Design and Application of a Bandpass Filter

Name: Christ-Brian Amedjonekou

Date: 04/29/2019

TCET 3102-E316 (Analog and Digital Com) Lab 4

Spring 2019, Section: E316, Code: 37251

Instructor: Song Tang

Table of Contents

TCET 3102-E316 (Analog and Digital Communications) Lab 4

April 29, 2019

Name: Christ-Brian Amedjonekou
Date: 4/28/2019
TCET 3102-E316 (Analog and Digital Communications) Lab 4
Spring 2019, Section: E316, Code: 37251
Instructor: Song Tang

0.0.1 Objective

- Design a bandpass (Butterworth) filter and utilize that filter to select/pass certain frequencies and reject other frequencies.

0.0.2 Equipment

- Computer Software

0.0.3 Theory

- Filters are used to remove unwanted parts of the input signal. In most cases, this is noise present outside the frequency band of the desired signal.
- In many applications we need to filter out a particular band of frequencies. This is why we use a passband filter: to isolate a particular band of frequencies. With passband filters we have a upper and lower limit for the frequencies we allow to pass. Anything outside of these limits will be attenuated (filtered out/removed).
- Pass-Band filters are usually created by combining High and Low Pass filters together.
- Just like in other sciences wehave ideal and practical Pass-Band Filters.

Ideal Case

- An ideal Pass-Band Filter will have no ripples (completely flat)
- It would also attentuate all frequencies outside of its upper and lower limit (pass-band/band-pass)
- It would transition from passband to stopband instantaneously
- In the real world there are no ideal cases

Practical (Real World) Case

- A practical Pass-Band Filter will have some ripples (not completely flat)
- It would fail attentuate all frequencies outside of its upper and lower limit (pass-band/band-pass)
  - This situation is call ***'Filter Roll-Off'***
- It would not transition from passband to stopband instantaneously
- Observed/used In the real world all the time.

How to Handle these imperfections

- Usually in design, engineers try to make ***'Filter Roll-Off'*** as narrow as possible
- This is done to allow close to ideal characteristics for filter functionality

0.0.4 Imported Packages

In [1]:
```
# These are the packages I'll need to solve this problem
import math as ma
import numpy as np
from matplotlib import pyplot as plt
from scipy.fftpack import fft, fftfreq
from scipy.signal import butter, buttord, freqz, lfilter
```

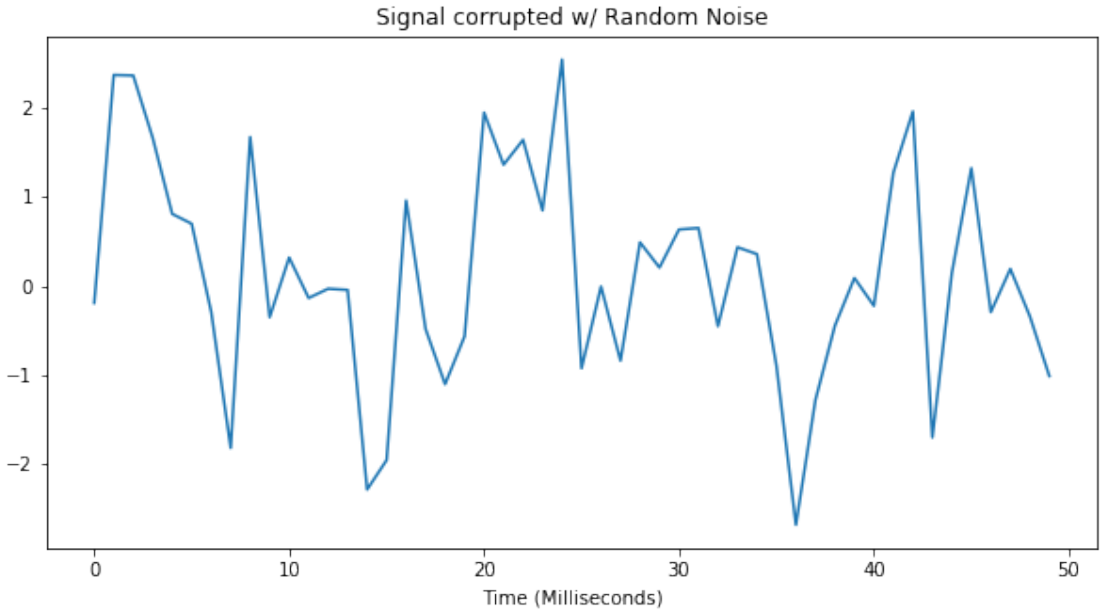
0.0.5 RUN 1: Generate a Noisy Signal

- **Step 1: Creating the Noisy Signal**

In [2]:
```
# The code below this comment is 'Step 1' of Run 1
# Sampling Frequency
Fs = 1000

# Sampling Time (T), Length of signal (L)
T, L = 1/Fs, 1000

# Time Vector (t)
t = np.linspace(0, L-1, 1000)*T

# We set 'x' to be the sum of 50 Hz and 100 Hz sinusiods
c, f = [0.6, 0.9, 1.2], [50, 100]
x = c[0]*np.sin(2*np.pi*f[0]*t) + c[1]*np.sin(2*np.pi*f[1]*t)

# Noisy signal Creation
noise = c[2]*np.random.randn(t.size)

# Signal w/ Noise
swN = x + noise

# Plots of Signal w/ Noise
```

```
        plt.figure(figsize= (10,5))
        plt.plot(Fs*t[:50], swN[:50])
        plt.title('Signal corrupted w/ Random Noise')
        plt.xlabel('Time (Milliseconds)')
```

```
Out[2]: Text(0.5, 0, 'Time (Milliseconds)')
```

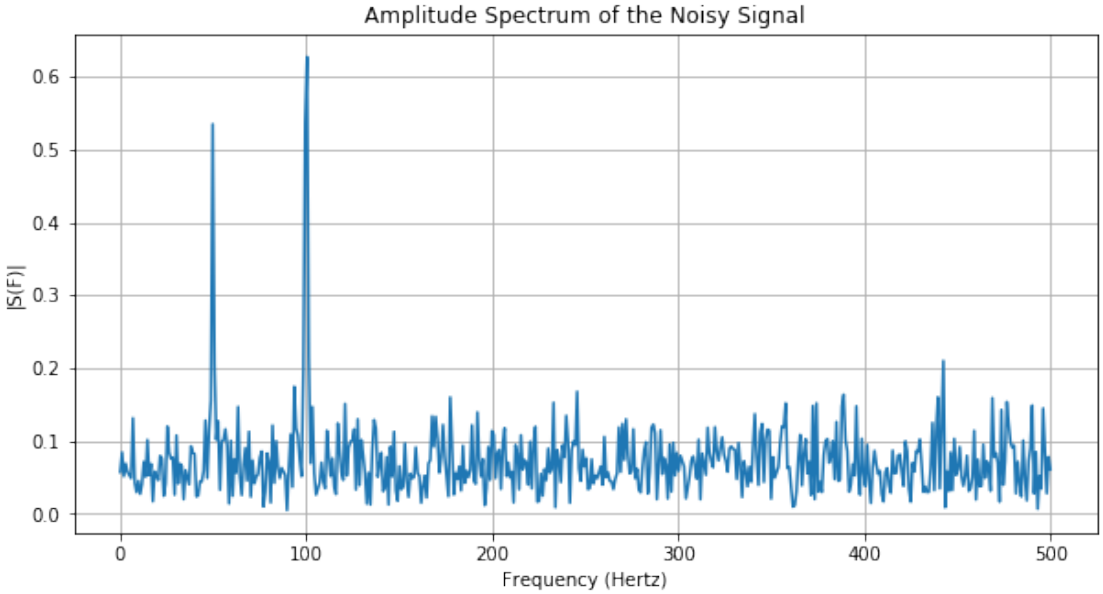


- **Step 2: Observing the Noisy Signal**

```
# Also the Next Power of 2 from Length 'y' function definition
def nextpow2(i):
    """ This is internal function used by fft(), because the FFT routine
   requires that the data size be a power of 2."""
    n = 1
    while n < i:
        n *= 2
    return n
```

```
In [3]: # The code below this comment is 'Step 2' of Run 1
        # Next Power of 2 from Length 'y' function definition
        def nextpow2(i):
            """ This is internal function used by fft(), because the FFT routine
           requires that the data size be a power of 2."""
            return 1 if i == 0 else 2**(i - 1).bit_length()

        # Next Power of 2 from Length '1000'
        NFFT = 2^nextpow2(L)
```

```
# Fast Fourier Transform of the Noisy Signal
s = fft(swN, NFFT)/L
f_ = Fs/2 * np.linspace(0, 1, int(NFFT/2+1))

# Plots of Signal w/ Noise
plt.figure(figsize= (10,5))
plt.plot(f_, 2*abs(s[:int(NFFT/2+1)]))
plt.title('Amplitude Spectrum of the Noisy Signal')
plt.xlabel('Frequency (Hertz)')
plt.ylabel('|S(F)|')
plt.grid(which= 'both')
```

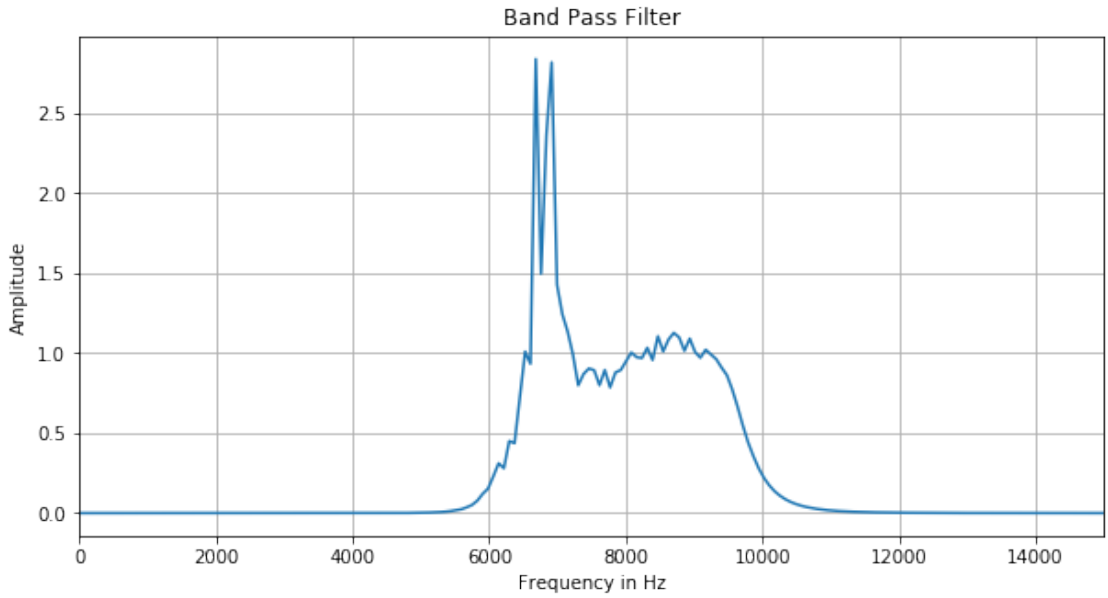


0.0.6 RUN 2: Design a Pass-Band (Butterworth) Filter

- **Step 1: Designing the Bandpass Filter**

In [12]:
```
# The code below this comment is 'Step 1' of Run 2
# Nyquist Frequency
Fn = Fs/2

# Bassband (Wp) and Stopband (Ws) frequencies normalized to Nyquist Frequencies
# Passband (Rp) and Stopband (Rs) ripples
Wp, Ws, Rp, Rs = np.array([40, 60])/Fn, np.array([30, 70])/Fn, 2, 35

# Finds the filter order for a given stopband and cuttoff frequencies
N, Wn = buttord(Wp, Ws, Rp, Rs)
```

```
b, a = butter(N, Wn, btype= 'bandpass')
omega_, H = freqz(b, a, 1024, fs= Fs)
xval = (omega_ * Fs)/(2 * np.pi)
yval = abs(H)

# Plot the Band Pass Filter
plt.figure(figsize= (10, 5))
plt.plot(xval, yval)
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude')
plt.title('Band Pass Filter')
plt.xlim(0, 15000)
plt.grid()
```




```
In [5]: (xval, yval.size)
```

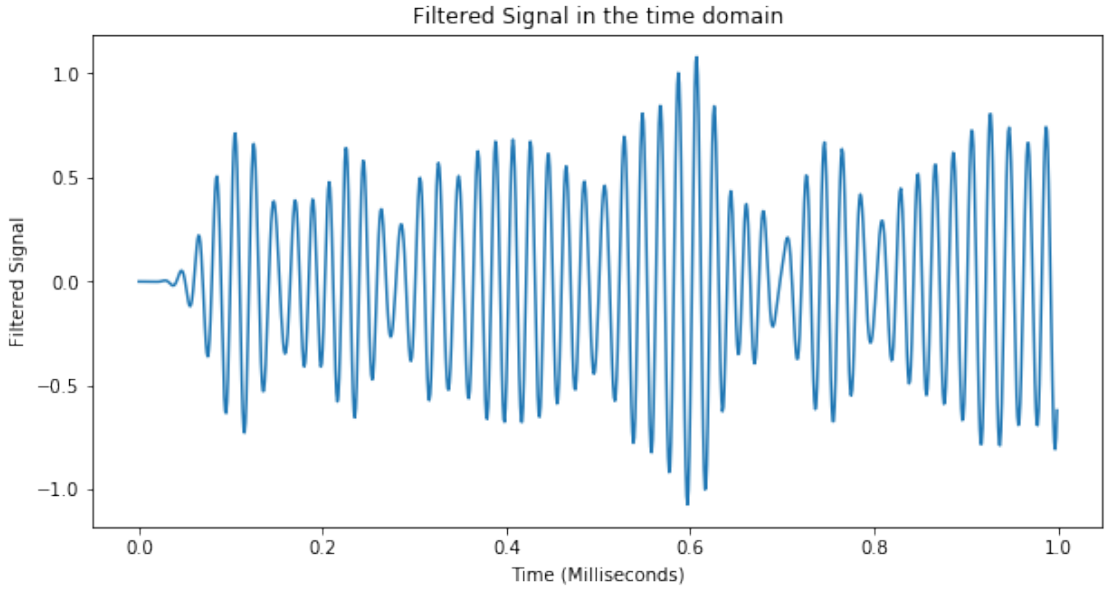
```
Out[5]: (array([0.00000000e+00, 7.77123746e+01, 1.55424749e+02, ...,
                7.93443344e+04, 7.94220468e+04, 7.94997592e+04]), 1024)
```

- **Step 2: Filtering the 50 Hz Signal from the Noisy Signal in RUN 1**

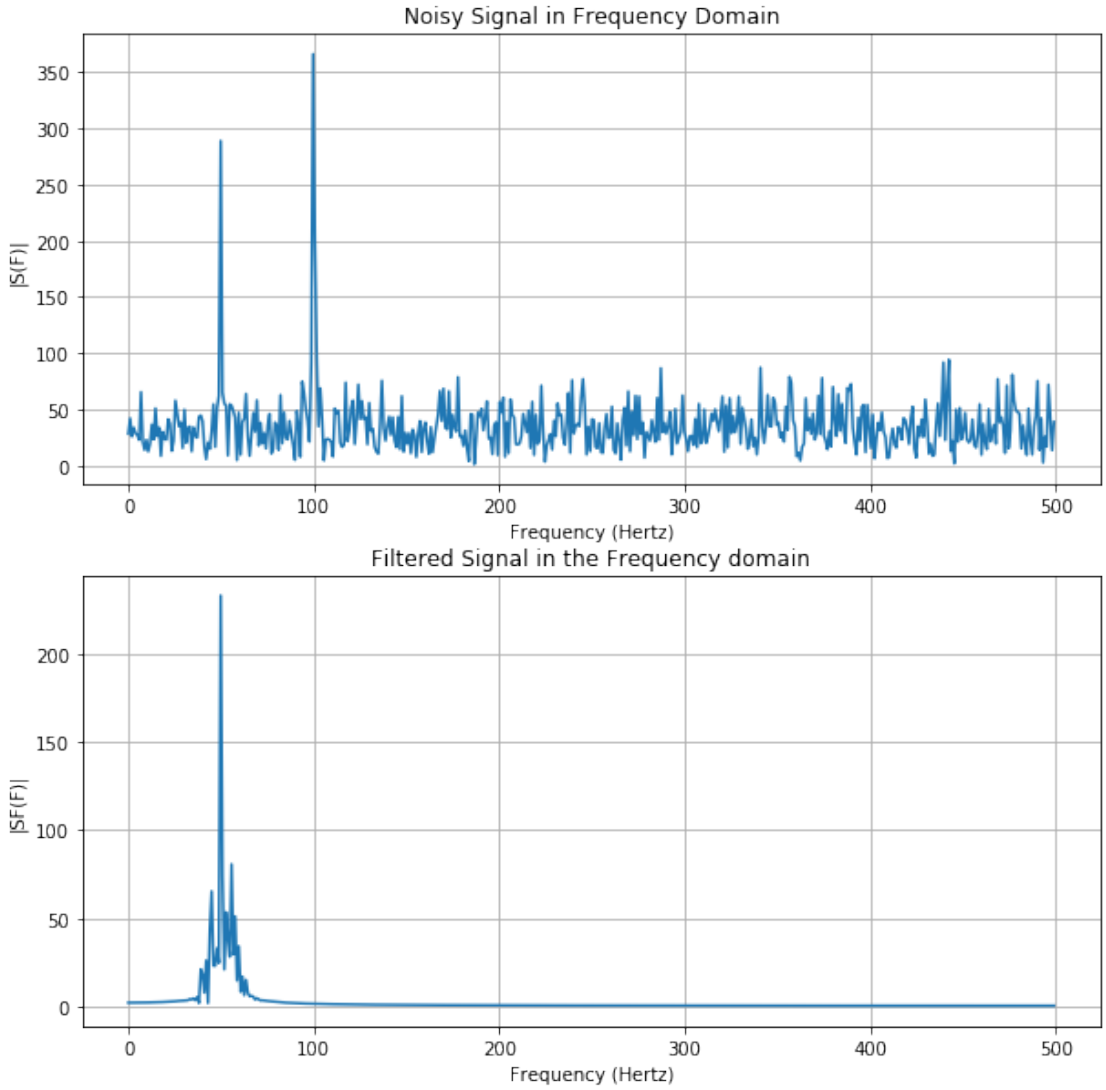
```
In [6]: # The code below this comment is 'Step 2' of Run 2
        # Filtered Signal in the time domain
        sf = lfilter(b,a,swN)

        # Plots of the filtered signal
        plt.figure(figsize= (10,5))
        plt.plot(t, sf[:1000])
```

```
plt.title('Filtered Signal in the time domain')
plt.xlabel('Time (Milliseconds)')
plt.ylabel('Filtered Signal')
```

Out[6]: Text(0, 0.5, 'Filtered Signal')



In [7]:
```
# The code below this comment is 'Step 2' of Run 2
# Noisy Signal & Filtered Signal in the Frequency domain
S = fft(swN, 1024)
SF = fft(sf, 1024)
_f = np.linspace(0, 511, 512)/512*Fn

# Plots of Noisy Signal in Frequency Domain
plt.figure(figsize= (10,10))
plt.subplot(2, 1, 1)
plt.plot(_f, abs(S[:512]))
plt.title('Noisy Signal in Frequency Domain')
plt.xlabel('Frequency (Hertz)')
plt.ylabel('|S(F)|')
plt.grid(which= 'both')

# Plots of Filtered Signal in the Frequency domain
plt.subplot(2, 1, 2)
plt.plot(_f, abs(SF[:512]))
plt.title('Filtered Signal in the Frequency domain')
plt.xlabel('Frequency (Hertz)')
plt.ylabel('|SF(F)|')
plt.grid(which= 'both')
```

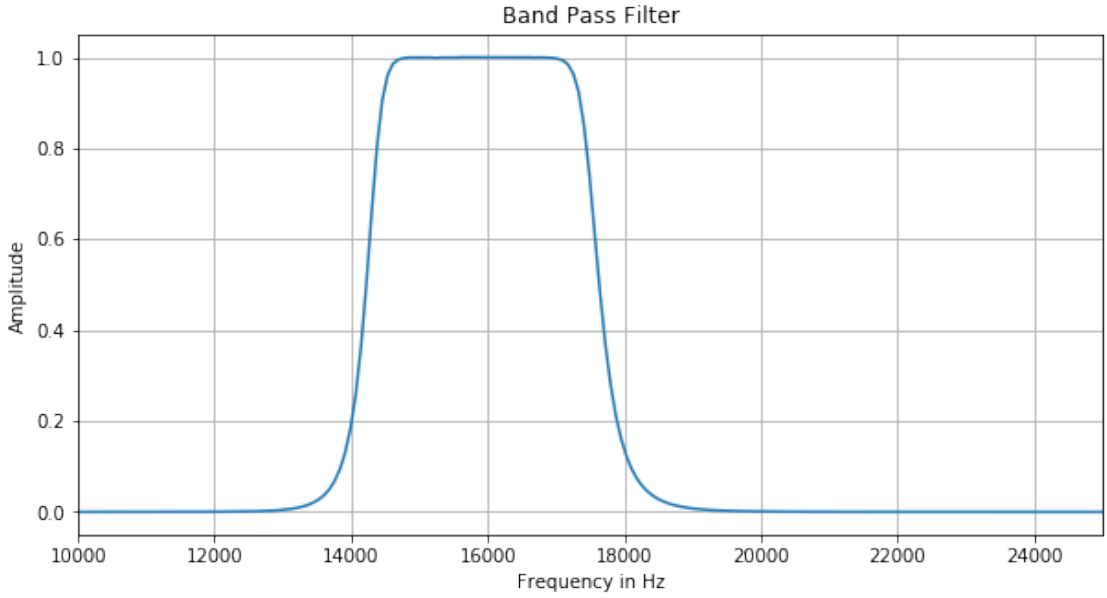


0.0.7 Questions

1. Change passband frequencies to (90 Hz - 110 Hz) and maximum passband ripple to 3 dB with 40 dB attenuation.

```
In [13]: # The code below this comment is 'Step 1' of Run 2
         # Nyquist Frequency
         Fn = Fs/2

         # Bassband (Wp) and Stopband (Ws) frequencies normalized to Nyquist Frequencies
         # Passband (Rp) and Stopband (Rs) ripples
         Wp, Ws, Rp, Rs = np.array([90, 110])/Fn, np.array([80, 120])/Fn, 3, 40

         # Finds the filter order for a given stopband and cuttoff frequencies
```

```
N, Wn = buttord(Wp, Ws, Rp, Rs)
b, a = butter(N, Wn, btype= 'bandpass')
omega_, H = freqz(b, a, 1024, fs= Fs)
xval = (omega_ * Fs)/(2 * np.pi)
yval = abs(H)

# Plot the Band Pass Filter
plt.figure(figsize= (10, 5))
plt.plot(xval, yval)
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude')
plt.title('Band Pass Filter')
plt.xlim(10000, 25000)
plt.grid()
```

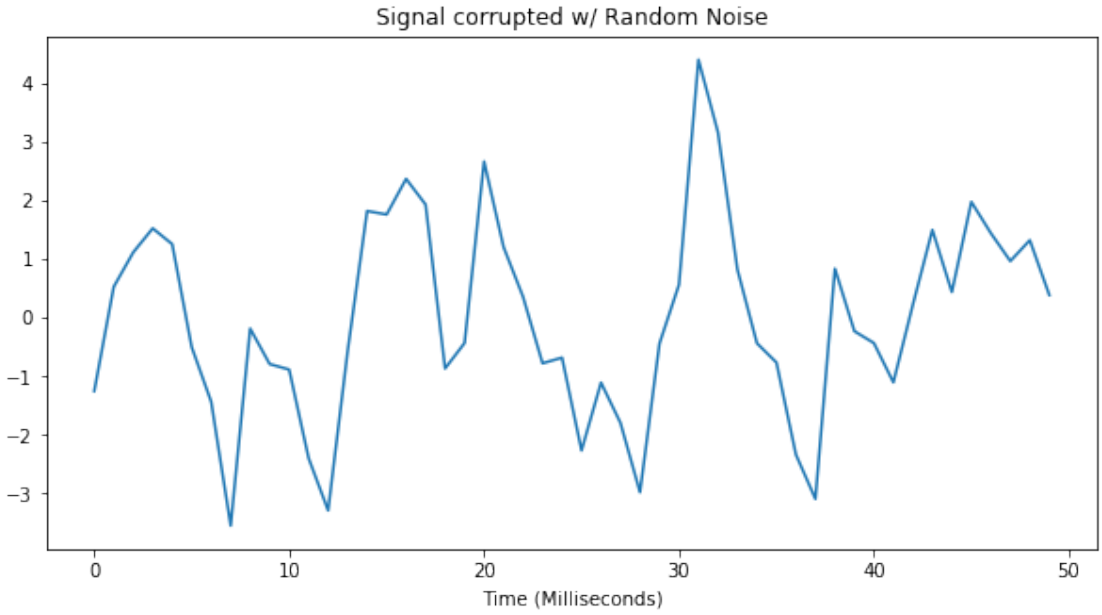


2. Change signal to 70 Hz snusoid of amplitude 1 and 100 Hz sinusoid of amplitude 0.9

```
In [9]: # The code below this comment is 'Step 1' of Run 1
        # Sampling Frequency
        Fs = 1000

        # Sampling Time (T), Length of signal (L)
        T, L = 1/Fs, 1000

        # Time Vector (t)
        t = np.linspace(0, L-1, 1000)*T

        # We set 'x' to be the sum of 50 Hz and 100 Hz sinusiods
```

```
c, f = [1, 0.9, 1.2], [70, 100]
x = c[0]*np.sin(2*np.pi*f[0]*t) + c[1]*np.sin(2*np.pi*f[1]*t)

# Noisy signal Creation
noise = c[2]*np.random.randn(t.size)

# Signal w/ Noise
swN = x + noise

# Plots of Signal w/ Noise
plt.figure(figsize= (10,5))
plt.plot(Fs*t[:50], swN[:50])
plt.title('Signal corrupted w/ Random Noise')
plt.xlabel('Time (Milliseconds)')
```

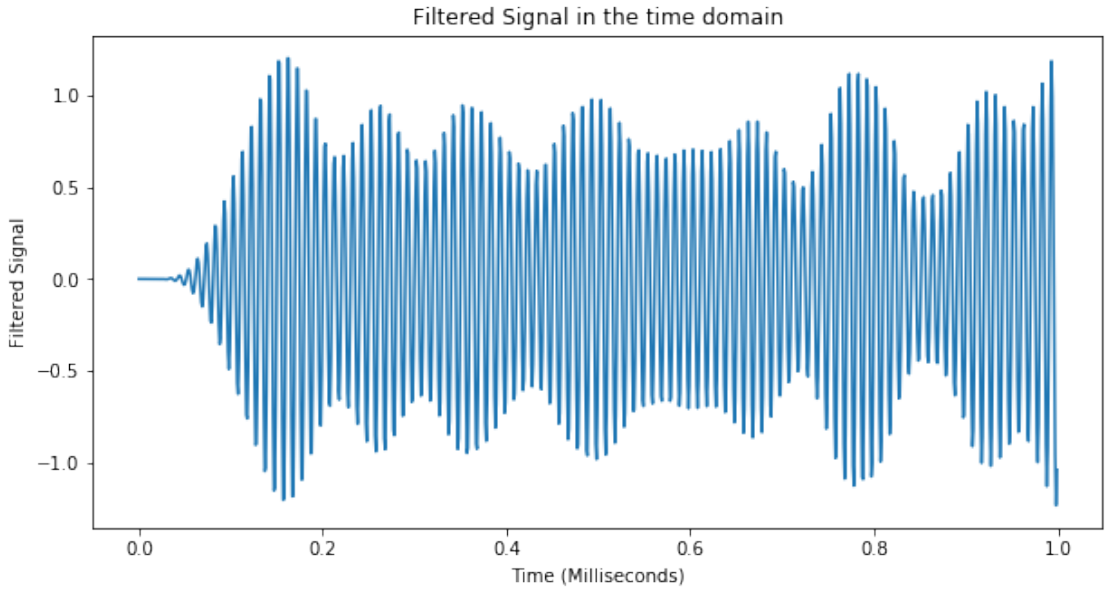
Out[9]: Text(0.5, 0, 'Time (Milliseconds)')



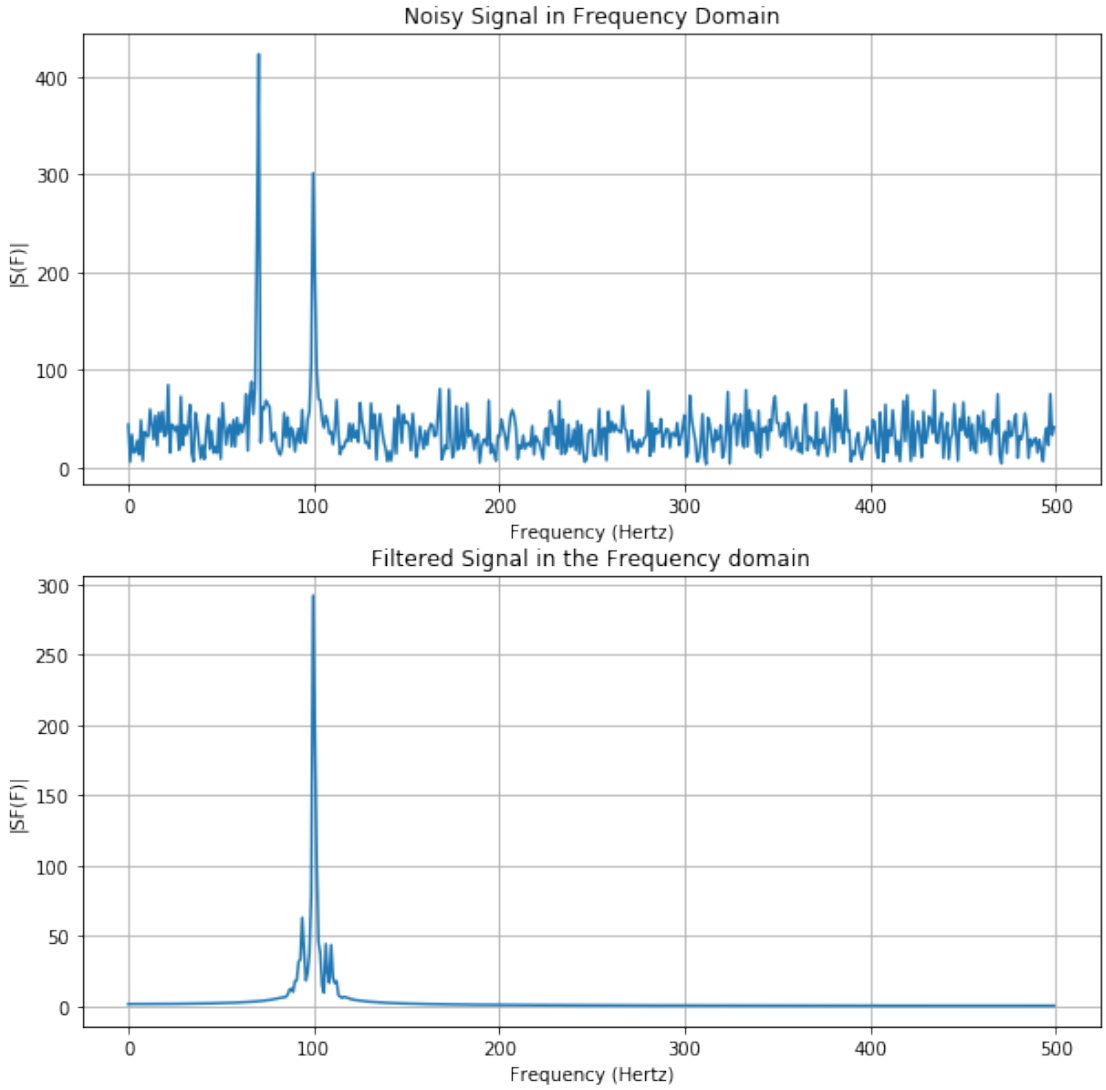
3. Filter the 100 Hz sinosoid signal from the noisy gnal and show frequency domain figures.

In [10]:
```
# The code below this comment is 'Step 2' of Run 2
# Filtered Signal in the time domain
sf = lfilter(b,a,swN)

# Plots of the filtered signal
plt.figure(figsize= (10,5))
plt.plot(t, sf[:1000])
plt.title('Filtered Signal in the time domain')
plt.xlabel('Time (Milliseconds)')
plt.ylabel('Filtered Signal')
```

Out[10]: Text(0, 0.5, 'Filtered Signal')



In [11]:
```
# The code below this comment is 'Step 2' of Run 2
# Noisy Signal & Filtered Signal in the Frequency domain
S = fft(swN, 1024)
SF = fft(sf, 1024)
_f = np.linspace(0, 511, 512)/512*Fn

# Plots of Noisy Signal in Frequency Domain
plt.figure(figsize= (10,10))
plt.subplot(2, 1, 1)
plt.plot(_f, abs(S[:512]))
plt.title('Noisy Signal in Frequency Domain')
plt.xlabel('Frequency (Hertz)')
plt.ylabel('|S(F)|')
plt.grid(which= 'both')

# Plots of Filtered Signal in the Frequency domain
plt.subplot(2, 1, 2)
plt.plot(_f, abs(SF[:512]))
plt.title('Filtered Signal in the Frequency domain')
plt.xlabel('Frequency (Hertz)')
plt.ylabel('|SF(F)|')
plt.grid(which= 'both')
```



0.0.8 Analysis

- **In the first run, we were able to see the signal corrupted with random noise from 0 to 50 ms time frame; This is shown in the time domain. For the frequency domain we see the frequencies 50 and 100 Hz of the original signal plus all the other smaller frequencies representing the noise. Following up w/ run 2, we were able create the Butterworth bandpass filter but this filter has some sort of ripple and is not completely flat. However, it is able to remove the random noise, shown later in Run 2. Running RUNs 1 & 2 w/ different values we were able to see the signal corrupted with random noise from 0 to 50 ms time frame; This is the same as before. We also see that the frequencies 90 and 100 Hz of the original signal plus all the other smaller frequencies representing the noise. This time our Butterworth bandpass filter is completely flat, and like before, only allows frequencies within the desired frequency range to pass, rejecting all others.**

0.0.9 Conclusion

- **I was successful in running the simluation of bandpass filters. I learned about functionality of bandpass filter; That it can accept range of accepted frequencies and reject all others by setting up the desired frequency frequency range.**