ECE 351 - Lab 3

Khoi Nguyen
https://github.com/3khoin

September 23, 2021

Contents

1 Introduction

This lab utilized Python in order to understand both how a convolution functions and its manual implementation with a user-defined function.

A convolution is an operation the resulting function of which describes how an input function affects the shape of the other input function. It can be defined as an integral involving the two functions, shown in the Equations section. Convolutions are useful for understanding how a ordered system solved through a differential equation, such as an RLC circuit, responds to a given input function.

2 Equations

Step and ramp functions:

$$u(t) = 0(t \leq 0), 1(t \geq 0)$$

$$r(t) = 0(t \leq 0), t(t \geq 0)$$

Convolution:

$$(f * g)(t) \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau$$

Part 1 Task 1 Equations

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

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

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

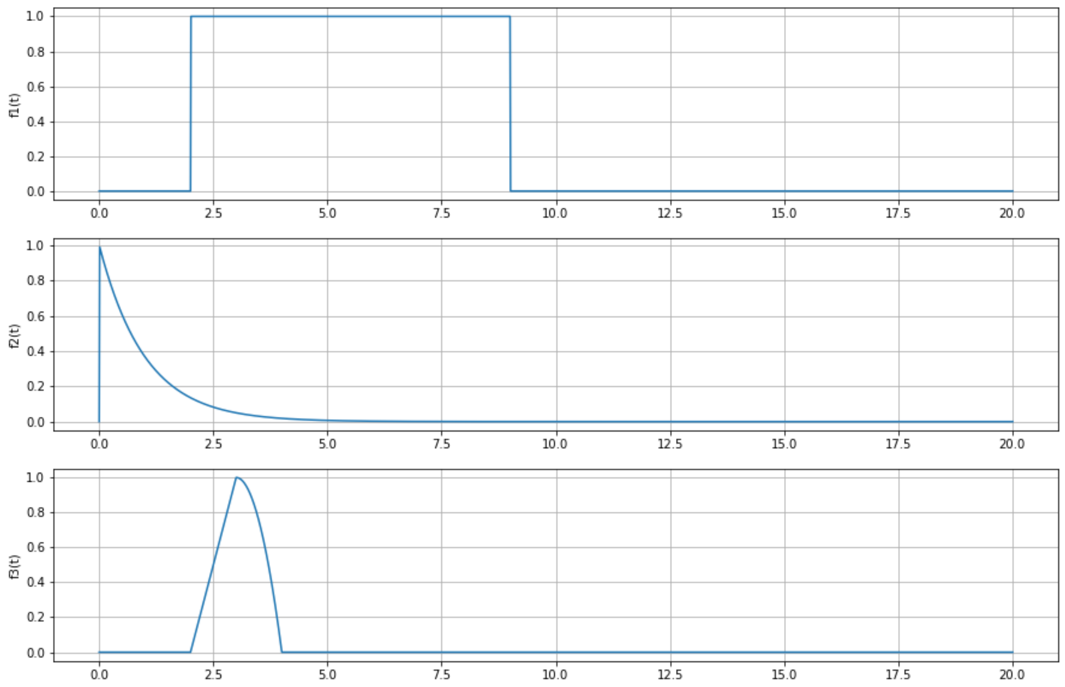
3 Methodology

We first imported the step and ramp functions that we defined in Lab 2, then utilized these to create signals, the equations for which are listed under ”Part 1 Task 1 Equations” in the Equations section. We plotted each in a single figure, with individual subplots, in the domain of 0 $0 \leq t \leq 20$, with a step size of $1 * 10^{-2}$.

We then got to the main portion of the lab: creating code to perform convolutions with two functions. This was done by examining the definition and graphical properties of a convolution, rather than just implementing its integral function. After the convolution function was defined manually, we verified the code by comparing its output to that of the scipy.signal.convolve() function. We used the user-defined convolution function to convolve f_1 and f_2, f_2 and f_3, and f_1 and f_3.

4 Results

Plotted below are the the equations labeled ”Part 1 Task 1 Equations” in the Equations section.



Part 1, Task 2

The code for the convolution function used in the lab is as listed:

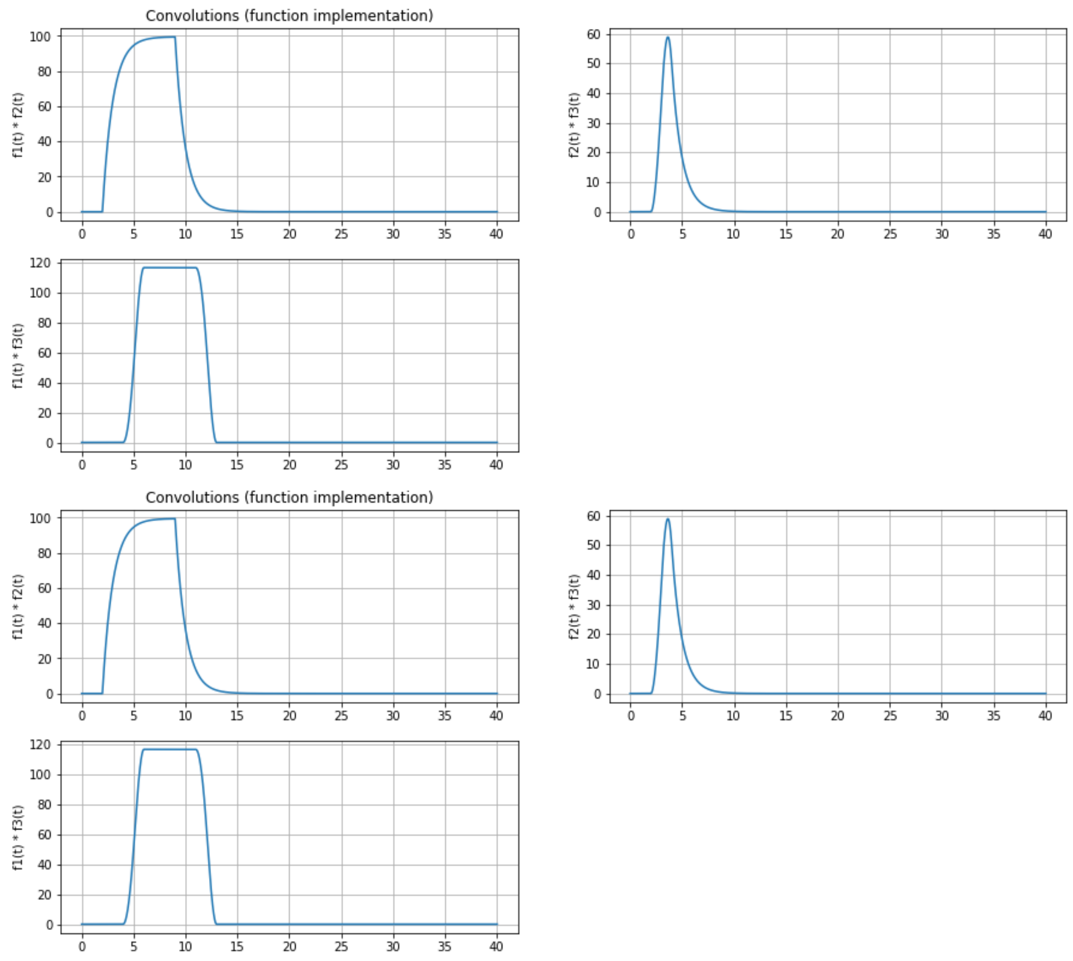
```python
def conv(f1, f2):
    f1new = np.append(f1, np.zeros((1, len(f2)-1)))
    f2new = np.append(f2, np.zeros((1, len(f1)-1)))
    result = np.zeros(f1new.shape)

    for i in range(len(f1) + len(f2) - 2):
      result[i] = 0
    for j in range(len(f1)):
      if(i - j + 1 > 0):
        try:
          result[i] += f1new[j] * f2new[i-j+1]
        except:
          print(i,j)
  return result
```

A description of the functionality is commented as listed:

```
# First, "clones" of the functions passed as arguments are created, with their
   lengths being extended by the length of the other. This is because the domain
   of a convolution combines that of its two input functions.
# A result variable, which is to be later returned, is created in the shape of one
    of the new function clones.
# The double for loop implements the definition of an integral; at each index i in
    the combined length of the two functions, the second function at a point i-j+1
    is appended with the product of its value at that point and the value of all
   points preceding it from the first function. This supports the definition of a
   convolution defined as the output function being the "percent overlap" of the
   two input functions.
# The result array is then returned.
```

Plotted below are the convolutions of f_1 and f_2, f_2 and f_3, and f_1 and f_3, respectively, the first subplot being implemented with our user-defined convolution function, and the other being a verification using the scipy.signal.convolve() function.



Part 2, Task 2, 3, and 4

5 Error Analysis

The most difficult part of this lab was trying to implement the loops that would correspond to the graphical convolution of the two input functions. Initially, I had attempted to implement a single while loop to iterate through all points of one function and multiply by the value of the corresponding point of the second function, and return this value to a third return array. Multiple problems surfaced from this implementation; chiefly, the returned function would have been just a product of the two functions, rather than a proper convolution. Additionally, the return value was misshapen and did not have the proper amount of elements to be plotted using np.arange() time domain. This was solved by combining the lengths of the input functions into a new array and returning that array. As for the initial implementation problem, we figured out that a double for loop would be necessary to multiply all of the necessary points for the convolution.

6 Questions

1. The initial brainstorming for the convolution function was done alone. The final product was formed using the TA demonstration.
2. As stated in the Error Analysis section, reasoning out the multiplication part of the convolution definition into code was the greatest challenge of this lab. The initial single for loop idea having been unsuccessful, we then reasoned out how to solve the problems of dimension check and iterating through both functions, using the provided code.
3. We approached the lab primarily with graphical convolution in mind, as it was the one that would be not only easiest to verify, but that which made most sense to us from a coding standpoint.
4. The expectations for this lab were exceedingly clear, if not slightly open-ended for the convolution implementation.

7 Conclusion