Workshop 3: Linear, Time-Invariant Systems

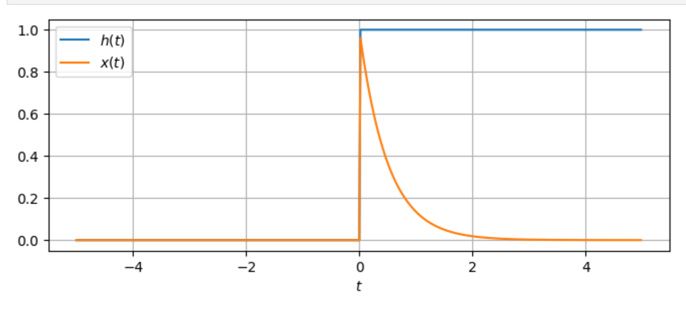
3.1 Continuous-Time Systems: Convolution Integral

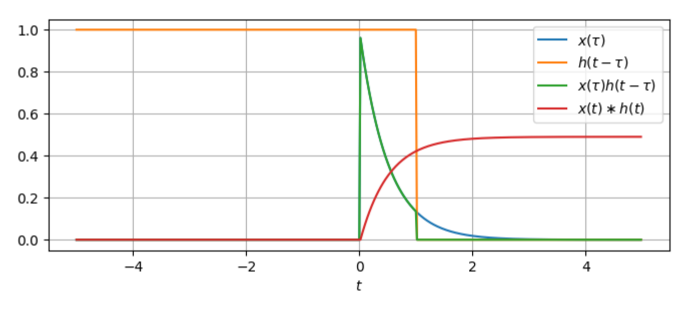
3.1.1 Implementing Convolution Using Numerical Integration

Task 1

In [35]:
```
# importing libraries
from scipy import integrate
import numpy as np
import matplotlib.pyplot as plt
```

In [36]:
```
h = lambda t: (t > 0) * 1.0
x = lambda t: (t > 0) * np.exp(-2 * t)  # a = −2
Fs = 50  # Sampling frequency for the plotting
T = 5  # Time range
t = np.arange(-T, T, 1 / Fs)  # Time samples

plt.figure(figsize=(8, 3))
plt.plot(t, h(t), label='$h(t)$')
plt.plot(t, x(t), label='$x(t)$')
plt.xlabel(r'$t$')
plt.grid()
plt.legend()

# Plotting
t_ = 1  # For illustration, choose some value for t
flipped = lambda tau: h(t_ - tau)
product = lambda tau: x(tau) * h(t_ - tau)
plt.figure(figsize=(8, 3))
plt.plot(t, x(t), label=r'$x(\tau)$')
plt.plot(t, flipped(t), label=r'$h(t − \tau)$')
plt.plot(t, product(t), label=r'$x(\tau)h(t −\tau)$')

# Computing the convolution using integration
y = np.zeros(len(t))
for n, t_ in enumerate(t):
    product = lambda tau: x(tau) * h(t_ - tau)
    y[n] = integrate.simps(product(t), t)  # Actual convolution at time t

plt.plot(t, y, label=r'$x(t)\ast h(t)$')  # Plotting the output y
plt.xlabel(r'$t$')
plt.legend()
plt.grid()
plt.show()
```





3.1.2 Convolving with a Signal Composed of Impulse Functions

Task 2

In [37]:
```
fs = 1000  # Sampling frequency for the plotting
T = 5  # Time range
t = np.arange(-T, T, 1 / fs)  # Time samples
delta = lambda t: np.array([fs / 10 if 0 < t_ and t_ < 1 / (fs / 10) else 0.0 for t_ in t])

y = integrate.simps(delta(t), t)
print(y)
```

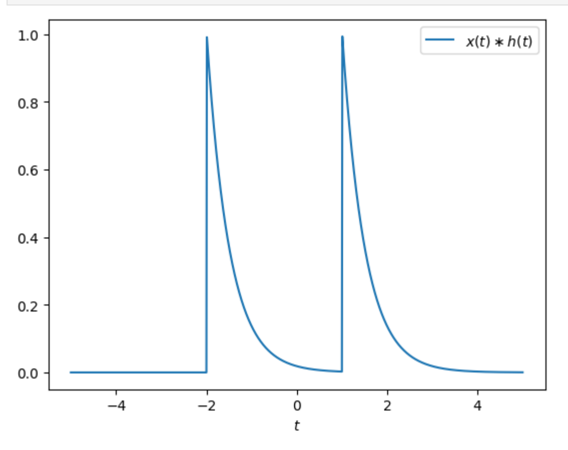
```
1.000000000000334
```

Task 3

y(t) = e^(-a(t+2)) + e^(-a(t-1))

Task 4

In [38]:
```
h = lambda t: delta(t + 2) + delta(t - 1)
x = lambda t: (t > 0) * np.exp(- 2 * t)  # a = -2
fs = 1000  # Sampling frequency for the plotting
T = 5  # Time range
t = np.arange(-T, T, 1 / fs)  # Time samples

# Computing the convolution using integration
y = np.zeros(len(t))
for n, t_ in enumerate(t):
    product = lambda tau: x(tau) * h(t_ - tau)
    y[n] = integrate.simps(product(t), t)  # Actual convolution at time t

plt.plot(t, y, label=r'$x(t)\ast h(t)$')  # Plotting the output y
plt.xlabel(r'$t$')
plt.legend()
plt.show()
```

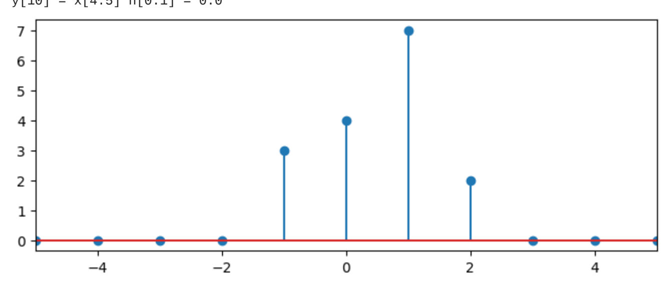


3.2 Discrete-Time Systems: Convolution Sum

Task 5

In [39]:
```
# input signal
x = np.array([0, 1, 1, 2, 0])

# unit impulse response
h = np.array([0, 0, 0, 3, 1, 0, 0])
hr = np.flip(h)
xo = 2
ho = 4

# Length of the output signal
y = np.zeros(len(x) + len(h) - 1)

for n in range(len(y)):
    xkmin = max(0, n - len(h) + 1)
    xkmax = min(len(x), n + 1)
    hkmin = max(0, len(h) - n - 1)
    hkmax = min(len(h), len(x) + len(h) - n - 1)
    y[n] = np.sum(x[xkmin:xkmax] * hr[hkmin:hkmax])
    print("y[{0}] = x[{1}:{2}]*h[{3}:{4}] = {5}".format(n, xkmin, xkmax, hkmin, hkmax, y[n]))
n = np.arange(-5, 6, 1)
fig, ax = plt.subplots(figsize=(8, 3))
ax.stem(n, y)
plt.xlim(-5, 5)
plt.show()
```

```
y[0] = x[0:1]*h[6:7] = 0.0
y[1] = x[0:2]*h[5:7] = 0.0
y[2] = x[0:3]*h[4:7] = 0.0
y[3] = x[0:4]*h[3:7] = 0.0
y[4] = x[0:5]*h[2:7] = 3.0
y[5] = x[0:5]*h[1:6] = 4.0
y[6] = x[0:5]*h[0:5] = 7.0
y[7] = x[1:5]*h[0:4] = 2.0
y[8] = x[2:5]*h[0:3] = 0.0
y[9] = x[3:5]*h[0:2] = 0.0
y[10] = x[4:5]*h[0:1] = 0.0
```

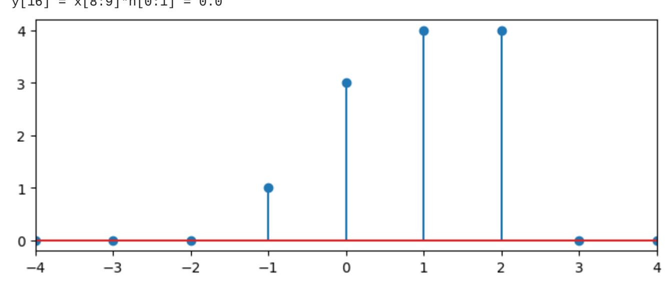


Comments: Output signal has 4 non-Zero values.
The maximum value of output signal is 7

Task 6

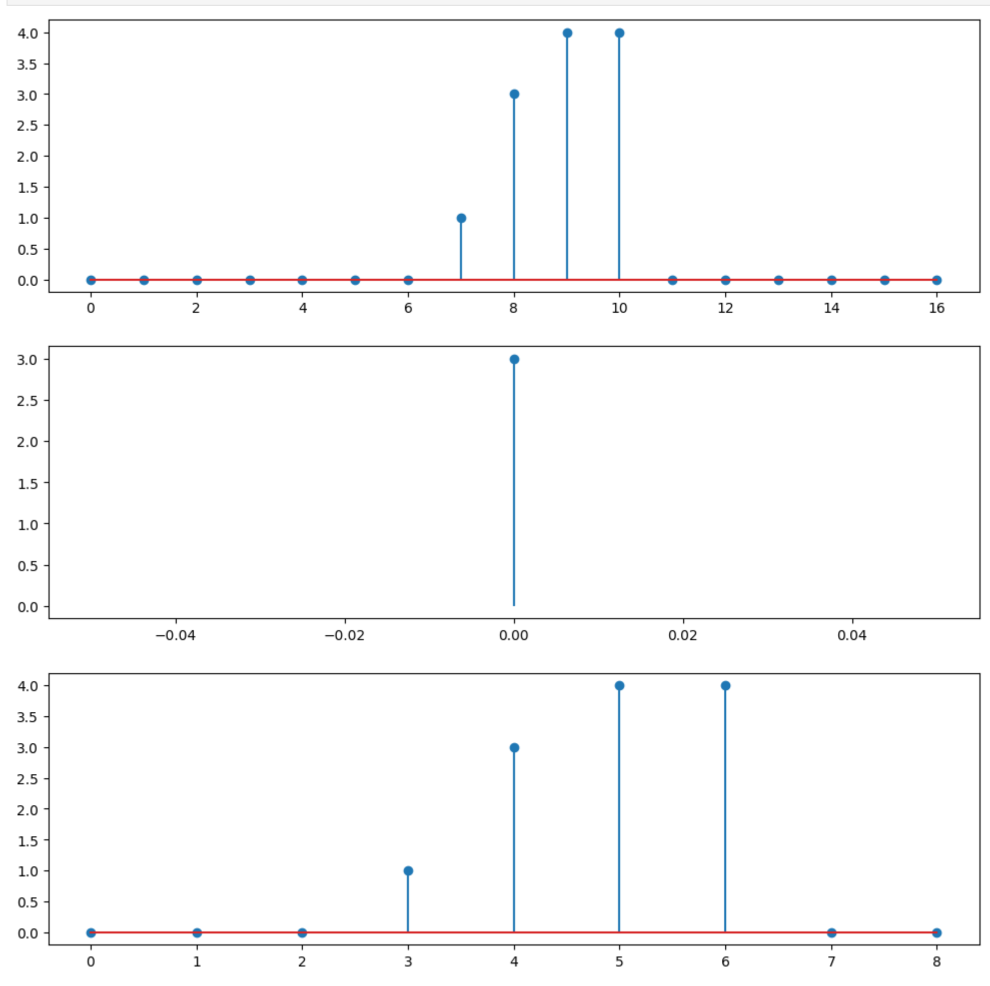
In [40]:
```
x = np.array([0, 0, 0, 1, 1, 2, 0, 0, 0])
h = np.array([0, 0, 0, 0, 1, 2, 0, 0, 0])
hr = np.flip(h)
xo = 2
ho = 4
y = np.zeros(len(x) + len(h) - 1)
for n in range(len(y)):
  xkmin = max(0, n - len(h) + 1)
  xkmax = min(len(x), n + 1)
  hkmin = max(0, len(h) - n -1)
  hkmax = min(len(h), len(x) + len(h) -n -1)
  y[n] = np.sum(x[xkmin:xkmax]*hr[hkmin:hkmax])
  print("y[{0}] = x[{1}:{2}]*h[{3}:{4}] = {5}".format(n, xkmin, xkmax, hkmin, hkmax, y[n]))

n=np.arange(-8,9,1)
fig,ax = plt.subplots(figsize=(8,3))
ax.stem(n,y)
plt.xlim(-4,4)
plt.show()
```

```
y[0] = x[0:1]*h[8:9] = 0.0
y[1] = x[0:2]*h[7:9] = 0.0
y[2] = x[0:3]*h[6:9] = 0.0
y[3] = x[0:4]*h[5:9] = 0.0
y[4] = x[0:5]*h[4:9] = 0.0
y[5] = x[0:6]*h[3:9] = 0.0
y[6] = x[0:7]*h[2:9] = 0.0
y[7] = x[0:8]*h[1:9] = 1.0
y[8] = x[0:9]*h[0:9] = 3.0
y[9] = x[1:9]*h[0:8] = 4.0
y[10] = x[2:9]*h[0:7] = 4.0
y[11] = x[3:9]*h[0:6] = 0.0
y[12] = x[4:9]*h[0:5] = 0.0
y[13] = x[5:9]*h[0:4] = 0.0
y[14] = x[6:9]*h[0:3] = 0.0
y[15] = x[7:9]*h[0:2] = 0.0
y[16] = x[8:9]*h[0:1] = 0.0
```



Task 7

In [41]:
```
from scipy import signal

fig, ax = plt.subplots(3, 1, figsize=(12, 12))

y = signal.convolve(x, h, 'full')  # Plotting the output y for full mode
ax[0].stem(y)

y = signal.convolve(x, h, 'valid')  # Plotting the output y for valid mode
ax[1].stem(y)

y = signal.convolve(x, h, 'same')  # Plotting the output y for same mode
ax[2].stem(y)

plt.show()
```

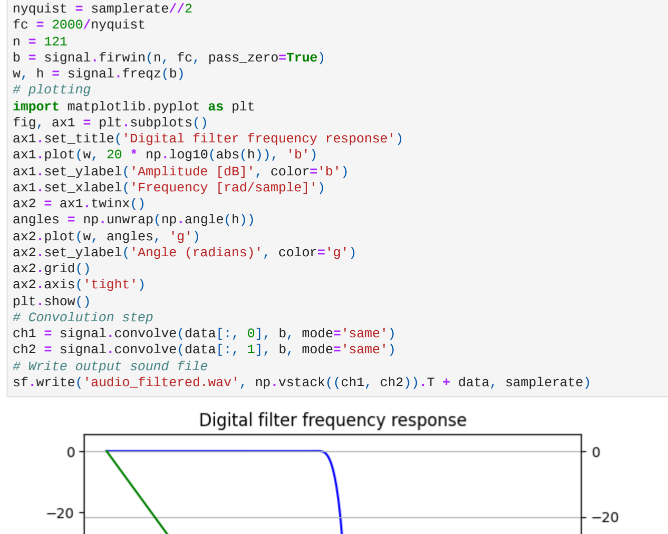


3.3 An Application in Audio Signal Filtering

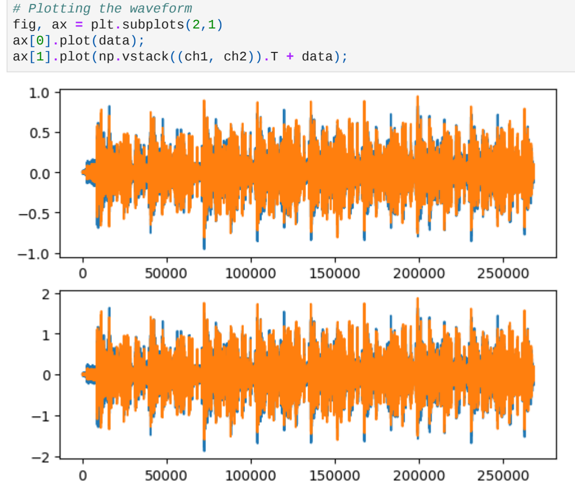
Task 8

In [42]:
```
# Importing libraries
from scipy import signal
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
```

In [43]:
```
data, samplerate = sf.read('audio_file.wav')
# Setting sampling rate
nyquist = samplerate//2
fc = 2000/nyquist
n = 121
b = signal.firwin(n, fc, pass_zero=True)
w, h = signal.freqz(b)
# plotting
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
ax1.set_title('Digital filter frequency response')
ax1.plot(w, 20 * np.log10(abs(h)), 'b')
ax1.set_ylabel('Amplitude [dB]', color='b')
ax1.set_xlabel('Frequency [rad/sample]')
ax2 = ax1.twinx()
angles = np.unwrap(np.angle(h))
ax2.plot(w, angles, 'g')
ax2.set_ylabel('Angle (radians)', color='g')
ax2.grid()
ax2.axis('tight')
plt.show()
# Convolution step
ch1 = signal.convolve(data[:, 0], b, mode='same')
ch2 = signal.convolve(data[:, 1], b, mode='same')
# Write output sound file
sf.write('audio_filtered.wav', np.vstack((ch1, ch2)).T + data, samplerate)
```



In [44]:
```
# Plotting the waveform
fig, ax = plt.subplots(2,1)
ax[0].plot(data);
ax[1].plot(np.vstack((ch1, ch2)).T + data);
```



Task 9

Lowpass filter

In [45]:
```
b, a = signal.cheby1(4, 5, 100, 'low', analog=True)

# Convolution step
ch1 = signal.convolve(data[:, 0], b, mode='same')
ch2 = signal.convolve(data[:, 1], b, mode='same')

# Create output sound file
sf.write('audio_filtered_lowpass.wav', np.vstack((ch1, ch2)).T + data, samplerate)
```

Butterworth Filter

In [46]:
```
b, a = signal.butter(4, 100, 'low', analog=True)

# Convolution step
ch1 = signal.convolve(data[:, 0], b, mode='same')
ch2 = signal.convolve(data[:, 1], b, mode='same')

# Create output sound file
sf.write('audio_filtered_butter.wav', np.vstack((ch1, ch2)).T + data, samplerate)
```

3.4 Convolution Sum in 2-D

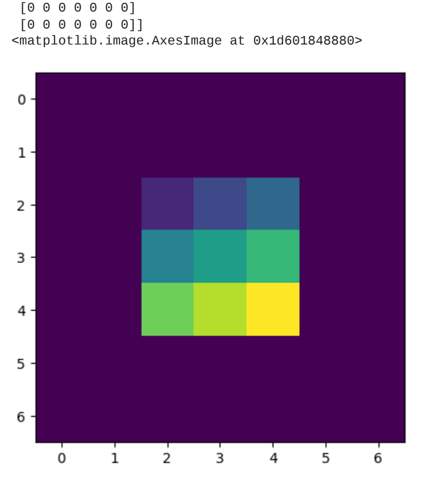
Task 10

Task 11

In [47]:
```
image = np.array([[0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0],
                  [0, 0, 1, 0, 0],
                  [0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0]])

filter = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

output = signal.convolve2d(image, filter)

# Display the output in bitpattern array
print(output)

# Display image
plt.imshow(output)
```

```
[[0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 1 2 3 0 0]
 [0 0 4 5 6 0 0]
 [0 0 7 8 9 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]]
```

Out[47]:
```
<matplotlib.image.AxesImage at 0x1d601848880>
```



3.5 Application: Using Convolution to Filter an Image

Task 12

In [48]:
```
import matplotlib.image as mpimg

x = mpimg.imread('allenkeys.png')
fig, ax = plt.subplots(1, 2)
ax[0].imshow(x, cmap='gray')

filter = np.array([[-1, 0, 1],
                   [-2, 0, 2],
                   [-1, 0, 1]])

filtered = signal.convolve2d(x, filter)
ax[1].imshow(filtered, cmap='gray')

plt.show()
```