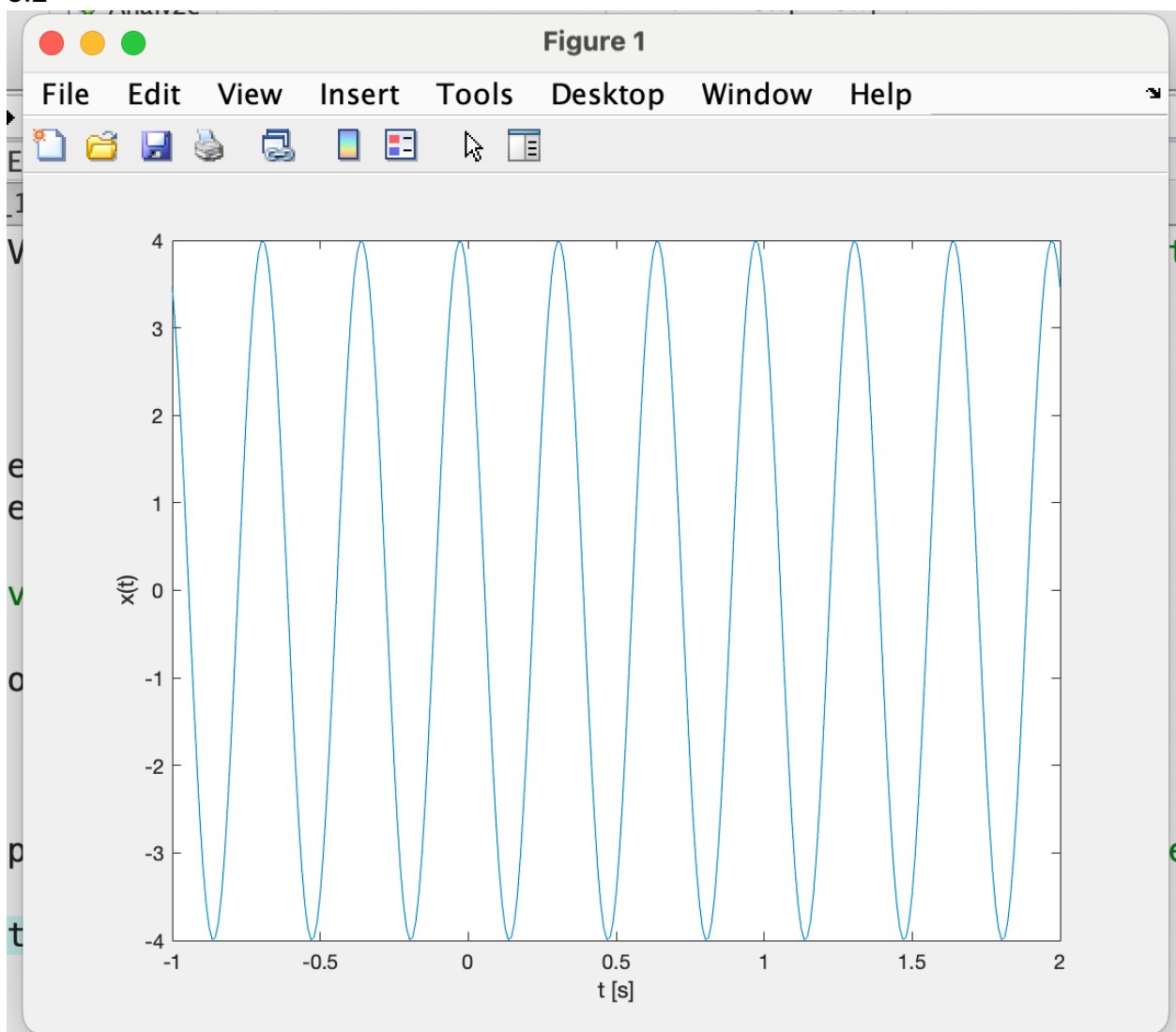


3.1 Show Debugging Features

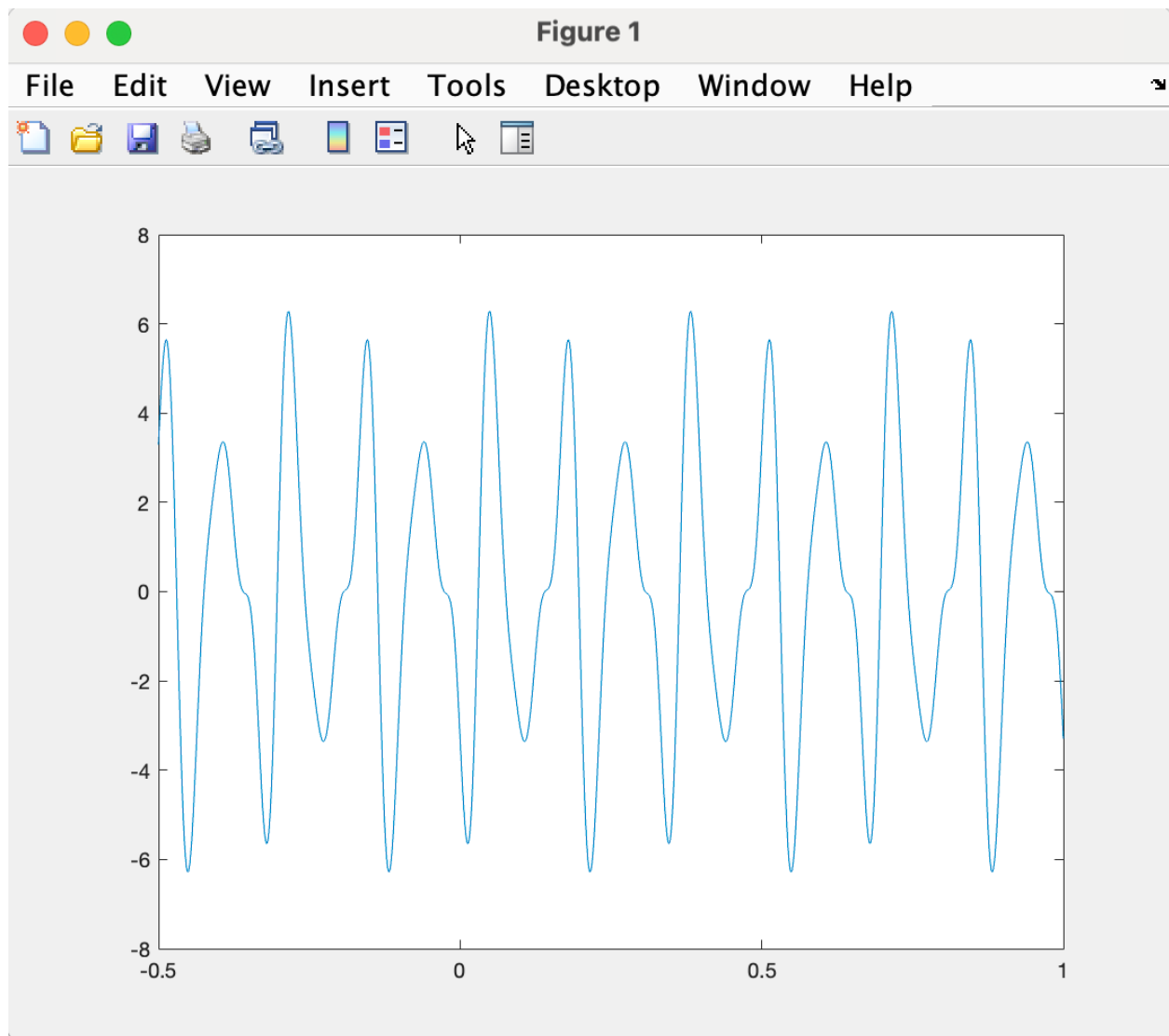
```
26 A = abs(sigIn.complexAmp);  
27  
28 tt = tstart: dt : dur + tstart; %-- Create the vector of times  
29  
30 → xx = A*cos(2 * pi * sigIn.freq * tt(0, 3) + angle(sigIn.complexAmp)); %-- Vectorize th  
31  
32 sigOut.times = tt; %-- Put vector of times into the output structure  
33 sigOut.values = xx; %-- Put values into the output structure  
34 end
```

Above shows how the code stops when we try to construct xx from only 4 samples of the vector tt

3.2

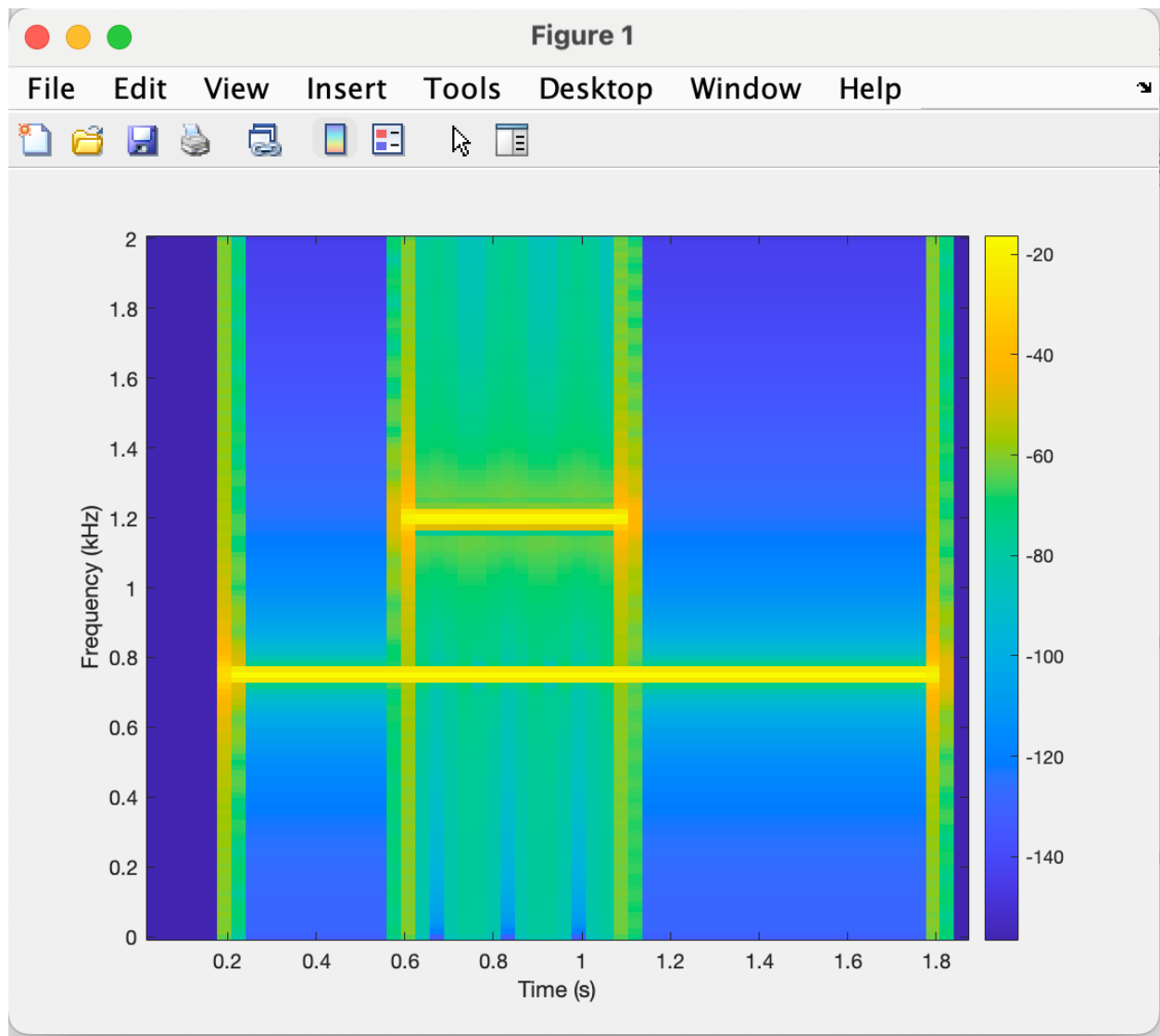


3.3.1



Period is .3343 seconds and is represents the fundamental frequency of 3 Hz, the GCD of all the frequencies used to construct this signal.

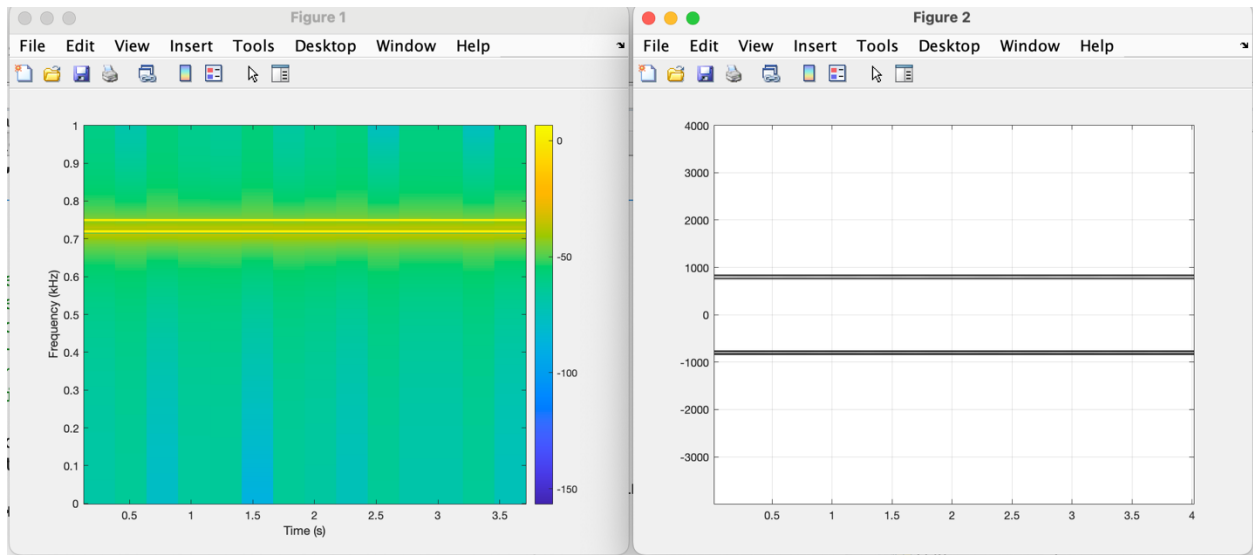
3.3.2



```
nStart = round(fs * tStart(kk))+1; %-- add one to avoid zero index
```

```
nStop = nStart+Lnew-1; %<===== Add code
```

3.4



```
testingBeat.values = real( testingBeat.X1*exp(j*2*pi*testingBeat.f1 *
testingBeat.times ) + testingBeat.X2*exp(j*2*pi*testingBeat.f2 *
testingBeat.times ) );
```

where the values from testingBeat struct is the values of the generated beat signal.

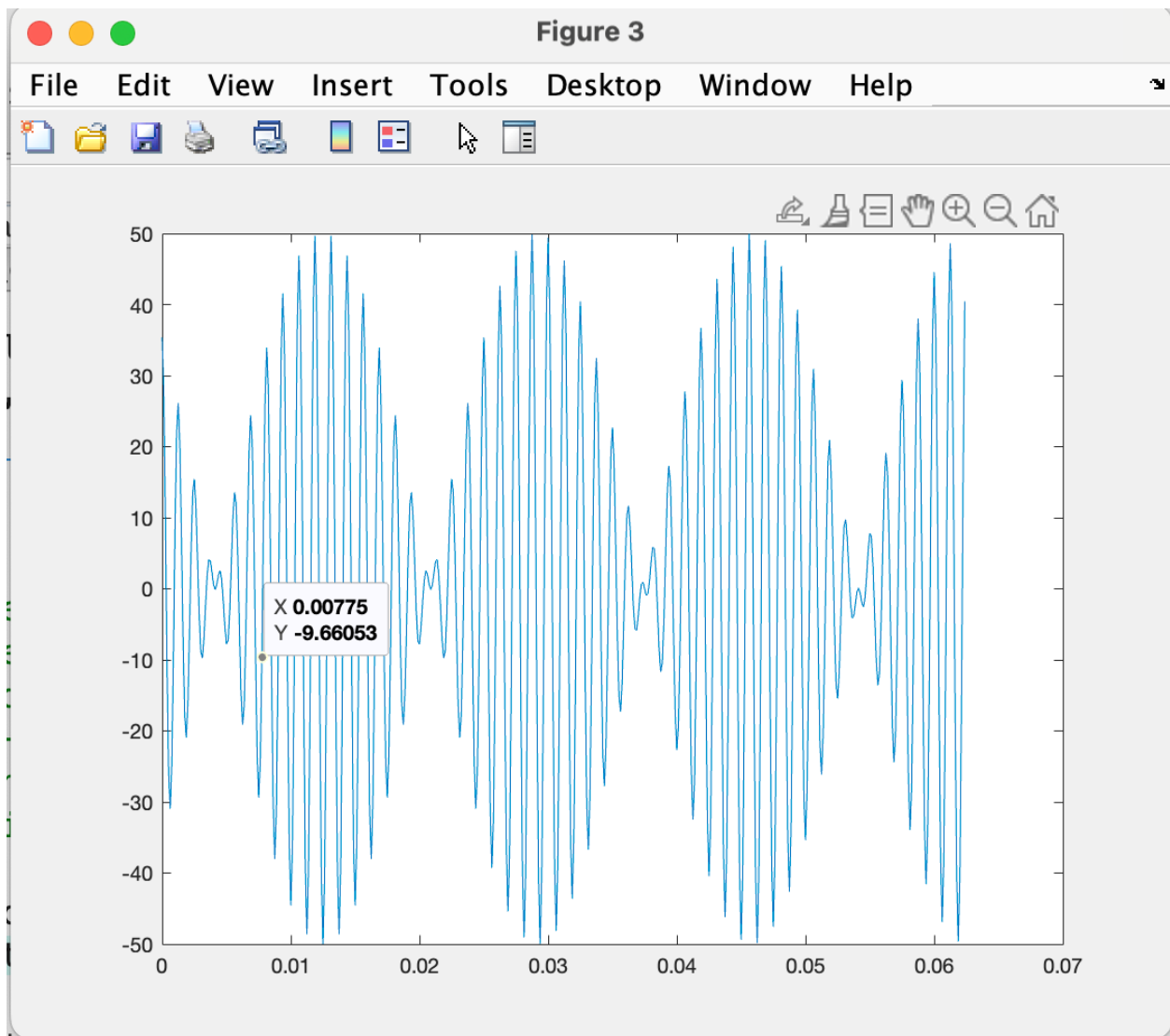
```
function sigBeatSum = sum2BeatStruct( sigBeatIn ) %
%--- Assume the five basic fields are present, plus the starting and ending
times
%--- Add the four fields for the parameters in Equation (4)
%
% sigBeatSum.f1, sigBeatSum.f2, sigBeatSum.X1, sigBeatSum.X2

sigBeatSum.f1 = sigBeatIn.fc-sigBeatIn.fDelt;
sigBeatSum.f2 = sigBeatIn.fc+sigBeatIn.fDelt;
% Amplitude --> See Eq. (4)
A1 = sigBeatIn.Amp / 2;
A2 = sigBeatIn.Amp / 2;

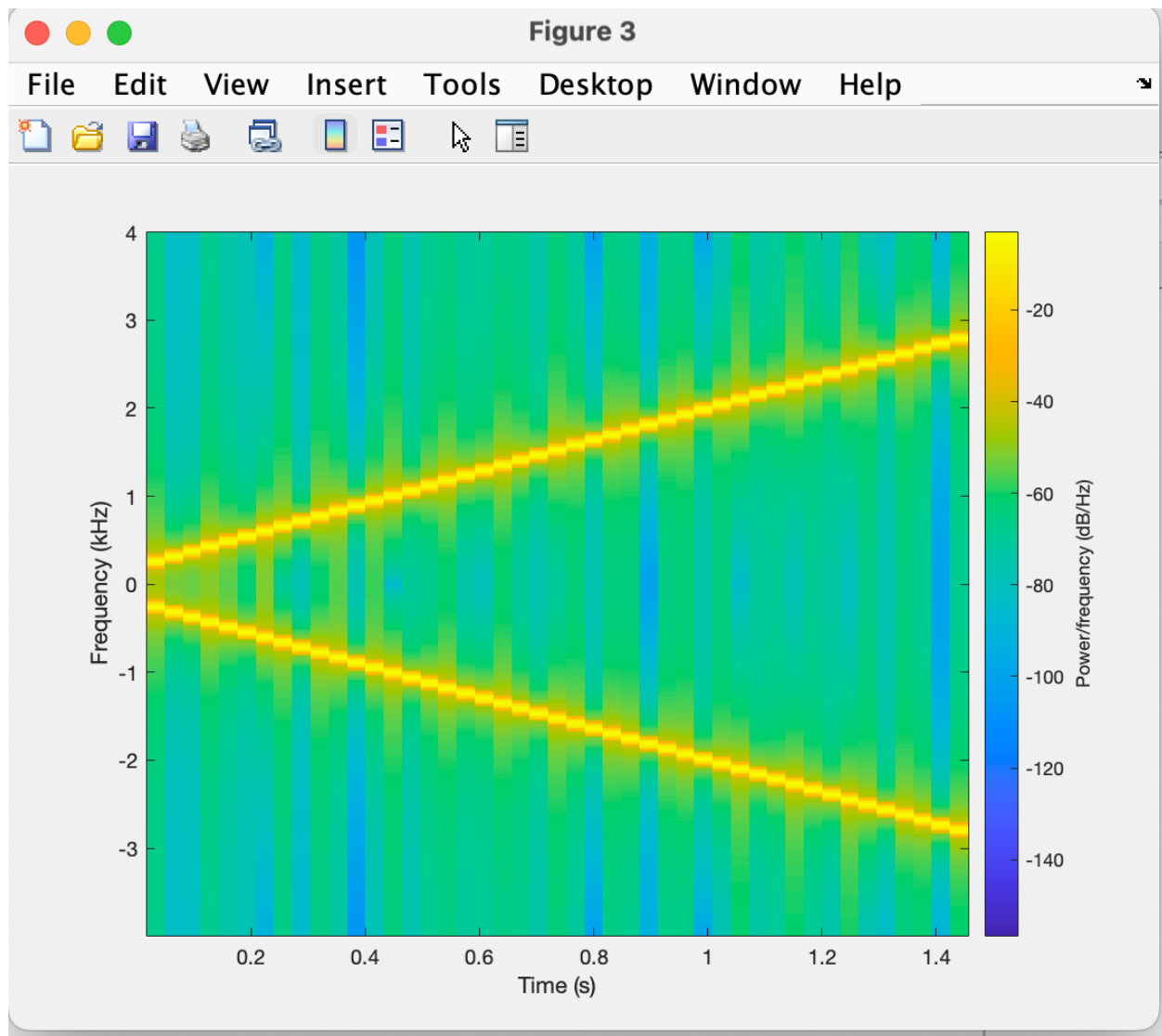
% Phase --> See Eq. (4)
phi1 = sigBeatIn.phic - sigBeatIn.phiDelt;
phi2 = sigBeatIn.phic + sigBeatIn.phiDelt;

% Compute complex amplitude
sigBeatSum.X1 = A1 * exp(j * phi1);
sigBeatSum.X2 = A2 * exp(j * phi2);

end
```



3.5



```
mu = .5*(sigLFM.slope);

f0 = sigLFM.f1;

psi = 2*pi*( f0*tt + mu*tt.*tt) + angle(sigLFM.complexAmp);
xx = real( abs(sigLFM.complexAmp) * exp(j*psi) );
```

Code for all

%% 3.2

```
mySig.freq = 3; %-- (in hertz)
```

```
mySig.complexAmp = 4*exp(j*pi/6);
```

```
dur = 3;
```

```
start = -1;
```

```
dt = 1/(32*mySig.freq);
```

```
mySigWithVals = makeCosVals( mySig, dur, start, dt );
```

```
dbstop if error
```

```
%- Plot the values in sigWithVals
```

```
plot( mySigWithVals.times, mySigWithVals.values) %<--- complete the plot  
statement
```

```
xlabel('t [s]')
```

```
ylabel('x(t)')
```

```
function sigOut = makeCosVals(sigIn, dur, tstart, dt ) %
```

```
freq = sigIn.freq;
```

```
X = sigIn.complexAmp;
```

```
%
```

```
%...(Fill in several lines of code)...
```

```
%
```

```
A = abs(sigIn.complexAmp);
```

```
tt = tstart: dt : dur + tstart%-- Create the vector of times
```

```
xx = A*cos(2 * pi * sigIn.freq * tt + angle(sigIn.complexAmp)); %--  
Vectorize the cosine evaluation
```

```
sigOut.times = tt; %-- Put vector of times into the output structure
```

```
sigOut.values = xx; %-- Put values into the output structure
```

```
end
```

%% 3.3.1

```
ss(1).freq = 21; ss(1).complexAmp = exp(j*pi/4);
```

```
ss(2).freq = 15; ss(2).complexAmp = 2i;
```

```
ss(3).freq = 9; ss(3).complexAmp = -4;
```

```
%
```

```
dur = 1;
```

```
tstart = -0.5;
```

```
dt = 1/(21*32); %-- use the highest frequency to define delta_t
```

```
%
```

```
ssOut = addCosVals( ss, dur, tstart, dt );
```

```
%
```

```
plot( ssOut.times, ssOut.values) %
```

```

function sigOut = addCosVals( cosIn, dur, tstart, dt )
%ADDCOSVALS Synthesize a signal from sum of sinusoids
%(do not assume all the frequencies are the same)
%
% usage: sigOut = addCosVals( cosIn, dur, tstart, dt )
%
% cosIn = vector of structures; each one has the following fields:
%   cosIn.freq = frequency (in Hz), usually none should be negative
%   cosIn.complexAmp = COMPLEX amplitude of the cosine
%
% dur = total time duration of all the cosines
% start = starting time of all the cosines
% dt = time increment for the time vector
%
% The output structure has only signal values because it is not necessarily a
sinusoid
%   sigOut.values = vector of signal values at t = sigOut.times
%   sigOut.times = vector of times, for the time axis
%
% The sigOut.times vector should be generated with a small time increment
that
%   creates 32 samples for the shortest period, i.e., use the period
%   corresponding to the highest frequency cosine in the input array of
structures.

```

```

% <--- Write your code here --->

```

```

tt = tstart:dt:dur;
xx = 0*tt;

```

```

for k = 1:length(cosIn)

```

```

    xNew = abs(cosIn(k).complexAmp) * cos(2*pi*cosIn(k).freq*tt +
angle(cosIn(k).complexAmp));

```

```

    xx(1:length(tt)) = xx(1:length(tt))+xNew;
end

```

```

sigOut.times = tt;
sigOut.values = xx;

```

```

end

```

```

%% 3.3.2
% Modify the following code from Prelab 2.6

```

```

amps = [1, 1] ;
freqs = [1200, 750] ;

```



```

phases = [0, 0] ;
fs = 4000;
tStart = [.6, .2];
durs = [.5, 1.6];

maxTime = max(tStart+durs) + 0.1; %-- Add time to show signal ending

durLengthEstimate = ceil(maxTime*fs);

tt = (0:durLengthEstimate)*(1/fs); %-- be conservative (add one)
xx = 0*tt; %--make a vector of zeros to hold the total signal

for kk = 1:length(amps)
    nStart = round(fs * tStart(kk))+1; %-- add one to avoid zero index

    xNew = shortSinus(amps(kk), freqs(kk), phases(kk), fs, durs(kk));

    Lnew = length(xNew);

    nStop = nStart+Lnew-1; %<===== Add code

    xx(nStart:nStop) = xx(nStart:nStop) + xNew;
end

tt = (1/fs)*(0:length(xx)-1);
figure
spectrogram(xx,256,[ ],[ ],fs,'yaxis'); colorbar

```

```

function xs = shortSinus(amp, freq, pha, fs, dur)
% amp = amplitude
% freq = frequency in cycle per second
% pha = phase, time offset for the first peak
% fs = number of sample values per second
% dur = duration in sec
%
tt = 0 : 1/fs : dur; % time indices for all the values
xs = amp * cos( freq*2*pi*tt + pha );
end

```

```

%% 3.4
%% Template of sigBeat Struct
% sigBeat.Amp = 10;      %-- B in Equation (3)
% sigBeat.fc = 480;      %-- center frequency in Eq. (3)
% sigBeat.phic = 0;      %-- phase of 2nd sinusoid in Eq. (3)
% sigBeat.fDelt = 20;    %-- modulating frequency in Eq. (3)
% sigBeat.phiDelt = -2*pi/3; %-- phase of 1st sinusoid Eq. (3)
% sigBeat.t1 = 1.1;      %-- starting time
% sigBeat.t2 = 5.2;      %-- ending time %
%
%----- extra fields for the parameters in Equation (4)
%
% sigBeat.f1             %-- frequencies in Equation (4)
% sigBeat.f2             %--

```

```

% sigBeat.X1      %-- complex amps for sinusoids in Equation (4)
% sigBeat.X2      %-- derived from A's and phi's
%
% sigBeat.values  %-- vector of signal values sigBeat.times
% sigBeat.times   %-- vector of corresponding times

%% 3.4(a)
% Complete the sum2BeatStruct function at the end

%% 3.4(b)
% Create a beat signal with two frequency components:
% one at 720 Hz and one at 750 Hz
fs = 2000;
sigBeat.Amp = 10; %-- B in Equation (3)
sigBeat.fc = 735; %-- center frequency in Eq. (3)
sigBeat.phic = 0; %-- phase of 2nd sinusoid in Eq. (3)
sigBeat.fDelt = 15; %-- modulating frequency in Eq. (3)
sigBeat.phiDelt = 0; %-- phase of 1st sinusoid Eq. (3)

sigBeat.t1 = 0; %-- starting time
sigBeat.t2 = 4; %-- ending time %

testingBeat = sum2BeatStruct( sigBeat );

testingBeat.times = sigBeat.t1:1/fs:sigBeat.t2;

testingBeat.values = real( testingBeat.X1*exp(j*2*pi*testingBeat.f1 *
testingBeat.times + angle(testingBeat.X1)) ...
+ testingBeat.X2*exp(j*2*pi*testingBeat.f2 * testingBeat.times +
angle(testingBeat.X2)) );

figure
spectrogram(testingBeat.values,1024,[ ],[ ],fs,'yaxis'); colorbar
soundsc(testingBeat.values, fs)

%% 3.4.1(a)
fs = 8000;
sigBeat.Amp = 50; %-- B in Equation (3)
sigBeat.fc = 800; %-- center frequency in Eq. (3)
sigBeat.phic = 0; %-- phase of 2nd sinusoid in Eq. (3)
sigBeat.fDelt = 30; %-- modulating frequency in Eq. (3)
sigBeat.phiDelt = pi/4; %-- phase of 1st sinusoid Eq.~(3)
sigBeat.t1 = 0; %-- starting time
sigBeat.t2 = 4.04; %-- ending time %

testingBeat = sum2BeatStruct( sigBeat );
testingBeat.times = sigBeat.t1:1/fs:sigBeat.t2;
testingBeat.values = real( testingBeat.X1*exp(j*2*pi*testingBeat.f1 *
testingBeat.times ) ...
+ testingBeat.X2*exp(j*2*pi*testingBeat.f2 * testingBeat.times ) );

figure
plot( testingBeat.times(1:500), testingBeat.values(1:500) )

```

```

%% 3.4.1(c)
plotspec(testingBeat.values+j*1e-12,fs,512); grid on, shg

%%
function sigBeatSum = sum2BeatStruct( sigBeatIn ) %
%--- Assume the five basic fields are present, plus the starting and ending
times
%--- Add the four fields for the parameters in Equation (4)
%
% sigBeatSum.f1, sigBeatSum.f2, sigBeatSum.X1, sigBeatSum.X2

sigBeatSum.f1 = sigBeatIn.fc-sigBeatIn.fDelt;
sigBeatSum.f2 = sigBeatIn.fc+sigBeatIn.fDelt;
% Amplitude --> See Eq. (4)
A1 = sigBeatIn.Amp / 2;
A2 = sigBeatIn.Amp / 2;

% Phase --> See Eq. (4)
phi1 = sigBeatIn.phic - sigBeatIn.phiDelt;
phi2 = sigBeatIn.phic + sigBeatIn.phiDelt;

% Compute complex amplitude
sigBeatSum.X1 = A1 * exp(j * phi1);
sigBeatSum.X2 = A2 * exp(j * phi2);

end

```

```

%% 3.5

```

```

myLFMsig.f1 = 200;
myLFMsig.t1 = 0;
myLFMsig.t2 = 1.5;
myLFMsig.slope = 1800;
myLFMsig.complexAmp = 10*exp(j*0.3*pi);
dt = 1/8000; % 8000 samples per sec is the sample rate
outLFMsig = makeLFMvals(myLFMsig,dt);
%- Plot the values in outLFMsig

plot(outLFMsig.times, outLFMsig.values)

%- Make a spectrogram for outLFMsig to see the linear frequency change
spectrogram(outLFMsig.values, 512,[ ],[ ],1/dt,'centered','yaxis')

function sigOut = makeLFMvals( sigLFM, dt )
% MAKELFMVALS      generate a linear-FM chirp signal
%
% usage: sigOut = makeLFMvals( sigLFM, dt )
% sigLFM.f1 = starting frequency (in Hz) at t = sigLFM.t1
% sigLFM.t1 = starting time (in secs)
% sigLFM.t2 = ending time

```

```

% sigLFM.slope = slope of the linear-FM (in Hz per sec)
% sigLFM.complexAmp = defines the amplitude and phase of the FM signal
% dt = time increment for the time vector, typically 1/fs (sampling
frequency)
%
% sigOut.values = (vector of) samples of the chirp signal
% sigOut.times = vector of time instants from t=t1 to t=t2
%
if( nargin < 2 ) %-- Allow optional input argument for dt
    dt = 1/8000; %-- 8000 samples/sec
end

%-----NOTE: use the slope to determine mu needed in psi(t)
%----- use f1, t1 and the slope to determine f0 needed in psi(t)

tt = sigLFM.t1:dt:sigLFM.t2;
mu = .5*(sigLFM.slope);

f0 = sigLFM.f1;

psi = 2*pi*( f0*tt + mu*tt.*tt) + angle(sigLFM.complexAmp);
xx = real( abs(sigLFM.complexAmp) * exp(j*psi) );

sigOut.times = tt;
sigOut.values = xx;

end

```