EXPERIMENT NO 3

NAME: Vishal Shashikant Salvi **UID: 2019230069**
CLASS: TE COMPS **BATCH: C**

Aim: To perform convolution of two discrete signals using the graphical method and tabular method.

Theory:

Linear Convolution:

The linear convolution expresses the result of passing an image signal *f* through a 2D linear convolution system *h* (or vice versa). The commutativity of the convolution is easily seen by making a substitution of variables in the double sum If *g*, *f*, and *h* satisfy the spatial convolution relationship (5.25), then their DSFT's satisfy

$G(U,V)=F(U,V)H(U,V)$,

hence convolution in the space domain corresponds directly to multiplication in the spatial frequency domain. This important property is significant both conceptually, as a simple and direct means for effecting the frequency content of an image, and computationally, since the linear convolution has such a simple expression in the frequency domain.

The 2D DSFT is the basic mathematical tool for analyzing the frequency domain content of 2D discrete-space images. However, it has a major drawback for digital image processing applications: the DSFT $F(U, V)$ of a discrete-space image $f(m, n)$ is continuous in the frequency coordinates (U, V); there are an uncountably infinite number of values to compute. As such, discrete (digital) processing or display in the frequency domain is not possible using the DSFT unless it is modified in some way. Fortunately, this is possible when the image *f* is of finite dimensions. In fact, by sampling the DSFT in the frequency domain we are able to create a computable Fourier-domain transform.

Circular Convolution:

Let us take two finite duration sequences x_1nn and x_2nn, having integer length as N. Their DFTs are X_1KK and X_2KK respectively, which is shown below −

$$X_1(K) = \sum_{n=0}^{N-1} x_1(n) e^{\frac{j2\Pi kn}{N}} \quad k = 0, 1, 2...N-1$$

$$X_2(K) = \sum_{n=0}^{N-1} x_2(n) e^{\frac{j2\Pi kn}{N}} \quad k = 0, 1, 2...N-1$$

Now, we will try to find the DFT of another sequence x_3nn, which is given as X_3K

$$X_3(K) = X_1(K) \times X_2(K)$$

By taking the IDFT of the above we get

$$x_3(n) = \frac{1}{N} \sum_{n=0}^{N-1} X_3(K) e^{\frac{j2\Pi kn}{N}}$$

After solving the above equation, finally, we get

$$x_3(n) = \sum_{m=0}^{N-1} x_1(m) x_2[((n-m))_N] \quad m = 0, 1, 2...N-1$$

Methods of Circular Convolution

Generally, there are two methods, which are adopted to perform circular convolution and they are −

- Concentric circle method,
- Matrix multiplication method.

Concentric Circle Method

Let x1(n)x1(n) and x2(n)x2(n) be two given sequences. The steps followed for circular convolution of x1(n)x1(n) and x2(n)x2(n) are

- Take two concentric circles. Plot N samples of x1(n)x1(n) on the circumference of the outer circle maintaining equal distance successive points in anti-clockwise direction.
- For plotting x2(n)x2(n), plot N samples of x2(n)x2(n) in clockwise direction on the inner circle, starting sample placed at the same point as 0^{th} sample of x1(n)x1(n)
- Multiply corresponding samples on the two circles and add them to get output.
- Rotate the inner circle anti-clockwise with one sample at a time.

Matrix Multiplication Method

Matrix method represents the two given sequence x1(n)x1(n) and x2(n)x2(n) in matrix form.

- One of the given sequences is repeated via circular shift of one sample at a time to form a N X N matrix.
- The other sequence is represented as column matrix.
- The multiplication of two matrices give the result of circular convolution.

Linear and Circular Convolution

This example shows how to establish an equivalence between linear and circular convolution.

Linear and circular convolution are fundamentally different operations. However, there are conditions under which linear and circular convolution are equivalent. Establishing this equivalence has important implications. For two vectors, x and y, the circular convolution is equal to the inverse discrete Fourier transform (DFT) of the product of the vectors' DFTs. Knowing the conditions under which linear and circular convolution are equivalent allows you to use the DFT to efficiently compute linear convolutions.

The linear convolution of an N-point vector, x, and an L-point vector, y, has length $N + L - 1$.

For the circular convolution of x and y to be equivalent, you must pad the vectors with zeros to length at least $N + L - 1$ before you take the DFT. After you invert the product of the DFTs, retain only the first $N + L - 1$ elements.

Create two vectors, x and y, and compute the linear convolution of the two vectors.

```
x = [2 1 2 1];
y = [1 2 3];
clin = conv(x,y);
```

The output has length 4+3-1.

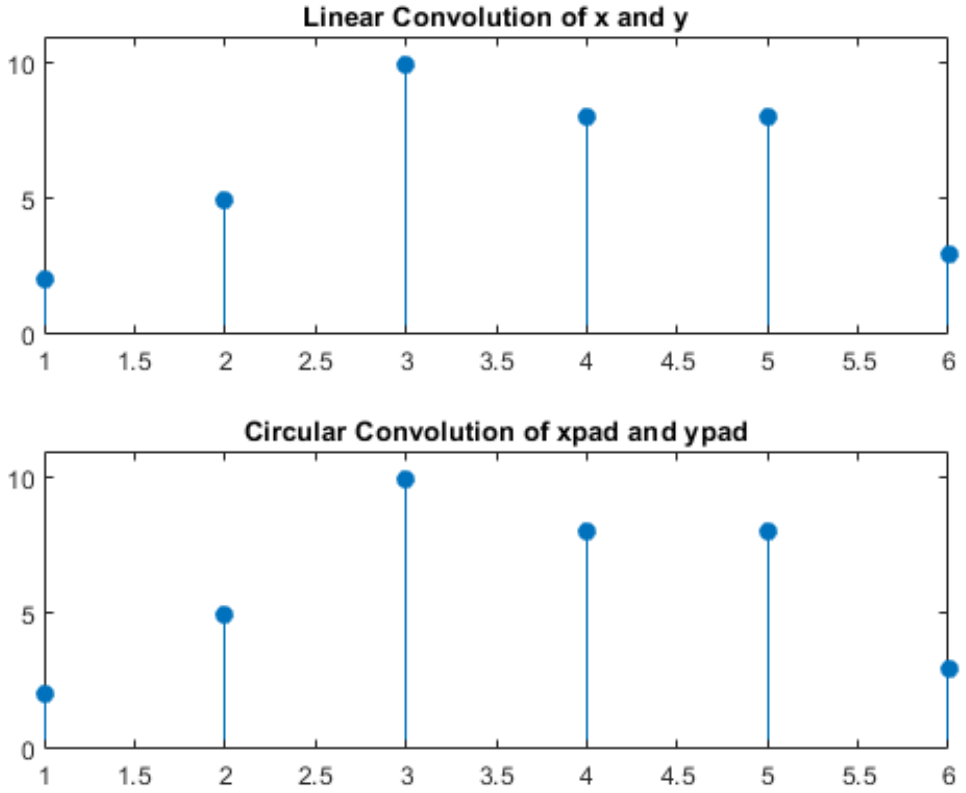
Pad both vectors with zeros to length 4+3-1. Obtain the DFT of both vectors, multiply the DFTs, and obtain the inverse DFT of the product.

```
xpad = [x zeros(1,6-length(x))];
ypad = [y zeros(1,6-length(y))];
ccirc = ifft(fft(xpad).*fft(ypad));
```

The circular convolution of the zero-padded vectors, xpad and ypad, is equivalent to the linear convolution of x and y. You retain all the elements of ccirc because the output has length 4+3-1.

Plot the output of linear convolution and the inverse of the DFT product to show the equivalence.

```
subplot(2,1,1)
stem(clin,'filled')
ylim([0 11])
title('Linear Convolution of x and y')

subplot(2,1,2)
stem(ccirc,'filled')
ylim([0 11])
title('Circular Convolution of xpad and ypad')
```



Pad the vectors to length 12 and obtain the circular convolution using the inverse DFT of the product of the DFTs. Retain only the first 4+3-1 elements to produce an equivalent result to linear convolution.

```
N = length(x)+length(y)-1;
xpad = [x zeros(1,12-length(x))];
ypad = [y zeros(1,12-length(y))];
ccirc = ifft(fft(xpad).*fft(ypad));
ccirc = ccirc(1:N);
```

The Signal Processing Toolbox™ software has a function, cconv, that returns the circular convolution of two vectors. You can obtain the linear convolution of x and y using circular convolution with the following code.

```
ccirc2 = cconv(x,y,6);
```

cconv internally uses the same DFT-based procedure illustrated in the previous example.

Linear Convolution	**Circular Convolution**
Linear convolution is a mathematical operation done to calculate the output of any Linear-Time Invariant (LTI)	Circular convolution is essentially the same process as linear convolution. Just like linear convolution, it involves the operation of folding a sequence, shifting

system given its input and impulse response.	it, multiplying it with another sequence, and summing the resulting products (We'll see what this means in a minute). However, in circular convolution, the signals are all periodic. Thus the shifting can be thought of as actually being a rotation. Since the values keep repeating because of the periodicity. Hence, it is known as circular convolution.
It is applicable for both continuous and discrete-time signals.	Circular convolution is also applicable for both continuous and discrete-time signals.
We can represent Linear Convolution as $y(n)=x(n)*h(n)$ Here, y(n) is the output (also known as convolution sum). x(n) is the input signal, and h(n) is the impulse response of the LTI system.	We can represent Circular Convolution as $y(n)=x(n)\oplus h(n)$ Here y(n) is a periodic output, x(n) is a periodic input, and h(n) is the periodic impulse response of the LTI system.
In linear convolution, both the sequences (input and impulse response) may or may not be of equal sizes. That is, they may or may not have the same number of samples. Thus the output, too, may or may not have the same number of samples as any of the inputs.	In circular convolution, both the sequences (input and impulse response) must be of equal sizes. They must have the same number of samples. Thus the output of a circular convolution has the same number of samples as the two inputs.
For example, consider the following signals: x(n): [1,2,3] h(n): [1,2,3,4,5] As you can see, the number of samples in the input and Impulse response signals is not the same. Still, linear convolution is possible. Here's how you calculate the number of samples in the output of linear convolution. L = M + N – 1	For the given example, circular convolution is possible only after modifying the signals via a method known as *zero padding.* In zero padding, zeroes are appended to the sequence that has a lesser size to make the sizes of the two sequences equal. Thus, for the given sequence, after zero-padding: x(n) = [1,2,3,0,0] Now both x(n) and h(n) have the same lengths. So circular convolution can take place. And the output of the

Where M is the number of samples in x(n). N is the number of samples in h(n). For the above example, the output will have (3+5-1) = 7 samples.	circular convolution will have the same number of samples. i.e., 5.
Graphically, when we perform linear convolution, there is a linear shift taking place. Check out the formula for a convolution. $\sum_{-\infty}^{\infty} x(k)h(n-k)$ There is a folding of the IR sequence, shifting it by n, multiplying it with another sequence (input), and summing the resulting products.	Graphically, when we perform circular convolution, there is a circular shift taking place. Alternatively, we can call it rotation.
It is possible to find the response of a filter using linear convolution.	It is possible to find the response of a filter using circular convolution after zero padding. I fact, we will be doing this in overlap-save and overlap-add methods — two essential topics in our digital signal processing course.
Linear convolution may or may not result in a periodic output signal.	The output of a circular convolution is always periodic, and its period is specified by the periods of one of its inputs.

Linear Convolution:

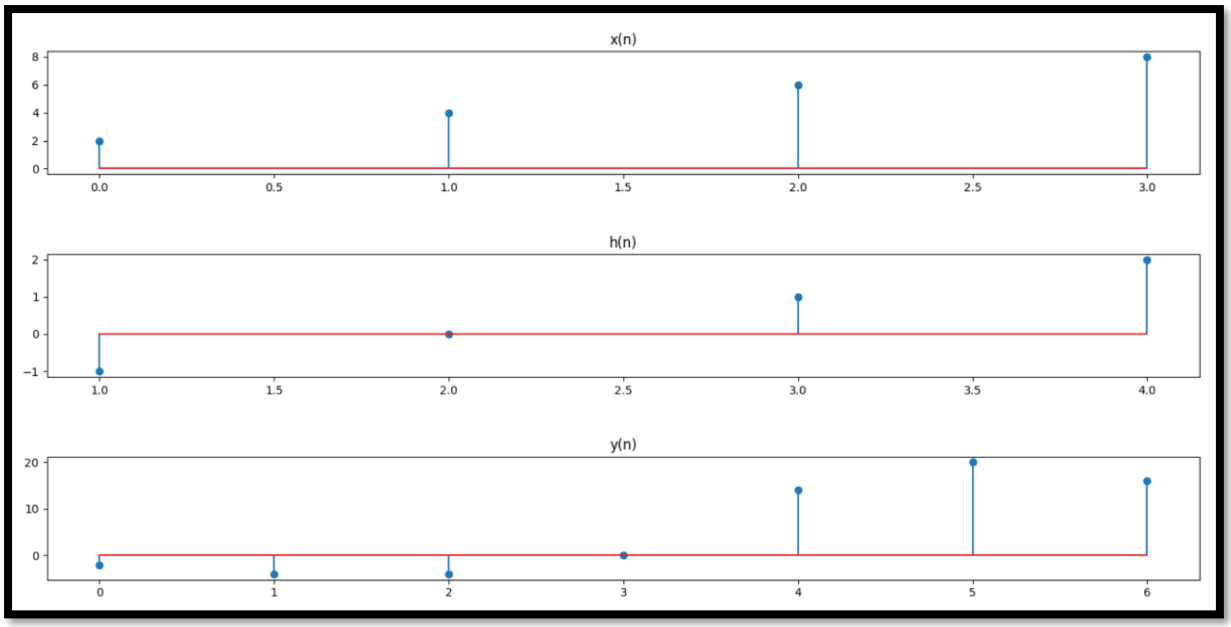
Code:

```
import numpy as np
from collections import defaultdict
from prettytable import PrettyTable
import matplotlib.pyplot as plt

x = PrettyTable()
xn = list(map(int, input("Enter value x(n): ").split()))
x_o = int(input("Enter value x start:"))
hn = list(map(int, input("Enter value h(n):").split()))
h_o = int(input("Enter value h start:"))

n = len(xn)
m = len(hn)
leng = n + m
di = defaultdict(int)
first_row = ['hn' + '\\' + 'xn'] + list(map(str, xn))
x.field_names = first_row

for i in range(m):
    row = [f'{hn[i]}']
```

```
    for j in range(n):
        val = xn[j] * hn[i]
        row.append(f'{val}')
        di[i + j] += (val)
    x.add_row(row)
ans = []

print("y(n)= ", end="")
for i in range(n + m - 1):
    ans.append(di[i])
print(ans)
print(x)

xaxis = np.array(range(leng - 1))
yaxis = np.array(ans)
xn1 = np.array(xn)
xn2 = np.array(range(x_o, len(xn)))

#plotting
ax = plt.subplot(3, 1, 1)
plt.stem(xn2, xn1, use_line_collection=True)
plt.title('x(n)')
hn1 = np.array(hn)
hn2 = np.array(range(h_o, (len(hn) + h_o)))

ax = plt.subplot(3, 1, 2)
plt.stem(hn2, hn1, use_line_collection=True)
plt.title('h(n)')

ax = plt.subplot(3, 1, 3)
plt.stem(xaxis, yaxis, use_line_collection=True)
plt.title('y(n)')
plt.show()
```

Output:

```
C:\Users\Vishal\AppData\Local\Programs\Python\Python38\python.exe C:/Python/DSPExp3.py
Enter value x(n): 2 4 6 8
Enter value x start:0
Enter value h(n):-1 0 1 2
Enter value h start:1
y(n)= [-2, -4, -4, 0, 14, 20, 16]
+-------+----+----+----+----+
| hn\xn | 2  | 4  | 6  | 8  |
+-------+----+----+----+----+
|   -1  | -2 | -4 | -6 | -8 |
|   0   | 0  | 0  | 0  | 0  |
|   1   | 2  | 4  | 6  | 8  |
|   2   | 4  | 8  | 12 | 16 |
+-------+----+----+----+----+
```



Circular Convolution:

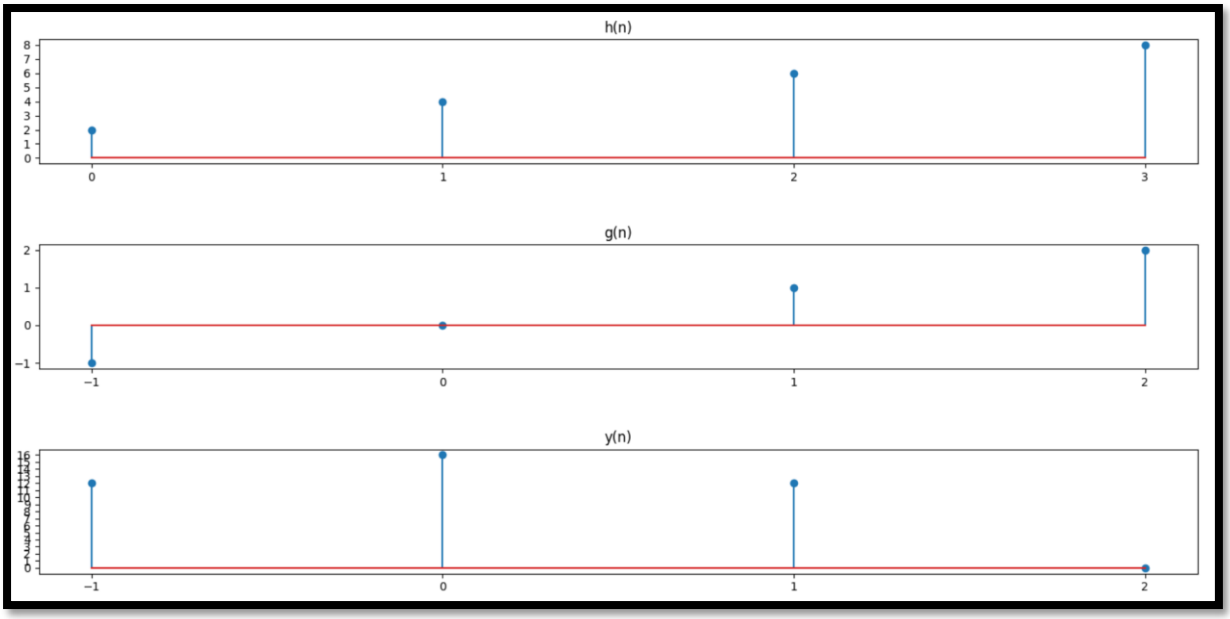
Code:

```
import matplotlib.pyplot as plt
def circular_convolution(h, g, h_start, g_start):
    if len(h) > len(g):
        h, g = g, h
    h += [0] * (len(g) - len(h))

    mat = [list(reversed(h[:i + 1])) + list(reversed(h[i + 1:])) for i in
range(len(h))]
    y = [0] * (len(h))
    y_start = h_start + g_start

    for i, row in enumerate(mat):
        for j, num in enumerate(row):
            y[i] += num * g[j]

    y_start = h_start + g_start
    y_x = list(range(-y_start, 1, 1)) + list(range(1, len(y) - y_start))
    h_x = list(range(-h_start, 1, 1)) + list(range(1, len(h) - h_start))
    g_x = list(range(-g_start, 1, 1)) + list(range(1, len(g) - g_start))
    #plotting
    ax = plt.subplot(3, 1, 1)
    plt.stem(h_x, h)
    start, end = ax.get_ylim()
    plt.yticks(range(int(start), int(end + 1)))
    plt.xticks(h_x)
    plt.title("h(n)")

    ax = plt.subplot(3, 1, 2)
    plt.stem(g_x, g)
    start, end = ax.get_ylim()
    plt.yticks(range(int(start), int(end + 1)))
    plt.xticks(g_x)
    plt.title("g(n)")

    ax = plt.subplot(3, 1, 3)
```

```
    plt.stem(y_x, y)
    start, end = ax.get_ylim()
    plt.yticks(range(int(start), int(end + 1)))
    plt.xticks(y_x)
    plt.title("y(n) = h(n).g(n)")

    print("Matrix is :")
    print((mat,))
    print("y(n) :", y)

    plt.tight_layout()
    plt.show()

xn = list(map(int, input("Enter value of x(n): ").split()))
x_o = int(input("Enter value of x start:"))
hn = list(map(int, input("Enter value of h(n):").split()))
h_o = int(input("Enter value of h start:"))
circular_convolution(xn, hn, x_o, h_o)
```

Output:

```
C:\Users\Vishal\AppData\Local\Programs\Python\Python38\python.exe C:/Python/matrix_dsp.py
Enter value of g(n): 2 4 6 8
Enter value of g start:0
Enter value of h(n):-1 0 1 2
Enter value of h start:1
Matrix is :
([[2, 8, 6, 4], [4, 2, 8, 6], [6, 4, 2, 8], [8, 6, 4, 2]],)
y(n) : [12, 16, 12, 0]
```



Conclusion:

Thus, we studied and implemented linear and circular convolution. The operation of discrete time circular convolution is defined such that it performs this function for finite length and periodic discrete time signals.

In each case, the output of the system is the convolution or circular convolution of the input signal with the unit impulse response.

References:

1) https://www.sciencedirect.com/topics/engineering/linear-convolution
2) https://technobyte.org/difference-between-linear-circular-convolution/
3) https://www.mathworks.com/help/signal/ug/linear-and-circular-convolution.html