```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks, lfilter
import scipy.signal as signal

#import os
#print(os.getcwd())  # Prints the current working directory

# Yariv Shossberger -316523406 , Ori Toker -314679713

################################ Q1 ######################################################################

# Load the data from the text file
data = np.loadtxt('314679713-proj_data.txt')

# Define the sampling frequency
fs = 37500  # Hz

# Calculate the time axis
N = len(data)
time = np.arange(N) / fs  # Time in seconds, creates an array of evenly spaced values

# Plot the data
plt.figure(figsize=(10, 6))
plt.plot(time, data)
plt.title('Signal vs Time')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()


################################ Q2 ######################################################################

# Zoom in to the first 2 milliseconds
zoom_time_limit = 0.002  # 2 milliseconds
zoom_samples = int(zoom_time_limit * fs)

plt.figure(figsize=(10, 6))
plt.plot(time[:zoom_samples], data[:zoom_samples])
plt.title('Signal vs Time (Zoomed to 2 ms)')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show()

################################ Q3 ######################################################################

# Compute the DFT using FFT
dft = np.fft.fft(data)
dft_magnitude = np.abs(dft)  # Get the magnitude
dft_magnitude_db = 20 * np.log10(dft_magnitude)  # Convert magnitude to dB

# Frequency axis in Hz
freqs = np.fft.fftfreq(N, 1/fs)

# Normalized frequency axis (0 to 2*pi radians/sample)
normalized_freqs = np.fft.fftfreq(N) * 2 * np.pi

# 3.1 - Plot linear magnitude with frequency in Hz
plt.figure(figsize=(10, 6))
plt.plot(freqs[:N//2], dft_magnitude[:N//2])  # Plot only the positive half of the spectrum
plt.title('DFT (Linear Magnitude, Frequency in Hz)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()

# 3.2 - Plot dB magnitude with frequency in Hz
plt.figure(figsize=(10, 6))
plt.plot(freqs[:N//2], dft_magnitude_db[:N//2])
plt.title('DFT (dB Magnitude, Frequency in Hz)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.grid(True)
plt.show()

# 3.3 - Plot dB magnitude with normalized frequency (0 to 2*pi rad/sample)
plt.figure(figsize=(10, 6))
plt.plot(normalized_freqs[:N//2], dft_magnitude_db[:N//2])
plt.title('DFT (dB Magnitude, Normalized Frequency)')
plt.xlabel('Normalized Frequency (radians/sample)')
plt.ylabel('Magnitude (dB)')
plt.grid(True)
plt.show()

################################ Q5 ######################################################################
```

```python
# Frequency range (0 to 2*pi radians/sample)
omega = np.linspace(0, 2 * np.pi, 75000)

# Frequency response H(e^jω) = 1 - 1.1 * e^(-jω)
H = 1 - 1.1 * np.exp(-1j * omega)

# Calculate the magnitude response (linear scale)
magnitude_response = np.abs(H)

# Plot the magnitude response with normalized frequency
plt.figure(figsize=(10, 6))
plt.plot(omega, magnitude_response)
plt.title('Frequency Response (Linear Magnitude, Normalized Frequency)')
plt.xlabel('Normalized Frequency (radians/sample)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()

################################# Q7 ########################################################################

# Function to plot pole-zero maps
def plot_pole_zero(poles, zeros, title):
    plt.figure(figsize=(6, 6))
    plt.axvline(0, color='black', lw=1)
    plt.axhline(0, color='black', lw=1)
    unit_circle = plt.Circle((0, 0), 1, color='blue', fill=False, linestyle='--', lw=1)
    plt.gca().add_artist(unit_circle)

    plt.scatter(np.real(zeros), np.imag(zeros), s=100, label='Zeros', marker='o')
    plt.scatter(np.real(poles), np.imag(poles), s=100, label='Poles', marker='x')

    plt.xlim(-2, 2)
    plt.ylim(-2, 2)
    plt.title(title)
    plt.xlabel('Real')
    plt.ylabel('Imaginary')
    plt.grid(True)
    plt.legend()
    plt.show()

# Original system H(z)
poles_H = [0]  # pole at z = 0
zeros_H = [1.1]  # zero at z = 1.1
plot_pole_zero(poles_H, zeros_H, 'Pole-Zero Map for H(z)')

# Minimum-phase system Hmin(z)
poles_Hmin = [0]  # pole at z = 0
zeros_Hmin = [0.909]  # zero at z = 0.909
plot_pole_zero(poles_Hmin, zeros_Hmin, 'Pole-Zero Map for Hmin(z)')

# All-pass system Hap(z)
poles_Hap = [0.909]  # pole at z = 0.909
zeros_Hap = [1.1]  # zero at z = 1.1
plot_pole_zero(poles_Hap, zeros_Hap, 'Pole-Zero Map for Hap(z)')

################################# Q8 ########################################################################

# Hmin(z) = (z - 0.909) / z
H_min = (np.exp(1j * omega) - 0.909) / np.exp(1j * omega)

# Hap(z) = (z - 1.1) / (z - 0.909)
H_ap = (np.exp(1j * omega) - 1.1) / (np.exp(1j * omega) - 0.909)

# Magnitude of the frequency responses
H_min_magnitude = np.abs(H_min)
H_ap_magnitude = np.abs(H_ap)

# Plot the magnitude response of Hmin(z)
plt.figure(figsize=(10, 6))
plt.plot(omega, H_min_magnitude)
plt.title('Frequency Response of Hmin(z) (Linear Magnitude, Normalized Frequency)')
plt.xlabel('Normalized Frequency (radians/sample)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()

# Plot the magnitude response of Hap(z)
plt.figure(figsize=(10, 6))
plt.plot(omega, H_ap_magnitude)
plt.title('Frequency Response of Hap(z) (Linear Magnitude, Normalized Frequency)')
plt.xlabel('Normalized Frequency (radians/sample)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()

################################# Additional Task ###########################################################
```

```
# Define the magnitude and phase of the frequency response
magnitude =np.sqrt(2.21 - 2.2 * np.cos(omega))
phase = omega  # Linear phase term

# Combine magnitude and phase for the complex frequency response
H = magnitude * np.exp(-(1j * phase))

# Plot magnitude and phase
fig, axs = plt.subplots(2, 1, figsize=(10, 8))

# Plot magnitude response
axs[0].plot(omega, magnitude, label="Magnitude |H(e^(jw))|")
axs[0].set_title("Magnitude Response of H(e^(jw))")
axs[0].set_xlabel("Frequency (ω)")
axs[0].set_ylabel("Magnitude")
axs[0].grid(True)
axs[0].legend()

# Plot phase response
axs[1].plot(omega, phase, label="Phase ∠H(e^(jw))", color='orange')
axs[1].set_title("Phase Response of H(e^(jw))")
axs[1].set_xlabel("Frequency (ω)")
axs[1].set_ylabel("Phase (radians)")
axs[1].grid(True)
axs[1].legend()

# Show plots
plt.tight_layout()
plt.show()

# Define the transfer function in terms of numerator and denominator coefficients
# H(z) = 1 - 1.1 z^(-1)
num = [1, -1.1]  # Corresponding to 1 - 1.1 z^(-1)
den = [1]        # No poles other than the origin

# Compute zeros, poles, and gain of the system
zeros, poles, _ = signal.tf2zpk(num, den)

# Plot zeros and poles on the complex plane
fig, ax = plt.subplots(figsize=(6, 6))
ax.plot(np.real(zeros), np.imag(zeros), 'o', label='Zeros', markersize=10)
ax.plot(np.real(poles), np.imag(poles), 'x', label='Poles', markersize=10)
ax.add_patch(plt.Circle((0, 0), 1, color='gray', fill=False, linestyle='--', label='Unit Circle'))

# Set plot limits and labels
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-1.5, 1.5)
ax.axhline(0, color='black', linewidth=0.5)
ax.axvline(0, color='black', linewidth=0.5)
ax.set_xlabel('Real Part')
ax.set_ylabel('Imaginary Part')
ax.set_title('Pole-Zero Plot of H(z)')
ax.legend()
ax.grid(True)

plt.show()

################################ Q10 ######################################################################

x = data # for better representation

# Initialize the corrected signal array (same length as input signal)
y = np.zeros_like(x)
y[0] = x[0]  # Start with the first input value

# Apply the difference equation y[n] = x[n] + 1.1 * y[n-1]
for n in range(1, len(x)):  # Start from n=1 since we need y[n-1]
    y[n] = x[n] + 0.8 * y[n-1]  # Use a smaller coefficient to avoid overflow

################################ Q11 ######################################################################

# Define the number of samples to ignore (transient region)
transient_samples = 1000  # Adjust this number based on visual inspection

# Remove the transient response by slicing the array
y_corrected = y[transient_samples:]  # Remove the transient region

# Plot the output signal after removing the transient response
plt.figure(figsize=(10, 6))
plt.plot(y_corrected, label='Corrected Signal (After Transient)', color='orange')
plt.title('Corrected Signal (After Transient Response)')
plt.xlabel('Sample Index (After Transient)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.legend()
plt.show()
```

```python
plt.plot(y_corrected[300:400], label='Corrected Signal (After Transient, Zoomed)', color='orange')  # Zoom in on 100 samples range
plt.title('Corrected Signal (After Transient, Zoomed)')
plt.xlabel('Sample Index (After Transient)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.legend()
plt.show()

################################# Q12 #######################################################################

# Perform the FFT (Discrete Fourier Transform) on the corrected signal
n = len(y_corrected)  # Length of the signal
y_fft = np.fft.fft(y_corrected)
frequencies = np.fft.fftfreq(n, d=1/fs)  # Frequency bins

# Take only the positive half of the frequencies (since FFT is symmetric for real signals)
y_fft = y_fft[:n//2]
frequencies = frequencies[:n//2]

# 12.1 - Plot DFT with Linear Magnitude Scale
plt.figure(figsize=(10, 6))
plt.plot(frequencies, np.abs(y_fft), label='Linear Magnitude', color='blue')
plt.title('DFT of Corrected Signal (Linear Magnitude Scale)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.legend()
plt.show()

# 12.2 - Plot DFT with dB Magnitude Scale
plt.figure(figsize=(10, 6))
plt.plot(frequencies, 20 * np.log10(np.abs(y_fft)), label='dB Magnitude', color='orange')
plt.title('DFT of Corrected Signal (dB Magnitude Scale)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (dB)')
plt.grid(True)
plt.legend()
plt.show()

################################# Q13 #######################################################################

# Calculate the magnitude of the DFT
magnitude = np.abs(y_fft)

# Find peaks in the magnitude spectrum
peaks, properties = find_peaks(magnitude, height=1000)  # Adjust height based on inspection

# Print detected peak frequencies
detected_frequencies = frequencies[peaks]
print("Detected frequencies (Hz):", detected_frequencies)

# Plot the magnitude spectrum with detected peaks
plt.figure(figsize=(10, 6))
plt.plot(frequencies, magnitude, label='DFT Magnitude')
plt.plot(frequencies[peaks], magnitude[peaks], 'rx', label='Detected Peaks')
plt.title('DFT with Detected Frequencies')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.legend()
plt.show()

################################# Q16 #######################################################################
n = np.arange(201)
h_bandpass = 0.14 * np.sinc(0.14 * (n - 100)) - 0.1266 * np.sinc(0.1266 * (n - 100))
h_window = 0.54 - 0.46 * np.cos(np.pi * n / 100)
h_windowed = h_bandpass * h_window


def calculate_manual_frequency_response(impulse_response, num_points):
    """
    Calculate the frequency response of a filter manually by performing the DFT.

    Parameters:
    impulse_response (numpy array): The impulse response of the filter.
    num_points (int): Number of points for the DFT (frequency resolution).

    Returns:
    w (numpy array): Normalized frequency values from 0 to 2π (radians/sample).
    H (numpy array): Magnitude of the frequency response.
    """
    # Length of the impulse response
    N = len(impulse_response)

    # Initialize the frequency response array
    H = np.zeros(num_points, dtype=complex)
```

```python
    # Compute DFT manually for each frequency bin
    for k in range(num_points):
        omega_k = (2 * np.pi * k) / num_points  # Normalized frequency (radians/sample)
        for n in range(N):
            H[k] += impulse_response[n] * np.exp(-1j * omega_k * n)

    # Compute magnitude of the frequency response
    magnitude = np.abs(H)

    # Frequency values (normalized from 0 to 2π)
    w = np.linspace(0, 2 * np.pi, num_points)

    return w, magnitude

# Example usage of the function with the bandpass filter impulse response
# Assuming `h_windowed` is the impulse response from the previous design

M = 201  # Number of points for the DFT (frequency resolution)
w, h = calculate_manual_frequency_response(h_windowed, M)

# Adjust the plot to normalize the frequency from 0 to 2pi
plt.plot(w, np.abs(h), label='Linear Magnitude')
plt.title('Frequency Response of the Designed Filter')
plt.xlabel('Normalized Frequency (radians/sample)')
plt.ylabel('Magnitude')
plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi],
           [r'$0$', r'$\pi/2$', r'$\pi$', r'$3\pi/2$', r'$2\pi$'])
plt.grid()
plt.show()

################################# Q18 #########################################################################

# Apply the filter to the input signal
filtered_signal = lfilter(h_windowed, 1.0, y_corrected)

################################# Q19 #########################################################################

# Plot the filtered signal
plt.figure()
plt.plot(filtered_signal, label='Filtered Signal')
plt.title('Filtered Signal')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

plt.figure()
plt.plot(filtered_signal[300:400], label='Filtered Signal')  # Zoom in on 100 samples range
plt.title('Filtered Signal (Zoomed)')
plt.xlabel('Sample Index')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

################################# Q20 #########################################################################

# Perform the FFT on the filtered signal
filtered_signal_fft = np.fft.fft(filtered_signal)
N_filtered = len(filtered_signal)  # Length of the filtered signal
frequencies_filtered = np.fft.fftfreq(N_filtered, d=1/fs)  # Frequency bins

# Take only the positive half of the FFT (since FFT is symmetric for real signals)
filtered_signal_fft = filtered_signal_fft[:N_filtered//2]
frequencies_filtered = frequencies_filtered[:N_filtered//2]

# Plot the DFT of the filtered signal (Linear Magnitude Scale)
plt.figure(figsize=(10, 6))
plt.plot(frequencies_filtered, np.abs(filtered_signal_fft), label='Filtered Signal FFT', color='blue')
plt.title('DFT of Filtered Signal (Linear Magnitude Scale)')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.legend()
plt.show()
```