ECE 351

Signals and Systems

Lab 3

Convolution Function

Submitted By :
Zachary Pfaff

Contents

Convolution Function

Zachary Pfaff

February 8th, 2022

1 Introduction

Convolution is a fundamental topic to be addressed in signals and systems. To convolve something in signals and systems is to combine two functions.The goal of lab 3 is to explore the functionality of convolution by creating a user defined function in python that will convolve two functions. By doing so, students will gain a better understanding of convolution as well as more experience in programming in python.

2 Equations

The equation below is the mathematical definition of convolution. For this lab, we will not focus on creating functions with integrals.

$$y(t) = h(t) * x(t)$$

Before creating the user defined convolution, three functions to be convolved will be created. The three equations below will be implemented into python and plotted.

$$f_1(t) = u(t-2) - u(t-9)$$

$$f_2(t) = e^{-t}u(t)$$

$$f_3(t) = r(t-2)[u(t-2) - u(t-3)] + r(t-4)[u(t-3) - u(t-4)]$$

3 Methodology

Before programming the convolution function, the three functions to be convolved needed to be implemented and plotted into python. To program these three functions, I referred back to lab 2 where the step and ramp functions were defined. The code below shows the functions being implemented in python.

```
#Code for ramp function
def ramp(t):
    y = np.zeros(t.shape)
    for i in range(len(t)):
        if t[i] < 0:
            y[i] = 0
        else:
            y[i] = t
    return y

#Code for step function
def step(t):
    y = np.zeros(t.shape)
```

```
    for i in range(len(t)):
        if t[i] < 0:
            y[i] = 0
        else:
            y[i] = 1
    return y

f1 = step(t-2) - step(t-9)
f2 = np.exp(-t)*step(t)
f3 = ramp(t-2)*(step(t-2)-step(t-3)) + ramp(4-t)*(step(t-3)-step(t
    -4))

plt.figure(figsize =(12 ,8))
plt.subplot (3,1,1)
plt.plot(t, f1)
plt.title('Function Plots')
plt.ylabel('Function 1')
plt.grid(True)
```

Listing 1: Functions To Be Convolved

When these defined functions are plotted, they result in the following graphs below in Figure 1.

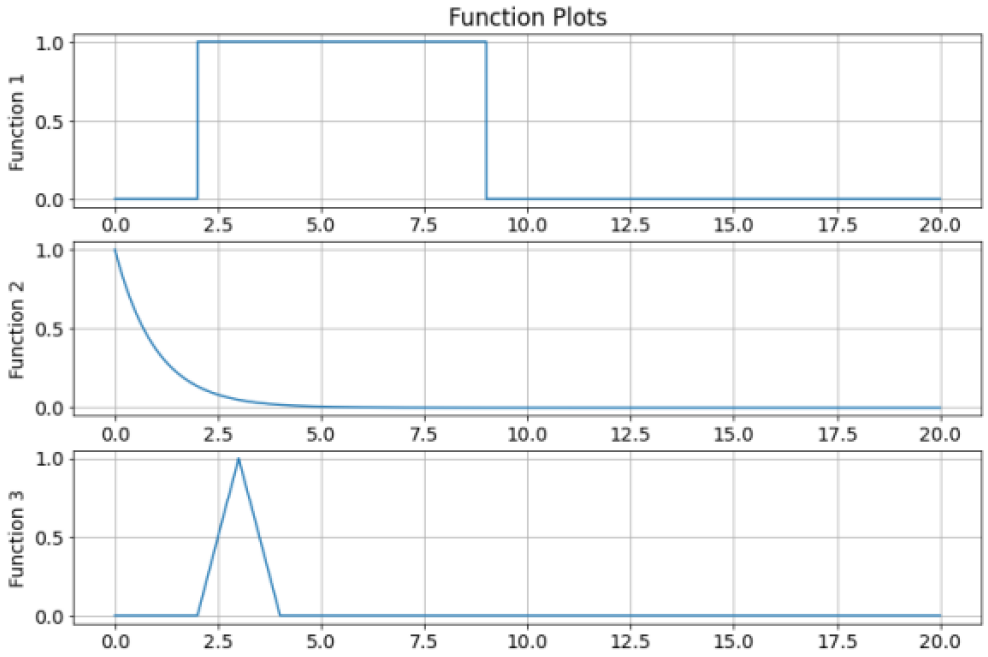

Figure 1: Plots of Functions to be Convolved

With the three functions created and plotted, the next step was to create the user de-

fined convolution function, then compare it to the numpy.convolve function already in the python command library. Thinking about convolution in a graphical sense, I knew that the gradual overlap of two convolving functions results in the convolved plot. Knowing this, I created two new arrays for each function to be convolved that includes the initial values of the already defined function plus an array of zeros that is equal to the size of the array of the opposite function. The code below shows the convolution function being implemented.

```
def my_conv(f1, f2):
    Nf1 = len(f1)    #create new arrays with same length as input
    signals
    Nf2 = len(f2)

    f1Extended = np.append(f1, np.zeros((1, Nf2 - 1)))
    f2Extended = np.append(f2, np.zeros((1, Nf1 - 1)))

    result = np.zeros(f1Extended.shape)
    for i in range(Nf2):
        for j in range(Nf1):
                try: result[i + j] += f1Extended[i] * f2Extended[j]
                except: print(i,j)

    return result
```

Listing 2: Convolution Function

From my understanding, the code for the function works by first creating two new arrays that have the same length as the two input signals. Next, two other arrays are created that share the same length as the input signals with extra zeros added into the arrays to compensate for the shifting of functions that will occur. A result array that is the same size as the extended zeros arrays is created and initialized, then finally two for loops are implemented to calculate the product of the two functions at each index, then shifted to the right by one index to repeat the loop.

4 Results

The user defined convolution function was used to convolve function f1 with f2, f1 with f3, and f2 with f3. The figure 2 below shows the three convolutions using my user defined convolution code. To compare the functionality of my user defined convolution, I used the numpy.convolution command to recreate the same graphs with the same convolved functions. The figure 3 below shows the three convolved functions using numpy.convolve.

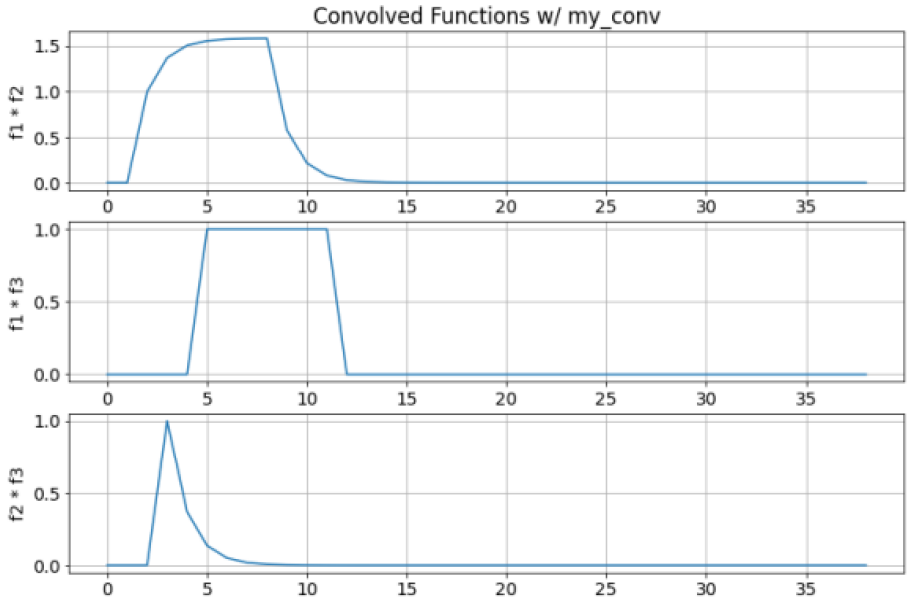

Figure 2: Convolved Functions w/ user defined convolution

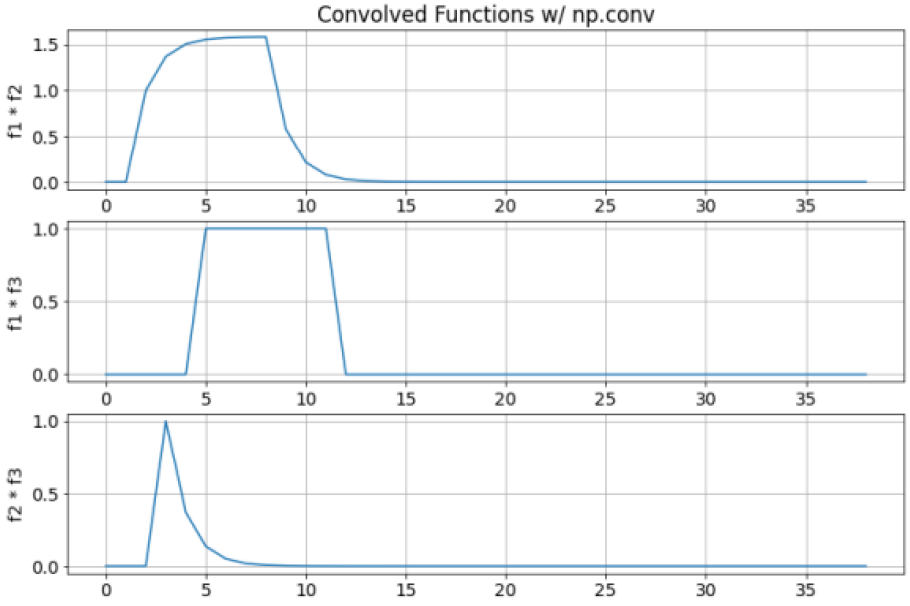

Figure 3: Convolved Functions w/ numpy.convolve

5 Error Analysis

I had some trouble with the initial programming of the convolution. My first

attempt at creating the convolution function is seen in the listing below.

```
1  def my_conv(f1, f2):
2      Nf1 = len(f1)
3      Nf2 = len(f2)
4      f1Extended = np.append(f1, np.zeros((1, Nf2-1))) #add arrays of
       zeros onto f1
5      f2Extended = np.append(f2, np.zeros((1, Nf1-1)))
6      result = np.zeros(f1Extended.shape)
7      for i in range(Nf2 + Nf1 - 2):
8          result[i] = 0
9
10     for j in range(Nf1):
11         if(i - j + 1 > 0 ): #try printing out i and j to test
       debugging
12                 try: result[i] += f1Extended[j] * f2Extended[i - j
       + 1]#try this
13                 except: print(i,j)
14
15     return result
```

Listing 3: Convolution Function

Although it was my intent to set the index of the result to the indexed convolution of both functions, the result array failed to change any of its values. This meant the result would always output an array of zeros as shown in the figure below.

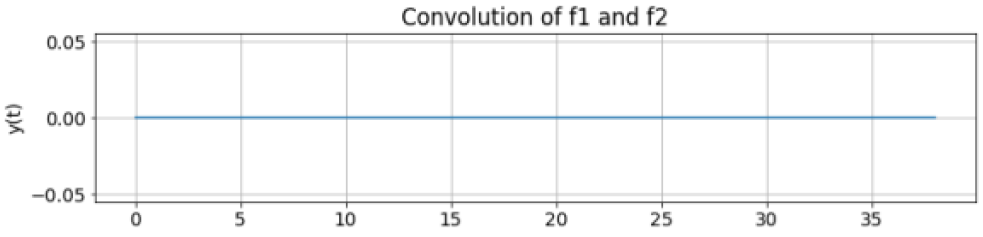

Figure 4: Convolution Resulting in Zero

I attempted to correct this problem by modifying line 12 of the code. When looking at the watch variables, index i was never changing which meant the result array at index i was always going to be zero since there is no value in index i for either f1 or f2. To change this, I set the result at index j instead since the j index would increment by 1 every time the for loops were run. Doing so resulted in the following line of code and the graphed convolution of f1 and f2.

```
1 result[j] += f1Extended[j] * f2Extended[j]
```

Listing 4: Modification of Line 12

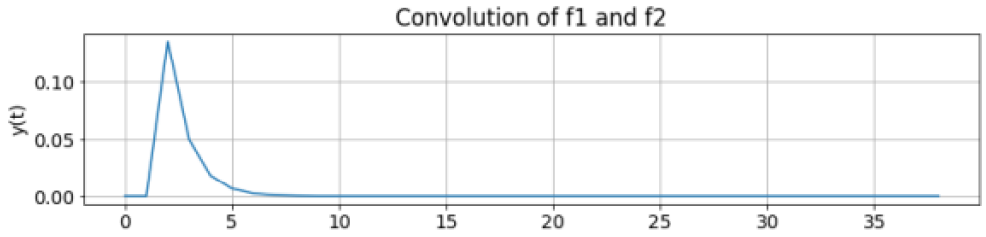

Figure 5: Second Attempt at Convolution

This was an improvement graphically, the problem here however is the j index for functions f1 and f2 are only being multiplied together without being taken into account the previously computed j index. What I needed to do was come up with a way to multiply the j values of both f1 and f2 while also adding on to what was calculated before the next index result.
Through trial and error, I found that if I remove line 8 of the code and set the range for the i index to one of the function lengths, I could take the product of both functions at every index j once for every index i, perfectly creating the intended convolution function as seen in listing 2.

6 Conclusion

By the end of Lab 3, I feel much more confident in my understanding of convolution. For the majority of the lab I worked alone, however after comparing my code with others it seemed my original convolution code was behaving differently, so I had to take a different approach which is what inevitable resulted in listing 2. My approach to the code used graphical convolution in mind, as I knew that in order to create the function I had to shift one of the functions into the other. It took me several attempts to debug the issue, and at some points I would make guesses, but after trial and error I eventually came to the correct implementation of the convolution function.