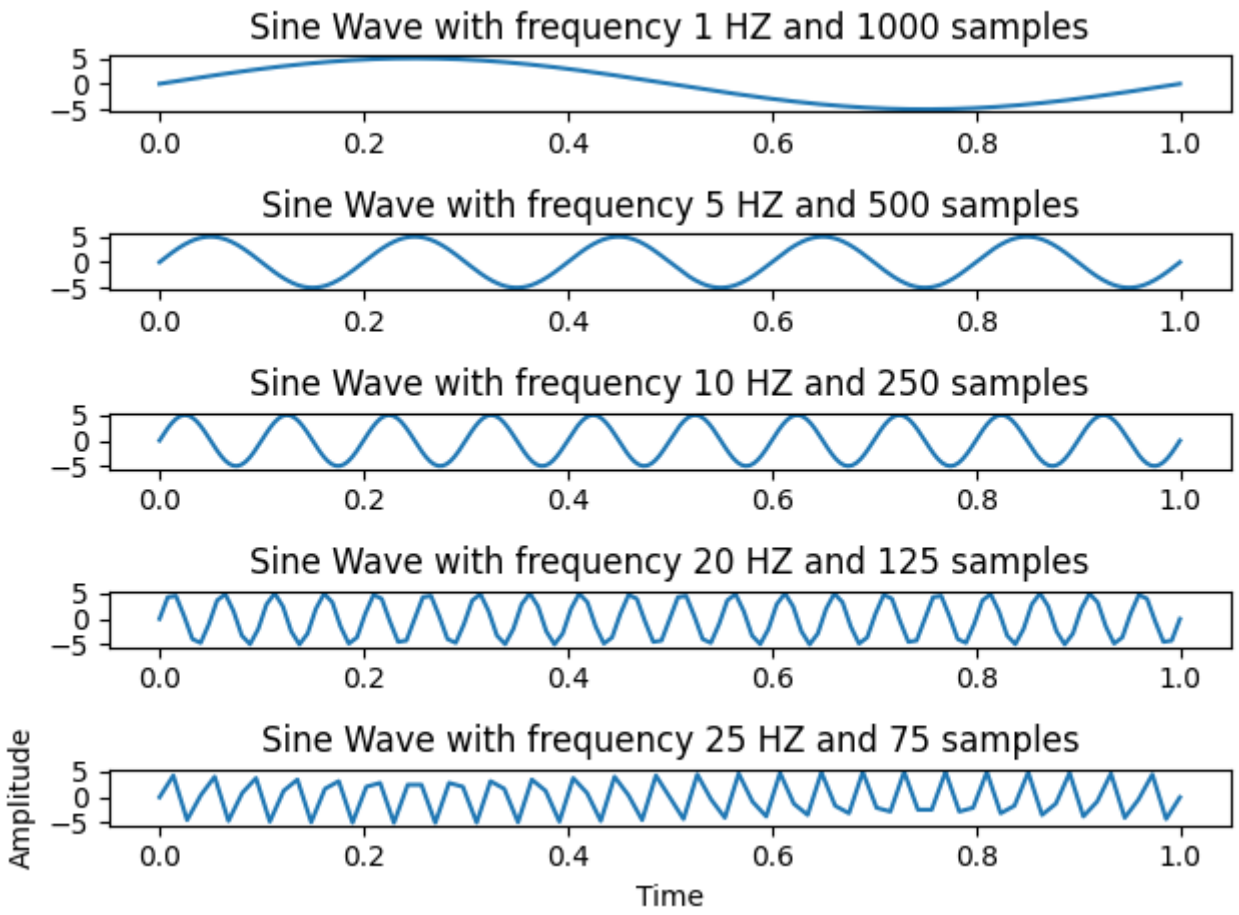
Communication Systems Core Competencies Report

Task 1

Aim:
The aim of this task was to create a function that is the discrete representation of f(t) = a * sin(2*pi*v*t), where v is the frequency in hertz for the wave and a is the amplitude, and then plot this equation in a graph for a range of different values.

Method:
Made a sin wave function with 3 distinct inputs, a, v, and t. For a range of values I created two arrays, v[] and ns[] (number of samples). I use a for loop to iterate through each array to create 5 sin waves with different input parameters. In each iteration of the for loop I create a subplot and add the newly created sin wave.

Results:



Conclusion:
Overall a higher sampling rate leads to a smoother, continuous wave. From these samples I can derive that when the sampling rate is less than 6X the frequency of the wave the reading begins to degrade and becomes discontinuous. Interestingly as the sampling rate becomes 3X the frequency a new low frequency wave begins to form on top of the original high frequency wave.

Task 2

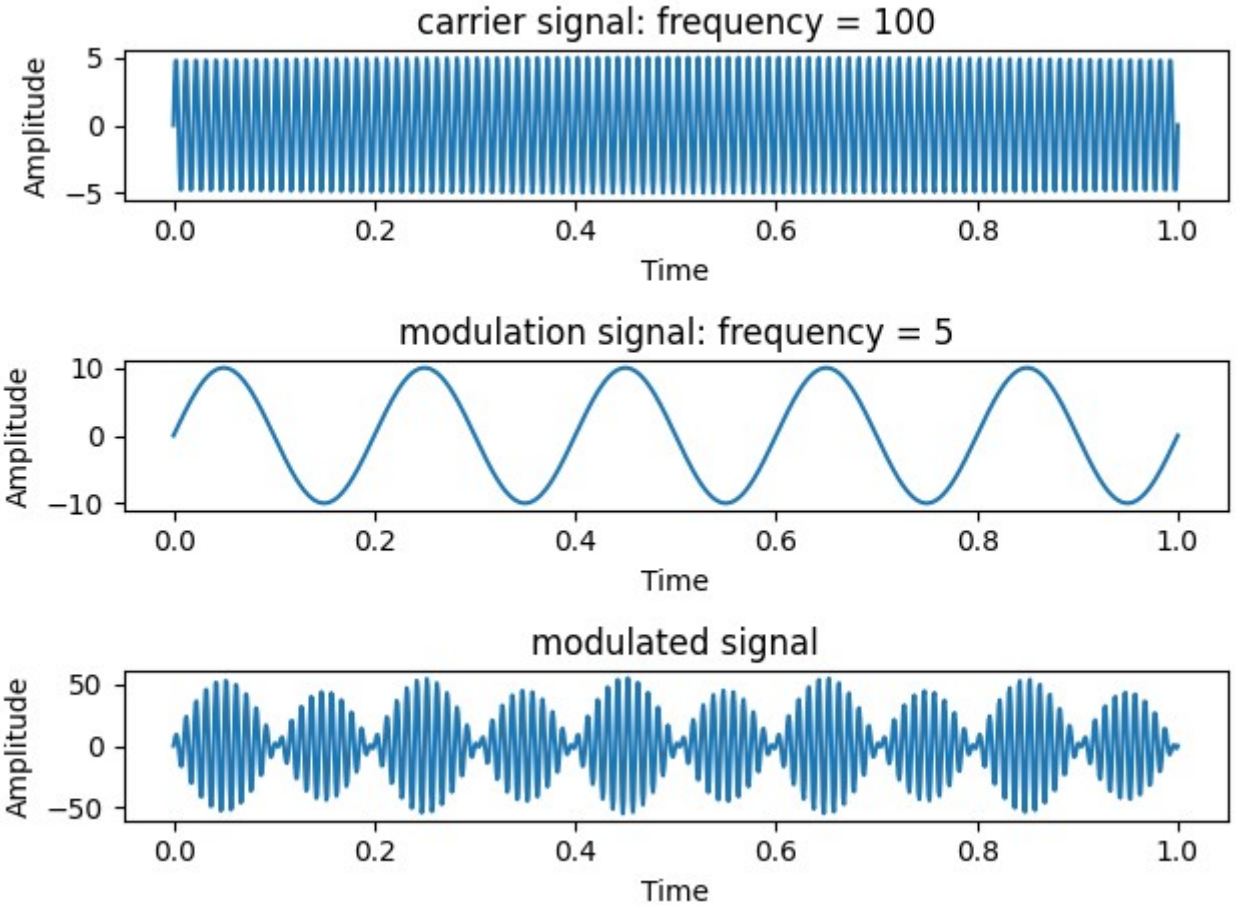
Aim:

The aim of this task was to replicate the amplitude modulation of a carrier wave with frequency y(t) = [DC + msin(2*pi*vm*t)]*A*sin(2*pi*vc*t). DC is the DC offset, vm is the modulation frequency, vc is the carrier frequency, m is modulation amplitude, and A is the single amplitude.

Method:

I created a function to produce an amplitude modulated wave using my sin function from task 1. The function simply calculates the product of the modulation and carrier waves to output an amplitude modulated wave.

Results:



Conclusion:

The results of this simulation show that the modulated signal is the exact product of the two input signals. The peaks amplify each other and the dips cancel each other out. This produces a wave that looks as if it is two waves at once. This wave can now be put into a Fourier transform to produce the original signals.

Task 3

Aim:

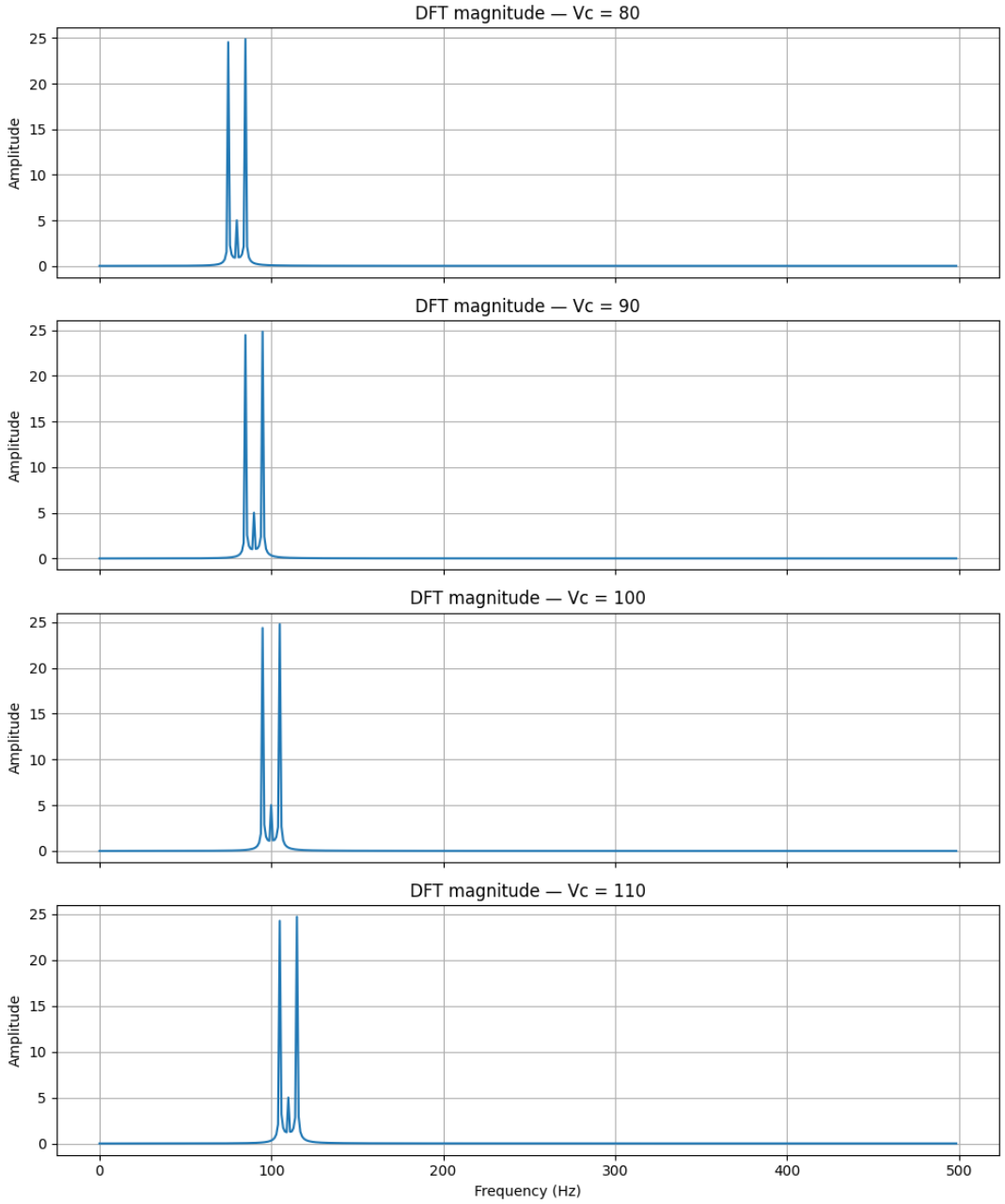
The aim of this task was to create a Discrete Fourier Transform function to find the frequency component of the modulated signal produced in task 2. We are then to compare our function to the fast Fourier transform function included in the numpy library. The function we are to replicate is as follows:

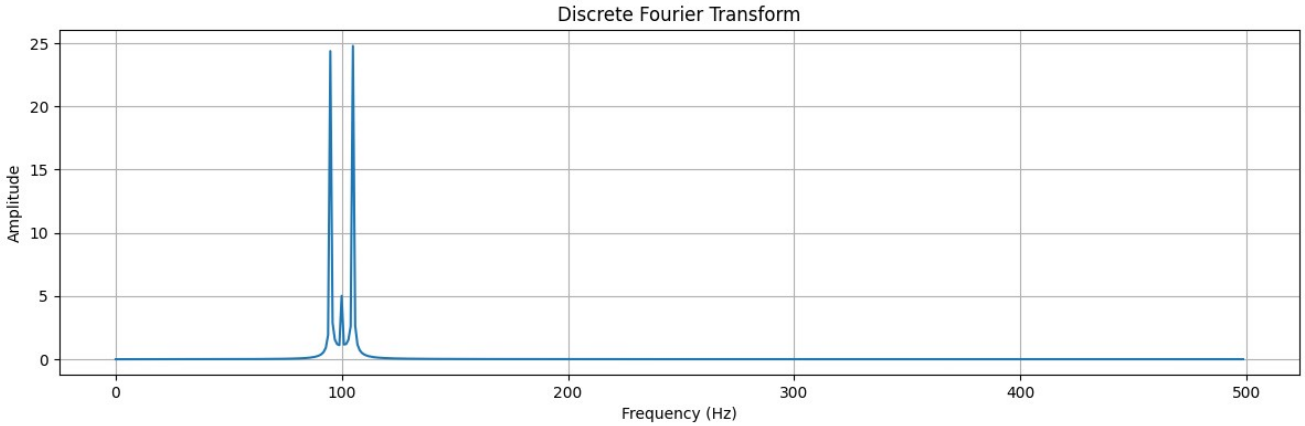
$$F_k = 1/N \sum_{n=0}^{N-1} f_n \cdot e^{-\frac{i2\pi kn}{N}}$$

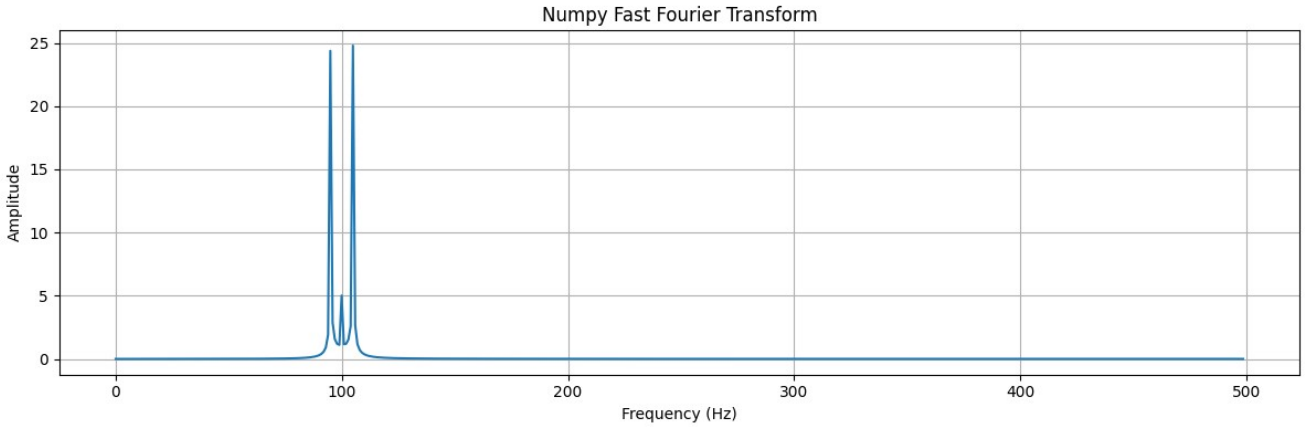
Method:

My discrete Fourier transform function takes a single input of a 1D array (in this case that array is the values from discrete sampling of a modulated signal) and applies the Fourier transform mathematics to output the frequency domain of the signal. The function creates an array (N), two more arrays n and k which are the row indices and column indices of N, and uses these values to execute the Equation given above. Our exponential e is calculated np.exp(). e.dot(x) multiplies all values of the matrix e and all values of the array x to output our final array X which gives us our frequency domain.

Results:







Conclusion:

The DFT function outputs the same results as the numpy fast fourier transform function. For different input frequencies the peaks in the frequency domain increase and decrease accordingly. With lower amplitude the amplitude of the frequency component will also decrease.

Task 4

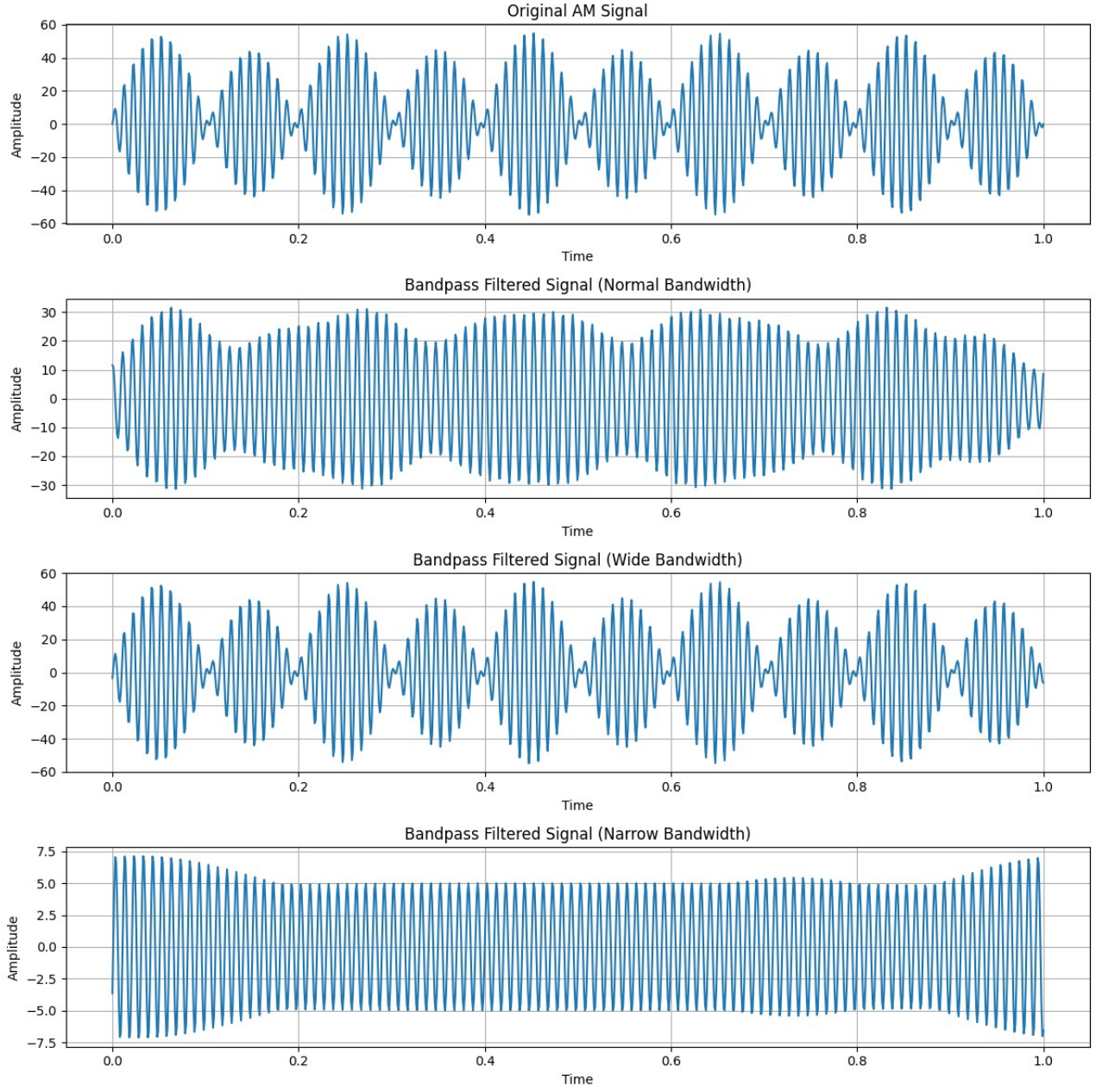
Aim:

The aim of this task was to create a band pass filter and pass the modulated signal from task 2 through it. Next we must use an inverse Fourier transform to reproduce the time series data from the frequency domain.

Method:

the bandpass filter function takes an input signal, sampling frequency low cut frequency, and high cut frequency. It uses np.fft to convert to frequency domain, and uses a mask to filter the signal only allowing frequencies between the high cut and the low cut to pass. At the end of the function the frequency domain is multiplied by the mask and an inverse Fourier transform is applied to output the time series of the filtered signal.

Results:



Conclusion:

Normal bandwidth applies balanced filtering, narrow bandwidth applies aggressive filtering which may result in loss of data, and wide bandwidth applies weak filtering which allows more frequencies to pass and may leave noise in the signal.

The bandwidth significantly affects the quality of the output signal. The nominal choice for this example seems to be normal bandwidth.

Task 1 Code

```python
import matplotlib.pyplot as plt
import numpy as np

a = 5 #amplitude
v = [1, 5, 10, 20, 25] #frequencies

st = 0 #start time
et = 1 #end time
ns = [1000, 500, 250, 125, 75] #number of samples

def sin(a, v, t):
    sin = a * np.sin(2 * np.pi * v * t) #Sine Wave Function
    return sin

#Formatting the graph
for i in range(len(v)):
    t = np.linspace(st, et, ns[i]) #time samples
    plt.subplot(len(v), 1, i+1) #(rows, cols, index)
    plt.plot(t, sin(a, v[i], t)) #plot the sine wave
    plt.title(f"Sine Wave with frequency {v[i]} HZ and {t.size} samples")

plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```

Task 2 Code

```python
import matplotlib.pyplot as plt
import numpy as np

A = 5 #amplitude
Vm = 5 #modulation frequency
m = 10 #modulation amplitude
DC = 1 #DC offset
Vc = 100 #carrier frequency

st = 0 #start time
et = 1 #end time
ns = 1000 #number of samples
t = np.linspace(st, et, ns)

def sin(a, v, t):
    sin = a * np.sin(2 * np.pi * v * t) #Sine Wave Function
    return sin

modulation = sin(m, Vm, t)
carrier = sin(A, Vc, t)

def amplitude_modulation(DC, modulation, carrier):
    am = (DC + modulation) * carrier
    return am

plt.subplot(3, 1, 1)
plt.plot(t, carrier)
plt.title(f"carrier signal: frequency = {Vc}")
plt.xlabel("Time")
plt.ylabel("Amplitude")

plt.subplot(3, 1, 2)
plt.plot(t, modulation)
plt.title(f"modulation signal: frequency = {Vm}")
plt.xlabel("Time")
plt.ylabel("Amplitude")

plt.subplot(3, 1, 3)
plt.plot(t, amplitude_modulation(DC, modulation, carrier))
plt.title(f"modulated signal")
plt.xlabel("Time")
plt.ylabel("Amplitude")

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

Task 3 Code

```python
import matplotlib.pyplot as plt
import numpy as np

A = 5 #amplitude
Vm = 5 #modulation frequency
m = 10 #modulation amplitude
DC = 1 #DC offset
Vc = 100 #carrier frequency
k = 1 #frequency index

st = 0 #start time
et = 1 #end time
N = 1000 #number of samples
n = np.linspace(st, et, N)

def sin(a, v, t):
    sin = a * np.sin(2 * np.pi * v * t) #Sine Wave Function
    return sin

def amplitude_modulation(m, Vm, A, Vc, n):
    """
    Function to generate an amplitude
    modulated signal based on a carrier signal
    and modulation signal
    """

    modulation = sin(m, Vm, n)
    carrier = sin(A, Vc, n)
    am = (DC + modulation) * carrier
    return am

def DFT(x): #Discrete Fourier Transform
    x = np.asarray(x, dtype=complex)
    N = len(x)
    n = np.arange(N)
    k = n.reshape((N, 1))
    e = np.exp(-2j * np.pi * k * n / N)

    X = e.dot(x)

    return X

def produce_plots(m, Vm, A, Vc, n):
    # Calculate frequency axis
    N = len(n)
    freq = np.fft.fftfreq(N, (n[1] - n[0]))

    # Calculate DFT and FFT
    dft_result = DFT(amplitude_modulation(m, Vm, A, Vc, n))
    fft_result = np.fft.fft(amplitude_modulation(m, Vm, A, Vc, n))

    # Plot results
    plt.figure(figsize=(12, 8))

    # DFT Plot
    plt.subplot(2, 1, 1)
    plt.title("Discrete Fourier Transform")
    plt.plot(freq[:N//2], 2/N * np.abs(dft_result[:N//2]))
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.grid(True)

    # FFT Plot
    plt.subplot(2, 1, 2)
    plt.title("Numpy Fast Fourier Transform")
    plt.plot(freq[:N//2], 2/N * np.abs(fft_result[:N//2]))
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude")
    plt.grid(True)

    plt.tight_layout()
    plt.show()

def produce_multiple_dft_plots(param_name, values, m, Vm, A, Vc, n):
    """
    Produce multiple DFT plots varying one parameter.
    param_name: 'm', 'Vm', 'A', or 'Vc'
    values: iterable of values to use for that parameter
    Other parameters default to the module-level values.
    """
    N = len(n)
    freq = np.fft.fftfreq(N, (n[1] - n[0]))[:N//2]
    values = list(values)
    rows = len(values)
    fig, axes = plt.subplots(rows, 1, figsize=(10, 3*rows), sharex=True)
    if rows == 1:
        axes = [axes]
    for ax, val in zip(axes, values):
        # select parameters
        params = {'m': m, 'Vm': Vm, 'A': A, 'Vc': Vc}
        params[param_name] = val
        am = amplitude_modulation(params['m'], params['Vm'], params['A'], params['Vc'], n)
        X = DFT(am)
        ax.plot(freq, 2/N * np.abs(X[:N//2]))
        ax.set_title(f"DFT magnitude — {param_name} = {val}")
        ax.set_ylabel("Amplitude")
        ax.grid(True)
    axes[-1].set_xlabel("Frequency (Hz)")
    plt.tight_layout()
    plt.show()

# Example: vary carrier frequency
#produce_multiple_dft_plots('Vc', [80, 90, 100, 110])
produce_plots(m, Vm, A, Vc, n)
```

Task 4 Code

```python
import numpy as np
import matplotlib.pyplot as plt

# Signal parameters (from Task 3)
A = 5  # amplitude
Vm = 5  # modulation frequency
m = 10  # modulation amplitude
DC = 1  # DC offset
Vc = 100  # carrier frequency
k = 1  # frequency index

st = 0  # start time
et = 1  # end time
N = 1000  # number of samples
n = np.linspace(st, et, N)

def sin(a, v, t):
    return a * np.sin(2 * np.pi * v * t)

def amplitude_modulation(m, Vm, A, Vc, n):
    modulation = sin(m, Vm, n)
    carrier = sin(A, Vc, n)
    am = (DC + modulation) * carrier
    return am

def bandpass_filter(signal, fs, lowcut, highcut):
    N = len(signal)
    # Get frequency components
    fft_signal = np.fft.fft(signal)
    freq = np.fft.fftfreq(N, 1/fs)

    # Create bandpass mask
    mask = np.zeros(N)
    mask[np.abs(freq) >= lowcut] = 1
    mask[np.abs(freq) >= highcut] = 0

    # Apply filter
    filtered_signal = np.fft.ifft(fft_signal * mask)
    return filtered_signal.real

# Generate signal
am_signal = amplitude_modulation(m, Vm, A, Vc, n)

# Apply bandpass filter
# Filter frequencies between carrier-modulation and carrier+modulation
lowcut = Vc - Vm
highcut = Vc + Vm
filtered_signal = bandpass_filter(am_signal, N, lowcut, highcut)

# Plot original and filtered signals
def plot():
    # Define different bandwidths
    bandwidths = [
        {'lowcut': Vc - Vm, 'highcut': Vc + Vm, 'label': 'Normal'},
        {'lowcut': Vc - 2*Vm, 'highcut': Vc + 2*Vm, 'label': 'Wide'},
        {'lowcut': Vc - 0.5*Vm, 'highcut': Vc + 0.5*Vm, 'label': 'Narrow'}
    ]

    plt.figure(figsize=(12, 12))

    # Plot original signal
    plt.subplot(4, 1, 1)
    plt.title("Original AM Signal")
    plt.plot(n, am_signal)
    plt.xlabel("Time")
    plt.ylabel("Amplitude")
    plt.grid(True)

    # Plot filtered signals with different bandwidths
    for i, bw in enumerate(bandwidths, start=2):
        filtered = bandpass_filter(am_signal, fs, bw['lowcut'], bw['highcut'])
        plt.subplot(4, 1, i)
```

```python
    # Plot filtered signals with different bandwidths
    for i, bw in enumerate(bandwidths, start=2):
        filtered = bandpass_filter(am_signal, fs, bw['lowcut'], bw['highcut'])
        plt.subplot(4, 1, i)
        plt.title(f"Bandpass Filtered Signal ({bw['label']} Bandwidth)")
        plt.plot(n, filtered)
        plt.xlabel("Time")
        plt.ylabel("Amplitude")
        plt.grid(True)

    plt.tight_layout()
    plt.show()

plot()
```