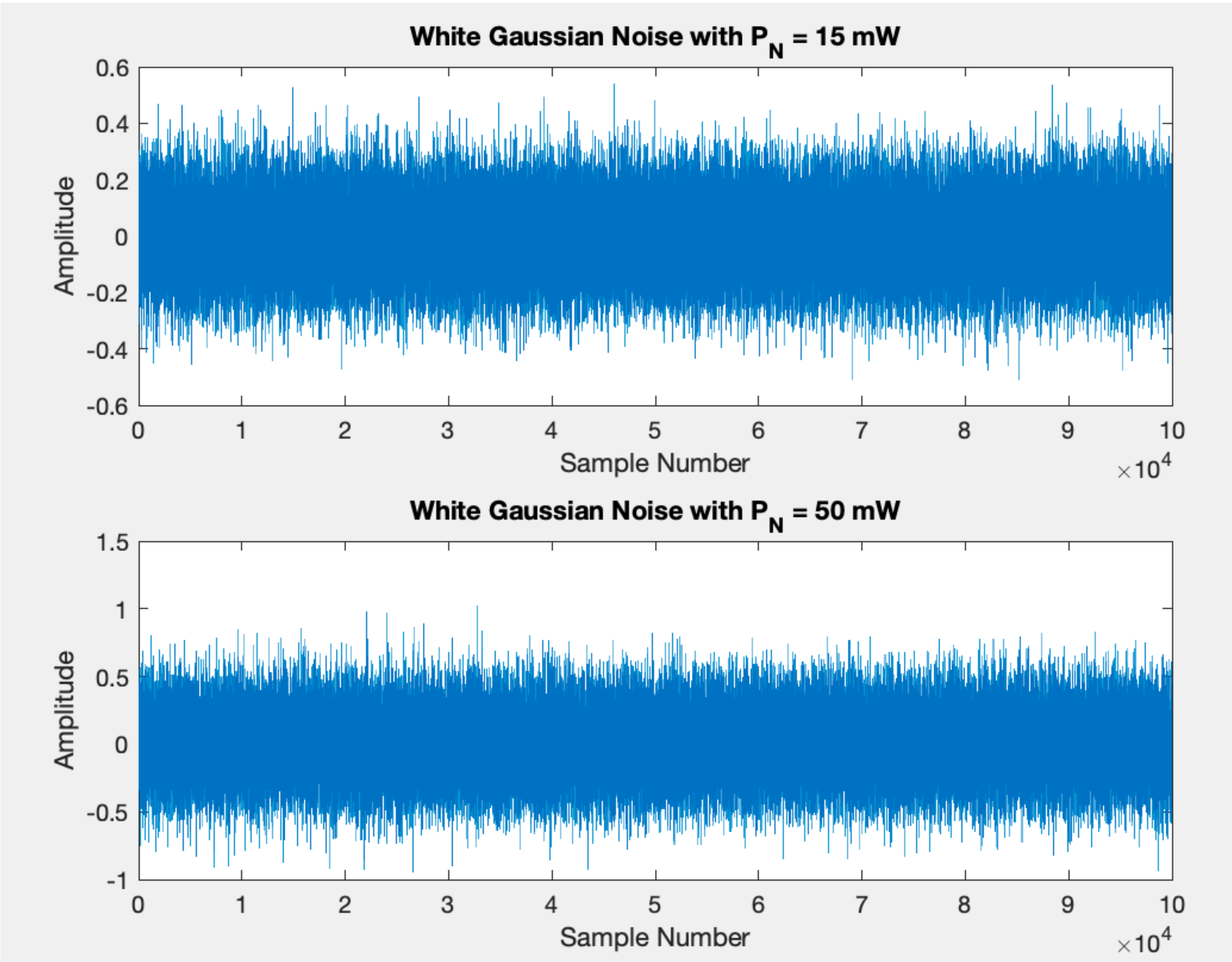
ECE 4700
Final Computer Project
David Baron-Vega – GF7068

PART 1:




```
>> ECE4700_FinalProjectTesting
Actual power of first noise signal: 0.014906 W
Actual power of second noise signal: 0.05018 W
```

The actual power values will vary each time the code is ran, since it is a random process. However, the more samples used for the plotting will decrease the standard deviation of the each value, yielding approximate values closer to the expected.
100,000 samples were used in the results shown above, which was also beneficial for more accurate results in proceeding steps.

Code used for Part 1:

Editor – /Users/david/Desktop/ECE 4700/ECE4700_ComputerProject_Final.m

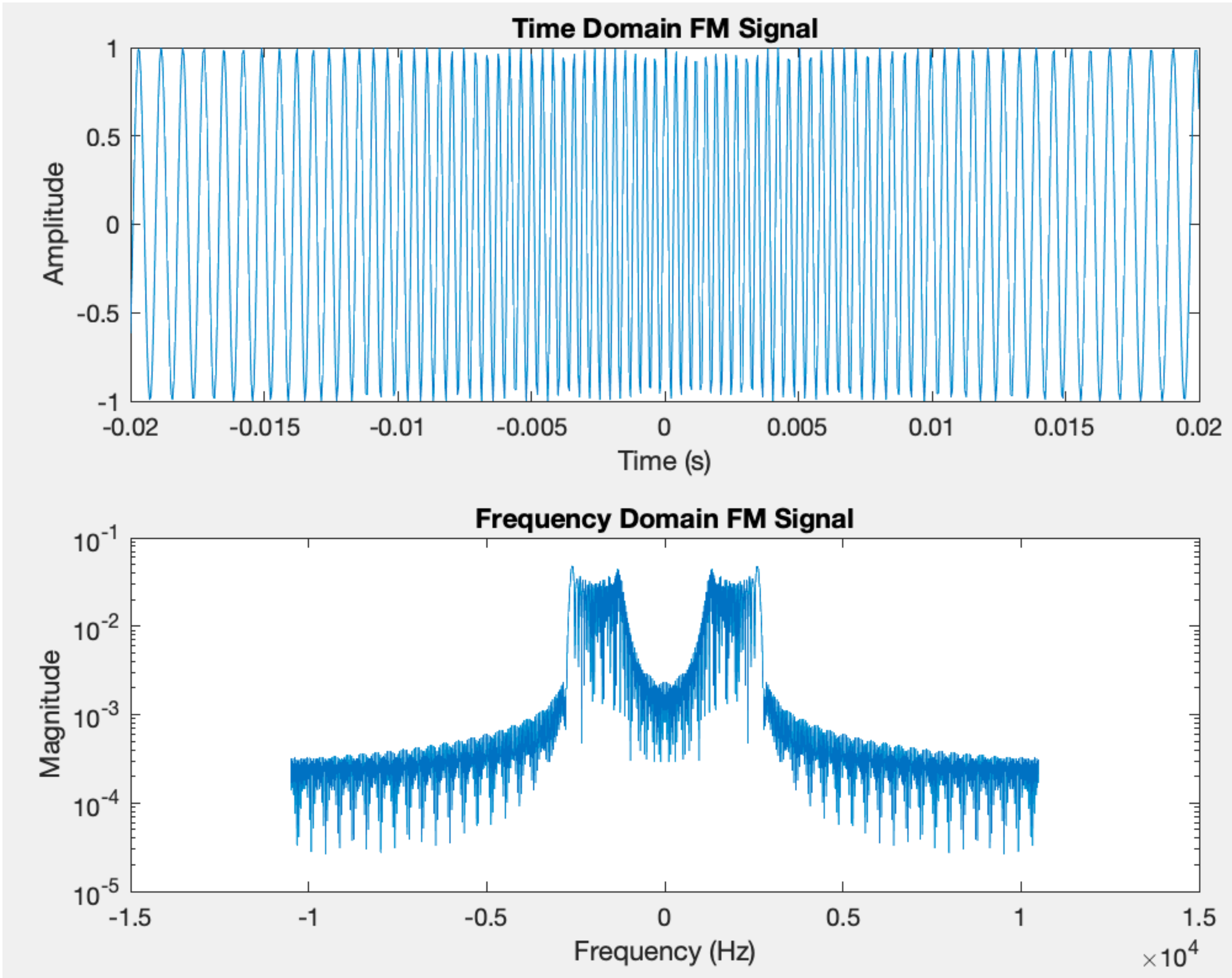
ECE4700_ComputerProject_Final.m

```matlab

%ECE 4700 - Computer Project - David Baron-Vega
%Access ID: GF7068

%Part 1: Generating White Gaussian Noise at specified power levels.

% Noise power levels in mW
P_N1 = 15e-3;  % 15 mW
P_N2 = 50e-3;  % 50 mW

%Number of samples Needs to be tuned to get an accurate and precise
%result for power.
N = 100000; %can be tuned for more/less accurate and noisy signals.
%If N is bigger, standard deviation of output noise becomes much smaller


%Generating the WGN samples
n1 = sqrt(P_N1) * randn(N, 1);
n2 = sqrt(P_N2) * randn(N, 1);

%Ploting the WGN noise samples
figure;
subplot(2,1,1);
plot(n1);
title('White Gaussian Noise with P_N = 15 mW');
xlabel('Sample Number');
ylabel('Amplitude');

subplot(2,1,2);
plot(n2);
title('White Gaussian Noise with P_N = 50 mW');
xlabel('Sample Number');
ylabel('Amplitude');


%Computing the actual average power of the two noise signals
actual_power_n1 = mean(n1.^2);
actual_power_n2 = mean(n2.^2);

%Displaying the computed power values
disp(['Actual average power of first noise signal: ', num2str(actual_power_n1), ' Watts']);
disp(['Actual average power of second noise signal: ', num2str(actual_power_n2), ' Watts']);

```

It was important to calculate the actual power as I did above. Because WGN is a random, ergodic process, it's ensemble mean is equal to its overall time average. Because we are generating random noise at a mu value of 0, the noise power is equal to the variance of the noise signal. This is why we square n1 and n2 above.

Part 2 Results:

Generating time-domain FM-modulated signal and frequency domain of signal using fft:

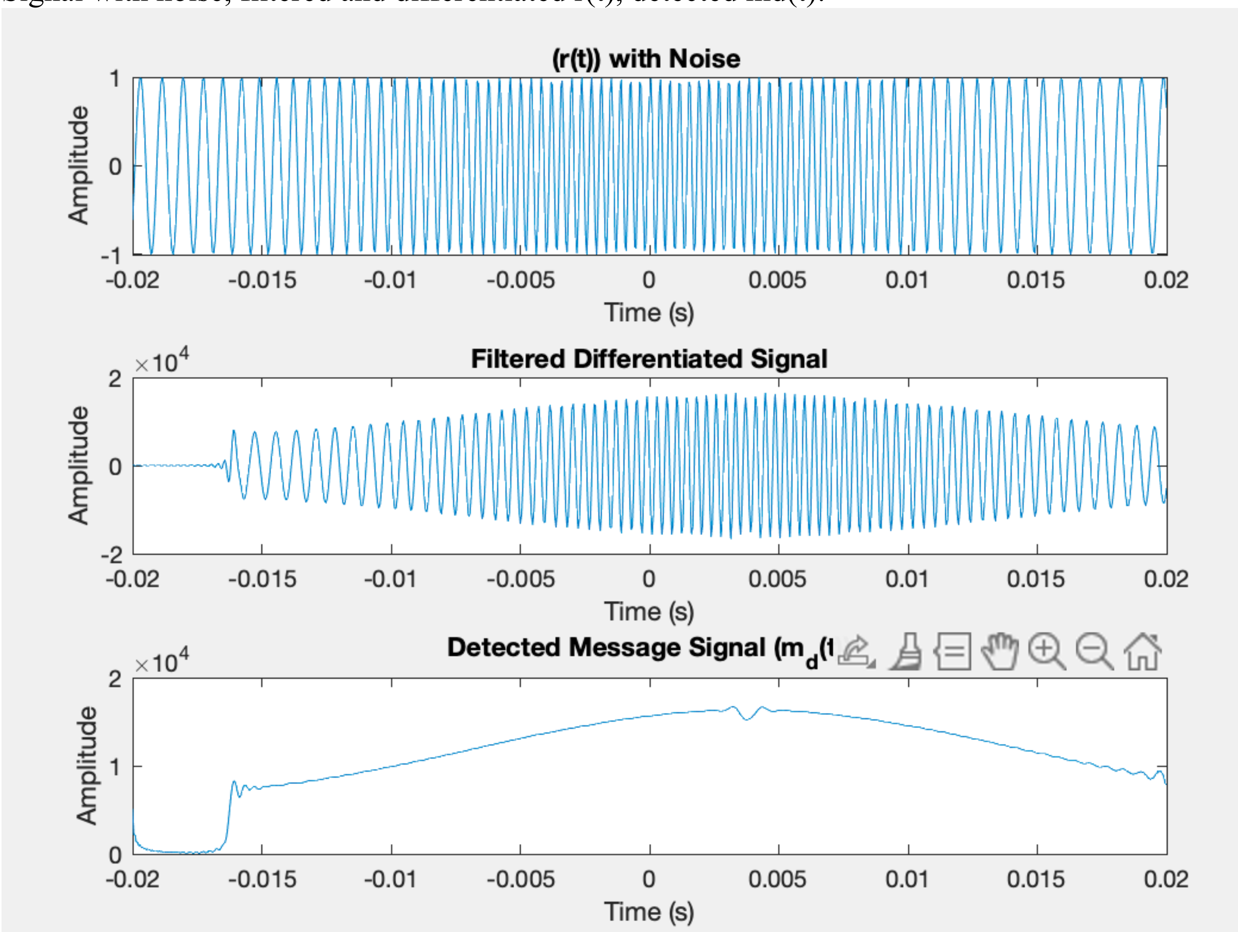

Time Domain FM Signal
Amplitude
1
0.5
0
-0.5
-1
-0.02
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
0.02
Time (s)
Frequency Domain FM Signal
Magnitude
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
-1.5
-1
-0.5
0
0.5
1
1.5
Frequency (Hz)
$\times 10^4$

Code used:

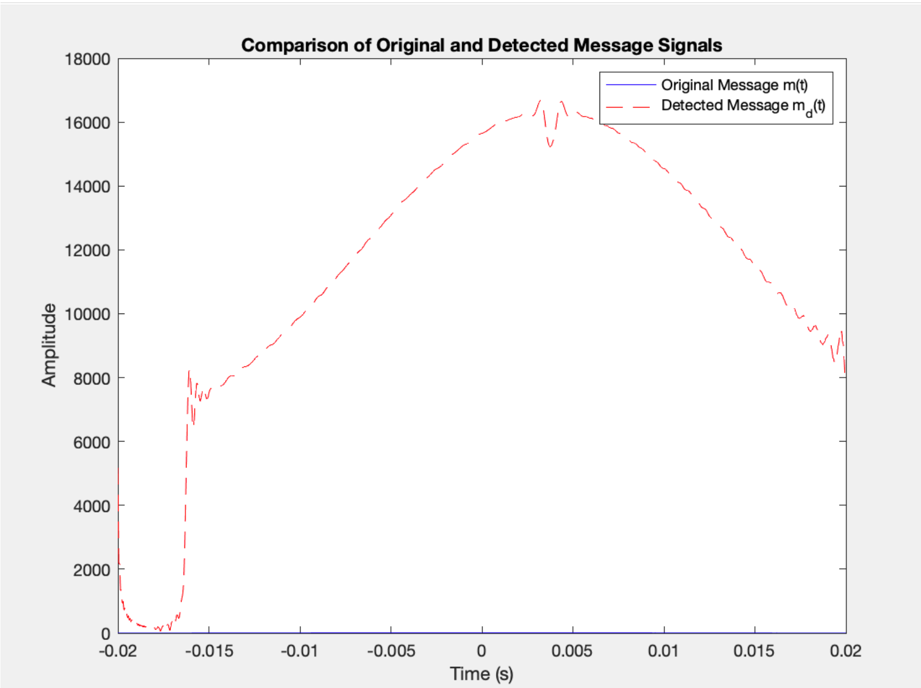
```
%%% PART 2: FM Modulation:

%Setting the signal's parameters
Ac = 1; %Carrier amplitude
kf = 200; %Frequency sensitivity (Hz/Volt)
fs = 21000; %Sampling frequency = 21KHz, much greater than fs > 2*B requirement.
fc = 1068; % 1000+ (my last 3 digits) are 068
t = -0.02:1/fs:0.02; %Time vector
ts = 1/fs; %Sampling interval

%Defining the band-limited message signal
mt = (2*sin(2*pi*20*t).^2)./((20*pi*t).^2);
mt(t == 0) = 1; %Correcting the sinc function at t = 0
figure;
plot(mt)
title('Message Signal m(t)');
xlabel('Time (s)');
ylabel('Amplitude');

%Integral of m(t) for FM
integral_mt = cumsum(mt)*ts;

%Creating the FM signal s(t)
st = Ac*cos(2*pi*fc*t + 2*pi*kf*integral_mt)

%Time domain results
figure;
subplot(2,1,1);
plot(t, st);
title('Time Domain FM Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Frequency domain representation
Lfft = 2^(nextpow2(length(t)) + 1); %Increasing the FFT resolution for more detail
S_fre = fft(st, Lfft);
S_fre = fftshift(S_fre)/Lfft; %Scaling the FFT output
freq = (-Lfft/2:Lfft/2-1)/(Lfft*ts); %Correcting frequency vector calculation, middle of graph is 0.

%Frequency domain results
subplot(2,1,2);
plot(freq, abs(S_fre));
title('Frequency Domain FM Signal');
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%Setting the y-axis to use a logarithmic scale to better visualize the FFT output
set(gca, 'YScale', 'log');
```

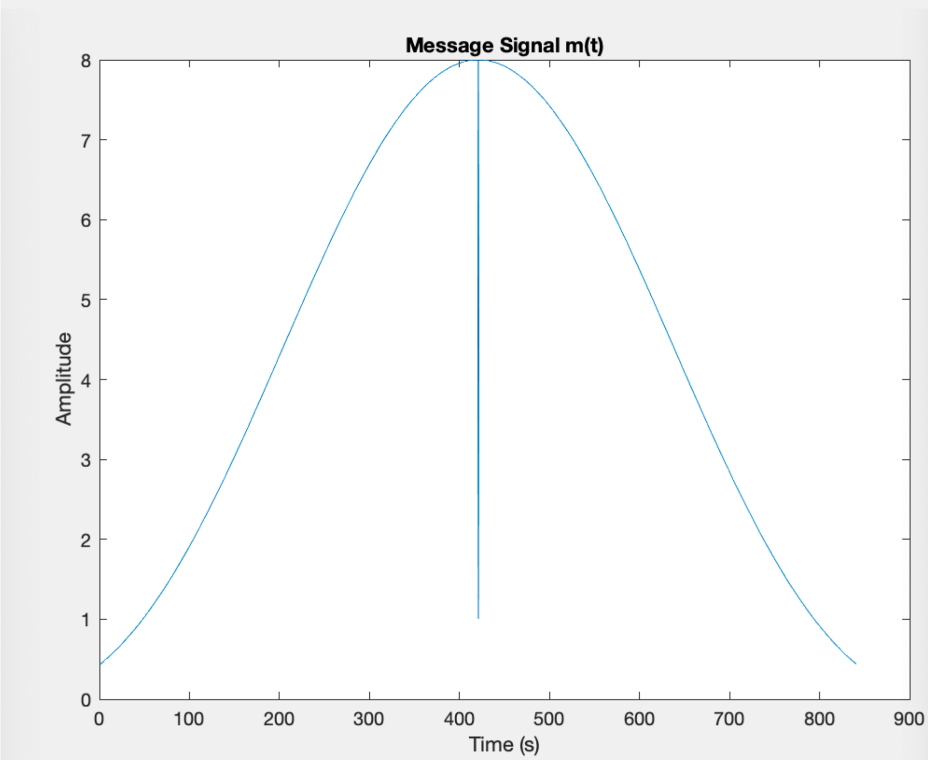
Comments:
Using the assigned value of kf, amplitude of 1, and an fc that is unique to my access id.
A large fs and very small ts was helpful in producing clear results. It was also important to set the origin of the message signal to a non-zero value to produce the frequency-domain representation accurately.

PART 3:
Signal with noise, filtered and differentiated r(t), detected md(t):



Comparing the original and detected:
The Amplitude of the detected is much, much, bigger! But the shape is fairly well retrieved.
Side by side comparison:
Ofcourse, the frequency of the retrieved signal is going to be higher. When this signal is differentiated, even after filtering performed before and after differentiated, the FM will come through as amplitude variations, so I believe this is what we would expect to see. If we needed the retrieved signal to be of lower amplitude, we would have to apply more filtering/limiting.


Comparison of Original and Detected Message Signals
Original Message m(t)
Detected Message m_d(t)
Amplitude
Time (s)
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
-0.02
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
0.02


Message Signal m(t)
Amplitude
Time (s)
8
7
6
5
4
3
2
1
0
0
100
200
300
400
500
600
700
800
900

Code:

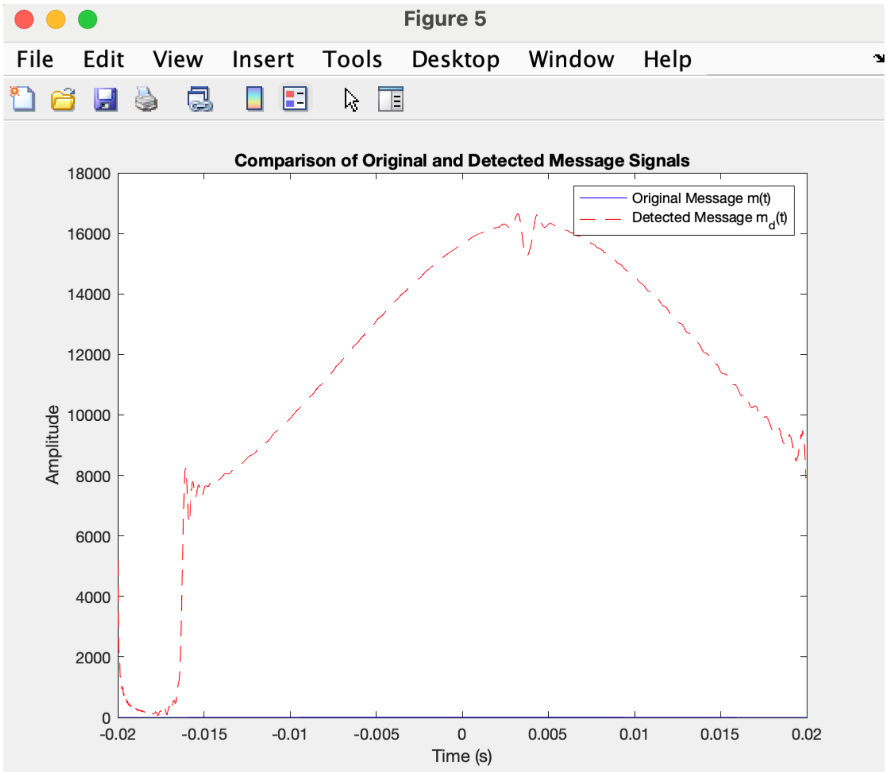
```
%%% PART 3: Noise in FM Channel and FM Demodulation


%Number of samples for noise should match the FM signal samples
N = length(t);
noise_power = 50e-3; %50 mW
n_t = sqrt(noise_power/fs) * randn(1,N); %Regenerating Noise signal:

%Creating the signal r(t):
r_t = st + n_t

%Time domain plot of r(t)
figure;
subplot(3,1,1);
plot(t, r_t);
title('(r(t)) with Noise ');
xlabel('Time (s)');
ylabel('Amplitude');
```

```
%Applying a limiter (optional)
%r_t = limiter_function(r_t);
%Applying a bandpass filter before differentiation
nyquist_freq = fs / 2;

%Lower and upper bounds for the bandpass filter must be strictly between 0 and 1!
%We can set a small value close to zero for the lower bound
lower_bound_normalized = (10/fs); %A small value close to zero but not zero
upper_bound_normalized = (4000/nyquist_freq); %Upper bound normalized and less than 1

bpf_before_diff = fir1(80, [lower_bound_normalized upper_bound_normalized]);
r_t_filtered = filter(bpf_before_diff, 1, r_t);




%Differentiate the signal, Check the orientation of the signal vector to
%add the necessary dimension,

%Kept bugging out here, idk why I need an extra zero in the array for this
%to differentiate tbh, one dimension was getting lost when differentiating
%I think.


diff_r_t = [diff(r_t_filtered), 0]; %Concatinate zero in the correct orientation

if isrow(r_t)
   diff_r_t = [diff(r_t_filtered), 0]
else
    diff_r_t = [diff(r_t_filtered); 0];
end
diff_r_t = diff_r_t / ts;           %Scale by the sampling interval



%Applying another bandpass filter after differentiation
%Lower and upper bounds for the BPF between 0-1

lower_bound_normalized = (10/fs);
upper_bound_normalized = (4000/nyquist_freq);

bpf_after_diff = fir1(80, [lower_bound_normalized upper_bound_normalized]);
filtered_diff_r_t = filter(bpf_after_diff, 1, diff_r_t);


%Envelope detection to retrieve m_d(t)
md_t = abs(hilbert(filtered_diff_r_t));
```

```
%Time domain plot of the filtered differentiated signal
subplot(3,1,2);
plot(t, filtered_diff_r_t);
title('Filtered Differentiated Signal');
xlabel('Time (s)');
ylabel('Amplitude');

%Time domain plot of the detected message signal m_d(t)
subplot(3,1,3);
plot(t, md_t);
title('Detected Message Signal (m_d(t))');
xlabel('Time (s)');
ylabel('Amplitude');

%Comparing m_d(t) with the original message signal m(t)
figure;
plot(t, mt, 'b', t, md_t, 'r--');
legend('Original Message m(t)', 'Detected Message m_d(t)');
title('Comparison of Original and Detected Message Signals');
xlabel('Time (s)');
ylabel('Amplitude');
```

I had to trial and error this part quite a bit with the filters, until my retrieved message resembled my original message better. Still, I couldn't lower the amplitude of the retrieved message enough as I would have liked. After differentiation, the vector used to plot the retrieved signal did not match the original length of the message, so I had to manually add values to the array in order to filter and then compute the retrieved message.

Part 4:
Comparing the original recovered message with new recovered messages with varying Kf values:



The comparison previously displays the comparison better, as the amplitude of the retrieved signal is much larger. The have the most similar shape when Kf is approximately 250. Anything beyond 400-500 for a Kf value already shows signs of overmodulation/sampling.

Code used:

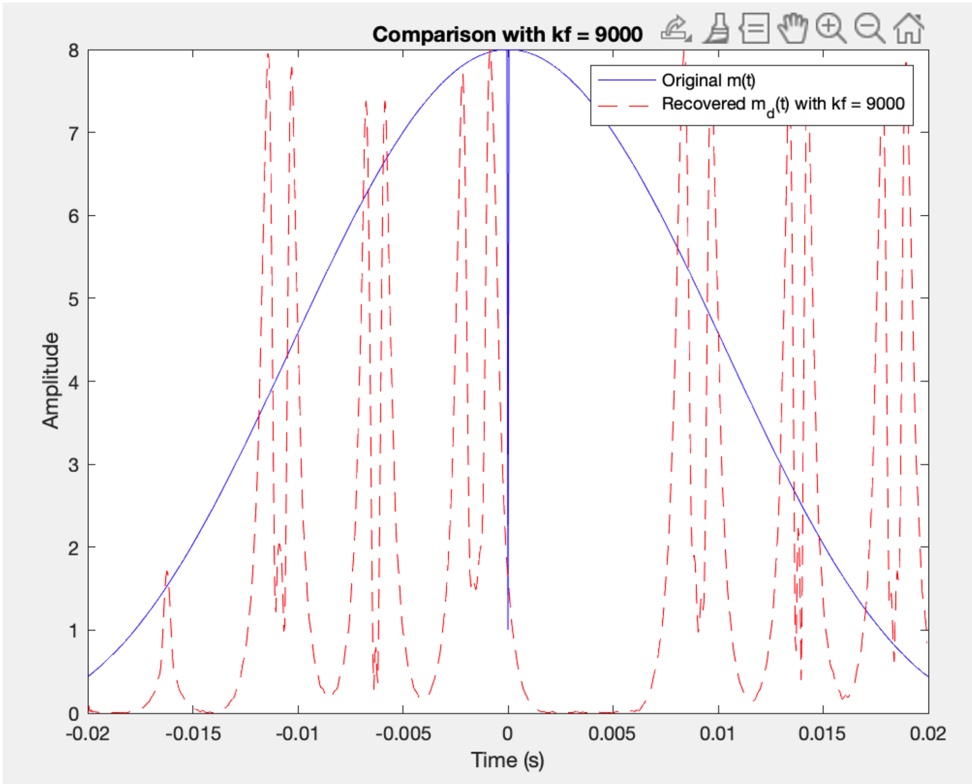
Editor – /Users/david/Desktop/ECE 4700/ECE4700_ComputerProject_F

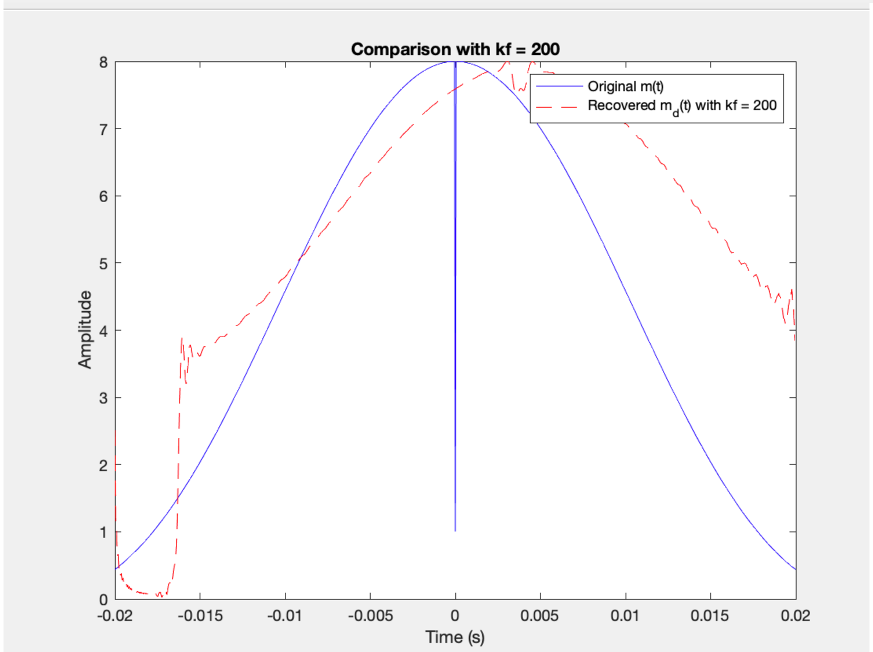
ECE4700_ComputerProject_Final.m

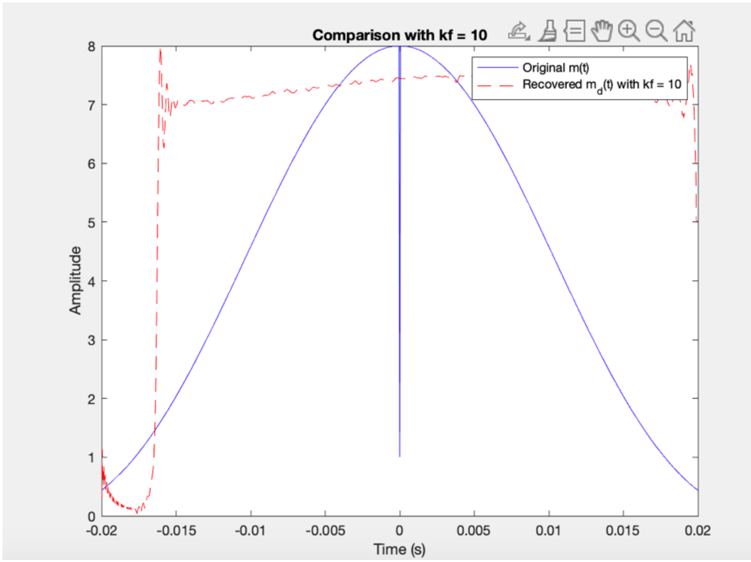
```matlab
% Part 4: Optimizing md(t) by modifiying kf:

%Setting kf values and initializing MSE storage
kf_values = [10, 200, 9000];  %Example values including one for over-modulation
mse_values = zeros(size(kf_values));
optimal_mse = inf;
optimal_kf = 0;

%Defining filters outside the loop:

nyquist_freq = fs / 2;
lower_bound_normalized = (10/fs);
upper_bound_normalized = (3600/nyquist_freq);
bpf_before_diff = fir1(100, [lower_bound_normalized upper_bound_normalized]);

%Loop over each kf value
for i = 1:length(kf_values)
    kf = kf_values(i);  %Current value of kf

    %FM Modulation with new kf value
    integral_mt = cumsum(mt) * ts;  %Recalculating integral with new kf
    st = Ac * cos(2*pi*fc*t + 2*pi*kf*integral_mt);  %new FM signal

    %Generating noise of same power 50mW and to match the FM signal samples
    n_t = sqrt(noise_power/fs) * randn(1, length(t));

    %Creating the noisy received signal r(t)
    r_t = st + n_t;

    %Applying the bandpass filter before differentiation
    r_t_filtered = filter(bpf_before_diff, 1, r_t);

    %Differentiating the signal
    diff_r_t = [diff(r_t_filtered), 0];

    %Applying a low-pass filter after differentiation
    B_m = 4000;
    bpf_after_diff = fir1(80, (B_m*2) / nyquist_freq);  %Low-pass filter parameters
    filtered_diff_r_t = filter(bpf_after_diff, 1, diff_r_t);

    %Envelope detection to retrieve m_d(t)
    md_t = abs(hilbert(filtered_diff_r_t));

    %Scaling the envelope-detected signal to match the amplitude of the original message
    scale_factor = max(mt) / max(md_t);  %scaling factor
    md_t_scaled = md_t * scale_factor;

    %Calculating the Mean Squared Error for optimization
    mse_values(i) = immse(mt(1:end-1), md_t_scaled(1:end-1));

    %Checking if this kf is better
    if mse_values(i) < optimal_mse
        optimal_mse = mse_values(i);
        optimal_kf = kf;
    end

    %Plotting the recovered message signal for current value of kf
    figure;
    plot(t, mt, 'b', t(1:end-1), md_t_scaled(1:end-1), 'r--');
    legend('Original m(t)', 'Recovered m_d(t) with kf = ' + string(kf));
    title(['Comparison with kf = ' + string(kf)]);
    xlabel('Time (s)');
    ylabel('Amplitude');
end

%Displaying the best kf value calculated in the above loop:
disp(['The best value of kf for FM demodulation is ' + string(optimal_kf)]);
```

Note: I provided examples of many different Kf values used, which helped to visualize the effect of under and over modulating the signal.

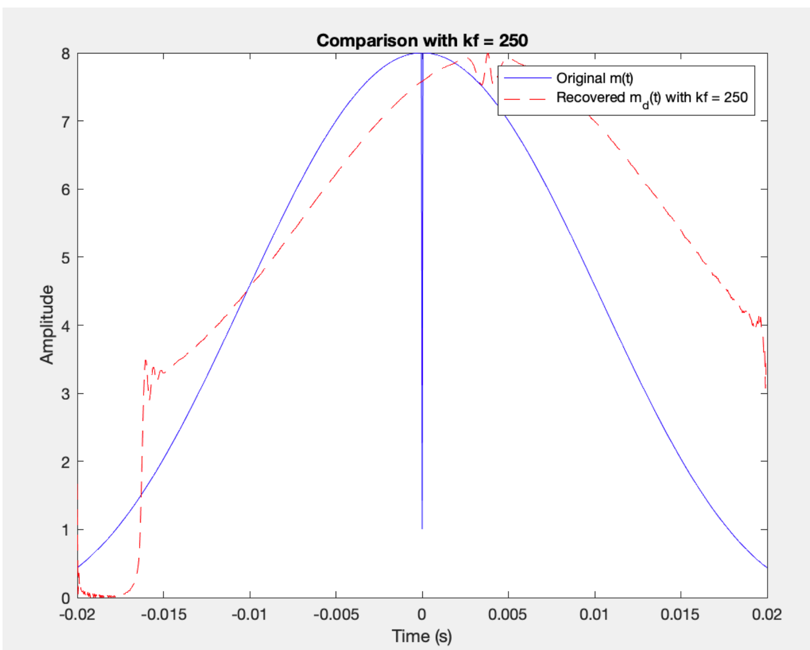
Further discussion of Kf:
The value of Kf in an FM system controls the modulation index, and how much the signal's frequency changes. If Kf is too low, the signal can be easily messed up by the noise, but if it's too high, it will use too much bandwidth and cause overmodulation, picking up high frequency disturbances. Finding the right Kf is about making the signal strong against noise without taking up unnecessary bandwidth. The best Kf gives you a clearer retrieved signal that has an ideal SNR.


Comparison with kf = 9000
Original m(t)
Recovered m_d(t) with kf = 9000
Amplitude
Time (s)
0
1
2
3
4
5
6
7
8
-0.02
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
0.02

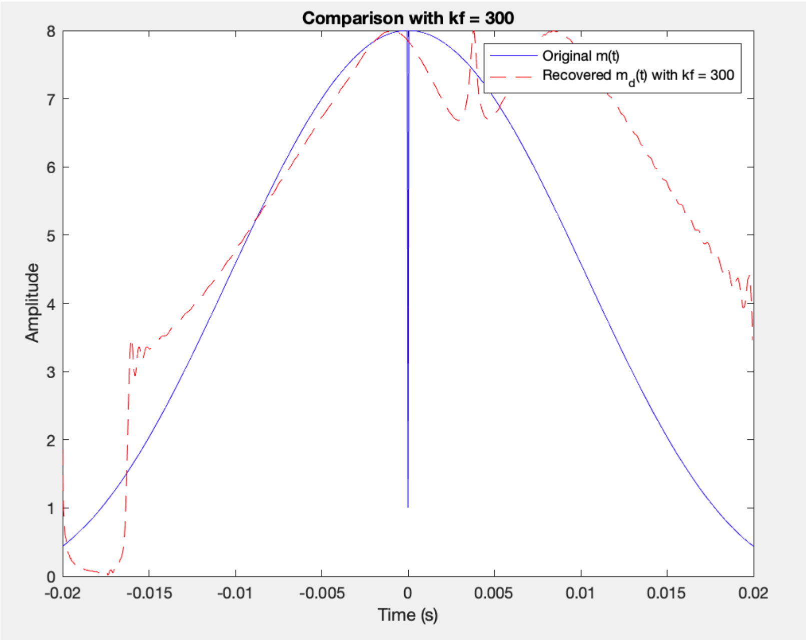

Comparison with kf = 200
Original m(t)
Recovered m_d(t) with kf = 200
Amplitude
Time (s)
0
1
2
3
4
5
6
7
8
-0.02
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
0.02



Comparison with kf = 10
Original m(t)
Recovered m_d(t) with kf = 10
Amplitude
Time (s)



Comparison with kf = 250
Original m(t)
Recovered m_d(t) with kf = 250
Amplitude
Time (s)



From the graphs above, we can once again see that a Kf of approximately 250-300 seems to be the best fit for our case.