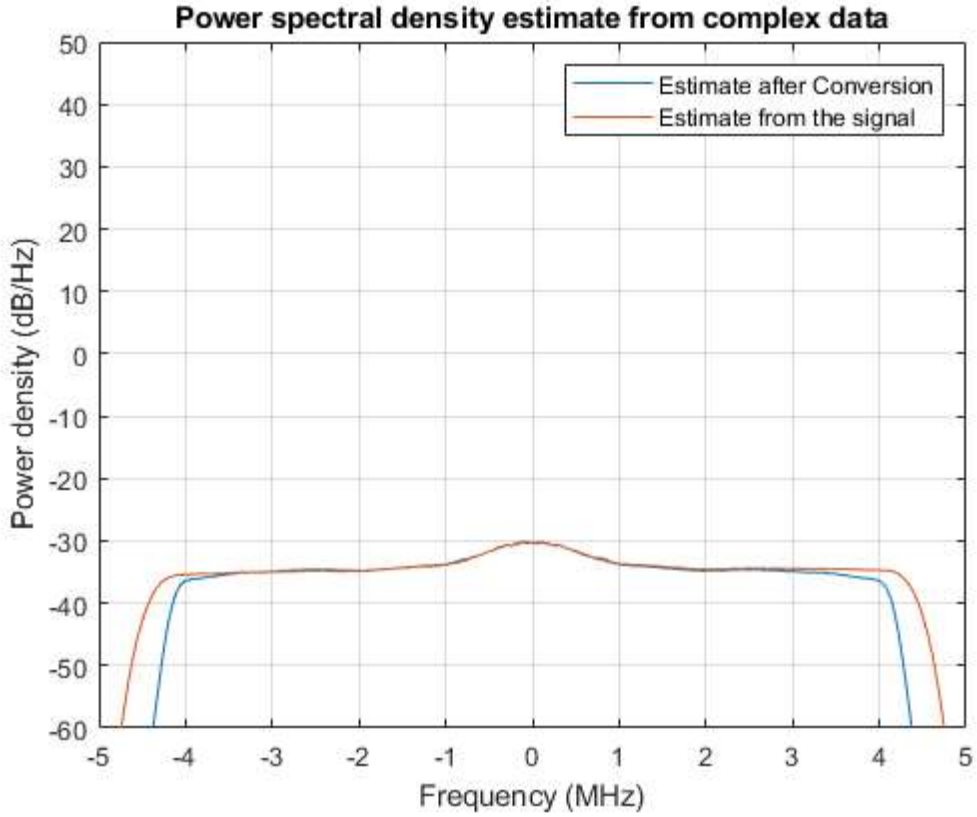


Convert back to baseband signal

```
T  = Tl/2;
[IVec,QVec] = if2iq(xVec,T,fIF);
%----- Compute power spectrum estimate
Y_recovered = IVec +1j+QVec;
[Syy3,fVec3] = pwelch(Y_recovered,hann(nfft),[],nfft,fsampIQ);
%----- Plot Comparative results
figure,
yLow3 = -60;
yHigh3 = 50;
T = nfft/fsampIQ;
delf = 1/T;
fcenter = (nfft/2)*delf;
fVec3 = fVec3 - fcenter;
Syy3 = [Syy3(nfft/2 + 1 : end); Syy3(1:nfft/2)];
plot(fVec3/1e6,10*log10(Syy3));
ylim([yLow3,yHigh3]);
grid on;
shg;
xlabel('Frequency (MHz)');
```

```
ylabel('Power density (dB/Hz)');

hold on,

yLow = -60;
yHigh = 50;
T = nfft/fsampIQ;
delf = 1/T;
fcenter = (nfft/2)*delf;
fVec = fVec - fcenter;
Syy = [Syy(nfft/2 + 1 : end); Syy(1:nfft/2)];
plot(fVec/1e6,10*log10(Syy));
ylim([yLow,yHigh]);
grid on;
shg;
xlabel('Frequency (MHz)');
ylabel('Power density (dB/Hz)');
title('Power spectral density estimate from complex data');
shg;
legend('Estimate after Conversion', 'Estimate from the signal')

fprintf(['After converting back to baseband with the if2iq function, the higher \n'...
    'frequecny signals are lost. This is due to low pass filtering that happens \n' ...
    'within the MATLAB decimate function when we convert the bandpass signal back \n'...
    'to baseband signal. Also, the output of the if2iq was scaled by 2 because \n' ...
    'the discrete bandpass signal after mxing and going through the low pass \n'...
    'filter has the half the magnitude of the original signal. \n'])
```

```
After converting back to baseband with the if2iq function, the higher
frequecny signals are lost. This is due to low pass filtering that happens
within the MATLAB decimate function when we convert the bandpass signal back
to baseband signal. Also, the output of the if2iq was scaled by 2 because
the discrete bandpass signal after mxing and going through the low pass
filter has the half the magnitude of the original signal.
```



Published with MATLAB® R2023a

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

Tl        = 2*T; % Quadrature sampling interval
r         = Tl/T; % Decimation factor
n         = (0:length(xVec)-1)';
InPhaComp = xVec.*2.*cos(2*pi*fIF*n*T); % In-phase component or I; Unfiltered;
QuadComp  = xVec.*2.*sin(2*pi*fIF*n*T); % Quadrature component or Q; Unfiltered;

% By default, decimate uses a lowpass Chebyshev Type I infinite
% impulse response (IIR) filter of order 8.
IVec = decimate(InPhaComp,r);
QVec = decimate(QuadComp,r);
```

```
Not enough input arguments.

Error in if2iq (line 36)
Tl        = 2*T; % Quadrature sampling interval
```

Published with MATLAB® R2023a

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

R  = 2; % for iq2if. (Given)
IVecResampled = interp(IVec,R);
QVecResampled = interp(QVec,R);
T=Tl/2;
n=(0:length(IVecResampled)-1)';
xVec = IVecResampled.*cos(2*pi*fIF*n*T)-QVecResampled.*sin(2*pi*fIF*n*T);
```

```
Not enough input arguments.

Error in iq2if (line 37)
IVecResampled = interp(IVec,R);
```

Published with MATLAB® R2023a

(b) $K_{max} = \left\lfloor \frac{F_H}{B} \right\rfloor$

$F_H = 1575.42 + \frac{4}{2} = 1577.42\ MHz$

$F_L = 1575.42 - \frac{4}{2} = 1573.42\ MHz$

$= \left\lfloor \frac{157.42}{4} \right\rfloor$

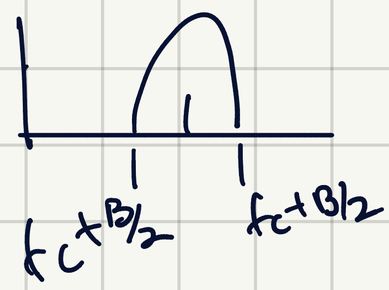
$= \lfloor 394.3550 \rfloor$

$= 394$

$f_{s,min} = \frac{2F_H}{K} = \frac{2(157.42)}{394} = 8.0072\ MHz$

$f_s = f_{s,min} / 0.8 = 10.0090\ MHz$

$\frac{f_s}{B} = \frac{10.0090}{4} = 2.5023$



Based on how the spikey hair plot looks like the wedge area near $K = 394$ will be super narrow, so even a slight variation will put us into the shaded reason and cause aliasing. You can't blindly increase sampling rate.

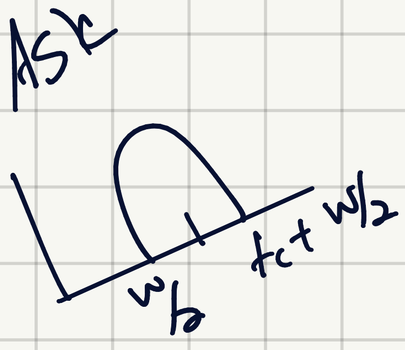
If $W = B/0.8$

$= 4/0.8 = 5\ MHz$

$f_H = mW \; ; \; m = 316$

$= 316(5) = 1580\ MHz$

$f_s = 2W = 10\ MHz$



$K_{max} = \left\lfloor \frac{1580}{5} \right\rfloor$

$= 316$

$f_{s,min} = \frac{2(1580)}{316} = 10\ MHz$

↑ this freqny range is right on one of the tips of a wedge, so it is allowed but even a slight variation will move the sampling freqny into the shaded area of the spikey hair plot and cause aliasing.

This is not enough to avoid aliasing because variation of sampling rate or carrier freqny could occur.

Implement GuardBands

Let's add two guard bands of $\Delta B_L = 0.5\ MHz$
$\Delta B_H = 0.5 MHz$

$$B = 4 MHz$$

$$B' = B + \Delta B_L + \Delta B_H$$
$$= 4 + 1 = 5$$

$$F_L' = F_L - \Delta B_L = 1573.42 - 0.5 = 1572.92\ MHz$$

$$F_H' = F_H + \Delta B_H = 1577.42 + 0.5 = 1577.92\ MHz$$

$$k'_{max} = \left\lfloor \frac{F_H'}{B'} \right\rfloor = \left\lfloor \frac{1577.92}{5} \right\rfloor = 315$$

$$\frac{2F_H'}{k'} \leq F_s \leq \frac{2F_L'}{k'-1}$$

$$\frac{2(1577.92)}{315} \leq F_s \leq \frac{2(1572.92)}{314}$$

$$(10.0185397 \leq F_s \leq 10.0185987)\ MHz$$

Contents

Load Data from rawtrimmed_158.bin

```
clear; close all; clc;
```

Get Signal

```
%----- Setup
Tfull = 0.5;         % Time interval of data to load
fs = 40e6/7;         % Sampling frequency (Hz)
N = fs*Tfull;
N = floor(N/16)*16;  % Number of data samples to load
nfft = 2^10;          % Size of FFT used in power spectrum estimation
fIF     = 1.610476e6;        % Intermediate frequency (Hz)

%----- Load data
fid = fopen("C:\Users\gsh04\Desktop\2024-Fall\GPS\exam2\problem 5\dataout_raw_trimmed_158.bin",'r','l');
[Y,count] = binloadSamples(fid,N,'dual');
Y = Y(:,1);
```

Genererate Code

```
%---- Generate all possible PRN (37 SVIDs or PRN Sign No.)
% LFSR Parameters:
nStages      = 10;
ciVec1       = [10, 3]';
ciVec2       = [10, 9, 8, 6, 3, 2,]';
a0Vec1       = ones(nStages,1);
a0Vec2       = ones(nStages,1);
% G2Delay      = [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];
% Oversampling Parameters:
Tc = 1e-3/1023;             % Chip interval in seconds
T  = 1/fs;                  % Bandpass Sampling time interval in seconds
delChip = T/Tc;             % Sampling interval in chips
Np = 2^nStages - 1;         % Period of the sequence in chips
Ns = length(Y);             % Number of Samples should equal to that of Y(signal)
Ta = 0.001;                 % Accumulation time in seconds
Nk = floor(Ta/T);           % Number of samples in one 1-ms accumulation
% Generate 37 Seqeuences and Oversample them:
codeOS = zeros(Nk,37);
G2tab = [2,6;3,7;4,8;5,9;1,9;2,10;1,8;2,9;3,10;2,3;3,4;5,6;6,7;7,8;...
    8,9;9,10;1,4;2,5;3,6;4,7;5,8;6,9;1,3;4,6;5,7;6,8;7,9;8,10;1,6;2,7;...
    3,8;4,9;5,10;4,10;1,7;2,8;4,10];
```

```
parfor j = 1:length(G2tab)
    [GoldSeq] = generateGoldLfsrSequenceCA(nStages,ciVec1,ciVec2,a0Vec1,...
        a0Vec2,G2tab(j,:));
    % Make code +1/-1 not +1/0
    GoldSeq = 2*GoldSeq - 1;
    % Oversample Code: It makes sense to oversample code, since the code
    % embedded within the signal is sampled at a higher rate than its chip
    % rate. Assuming that the code I generate is sampled at the chip rate,
    % oversampling my code I generated at the rate the signal is sampled
    % will allow my code to correlate with the code embedded in the signal
    GoldSeqOS = oversampleSpreadingCode(GoldSeq,delChip,0,Nk,Np);
    codeOS(:,j) = GoldSeqOS;
end
%

fD = [-300000:100:0];
tk = [0:Nk-1]'*T;
PF = 0.05;
sigmaIQ = 149;
threshold = 39.5;
CN0 = zeros(37,1);
for mm =1:37
    for kk = 1:length(fD)
        Cr = fft(codeOS(:,mm));
        fi = fD(kk) + fIF;
        xkTilde = Y(1:Nk).*exp(-1i*2*pi*fi*tk);
        XrTilde = fft(xkTilde);
        Zr = XrTilde.*(conj(Cr));
        zk = ifft(Zr);
        [maxValue,kmax] = max(abs(zk).^2);
        CN0(mm) =10*log10((maxValue-2*sigmaIQ^2)/(2*sigmaIQ^2*Ta));
        if CN0(mm)  > threshold
            signalStrenghth(mm)=CN0(mm);
            start_time(mm) = tk(kmax+1)*10^6;
            apparent_fD(mm) = fD(kk);
            disp('-----------------------------------------------------------')
            disp(['PRN :',num2str(mm)])
            disp(['Apparent Doppler Frequency: ', num2str(apparent_fD(mm)), ' Hz']);
            disp(['Approximate Start Time from first sample: ', num2str(start_time(mm)), ' microseconds']);
            disp (['C/N0: ', num2str(CN0(mm))])
            break;
        end
    end
end
```

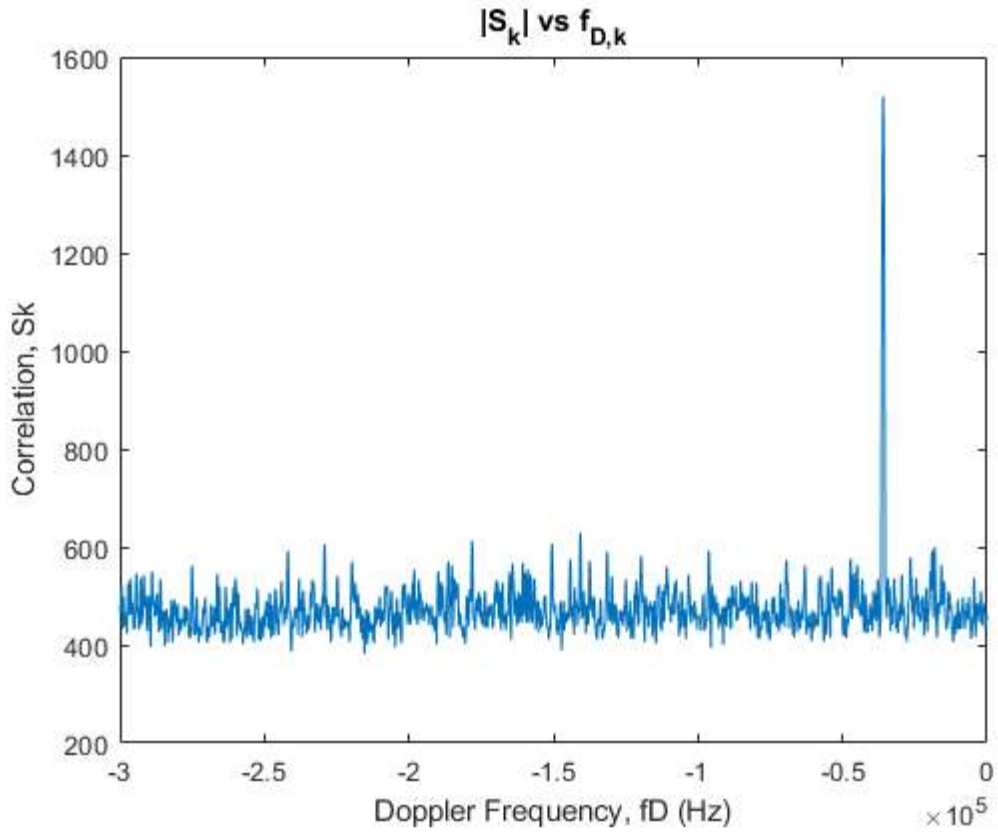
```
-----------------------------------------------------------
PRN :1
Apparent Doppler Frequency: -34300 Hz
Approximate Start Time from first sample: 411.075 microseconds
C/N0: 41.491
-----------------------------------------------------------
PRN :11
Apparent Doppler Frequency: -36500 Hz
Approximate Start Time from first sample: 578.55 microseconds
C/N0: 41.8547
-----------------------------------------------------------
PRN :18
```

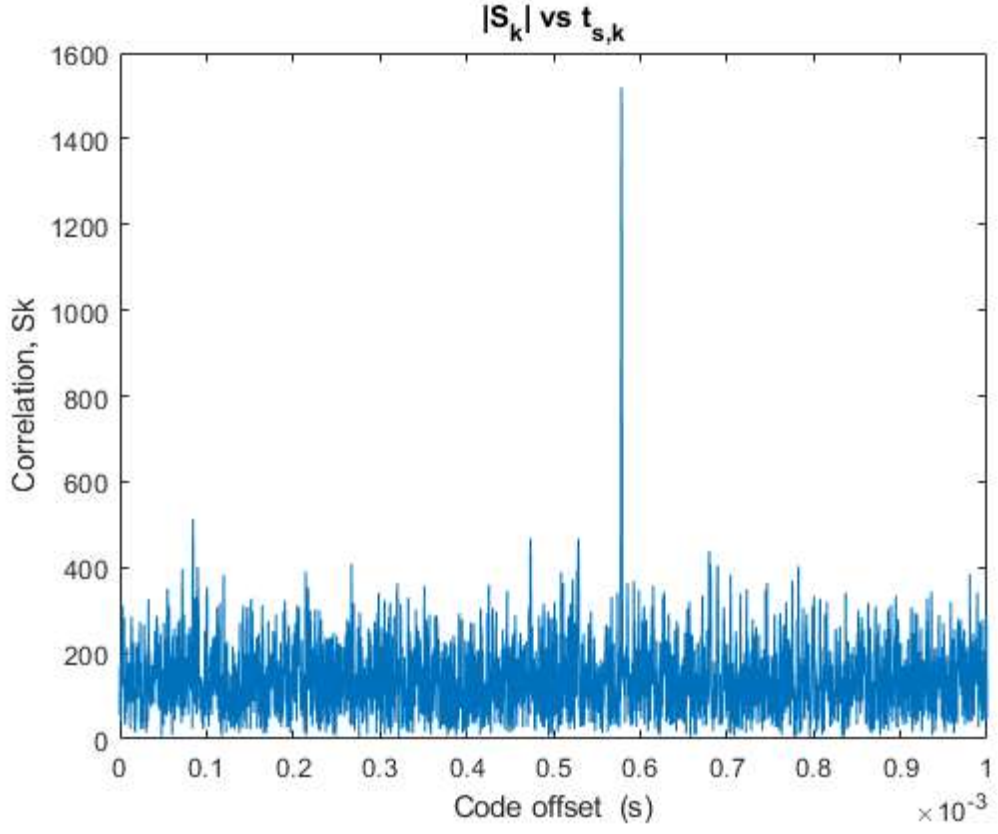
```
Apparent Doppler Frequency: -34800 Hz
Approximate Start Time from first sample: 370.125 microseconds
C/N0: 41.5946
------------------------------------------------------------
PRN :21
Apparent Doppler Frequency: -135200 Hz
Approximate Start Time from first sample: 42.875 microseconds
C/N0: 39.7039
------------------------------------------------------------
PRN :29
Apparent Doppler Frequency: -287200 Hz
Approximate Start Time from first sample: 584.85 microseconds
C/N0: 39.514
```

Find the best estimates for fd and ts

```
[~,strongPrn] = max(CN0)

% Sk vs fdk
for hh = 1:length(fD)
    Cr = fft(codeOS(:,strongPrn));
    fi = fD(hh) + fIF;
    xkTilde = Y(1:Nk).*exp(-1i*2*pi*fi*tk);
    XrTilde = fft(xkTilde);
    Zr = XrTilde.*(conj(Cr));
    zk = ifft(Zr);
    Sk(hh) = max(abs(zk));
end
figure,
plot(fD,Sk)
ylabel('Correlation, Sk')
xlabel('Doppler Frequency, fD (Hz)')
title(['|S_k| vs f_{D,k}'])

% Sk vs tsk
[~,idx] = max(Sk);
fd_best = fD(idx);
Cr = fft(codeOS(:,strongPrn));
fi = fd_best + fIF;
xkTilde = Y(1:Nk).*exp(-1i*2*pi*fi*tk);
XrTilde = fft(xkTilde);
Zr = XrTilde.*(conj(Cr));
zk = ifft(Zr);
Sk = abs(zk);
figure,
plot(tk,Sk)
ylabel('Correlation, Sk')
xlabel('Code offset  (s)')
title(['|S_k| vs t_{s,k}'])
```

```
strongPrn =

    11
```





Weak Signal Search

```
fD = [-300000:100:0];
tk = [0+110e-3:Nk-1+110e-3]'*T;
```

```
threshold = 36;
CN0 = zeros(37,1);
for mm = 31
    for kk = 1:length(fD)
        Cr = fft(codeOS(:,mm));
        fi = fD(kk) + fIF;
        Y = Y(round(110e-3/T):end);
        xkTilde = Y(1:Nk).*exp(-1i*2*pi*fi*tk);
        XrTilde = fft(xkTilde);
        Zr = XrTilde.*(conj(Cr));
        zk = ifft(Zr);
        [maxValue,kmax] = max(abs(zk).^2);
        CN0(mm) =10*log10((maxValue-2*sigmaIQ^2)/(2*sigmaIQ^2*Ta));
        if CN0(mm)  > threshold
            signalStrenghth(mm)=CN0(mm);
            start_time(mm) = tk(kmax+1)*10^6;
            apparent_fD(mm) = fD(kk);
            disp('------------------------------------------------------------')
            disp(['PRN :',num2str(mm)])
            disp(['Apparent Doppler Frequency: ', num2str(apparent_fD(mm)), ' Hz']);
            disp(['Approximate Start Time from first sample: ', num2str(start_time(mm)), ' microseconds']);
            disp (['C/N0: ', num2str(CN0(mm))])
            break;
        end
    end
end
```

```
------------------------------------------------------------
PRN :31
Apparent Doppler Frequency: -300000 Hz
Approximate Start Time from first sample: 173.7942 microseconds
C/N0: 36.9558
```

Published with MATLAB® R2023a

Contents


```
clear all; close all; clc;
```

White Frequency Noise (Phase Random Walk)

```
%----Simulation Setup
% Number of simulations
ensemble = 1000;
% Number of Samples per simulation;
Ns = 49999;
% Sampling interval
T = 0.001; % 1ms Given
% Noise parameters
sigma_omega = 0.01;     % radians

%----- Simulation
DeltaOmega = sigma_omega*randn(Ns,ensemble);
DeltaTheta_omega = cumsum(DeltaOmega,1);
for ii= 1: ensemble
    Ccoh(ii) =computeCoherence(DeltaTheta_omega(:,ii),Ns);
end
Ccoh2_mean = mean(Ccoh.^2)
tau = T*Ns;
fprintf(['Coherence Time from white frequency noise: %f \n'],tau)
```

```
Ccoh2_mean =

    0.5138

Coherence Time from white frequency noise: 49.999000
```

White Frequency Rate Noise (Frequency Random Walk)

```
%---- Simulation Setup
% Number of simulations
ensemble = 1000;
% Number of Samples per simulation;
Ns_alpha = 1600;
% Sampling interval
T = 0.001; % 1ms Given
% Noise parameters
sigma_alpha = 0.0001;   % radians


%----- Simulation
DeltaAlpha = sigma_alpha*randn(Ns_alpha,ensemble);
DeltaOmega_alpha = cumsum(DeltaAlpha,1);
```

```
DeltaTheta_alpha = cumsum(DeltaOmega_alpha,1);
for ii= 1: ensemble
    Ccoh_alpha(ii) =computeCoherence(DeltaTheta_alpha(:,ii),Ns_alpha);
end
Ccoh2_mean_alpha = mean(Ccoh_alpha.^2)
tau_alpha = T*Ns_alpha;
fprintf(['Coherence Time from white frequency Rate noise: %f \n'],tau_alpha)
```

```
Ccoh2_mean_alpha =

    0.4887

Coherence Time from white frequency Rate noise: 1.600000
```

White Phass noise

```
%----Simulation Setup
% Number of simulations
ensemble = 1000;
% Number of Samples per simulation;
Ns = 10000;
% Sampling interval
T = 0.001; % 1ms Given
% Noise parameters
sigma = 0.8;     % radians

%----- Simulation
Delta = sigma*randn(Ns,ensemble);
for ii= 1: ensemble
    Ccoh(ii) =computeCoherence(Delta(:,ii),Ns);
end
Ccoh2_mean = mean(Ccoh.^2)
tau = T*Ns;
fprintf(['Coherence Time from white frequency noise: Ccoh^2 stalls at 0.5275'])

fprintf(['\n ---------------------------------------------------\n'])
fprintf(['It doesn''t make sense to estimate the coherence time for the \n'...
        'white phase noise process because with the coherence time, we also seek \n ' ...
        'how many samples we can accumulate before the coherence drops to a value.\n' ...
        'The individual samples of this white noise are completely independent \n' ...
        'of each other and random. The samples are inherently not related to each other.'])
```

```
Ccoh2_mean =

    0.5274

Coherence Time from white frequency noise: Ccoh^2 stalls at 0.5275
 ---------------------------------------------------
It doesn't make sense to estimate the coherence time for the
white phase noise process because with the coherence time, we also seek
 how many samples we can accumulate before the coherence drops to a value.
The individual samples of this white noise are completely independent
of each other and random. The samples are inherently not related to each other.
```

Published with MATLAB® R2023a

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

Ccoh = abs((1/N)*sum(exp(1i*DeltaThetaVec(1:N))));
```

```
Not enough input arguments.

Error in computeCoherence (line 26)
Ccoh = abs((1/N)*sum(exp(1i*DeltaThetaVec(1:N))));
```

Published with MATLAB® R2023a

(6)

We want to find...

$$\Lambda(z) = \frac{p(z|\theta \in \chi_1)}{p(z|\theta \in \chi_0)} \underset{H_0}{\overset{H_1}{\gtrless}} U$$

$$p(z|\theta \in \chi_0) = \frac{1}{2\pi\sigma^2}\left[-\frac{1}{2\sigma^2}(z-\mu)^T(z-\mu)\right]$$

$$= \frac{1}{2\pi\sigma^2}\left[e^{-\frac{1}{2\sigma^2}(z-\mu)^T(z-\mu)}\right]$$

$$= \frac{1}{2\pi\sigma^2}\left[e^{-\frac{1}{2\sigma^2}z^Tz}\right]$$

$$p(z|\theta \in \chi_0) = \int_0^{2\pi} p(z|\theta_1 = A, \theta_2 = \xi)\, p_2(\xi)\, d\xi$$

$$= \frac{1}{4\pi^2\sigma^2}\int_0^{2\pi} e^{-\frac{1}{2\sigma^2}[z-m(A,\xi)]^T[z-m(A,\xi)]}\, d\xi$$

$$\underline{p(z|\theta \in \chi_1)} = \frac{1}{4\pi^2\sigma^2}\int_0^{2\pi} e^{\frac{1}{2\sigma^2}[z-m(A,\xi)]^T[z-m(A,\xi)]}\, d\xi$$

$$p(z|\theta \in \chi_0)$$

$$\frac{1}{2\pi\sigma^2}\; e^{-\frac{1}{2\sigma^2}z^Tz}$$

$$m = m(\theta_1,\theta_2) = \begin{bmatrix} A\cos\theta_2 \\ A\sin\theta_2 \end{bmatrix} = \frac{e^{\frac{1}{2\sigma^2}z^Tz}}{2\pi}\int_0^{2\pi} e^{\frac{-1}{2\sigma^2}[z-m(A,\xi)]^T[z-m(A,\xi)]}\, d\xi$$

$$\underline{z} \triangleq [z_1, z_2]$$

$$z^Tz - 2z^Tm(A,\xi) + m(A,\xi)^Tm(A,\xi)$$

$$z_1 = A\cos\theta_2 + w_1$$

$$z_2 = A\sin\theta_2 + w_2$$

$$z^Tz = [A\cos\theta_2 + w_1 \quad A\sin\theta_2 + w_2]\begin{bmatrix} A\cos\theta_2 + w_1 \\ A\sin\theta_2 + w_2 \end{bmatrix}$$

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}$$

$$= [m+\omega]^T[m+\omega]$$

$$= m^Tm + \cancel{m^T\omega + \omega^Tm + \omega^T\omega}^{\,0}$$

0, since $\omega \sim N(0,\sigma^2)$

$$= \begin{bmatrix} A\cos\theta_2 & A\sin\theta_2 \end{bmatrix} \begin{bmatrix} A\cos\theta_2 \\ A\sin\theta_2 \end{bmatrix}$$

$$= A^2\cos^2\theta_2 + A^2\sin^2\theta_2$$

$$Z^TZ = A^2$$

$[z_1 z_2]$

$$Z^TZ - 2Z^T\begin{bmatrix} A\cos\theta_2 \\ A\sin\theta_2 \end{bmatrix} + \begin{bmatrix} A\cos\theta_2 & A\sin\theta_2 \end{bmatrix} \begin{bmatrix} A\cos\theta_2 \\ A\sin\theta_2 \end{bmatrix}$$

$$= A^2 - 2Az_1\cos\theta_2 - 2Az_2\sin\theta_2 + A^2$$

$$= 2A^2 - 2A(z_1\cos\theta_2 - z_2\sin\theta_2)$$

$$\Lambda(z) = \frac{e^{\frac{A^2}{2\sigma^2}}}{2\pi} \int_0^{2\pi} \exp\left[-\frac{1}{2\sigma^2}(2A^2 - 2A(z_1\cos\theta_2 - z_2\sin\theta_2)\right] d\psi$$

$$= \frac{1}{2\pi} e^{\frac{A^2}{2\sigma^2}} e^{-\frac{A^2}{\sigma^2}} \int_0^{2\pi} \exp\left[\frac{A}{\sigma^2}(z_1\cos\theta_2 + z_2\sin\theta_2)\right] d\psi$$

$$= \frac{e^{\frac{-A^2}{2\sigma^2}}}{2\pi} \int_0^{2\pi} \exp\left[\frac{A}{\sigma^2}(z_1\cos\xi + z_2\sin\xi)\right] d\xi$$

divide this term over to v

$$\Rightarrow \int_0^{2\pi} \exp\left[\frac{A}{\sigma^2}(z_1\cos\xi + z_2\sin\xi)\right] d\xi \gtrless v' = \frac{2\pi v}{e^{\frac{-A^2}{2\sigma^2}}}$$

$$z_1\cos\xi = \sum x(j)\cos\xi \qquad z_2 = \sum x(j)\sin\xi$$

Collection of observations

$$\Lambda'(z) = \int_0^{2\pi} \exp\left[\frac{A}{\sigma^2}\left(\sum_j x(j)\cos\xi + \sum_j x(j)\sin\xi\right)\right] d\xi \underset{H_0}{\overset{H_1}{\gtrless}} v'$$

Since ...

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

$$\cos(a+b) = \cos a \cos b - \sin a \sin b$$

$$S(j,\xi) \triangleq D\, C\left(\cos[2\pi f_{IF}\tau_j]\cos[\xi] - \sin[2\pi f_{IF}\tau_j]\sin[\xi]\right)$$

$$= S_c(j)\cos\xi - S_s(j)\sin\xi$$

$$\Lambda'(z) = \int_0^{2\pi} \exp\left[\frac{A}{\sigma^2}\left(\sum x(j)\,(\cos\xi + \sin\xi)\right]\right] d\xi$$

$$= \int_0^{2\pi} \exp\left[\frac{A}{\sigma^2}\left[\sum x(j)\, S(j,\xi)\right]\right] d\xi$$

$$= \int_0^{2\pi} \exp\left[\frac{A}{\sigma^2}\left(\sum x(j)\left[S_c(j)\cos\xi - S_s(j)\sin\xi\right]\right)\right] d\xi$$

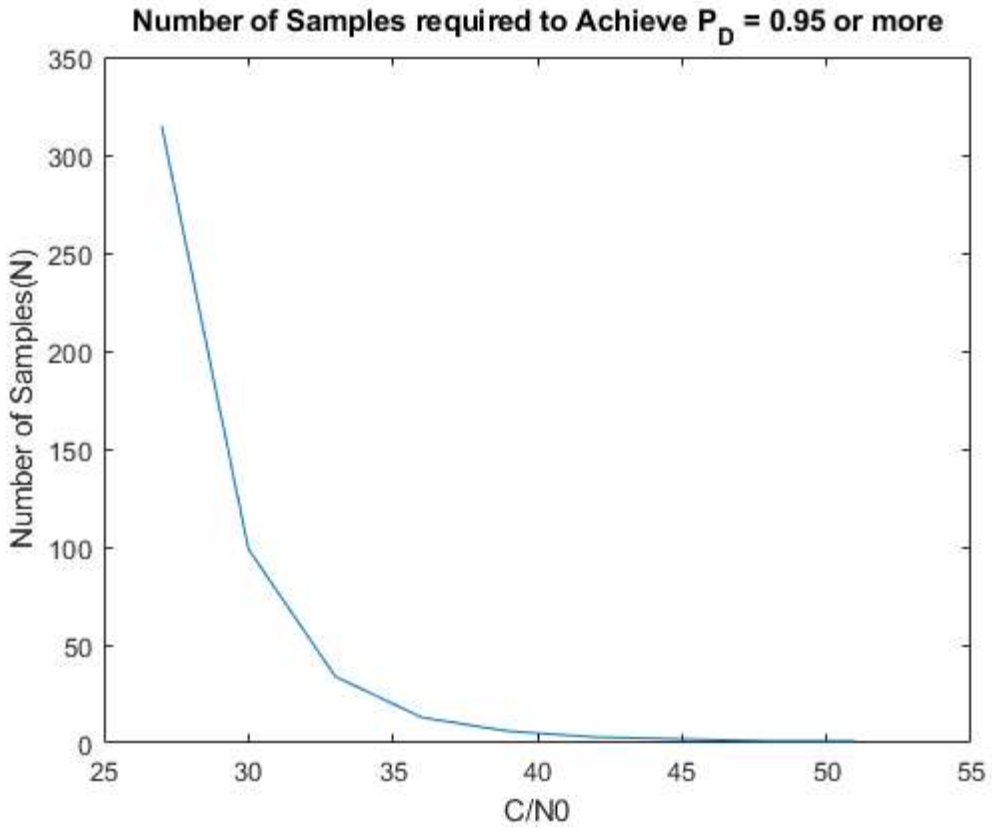
A is not necessary because it is again a scaling term that can be incorporated into threshold instead of forming detection statistic.

Contents


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

change N

```
%----- Setup
clear; clc;
CN0 = [27,30,33,36,39,42,45,48,51];
Pd=1;
testN = flip([1:500],2);
for n = 1:length(CN0)
    Pd=1;
    for k = 1: length(testN)
        s.C_N0dBHz = CN0(n);
        s.N = testN(k);
        s.PfaAcq = 0.0001;
        s.Ta = 0.001;
        s.fMax = 7000;
        s.nCodeOffsets = 1023*5;
        s.ZMax = 1000;
        s.delZ = 0.1;

        %----- Execute
        [pZ_H0,pZ_H1,lambda0,Pd,ZVec] = performAcqHypothesisCalcs(s);
        if Pd <= 0.95
            N(n) =testN(k)+1;
            break;
        end
    end
end

% Plot Results
figure,
plot(CN0,[N,1,1])
xlabel('C/N0')
ylabel('Number of Samples(N)')
title('Number of Samples required to Achieve P_D = 0.95 or more')
```

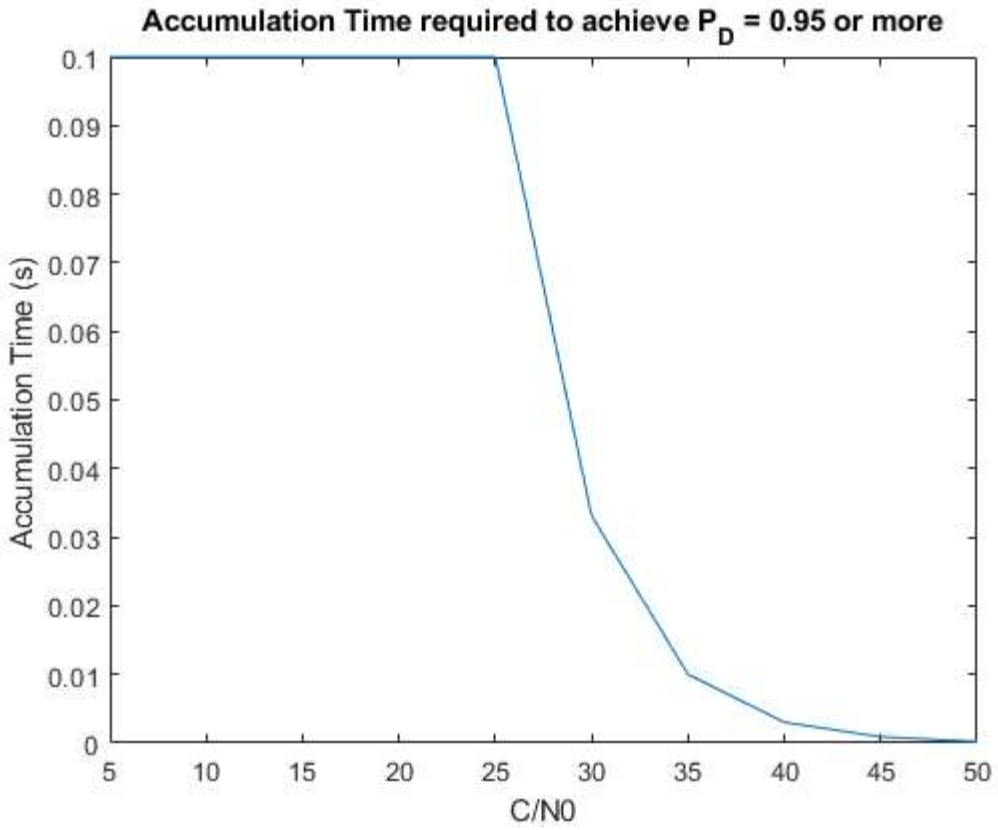


Change Ta

```
%----- Setup
clear; clc;
CN0 = linspace(5,50,10);
testTa = flip([0.0001:0.0001:0.1]);
for n = 1:length(CN0)
    Pd=1;
    for k = 1: length(testTa)
        s.C_N0dBHz =CN0(n);
        s.N = 1;
        s.PfaAcq = 0.001;
        s.Ta = testTa(k);
        s.fMax = 7000;
        s.nCodeOffsets = 1023*5;
        s.ZMax = 1000;
        s.delZ = 0.1;

        %----- Execute
        [pZ_H0,pZ_H1,lambda0,Pd,ZVec] = performAcqHypothesisCalcs(s);
        if Pd <= 0.95
            T(n) =testTa(k);
            break;
        end
    end
end

% Plot Results
figure,
plot(CN0,T)
xlabel('C/N0')
```

```
ylabel('Accumulation Time (s) ')
title('Accumulation Time required to achieve P_D = 0.95 or more')


fprintf(['Coherent integration is more efficient than noncoherent integration. \n' ...
    'For C/N0 = 30, you require more 99 accumulations for the noncoherent \n' ...
    'integration with accumulation time = 0.001 second, which means the total \n' ...
    'accumulation time is 0.099 seconds. In contrast, coherernt integration \n' ...
    'required only one accumulation for 0.0331 second to achieve PD = 0.95.'])

%
% %----- Visualize the results
% figure(2);
% [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');
% 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);
% shg
```

```
Coherent integration is more efficient than noncoherent integration.
For C/N0 = 30, you require more 99 accumulations for the noncoherent
integration with accumulation time = 0.001 second, which means the total
accumulation time is 0.099 seconds. In contrast, coherernt integration
required only one accumulation for 0.0331 second to achieve PD = 0.95.
```



Published with MATLAB® R2023a

```
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:
%
%
% Z =        N  |      |^2
%           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), E[nIk nIi] = E[nQk nQi] = 1 for k = i and 0
% for k != i, and E[nIk nQi] = 0 for all 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. This procedure is
%                       straightforward if we assume that the detection
%                       statistics from the search cells are independent
%                       of one another.
%       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:
%
%
%+=========================================================================+
sigma_IQ = 1;
CN0 = 10^(s.C_N0dBHz/10);    % Convert to linear scale
Abark = sqrt(CN0*sigma_IQ^2*s.Ta*8/s.N^2);
rhok = (s.N*Abark)/(2*sigma_IQ);
lambda = s.N*rhok^2;
ZVec = [0:s.delZ:s.ZMax];
pZ_H0=chi2pdf(ZVec,2*s.N);
pZ_H1=ncx2pdf(ZVec,2*s.N,lambda);

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

PF = 1-(1-s.PfaAcq)^(1/ nCells);

lambda0 = chi2inv(1-PF,2*s.N);

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

```
Not enough input arguments.
```

```
Error in performAcqHypothesisCalcs (line 94)
CN0 = 10^(s.C_N0dBHz/10);    % Convert to linear scale
```

Published with MATLAB® R2023a