Programming Assignment 4

Q1. Z transform

In [1]:
```
import math
import numpy as np
import collections
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

$$X(z) = \sum_{n=-\infty}^{\infty} a^n u(n) z^{-n} = \sum_{n=0}^{\infty} a^n z^{-n} = \sum_{n=0}^{\infty} (az^{-1})^n$$

To find the region of convergence, we need $|a/z| < 1$. Let us represent z in polar coordinates such that the radial part of z is given by r. hance, we effectively need $a/r < 1$ (if a is real).

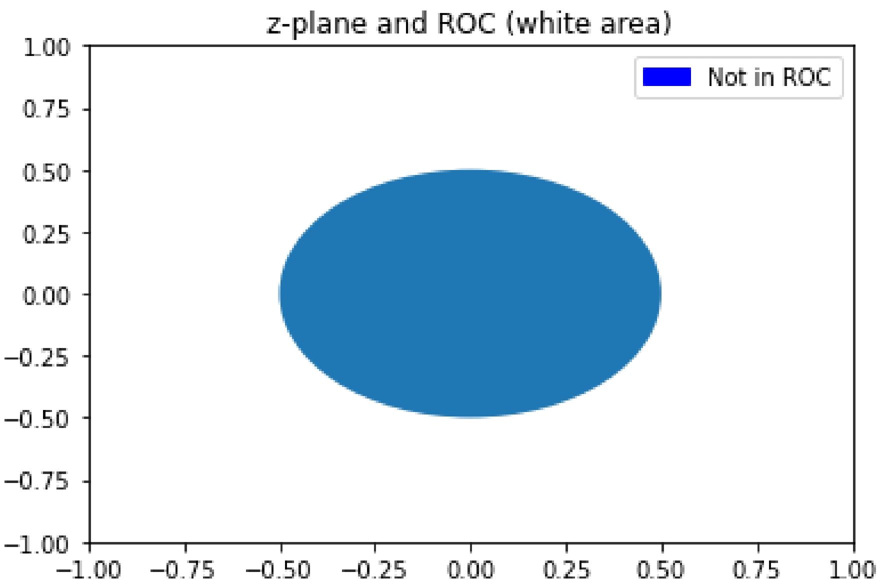
we take equally spaced values from 0 to 10 with spacing of 0.01 to find the limiting r where $a/r = 1$ and then check if the ratio is less than 1 for vslues of r greater than the limiting value or lesser than the limiting value and print the result.

In [2]:
```
r = np.linspace(0,10,1001)
base = 1/2
roc = []
limit = 0
for ri in r:
    if ri == 0:
        continue
    if base/ri < 1:
        roc.append(ri)
    if base/ri == 1:
        limit = ri
if roc[1]>roc[0]:
    print("ROC is the area outside the circle of radius :",limit )
else:
    print("ROC is the area inside the circle of radius :",limit )
```

```
ROC is the area outside the circle of radius : 0.5
```

Now that we know the ROC, we can now plot the same

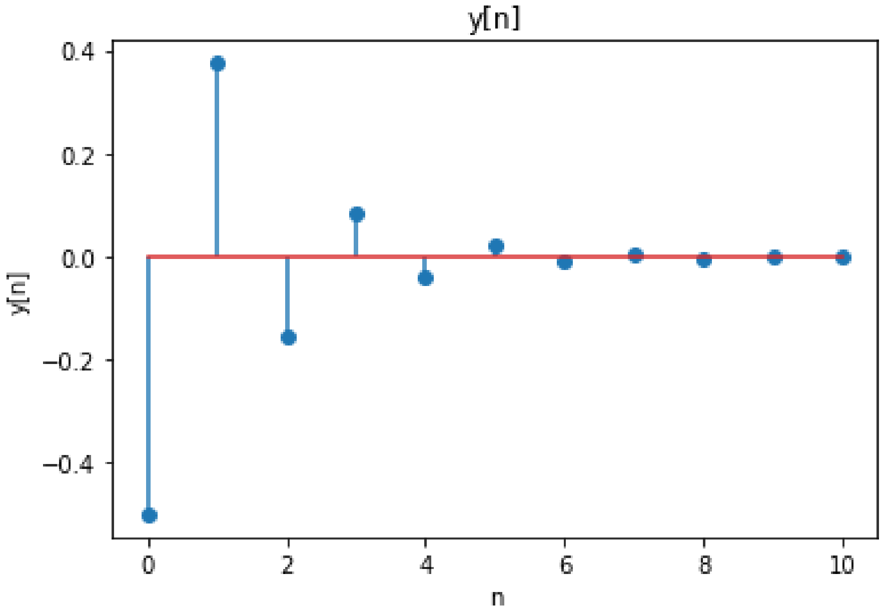
In [3]:
```
fig, ax = plt.subplots()
ax.set(xlim=(-1, 1), ylim = (-1, 1))
a_circle = plt.Circle((0, 0), .5,label='Not in ROC')
ax.add_artist(a_circle)
patches = [mpatches.Patch(color="blue", label="Not in ROC")]
plt.legend(handles=patches)
plt.title("z-plane and ROC (white area)")
plt.show()
```



We now have the recursive relation : $y[n-1]+2y[n]=x[n]$ and the input signal : $x[n]=(\frac{1}{4})^n u[n]$. We also have the initial condition y[-1] = 2

In [4]:

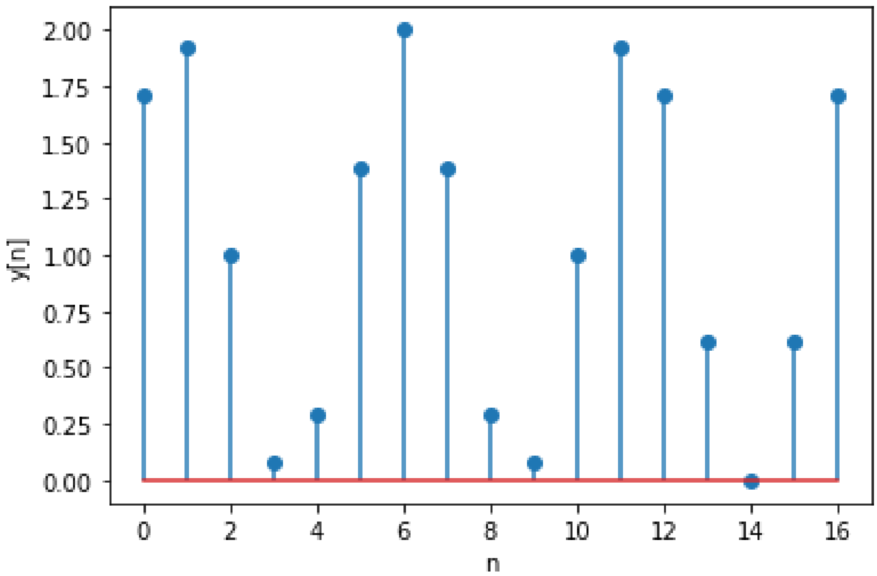
```
x=np.arange(0,11)
def unitstep(x):
    if x>=0:
        return 1
    else:
        return 0
def prev(x,y):
    if x<0:
        return 2
    else:
        return y[n-1]
y=[]
for n in x:
    y.append((math.pow(1/4,n)*unitstep(n)-prev(n-1,y))/2)
plt.stem(x,y)
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('y[n]')
plt.show()
```



Q2. Filtering

The input signal given is:

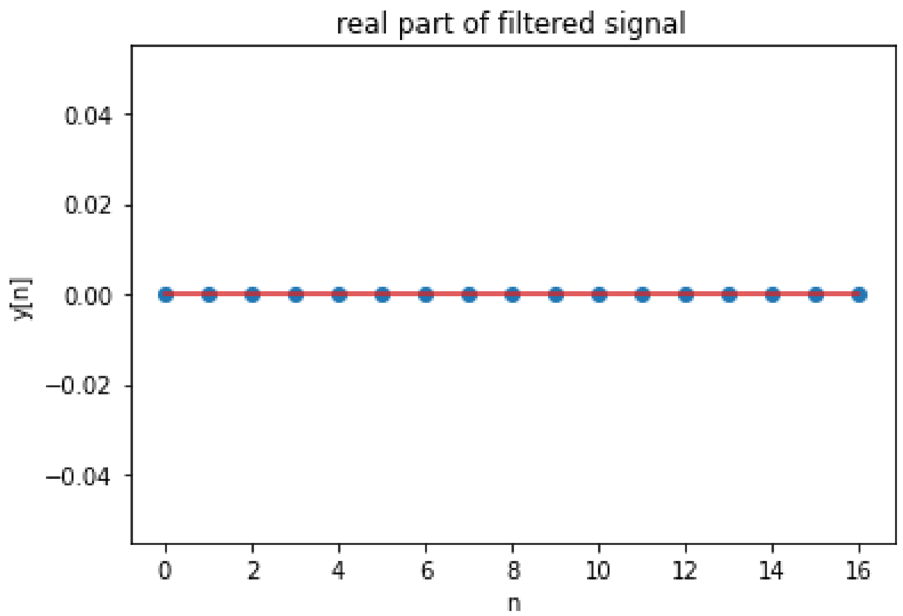
In [5]:
```
import math
import numpy as np
import collections
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

x=np.arange(0,17)
y=[]
for n in x:
    y.append(1+np.sin((3*np.pi*n)/8+np.pi/4))
plt.stem(x,y)
plt.xlabel("n")
plt.ylabel("y[n]")
plt.show()
```

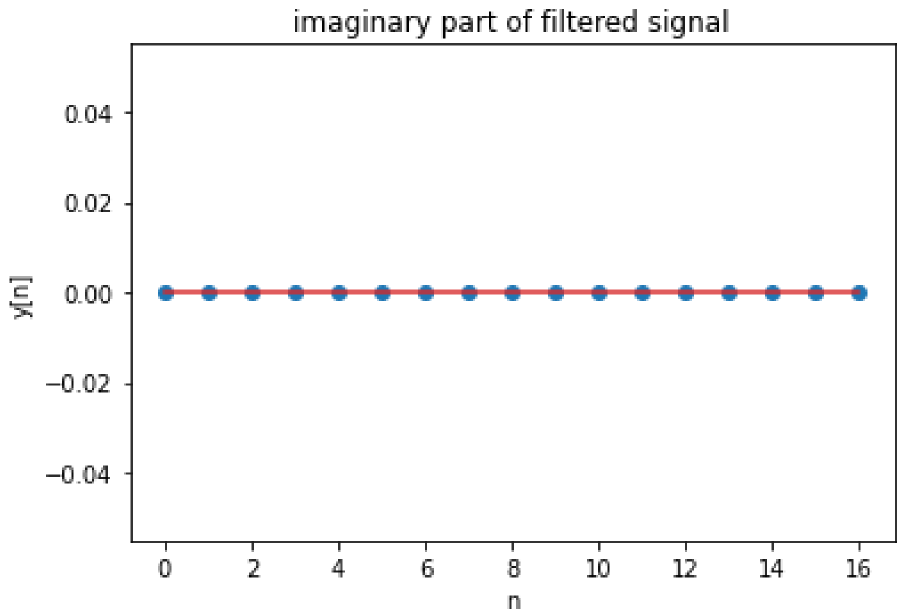


$$y[n] = \sum_{k=\langle N\rangle} a_k H(e^{j2\pi k/N})e^{jk(2\pi/N)n}.$$

In [6]:

```
N=16
ak=[]
for k in x:
    s=0
    for n in range(0,N+1):
        s=s+(y[n]*np.exp(-1j*k*n*2*np.pi/N))
    ak.append(s/N)

def fil(x):
    if x<=-np.pi/3 and x>=-5*np.pi/12:
        return 1
    elif x<=5*np.pi/12 and x>=np.pi/3:
        return 1
    elif x<=-19*np.pi/12 and x>=-5*np.pi/3:
        return 1
    elif x<=5*np.pi/3 and x>=19*np.pi/12:
        return 1
    else:
        return 0

yf=[]
for n in x:
    s=0
    for k in range(0,N+1):
        s=s+ak[k]*fil(np.exp(2*np.pi*k/N))*np.exp(1j*k*2*np.pi*n/N)
    yf.append(s)


yfr=[]
yfimg=[]
for i in yf:
    yfr.append(i.real)
    yfimg.append(i.imag)

plt.stem(x,yfr)
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('real part of filtered signal')
plt.show()

plt.stem(x,yfimg)
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('imaginary part of filtered signal')
plt.show()
```



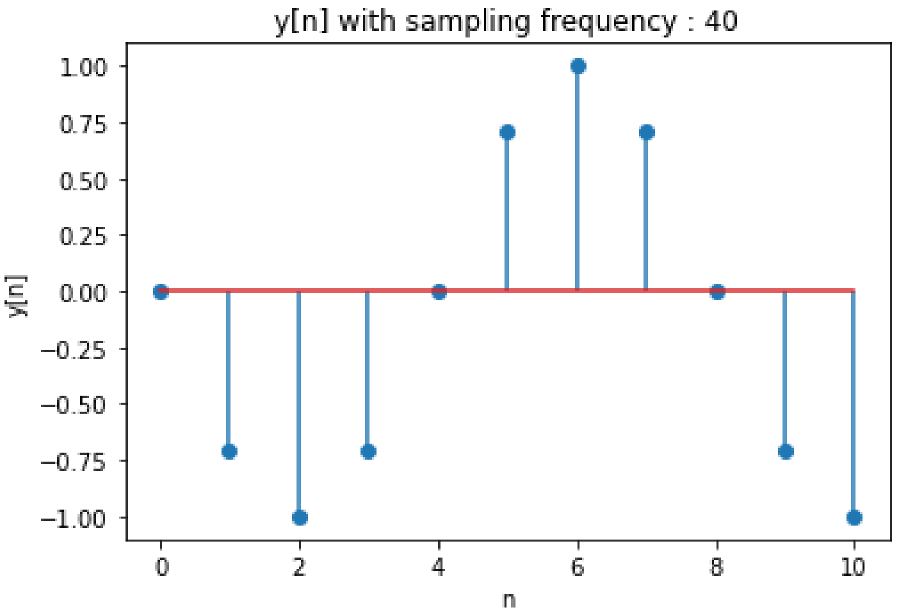


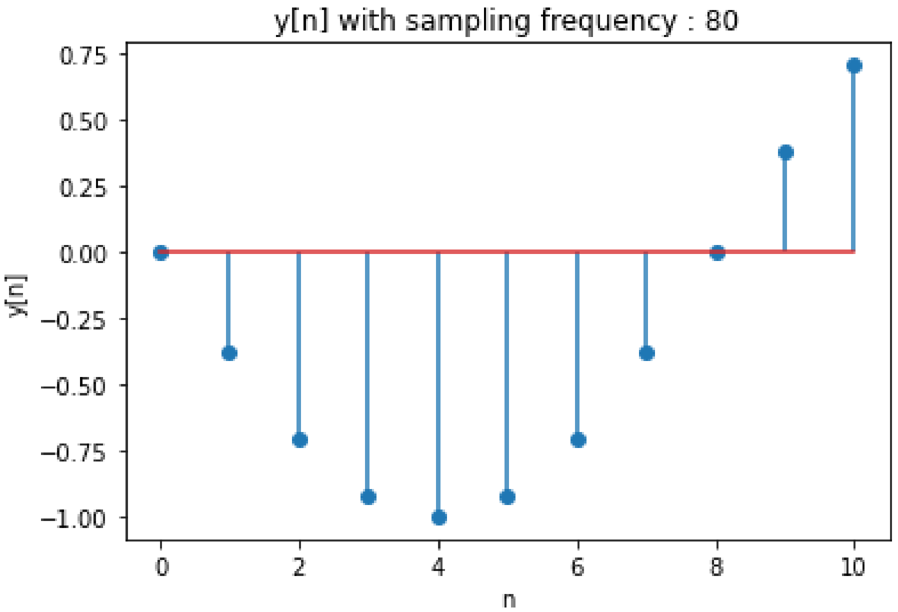
Q3. Sampling

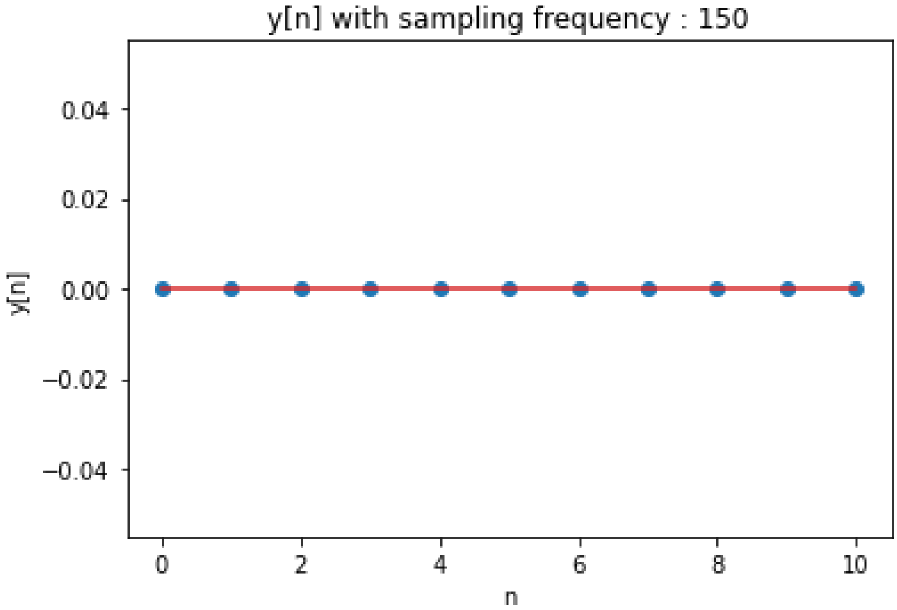
In [7]:

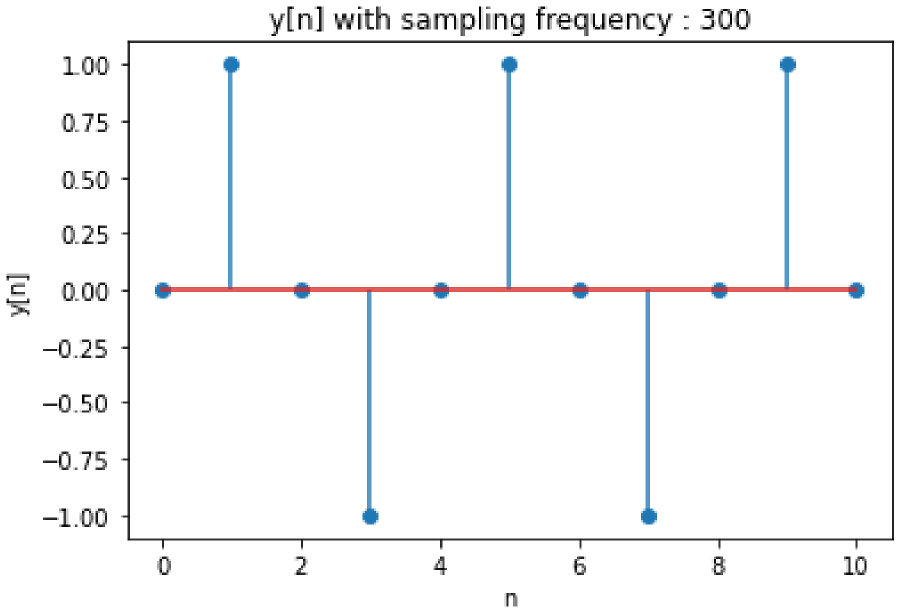
```
import math
import numpy as np
import collections
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

x=np.arange(0,11)
fa=75
fs=[40,80,150,300,600]
for i in fs:
    y=np.sin((2*np.pi*fa*x)/i)
    yC = []
    for Y in y:
        if Y > -1/math.pow(10,14) and Y < 1/math.pow(10,14):
            yC.append(0)
        else:
            yC.append(Y)
    y = yC
```

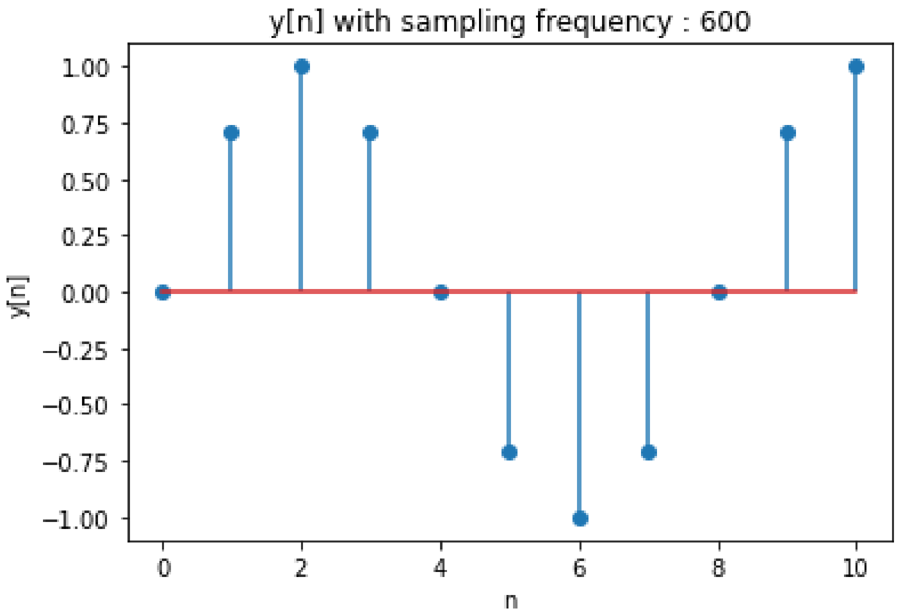
```
    plt.stem(x,y)
    plt.xlabel('n')
    plt.ylabel('y[n]')
    plt.title('y[n] with sampling frequency : '+str(i))
    plt.show()
```









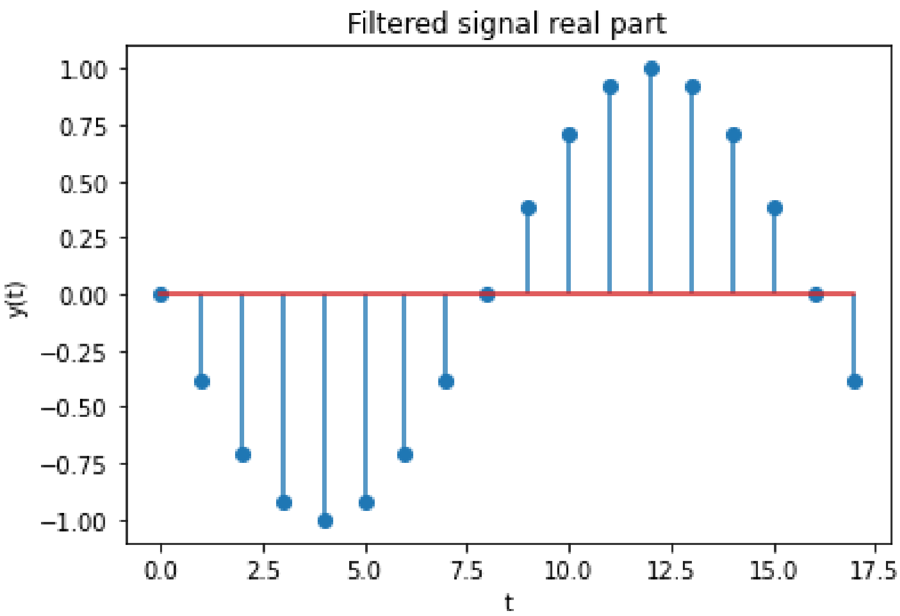


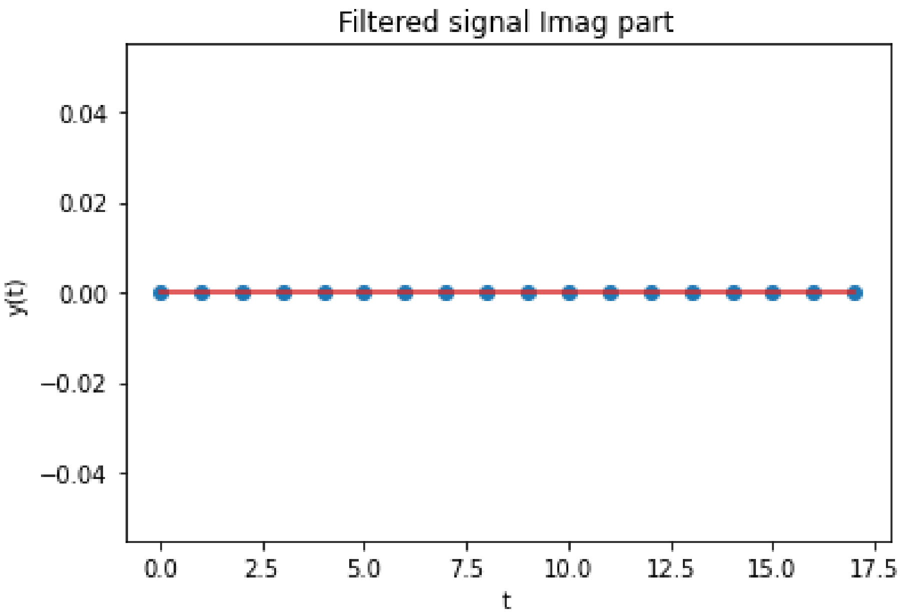
An ideal LPF with cut-off fc = 100 Hz

$$H(j\omega) = \begin{cases} 1, & |\omega| \leq \omega_c \\ 0, & |\omega| > \omega_c \end{cases} ,$$

In [8]:

```
x=np.arange(0,18)
y=np.sin(150*np.pi*x/80)
N=16
ak=[]
for k in x:
    s=0
    for n in range(0,N+1):
        s=s+(y[n]*np.exp(-1j*k*n*2*np.pi/N))
    ak.append(s/N)

def lpfil(x):
    if x<=200*np.pi and x>=-200*np.pi:
        return 1
    else:
        return 0

yf=[]
for n in x:
    s=0
    for k in range(0,N+1):
        s=s+(ak[k]*lpfil(np.exp(2*np.pi*k/N))*np.exp(1j*k*2*np.pi*n/N))
    yf.append(s)


yfR = []
yfI = []

for j in yf:
    yfR.append(j.real)
#     yfI.append(j.imag)
    if j.imag > -1/math.pow(10,14) and j.imag < 1/math.pow(10,14):
            yfI.append(0)
    else:
            yfI.append(j.imag)

plt.stem(x,yfR)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Filtered signal real part')
plt.show()
plt.stem(x,yfI)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Filtered signal Imag part')
plt.show()
```





An ideal BPF with PB between 60Hz and 80Hz.

In [9]:

```
#BPF
def bpfil(x):
    if x<=160*np.pi and x>=120*np.pi:
        return 1
    elif x>=-160*np.pi and x<=-120*np.pi:
        return 1
    else:
        return 0

yf=[]
for n in x:
    s=0
    for k in range(0,N+1):
        s=s+(ak[k]*bpfil(np.exp(2*np.pi*k/N))*np.exp(1j*k*2*np.pi*n/N))
    yf.append(s)

yfR = []
yfI = []
```

```
for j in yf:
    yfR.append(j.real)

    yfI.append(j.imag)
plt.stem(x,yfR)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Filtered signal real part')
plt.show()
plt.stem(x,yfI)
plt.xlabel('t')
plt.ylabel('y(t)')
plt.title('Filtered signal Imag part')
plt.show()
```