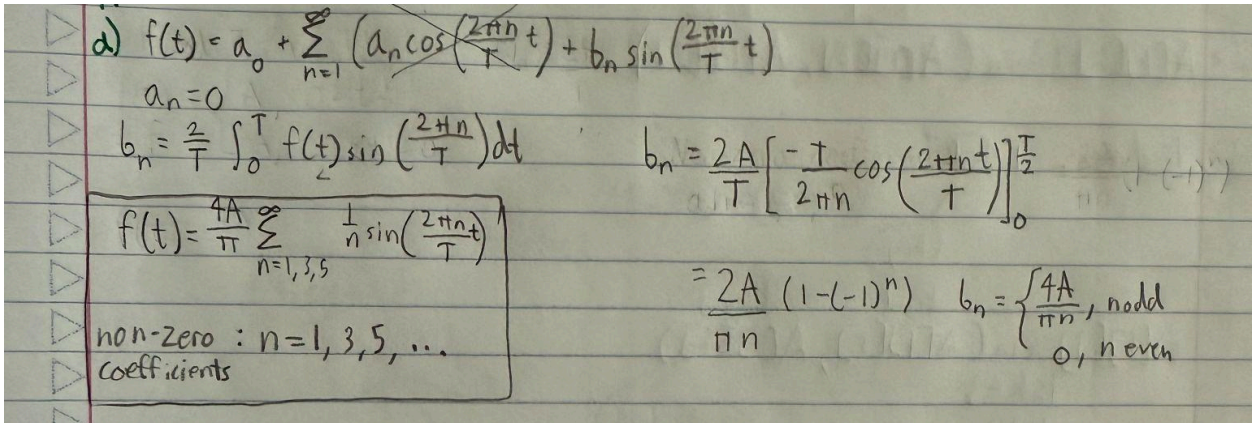
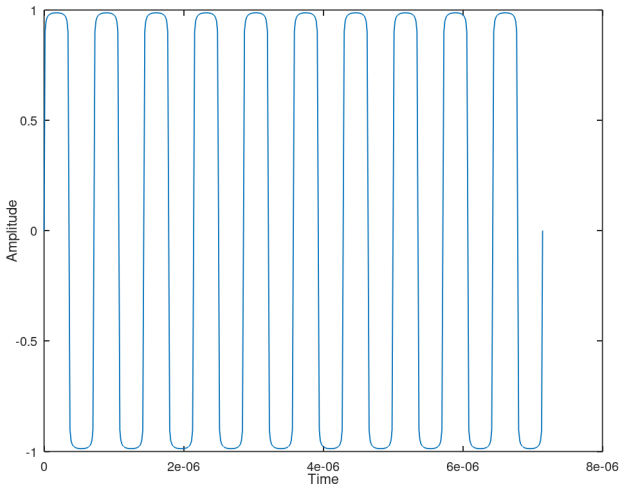
1.

a)



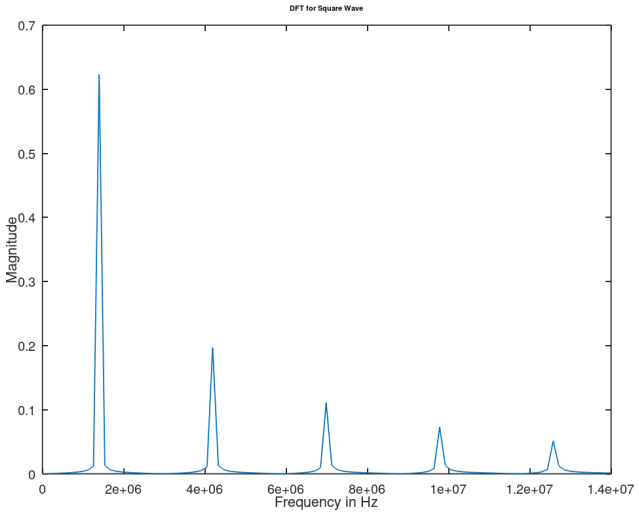
b)

```
A = 1;
f0 = 1.4e6;
T = 1 / f0;
Tmax = 10 * T;
fs = 50 * f0;
dt = 1 / fs;
t = 0:dt:Tmax;
N = 25;
square_wave = zeros(size(t));
for n = 1:2:(2*N - 1)
    bn = (4 * A) / (pi * n);
    square_wave += bn * sin(2 * pi * n * f0 * t);
end
figure;
plot(t, square_wave);
xlabel('Time');
ylabel('Amplitude');
title('Square Wave Fourier);
```



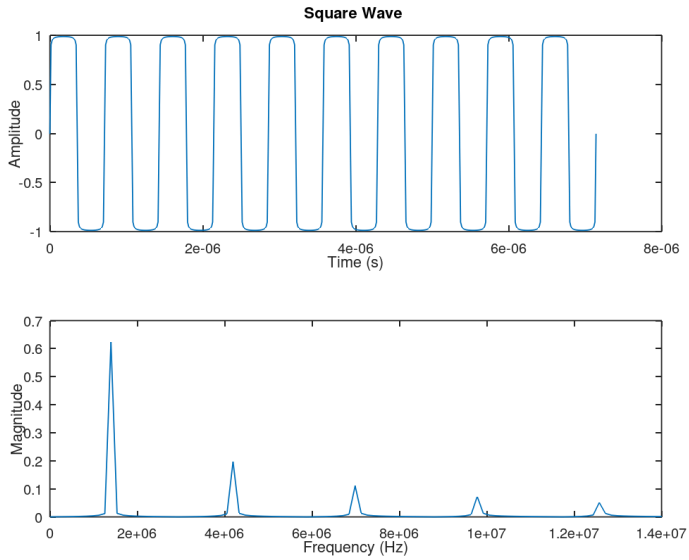
c)

```
Y = fft(square_wave);
N_fft = length(Y);
f = (0:N_fft-1)*(fs/N_fft);

figure;
plot(f, abs(Y)/N_fft);
xlim([0, 10*f0]);
xlabel('Frequency in Hz');
ylabel('Magnitude');
title('DFT for Square Wave');
```



d)

```
A = 1;
f0 = 1.4e6;
T = 1 / f0;
Tmax = 10 * T;
fs = 50 * f0;
dt = 1 / fs;
t = 0:dt:Tmax;
N = 25;
square_wave = zeros(size(t));

for n = 1:2:(2*N - 1)
    bn = (4 * A) / (pi * n);
    square_wave += bn * sin(2 * pi * n * f0 * t);
end
Y = fft(square_wave);
N_fft = length(Y);
f = (0:N_fft-1)*(fs/N_fft);
figure;

subplot(2, 1, 1);
plot(t, square_wave);
xlabel('Time (s)');
ylabel('Amplitude');
title('Square Wave ');
subplot(2, 1, 2);
plot(f, abs(Y)/N_fft);
xlim([0, 10*f0]);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude of DFT’);
```



2. Yes we observe the Gibbs effect in this problem because we are approximating a square wave which has discontinuities in it while the fourier is made with continuous sine waves which do not have any jumps or discontinuities. Therefore at the parts where the square wave switched from high to low there are little ripples.

3.

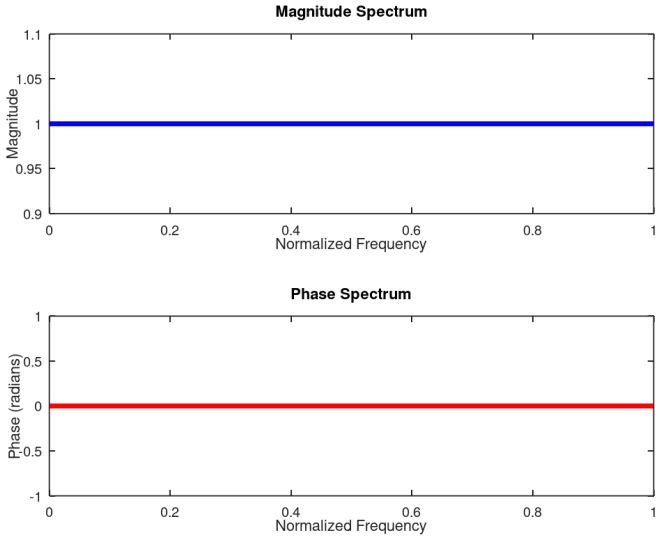
a)

```
N = 1000;
delta = zeros(1, N);
delta(1) = 1;
```

b)

```
Y = fft(delta);
f = (0:N-1);

figure;
subplot(2,1,1);
plot(f, abs(Y));
title('Magnitude of DFT');
```

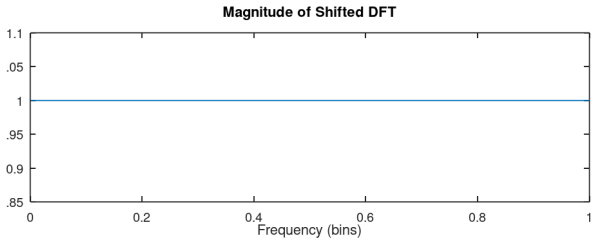
```
xlabel('Frequency bin');
ylabel('|Y[k]|');
subplot(2,1,2);
plot(f, angle(Y));
title('Phase of DFT');
xlabel('Frequency bin');
ylabel('Phase (radians)');
```

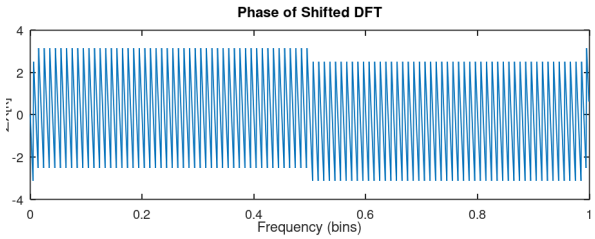


c)

```
delta_shifted = zeros(1, N);
delta_shifted(101) = 1;
Y_shifted = fft(delta_shifted);
figure;

subplot(2,1,1);
plot(f, abs(Y_shifted));
title('Magnitude of DFT for δ[n] at n = 100');
xlabel('Frequency bin');
ylabel('|Y[k]|');
subplot(2,1,2);
plot(f, angle(Y_shifted));
title('Phase of DFT for δ[n] at n = 100');
```

```
xlabel('Frequency bin');
ylabel('Phase (radians)');
```





d)

There is a linear phase in the frequency domain.

```
unwrapped_phase = unwrap(phase_shifted);
figure;
plot(f, unwrapped_phase);
title('Unwrapped Phase]');
xlabel('Frequency');
ylabel('Unwrapped ');
```

e)

```
p = polyfit(f_radians, unwrapped_phase, 1);
slope = p(1);
group_delay = -slope;
fprintf('Estimated group delay: %.2f samples\n', group_delay);
```

```
Estimated group delay: 100.00 samples
```

f) When we add noise to the signal it creates random ripples/jumps in the phase spectrum . Since the group delay is considered a negative slope of the unwrapped phase, the noise causes the estimate to be not as accurate.

4.

a)

```
Fs = 10e6;
f = 100e3;
Tmax = 6e-3;
A = 1.0;

t = 0 : 1/Fs : Tmax;
x = A * sin(2*pi*f*t);
fprintf('Number of samples: %d\n', length(x));
```

b)

```
x(find(x > 0.75)) = 0.75;
x(find(x < -0.75)) = -0.75;
```

c)

```
N = length(x);
X = fft(x);
f_axis = (0:N-1) * Fs / N;
half = floor(N/2);
plot(f_axis(1:half), abs(X(1:half)));
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum of Clipped Signal');
grid on;
```

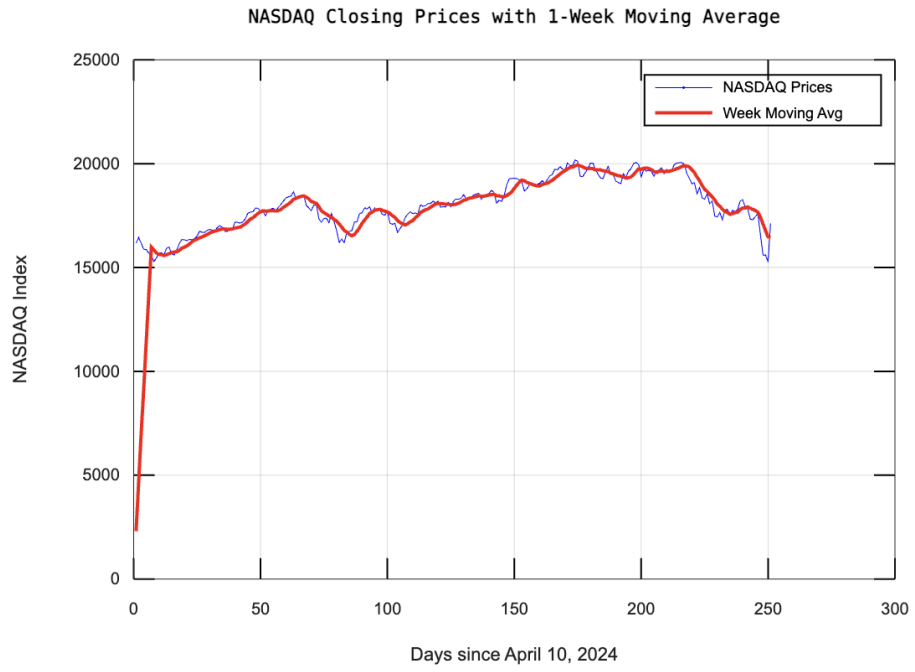
d)

Yes, we observe harmonics at different multiples of the fundamental frequency like 100 kHz, 200 kHz, 300 kHz, and more. When the sine wave is clipped, it behaves similar to a square wave which has harmonics at odd increments (3,5,7,...). Clipping creates non-linear distortion in the signal and gives it these harmonic ripples.

5.

a) b) c)

```
data = load('nasdaq.txt');
days = data(:, 1);
values = data(:, 2);

window = 7;
weights = ones(window, 1) / window;
moving_avg = filter(weights, 1, values);

plot(days, values, 'b', 'DisplayName', 'NASDAQ Prices');
hold on;
plot(days, moving_avg, 'r', 'LineWidth', 2, 'DisplayName', 'Week
Moving Avg');
xlabel('Days since April 10, 2024');
ylabel('NASDAQ Index');
title('NASDAQ Closing Prices with 1-Week Moving Average');
legend show;
grid on;
```



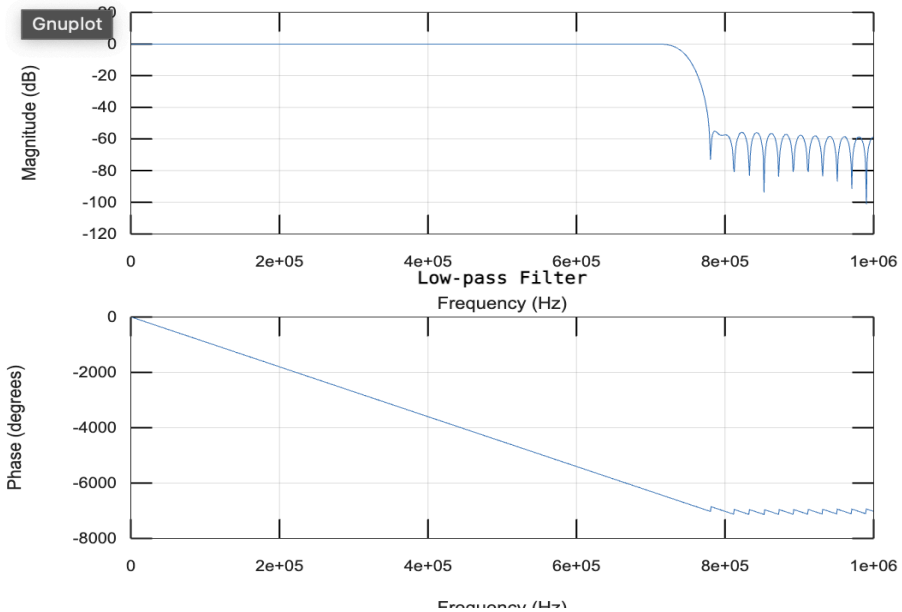
d) The average filter doesn't precisely capture the rapid changes in economic policy because in order to reduce the noise of the graph, it focuses on longer term trends in the data instead of short fluctuations.

e) The lag of the moving average filter is about half the window size so it will be equal to 3. Therefore the moving avg filter has a lag of about 3 days.

6.

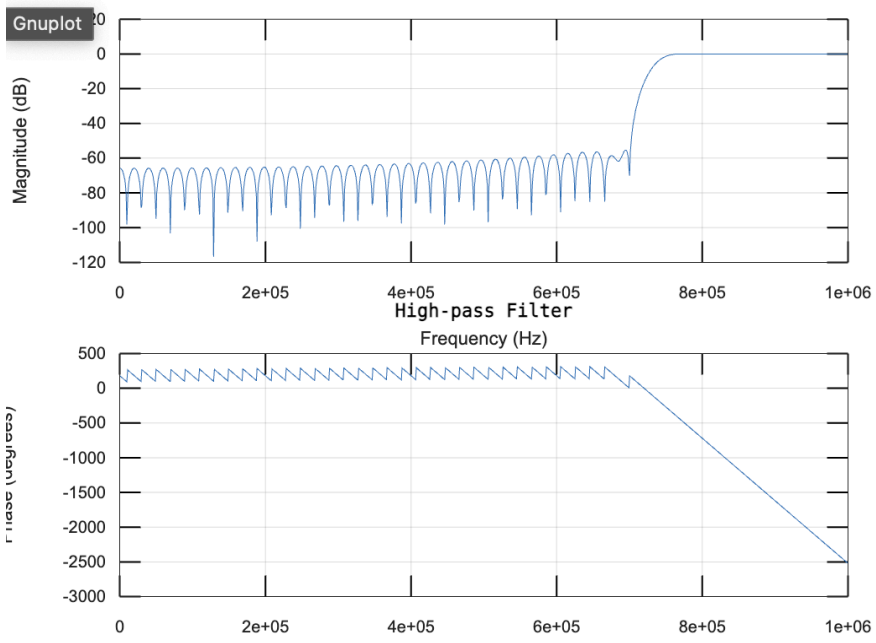
a)

```
Fs = 2e6;
fc = 745e3;
M = 101;

n = 0:M-1;
mid = (M-1)/2;
fcn = fc / (Fs/2);
h = sin(pi*fcn*(n-mid)) ./ (pi*(n-mid));
h(mid+1) = fcn;
w = hamming(M)';
hlow = h .* w;

freqz(hlow, 1, 1024, Fs);
title("Low-pass Filter");
```



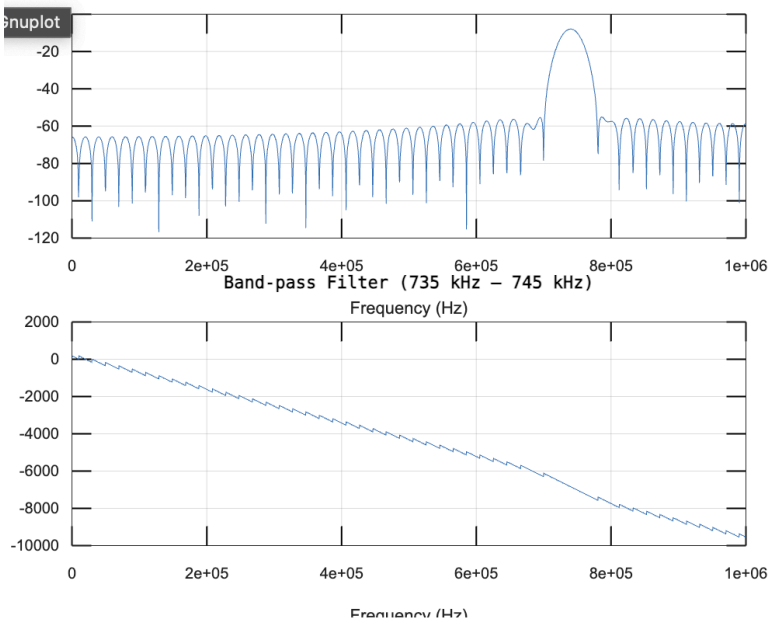
b)

```
fc2 = 735e3;
fcn2 = fc2 / (Fs/2);

h2 = sin(pi*fcn2*(n-mid)) ./ (pi*(n-mid));
h2(mid+1) = fcn2;

w2 = hamming(M)';
hlow2 = h2 .* w2;

hhigh = -hlow2;
hhigh(mid+1) = 1 - hlow2(mid+1);

freqz(hhigh, 1, 1024, Fs);
title("High-pass Filter ");
```



c)

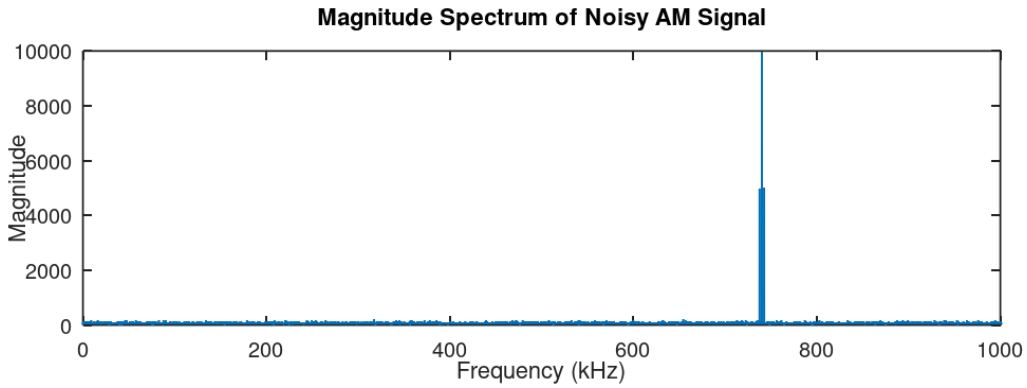
```
hband = conv(hlow, hhigh);
freqz(hband, 1, 2048, Fs);
title("Band-pass Filter (735 kHz – 745 kHz)");
```



d)

```
duration = 0.01;
t = 0:1/Fs:duration-1/Fs;
f_audio = 2.5e3;
audio = sin(2*pi*f_audio*t);

f_carrier = 740e3;
carrier = cos(2*pi*f_carrier*t);
am_signal = (1 + audio) .* carrier;
noise = 0.5 * randn(size(t));
noisy_signal = am_signal + noise;

N = length(t);
Y = fft(noisy_signal);
f = linspace(0, Fs, N);

plot(f/1e3, abs(Y));
xlabel("Frequency (kHz)");
ylabel("Magnitude");
```

```
title("Magnitude Spectrum of Noisy AM Signal");
xlim([0 Fs/2/1e3]);
```

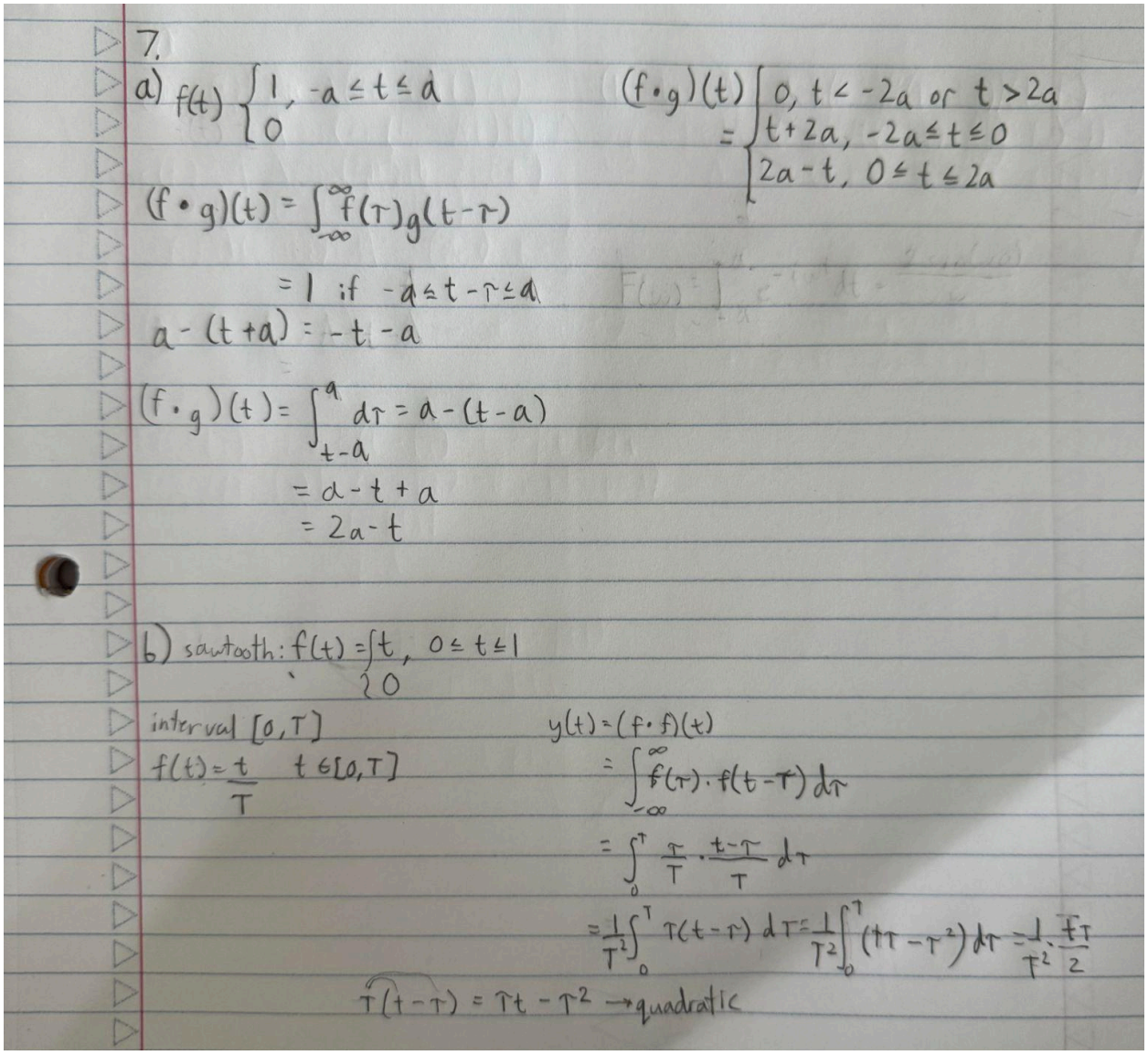


e)

```
filtered_signal = conv(noisy_signal, hband, 'same');
Yf = fft(filtered_signal);

plot(f/1e3, abs(Yf));
xlabel("Frequency (kHz)");
ylabel("Magnitude");
title("Filtered Signal Spectrum (Band-pass around 740 kHz)");
xlim([0 Fs/2/1e3]);
```

7.

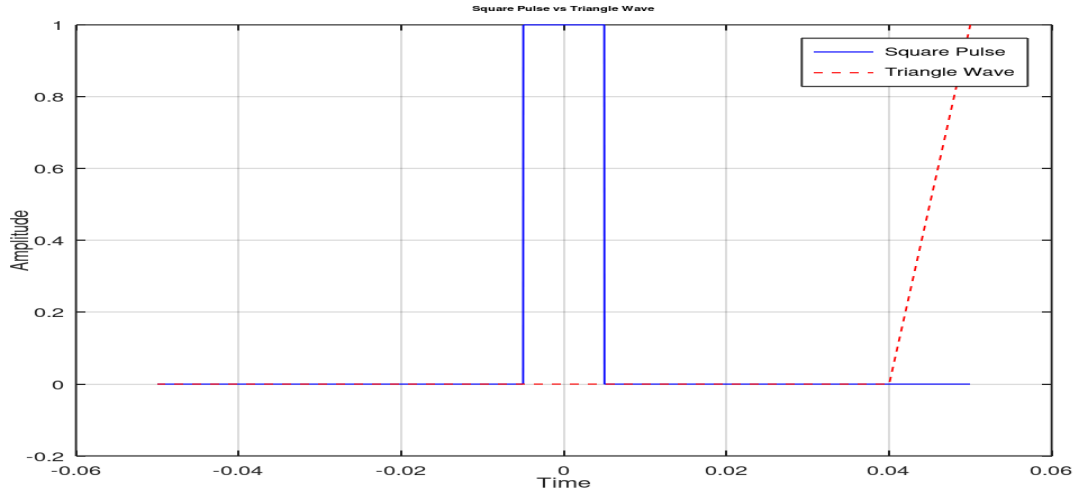
a) b)



Bonus:

```
fs = 44100;
duration = 0.1;
t = linspace(-0.05, 0.05, fs * duration);
width = 0.01;
square_pulse = double(abs(t) < width/2);

N = length(square_pulse) * 2;
```

```
F1 = fft(square_pulse, N);
F2 = fft(square_pulse, N);
triangle_wave = ifft(F1 .* F2);
triangle_wave = real(triangle_wave);
triangle_wave = triangle_wave(1:length(t));
square_pulse = square_pulse / max(abs(square_pulse));
triangle_wave = triangle_wave / max(abs(triangle_wave));

figure;
plot(t, square_pulse, 'b', t, triangle_wave, 'r--');
legend('Square Pulse', 'Triangle Wave');
title('Square Pulse vs Triangle Wave');
xlabel('Time');
ylabel('Amplitude');
grid on;

disp("square pulse");
sound(square_pulse, fs);
pause(duration + 0.5);
disp("triangle wave");
sound(triangle_wave, fs);
pause(duration + 0.5);
```



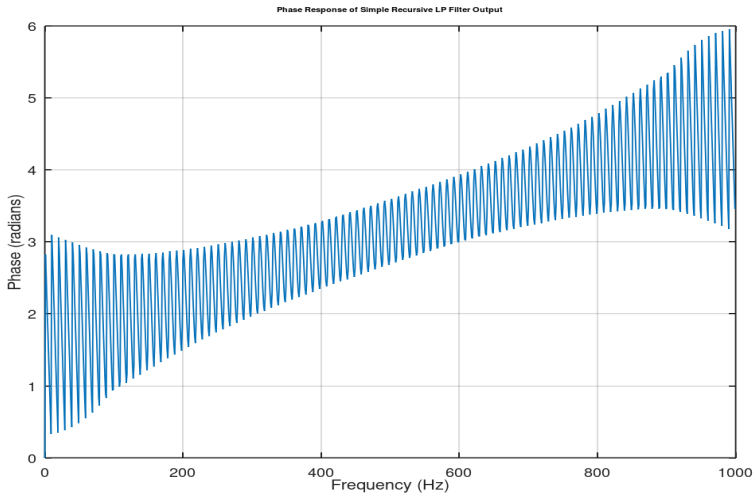
8.

a)

```
fs = 1000;
duration = 1;
t = 0:1/fs:duration-1/fs;
cutoff_freq = 100;
step_start_time = 0.1;
step_start_index = round(step_start_time * fs) + 1;
step_pulse = zeros(size(t));
step_pulse(step_start_index:end) = 1;
RC = 1 / (2 * pi * cutoff_freq);
dt = 1 / fs;
alpha = dt / (RC + dt);

output_signal = zeros(size(step_pulse));
y_prev = 0;
for n = 1:length(step_pulse)
  output_signal(n) = (1 - alpha) * step_pulse(n) + alpha * y_prev;
  y_prev = output_signal(n);
end
disp("Filter coefficient (alpha):");
disp(alpha);
```

b)

```
N = length(output_signal);
fft_output = fft(output_signal);
f = (0:N-1) * fs / N;
phase_output = angle(fft_output);
figure;
plot(f, unwrap(phase_output));
title('Phase Response of Simple Recursive LP Filter Output');
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
grid on;
```



c)

The phase curves at higher frequencies so it appears to be non-linear.

9.

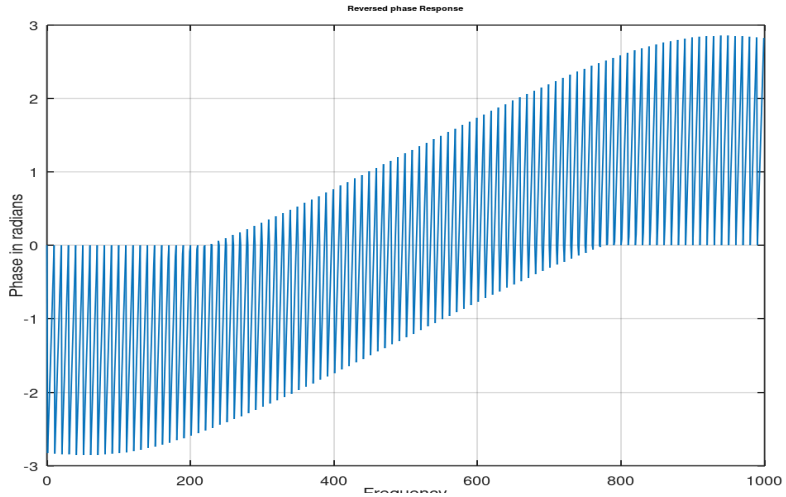
a)

```
reversed_step_pulse = flip(step_pulse);
```

b)

```
output_reversed = zeros(size(reversed_step_pulse));
y_prev_reversed = 0;
for n = 1:length(reversed_step_pulse)
  output_reversed(n) = (1 - alpha) * reversed_step_pulse(n) + alpha *
y_prev_reversed;
  y_prev_reversed = output_reversed(n);
end
N_reversed = length(output_reversed);
fft_output_reversed = fft(output_reversed);
f_reversed = (0:N_reversed-1) * fs / N_reversed;
phase_output_reversed = angle(fft_output_reversed);

figure;
plot(f_reversed, unwrap(phase_output_reversed));
title('Reversed phase Response');
```

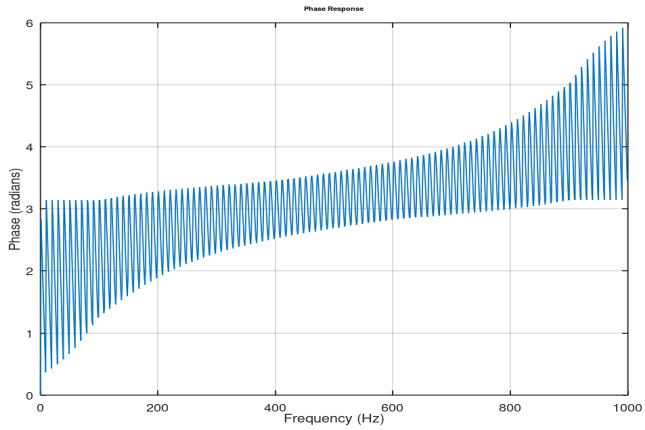
```
xlabel('Frequency');
ylabel('Phase in radians');
grid on;
```



c)

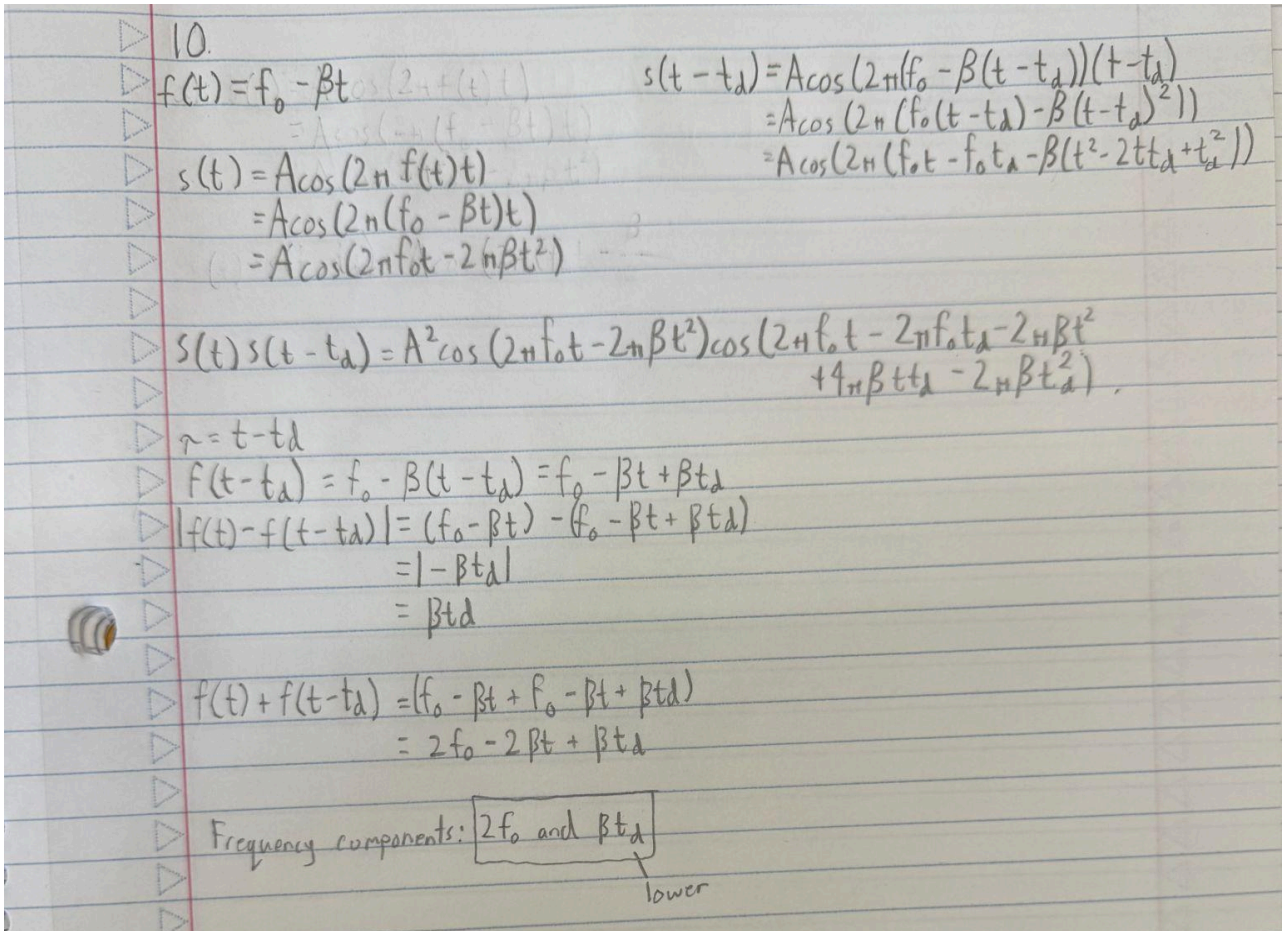
```
% First pass
output_forward = zeros(size(step_pulse));
y_prev_forward = 0;
for n = 1:length(step_pulse)
  output_forward(n) = (1 - alpha) * step_pulse(n) + alpha * y_prev_forward;
  y_prev_forward = output_forward(n);
end
% Reverse output of first
reversed_output_forward = flip(output_forward);
% Second pass
output_backward = zeros(size(reversed_output_forward));
y_prev_backward = 0;
for n = 1:length(reversed_output_forward)
  output_backward(n) = (1 - alpha) * reversed_output_forward(n) + alpha *
y_prev_backward;
  y_prev_backward = output_backward(n);
end
% Reverse output of second
final_output = flip(output_backward);
N_final = length(final_output);
fft_final_output = fft(final_output);
f_final = (0:N_final-1) * fs / N_final;
phase_final_output = angle(fft_final_output);
```

```
plot(f_final, unwrap(phase_final_output));
title('Phase Response');
xlabel('Frequency (Hz)');
ylabel('Phase (radians)');
grid on;
```



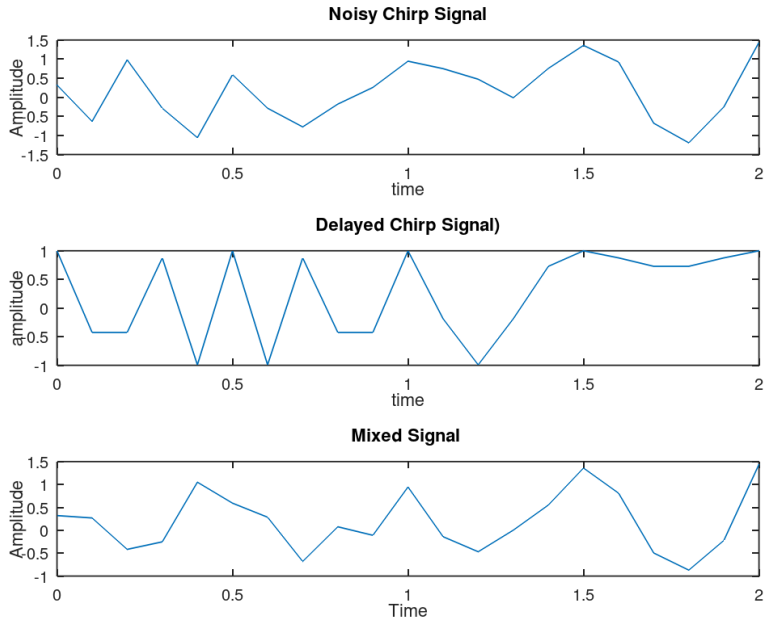
10. Bonus

a)



b)

```
fs = 10e6;
t = 0:1/fs:duration;
beta = 2e12;
f0 = 5e6;
td = 0.5e-6;
A = 1;
ft = f0 - beta * t;
st = A * cos(2 * pi * ft .* t);

noise_level = 0.5;
noise = noise_level * randn(size(st));
noisy_st = st + noise;

ft_delayed = f0 - beta * (t - td);
st_delayed = A * cos(2 * pi * ft_delayed .* (t - td));
mixed_signal = noisy_st .* st_delayed;
figure;
subplot(3, 1, 1);
plot(t * 1e6, noisy_st);
title('Noisy Chirp Signal');
xlabel('time');
ylabel('Amplitude');

subplot(3, 1, 2);
plot(t * 1e6, st_delayed);
title('Delayed Chirp Signal)');
xlabel('time');
ylabel('amplitude');

subplot(3, 1, 3);
plot(t * 1e6, mixed_signal);
title('Mixed Signal ');
xlabel('Time');
ylabel('Amplitude');
```


Noisy Chirp Signal
Amplitude
time
Delayed Chirp Signal)
amplitude
time
Mixed Signal
Amplitude
Time