In [1]:
```
import numpy as np
import scipy
import scipy.signal
import matplotlib.pyplot as plt
```

In [2]:
```
def ampl_res(h):
    M = len(h)
    L = int(np.floor(M / 2))
    if not np.array_equal(np.fix(np.abs(h[0:L] * (10**10))), np.fix(np.abs(h[M
- L:M]) * (10**10))[::-1]):
        print("Error: This is not a linear-phase impulse reponse.")
    if 2 * L != M:
        if h[0] == h[M-1]:
            print("*** Type-1 Linear-Phase Filter ***")
        elif h[0] != h[M-1]:
            print("*** Type-3 Linear-Phase Filter ***")
    else:
        if h[0] == h[M-1]:
            print("*** Type-2 Linear-Phase Filter ***")
        elif h[0] != h[M-1]:
            print("*** Type-4 Linear-Phase Filter ***")

# Test script
n = np.arange(0, 11)
h_I = (0.9)**np.abs(n - 5) * np.cos(np.pi * (n - 5) / 12)
ampl_res(h_I)

n = np.arange(0, 10)
h_II = (0.9)**np.abs(n - 4.5) * np.cos(np.pi * (n - 4.5) / 11)
ampl_res(h_II)

n = np.arange(0, 11)
h_III = (0.9)**np.abs(n - 5) * np.sin(np.pi * (n - 5) / 12)
ampl_res(h_III)

n = np.arange(0, 10)
h_IV = (0.9)**np.abs(n - 4.5) * np.sin(np.pi * (n - 4.5) / 11)
ampl_res(h_IV)
```

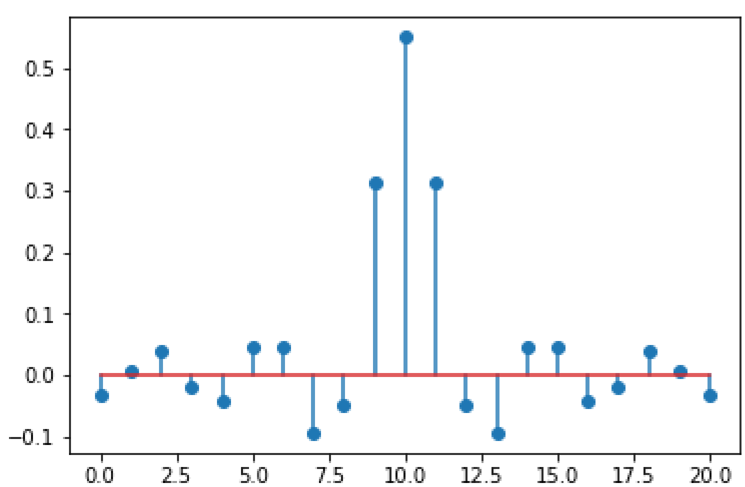
```
*** Type-1 Linear-Phase Filter ***
*** Type-2 Linear-Phase Filter ***
*** Type-3 Linear-Phase Filter ***
*** Type-4 Linear-Phase Filter ***
```

In [3]:
```
def db2delta(Rp, As):
    # Converts dB specs Rp and As into absolute specs delta1 and delta2
    # delta1 = Passband tolerance
    # delta2 = Stopband tolerance
    # Rp = Passband ripple
    # As = Stopband attenuation

    delta1 = (((10)**(Rp / 20)) - 1) / (((10)**(Rp / 20)) + 1)
    delta2 = (1 + delta1) * (10**(-As / 20))

    # You need to fill in here
    return delta1, delta2
```

In [4]:
```
def delta2db(delta1, delta2):
    # Converts absolute specs delta1 and delta2 into dB specs Rp and As
    # Rp = Passband ripple
    # As = Stopband attenuation
    # delta1 = Passband tolerance
    # delta2 = Stopband tolerance

    Rp = -20 * np.log10((1 - delta1) / (1 + delta1))
    As = -20 * np.log10(delta2 / (1 + delta1))

    # You need to fill in here
    return Rp, As
```

In [6]:
```
# Test script
Rp, As = delta2db(0.01, 0.001)  # Rp: 0.1737, As: 60.0864
print(Rp, As)
delta1, delta2 = db2delta(0.25, 50)  # delta1: 0.0144, delta2: 0.0032
print(delta1, delta2)



'''HW3_3_2 code format, you need to fill in ...'''



def ideal_lp(wc, M):
    # Ideal LowPass filter computation
    # hd = ideal impulse response between 0 to M-1
    # wc = cutoff frequency in radians
    # M = length of the ideal filter

    # You need to fill in here
    n = np.arange(0, M)

    alpha = int((M - 1) / 2)

    m = n - alpha + np.spacing(1)

    hd = np.sin(wc * m) / (np.pi * m)
    return hd

# Test script
hd = ideal_lp(0.55 * np.pi, 21)
n = np.arange(21)

plt.stem(n, hd)
plt.show()
```

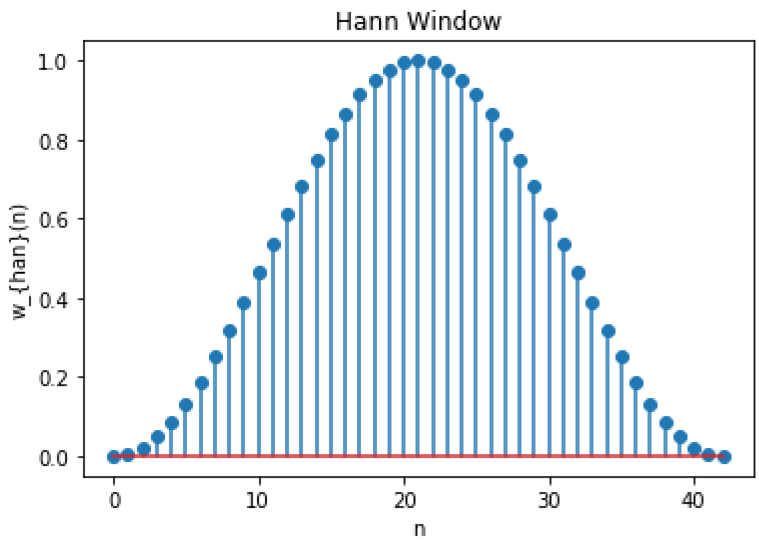
```
0.17372358370185334 60.086427475652854
0.014390163418102826 0.003207783352471618
```

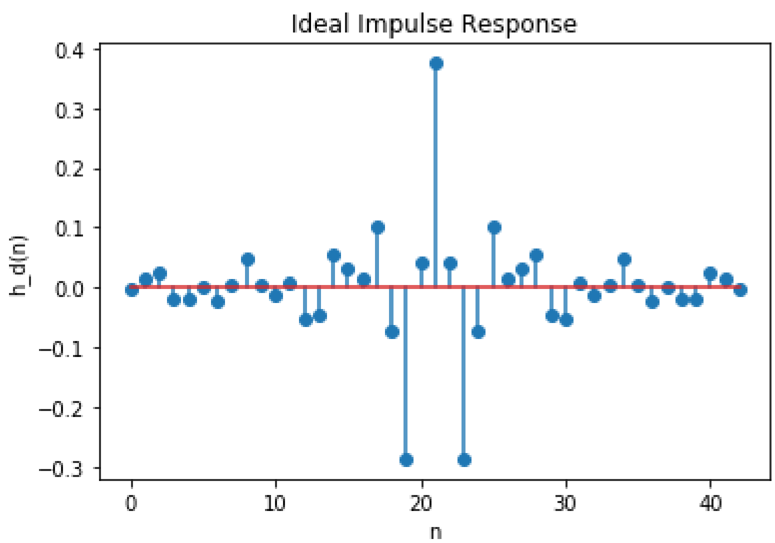


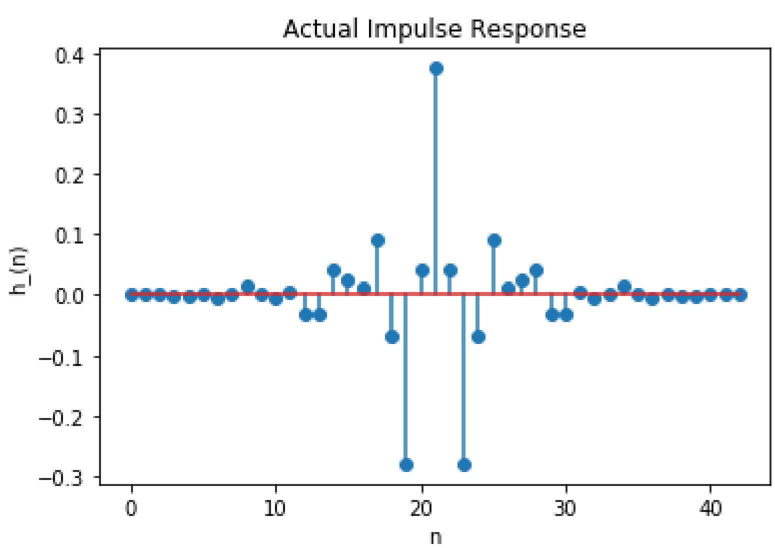
In [7]:
```
def freqz_m(b, a):
    # Modified version of scipy.signal.freqz()
    # db    = Relative magnitude in dB computed over 0 to pi radians
    # mag = absolute magnitude computed over 0 to pi radians
    # pha   = Phase response in radians over 0 to pi radians
    # W     = 501 frequency samples between 0 to pi radians
    # H     = The frequency response, as complex numbers.
    # b     = numerator polynomial of H(z) (for FIR: b=h)
    # a     = denominator polynomial of H(z) (for FIR: a=[1])

    W, H = scipy.signal.freqz(b, a, 1000, 'whole')
    # print(H)
    W = W[0:501]
    H = H[0:501]
    mag = np.abs(H)
    db = 20 * np.log10((mag) / np.max(mag))
    pha = np.angle(H)
    return db, mag, pha


"""
Step 1: Define some specifications
"""

# Specification
ws1 = 0.2 * np.pi
wp1 = 0.35 * np.pi
wp2 = 0.55 * np.pi
ws2 = 0.75 * np.pi
Rp  = 0.25
As  = 40

"""
Step 2: Preprocess and get the signal length
"""

# Select the min(delta1, delta2) since delta1=delta2 in window design
delta1, delta2 = db2delta(Rp, As)
if (delta1 < delta2):
    delta2 = delta1
    print('Delta1 is smaller than delta2')
    Rp, As = delta2db(delta1, delta2)

#
tr_width = min((wp1 - ws1), (ws2 - wp2))

M = np.ceil(6.2 * np.pi / tr_width)
M = 2 * np.floor(M / 2) + 1
print(M)

"""
Step 3: Apply Hanning Window. Plot 1: Hann Window.
"""

n = np.arange(M)
w_han = np.hanning(M)
```

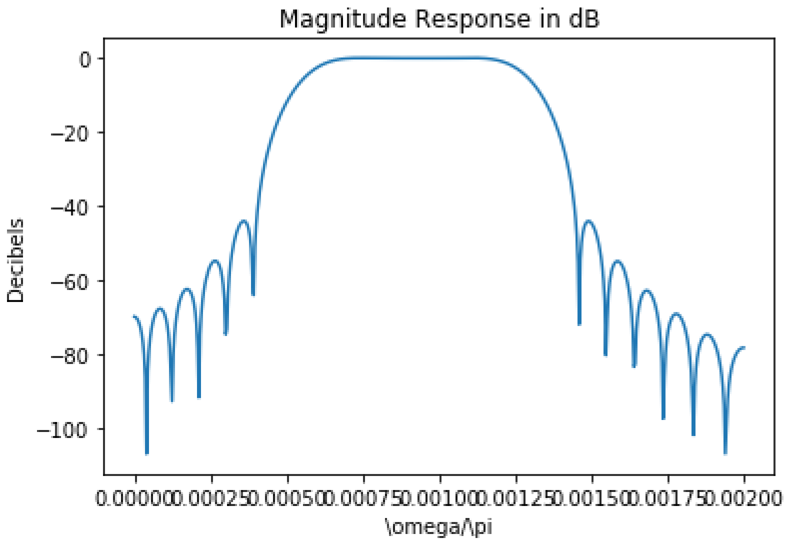
```
plt.stem(n, w_han)
plt.xlabel('n')
plt.ylabel('w_{han}(n)')
plt.title('Hann Window')
plt.show()

"""
Step 4: Use low pass filter to create a band-pass filter. Plot 2: Ideal Impuls
e Response.
"""
wc1 = (ws1 + wp1) / 2
wc2 = (ws2 + wp2) / 2

hd = ideal_lp(wc2, M) - ideal_lp(wc1, M)

plt.stem(n, hd)
plt.xlabel('n')
plt.ylabel('h_d(n)')
plt.title('Ideal Impulse Response')
plt.show()

"""
Step 5: Plot 3: Actual Impulse Response
"""

h = np.multiply(hd, w_han)
plt.stem(n, h)
plt.xlabel('n')
plt.ylabel('h_(n)')
plt.title('Actual Impulse Response')
plt.show()

"""
Use the function freqz_m in freqz_m.py to compute the magnitude. Plot 4: Magni
tude Response in dB
"""
db, mag, pha = freqz_m(h, 1)
delta_w = np.pi / 500
w = np.linspace(0, delta_w, 501)

plt.plot(w / np.pi, db)
plt.title('Magnitude Response in dB')
plt.xlabel('\omega/\pi')
plt.ylabel('Decibels')
plt.show()
```

```
43.0
```


Hann Window
w_{han}(n)
n


Ideal Impulse Response
h_d(n)
n


Actual Impulse Response
h_(n)
n


Magnitude Response in dB
Decibels
0
−20
−40
−60
−80
−100
0.000000.000250.000500.000750.001000.001250.001500.001750.00200
\omega/\pi

In [ ]: