Programming Assignment 3

1.

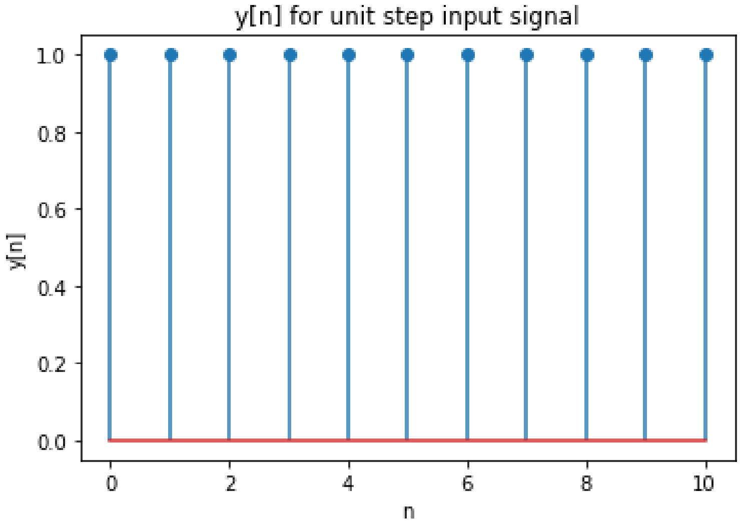
The recursive form of the given signal is:

$$(n+1)y[n] - ny[n-1] = x[n]$$

Using the recursive form of the signal, we find the signal y[n] in the following snippet. We use the input signal as a unit step signal and assume causality and LTI properties for the system.

In [100…

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

#1
x=np.arange(0,11)
def unitstep(x):
    if x>=0:
        return 1
    else:
        return 0

def prev(x,y):
    if x<0:
        return 0
    else:
        return y[n-1]

def delta(x):
    if x==0:
        return 1
    else:
        return 0

#the corresponding recursive equation is (n+1)y[n]-ny[n-1]=x[n]
#assuming x[n] to be unitstep function and the system to be causal and LTI
#which means output is zero when input is zero
y=[]
for n in x:
    if n>=0:
        y.append((unitstep(n)+n*prev(n-1,y))/(n+1))
    else:
        y.append(0)
plt.stem(x,y)
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('y[n] for unit step input signal')
plt.show()
```



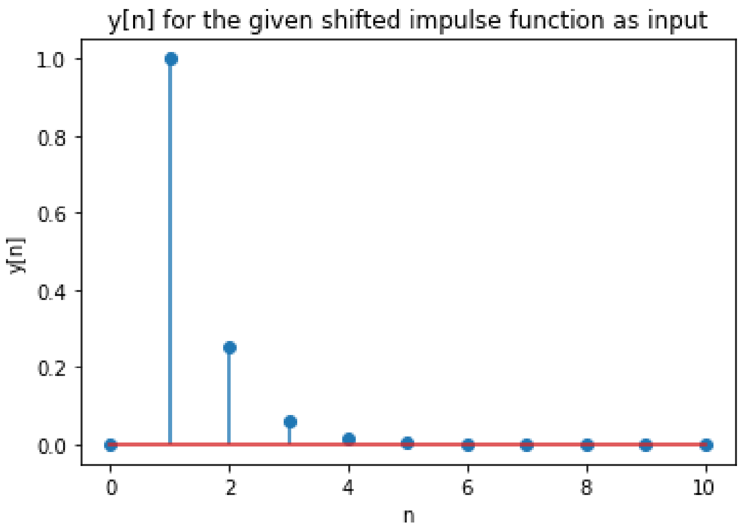
we now use the given recursion function:

$$y[n] = \frac{1}{4}y[n-1] + x[n]$$

The input signal is $x[n] = \delta(n-1)$

In [101…

```
y=[]
for n in x:
    y.append(1/4*prev(n-1,y)+delta(n-1))
plt.stem(x,y)
plt.xlabel('n')
plt.ylabel('y[n]')
plt.title('y[n] for the given shifted impulse function as input')
plt.show()
```



2.

The given signal is:

$$x[n] = \sum_{m=-\infty}^{\infty} \{4\delta[n-4m] + 8\delta[n-1-4m]\}$$

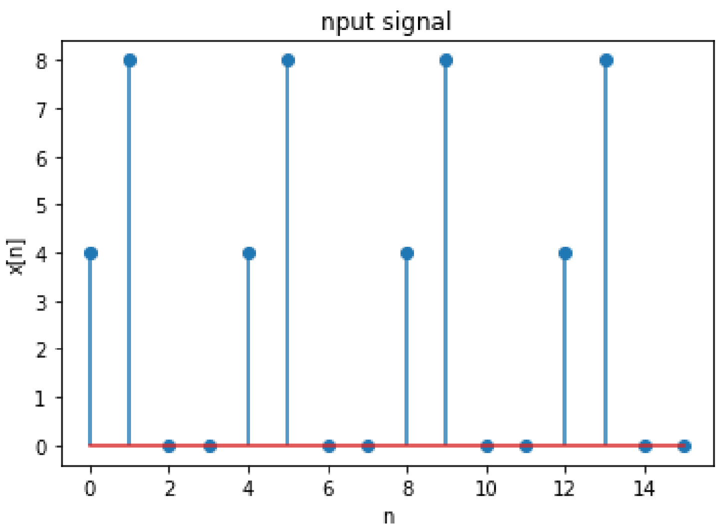
For a given n, we have : $\delta[n-4m] = 0$ unless n is a multiple of 4. Similarily we have: $\delta[n-4m-1] = 0$ unless n leaves a remainder 1 when divided by 4.

Hence, n is only non-zero when n is a multiple of 4 or it leaves 1 when divided by 4. Moreover, it is clear that when n is a multiple of 4, value of x[n] = 4 and when n leaves a remainder 1 upon division by 4, x[n] takes the value 8.

We now use this result to construct the input signal below:

In [102…

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

x=np.arange(0,16)
def delta(x):
    if x.is_integer():
        return 1
    else:
        return 0
def dataY(n):
    return 4*delta(n/4)+8*delta((n-1)/4)
y=[]

for n in x:
    y.append(dataY(n))

plt.stem(x,y)
plt.xlabel('n')
plt.ylabel('x[n]')
plt.title('nput signal')
plt.show()
#from the plot it can be seen clearly that x[n] is periodic
#values oscillate as 4,8,0,0 and this pattern goes on repeating
print('periodic with period = '+str(4))
N=4#fundamental period
```




```
periodic with period = 4
```

As seen above, the signal is periodic with a fundamental period equal to 4 ie, $P_0 = 4$

Evaluating the fourier series coeffecients for the signal:

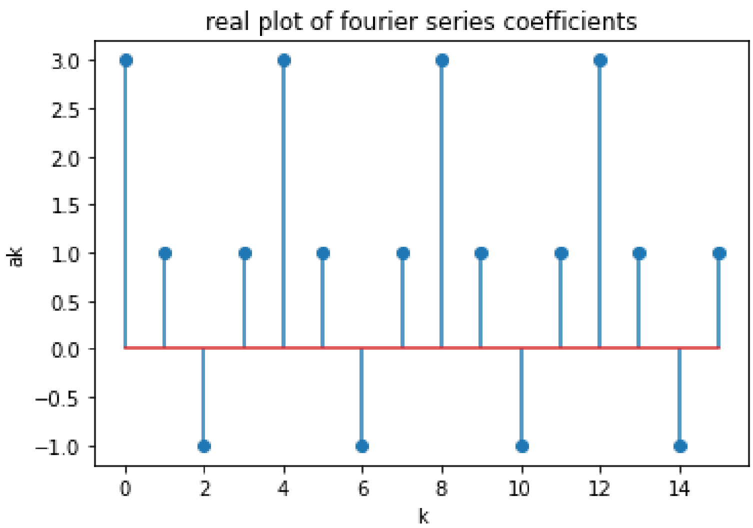
We shall plot the coeffecients of the following DTFS representation of the given signal

$$x[n] = \sum_{k=-\infty}^{\infty} \{a_k e^{jnk2\pi/N}\}$$

Here, $a_k$can be evaluated by using the relation: \begin{equation*}a_k= 1/N \sum_{n=}^{}\{x[n]e^{-jnk2\pi/N}\}\end{equation*} The following code uses the above relation to evaluate the coeffecients and plot them with respect to k.

In [103…

```
ak=[]
def akGenerator(n,k):
    return (dataY(n)*np.exp(-1j*k*n*np.pi/2))
for k in x:
    s=0
    for n in range(0,N):
        s=s+akGenerator(n,k)
    ak.append(s/4)
akr=[]
akimg=[]
for i in ak:
    akr.append(i.real)
    akimg.append(i.imag)
plt.stem(x,akr)
plt.xlabel('k')
plt.ylabel('ak')
plt.title('real plot of fourier series coefficients')
plt.show()

plt.stem(x,akimg)
plt.xlabel('k')
plt.ylabel('ak')
plt.title('imaginary plot of fourier series coefficients')
plt.show()
```



Properties of DTFS

We shall now verify the following properties of DTFS for the given signal
Our strategy to verify it is to get the coefficients of the DTFS in 2 ways -

- By doing an operation on the signal in the time domain - Case 1
- By doing the corresponding operation on the coefficients - Case 2

Time shift property

Here, we shall shift the signal by n0 = 2 and perform the the following two operations:

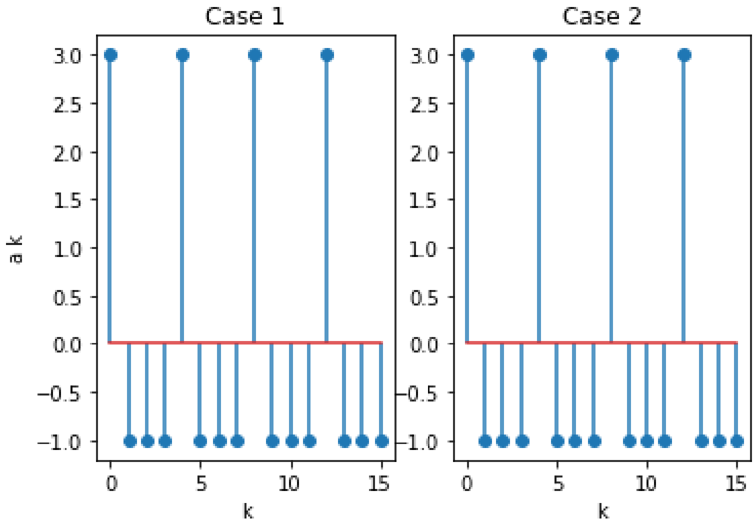
$x[n - n_0]$ $\qquad\qquad$ $a_k e^{-jk(2\pi/N)n_0}$

In [104…

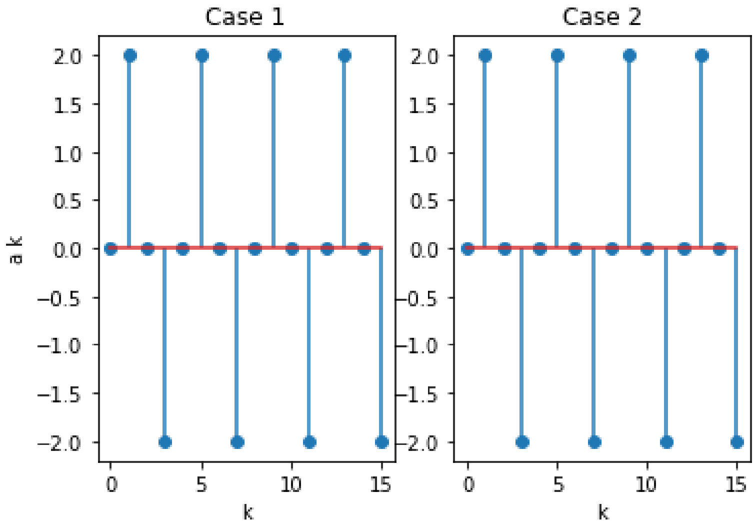
```
def dataYn(n):
    return 4*delta((n-no)/4)+8*delta((n-no-1)/4)

def aknGenerator(n):
    return (yn[n]*np.exp(-1j*k*n*np.pi/2))

no=2
yn=[]
for n in x:
    yn.append(dataYn(n))
akn=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(aknGenerator(n))
    akn.append(s/4)
temp=[]
for k in x:
    temp.append(ak[k]*np.exp(-1j*k*np.pi*no/2))

aknR = []
aknI = []
tempR = []
tempI = []
for i in range(len(akn)):
    aknR.append(akn[i].real)
    aknI.append(akn[i].imag)
    tempI.append(temp[i].imag)
    tempR.append(temp[i].real)
print("Verifying time shifting property")
print("Real part of coefficients")
plt.subplot(1,2,1)
plt.stem(x,aknR)
plt.xlabel('k')
plt.ylabel('a k')
plt.title("Case 1")
plt.subplot(1,2,2)
plt.stem(x,tempR)
plt.xlabel('k')
plt.title("Case 2")
plt.show()
#both plots are same so timeshifting property verified
print("Imaginary part of coefficients")
plt.subplot(1,2,1)
plt.stem(x,aknI)
plt.xlabel('k')
plt.title("Case 1")

plt.ylabel('a k')
```

```
plt.subplot(1,2,2)
plt.stem(x,tempI)
plt.title("Case 2")

plt.xlabel('k')
plt.show()
```

```
Verifying time shifting property
Real part of coefficients
```




```
Imaginary part of coefficients
```



The corresponding plots obtained in both cases are identical. Hence we find that Time Shifting property is verified. Now we shall use the time shifted signal and the original signal to check for linearity of the same

Linearity:

To show Linearity, we take the linear combination of two signals and take the DTFS coefficients of the final signal and compare it with the same linear combination of the coefficients of DTFS of the two initial signals. We plot the resultant signals in either case and make inferences from the established plots.

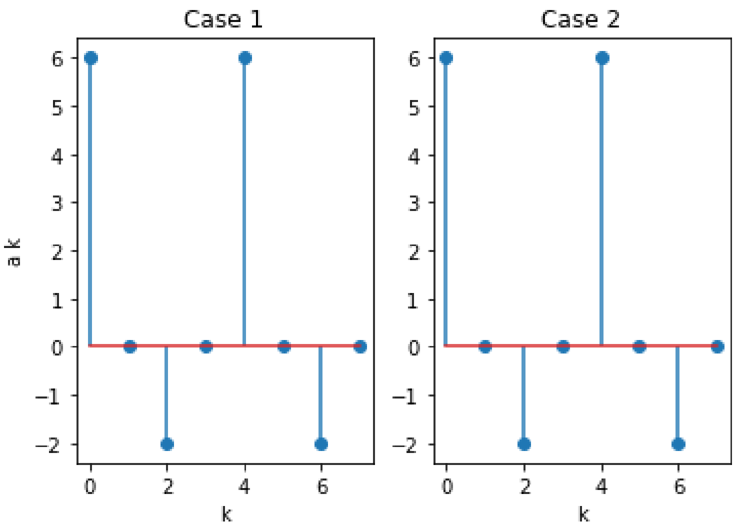
$$Ax[n] + By[n] \qquad\qquad Aa_k + Bb_k$$

In [132...

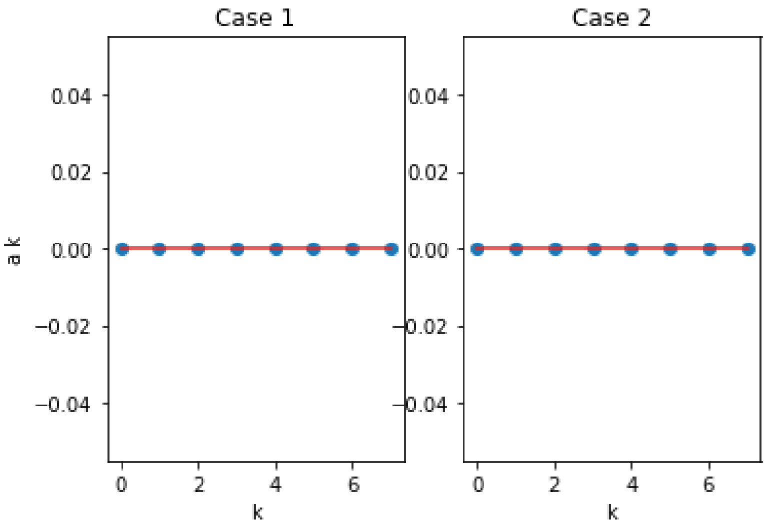
```
#properties of DTFS
#time shifting

#linearity property
#x[n] and x[n-no] are input signals
a=b=1
```

```
bk=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+((yn[n]+y[n])*np.exp(-1j*k*n*np.pi/2))
    bk.append(s/4)

temp=[ak[i]+akn[i] for i in x]
bkR = []
bkI = []
tempR = []
tempI = []
for i in range(len(akn)):
    bkR.append(bk[i].real)
    if bk[i].imag<1/math.pow(10,13) and bk[i].imag> -1 / math.pow(10,13):
        bkI.append(0)
    else:
        bkI.append(bk[i].imag)
    if temp[i].imag<1/math.pow(10,13) and temp[i].imag> -1 / math.pow(10,13):
        tempI.append(0)
    else:
        tempI.append(temp[i].imag)
    tempR.append(temp[i].real)
print("Verifying time linearity property")

print("Real part of coefficients")
plt.subplot(1,2,1)
plt.stem(x,bkR)
plt.title("Case 1")

plt.xlabel('k')
plt.ylabel('a k')
plt.subplot(1,2,2)
plt.stem(x,tempR)
plt.title("Case 2")

plt.xlabel('k')
plt.show()
#both plots are same so timeshifting property verified
print("Imaginary part of coefficients")
plt.subplot(1,2,1)
plt.stem(x,bkI)
plt.title("Case 1")

plt.xlabel('k')
plt.ylabel('a k')
plt.subplot(1,2,2)
plt.stem(x,tempI)
plt.title("Case 2")

plt.xlabel('k')
plt.show()
```

```
Verifying time linearity property
Real part of coefficients
```




```
Imaginary part of coefficients
```



The computed DTFS coefficients are equal in both the cases as infered from the plots. Hence, linearity holds.

Frequency shift:

We shall use a strategy similar to the previous cases and compare the plots in the two cases to verify the property.

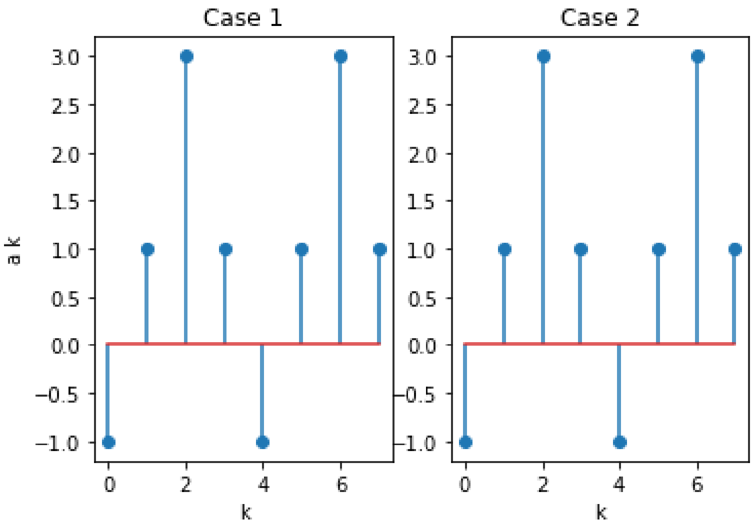
$$e^{jM(2\pi/N)n}x[n] \qquad\qquad a_{k-M}$$

In [131…

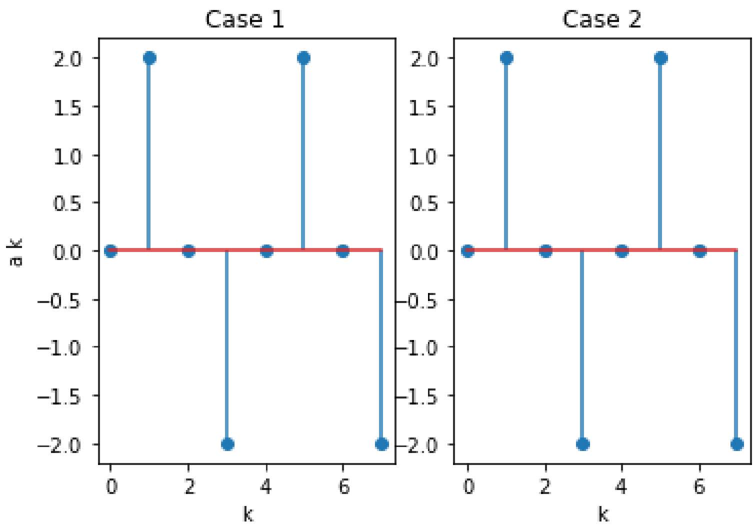
```
#frequency shift
M=2
yfs=[]
for n in x:
    yfs.append(np.exp(1j*M*np.pi*n/2)*(4*delta(n/4)+8*delta((n-1)/4)))

bk=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(yfs[n]*np.exp(-1j*k*n*np.pi/2))
    bk.append(s/4)
a_list=collections.deque(ak)
a_list.rotate(- M)
temp=list(a_list)

def plotter(x,bk,temp):
    """Utility Function to plot"""
```

```
    bkR = []
    bkI = []
    tempR = []
    tempI = []
    for i in range(len(akn)):
        bkR.append(bk[i].real)
        if bk[i].imag<1/math.pow(10,13) and bk[i].imag> -1 / math.pow(10,13):
            bkI.append(0)
        else:
            bkI.append(bk[i].imag)
        if temp[i].imag<1/math.pow(10,13) and temp[i].imag> -1 / math.pow(10,13):
            tempI.append(0)
        else:
            tempI.append(temp[i].imag)
        tempR.append(temp[i].real)
    print("Real part of coefficients")
    plt.subplot(1,2,1)
    plt.stem(x,bkR)
    plt.title("Case 1")
    plt.xlabel('k')
    plt.ylabel('a k')
    plt.subplot(1,2,2)
    plt.stem(x,tempR)
    plt.title("Case 2")
    plt.xlabel('k')
    plt.show()
    print("Imaginary part of coefficients")
    plt.subplot(1,2,1)
    plt.stem(x,bkI)
    plt.title("Case 1")

    plt.xlabel('k')
    plt.ylabel('a k')
    plt.subplot(1,2,2)
    plt.stem(x,tempI)
    plt.title("Case 2")

    plt.xlabel('k')
    plt.show()

plotter(x,bk,temp)
```

Real part of coefficients



Imaginary part of coefficients



The real and imaginary plots of the two cases are identical and hence the property has been verified

Multiplication Property

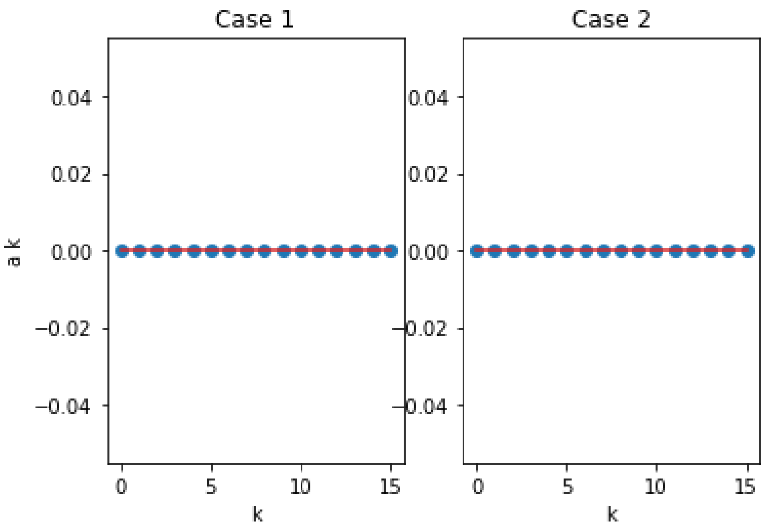
Multiplication Property states that multiplication in time domain leads to periodic convolution in the frequency domain.

$$x[n]y[n] \qquad \sum_{l=\langle N\rangle} a_l b_{k-l}$$

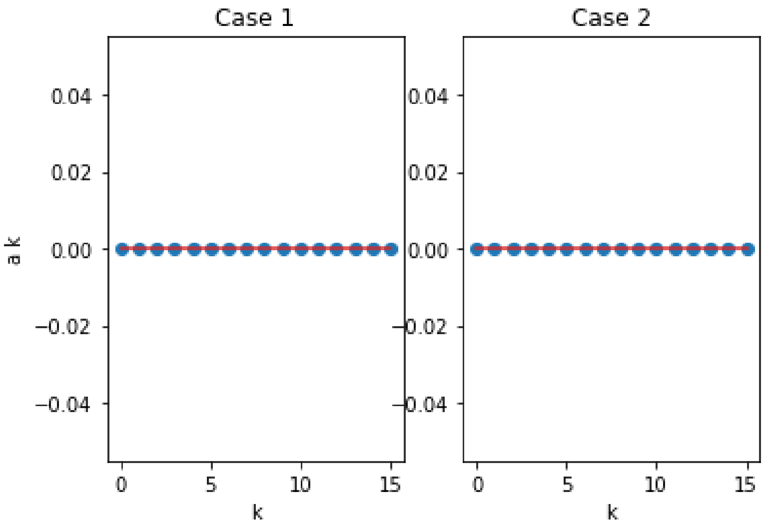
In [107…

```
#x[n] and x[n-no] be two signals
def dataYmp(n):
    return dataY(n)*dataYn(n)

ymp=[dataYmp(i) for i in x]
bk=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(dataYmp(n)*np.exp(-1j*k*n*np.pi/2))
    bk.append(s/4)
temp=[]
def dataErrorHandler(data):
    """utility function to handle internal error of python library"""
    if data <0 and data > -1/math.pow(10,-13):
        return 0
    if data >0 and data < 1/math.pow(10,13):
        return 0
    return data
for k in x:
    s=0
    for l in range(N):
        s=s+ak[l]*akn[k-l]
    temp.append(dataErrorHandler(s))
plotter(x,bk,temp)
```

```
Real part of coefficients
```




```
Imaginary part of coefficients
```



The real and imaginary plots of the two cases are identical and hence the property has been verified

Convolution Property

Analogous to the previous case, periodic convolution in the time domain leads to multiplication in the frequency domain

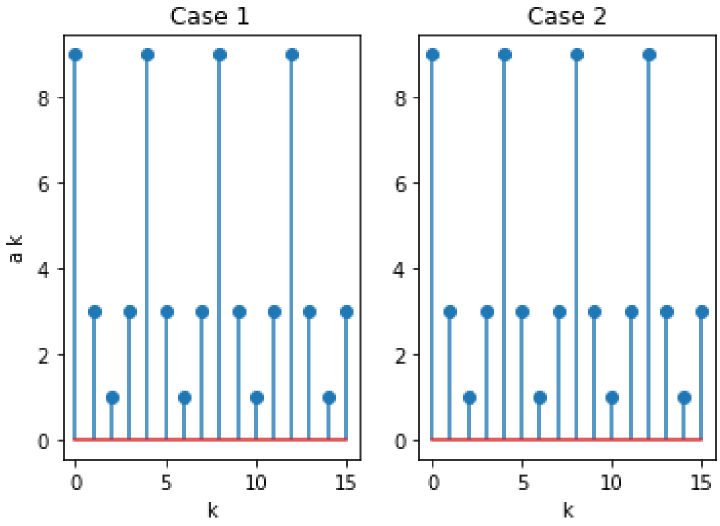
$$\sum_{r=\langle N\rangle} x[r]y[n-r] \qquad\qquad N a_k b_k$$

In [108…

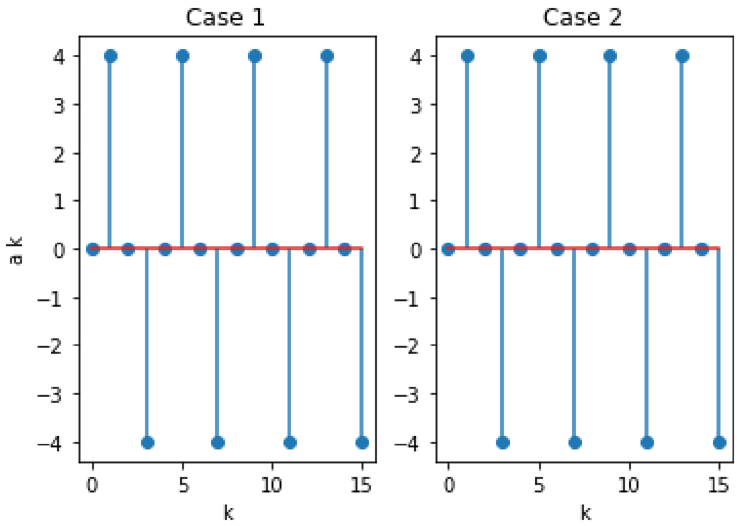
```
data = []
for i in range(len(x)):
   s = 0
   for l in range(N):
     s+=dataY(l)*dataYn(i-l)
   data.append(s/4)
ymp=data
bk=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(data[n]*np.exp(-1j*k*n*np.pi/2))
    bk.append(s/4)
temp=[ak[i]*akn[i] for i in x]
def dataErrorHandler(data):
    if data <0 and data > -1/math.pow(10,-13):
```

```
        return 0
    if data >0 and data < 1/math.pow(10,13):
        return 0
    return data
plotter(x,bk,temp)
```

```
Real part of coefficients
```




```
Imaginary part of coefficients
```



The real and imaginary plots of the two cases are identical and hence the property has been verified

Difference Property

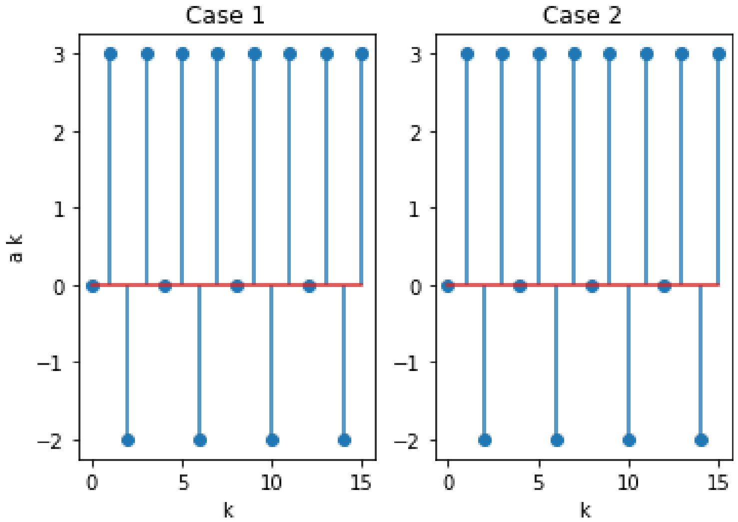
$$x[n] - x[n-1] \qquad\qquad (1 - e^{-jk(2\pi/N)})a_k$$

In [109…

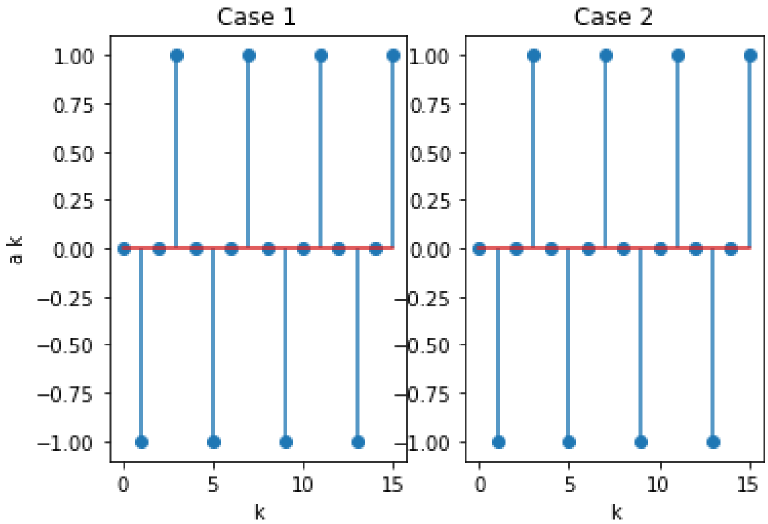
```
#difference property
no=1
yn=[]
for n in range(-1,11):
    yn.append(4*delta((n-no)/4)+8*delta((n-no-1)/4))
ydiff=[y[i]-y[i-1] for i in x]
bk=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(ydiff[n]*np.exp(-1j*k*n*np.pi/2))
    bk.append(s/4)
temp=[]
for k in x:
```

```
    temp.append(ak[k]*(1-np.exp(-1j*k*np.pi*no/2)))
plotter(x,bk,temp)
```

```
Real part of coefficients
```




```
Imaginary part of coefficients
```



The real and imaginary plots of the two cases are identical and hence the property has been verified

Symmetry Property

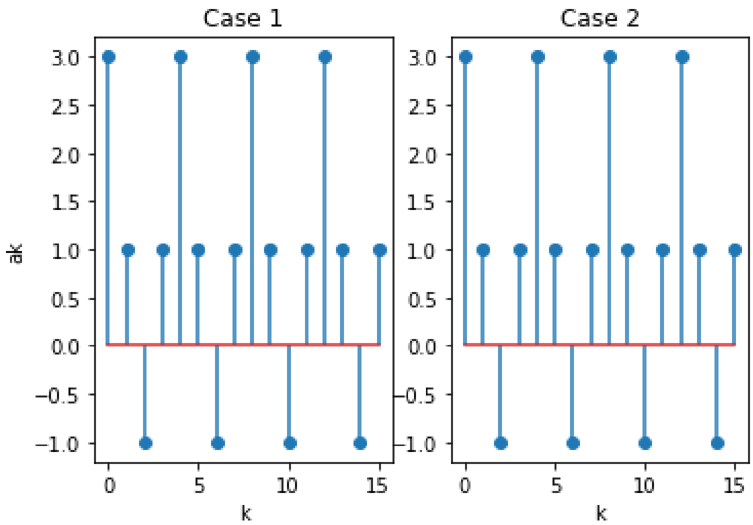
$$x[n] \text{ real} \qquad \begin{cases} a_k = a^*_{-k} \\ \mathcal{R}e\{a_k\} = \mathcal{R}e\{a_{-k}\} \\ \mathcal{I}m\{a_k\} = -\mathcal{I}m\{a_{-k}\} \\ |a_k| = |a_{-k}| \\ \measuredangle a_k = -\measuredangle a_{-k} \end{cases}$$

In [110…

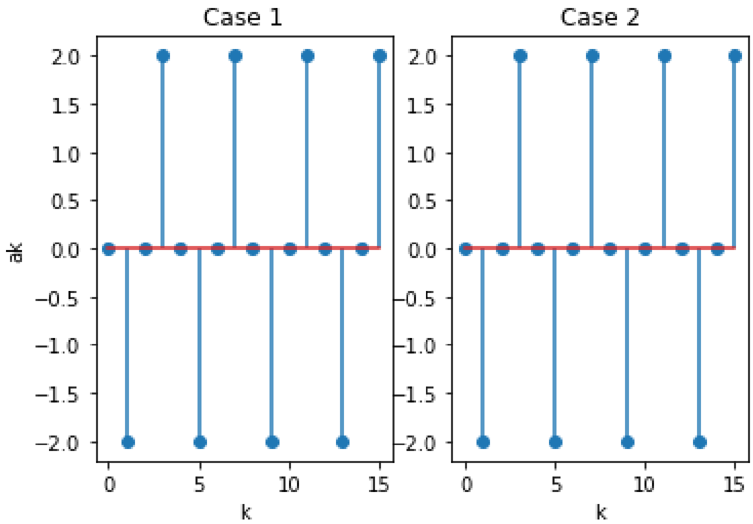
```
a_k=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(y[n]*np.exp(-1j*-1*k*n*np.pi/2))
    a_k.append(s/4)
a__k=[np.conj(i) for i in a_k]
a__kr=[]
a__kimg=[]
for i in ak:
```

```
    a__kr.append(i.real)
    a__kimg.append(i.imag)

print("real part of coefficients")
plt.subplot(1,2,1)
plt.stem(x,akr)
plt.title('Case 1')
plt.xlabel('k')
plt.ylabel('ak')
plt.subplot(1,2,2)
plt.stem(x,a__kr)
plt.xlabel('k')
plt.title('Case 2')
plt.show()

print("imaginary part of coefficients")
plt.subplot(1,2,1)
plt.stem(x,akimg)
plt.title('Case 1')
plt.xlabel('k')
plt.ylabel('ak')
plt.subplot(1,2,2)
plt.stem(x,a__kimg)
plt.xlabel('k')
plt.title('Case 2')
plt.show()
```

```
real part of coefficients
```




```
imaginary part of coefficients
```



The real and imaginary plots of the two cases are identical and hence the property has been verified

3.

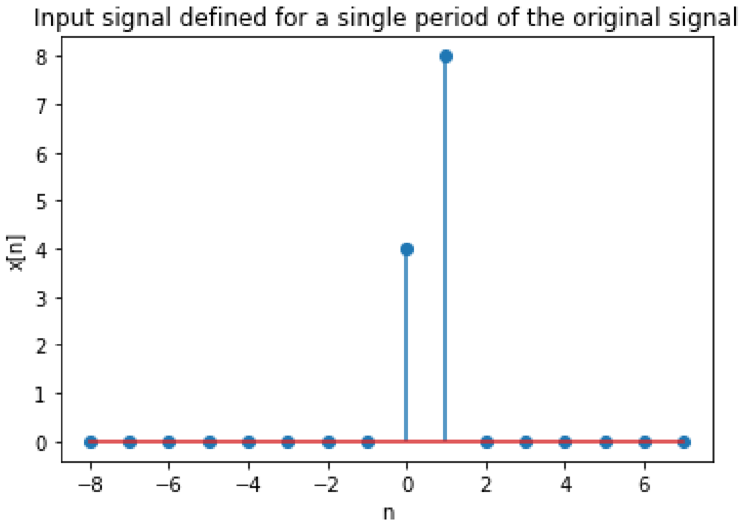
The given signal is:

$$x[n] = \sum_{m=-\infty}^{\infty} \{4\delta[n-4m] + 8\delta[n-1-4m]\}$$

Now we shall find the DTFT coefficients for a period of the given signal. Since we are considering only a single period of the signal, the signal is aperiodic and will correspond to a periodic and continuous frequency domain representation. The following result shall be used for defining the same:

In [111…

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

x=np.arange(-8,8)
def delta(x):
    if x.is_integer():
        return 1
    else:
        return 0
y=[]
for n in x:
    if n<4 and n>=0:
        y.append(4*delta(n/4)+8*delta((n-1)/4))
    else:
        y.append(0)
plt.title("Input signal defined for a single period of the original signal")
plt.xlabel("n")
plt.ylabel("x[n]")
plt.stem(x,y)
plt.show()
```



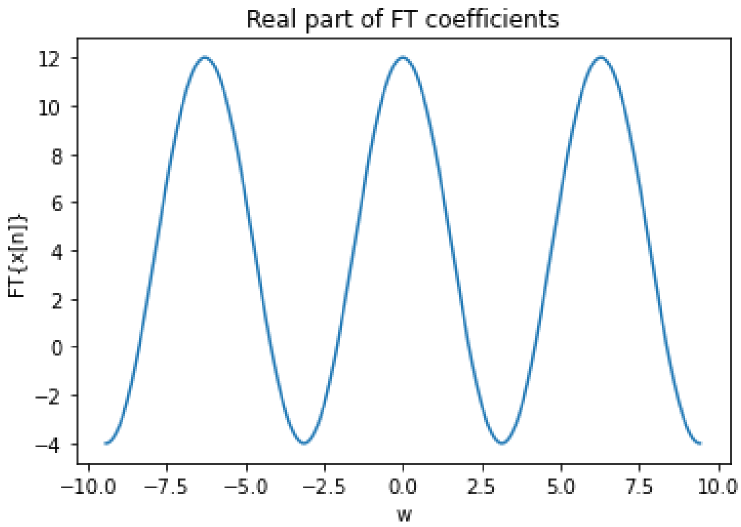
We shall now use the following relations to compute the DTFT coefficients of the given signal. We shall plot the real and imaginary parts of the coefficient separately with the frequency being the dependant variable. The following results are used to compute the coefficients:

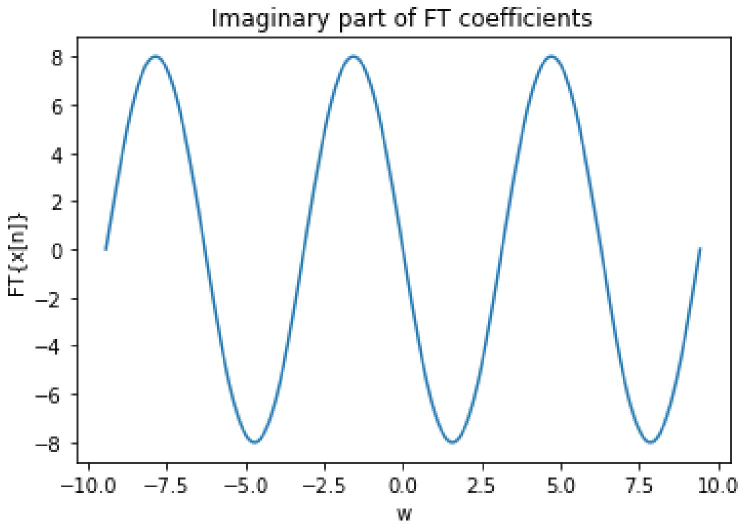
$$x[n] = \frac{1}{2\pi} \int_{2\pi} X(e^{j\omega}) e^{j\omega n} \, d\omega,$$

$$X(e^{j\omega}) = \sum_{n=-\infty}^{+\infty} x[n] e^{-j\omega n}.$$

In [112…

```
omega=np.linspace(-3*np.pi,3*np.pi,500)

def ftCoefficients(y):
    x_FT = []

    for w in omega:
        data = 0
        for i in range(len(x)):
            data += y[i]*np.exp(-1j*w*x[i])
        x_FT.append(data)
    return x_FT

x_FT = ftCoefficients(y)
x_FT_R = []
x_FT_I = []
for data in x_FT:
    x_FT_R.append(data.real)
    x_FT_I.append(data.imag)
plt.title("Real part of FT coefficients")
plt.xlabel("w")
plt.ylabel("FT{x[n]}")
plt.plot(omega,x_FT_R)

plt.show()
plt.title("Imaginary part of FT coefficients")
plt.xlabel("w")
plt.ylabel("FT{x[n]}")
plt.plot(omega,x_FT_I)

plt.show()
```





The real part of coefficients must be even function and periodic while the imaginary part must be odd and periodic which is evident from the above plot.

Properties of DTFT

We shall now verify the following properties of DTFT for the given signal in the way similar to the one in previous case.
Our strategy remains the same and is to verify it is to get the coefficients of the DTFS in 2 ways -

- By doing an operation on the signal in the time domain - Case 1
- By doing the corresponding operation on the coefficients - Case 2

Time shifting property

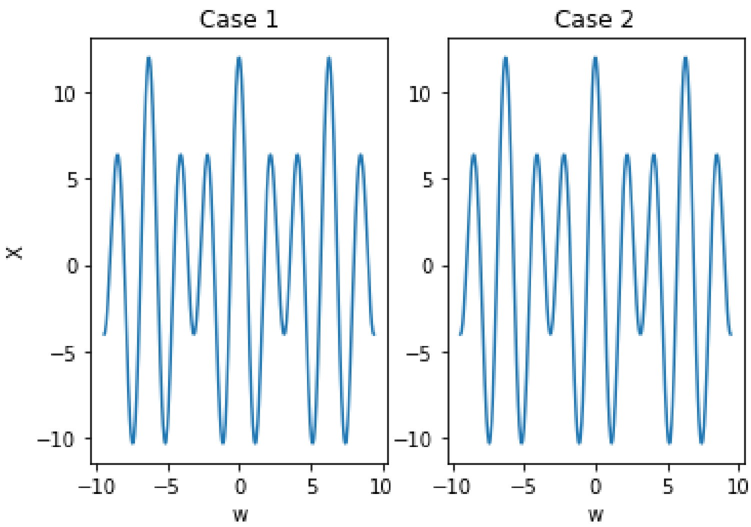
$$x[n-n_0] \overset{\mathfrak{F}}{\longleftrightarrow} e^{-j\omega n_0} X(e^{j\omega})$$

In [113…

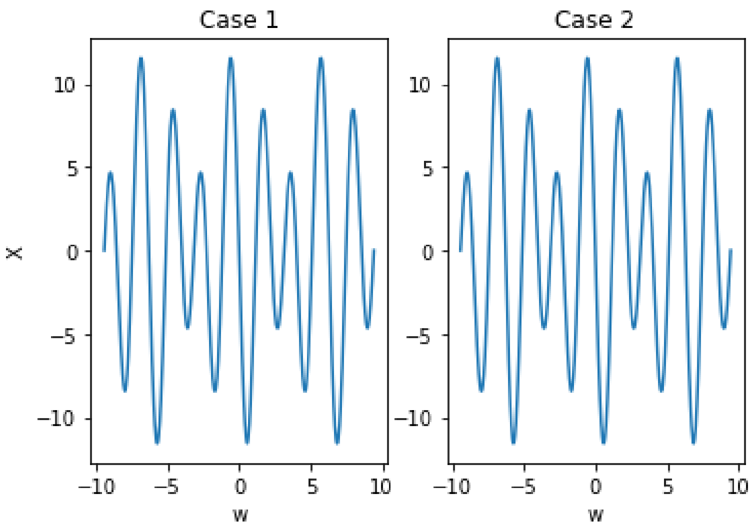
```
#properties of DTFT
#time shifting
n0=2
yn=[]
for n in x:
```

```python
    if n-n0>=0 and n-n0<4:
        yn.append(4*delta((n-n0)/4)+8*delta((n-n0-1)/4))
    else:
        yn.append(0)
x_FTn = ftCoefficients(yn)
temp=[]
for i in range(len(omega)):
    temp.append(x_FT[i]*np.exp(-1j*omega[i]*n0))

def nullChecker(n):
    if n < 1/math.pow(10,10) and n > -1/math.pow(10,10) and n!=0:
        return 0
    return n

def plotter(temp,ftn,w,label):
    tempR = []
    tempI = []
    ftnR = []
    ftnI = []
    for i in range(len(temp)):
        tempR.append(temp[i].real)
        tempI.append(nullChecker(temp[i].imag))
        ftnR.append(ftn[i].real)
        ftnI.append(nullChecker(ftn[i].imag))
    print("Real parts")
    plt.subplot(1,2,1)
    plt.plot(w,ftnR)
    plt.title("Case 1")
    plt.xlabel('w')
    plt.ylabel('X')
    plt.subplot(1,2,2)
    plt.plot(w,tempR)
    plt.title("Case 2")
    plt.xlabel('w')
#     plt.title(label)
    plt.show()
    print("Imaginary parts")
    plt.subplot(1,2,1)
    plt.plot(w,ftnI)
    plt.title("Case 1")
    plt.xlabel('w')
    plt.ylabel('X')
    plt.subplot(1,2,2)
    plt.plot(w,tempI)
    plt.title("Case 2")
    plt.xlabel('w')
#     plt.title(label)
    plt.show()
plotter(temp,x_FTn,omega,"time shift")
```

```
Real parts
```




```
Imaginary parts
```



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

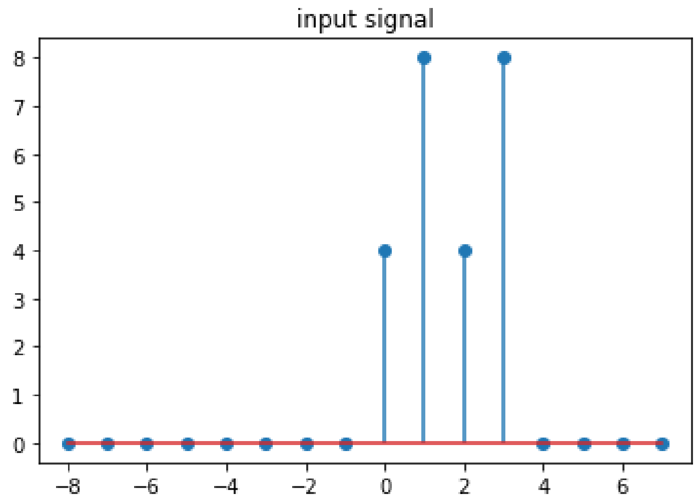
Linearity

To demonstrate linearity, we shall take $yl[n] = y[n] + yn[n]$ and use the strategy discussed already

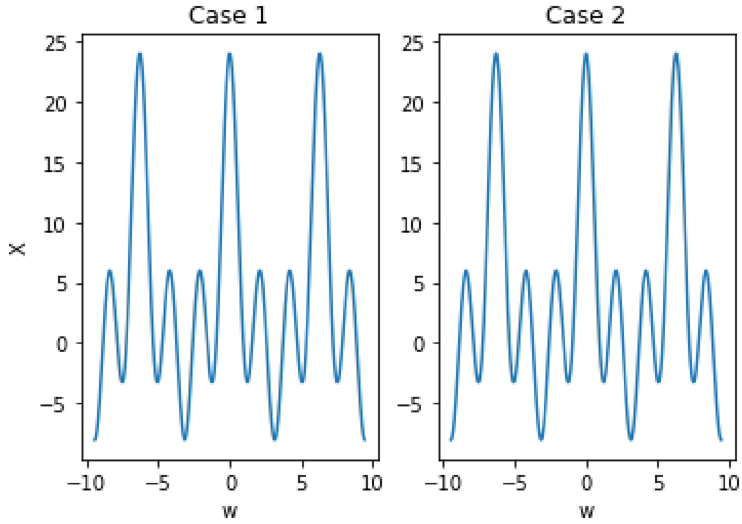
$$a x_1[n] + b x_2[n] \stackrel{\mathfrak{F}}{\longleftrightarrow} a X_1(e^{j\omega}) + b X_2(e^{j\omega}).$$

In [114...

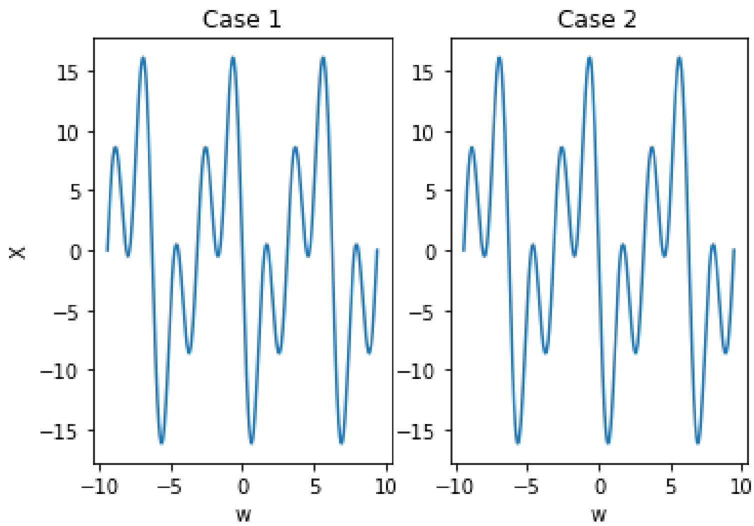
```
#linearity
#x[n] and x[n-no] are input signals
yl=[y[i]+yn[i] for i in range(len(x))]
plt.stem(x,yl)
plt.title("input signal")
plt.show()
FTl = ftCoefficients(yl)
temp=[x_FT[i]+x_FTn[i] for i in range(len(omega))]
plotter(temp,FTl,omega,"linearity")
```



Real parts



Imaginary parts



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

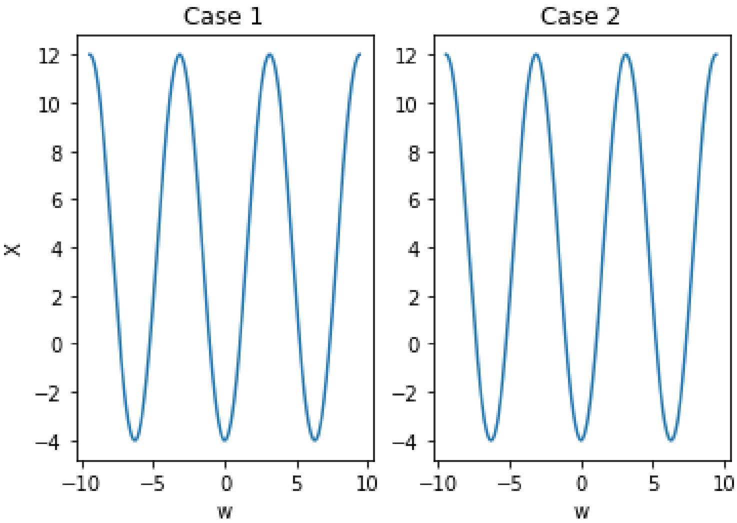
Frequency Shifting

$$e^{j\omega_0 n}x[n] \xleftrightarrow{\mathfrak{F}} X(e^{j(\omega-\omega_0)}).$$

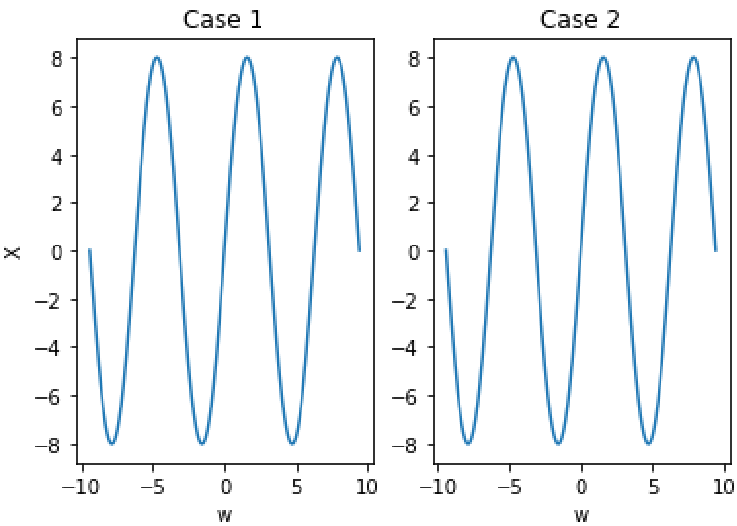
In [115…

```
w0 = np.pi
x_FTw = []
for w in omega:
        data = 0
        for i in range(len(x)):
            data += y[i]*np.exp(-1j*(w-w0)*x[i])
        x_FTw.append(data)

def FTcoeff(y,w):
    data = 0
    for i in range(len(x)):
        data += y[i]*np.exp(-1j*w*x[i])
    return data
temp = []
for w in omega:
    temp.append(FTcoeff(y,w-w0))

plotter(temp,x_FTw,omega,"periodicity")
```

Real parts



Imaginary parts



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Periodicity

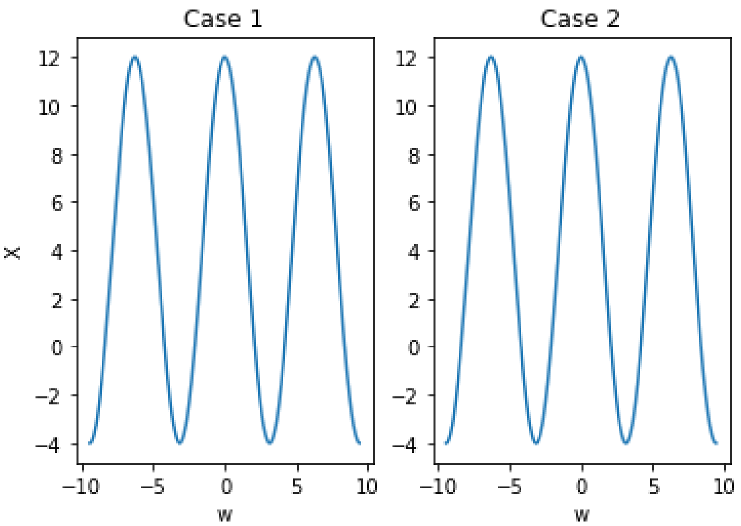
We basically perform a frequency shift of 2pi

$$\boxed{X(e^{j(\omega+2\pi)}) = X(e^{j\omega}).}$$

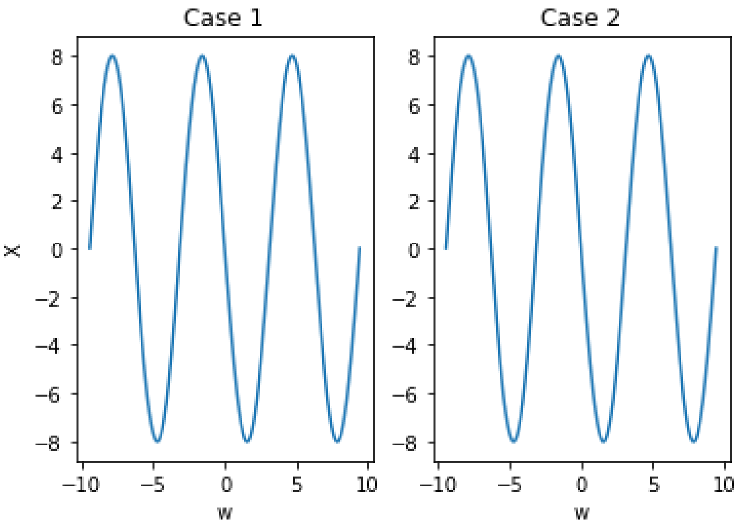
In [116...

```
w0 = 2*np.pi
x_FTw = []
for w in omega:
        data = 0
        for i in range(len(x)):
            data += y[i]*np.exp(-1j*(w+w0)*x[i])
        x_FTw.append(data)

plotter(x_FT,x_FTw,omega,"periodicity")
```

```
Real parts
```




```
Imaginary parts
```



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

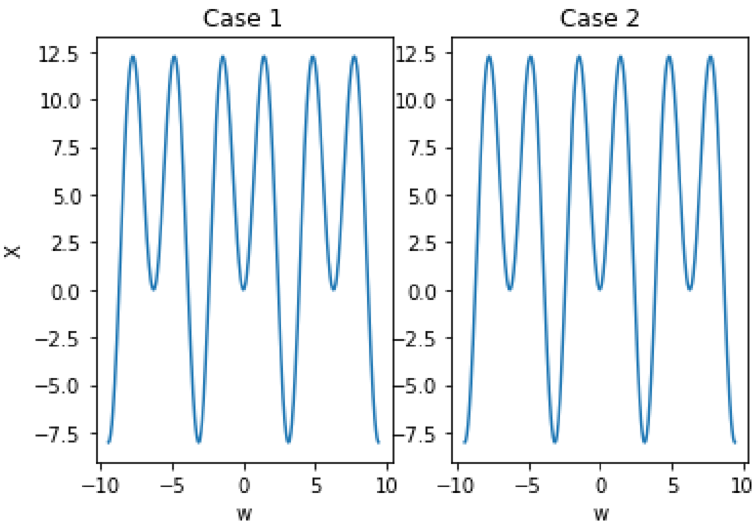
Difference Property

$$x[n] - x[n-1] \xleftrightarrow{\mathfrak{F}} (1 - e^{-j\omega})X(e^{j\omega}).$$

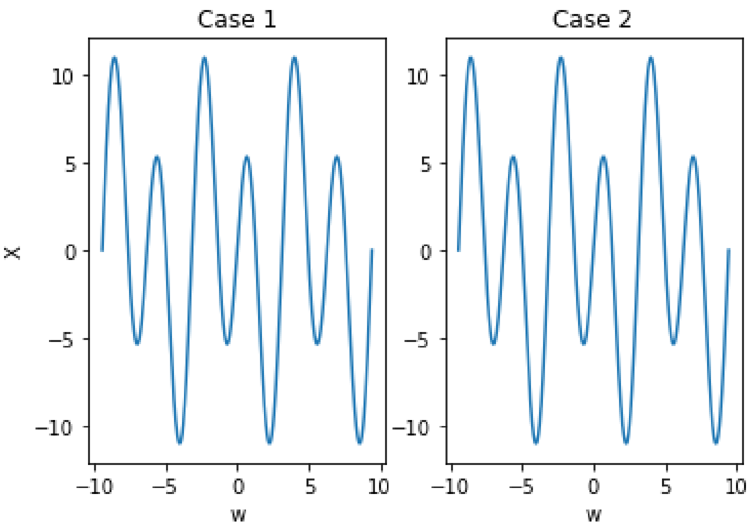
In [117…

```
#difference property
no=1
yn=[]
# print(x)
for n in x:
    if n-no<4 and n-no>=0:
        yn.append(4*delta((n-no)/4)+8*delta((n-no-1)/4))
    else:
        yn.append(0)
ydiff=[y[i]-yn[i] for i in range(len(y))]
x_FTdiff = ftCoefficients(ydiff)
temp=[]
for i in range(len(omega)):
    temp.append((1-np.exp(-1j*omega[i]))*x_FT[i])
plotter(temp,x_FTdiff,omega,"periodicity")
```

Real parts



Imaginary parts



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Time Reversal

$$x[-n] \stackrel{\mathfrak{F}}{\longleftrightarrow} X(e^{-j\omega}).$$

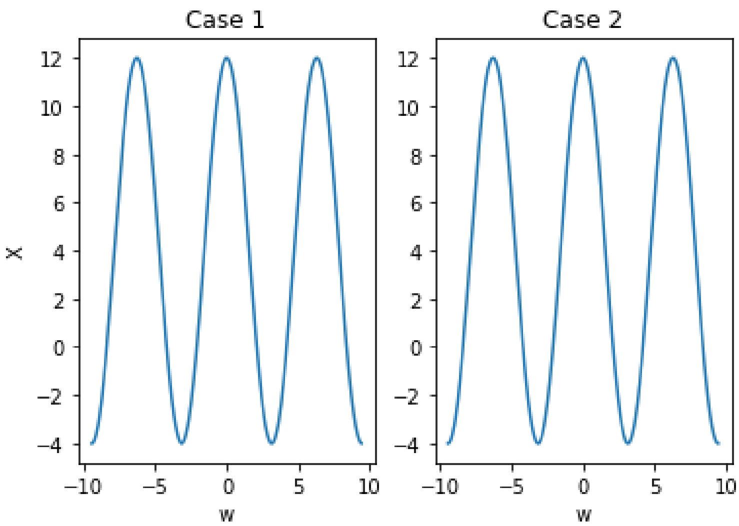
As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Symmetry property

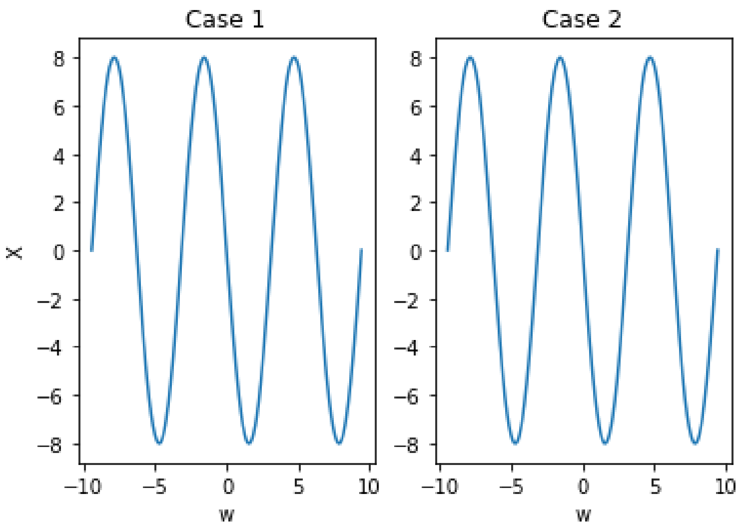
In [118...

```
#symmetry
def symmft(y):
    x_FT = []
    for w in omega:
        data = 0
        for i in range(len(x)):
            data += y[i]*np.exp(1j*w*x[i])
        x_FT.append(data)
    return x_FT

FTsy=symmft(y)
FTsy1=[np.conj(i) for i in FTsy]
print("X(e^jw) = X*(e^-jw)")
plotter(FTsy1,x_FT,omega,"symmetry property")
print("Re(X(e^jw)) = Re (X(e^-jw))")
print("Im(X(e^jw)) = -Im(X(e^-jw))")
plotter(FTsy,x_FT,omega,"symmetry property")
```

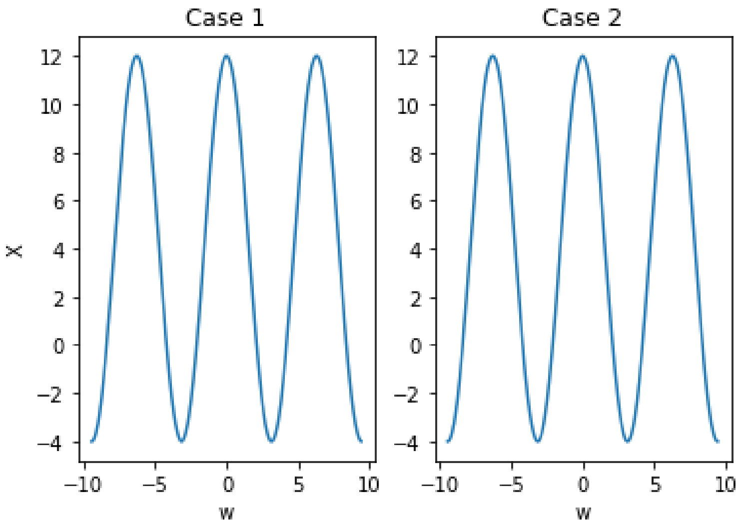
```
X(e^jw) = X*(e^-jw)
Real parts
```



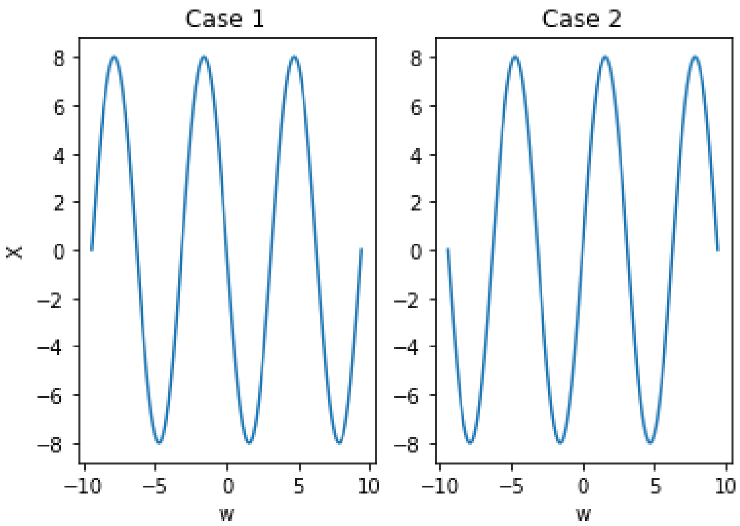

```
Imaginary parts
```




```
Re(X(e^jw)) = Re (X(e^-jw))
Im(X(e^jw)) = -Im(X(e^-jw))
Real parts
```




```
Imaginary parts
```



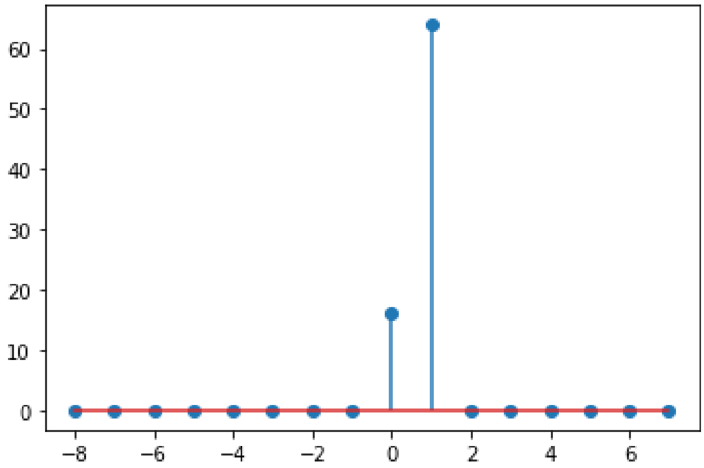
As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Multiplication Property

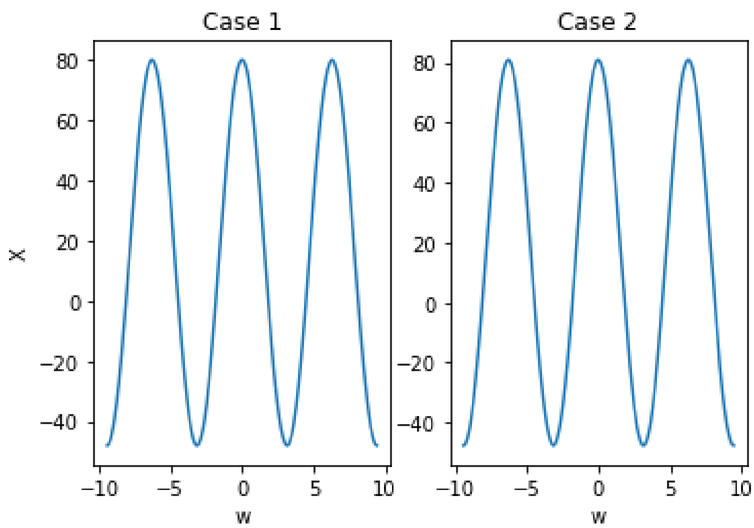
$x[n]y[n]$ $\qquad$ $\frac{1}{2\pi}\int_{2\pi} X(e^{j\theta})Y(e^{j(\omega-\theta)})d\theta$

In [119…

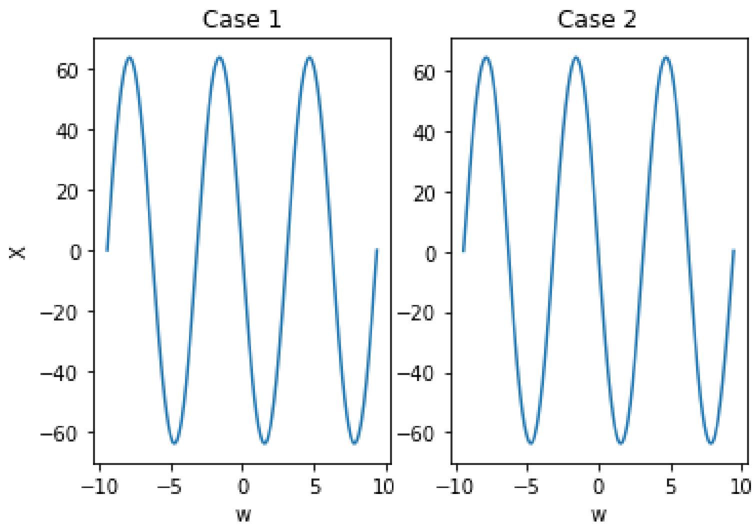
```
y_multiplication = []
for i in range(len(y)):
    y_multiplication.append(y[i]*yl[i])
plt.stem(x,y_multiplication)
plt.show()
ft_multiplication = ftCoefficients(y_multiplication)
temp_multiplication = []
for w in omega:
    data = 0
    for t in range(len(omega)):
        if omega[t] >=0 and omega[t] <= 2*np.pi:
            data+= FTcoeff(y,t)*FTcoeff(yl,w-t)*(6*np.pi/500)
        if omega[t] > 2*np.pi:
            break
    temp_multiplication.append(data/(np.pi*2))
plotter(temp_multiplication,ft_multiplication,omega,"multiplication property")
```




```
Real parts
```




```
Imaginary parts
```



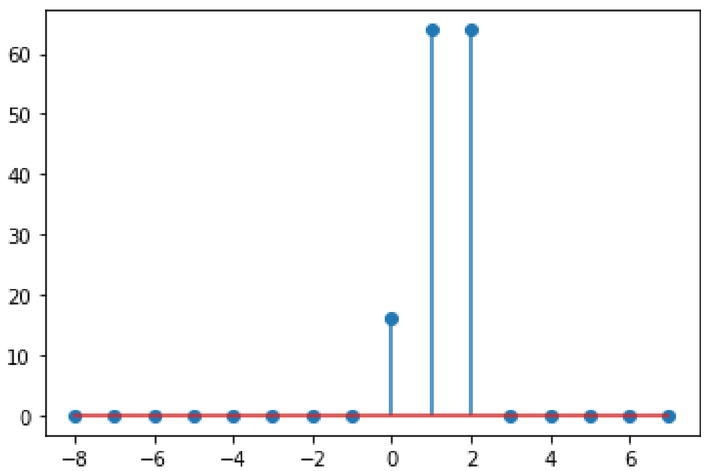
As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Convolution Property

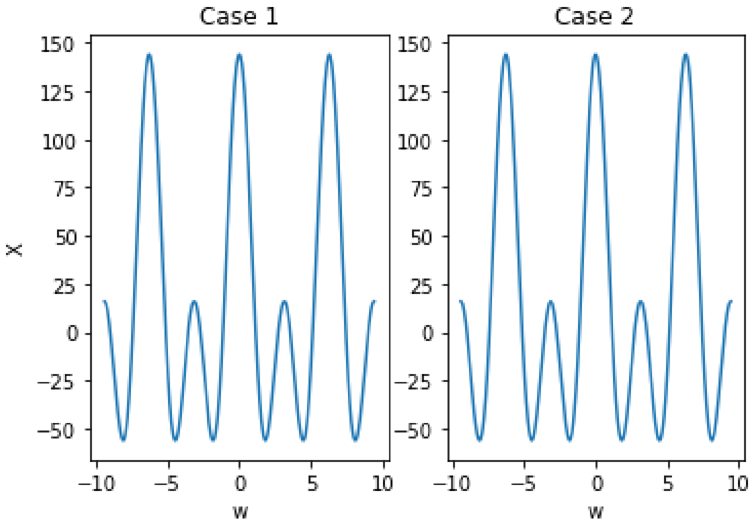
$$x[n] * y[n] \qquad\qquad X(e^{j\omega})Y(e^{j\omega})$$

In [120…

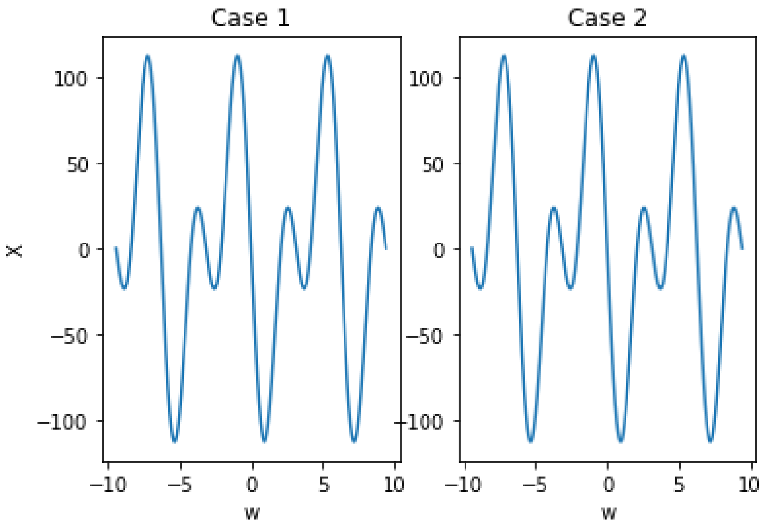
```
y_convolution = []
for i in range(len(x)):
    data = 0
    for k in range(len(x)):
#      input signal is non-zero only at n=0,1
        if x[k] < 2 or x[k] >= 0:
            data += y[x[k]] * y[x[i]-x[k]]
    y_convolution.append(data)
x_FTconv = ftCoefficients(y_convolution)
plt.stem(x,y_convolution)
plt.show()

temp_convolution = [n*n for n in x_FT]
plotter(temp_convolution,x_FTconv,omega,"convolution property")
```



```
Real parts
```




```
Imaginary parts
```



Now we shall do some extensions of problem 3 of the assignment. In the assignment, we were limited to a period of the given signal. Let us now remove that restriction to see what happens:

3.

The given signal is:

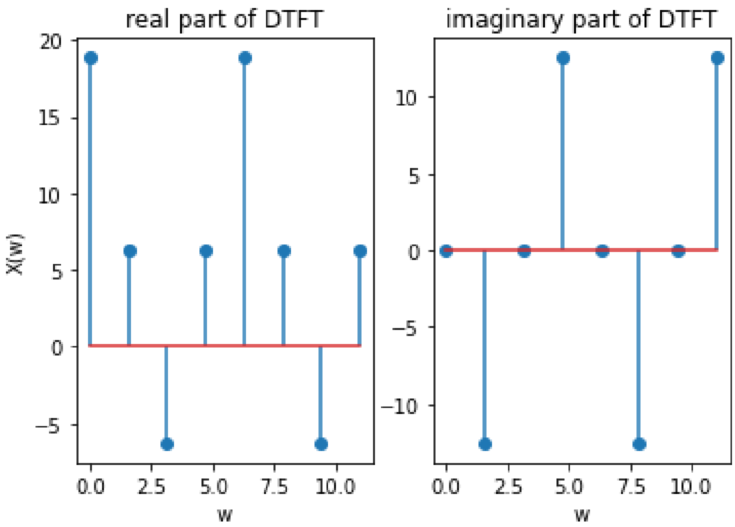
$$x[n] = \sum_{m=-\infty}^{\infty} \{4\delta[n-4m] + 8\delta[n-1-4m]\}$$

Now we shall find the DTFT coefficients for the given signal. Since the given signal is a periodic signal, the frequency domain representation will be impulses which are periodic in nature.

$$X(e^{j\omega}) = \sum_{k=-\infty}^{+\infty} 2\pi a_k \delta\left(\omega - \frac{2\pi k}{N}\right),$$

In [133…

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

x=np.arange(0,8)

def delta(x):
    if x.is_integer():
        return 1
    else:
        return 0
#3
y=[]
for n in x:
    y.append(4*delta(n/4)+8*delta((n-1)/4))

N=4#fundamental period

def FSCoeff(y,N):
    ak=[]#fourier series coefficient
    for k in x:
        s=0
        for n in range(0,N):
            s=s+(y[n]*np.exp(-1j*k*n*np.pi/2))
        ak.append(s/4)
    return ak
ak = FSCoeff(y,N)
w=[]
for k in x:
    w.append((2*np.pi*k)/N)

ft=[2*np.pi*ak[i] for i in x]
ftr=[]
ftimg=[]
for i in ft:
    ftr.append(i.real)
    ftimg.append(i.imag)

plt.subplot(1,2,1)
plt.stem(w,ftr)
plt.xlabel('w')
plt.ylabel('X(w)')
plt.title('real part of DTFT')
plt.subplot(1,2,2)
plt.stem(w,ftimg)
plt.xlabel('w')
plt.title('imaginary part of DTFT')
plt.show()
```



Properties of DTFT

We shall now verify the following properties of DTFT for the given signal in the way similar to the one in previous case.
Our strategy remains the same and is to verify it is to get the coefficients of the DTFS in 2 ways -

- By doing an operation on the signal in the time domain - Case 1
- By doing the corresponding operation on the coefficients - Case 2

Time shifting property

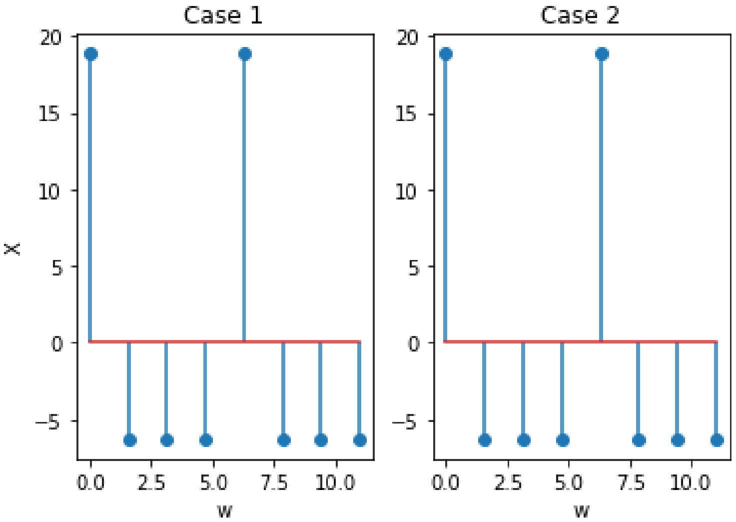
$$x[n-n_0] \stackrel{\mathfrak{F}}{\longleftrightarrow} e^{-j\omega n_0}X(e^{j\omega})$$

In [134...

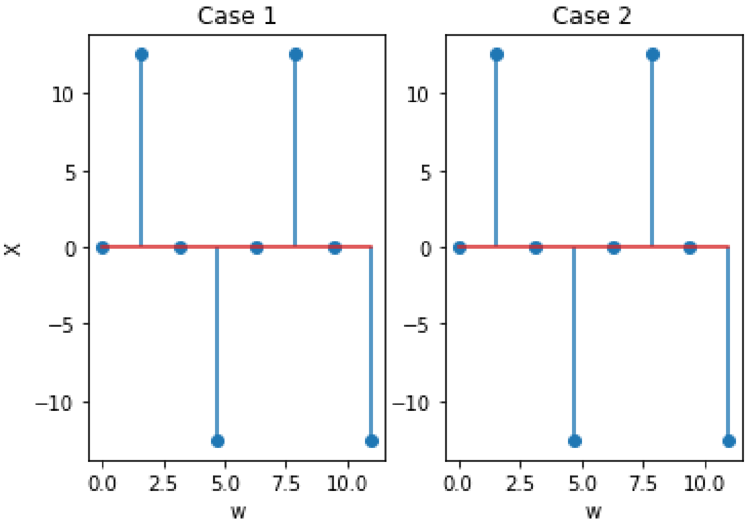
```
#properties of DTFT
#time shifting
no=2
yn=[]
for n in x:
    yn.append(4*delta((n-no)/4)+8*delta((n-no-1)/4))

akn = FSCoeff(yn,N)
ftn=[2*np.pi*akn[i] for i in x]
temp=[]
for i in range(len(x)):
    temp.append(ft[x[i]]*np.exp(-1j*w[i]*no))

def nullChecker(n):
    if n < 1/math.pow(10,10) and n > -1/math.pow(10,10) and n!=0:
        return 0
    return n

def plotter(temp,ftn,w,label):
    tempR = []
    tempI = []
    ftnR = []
    ftnI = []
    for i in range(len(temp)):
        tempR.append(temp[i].real)
        tempI.append(nullChecker(temp[i].imag))
```

```
        ftnR.append(ftn[i].real)
        ftnI.append(nullChecker(ftn[i].imag))
    print("Real parts")
    plt.subplot(1,2,1)
    plt.stem(w,ftnR)
    plt.title("Case 1")
    plt.xlabel('w')
    plt.ylabel('X')
    plt.subplot(1,2,2)
    plt.stem(w,tempR)
    plt.title("Case 2")
    plt.xlabel('w')
#     plt.title(label)
    plt.show()
    print("Imaginary parts")
    plt.subplot(1,2,1)
    plt.stem(w,ftnI)
    plt.title("Case 1")
    plt.xlabel('w')
    plt.ylabel('X')
    plt.subplot(1,2,2)
    plt.stem(w,tempI)
    plt.title("Case 2")
    plt.xlabel('w')
#     plt.title(label)
    plt.show()
plotter(temp,ftn,w,"time shift")
```

```
Real parts
```




```
Imaginary parts
```



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Linearity

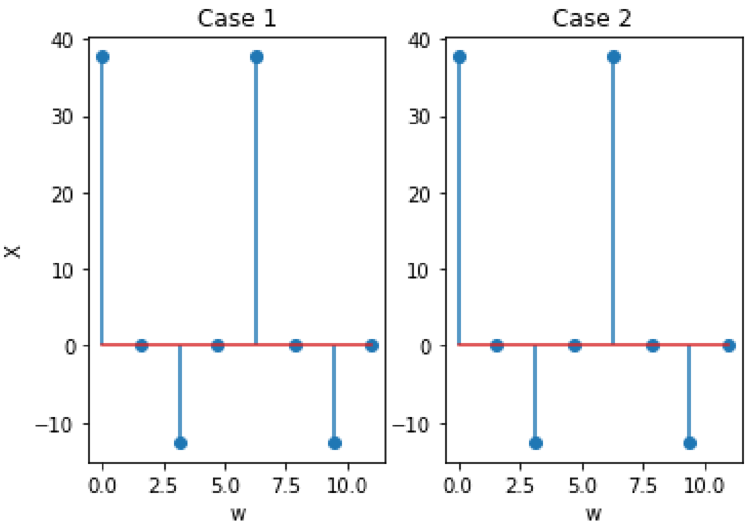
To demonstrate linearity, we shall take $yl[n] = y[n] + yn[n]$ and use the strategy discussed already

$$ax_1[n] + bx_2[n] \overset{\mathfrak{F}}{\longleftrightarrow} aX_1(e^{j\omega}) + bX_2(e^{j\omega}).$$

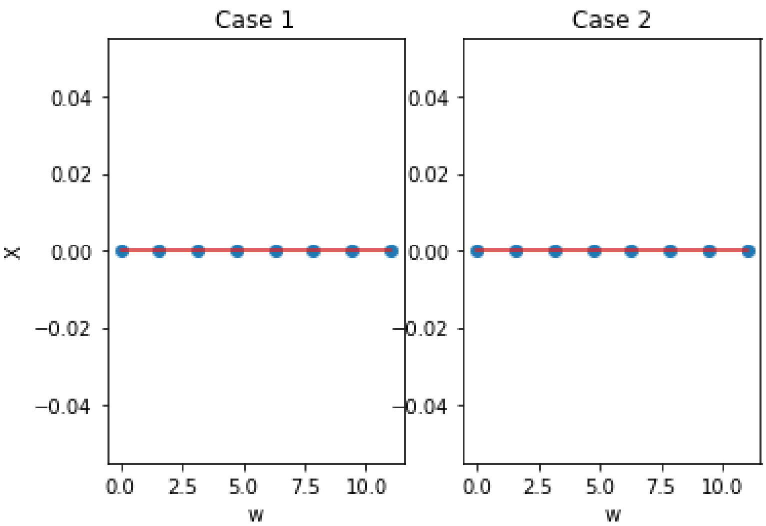
In [135…

```python
yl = []
for i in range(len(yn)):
    yl.append(yn[i]+y[i])
akl = FSCoeff(yl,N)
ftl=[2*np.pi*akl[i] for i in x]
temp = []
for i in range(len(x)):
    temp.append(ftn[i]+ft[i])
plotter(temp,ftl,w,"linearity")
```

```
Real parts
```




```
Imaginary parts
```



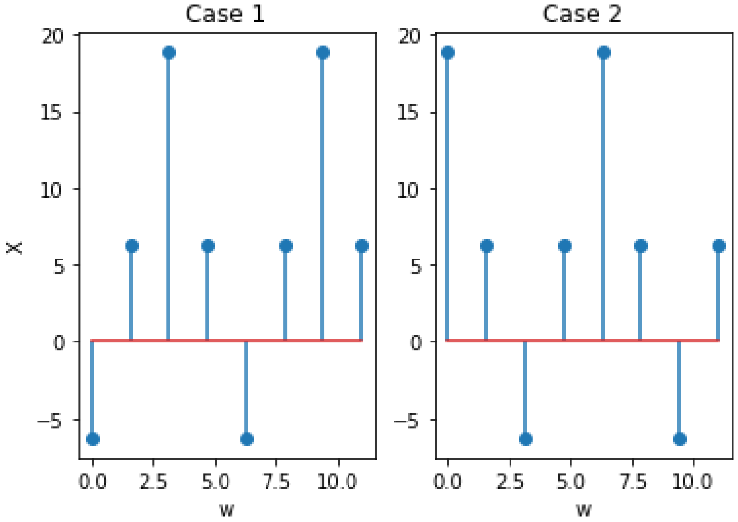
As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Frequency Shifting

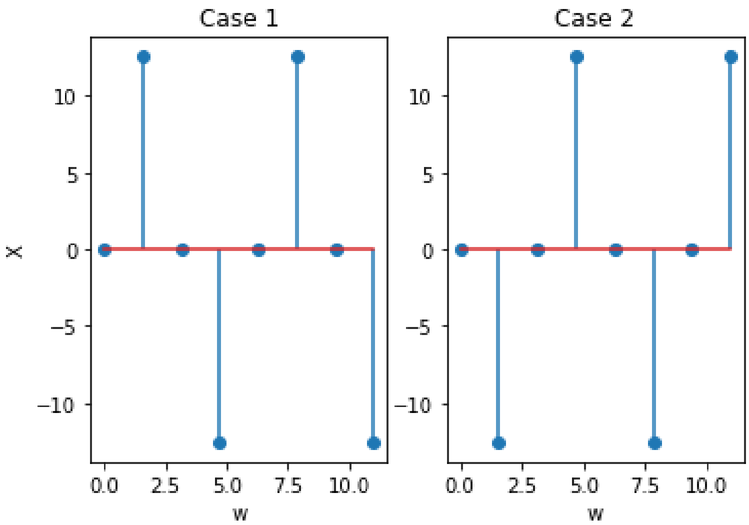
$$\boxed{e^{j\omega_0 n}x[n] \stackrel{\mathfrak{F}}{\longleftrightarrow} X(e^{j(\omega-\omega_0)}).}$$

```
In [136…

w0 = np.pi
yw0 = []
for n in x:
    yw0.append((4*delta(n/4)+8*delta((n-1)/4))*np.exp(1j*w0*n))
akw0 = FSCoeff(yw0,N)
ftw0=[2*np.pi*akw0[i] for i in x]
plotter(ft,ftw0,w,"frequency shift")
```

```
Real parts
```




```
Imaginary parts
```



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Periodicity

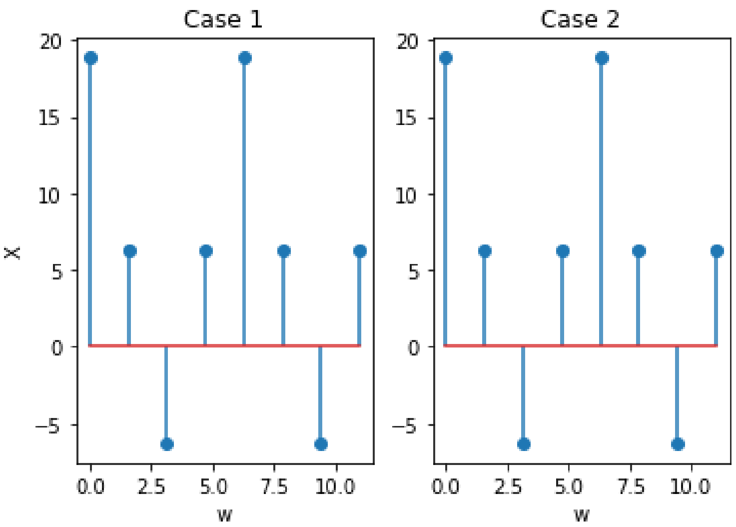
We basically perform a frequency shift of 2pi

$$\boxed{X(e^{j(\omega+2\pi)}) = X(e^{j\omega}).}$$

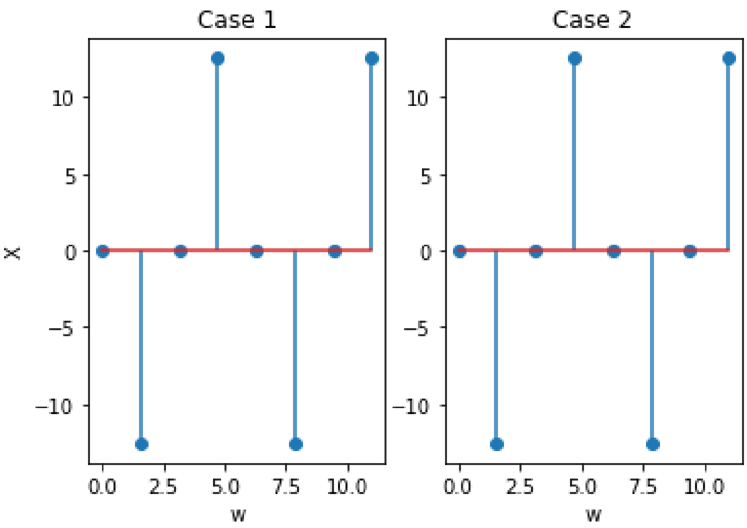
In [137…

```
w0 = 2*np.pi
yw0 = []
for n in x:
    yw0.append((4*delta(n/4)+8*delta((n-1)/4))*np.exp(1j*w0*n))
akw0 = FSCoeff(yw0,N)
ftw0=[2*np.pi*akw0[i] for i in x]
plotter(ft,ftw0,w,"periodicity")
```

```
Real parts
```




```
Imaginary parts
```



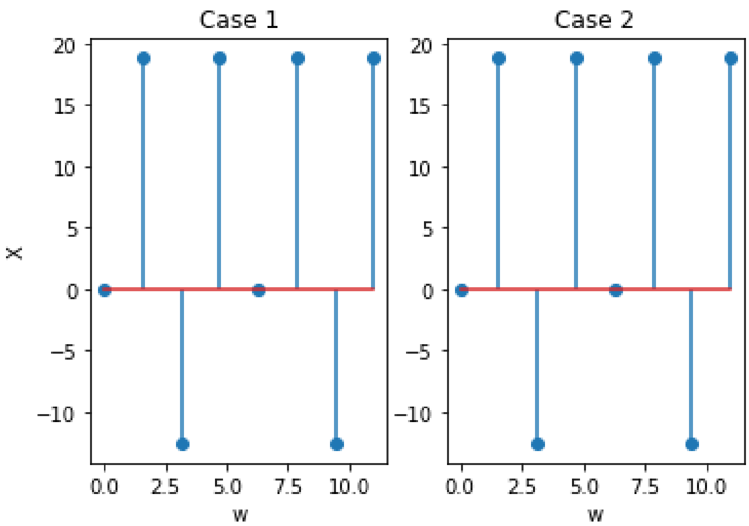
As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Difference Property

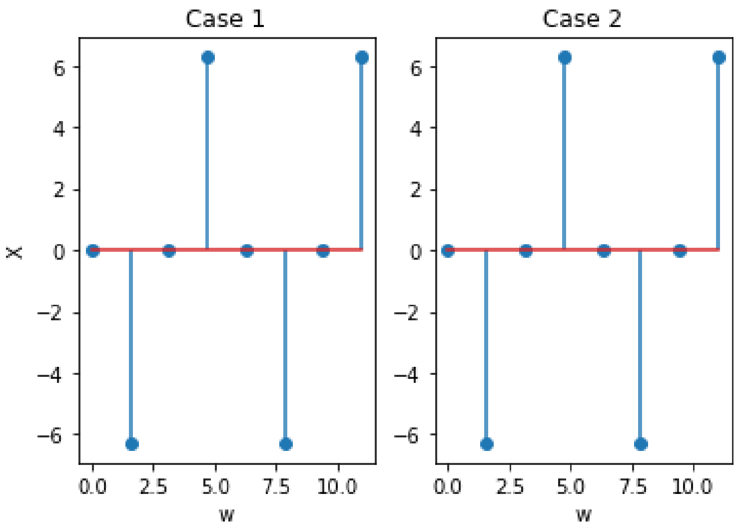
$$\boxed{x[n] - x[n-1] \stackrel{\mathfrak{F}}{\longleftrightarrow} (1 - e^{-j\omega})X(e^{j\omega}).}$$

```
In [138…  #difference property
no=1
yn=[]
for n in range(-1,11):
    yn.append(4*delta((n-no)/4)+8*delta((n-no-1)/4))
ydiff=[y[i]-y[i-1] for i in x]
bk=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(ydiff[n]*np.exp(-1j*k*n*np.pi/2))
    bk.append(s/4)
fdiff=[2*np.pi*bk[i] for i in x]
temp=[]
for i in x:
    temp.append((1-np.exp(-1j*w[i]))*ft[i])
plotter(temp,fdiff,w," ")
```

Real parts



Imaginary parts



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

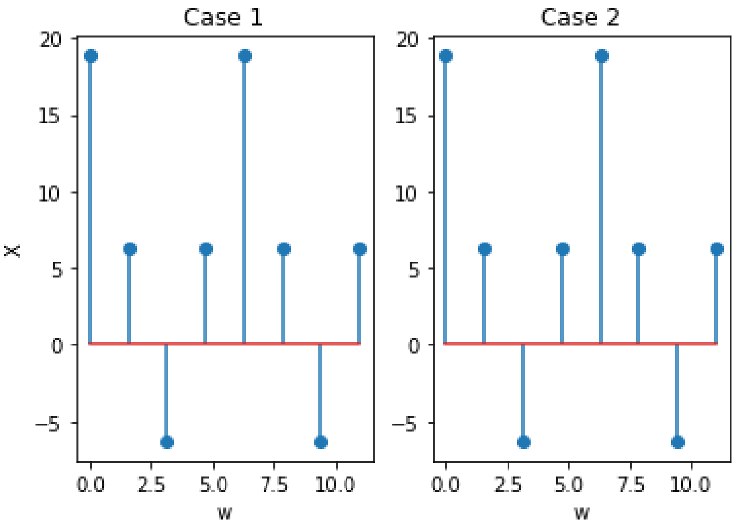
Time Reversal

$$x[-n] \xleftrightarrow{\mathfrak{F}} X(e^{-j\omega}).$$

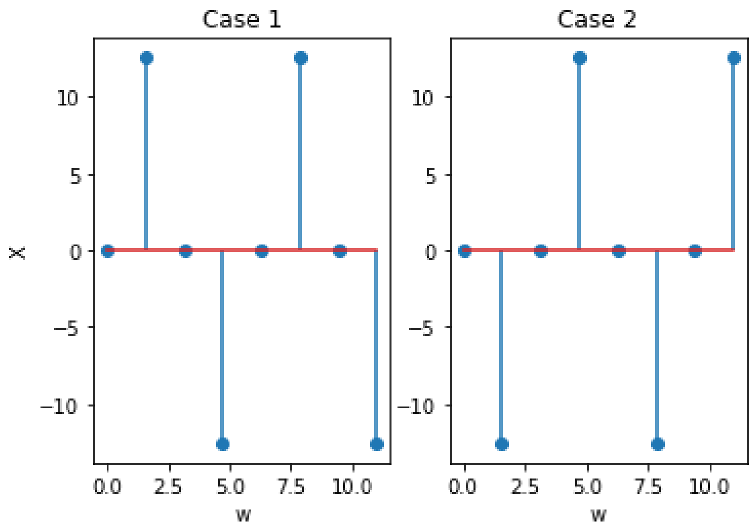
In [140…

```
#time reversal
yrev=[]
for n in x:
    yrev.append(4*delta(-1*n/4)+8*delta((-1*n-1)/4))

bk=[]
for k in x:
    s=0
    for n in range(0,N):
        s=s+(yrev[n]*np.exp(-1j*k*n*np.pi/2))
    bk.append(s/4)

frev=[2*np.pi*bk[i] for i in x]
temp=ft.copy()
plotter(temp,frev,w," ")

# plt.subplot(1,2,1)
# plt.stem(w,frev)
# plt.xlabel('w')
# plt.ylabel('X')
# plt.subplot(1,2,2)
# plt.stem(w,temp)
# plt.xlabel('w')
# plt.ylabel('X')
# plt.title('time reversal')
# plt.show()
```

Real parts



Imaginary parts

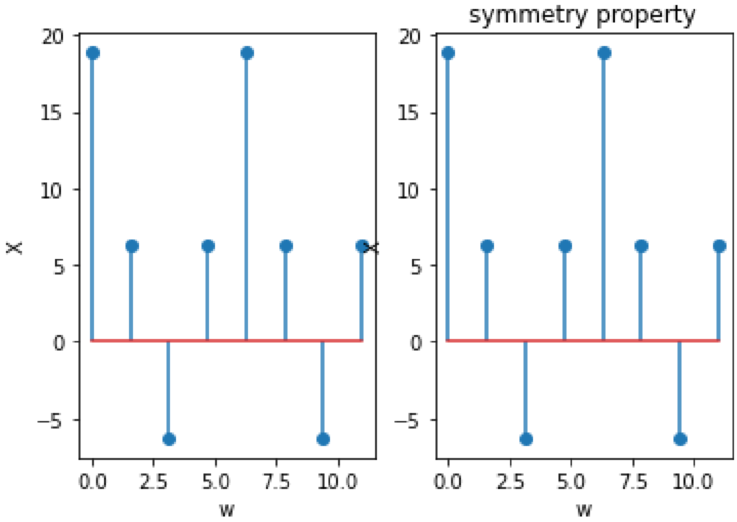


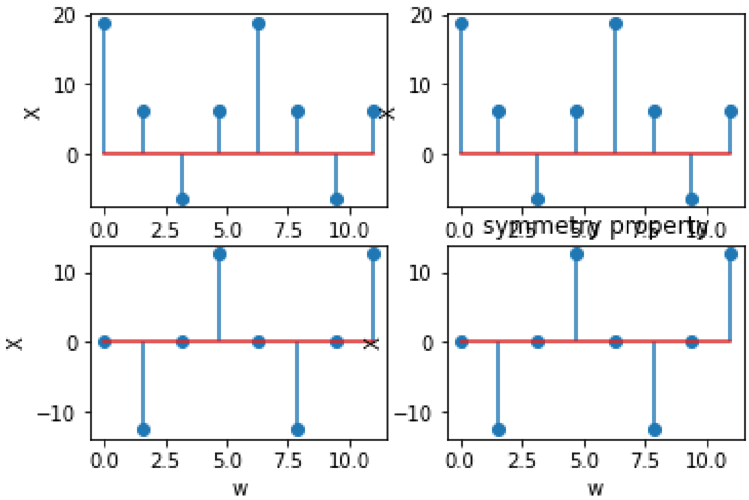
As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

Symmetry property

In [141…

```
#symmetry property
#x[n] is real signal
fsym=[np.conj(i) for i in frev]
plt.subplot(1,2,1)
plt.stem(w,ft)
plt.xlabel('w')
plt.ylabel('X')
plt.subplot(1,2,2)
plt.stem(w,fsym)
plt.xlabel('w')
plt.ylabel('X')
plt.title('symmetry property')
plt.show()

frevr=[]
frevimg=[]
for i in frev:
    frevr.append(i.real)
    frevimg.append(i.imag)

for i in x:
    frevimg[i]=-1*frevimg[i]
plt.subplot(2,2,1)
plt.stem(w,ftr)
plt.xlabel('w')
plt.ylabel('X')
plt.subplot(2,2,2)
plt.stem(w,frevr)
plt.xlabel('w')
plt.ylabel('X')
plt.subplot(2,2,3)
plt.stem(w,ftimg)
plt.xlabel('w')
plt.ylabel('X')
plt.subplot(2,2,4)
plt.stem(w,frevimg)
plt.xlabel('w')
plt.ylabel('X')
plt.title('symmetry property')
plt.show()
```





As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

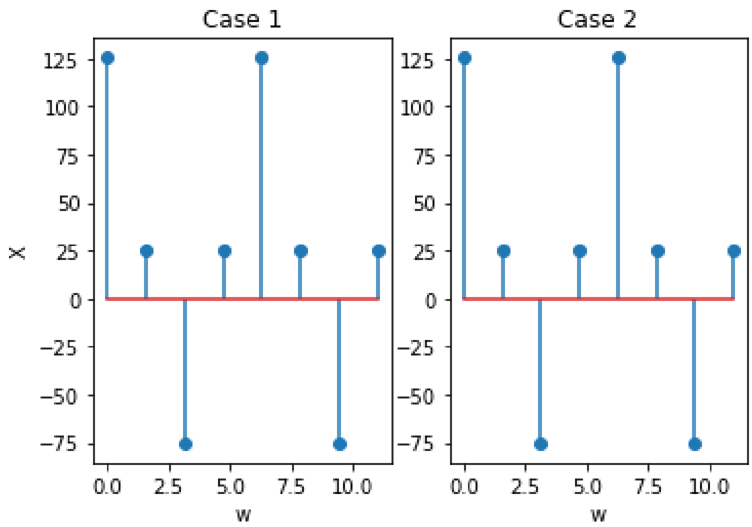
Multiplication Property

$$x[n]y[n] \qquad \frac{1}{2\pi}\int_{2\pi} X(e^{j\theta})Y(e^{j(\omega-\theta)})d\theta$$

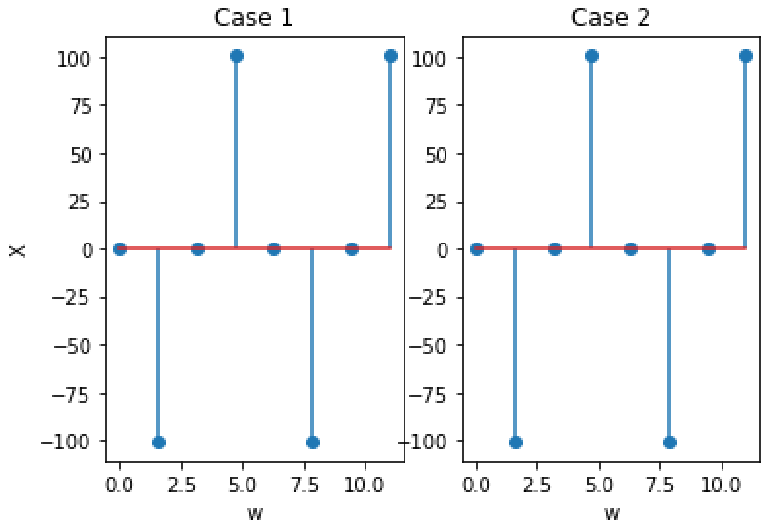
In [142…

```
y_multiplication = []
for i in range(len(y)):
    y_multiplication.append(y[i]*yl[i])
ak_multiplication = FSCoeff(y_multiplication,N)
ft_multiplication=[2*np.pi*ak_multiplication[i] for i in x]
temp_multiplication = []
def dataFinder(w0,i):
    if i-w0 < 0:
        return ft[w0]*ftl[i-w0+4]
    else:
        return ft[w0]*ftl[i-w0]
for i in x:
    data = 0
    for w0 in range(0,4):
         data += dataFinder(w0,i)
    temp_multiplication.append(data/(np.pi*2))
plotter(temp_multiplication,ft_multiplication,w,"multiplication property")
```

```
Real parts
```



Imaginary parts



As it is visible, the plots in the two cases are evidently the same. Hence, the property is verified

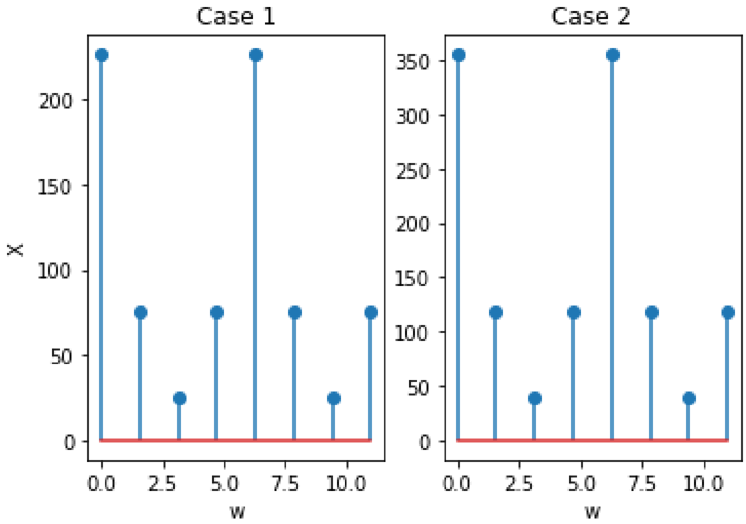
Convolution Property

$$x[n] * y[n] \qquad\qquad X(e^{j\omega})Y(e^{j\omega})$$

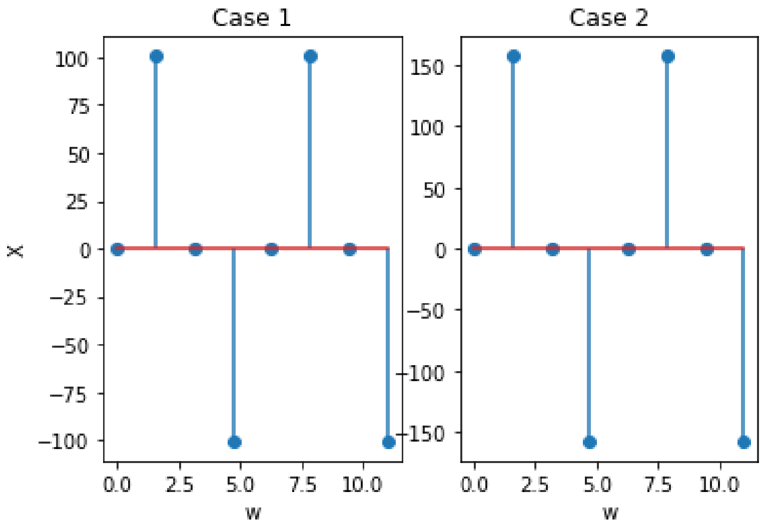
In [143...

```
y_convolution = []
def dataFinderConvolution(k,n):
     if n-k < 0:
        return y[k]*yn[n-k+4]
     return y[k]*yn[n-k]

for n in x:
    data = 0
    for k in range(N):
        data+= dataFinderConvolution(k,n)
    y_convolution.append(data)

ak_convolution = FSCoeff(y_convolution,N)
ft_convolution=[2*np.pi*ak_convolution[i] for i in x]

temp_convolution = []
for i in x:
    temp_convolution.append(ft[i]*ftn[i])
plotter(temp_convolution,ft_convolution,w,"convolution property")
```

Real parts



Imaginary parts



THE END