EE 325 Digital Signal Processing

Laboratory Assignment 2

E/19/445

Libraries

In [ ]:
```
import numpy as np
import math
import sympy as sp
from sympy import latex
from scipy import signal
from scipy.signal import butter, freqs, TransferFunction, group_delay, impulse
from scipy.signal import cheby1, cheby2, filtfilt, buttord, cheb1ord
from scipy.signal import cheb2ord, firwin, firwin2,freqz
from scipy.fftpack import fft, fftfreq
import matplotlib.pyplot as plt
```

Filter Specifications

In [ ]:
```
x, y, a, b, c = 1, 9, 4, 4, 5

d = a + b + c

if d >= 10:
    d1 = int(str(d)[0])
    d2 = int(str(d)[1])
    d = d1*d2

w_p = 100 + np.sqrt(1.1*a + 11*b + 101*c)
w_s = w_p*(1+np.sqrt(d/10))

delta_s = 0.1
delta_p = 0.9
delta_t = 0.1
```

Butterworth Filter Order

In [ ]:
```
analog_w_p = np.tan(w_p*180/(2*np.pi))
analog_w_s = np.tan(w_s*180/(2*np.pi))

butterworth_order = buttord(analog_w_p, analog_w_s, delta_s, delta_p, analog=True)[0]
print(f"Butterworth order: {butterworth_order}")
```

```
Butterworth order: 3
```

Chebyshev Type I Order

In [ ]:
```
cheby1_order = cheb1ord(analog_w_p, analog_w_s, delta_s, delta_p, analog=True)[0]
print(f"Chebyshev Type I order: {cheby1_order}")
```

```
Chebyshev Type I order: 2
```

Chebyshev Type II Order

In [ ]:
```python
cheby2_order = cheb2ord(analog_w_p, analog_w_s, delta_s, delta_p, analog=True)[0]
print(f"Chebyshev Type II order: {cheby2_order}")
```

```
Chebyshev Type II order: 2
```

Butterworth Filter

In [ ]:
```python
critical_freq = (analog_w_p)/(1/delta_p**2 - 1)**(2/butterworth_order)

s = sp.symbols('s')

b, a = butter(butterworth_order, critical_freq, 'low', analog=True)

tf = TransferFunction(b, a)

numerator = 0
for i in range(len(b)-1,-1,-1):
    numerator += b[i]*s**i

denominator = 0
for i in range(len(a)-1,-1,-1):
    denominator += a[i]*s**i

tf = numerator/denominator

# print(latex(tf))

z = sp.symbols('z')

transform = (1-z)/(1+z)
tf2 = tf.subs(s, transform)
# print(latex(tf))
```

Butterworth Filter Properties

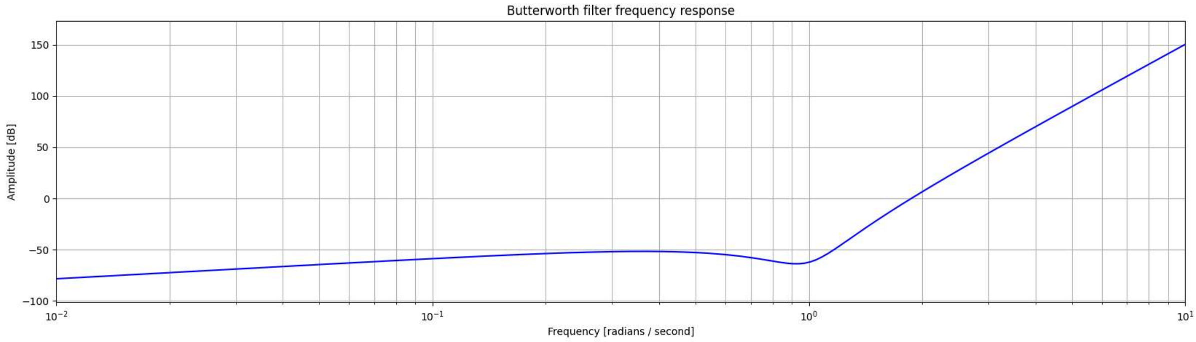
Transfer Function of the Butterworth Filter

$$= \frac{690.36995054605}{690.36995054605s^3 + 156.225257379612s^2 + 17.6762698202767s + 1.0} \tag{1}$$

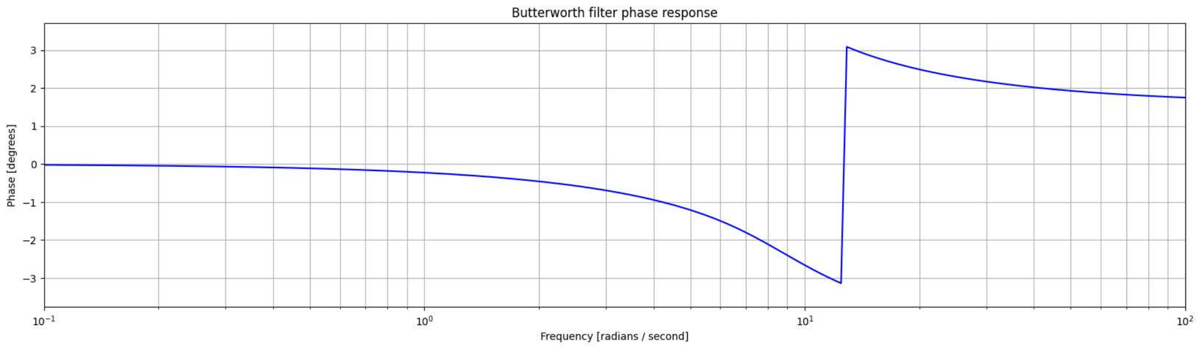
Thus the Transfer function in discrete domain

$$= \frac{690.36995054605}{\frac{690.36995054605(1-z)^3}{(z+1)^3} + \frac{156.225257379612(1-z)^2}{(z+1)^2} + \frac{17.6762698202767\cdot(1-z)}{z+1} + 1.0} \tag{2}$$

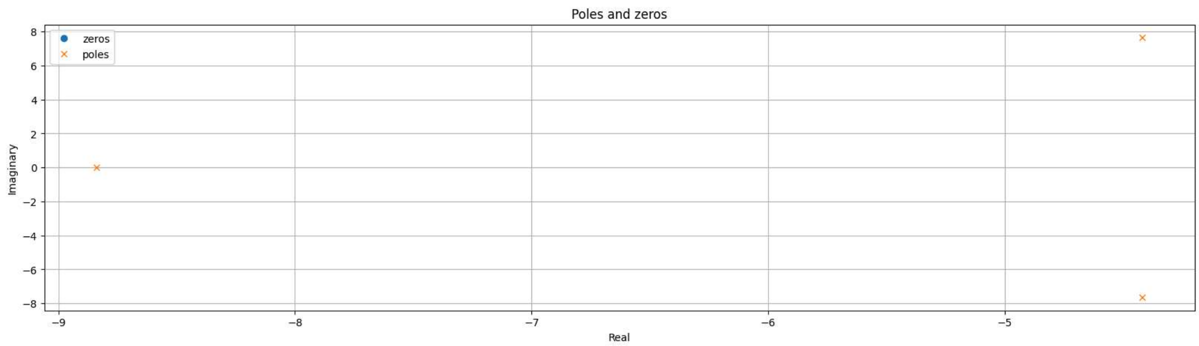
In [ ]:
```python
w, mag = freqs(b, a)
plt.figure(figsize=(20, 5))
plt.semilogx(w, 20 * np.log10(mag), 'b')
plt.title('Butterworth filter frequency response')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()
```



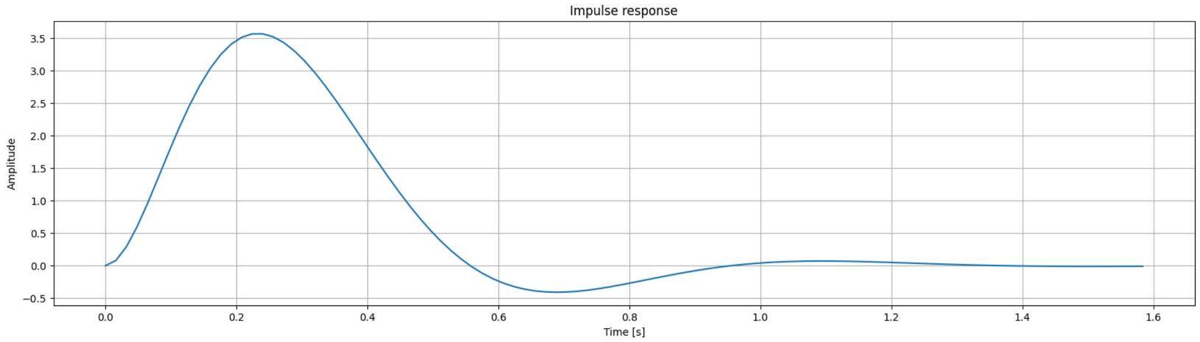
In [ ]:
```
plt.figure(figsize=(20, 5))
plt.semilogx(w, np.angle(mag), 'b')
plt.title('Butterworth filter phase response')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Phase [degrees]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()
```



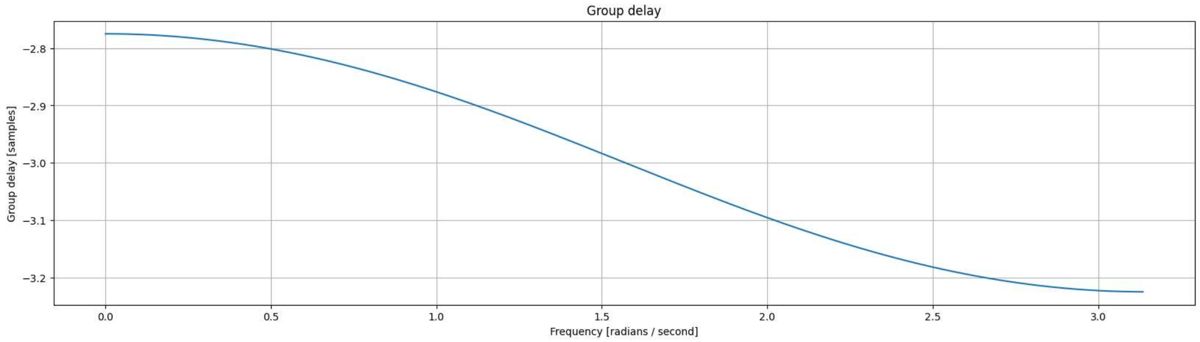
In [ ]:
```
tf = TransferFunction(b, a)
plt.figure(figsize=(20, 5))
plt.plot(tf.zeros.real, tf.zeros.imag, 'o', label='zeros')
plt.plot(tf.poles.real, tf.poles.imag, 'x', label='poles')
plt.title('Poles and zeros')
plt.xlabel('Real')
plt.ylabel('Imaginary')
plt.legend()
plt.grid()
plt.show()
```



In [ ]:
```
plt.figure(figsize=(20, 5))
t, h = impulse(tf)
plt.plot(t, h)
plt.title('Impulse response')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```



In [ ]:
```
w, gd = group_delay((b, a))
plt.figure(figsize=(20, 5))
plt.plot(w, gd)
plt.title('Group delay')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Group delay [samples]')
plt.grid()
plt.show()
```



Chebyshev I Filter

In [ ]:
```
cheby1_critical_freq = analog_w_p

b,a = cheby1(cheby1_order, 1, cheby1_critical_freq, 'low', analog=True)

tf = TransferFunction(b, a)

numerator = 0
for i in range(len(b)-1,-1,-1):
    numerator += b[i]*s**i

denominator = 0
for i in range(len(a)-1,-1,-1):
    denominator += a[i]*s**i

tf = numerator/denominator
# print(latex(tf))

tf = tf.subs(s, transform)
# print(latex(tf))
```

Chebyshev I Properties

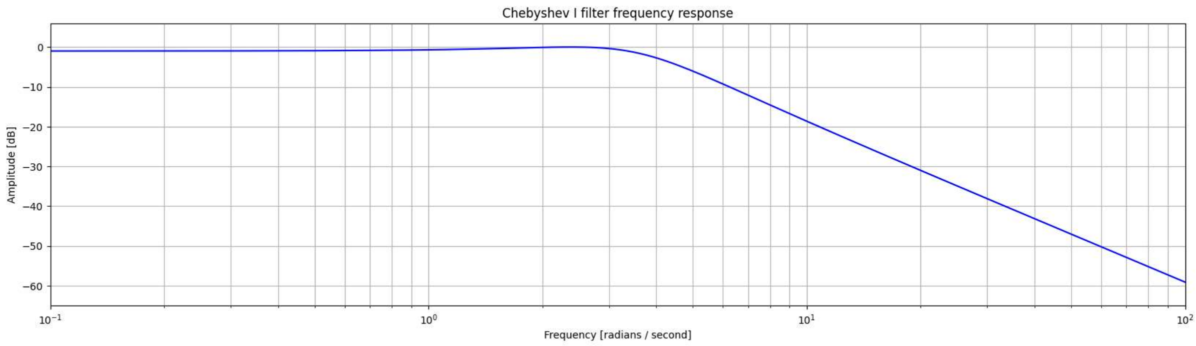
Analog Domain Tranfer Function

$$= \frac{11.1035545142627}{12.4583930733506s^2 + 3.69008973222545s + 1.0} \tag{3}$$

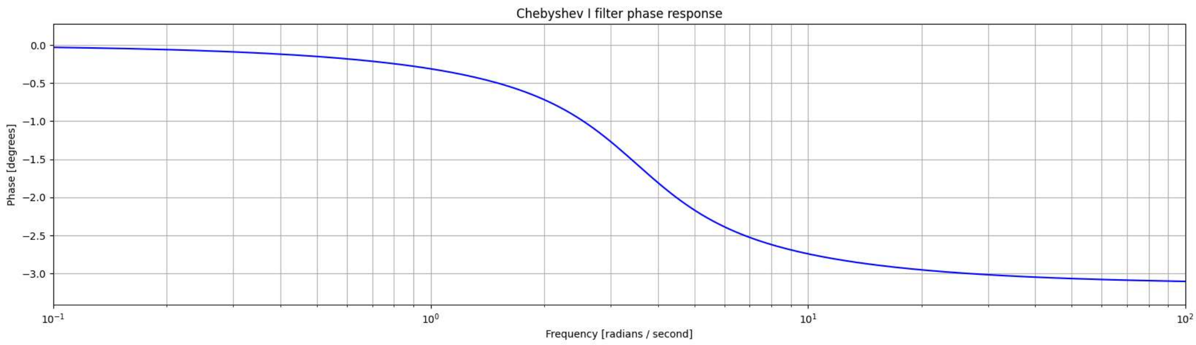
Thus, Discrete Domain Transfer Function

$$= \frac{11.1035545142627}{\frac{12.4583930733506(1-z)^2}{(z+1)^2} + \frac{3.69008973222545\cdot(1-z)}{z+1} + 1.0} \tag{4}$$

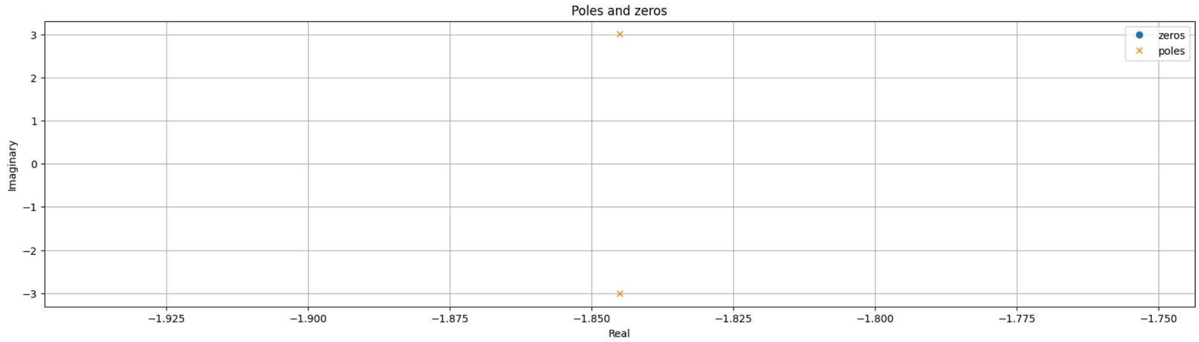
In [ ]:
```
w, mag = freqs(b, a)
plt.figure(figsize=(20, 5))
plt.semilogx(w, 20 * np.log10(mag), 'b')
plt.title('Chebyshev I filter frequency response')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()
```



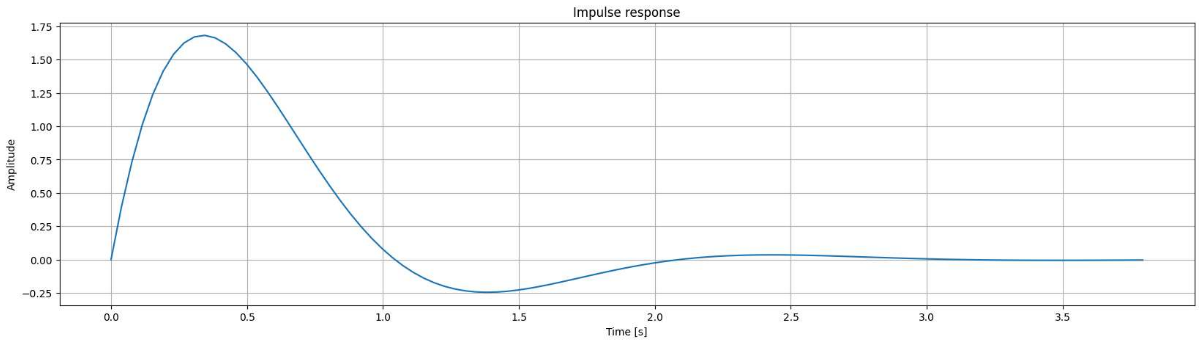
In [ ]:
```
plt.figure(figsize=(20, 5))
plt.semilogx(w, np.angle(mag), 'b')
plt.title('Chebyshev I filter phase response')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Phase [degrees]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()
```



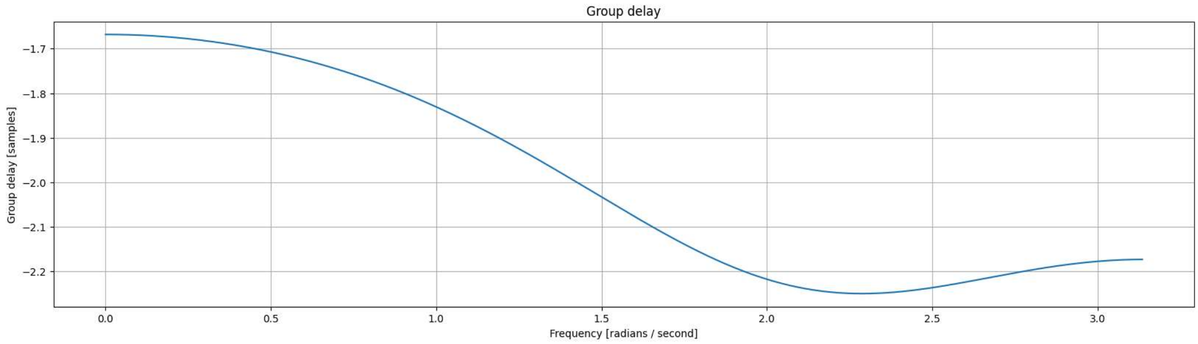
In [ ]:
```
tf = TransferFunction(b, a)
plt.figure(figsize=(20, 5))
plt.plot(tf.zeros.real, tf.zeros.imag, 'o', label='zeros')
plt.plot(tf.poles.real, tf.poles.imag, 'x', label='poles')
plt.title('Poles and zeros')
plt.xlabel('Real')
plt.ylabel('Imaginary')
plt.legend()
plt.grid()
plt.show()
```



In [ ]:
```
t, h = impulse(tf)
plt.figure(figsize=(20, 5))
plt.plot(t, h)
plt.title('Impulse response')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.grid()
plt.show()
```



In [ ]:
```
w, gd = group_delay((b, a))
plt.figure(figsize=(20, 5))
plt.plot(w, gd)
plt.title('Group delay')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Group delay [samples]')
plt.grid()
plt.show()
```



Chebyshev II Filter

In [ ]:
```
cheby2_critical_freq = analog_w_s

b,a = cheby2(cheby2_order, 1, cheby2_critical_freq, 'low', analog=True)

tf = TransferFunction(b, a)

numerator = 0
for i in range(len(b)-1,-1,-1):
```

```
    numerator += b[i]*s**i

denominator = 0
for i in range(len(a)-1,-1,-1):
    denominator += a[i]*s**i

tf = numerator/denominator
# print(latex(tf))

tf = tf.subs(s, transform)
# print(latex(tf))
```

Chebyshev Type II Properties

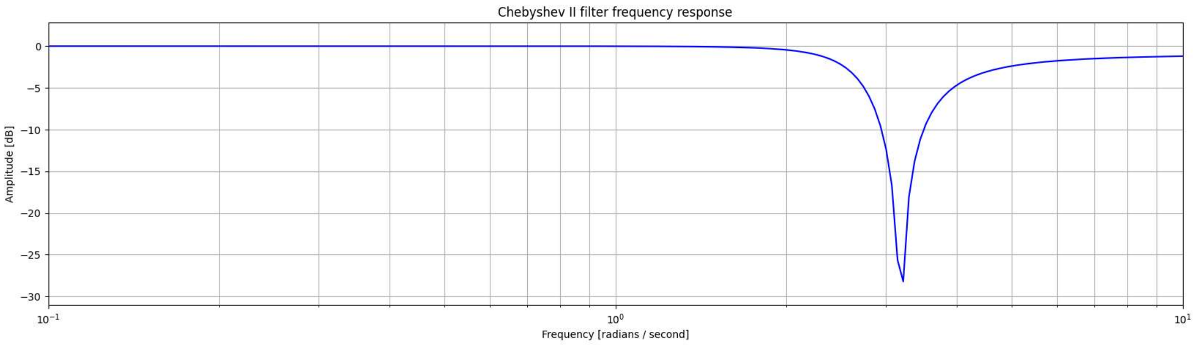
Analog Domain Transfer Function

$$= \frac{9.04490702776034 s^{2} + 0.891250938133745}{9.04490702776034 s^{2} + 1.40258700545558 s + 1.0} \tag{5}$$

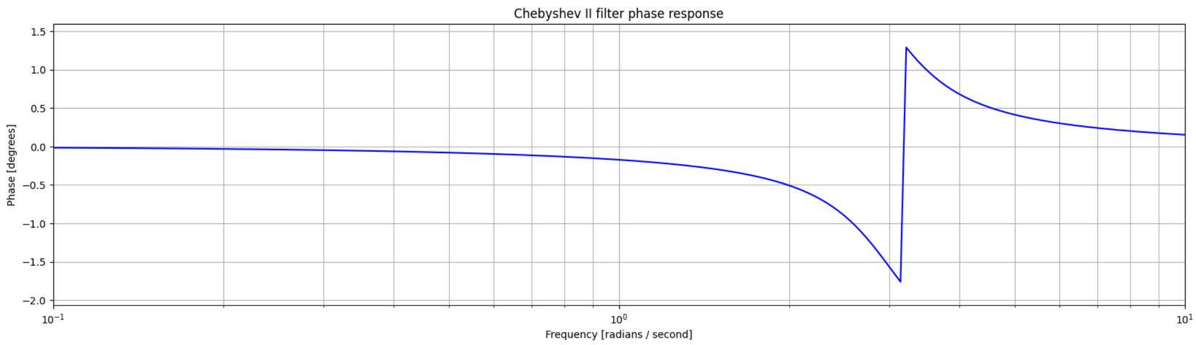
Discrete Domain Transfer Function

$$= \frac{\frac{9.04490702776034 (1 - z)^{2}}{(z + 1)^{2}} + 0.891250938133745}{\frac{9.04490702776034 (1 - z)^{2}}{(z + 1)^{2}} + \frac{1.40258700545558 \cdot (1 - z)}{z + 1} + 1.0} \tag{6}$$

In [ ]:
```
w, mag = freqs(b, a) #type ignore
plt.figure(figsize=(20, 5))
plt.semilogx(w, 20 * np.log10(mag), 'b')
plt.title('Chebyshev II filter frequency response')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Amplitude [dB]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()
```



In [ ]:
```
plt.figure(figsize=(20, 5))
plt.semilogx(w, np.angle(mag), 'b')
plt.title('Chebyshev II filter phase response')
plt.xlabel('Frequency [radians / second]')
plt.ylabel('Phase [degrees]')
plt.margins(0, 0.1)
plt.grid(which='both', axis='both')
plt.show()
```

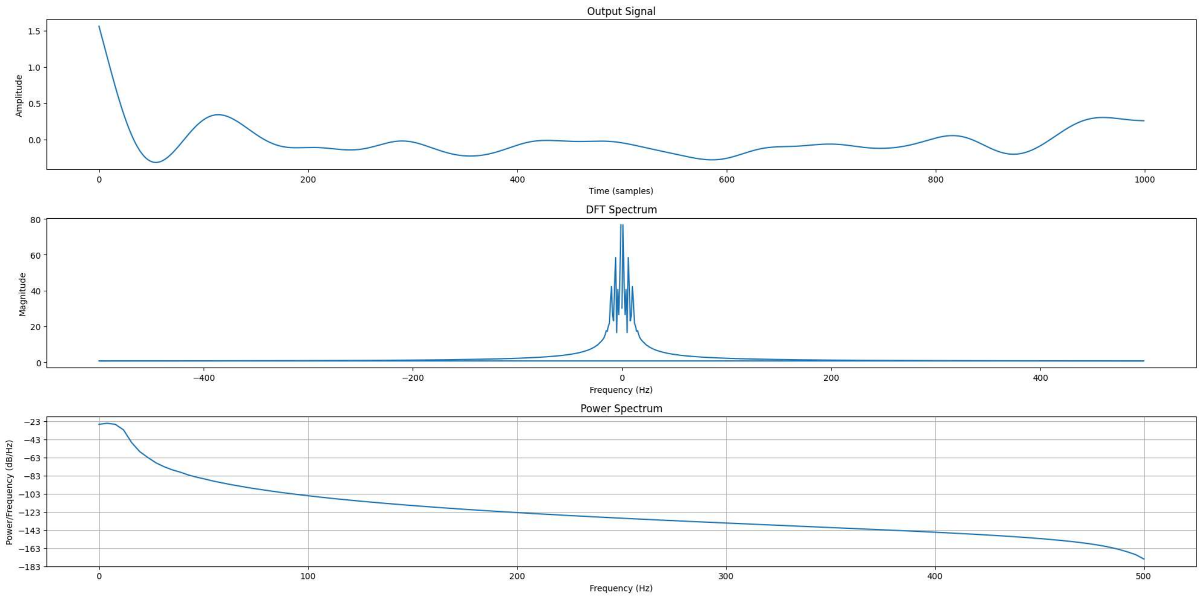


Question 4

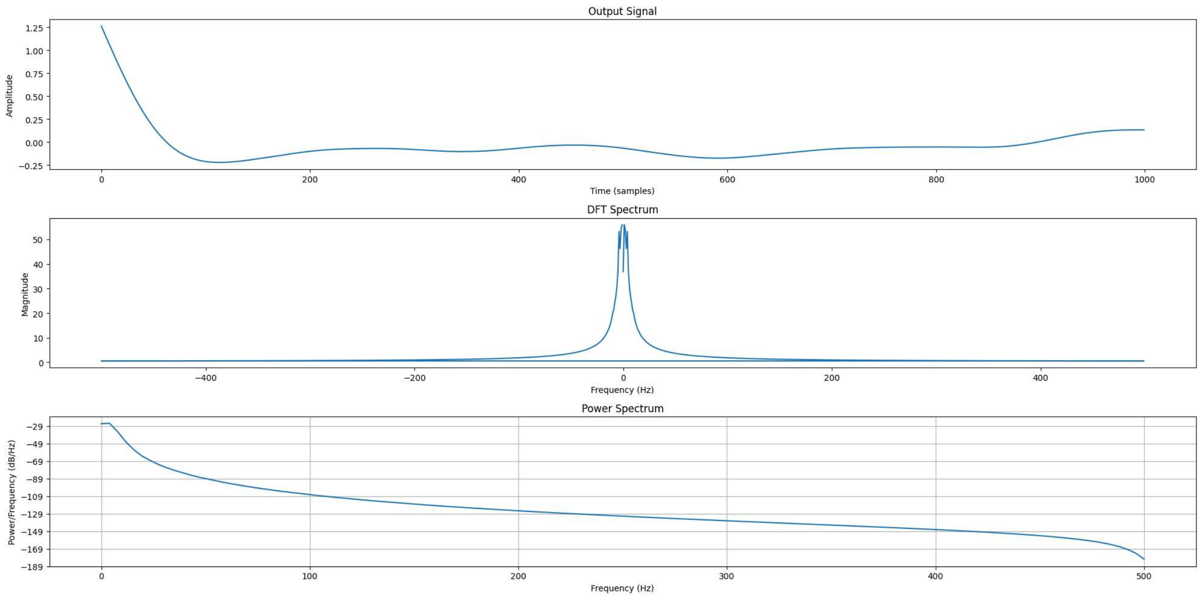
In [ ]:
```
np.random.seed(0)
num_samples = 1000
noise = np.random.normal(0, 1, num_samples)

fs = 1000
cutoff_freq = analog_w_s
b, a = butter(butterworth_order, critical_freq, 'low', fs=fs)

filtered_output = filtfilt(b, a, noise)

dft = np.fft.fft(filtered_output)
freq = np.fft.fftfreq(len(filtered_output), d=1/fs)
```

In [ ]:
```
def plot_all(filtered_output, dft, freq, fs):
    plt.figure(figsize=(20, 10))
    plt.subplot(3, 1, 1)
    plt.plot(filtered_output)
    plt.title('Output Signal')
    plt.xlabel('Time (samples)')
    plt.ylabel('Amplitude')

    # DFT spectrum
    plt.subplot(3, 1, 2)
    plt.plot(freq, np.abs(dft))
    plt.title('DFT Spectrum')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Magnitude')

    # Power spectrum
    plt.subplot(3, 1, 3)
    plt.psd(filtered_output, Fs=fs)
    plt.title('Power Spectrum')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Power/Frequency (dB/Hz)')

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
plot_all(filtered_output, dft, freq, fs)
```



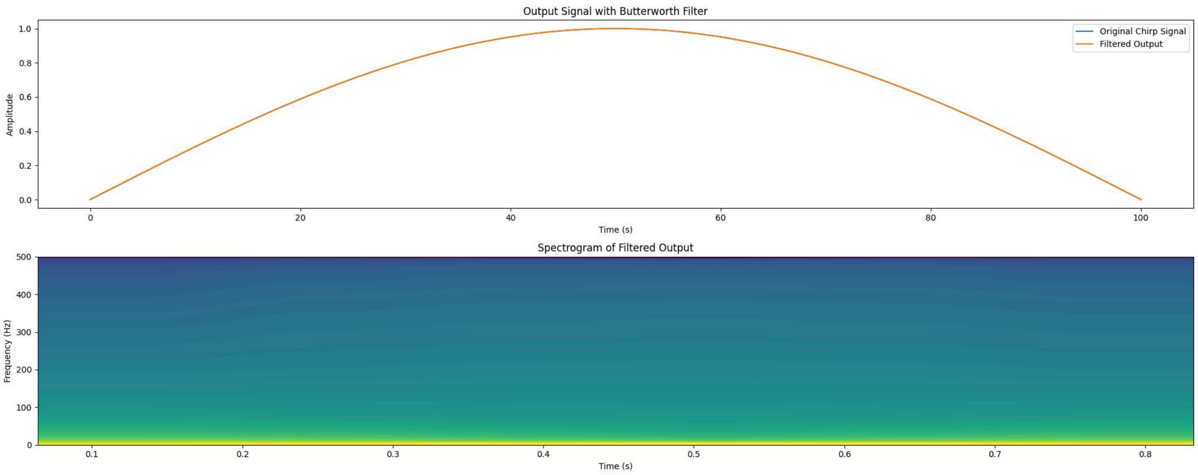
In [ ]:
```
b,a = cheby1(cheby1_order, 1, cheby1_critical_freq, 'low',analog=False, fs=fs)

filtered_output = filtfilt(b, a, noise)

dft = fft(filtered_output)
freq = fftfreq(len(filtered_output), d=1/fs)

plot_all(filtered_output, dft, freq, fs=1000)
```



Question 5

In [ ]:
```
t = np.linspace(0, 100, 1000)
chirp_signal = np.sin(2 * np.pi * t / 200)

order = 3
fs = 1000  # Sampling frequency (Hz)
cutoff_freq = 100  # Cutoff frequency (Hz)

# Design the Butterworth filter
b, a = butter(order, cutoff_freq / (fs / 2), btype='low')

# Apply the filter to the chirp signal
filtered_output = filtfilt(b, a, chirp_signal)
```

```
# Plot the output signal
plt.figure(figsize=(20, 8))
plt.subplot(2, 1, 1)
plt.plot(t, chirp_signal, label='Original Chirp Signal')
plt.plot(t, filtered_output, label='Filtered Output')
plt.title('Output Signal with Butterworth Filter')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()

# Plot the spectrogram
plt.subplot(2, 1, 2)
plt.specgram(filtered_output, Fs=fs, NFFT=256, noverlap=128)
plt.title('Spectrogram of Filtered Output')
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')

plt.tight_layout()
plt.show()
```



FIR Filter

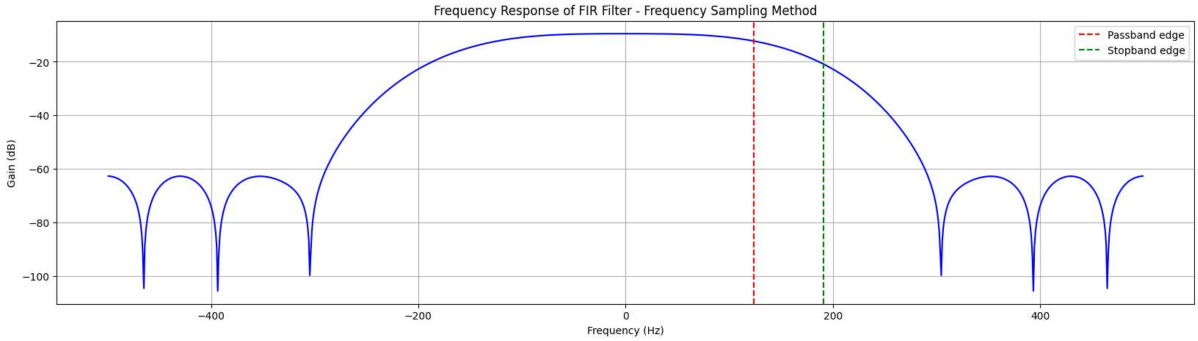
Window Function

In [ ]:

```
fs = 1000
normalized_w_p = analog_w_p / fs
normalized_w_s = analog_w_s / fs
cutoff_freq = (normalized_w_p + normalized_w_s) / 2

b = firwin2(numtaps=15,freq=[0, w_p, w_s, 0.5*fs], gain=[1, 1, 0, 0], nyq=500)

print(b)

w, h = freqz(b, 3)

w_neg = -w[::-1]
h_neg = h[::-1]

w = np.concatenate((w_neg, w))
h = np.concatenate((h_neg, h))

plt.figure(figsize=(20, 5))
plt.plot(0.5 * fs * w / np.pi, 20 * np.log10(abs(h)), 'b')
plt.axvline(w_p, color='r', linestyle='--', label='Passband edge')
plt.axvline(w_s, color='g', linestyle='--', label='Stopband edge')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain (dB)')
```

```
plt.grid()
plt.legend()
plt.title('Frequency Response of FIR Filter - Frequency Sampling Method')
plt.show()
```

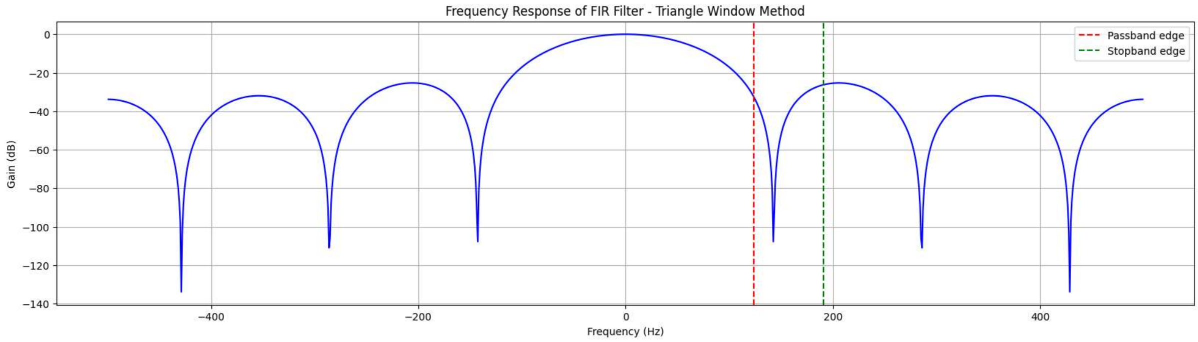
```
[ 0.00157896 -0.001827   -0.01342857 -0.02306562  0.01086879  0.11769634
  0.25265885  0.31555683  0.25265885  0.11769634  0.01086879 -0.02306562
 -0.01342857 -0.001827    0.00157896]
```



Triangular Window

In [ ]:

```
b = firwin(15, cutoff_freq, window='bartlett', pass_zero=True, scale=True, nyq=500)

w, h = freqz(b, 1)

w_neg = -w[::-1]
h_neg = h[::-1]

w = np.concatenate((w_neg, w))
h = np.concatenate((h_neg, h))

plt.figure(figsize=(20, 5))
plt.plot(0.5 * fs * w / np.pi, 20 * np.log10(abs(h)), 'b')
plt.axvline(w_p, color='r', linestyle='--', label='Passband edge')
plt.axvline(w_s, color='g', linestyle='--', label='Stopband edge')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain (dB)')
plt.grid()
plt.legend()
plt.title('Frequency Response of FIR Filter - Triangle Window Method')
plt.show()
```



Hann Window

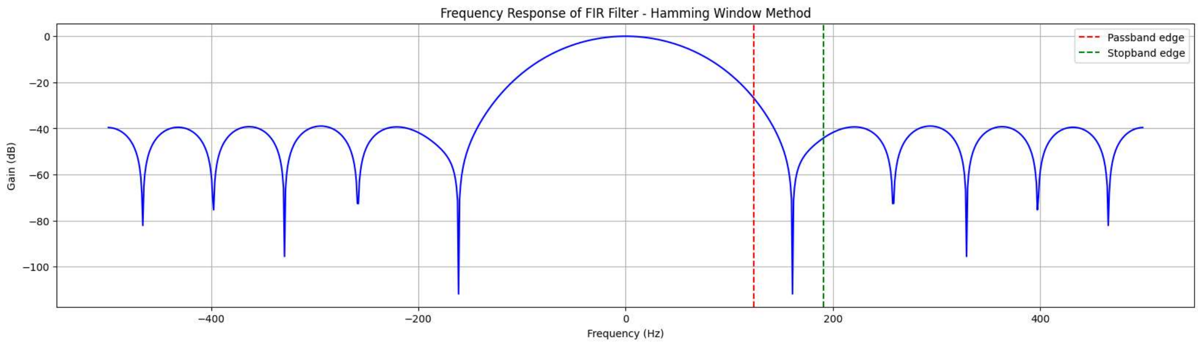
In [ ]:

```
b = firwin(15, cutoff_freq, window='hann', pass_zero=True, scale=True, nyq=500)

w, h = freqz(b, 1)

w_neg = -w[::-1]
h_neg = h[::-1]
```

```
w = np.concatenate((w_neg, w))
h = np.concatenate((h_neg, h))

plt.figure(figsize=(20, 5))
plt.plot(0.5 * fs * w / np.pi, 20 * np.log10(abs(h)), 'b')
plt.axvline(w_p, color='r', linestyle='--', label='Passband edge')
plt.axvline(w_s, color='g', linestyle='--', label='Stopband edge')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain (dB)')
plt.grid()
plt.legend()
plt.title('Frequency Response of FIR Filter - Hann Window Method')
plt.show()
```

Hamming Window

In [ ]:
```
b = firwin(15, cutoff_freq, window='hamming', pass_zero=True, scale=True, nyq=500)

w, h = freqz(b, 1)

w_neg = -w[::-1]
h_neg = h[::-1]

w = np.concatenate((w_neg, w))
h = np.concatenate((h_neg, h))

plt.figure(figsize=(20, 5))
plt.plot(0.5 * fs * w / np.pi, 20 * np.log10(abs(h)), 'b')
plt.axvline(w_p, color='r', linestyle='--', label='Passband edge')
plt.axvline(w_s, color='g', linestyle='--', label='Stopband edge')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain (dB)')
plt.grid()
plt.legend()
plt.title('Frequency Response of FIR Filter - Hamming Window Method')
plt.show()
```



Blackman Window

In [ ]:

```
b = firwin(15, cutoff_freq, window='blackman', pass_zero=True, scale=True, nyq=500)

w, h = freqz(b, 1)

w_neg = -w[::-1]
h_neg = h[::-1]

w = np.concatenate((w_neg, w))
h = np.concatenate((h_neg, h))

plt.figure(figsize=(20, 5))
plt.plot(0.5 * fs * w / np.pi, 20 * np.log10(abs(h)), 'b')
plt.axvline(w_p, color='r', linestyle='--', label='Passband edge')
plt.axvline(w_s, color='g', linestyle='--', label='Stopband edge')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain (dB)')
plt.grid()
plt.legend()
plt.title('Frequency Response of FIR Filter - Blackman Window Method')
plt.show()
```