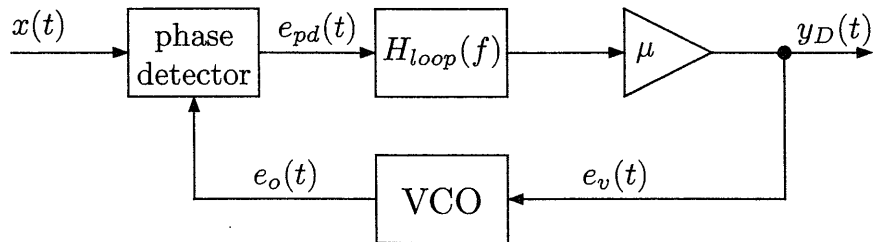
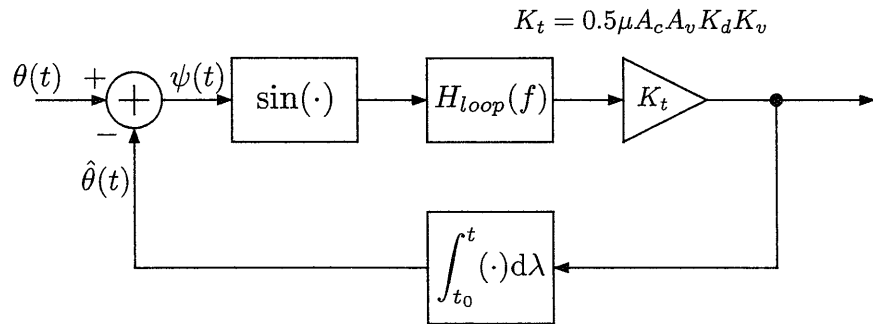


Problem 4. [30 pts. total] *Passband Simulation of a PLL.*

In this problem we wish to use the code from Computer Example 4.4 to create a passband simulation of a PLL and experiment with it. A passband model refers to the block diagram:



In the figure the external input signal is $x(t) = A_c \cos(2\pi f_c t + \theta(t))$ and the VCO output signal is $e_o(t) = A_v \sin(2\pi f_c t + \hat{\theta}(t))$. Then using various models for the blocks in the passband block diagram we were able to simplify to a baseband block diagram:



- (a) The first step is to create an input signal with a time-varying phase as in the above equation. That is we will implement $x(t) = A_c \cos(2\pi f_c t + \theta(t))$ over some interval of time $0 \leq t \leq T_f$. Since the simulation will start at $t = 0$ and there may be transients associated with startup we will pick a time t_1 where phase disturbances occur. Since you need now to represent the signal at the carrier frequency f_c the sampling rate of the simulation will need to be significantly higher than in the baseband case. Pick a sampling rate at least 10 times higher than f_c . Let $f_c = 1000$ Hz and simulate for at

least a few seconds. For a user-selectable frequency or phase step write the code that will generate the input signal $x(t)$. Experiment with it and verify the operation.

- (b) The baseband code from Computer Example 4.4 will produce a sampled version of the VCO output phase variable $\hat{\theta}(t)$. Explain how you would use it to compute the VCO output signal. Note that while the input $x(t)$ can be pre-computed, the VCO output $e_o(t)$ cannot be pre-computed because $\hat{\theta}(t)$ is a function of $\{\theta(s) : s \leq t\}$ owing to the memory created by the feedback.
- (c) To use the baseband code in the passband simulation we are trying to build requires us to compute the phase error terms directly from the f_c sinusoids rather than from the variables `s1` and `s2 = sin(s1)`. Design and code a multiplier-like phase detector which takes two quadrature sine waves as input and produces a phase error variable comparable to the sine of the phase error. You will need to average the output of the multiplier over a few cycles of the input sinusoids.
- (d) Then you should be able to close the loop and run a simulation at passband. Try to compare with some of the numbers used in the previous baseband simulations.
- (e) While you can plot the time evolution of the baseband phase variables to check the performance of your passband simulation, it is not terribly helpful to directly plot the input and VCO output sinusoids because their frequencies are so much higher than that of the baseband phase variables. Consider using the following code, which takes two sinusoidal vectors of length `npts` and creates a *movie* that shows how the loop evolves to lock. This is an example of a Lissajous curve. Code snippet follows ...

```

%% Beginning of Postprocessor

nn = 100;
kk = floor(npts/nn);

%Make a movie
pause on

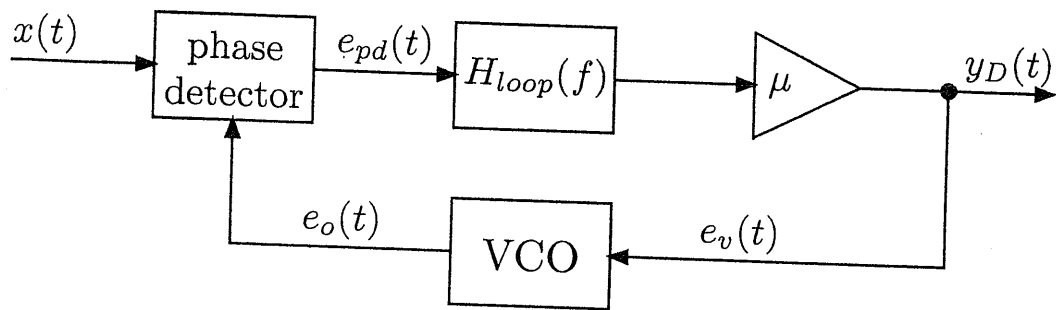
for i = 0:(kk-1)
    plot(s(1+i*nn:(i+1)*nn),svco(1+i*nn:(i+1)*nn))
    axis square;
    title(['Time = ',num2str(t((i+1)*nn))])
    xlabel('Input Sinusoid with Phase/Frequency Perturbation')
    ylabel('VCO Output Sinusoid')
    pause(0.1)
end

pause off

% End of Postprocessor

```

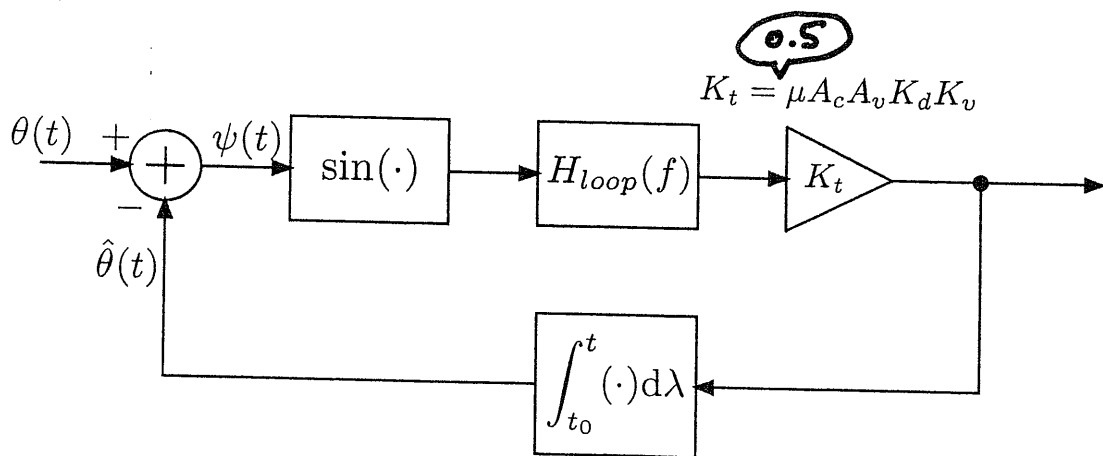
Problem 4 Modify CE 3.4 to create passband simulation of PLL and experiment with it. A passband model refers to the block diagram:



Above is bandpass model for PLL where

- External input signal: $x(t) = A_c \cos(2\pi f_c t + \theta(t))$.
- VCO output signal: $e_o(t) = A_v \sin(2\pi f_c t + \hat{\theta}(t))$.

Using various models for the blocks in the above diagram we are able to generate a baseband model:



CE 3.4 implements a baseband model such as that above. From the solution to a previous problem we found the block diagram description of the simulation loop in CE 3.4.

(a)

Input signal

$$x(t) = A_c \cos(2\pi f_c t + \Theta(t)) \quad 0 \leq t \leq T_f$$

Since our simulation will start at $t=0$ and there may be a transient associated with startup we will pick a time t_1 where phase disturbances occur

Phase Step

$$\Theta(t) = \Delta_\theta u(t-t_1)$$

Freq. Step

$$\Theta(t) = \Delta_\omega (t-t_1) u(t-t_1)$$

Recall the instantaneous frequency is $\frac{1}{2\pi} \dot{\Theta}(t)$. Therefore

$$\frac{1}{2\pi} \dot{\Theta}(t) = \frac{\Delta_\omega}{2\pi} u(t-t_1)$$

↳ size of the freq. step.

Will implement via sampling. Let $f_s = 1/T_s$ say its some multiple of f_c

$$f_s = N_s f_c \quad (N_s = 10 \text{ or so}).$$

Then input to the simulation will be a discrete-time signal

$$\begin{aligned} x[n] &= \cos(2\pi f_c n T_s + \Theta(n T_s)) \quad 0 \leq n \leq \lceil T_f / T_s \rceil \\ &= \cos(2\pi n / N_s + \Theta(n T_s)) \end{aligned}$$

Given the input phase signal, we can directly compute $x[n]$ in MATLAB.

$$\begin{aligned} T_s f_s &= 1 = N_s (f_c T_s) \\ \implies f_c T_s &= 1/N_s \end{aligned}$$

(b)

In the same fashion if we have the VCO output phase variable

$$\hat{\Theta}(t)$$

we can compute the VCO output signal

$$e_o[n] = \cos\left(2\pi n/N_s + \hat{\Theta}(nT_s)\right)$$

But note: While $x[n]$ can be pre-computed, $e_o[n]$ cannot since $\hat{\Theta}(nT_s)$ is a function of $\Theta(t)$ and it has memory.

The modifications made to the CE3.4 code include:

- 1) We precompute the input sinusoid "s" and allow for phase and frequency steps.
- 2) We remove variable "S1" (the current phase error) since it now needs to be computed from the input and VCO sinusoids.
- 3) The "vco_out" is used to create the VCO sinusoidal signal.
- 4) The a multiplier - LPF phase detector is to directly create the phase error variable "S2" (the sine of the phase error).
- 5) Finally we show plots of the input sinusoid vs. the vco output signal to show lock.

(c) See code for the passband phase detector.

(d) Comparison

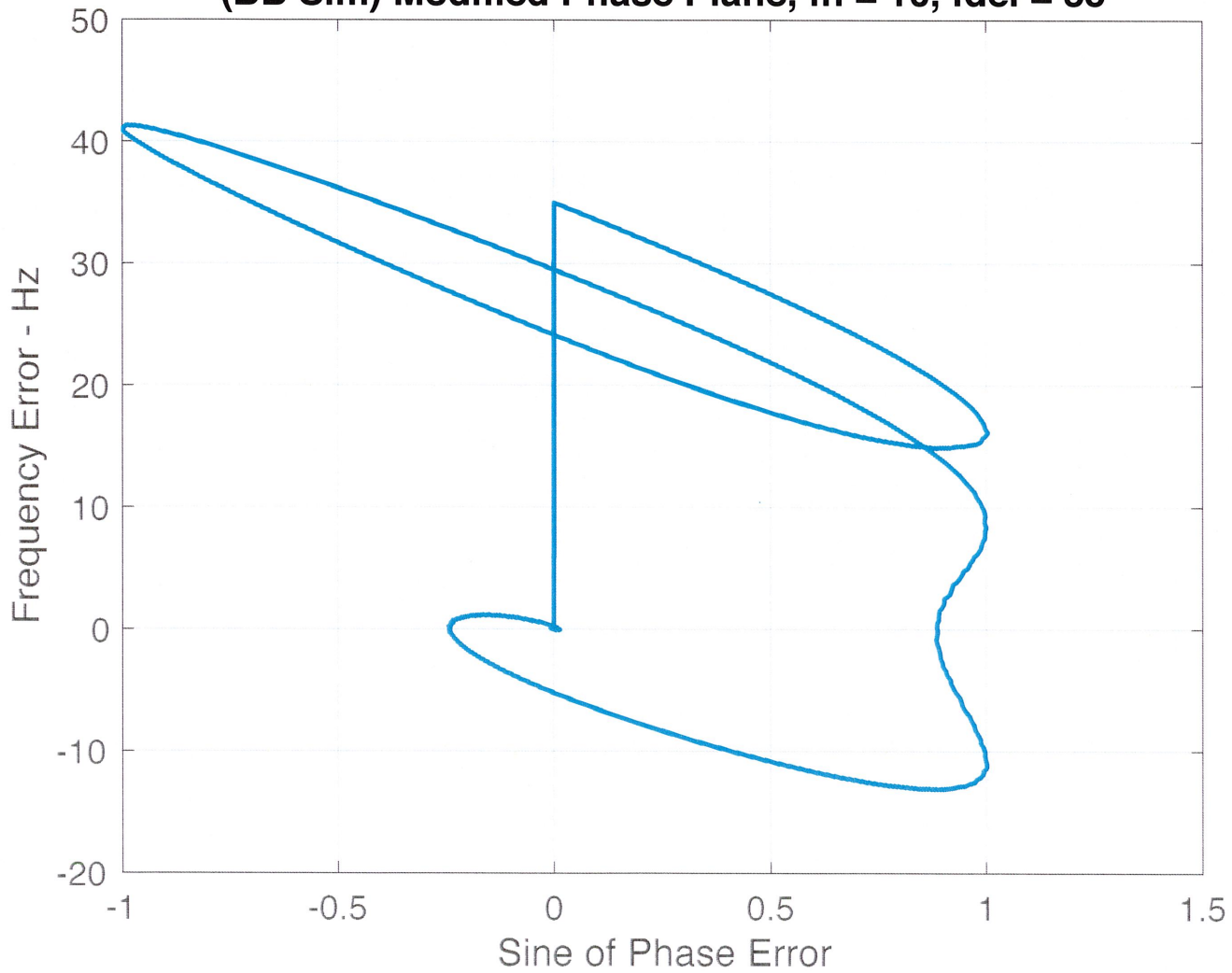
Note that the BB simulation locks after one cycle slip @ $f_n = 10\text{Hz}$ and $f_d = 35\text{Hz}$ while the PB simulation takes 13 cycle slips before locking.

Note that the passband S2 variable can't support the same amplitude, before slipping... (as BB)

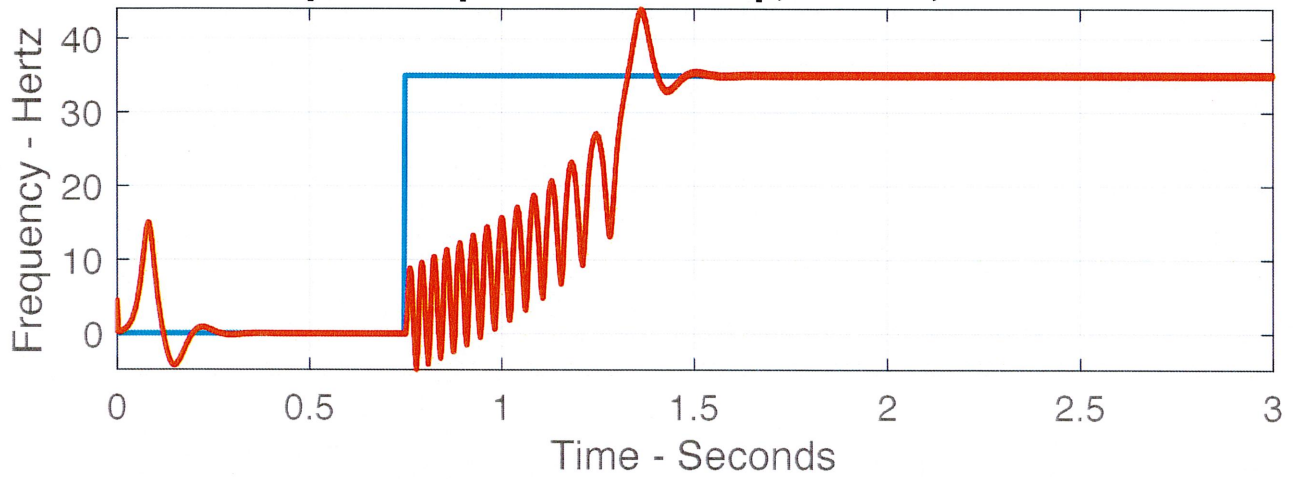
To improve I would work on the PB phase detector.

(e) Run code to see this.

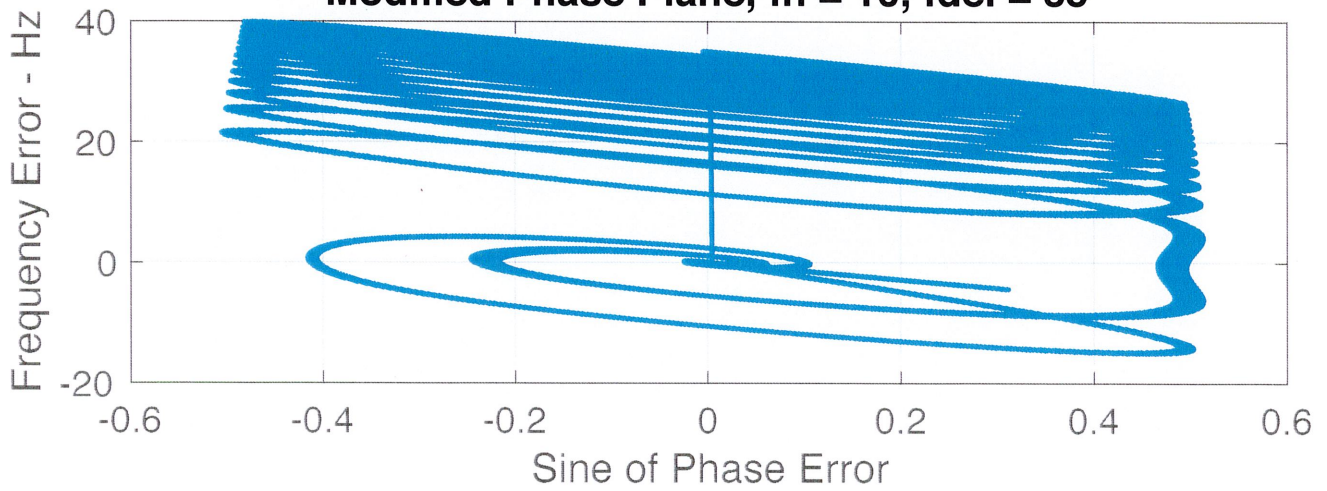
(BB Sim) Modified Phase Plane, $f_n = 10$, $f_{del} = 35$



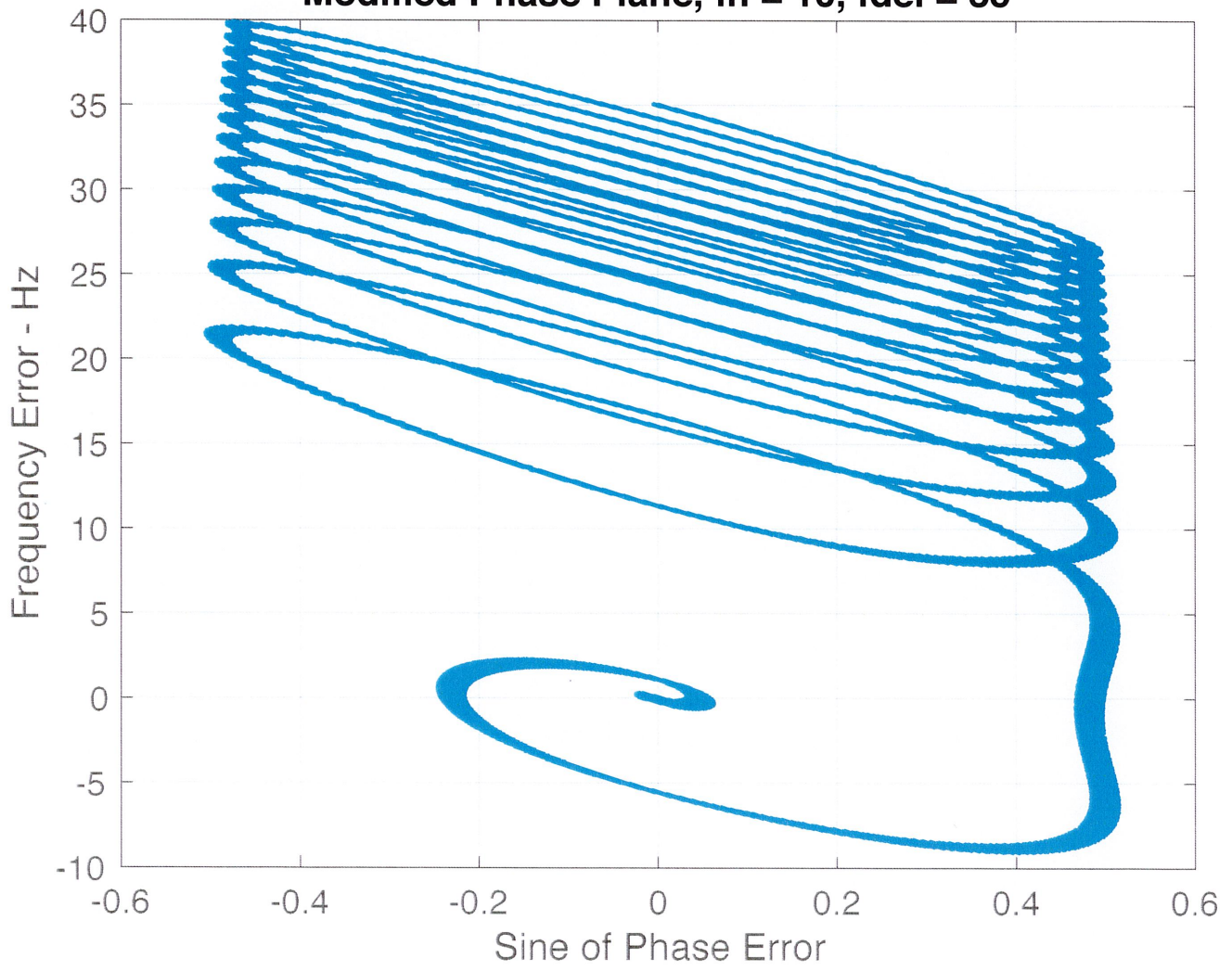
Input Freq. and VCO Freq., $f_n = 10$, $f_{del} = 35$



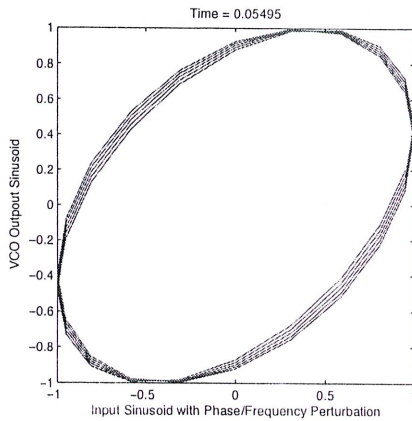
Modified Phase Plane, $f_n = 10$, $f_{del} = 35$



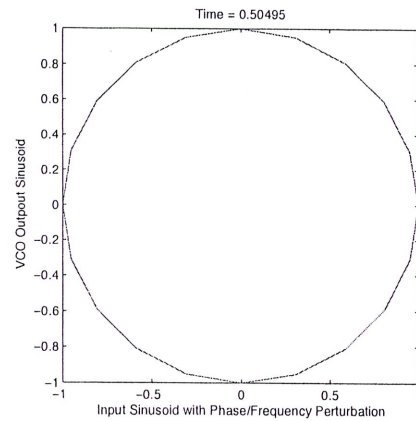
Modified Phase Plane, $f_n = 10$, $f_{del} = 35$



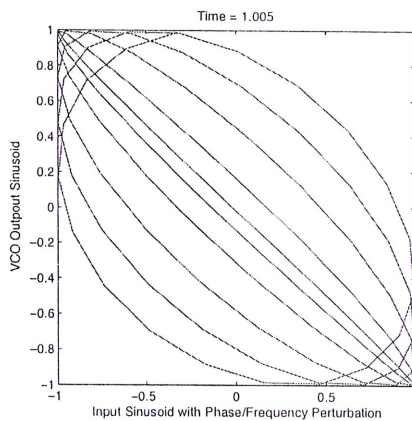
Snapshots of Lissajous Figs.



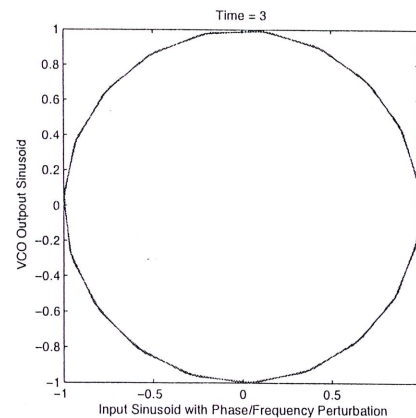
Shortly after simulation start. Not quite locked.



Locked before disturbance



After disturbance. Unlocked and clear frequency offset



Locked again after disturbance

```
%% Passband Simulation of a PLL. Problem 4.
%
% File = THE_P4.m
%
% For Part (a): This is the signal s shown in the code below.
%
% For Part (b): This is directly computed in the simulation loop as the
%   variable svco.
%
% For Part (c): Phase detector is created with a multiplier and the use of
%   the sliding window average over a nominal 5 cycles of the sinusoid.
%
% For Part (d): Loop is closed by connecting the phase detector output with
%   the code from the original computer example. In order to compare to
%   the baseband simulation we have to solve a few problems. First, the
%   baseband variable s1 is not present in the passband simulation.
%   However, we do have available s2 = sin(s1) [exactly, in baseband;
%   approximately in passband]. Therefore, I will modify the baseband
%   simulation to plot the frequency error vs. s2 in order to compare
%   with passband. I will also need to save frequency information in
%   order to create the plot. Finally, there is an initial phase
%   transient present in the passband simulation that does not happen
%   in the baseband simulation. This happens because the low pass
%   filter in the passband phase detector has a long transient in its
%   startup response.
%
% For Part (e): Lissajous implemented in post processor.

clear all;           %Be safe
close all;

%% Beginning of preprocessor
%

% General parameters
fc = 1000;           %Carrier frequency
Ns = 20;            %Oversampling factor
fs = Ns*fc;         %Simulation sampling rate
Tf = 3;             %Simulation length in sec
npts = floor(Tf*fs); %Number of simulation points
t = (0:(npts-1))/fs; %Time vector

% Create the input sinusoid with a phase perturbation
t1 = Tf/4;          %Time to start phase disturbance
n1 = ceil(t1*fs);   %Sample point to start phase disturbance
phidel = 0;         %Size of phase step in rad
fdel = 35;          %Size of freq step in Hz

step_delayed = [zeros(1,n1-1) ones(1,npts - n1 + 1)];

phin = phidel*step_delayed + 2*pi*fdel*( (t - t1) .* step_delayed);
fin = fdel * step_delayed;

s = cos(2*pi*fc*t + phin);

% Parameters of PLL

fn = 10;            %Loop natural frequency in Hz
```

```
zeta = 0.707;           %Loop damping factor

Kt = 4*pi*zeta*fn;     %Loop gain
a = pi*fn/zeta;       %Loop filter parameter

% Phase detector parameters
nc = 5;               %Number of carrier cycles over which to avg
navg = floor(nc*fs/fc);

filt_in_last = 0;
filt_out_last = 0;
vco_in_last = 0;
vco_out = 0;
vco_out_last = 0;

sphierror = zeros(1,npts);
fvco = zeros(1,npts);

% End of preprocessor

%% Beginning of Simulation Loop
%

T = 1/fs;
svco = zeros(1,npts);

for i = 1:npts

    svco(i) = sin(2*pi*fc*t(i) + vco_out);

    % This code implements a simple phase detector
    if i <= navg
        temp = s(1:i) .* svco(1:i);
        s2 = mean(temp);
    else
        temp = s(i - navg:i) .* svco(i - navg:i);
        s2 = mean(temp);
    end

    sphierror(i) = s2;

    s3 = Kt*s2;

    filt_in = a*s3;
    filt_out = filt_out_last + (T/2)*(filt_in + filt_in_last);
    filt_in_last = filt_in;
    filt_out_last = filt_out;
    vco_in = s3 + filt_out;
    vco_out = vco_out_last + (T/2)*(vco_in + vco_in_last);
    vco_in_last = vco_in;
    vco_out_last = vco_out;
    sphierror(i) = s2;
    fvco(i) = vco_in/(2*pi);

end

freqerror = fin - fvco;
```

```
% End of Simulation Loop

%% Beginning of Postprocessor
%

nn = 100;
kk = floor(npts/nn);

%Make a movie
pause on

for i = 0:(kk-1)
    plot(s(1+i*nn:(i+1)*nn),svco(1+i*nn:(i+1)*nn))
    axis square;
    title(['Time = ',num2str(t((i+1)*nn))])
    xlabel('Input Sinusoid with Phase/Frequency Perturbation')
    ylabel('VCO Output Sinusoid')
    pause(0.1)
end

pause off

% End of Postprocessor

%% Postprocessor 2

Tstring1 = ['Input Freq. and VCO Freq., fn = ', num2str(fn), ', fdel = ', num2str(fdel)];

figure

subplot(2,1,1)
plot(t,fin,t,fvco,'LineWidth',2)
title(Tstring1)
xlabel('Time - Seconds')
ylabel('Frequency - Hertz')
grid
set(gca,'FontSize',14)

Tstring2 = ['Modified Phase Plane, fn = ', num2str(fn), ', fdel = ', num2str(fdel)];

subplot(2,1,2)
plot(spherror,freqerror,'LineWidth',2)
title(Tstring2)
xlabel('Sine of Phase Error')
ylabel('Frequency Error - Hz')
grid
set(gca,'FontSize',14)

figure

% In this figure we leave out the beginning phase transient on startup in
% order to better compare with the baseband simulation.

Tstring3 = ['Modified Phase Plane, fn = ', num2str(fn), ', fdel = ', num2str(fdel)];

plot(spherror(n1:end),freqerror(n1:end),'LineWidth',2)
```

```
title(Tstring2)
xlabel('Sine of Phase Error')
ylabel('Frequency Error - Hz')
grid
set(gca, 'FontSize', 14)
```