ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 354E

SIGNAL AND SYSTEMS FOR COMPUTER ENGINEERING

HOMEWORK REPORT

HOMEWORK NO	:	1
DUE DATE	:	18.03.2019

Esin Ece AYDIN - 150160151

SPRING 2019

Contents

FRONT COVER

CONTENTS

1 INTRODUCTION 1

2 REQUIREMENTS 1

2.1 First Function - functionA.py 1
2.2 Second Function - functionB.py 4
2.3 Third Function - functionC.py 7
2.4 Fourth Function - functionD.py 9

3 INTERPRETATION OF THE RESULTS 10

4 CONCLUSION 10

1 INTRODUCTION

In this homework, I wrote 4 Python codes that perform different tasks such as visualizing the non-periodic discrete time signals and calculating convolution of them. These Python codes which are named as functionA.py, functionB.py, functionC.py, functionD.py explained in detail in the Section 2

2 REQUIREMENTS

According two given amplitude values with respect to sampling numbers within the predefined boundaries, I visualized two non-periodic discrete time signals which to be named as x[n] and y[n]. I also calculated convolution of these two signals. I used the given example discrete signals inputs for demonstration of all my functions.

2.1 First Function - functionA.py

In the first function, only the given sample values were plotted. The blue line with blue dot belongs to the signal x[n], and the yellow line with yellow dot belongs to the signal y[n] as shown in the Figures 1, 2, 3, and 4.

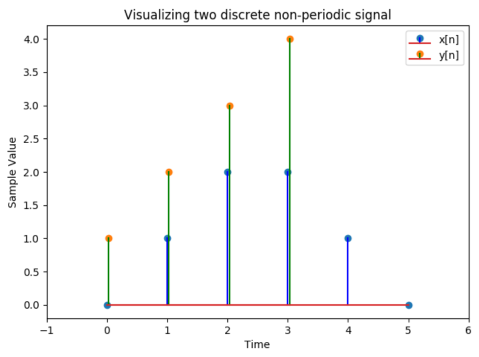


Figure 1: Graph A.1

- x[0] = 0
- x[1] = 1
- x[2] = 2
- x[3] = 2
- x[4] = 1
- x[5] = 0

- y[0] = 1
- y[1] = 2
- y[2] = 3
- y[3] = 4

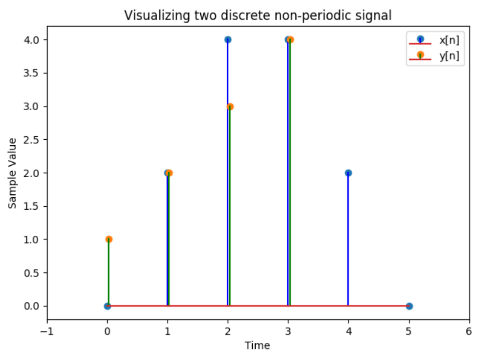


Figure 2: Graph A.2

- x[0] = 0
- x[1] = 2
- x[2] = 4
- x[3] = 4
- x[4] = 2
- x[5] = 0

- y[0] = 1
- y[1] = 2
- y[2] = 3
- y[3] = 4

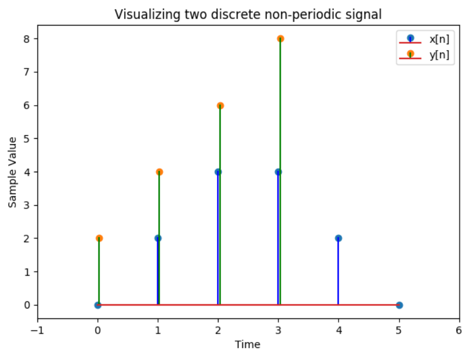


Figure 3: Graph A.3

- x[0] = 0
- x[1] = 2
- x[2] = 4
- x[3] = 4
- x[4] = 2
- x[5] = 0

- y[0] = 2
- y[1] = 4
- y[2] = 6
- y[3] = 8

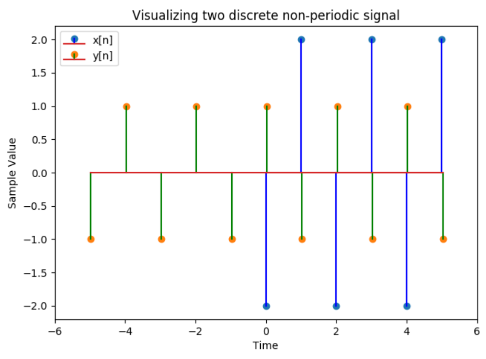


Figure 4: Graph A.4

- x[0] = -2
- x[1] = 2
- x[2] = -2
- x[3] = 2
- x[4] = -2
- x[5] = 2

- y[-5] = -1
- y[-4] = 1
- y[-3] = -1
- y[-2] = 1
- y[-1] = -1
- y[0] = 1

- y[1] = -1
- y[2] = 1
- y[3] = -1
- y[4] = 1
- y[5] = -1

2.2 Second Function - functionB.py

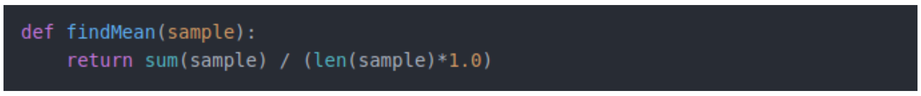
The second function's purposes are calculating the standard normalized form of signals and plotting them into the graph. The calculation procedures were done according to these equations:

$$\overline{x} = \frac{(x[0] + x[1] + x[2] + x[3] + ... + x[n])}{(n+1)} \tag{1}$$

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{2}$$

$$z = \frac{x - \overline{x}}{\sigma} \tag{3}$$

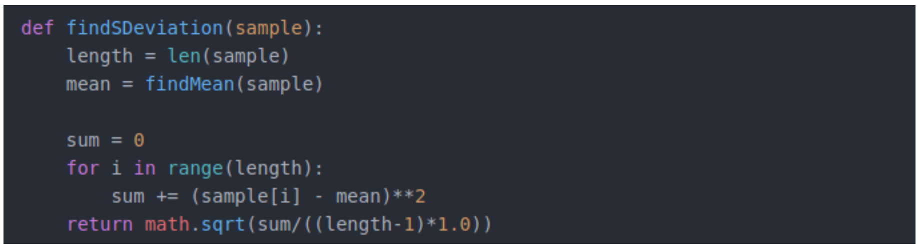
The mean of sample is calculated by 'findMean' function as shown in the Figure 5.



```
def findMean(sample):
    return sum(sample) / (len(sample)*1.0)
```

Figure 5: Function to calculate mean

Afterwards, by using standard deviation formula in (2), I declared a function which is named as 'findSDeviation' as shown in the Figure 6.

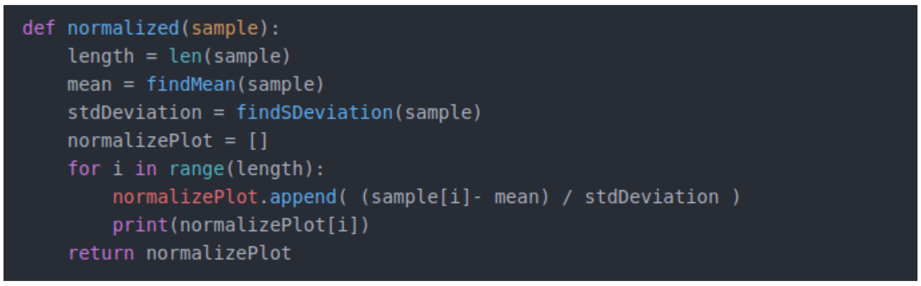


```
def findSDeviation(sample):
    length = len(sample)
    mean = findMean(sample)

    sum = 0
    for i in range(length):
        sum += (sample[i] - mean)**2
    return math.sqrt(sum/((length-1)*1.0))
```

Figure 6: Function to calculate standard deviation

The standard normalization of sample values are calculated according to the equation in (3) with the function with a name 'normalized' which is shown in Figure 7.



```
def normalized(sample):
    length = len(sample)
    mean = findMean(sample)
    stdDeviation = findSDeviation(sample)
    normalizePlot = []
    for i in range(length):
        normalizePlot.append( (sample[i]- mean) / stdDeviation )
        print(normalizePlot[i])
    return normalizePlot
```

Figure 7: Function to calculate standard normalization form of a signal

This function has only one argument which its type is list. Furthermore, this argument is a signal that will be normalized. Firstly, the function wants to know 3 information about signal. These are the length of signal, its mean, and its standard deviation. In order to that, necessary function calls is made and return values assigned to variables. After that, it defines an empty list which will be inserted the values obtained by calculation process.

The observed results according to given example discrete signals inputs are listed below:

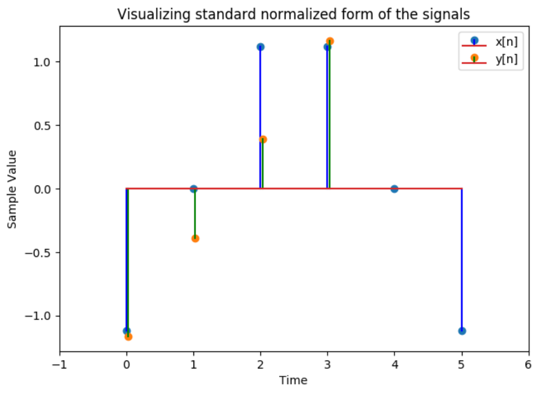


Figure 8: Graph B.1

- x[0] = -1.118033988749895
- x[1] = 0.0
- x[2] = 1.118033988749895
- x[3] = 1.118033988749895
- x[4] = 0.0
- x[5] = -1.118033988749895

- y[0] = -1.161895003862225
- y[1] = -0.3872983346207417
- y[2] = 0.3872983346207417
- y[3] = 1.161895003862225

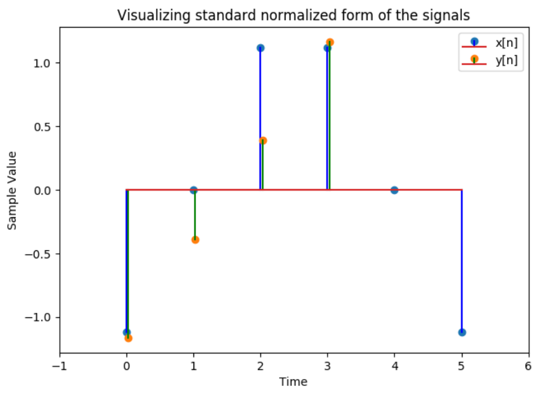


Figure 9: Graph B.2

- x[0] = -1.118033988749895
- x[1] = 0.0
- x[2] = 1.118033988749895
- x[3] = 1.118033988749895
- x[4] = 0.0
- x[5] = -1.118033988749895

- y[0] = -1.161895003862225
- y[1] = -0.3872983346207417
- y[2] = 0.3872983346207417
- y[3] = 1.161895003862225

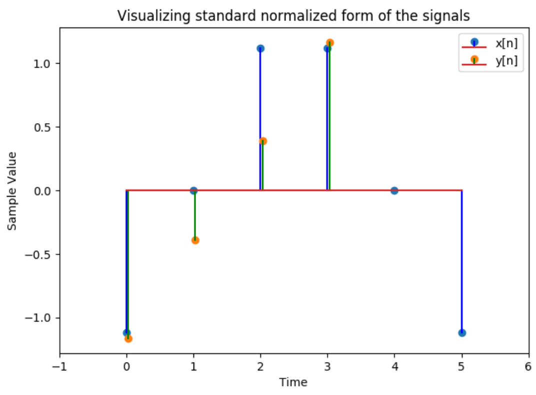


Figure 10: Graph B.3

- x[0] = -1.118033988749895
- x[1] = 0.0
- x[2] = 1.118033988749895
- x[3] = 1.118033988749895
- x[4] = 0.0
- x[5] = -1.118033988749895

- y[0] = -1.161895003862225
- y[1] = -0.3872983346207417
- y[2] = 0.3872983346207417
- y[3] = 1.161895003862225

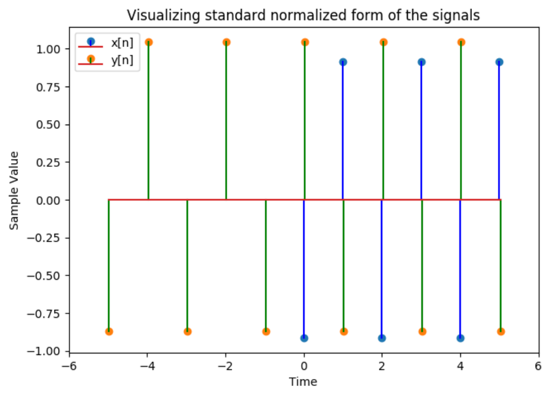


Figure 11: Graph B.4

- x[0] = -0.9128709291752769
- x[1] = 0.9128709291752769
- x[2] = -0.9128709291752769
- x[3] = 0.9128709291752769
- x[4] = -0.9128709291752769
- x[5] = 0.9128709291752769

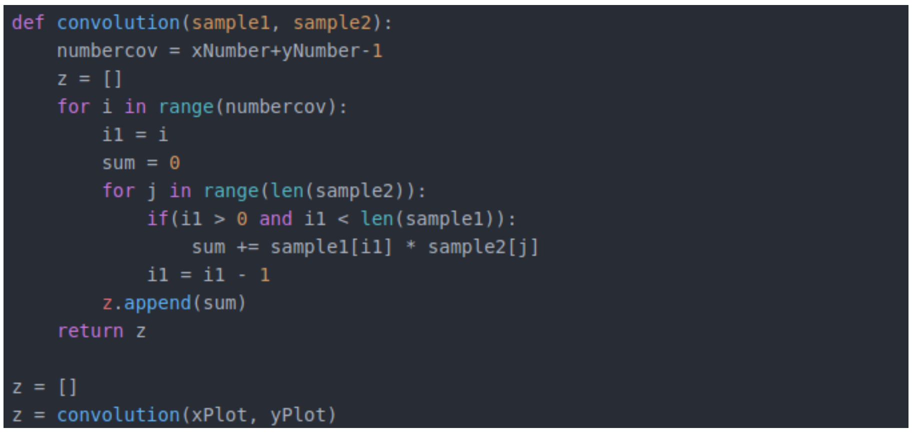
- y[-5] = -0.8703882797784893
- y[-4] = 1.0444659357341872
- y[-3] = -0.8703882797784893
- y[-2] = 1.0444659357341872
- y[-1] = -0.8703882797784893
- y[0] = 1.0444659357341872
- y[1] = -0.8703882797784893
- y[2] = 1.0444659357341872
- y[3] = -0.8703882797784893
- y[4] = 1.0444659357341872
- y[5] = -0.8703882797784893

2.3 Third Function - functionC.py

In this function, I calculated the convolution of the given signals. However, given two signals have different number of sample values. To overcome this challenge, I consider that knowledge:

- When performing the convolution of 2 signals is calculated, x and y , with lengths N_x and N_y , the resulting signal is length $N = N_x + N_y - 1$

In Figure 12, the convolution function which is written by me is shown. The variable 'numbercov' holds the length of the result signal. The result signal is named as z[n].

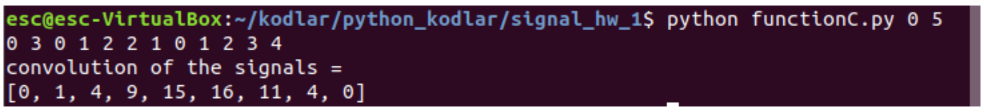


```python
def convolution(sample1, sample2):
    numbercov = xNumber+yNumber-1
    z = []
    for i in range(numbercov):
        i1 = i
        sum = 0
        for j in range(len(sample2)):
            if(i1 > 0 and i1 < len(sample1)):
                sum += sample1[i1] * sample2[j]
            i1 = i1 - 1
        z.append(sum)
    return z

z = []
z = convolution(xPlot, yPlot)
```

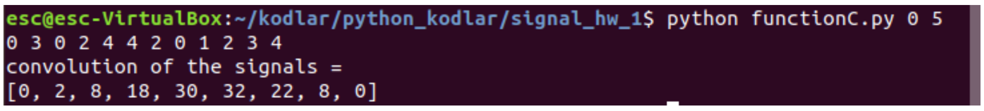
Figure 12: Function to calculate convolution

The results are listed in order according to given example discrete signals inputs in Figure 13, Figure 14, Figure 15, and Figure 16



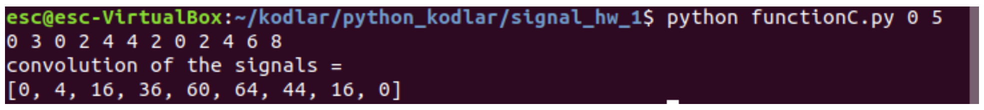
```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionC.py 0 5
0 3 0 1 2 2 1 0 1 2 3 4
convolution of the signals =
[0, 1, 4, 9, 15, 16, 11, 4, 0]
```

Figure 13: Result of C.1



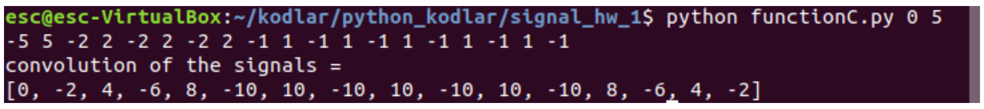
```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionC.py 0 5
0 3 0 2 4 4 2 0 1 2 3 4
convolution of the signals =
[0, 2, 8, 18, 30, 32, 22, 8, 0]
```

Figure 14: Result of C.2



```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionC.py 0 5
0 3 0 2 4 4 2 0 2 4 6 8
convolution of the signals =
[0, 4, 16, 36, 60, 64, 44, 16, 0]
```

Figure 15: Result of C.3



```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionC.py 0 5
-5 5 -2 2 -2 2 -2 2 -1 1 -1 1 -1 1 -1 1 -1 1 -1
convolution of the signals =
[0, -2, 4, -6, 8, -10, 10, -10, 10, -10, 10, -10, 8, -6, 4, -2]
```

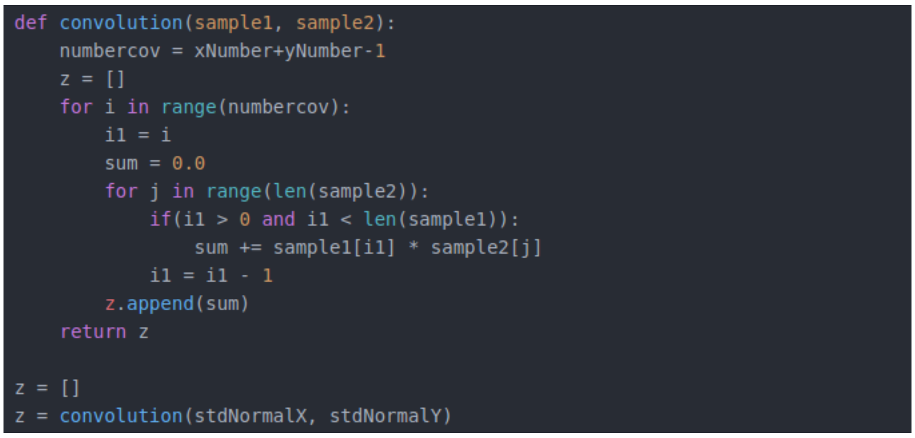
Figure 16: Result of C.4

2.4 Fourth Function - functionD.py

In the last function, I calculated convolution of standard normalized forms of the signals which is obtained in functionB.py.

In Figure 17, the convolution function is shown. Its differences from functionC.py's 'convolution' function are those:

- The 'sum' variable's initial value is 0.0 which shows that it is a float variable.
- When we call this function, the parameters will be the standard normalized form of x[n] and y[n] signals which are named as 'stdNormalX', and 'stdNormalY'.

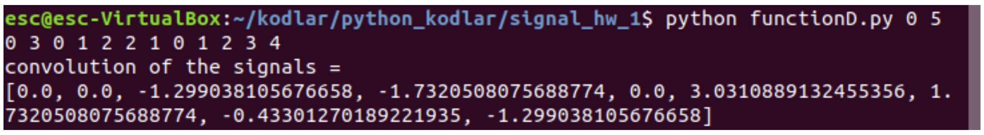


```python
def convolution(sample1, sample2):
    numbercov = xNumber+yNumber-1
    z = []
    for i in range(numbercov):
        i1 = i
        sum = 0.0
        for j in range(len(sample2)):
            if(i1 > 0 and i1 < len(sample1)):
                sum += sample1[i1] * sample2[j]
            i1 = i1 - 1
        z.append(sum)
    return z

z = []
z = convolution(stdNormalX, stdNormalY)
```

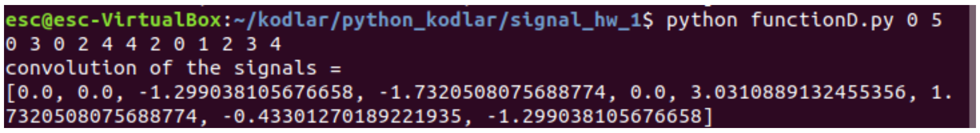
Figure 17: Function to calculate convolution

In the below figures, my results are shown.



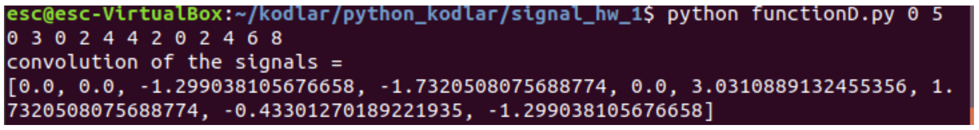
```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionD.py 0 5
0 3 0 1 2 2 1 0 1 2 3 4
convolution of the signals =
[0.0, 0.0, -1.299038105676658, -1.7320508075688774, 0.0, 3.0310889132455356, 1.
7320508075688774, -0.43301270189221935, -1.299038105676658]
```

Figure 18: Result of D.1



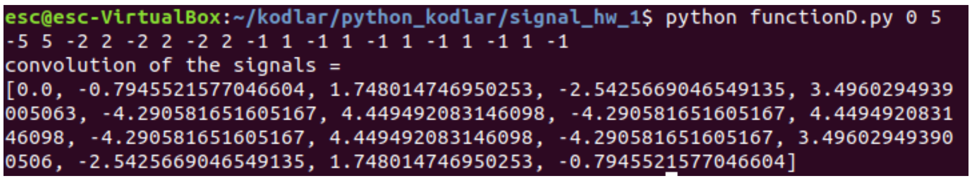
```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionD.py 0 5
0 3 0 2 4 4 2 0 1 2 3 4
convolution of the signals =
[0.0, 0.0, -1.299038105676658, -1.7320508075688774, 0.0, 3.0310889132455356, 1.
7320508075688774, -0.43301270189221935, -1.299038105676658]
```

Figure 19: Result of D.2



```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionD.py 0 5
0 3 0 2 4 4 2 0 2 4 6 8
convolution of the signals =
[0.0, 0.0, -1.299038105676658, -1.7320508075688774, 0.0, 3.0310889132455356, 1.
7320508075688774, -0.43301270189221935, -1.299038105676658]
```

Figure 20: Result of D.3



```
esc@esc-VirtualBox:~/kodlar/python_kodlar/signal_hw_1$ python functionD.py 0 5
-5 5 -2 2 -2 2 -2 2 -1 1 -1 1 -1 1 -1 1 -1 1 -1
convolution of the signals =
[0.0, -0.7945521577046604, 1.748014746950253, -2.5425669046549135, 3.4960294939
005063, -4.290581651605167, 4.449492083146098, -4.290581651605167, 4.4494920831
46098, -4.290581651605167, 4.449492083146098, -4.290581651605167, 3.49602949390
0506, -2.5425669046549135, 1.748014746950253, -0.7945521577046604]
```

Figure 21: Result of D.4

3 INTERPRETATION OF THE RESULTS

I observed that, if there is a ratio between the values of the signals, the result obtained from convolution of each signals with another signal has the same ratio. To be more descriptive; for example, if a signal $x_1[n]$ is given as [0, 1, 2, 2, 1, 0] and the other signal $x_1[n]$ is given as [0, 2, 4, 4, 2, 0]. Lastly $y[n]$ is given as [1, 2, 3, 4] . Convolution of the signals $x_1[n]$ and y[n] is calculated as [0, 1, 4, 9, 18, 16, 11, 4, 0] and the convolution of $x_2[n]$ and y[n] is calculated as [0, 2, 8, 18, 36, 32, 22, 8, 0]. As seen, the results are double of each other, because $x_1[n]$ equals the twice of $x_2[n]$.

4 CONCLUSION

By given example discrete signals inputs I obtained 4 outputs from each program, I have 16 outputs in total. The first two program plotted the necessary values to the graph. The last two, calculated the convolution of two signals.
Python was a programming language I didn't use much before. With this homework I had chance to do practice with Python. I have learned a few Python amenities compared to the C/C++ languages. In the simplest, I didn't know how to get input as an argument.