FM Radio Lab

Lauren Anfenson and Anusha Datar

September 26, 2019

1 Exercise 1 : Envelope Detector Approach

For this first exercise, we are supposing that we want to decode a signal that is broadcasting at $f_t = 90.9MHz$, with a bandwidth of 80kHz. If we set $f_c = 90.8MHz$ and assume that our phase shift is negligible, the signal $y_I(t)$ will look something like the picture below.

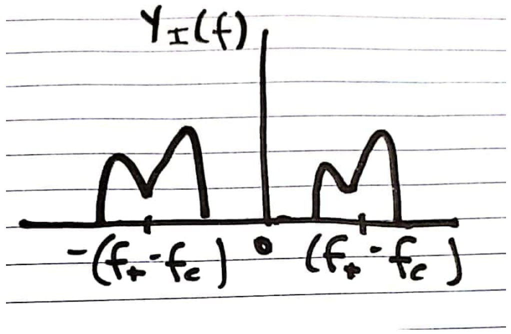

Figure 1: Example plot of a signal decoded with $f_c = 90.8MHz$.

We can also show mathematically that the $y_I(t)$ measured by our receiver is about equivalent to a frequency-modulated signal with a lower carrier frequency than f_t. Our received signal $y(t)$ is multiplied with some phase-shifted cosine by the receiver. We can represent this new signal as

$$\tilde{y}(t) = A_c cos(2_t t + \phi(t))cos(2_c t + \phi(t)) \tag{1}$$

We can then use the trig identity for the product of two cosines to rewrite this equation as

$$\frac{A_c}{2}(cos(2(f_t + f_c) + 2\phi(t)) + cos(2(f_t - f_c))) \tag{2}$$

The next step is to low-pass filter the signal. Any high-frequency components of the signal will be attenuated by the filter – so, we can ignore the $f_t + f_c$ component of the signal as it will be filtered out. We also assume that the phase shift, $\phi(t)$, is negligible. This leaves us with the expression

$$y_I(t) = \frac{A_c}{2} cos(2(f_t - f_c)) \tag{3}$$

This shows that the signal $y_I(t)$ is roughly equivalent to a scaled version of the input signal, $y(t)$, with a carrier frequency of $f_t - f_c$.

We used the provided RTL-SDR MATLAB sample script to collect data while tuned to $90.8MHz$. The plot below shows the FFT of y_I from that collected data.

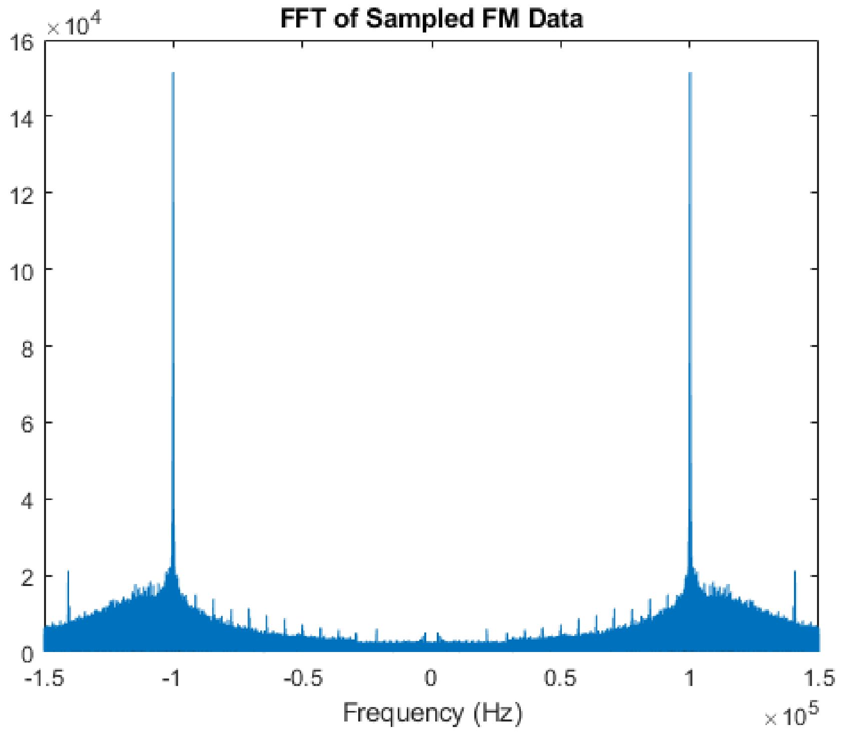

Figure 2: Fourier transform of the FM data collected with our RTL-SDR.

We can see that this about matches what we predicted in Figure 2 – there are two distinct groupings of frequencies, one positive and one negative, centered on $f_t - f_c$, or 100kHz.

Looking at the time-domain plot of the signal y_I, we can zoom in and see the effects of frequency modulation on the signal. The plot below shows 1000 of the 3 000 000 samples that were taken to clearly show the varying frequency of the signal, coupled with its constant amplitude, that are characteristic of a frequency modulated signal.

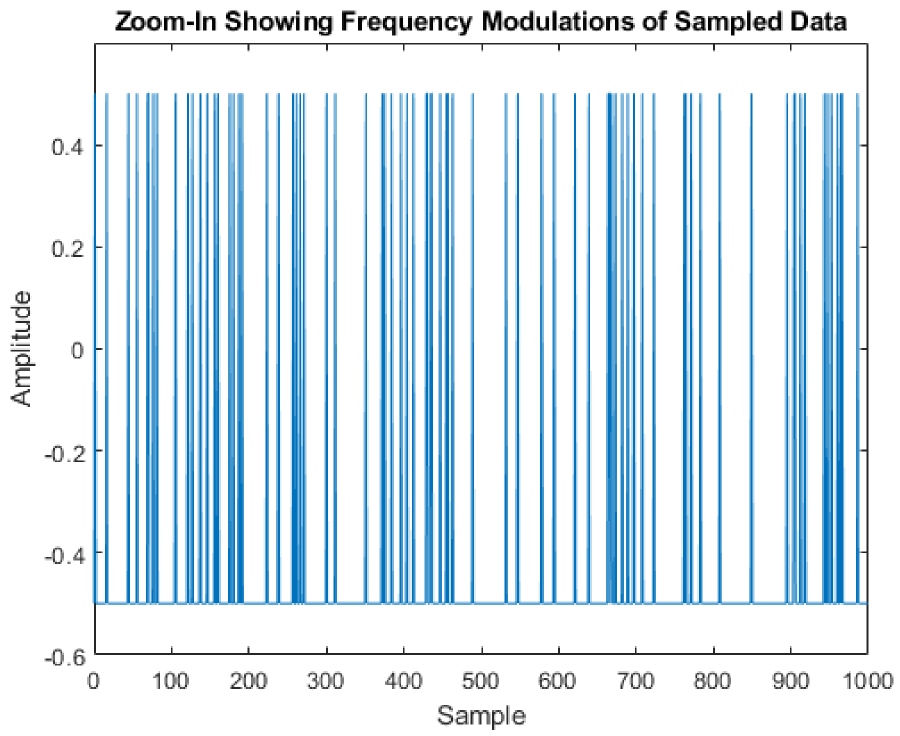

Figure 3: Zoom-in to show the changing frequencies within the signal.

In order to actually listen to the sound that this radio signal is trying to convey, we needed to create a receiver that could demodulate the FM signal into something as close to the original, un-modulated signal as possible. Our first approach to this took the derivative of the signal, cut off its negative values, and then used an envelope detector to recover the original signal. A block diagram of a receiver following this approach is shown below.

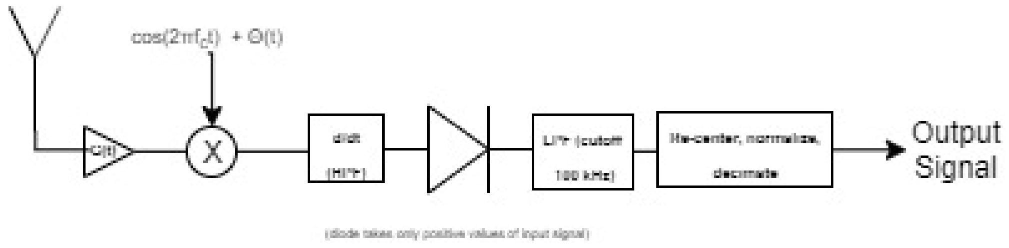
cos(2πf_ct) + Θ(t)
C(t)
X
d/dt (HPF)
LPF (cutoff 100 kHz)
Re-center, normalize, decimate
Output Signal
(diode takes only positive values of input signal)

Figure 4: Block diagram of the envelope detector approach to detecting the signal using the in-phase data.

The plot below shows the received FM signal in the time domain after the first demodulation step, taking its derivative and zeroing out its negative components. The signal has been normalized so that its maximum value is 1.

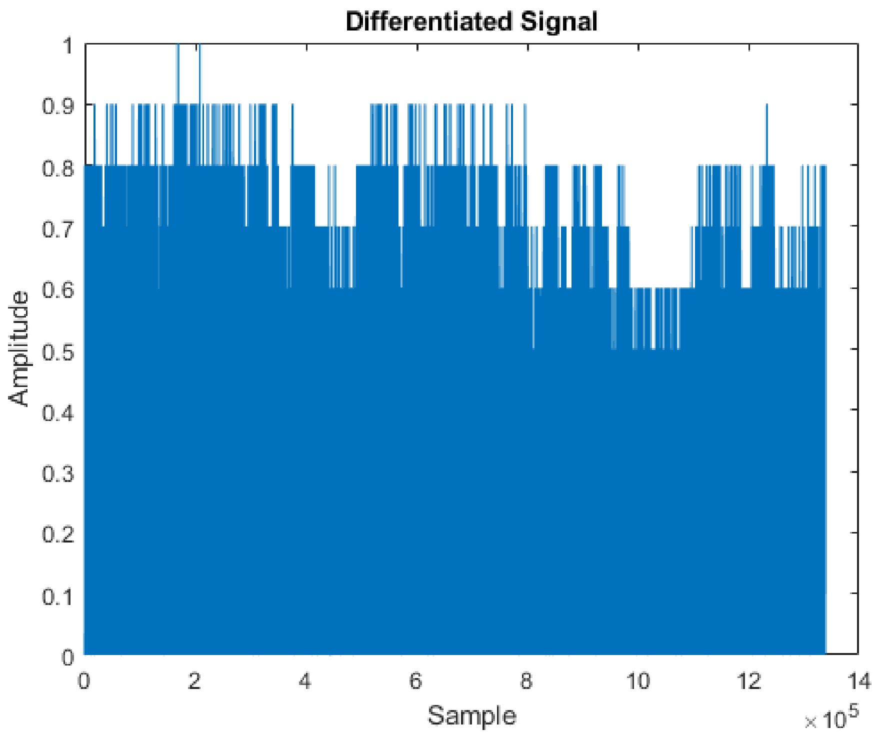

Figure 5: Plot showing what the signal looks like after the first steps of demodulation – taking its derivative and zeroing out any negative components.

In order to envelope detect the new signal we applied a low-pass filter with an RC value that was much greater than $\frac{1}{100000}$, the inverse of our carrier frequency, and much less than $\frac{1}{90.8x10^9}$, the inverse of our tuning frequency. This allows us to follow the curve of the signal closely without picking up too much of the high frequency aspects of the signal that we are trying to filter out. The value we chose is about $\frac{1}{200000}$. This may not seem "much greater than" our carrier frequency, but it is the value that gave us the cleanest results, with some trial and error, in our final demodulated signal.

The plot below shows the derivative of our signal overlayed with the low-pass filtered signal. With a zoom-in to 1000 samples, we can see the effects of the envelope detector as it smooths over the high-frequency modulations and leaves us with the outline – or envelope – that more closely represents the original, un-modulated signal.

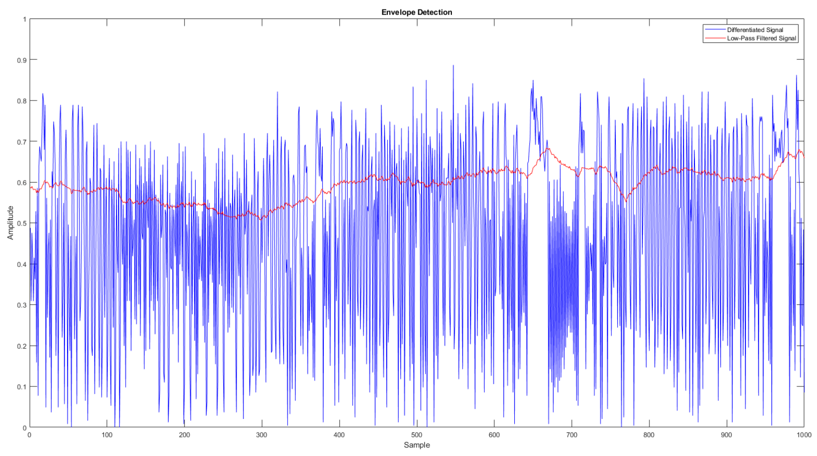

Figure 6: Plot showing the effects of using a low-pass filter for envelope detection on the signal.

Finally, we subtracted the mean out of the signal and normalized it to 0.1 for volume control. The sampling rate was decimated by 4 so the sound could be played by our computers. Our final demodulated signal using this approach can be found on our Soundcloud, linked at the bottom of our document.

2 Exercise 2 : FM Decoding Using I and Q Channels

The block diagram below shows the procedure through which to decode an audio signal using both the in-phase and quadrature data.

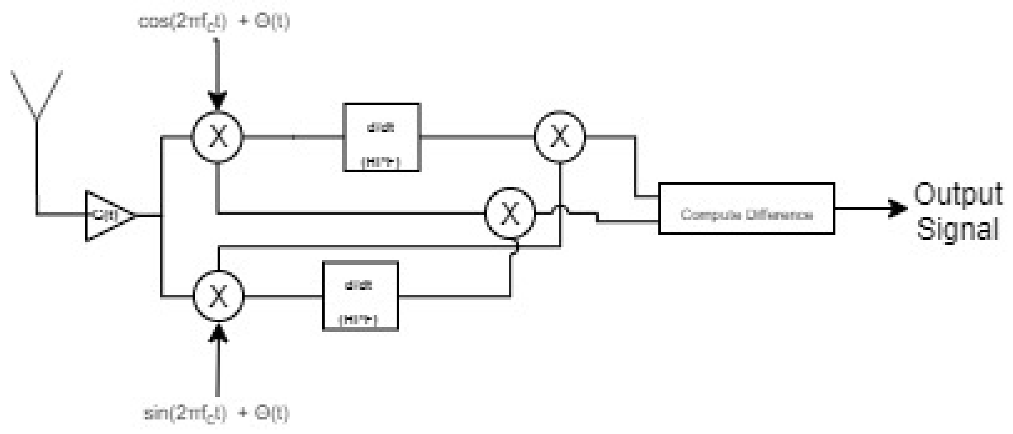

Figure 7: Block diagram of the in-phase and quadrature signal-based detector.

Mathematically, this process will produce a signal proportional to the original transmitted message, assuming that θ(t) is small and slowly varying. To show this, we can start with the equation:
$\hat{m} = (\frac{d}{dt}y_Q(t))(y_I(t)) - (\frac{d}{dt}y_I(t))(y_Q(t))$
We know that the received signal signal is equal to
$Gcos(2\pi f_c t + \phi(t))$ where $\phi(t) = 2\pi k_f \int_{-\infty}^{t} m(\tau)d\tau$
For the in-phase data, we first multiply by $cos(2\pi f_c t + \theta(t))$:
$Gcos(2\pi f_c t + \phi(t))cos(2\pi f_c t + \theta(t))$
Using the difference of cosines identity :
$y_I = \frac{1}{2}Gcos(4\pi f_c t + \phi(t) + \theta(t))cos(\phi(t) - \theta(t))$

Then, using a low pass filter to filter out the higher frequencies:
$y_I = \frac{1}{2}Gcos(\phi(t) - \theta(t))$
Then, expanding the difference inside the cosine :
$y_I = \frac{1}{2}G(cos(\phi(t))cos(\theta(t)) + sin(\phi(t))sin(\theta(t)))$
Because θ is small, we can assume that $sin(\theta) = 0$ and $cos(\theta) = 1$. therefore, our equation for y_I simply becomes:
$y_I = \frac{1}{2}cos(\phi(t))$

We can use a similar procedure to find the function for the quadrature data. $y_Q(t)$ equals
$Gcos(2\pi f_c t + \phi(t))sin(2\pi f_c t + \theta(t))$
Using the difference of sines identity :
$y_Q = \frac{1}{2}Gsin(4\pi f_c t + \phi(t) + \theta(t))sin(\phi(t) - \theta(t))$
Then, using a low pass filter to filter out the higher frequencies:
$y_Q = \frac{1}{2}Gsin(\phi(t) - \theta(t))$
Then, expanding the difference inside the sine :
$y_Q = \frac{1}{2}G(sin(\phi(t))cos(\theta(t)) + cos(\phi(t))sin(\theta(t)))$
Because θ is small, we can assume that $sin(\theta) = 0$ and $cos(\theta) = 1$. therefore, our equation for y_Q simply becomes:
$y_Q = \frac{1}{2}sin(\phi(t))$

We now know that : $y_I = \frac{1}{2}cos(\phi(t))$
$y_Q = \frac{1}{2}sin(\phi(t))$
$\frac{d}{dt}y_I = \frac{1}{2}Gsin(2\pi k_f \int_{-\infty}^{t} m(\tau)d\tau)(2\pi k_f m(t)) = \frac{1}{2}Gsin(\phi(t))2\pi k_f m(t))$
$\frac{d}{dt}y_Q = \frac{1}{2}Gcos(2\pi k_f \int_{-\infty}^{t} m(\tau)d\tau)(2\pi k_f m(t)) = \frac{1}{2}Gcos(\phi(t))2\pi k_f m(t))$
Then, using the original equation:
$\hat{m} = (\frac{d}{dt}y_Q(t))(y_I(t)) - (\frac{d}{dt}y_I(t))(Y_Q(t))$
$\hat{m} = \frac{1}{2}Gcos(\phi(t))2\pi k_f m(t))(\frac{1}{2}cos(\phi(t))) - (\frac{1}{2}Gsin(\phi(t))2\pi k_f m(t))(\frac{1}{2}sin(\phi(t)))$
$\hat{m} = \frac{1}{4}G^2 2\pi k_f m(t)(cos^2(\phi(t)) + sin^2(\phi(t)))$
$\hat{m} = \frac{1}{4}G^2 2\pi k_f m(t)$
This is roughly proportional to the original message signal ($m(t)$) - the output will contain the original message scaled by $\frac{G^2}{4}2\pi k_f$.

We can then implement this in software with the following code:

```
%% Decode the IQ data based on the equation in part a.
cutoff = 100000;
message = (diff(y_Q).*y_I(1:size(y_I)-1)) - (diff(y_I).*y_Q(1:size(y_I)-1));
mean_sub_message = message - 2.5.*mean(message); % Center the data from the message.
% Follow the same normalization procedure from the first exercise.
norm_sub_message = 0.1.*(mean_sub_message - min(mean_sub_message)) ./ ...
(max(mean_sub_message) - min(mean_sub_message));
decimate_message = decimate(norm_sub_message, 4);
plot_FT(decimate_message, fs./4);
%% Play the sound of the decoded message.
sound(decimate_message, fs./4)
```

3 Source Code and Files

All of the source code used in this experiment is posted on:
`https://github.com/anushadatar/PlantBasedMilk-FMRadio`.
The decoded audio file for experiment one is posted on:
`https://soundcloud.com/user-674765397/exercise1-audio-2`

The decoded audio file for experiment two is posted on :
`https://soundcloud.com/user-674765397/exercise2-audio`.