DEPARTMENT OF PHYSICS
UNIVERSITY OF COLOMBO

PH 3034- Digital Image Processing – I
2021

Convolution

Name: Umeshika Dissanayaka

Index Number: s14320

Date: 03/10/2021

TABLE OF CONTENTS

1 INTRODUCTION 1

2 METHODOLOGY 2

2.1 Exercise 1-1D convolution for discrete functions 2

2.1.1 Part A 2

2.1.2 Part B 4

2.2 Exercise 2- 1D convolution for continuous functions upon vectoring them 7

2.2.1 Part A 7

2.2.2 Part B 10

2.2.3 Part C 13

2.3 Exercise 3- Conversion of 1D discrete convolution function to the 2D scenario 17

2.3.1 Part A 17

2.3.2 Part B 22

3 REFERENCES 24

TABLE OF FIGURES

Figure 2.1.1-1 The graph of f-signal, g-kernel and Convolution for discrete function……..4
Figure 2.1.1-2 The graph of f-signal, different g-kernel and Convolution for discrete function……6
Figure 2.1.1-3 The graph of f-signal, g-kernel and Convolution for continuous function……..10

Figure 2.1.1-4 The graph of f-signal, g-kernel when g(x)=x and Convolution for continuous function……13
Figure 2.1.1-5- The graph of f-signal, g-kernel when g(x)=e^(1-x) and Convolution for continuous function……16
Figure 2.1.1-6 The image of signal f which is taken by using image function -image(f)…… ……..17
Figure 2.1.1-7 2.1.1-8 The image of signal f which is taken by using the command of plt.imshow(f,cmap='gray')…… …17
Figure 2.1.1-9- The image of signal f which is taken by using image function -image(f)…… …… …20
Figure 2.3.1 5 The image of kernel g which is taken by using image function -image(g,1)…….20
Figure 2.1.1-10 The image of reversed kernel which is taken by using image function -image(g_r,1)……20
Figure 2.1.1-11 The image of 2D convolution function which is taken by using image function -image(cov1)……,,….20
Figure 2.1.1-12 The image for signal in part B……21
Figure 2.1.1-13 The image for kernel part B……21
Figure 2.1.1-14 The image of signal f which is taken by using cv2.imshow("f-signal",img)…….22
Figure 2.1.1-15 The image of 2D convolution function which is taken by using cv2.imshow("2D-convolution",conv1)……..22

1 INTRODUCTION

In mathematics, the element of x is in the component of a set X uniquely is called, the domain of the function. In addition, the function is exceptionally addressed by the arrangement, everything being equal (x, f (x)), called the graph. In numerous fields of probability, physics, statistics, differential equations, spectroscopy, signal processing and image processing, computer vision and engineering are used functions for their various purposes and its applications. In these fields, are needed to operate new functions using given two functions. At that time, the mathematical operation of a convolution comes to play. It is implemented as the basis of the result of a new function after an integrated multiplier of both input functions which one input function is called signal and the other input function is reversed and called the kernel. The convolution can be categorized using dimensions which are one dimensional Convolution and two-dimensional convolutions. And also, one dimensional Convolution can be categorized using data types of function which are Convolutions for continuous functions and Convolutions for desecrate functions. This report is mainly based on one dimensional Convolution for continuous functions, desecrate function and two-dimensional scenarios. Furthermore, the main objective of this practical was to gain brief knowledge about convolution operation, Implementation and visualization of convolution for one-dimensional functions for discrete functions, applying and visualization discrete convolution function for continuous functions upon vectoring them, defining new algorithm to show images properly, implementation of the one-dimensional discrete convolution function to the two-dimensional scenario such as Applying the convolution operator to images.

Currently, convolution is designed and implemented using computational numerical analysis such as finite impulse response filters in signal processing and image processing using algorithms. Therefore, the main purpose of this practical was to design, Implement and visualize the convolution for one- and two-dimensional functions for discrete and continuous functions by using the high-level programming language of Python 03. Python 03 is an object-oriented scripting language, and it has been easy to understand. Currently, python 03 is used in a lot of computational areas because it is utilized familiar English syntaxes which are easy to understand and learn. Not only that it is free to open software, and everyone can download it freely. And also, python has standard libraries which can import and get the output easily. There are the vital benefits of learning python that has allowed debugging of snippets of code and interactive testing. There are various fundamental packages are used for numerical calculation and graphical representation such as Opencv, math, NumPy and matplotlib.

2 METHODOLOGY

2.1 Exercise 1-1D convolution for discrete functions

2.1.1 Part A

consider the following functions f and g

$$x \in [1,20], x \in \mathbb{N}, \qquad f(x) = \begin{cases} 1 \; when \; x = 6 \\ 0 \; otherwise \end{cases}$$

$$g(x) = \begin{cases} 4 \; when \; x = 0 \\ 1 \; when \; x = 1 \\ 7 \; when \; x = 2 \end{cases}$$

- Firstly, the numerical background of the problem was identified and implemented the mathematical functions and steps were before creating the python code. Therefore this is discretized convolution operation which can be given as

$$(\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{X}) = \int_{-\infty}^{+\infty} \boldsymbol{f}(\boldsymbol{X}).\boldsymbol{g}(\boldsymbol{x} - \boldsymbol{X})\boldsymbol{dX} \Rightarrow (\boldsymbol{f} * \boldsymbol{g})(\boldsymbol{n}) = \sum_{\boldsymbol{m}=-\infty}^{\infty} \boldsymbol{f}(\boldsymbol{m})\boldsymbol{g}(\boldsymbol{n} - \boldsymbol{m})$$

- Firstly the library of “NumPy”, “matplotlib” and “datetime” was imported and created the variable that was named of “start” to get the starting time when the code was run.
- The g(x) and f(x) were Implemented as vectors in python with its limitations. In here, f(x) was called signal.
- The g vector was the kernel and reversed kernel was taken reversely by using g(x)
- The convolution of f and g functions were Implemented by using discretized convolution operation.
- Then the code executing time is calculated using the difference of the start time and end time.
- 60.832 ms was taken to execute the code.
- Finally, the graph of f-signal, g-kernel and convolution was plotted in the same figure by using the subplot function.

In this code, the “plotgraph “function was defined to plot the graphs in the same figure. Therefore, the f signal graph was plot using “plt. subplot” command and “plt.xlim” that was used for getting the limited x-axis. Likewise, again g kernel and convolution were subplots. And each graph was labelled their x-axis, y-axis and title.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  1 17:27:49 2021

@author: Umeshika
"""
import numpy as np #import libraries
import matplotlib.pyplot as plt
from datetime import datetime
#get the starting time when the code is run
starttime=datetime.now()

def plotgraphs(f,g,h): # define the plot function
    plt.subplots_adjust(hspace=1.30)# set the gap between each graph
    plt.subplot(3,1,1)#first plot of the three graph
    plt.xlim(0.0,20.0) #set x axis limitation
    plt.stem(x,f) #plot signal
    plt.title("The graph of f -signal")#set the title
    plt.xlabel('x', fontsize = 9)#x axis label
    plt.ylabel('f (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,2)#second plot of the three graph
    plt.xlim(0.0,20.0)#set x axis limitation
    plt.stem([0,1,2],g)#plot kernel
    plt.title("The graph of g -kernel")#set the title
    plt.xlabel('x', fontsize = 9)#x axis label
    plt.ylabel('g (x)', fontsize = 9)#y axis label

    plt.xticks(x)
    plt.subplot(3,1,3)#Third plot of the three graph
    plt.xlim(0.0,20.0) #set x axis limitation
    plt.stem(x,h)#plot convolution
    plt.title("The graph of Convolution")#set the title
    plt.xlabel('x', fontsize = 9)#x axis label
    plt.ylabel('( f *g ) (X)', fontsize = 9)#y axis la

#x vector was created in range 1 to 20 with stepsize 1
x=np.arange(1,21,1)
f=np.zeros(20)#set zeros for f signal values
f[5]=1#create f signal
g= np.array([4,1,7])#create g kernel vector
g_r = g[::-1]#get the reverse g kernel
cov1=f*0;#set zeros for convolution vector

#%%get dot product of signal and kernel to get convolution
for i in range(f.size-g_r.size):
    cov1[i] =np.dot(f[i:i+g_r.size],g_r)

plotgraphs(f,g,cov1)#plot the graph
"""The code executing time is calculated using
 difference of the start time and end time"""
print (datetime.now()-starttime)
```

In this code convolution function was generated using for loop and without using convolution python library. Therefore, the convolution was calculated for given f signal and g kernel using dot product of f signal and g reversed kernel in range of 0 to (f signal size- g reserved kernel)

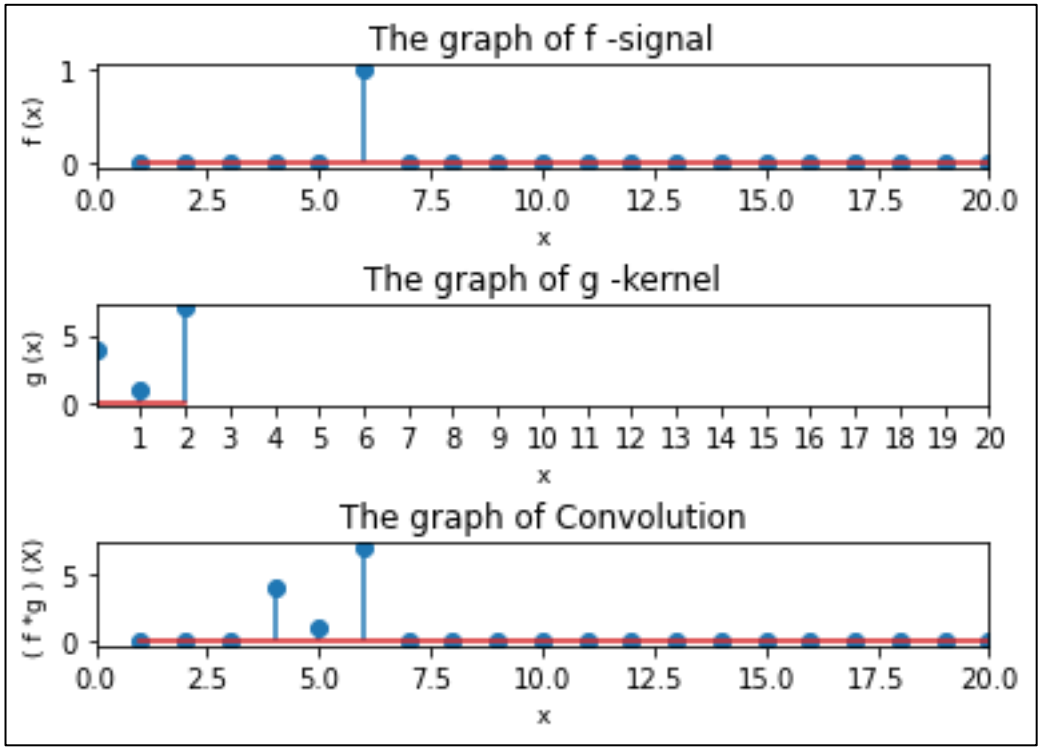


Figure 2.1.1-1 The graph of f-signal, g-kernel and Convolution for discrete function

This f- signal graph shows that its responses is visualized a delta function and also it is normalized impulse because its all of x points values has zeros except x=6 point which is going to the value 1. And also the graph of g-kernel is acted like an impulse response in x range of 0 to 2. The graph of convolution shows that the output responses are given in x point of 4,5,6 and its values are 4,1,7 respectively other are zeros.

2.1.2 Part B

- The same steps as mentioned in part A, were repeated and defined different kernel (g(x) functions) like below and the g vector was kernel, and it was taken reversely by using g(x)
- The convolution of f and g functions were Implemented by using discretized convolution operation.
- Then the code executing time is calculated using difference of the start time and end time.
- 62.817 ms was taken to execute the code.

- Finally, the graph of f-signal, g-kernel and convolution was plotted in the same figure by using the subplot function.

$$g(x) = \begin{cases} 2\ when\ x = 2 \\ 3\ when\ x = 3 \\ 6\ when\ x = 4 \\ 7\ when\ x = 5 \\ 6\ when\ x = 6 \\ 3\ when\ x = 7 \\ 2\ when\ x = 8 \\ 2\ when\ x = 13 \\ 3\ when\ x = 14 \\ 6\ when\ x = 15 \\ 7\ when\ x = 16 \\ 6\ when\ x = 17 \\ 3\ when\ x = 18 \\ 2\ when\ x = 19 \end{cases}$$

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  1 22:19:47 2021

@author: Umeshika
"""
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
#The variable is created to get the starting time when the code is run
starttime=datetime.now()

def plotgraphs(f,g,h): # define the plot function
    plt.subplots_adjust(hspace=1.30)# set the gap between each graph
    plt.subplot(3,1,1)#first plot of the three graph
    plt.xlim(0.0,20.0)#set x axis limitation
    plt.stem(x,f)#plot signal
    plt.title("The graph of f -signal")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('f (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,2)#second plot of the three graph
    plt.xlim(0.0,20.0)#set x axis limitation
    plt.stem([2,3,4,5,6,7,8,13,14,15,16,17,18,19],g)
    plt.title("The graph of g -kernel")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('g (x)', fontsize = 9)#y axis label

    plt.xticks(x)
    plt.subplot(3,1,3)#Third plot of the three graph
    plt.xlim(0.0,20.0)#set x axis limitation
    plt.stem(x,h)#plot convolution
```

```
    plt.title("The graph of Convolution")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('( f *g ) (X)', fontsize = 9)#y axis la
#x vector was created in range 1 to 20 with stepsize 1
x=np.arange(1,21,1)
f=np.zeros(20)#set zeros for f signal values
f[5]=1#create f signal
g= np.array([2,3,6,7,6,3,2,2,3,6,7,6,3,2])#create g kernel vector
g_r = g[::-1]#get the reverse g kernel
cov1=f*0;#set zeros for convolution vector

#%%#%%get dot product of signal and kernel to get convolution
for i in range(f.size-g_r.size):
    cov1[i] =np.dot(f[i:i+g_r.size],g_r)

plotgraphs(f,g,cov1)#plot the graph
"""The code executing time is calculated using
             difference of the start time and end time"""
print (datetime.now()-starttime)
```

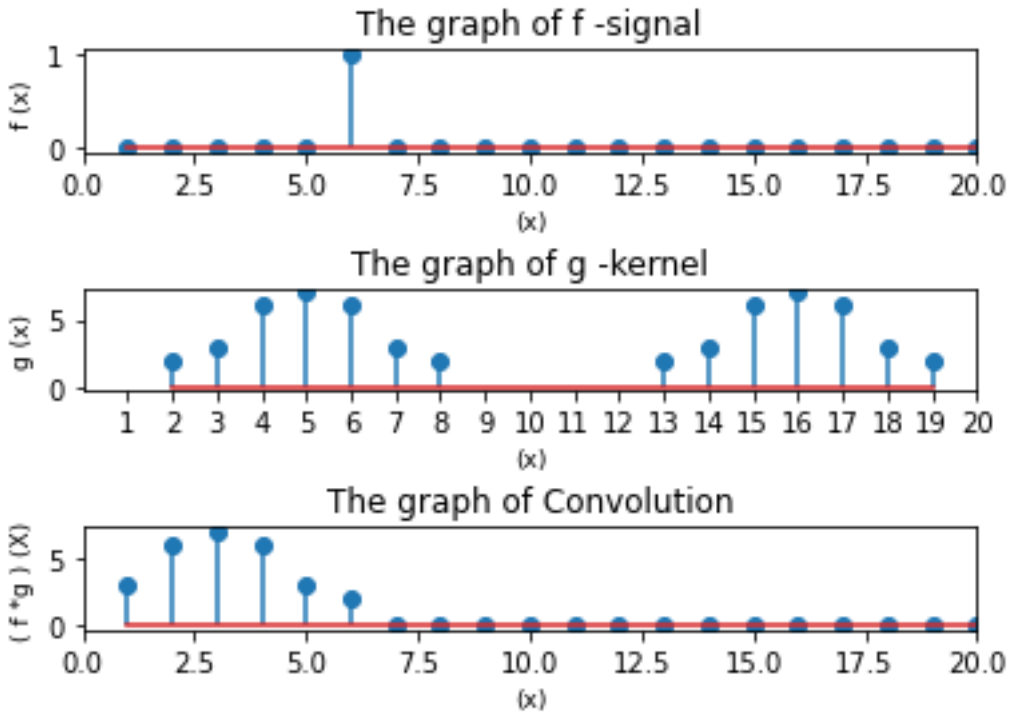


Figure 2.1.2-1 The graph of f-signal, different g-kernel and Convolution for discrete function

This f- signal graph shows that its responses are visualized as a delta function and also it is normalized impulse because its all of x points values has zeros except x=6 point which is going to the value 1. And also the graph of g-kernel is acted like an impulse response in x range of 2 to 8 and 13 to 19 with values [2,3,6,7,6,3,2,2,3,6,7,6,3,2] respectively. The graph of convolution shows that the output responses are given in x point of 1 to 6 and its values are 3,6,7,6,3,2 respectively other are zeros.

2.2 Exercise 2- 1D convolution for continuous functions upon vectoring them

2.2.1 Part A

Implement the following functions and vectorize them (use step size as 0.01)

$x \in [0,3]$ $\quad f(x) = \begin{cases} 1 & \text{when } 1 < x < 2 \\ 0 & \text{otherwise} \end{cases}$

$x \in [0,1]$ $\quad g(x) = \begin{cases} 1 & \text{when } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$

- Firstly, the numerical background of the problem was identified and implemented the mathematical functions and steps were before creating the python code. Therefore this is a continuous convolution operation that can be given as

$$(f * g)(X) = \int_{-\infty}^{+\infty} f(X).g(x - X)dX$$

- Firstly the library of “NumPy”, “matplotlib” and “datetime” was imported and created the variable that was named of “start” to get the starting time when the code was run.
- The g(x) and f(x) were Implemented as vectors in python with its limitations by using if functions. In here, f(x) was called signal. The step size was taken as 0.01
- The maximum and minimum values of signal and kernel were defined and each x vector was also defined.
- The g vector was the kernel, and the reverse kernel was taken reversely by using g(x)
- The convolution of f and g functions were Implemented by using continuous convolution operation.
- Then the code executing time is calculated using the difference of the start time and end time.
- 56.864 ms was taken to execute the code.
- Finally, the graph of f-signal, g-kernel and convolution was plotting them in the same figure by using the subplot function.

In this code, the “convolutionplot “function was defined to plot the graphs in the same figure. Therefore, the f signal graph was plot using “plt. subplot” command and “plt.xlim” that was used for get the limited x-axis. Likewise, again g kernel and convolution were subplot. And each graph was labeled their x-axis, y-axis and title.

In continuous convolution, the signal and kernel were obtained using separate for loops with increments of i=i+1 for signal and j=j+1 for kernel. The signal and kernel were given value 1 for particular x range others are zeros.

In this code convolution function was generated using for loop and without using convolution python library. Therefore, the convolution was calculated for given f signal and g kernel using dot product of f signal and g reversed kernel in range of i to (i+ reverse kernel size)

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  2 14:24:34 2021

@author: Umeshika
"""
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
'''The variable is created to get the
 starting time when the code is run'''
starttime=datetime.now()

def f(x):#define the f signal
    if x>1 and x<2:
        return 1
    return 0

def g(y): #define the g kernel
    if 0<y<1:
        return 1;
    return 0;
#define the plot function to get the all graph in one plot
def convolutionplot(f,g,c):
    plt.subplots_adjust(hspace=1.8)# set the gap between each graph
    plt.subplot(3,1,1)#first plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals,f)#plot signal
    plt.title("The graph of f -signal")
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('f (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,2)#second plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals2,g)#plot kernel
    plt.title("The graph of g -kernel")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
```

```
    plt.ylabel('g (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,3)#Third plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals,c)#plot convolution
    plt.title("The graph of Convolution")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('f (x).g (x)', fontsize = 9)#y axis la

#%%
step_size=0.01# step size continously increment

signal_x_min= 0#minimun value of signal
signal_x_max= 3#maximun value of signal
kernel_x_min= 0#minimun value of kernel
kernel_x_max= 1#maximun value of kernel
#x vector was created for signal in range 1 to 20 with stepsize 1
x_vals=np.arange(signal_x_min,signal_x_max+step_size, step_size)
signal=x_vals*0#set zeros for f signal values
x_vals2=np.arange(kernel_x_min,kernel_x_max+step_size, step_size)
#x vector was created for kernel in range 1 to 20 with stepsize 1
kernel=x_vals2*0
#Initialized values
i=0
j=0

#%%create f signal
for x in x_vals:
    signal[i]=f(x)
    i=i+1
#Create g kernel
for y in x_vals2:
    kernel[j]=g(y)
    j=j+1

kernel_r=kernel[::-1]#get the reverse g kernel

cov1=signal*0;#set zeros for convolution vector
#%%get dot product of signal and kernel to get convolution
for i in range (signal.size-kernel_r.size):
    cov1[i] =np.dot(signal[i:i+kernel_r.size],kernel_r)

convolutionplot(signal,kernel,cov1) #plot the graphs

"""The code executing time is calculated using
 difference of the start time and end time"""
print (datetime.now()-starttime)
```

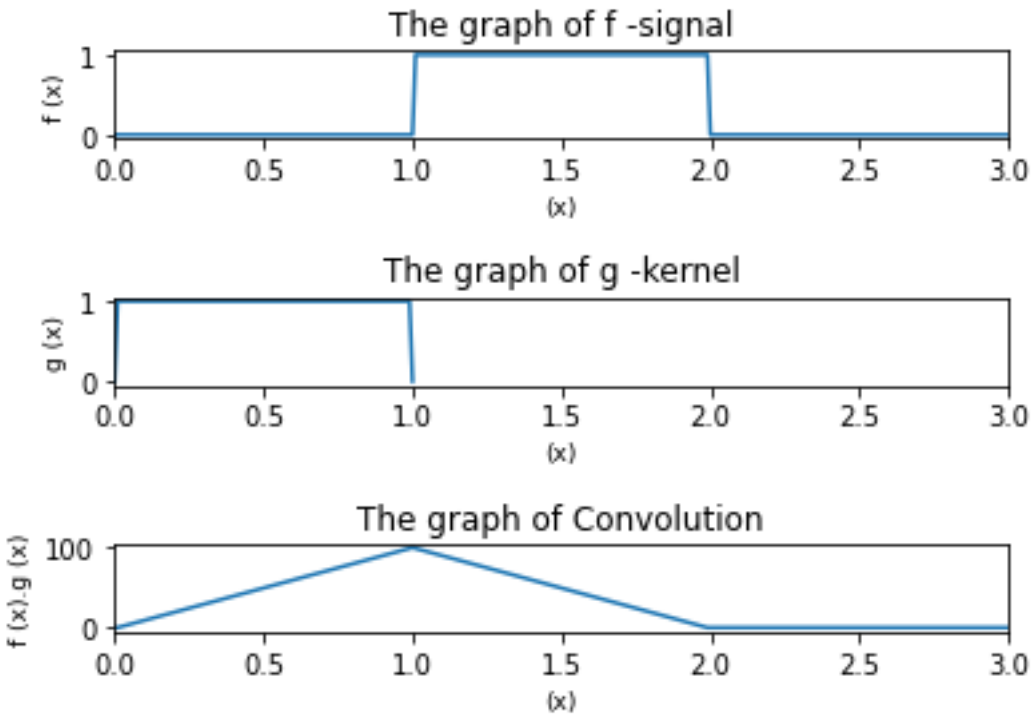


Figure 2.2.1-1 The graph of f-signal, g-kernel and Convolution for continuous function

This f- signal graph shows that its responses are visualized square pulse with x range in 1 to 2. And also the graph of the g-kernel is illustrated the square pulse in the x range of 0 to 1. The graph of convolution shows that the output responses are given a range of 0 to 2 and its maximum value is given as 100 and when x is greater than two, convolution values are zeros. The sharp square impulse and triangle impulse can be obtained when decreasing step size such as 0.001.

2.2.2 Part B

- The same steps as mentioned in part A, were repeated and defined different kernel (g(x) functions) like below and the g vector was the kernel, and it was taken reversely by using g(x)

$$x \in [0,1] \qquad g(x) = \begin{cases} x & \text{when} \quad 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

- The convolution of f and g functions were Implemented by using continuous convolution operation.
- Then the code executing time is calculated using the difference of the start time and end time.
- 46.865 ms was taken to execute the code.

- Finally, the graph of f-signal, g-kernel and convolution were plotted them in the same figure by using the subplot function.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  2 14:24:34 2021

@author: Umeshika
"""
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
#The variable is create to get the starting time when the code is run
starttime=datetime.now()

def f(x):#define the f signal
    if x>1 and x<2:
        return 1
    return 0

def g(y): #define the g kernel
    if 0<y<1:
        return y; #give the y
    return 0;
#define the plot function to get the all graph in one plot
def convolutionplot(f,g,c):
    plt.subplots_adjust(hspace=1.8)# set the gap between each graph
    plt.subplot(3,1,1)#first plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals,f)#plot signal
    plt.title("The graph of f -signal")
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('f (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,2)#second plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals2,g)#plot kernel
    plt.title("The graph of g -kernel")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('g (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,3)#Third plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals,c)#plot convolution
    plt.title("The graph of Convolution")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('f (x).g (x)', fontsize = 9)#y axis la

#%%
step_size=0.01# step size continously increment

signal_x_min= 0#minimun value of signal
```

```
signal_x_max= 3#maximun value of signal
kernel_x_min= 0#minimun value of kernel
kernel_x_max= 1#maximun value of kernel
#x vector was created for signal in range 1 to 20 with stepsize 1
x_vals=np.arange(signal_x_min,signal_x_max+step_size, step_size)
signal=x_vals*0#set zeros for f signal values
x_vals2=np.arange(kernel_x_min,kernel_x_max+step_size, step_size)
#x vector was created for kernel in range 1 to 20 with stepsize 1
kernel=x_vals2*0
#initialized values
i=0
j=0

#%%create f signal
for x in x_vals:
    signal[i]=f(x)
    i=i+1
#create g kernel
for y in x_vals2:
    kernel[j]=g(y)
    j=j+1

kernel_r=kernel[::-1]#get the reverse g kernel

cov1=signal*0;#set zeros for convolution vector
#%%get dot product of signal and kernel to get convolution
for i in range (signal.size-kernel_r.size):
    cov1[i] =np.dot(signal[i:i+kernel_r.size],kernel_r)

convolutionplot(signal,kernel,cov1) #plot the graphs

"""The code executing time is calculated using
 difference of the start time and end time"""
print (datetime.now()-starttime)
```

In this code, the “convolutionplot “function was defined to plot the graphs in the same figure. Therefore, the f signal graph was plot using “plt. subplot” command and “plt.xlim” that was used for get the limited x axis. Likewise, again g kernel and convolution were subplot. And each graph was labeled their x axis, y axis and title.

In continuous convolution, the signal and kernel were obtained using separate for loops with increments of i=i+1 for signal and j=j+1 for kernel. The signal was given value 1 for particular x range others are zeros.And also the kernel was given as y values for considering x range.

In this code convolution function was generated using for loop and without using convolution python library. Therefore, the convolution was calculated for given f signal and g kernel using dot product of f signal and g reversed kernel in range of i to (i+ reverse kernel size)

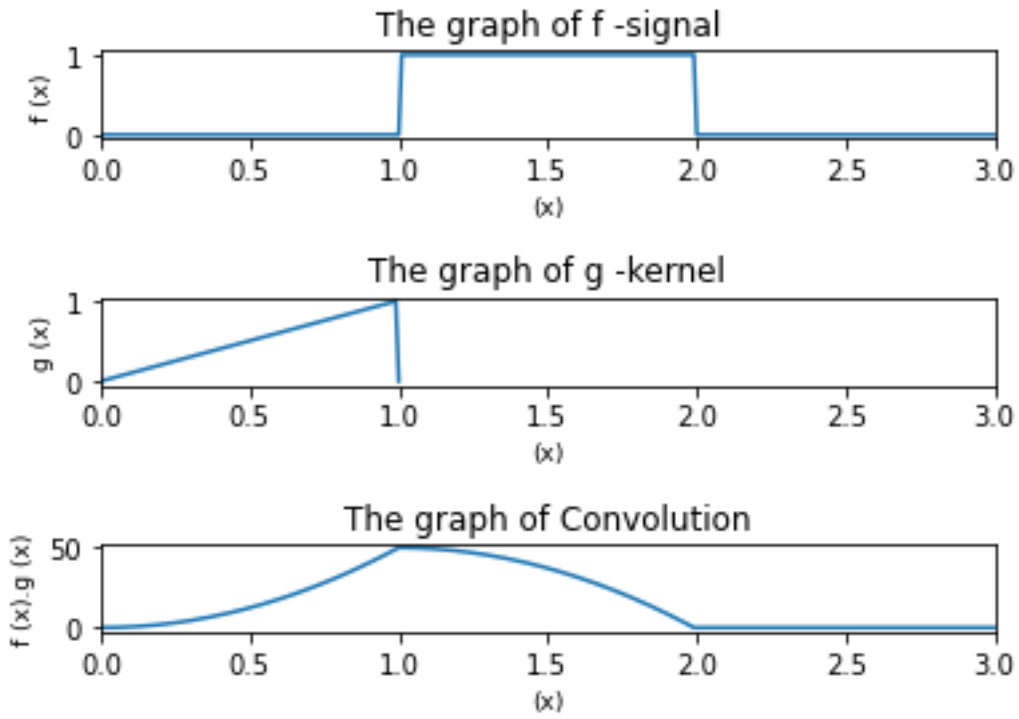


Figure 2.2.2-1 The graph of f-signal, g-kernel when g(x)=x and Convolution for continuous function

This f- signal graph shows that its response is a visualized square pulse with x range in 1 to 2. And also the graph of the g-kernel has illustrated the right triangle pulse in the x range of 0 to 1. The graph of convolution shows that the output responses are given a range of 0 to 2 and its maximum value is given as 50 and when x is greater than two, convolution values are zeros. The sharpness of impulses' shapes can be increased when decreasing step size such as 0.001.

2.2.3 Part C

- The same steps as mentioned in part A, were repeated and defined different kernel (g(x) functions) like below and the g vector was the kernel, and it was taken reversely by using g(x)

$$x \in [0,1] \qquad g(x) = \begin{cases} e^{1-x} & \text{when } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

- The math library was imported and calculated power of the (1-x) in exponential. After that g keener was created using it.
- The convolution of f and g functions were Implemented by using continuous convolution operation.

- Then the code executing time is calculated using the difference of the start time and end time.
- 38.871 ms was taken to execute the code.
- Finally, the graph of f-signal, g-kernel and convolution were plot them in the same figure by using subplot function.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  2 14:24:34 2021

@author: Umeshika
"""
import numpy as np
import matplotlib.pyplot as plt
import math
from datetime import datetime
#The variable is create to get the starting time when the code is run
starttime=datetime.now()

def f(x):#define the f signal
    if x>1 and x<2:
        return 1
    return 0

def g(y): #define the g kernel
    if 0<y<1:
        return math.exp( 1-y );#define exponetial^(1-y)
    return 0;
#define the plot function to get the all graph in one plot
def convolutionplot(f,g,c):
    plt.subplots_adjust(hspace=1.8)# set the gap between each graph
    plt.subplot(3,1,1)#first plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals,f)#plot signal
    plt.title("The graph of f -signal")
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('f (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,2)#second plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals2,g)#plot kernel
    plt.title("The graph of g -kernel")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('g (x)', fontsize = 9)#y axis label

    plt.subplot(3,1,3)#Third plot of the three graph
    plt.xlim(0.0,3.0)#set x axis limitation
    plt.plot(x_vals,c)#plot convolution
    plt.title("The graph of Convolution")#set the title
    plt.xlabel('(x)', fontsize = 9)#x axis label
    plt.ylabel('f (x).g (x)', fontsize = 9)#y axis la
```

```
46
47 #%%
48 step_size=0.01# step size continously increment
49
50 signal_x_min= 0#minimun value of signal
51 signal_x_max= 3#maximun value of signal
52 kernel_x_min= 0#minimun value of kernel
53 kernel_x_max= 1#maximun value of kernel
54 #x vector was created for signal in range 1 to 20 with stepsize 1
55 x_vals=np.arange(signal_x_min,signal_x_max+step_size, step_size)
56 signal=x_vals*0#set zeros for f signal values
57 x_vals2=np.arange(kernel_x_min,kernel_x_max+step_size, step_size)
58 #x vector was created for kernel in range 1 to 20 with stepsize 1
59 kernel=x_vals2*0
60 #initialized values
61 i=0
62 j=0
63
64 #%%create f signal
65 for x in x_vals:
66     signal[i]=f(x)
67     i=i+1
68 #create g kernel
69 for y in x_vals2:
70     kernel[j]=g(y)
71     j=j+1
72
73 kernel_r=kernel[::-1]#get the reverse g kernel
74
75 cov1=signal*0;#set zeros for convolution vector
76 #%%get dot product of signal and kernel to get convolution
77 for i in range (signal.size-kernel_r.size):
78     cov1[i] =np.dot(signal[i:i+kernel_r.size],kernel_r)
79
80 convolutionplot(signal,kernel,cov1) #plot the graphs
81
82 """The code executing time is calculated using
83  difference of the start time and end time"""
84 print (datetime.now()-starttime)
```

In this code the “convolutionplot “function was defined to plot the graphs in the same figure. Therefore, the f signal graph was plot using “plt. subplot” command and “plt.xlim” that was used for get the limited x axis. Likewise, again g kernel and convolution were subplot. And each graph was labeled their x axis, y axis and title.

In continuous convolution, the signal and kernel were obtained using separate for loops with increments of i=i+1 for signal and j=j+1 for kernel. The signal was given value 1 for particular x range others are zeros. And also the kernel was given as power of the (1-y) in exponential values for considering y range.

In this code convolution function was generated using for loop and without using convolution python library. Therefore, the convolution was calculated for a given f signal and g kernel using dot product of f signal and g reversed kernel in range of i to (i+ reverse kernel size) .

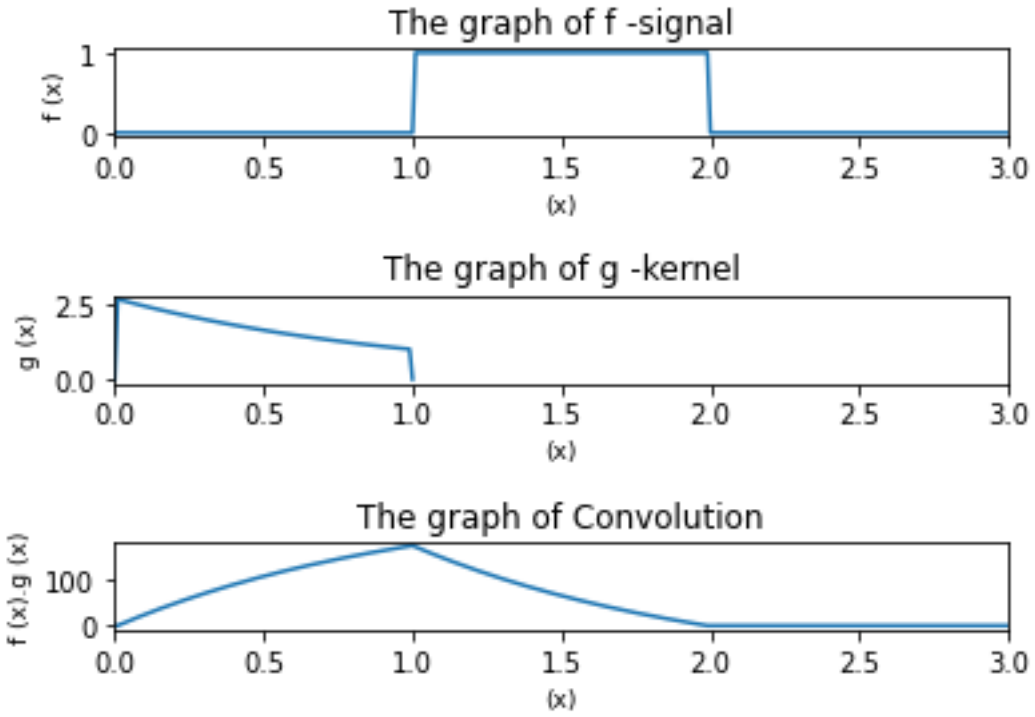


Figure 2.2.3-1- The graph of f-signal, g-kernel when g(x)=e^(1-x) and Convolution for continuous function

This f- signal graph shows that its responses is a visualized square pulse with x range in 1 to 2. And also the graph of the g-kernel is illustrated the exponential decaying pulse in x range of 0 to 1. The graph of convolution shows that the output responses are given range of 0 to 2 and its maximum value is given more than 100 and when x is greater than two, convolution values are zeros. The sharpness of impulses' shapes can be increased when decreasing step size such as 0.001.

2.3 Exercise 3- Conversion of 1D discrete convolution function to the 2D scenario

2.3.1 Part A

```
def image(data,grid_size=5,grid="on",cmap1='gray'):
    d=data
    plt.figure()
    plt.imshow(d,cmap=cmap1,extent=[0,d.shape[0],d.shape[1],0])
    if grid=="on":
        plt.pcolormesh(d, edgecolors='midnightblue', linestyle='dotted',linewidth=1,cmap='gray')
    ax = plt.gca()
    ax.set_xticks(np.arange(0.5, d.shape[0], grid_size))
    ax.set_xticklabels((np.arange(0.5, d.shape[0], grid_size)-0.5).astype(int))
    ax.set_yticks(np.arange(0.5, d.shape[1], grid_size))
    ax.set_yticklabels((np.arange(0.5, d.shape[1], grid_size)-0.5).astype(int))
    plt.show()
```

The new function is created called image which included four arguments such as data that is going to plot, the grid size is defined, the grid is on or off and cmp1 is called colour map which is assigned gray colour. The variable is defined d for data. The new figure is taken using plt.figure() command. After that using plt.imshow() command, the figure is obtained within defined ticked range, assigned colour map colour and data. If the grid is on plot the grid within midnight blue color and 01 widths of line width using plt.pcolormesh command. The current image axis is taken using plt.gca() command. Then the x tick size is changed to obtained the needed place and x axis is label with integers using ax.set_xticklabels command. Likewise, the y tick size is changed to obtained the needed place and y axis is label with integers using ax.set_yticklabels command. Finally, the image is a plot by using plt.show().

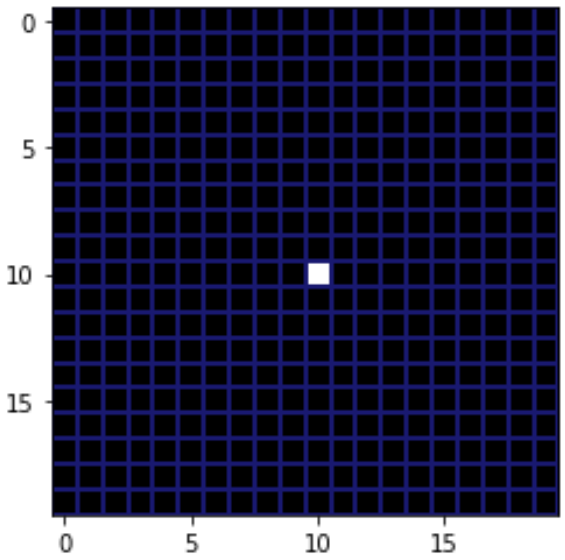


Figure 2.3.1-1 The image of signal f which is taken by using image function -image(f)

The image function is given the best visualization f signal than normal plt.imshow command because x and y axis ticks are Cleary illustrated in here using integers. Not only that midnight blue color grid is clearly visualized each element of two-dimensional array. Therefore, the signal can be quickly determined when analyzing this figure which is obtaining by using image function

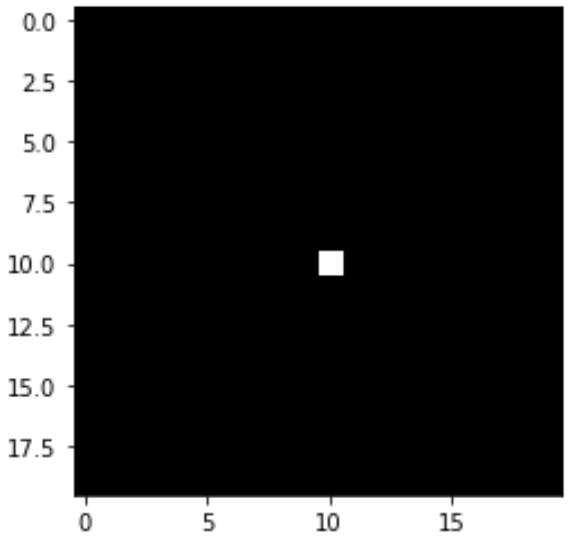
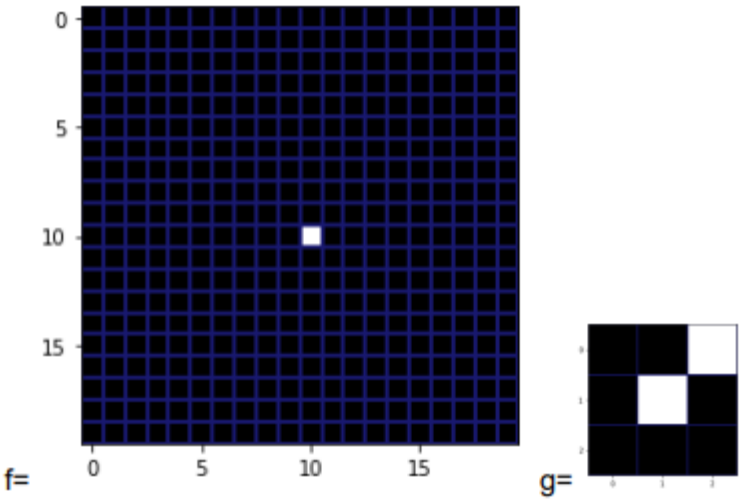


Figure 2.3.1-2 2.3.1-3 The image of signal f which is taken by using command of plt.imshow(f,cmap='gray')

The command of plt.imshow is not Cleary illustrated in x and y axis ticks because it has float number in the axiz. Not only that without grid is not clearly visualized each element of two-dimensional array. Therefore, the signal can not be quickly determined when analyzing this figure which is obtaining by using plt.imshow command

- Firstly, the numerical background of the problem was identified and implemented the mathematical functions and steps were before creating the python code. Therefor this is two dimensional convolution operation.
- Firstly the library of “NumPy” , “matplotlib” and “datetime” was imported and created the variable that was named of “start” to get the starting time when the code was run.
- The given g(x) and f(x) were Implemented as vectors in python with its limitations. In here, f(x) was called signal.



- The g vector was the kernel and reversed kernel was taken reversely by using g(x)
- The convolution of f and g functions were Implemented by using 2-dimensional convolution operation.
- Then the code executing time is calculated using the difference of the start time and end time.
- 0.393 s was taken to execute the code.
- Finally, the graph of f-signal, g-kernel and convolution was plot them one by one by using image function.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  2 17:40:13 2021

@author: Umeshika
"""
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
#The variable is create to get the starting time when the code is run
starttime=datetime.now()
#the image funtion is defined
def image(data,grid_size=5,grid="on",cmap='gray'):
    d=data
    plt.figure()
    plt.imshow(d,cmap='gray',extent=[0,d.shape[0],d.shape[1],0])
    if grid=="on":
        plt.pcolormesh(d, edgecolors='midnightblue',
                       linestyle='dotted',linewidth=1,cmap='gray')
    ax = plt.gca()
    ax.set_xticks(np.arange(0.5, d.shape[0], grid_size))
    ax.set_xticklabels((np.arange(0.5, d.shape[0], grid_size)-0.5).astype(int))
    ax.set_yticks(np.arange(0.5, d.shape[1], grid_size))
    ax.set_yticklabels((np.arange(0.5, d.shape[1], grid_size)-0.5).astype(int))
```

```
    plt.show()
#%%
f=np.zeros([20,20])#set zeros for f signal values
f[10,10]=1#create f signal
f[15,15]=0

g=np.array([[0,0,1],[0,1,0],[0,0,0]])#create g kernel 2D vector
g_r=np.rot90(g,2)#get the reverse g kernel

cov1=f*0#set zeros for convolution vector
#%%#%%get dot product of signal and kernel to get convolution
for i in range(f.shape[0]-g_r.shape[0]):
    for j in range(f.shape[1]-g_r.shape[1]):
        cov1[i:i+g_r.shape[0],j:j+g_r.shape[1]]=np.multiply(f[i:i+g_r.shape[0],
                                                            j:j+g_r.shape[1]],g_r)
        cov1[i,j]=np.sum(cov1[i:i+g_r.shape[0],j:j+g_r.shape[1]])
#%%
image(f)#plot signal using image function
plt.imshow(f,cmap='gray')#plot signal normal way
image(g,1)#plot kernel using image function
image(g_r,1)#plot reversed kernel using image function
image(cov1)#plot signal using image function
"""The code executing time is calculated using
 difference of the start time and end time"""
print (datetime.now()-starttime)
```

In this code the “image “function was defined to plot the graphs in the clear figure. The f signal and g kernet was taken using np.array function defining the elements like matrix. The reverse kernel was taken by using np.rot90() command. It was given flip image of the considering image.

In 2D convolution, the signal and kernel were obtained using double for loops in range of i to difference of signal size and kernel size that were obtained using f.shape and g_r.shape commands and as same as j to difference of signal size and kernel size. Using np.multipy command 2D convolution function was calculated for given f signal and g kernel.

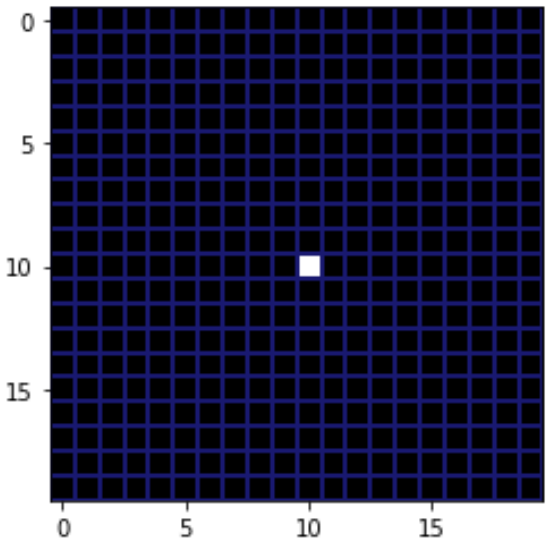


Figure 2.3.1-4- The image of signal f which is taken by using image function -image(f)

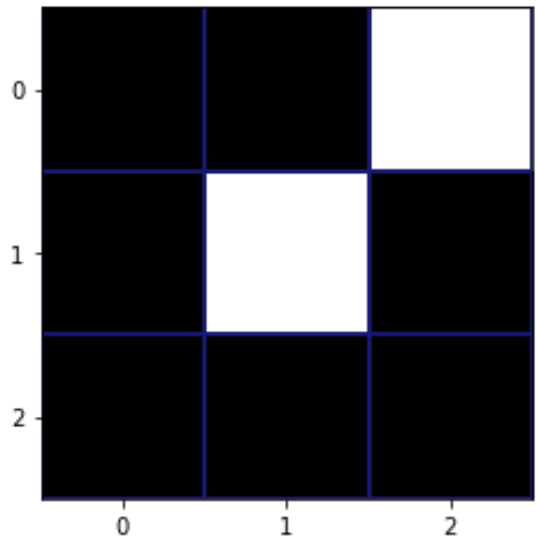


Figure 2.3.1-5 The image of kernel g which is taken by using image function -image(g,1)

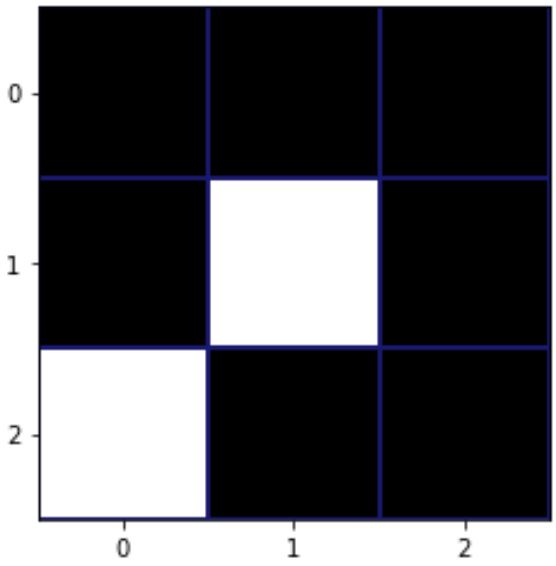


Figure 2.3.1-6 The image of reversed kernel which is taken by using image function -image(g_r,1)

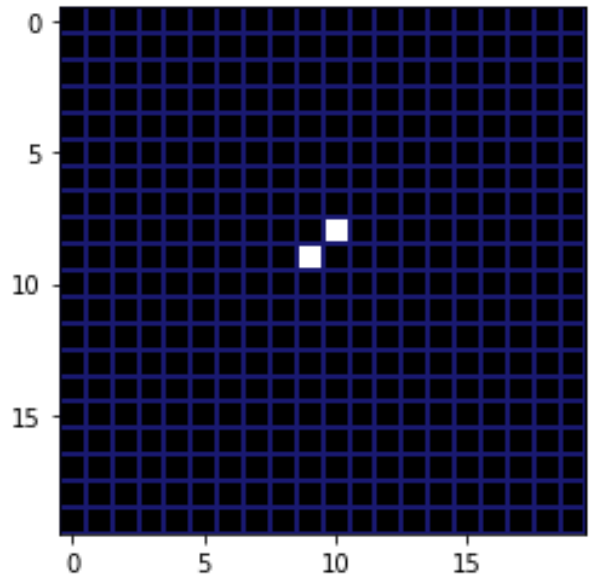


Figure 2.3.1-7 The image of 2D convolution function which is taken by using image function -image(cov1)

2.3.2 Part B

- Firstly, the numerical background of the problem was identified and implemented the mathematical functions and steps were before creating the python code. Therefor this is two-dimensional convolution operation.
- Firstly the library of “cv2”,” skimage”,“NumPy” and “datetime” was imported and created the variable that was named of “start” to get the starting time when the code was run.
- The given g and f were Implemented as vectors in python with its limitations. In here, f was called signal.

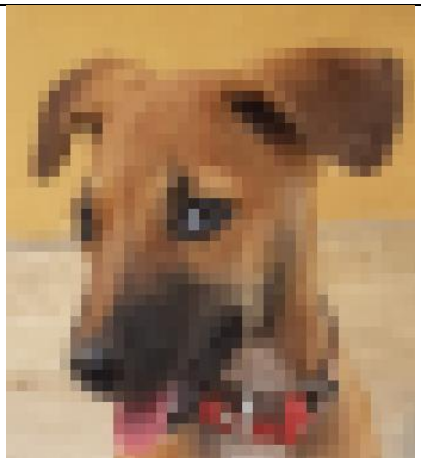


Figure 2.3.2-1 The image for signal in part B

g=

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Figure 2.3.2-2 The image for kernel part B

- The g vector was kernel, and gaussian kernel was taken by np.array
- The convolution of f and g image were Implemented by using 2 dimensional convolution operation.
- Then the code executing time is calculated using difference of the start time and end time.
- 40.766504 s was taken to execute the code.
- Finally, the graph of f-signal and convolution were plot them one by one by cv2.imshow.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct  3 06:54:41 2021

@author: Umeshika
"""
#import Libraries
from skimage import io, img_as_float
import numpy as np
import cv2
from datetime import datetime

#%%
```

```
#The variable is create to get the starting time when the code is run
starttime=datetime.now()
#get the image in gray color scale for signal
img= img_as_float(io.imread("archie_pix.jpg",as_gray=True))
#create kernel
g=np.ones((5,5),np.float32)/25
gaussian_g=np.array([[1/16,1/8,1/16],
                     [1/8,1/4,1/8],
                     [1/16,1/8,1/16]])
#get the convolution
conv1=cv2.filter2D(img, -1, gaussian_g)
cv2.imshow("f-signal",img)#plot f signal
cv2.imshow("2D-convolution",conv1)#plot convolution
cv2.waitKey(0)#delay 0
cv2.destroyAllWindows()
"""The code executing time is calculated using
 difference of the start time and end time"""

print (datetime.now()-starttime)
```

In this code cv2.filter2D library was used to get the convolution and img_as_float was used to read image as signal.

Figure 2.3.2-3 The image of signal f which is taken by using cv2.imshow("f-signal",img)

Figure 2.3.2-4 The image of 2D convolution function which is taken by using cv2.imshow("2D-convolution",conv1)

When comparing both images , It can be shown that convolution figure has more Cleary visualized than f signal figure.

3 REFERENCES

[1] *Image Filtering Using Convolution in OpenCV. LearnOpenCV [online] Available at: https://learnopencv.com/image-filtering-using-convolution-in-opencv/ [Accessed 2 Oct.2021].*

[2] *Convolutions with OpenCV and Python. pyimagesearch. [online] Available at: https://www.pyimagesearch.com/2016/07/25/convolutions-with-opencv-and-python/ [Accessed 2 Oct.2021].*

[3] *Calculating the Convolution of Two Functions With Python. Start it Up [online] Available at: https://medium.com/swlh/calculating-the-convolution-of-two-functions-with-python-8944e56f5664 [Accessed 2 Oct.2021].*

[4] *Python 3 - Number exp() Method. Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/python3/number_exp.htm [Accessed 2 Oct.2021].*

[5] *Function definition. Math Insight. [online] Available at: https://mathinsight.org/definition/function [Accessed 2 Oct.2021].*