PHY408 Lab One: Convolution Problems

ZhenDi Pan 1003241823

2020-02-05

1 A Discrete Convolution Program

Collaborators: None.

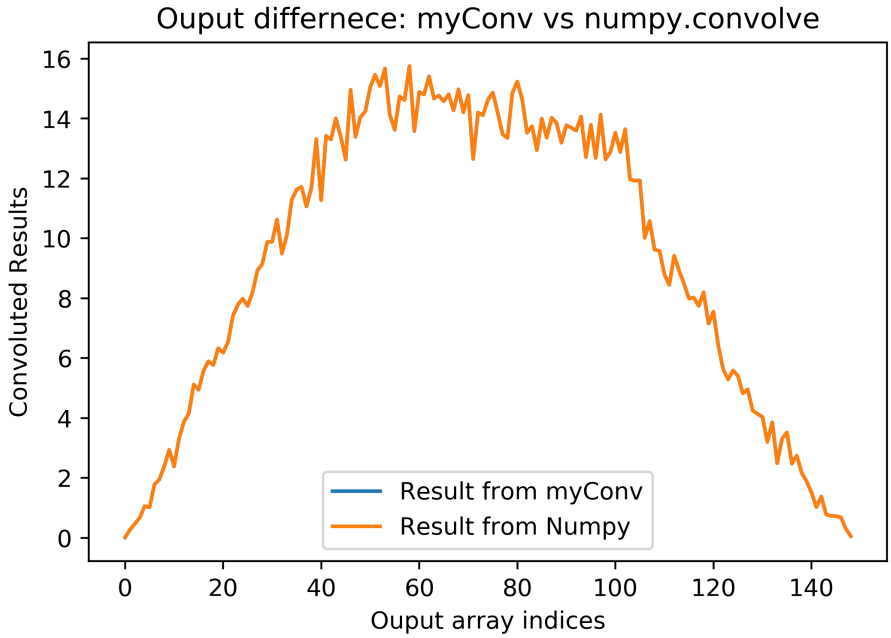
1. The length of $\{g_n\}$ is $N_f + N_w - 1$. Consider two arrays f and w, For the first array, we are assuming $f[0], \ldots, f[N_f - 1]$ are non-zero, and for the second array $w[0], \ldots, w[N_w - 1]$ are non zero. Then the number of indices which $\{g_n\}$ is non-zero is $0 \leq n \leq N_f + N_w - 2$, so it is $N_f + N_w - 1$ terms. Geometrically, it is more intuitive. The number of non-zero points when we flip the second array and overlap it with the first one, the number of points it overlaps start from the first point the arrays overlap until the last non-zero point of second array exiting the last non-zero point of the first array, this gives us a total of non-zero points $N_f + N_w - 1$.

2. The function is included as below:

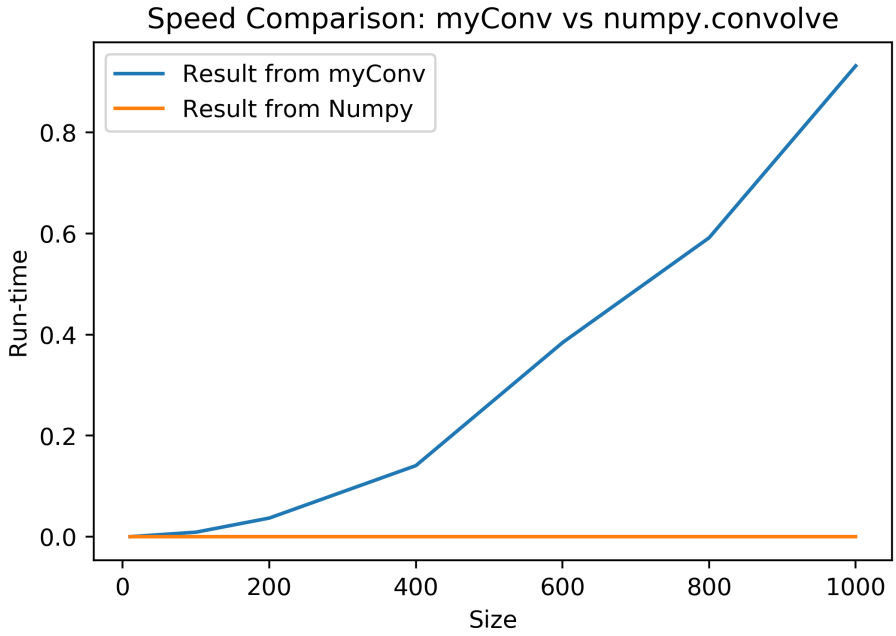
```
def myConv(f, w):
    result = np.zeros(len(f)+len(w)-1)
    for i1 in range(len(result)):
        for i2 in range(len(f)):
            if i1-i2>=0 and i1-i2< len(w):
                result[i1] = result[i1] + f[i2] * w[i1-i2]
    return result
```

3. I generated two random time series with $N_f = 50$ and $N_w = 100$ drawing each element from U[0,1]. The output from myConv is exactly the same as numpy.convolve. In the code, I printed out the first

five outputs and they are identical. Therefore, plot shown on the next page is as expected, two lines overlap completely, so the blue line is not visible(aligns on top of each other).



4. The run-time of myConv is significantly slower than numpy.convolve, especially for large arrays. The reason is that I simply implemented my function using the discrete convolution formula, which can be achieved by two loops. This is clearly not the most elegant way to implement this function. I looked at the source code for numpy.convolve, it uses correlation matrix to compute convolution, this method is much faster. For the suggested cases in handout, when I set $N_f = N_w = 10000$, my function takes almost 90 seconds to run, so I modified the cases to be [10,50,100,200,400,600,800,1000]. Within this range, numpy.convolve executes almost instantaneously. As we can see from the plot shown on the next page, numpy.convolve almost executes instantaneously whereas myConv takes a long time to compute larger arrays.



2 Simple Physical System: RL Circuit Response

Collaborators: Name: Will Zeng,

1. The Kirchhoffs voltage law states that in any closed loop circuit, the total voltage around the loop is equal to the sum of all the voltage drops within the same loop. So we have $a(t) = I(t)R + b(t)$, applying this to our RL circuit and assuming the input voltage across the pair is $a(t) = H(t)$, we obtain the equation $\frac{dI(t)}{dt}L + I(t)R = H(t)$, where $b(t) = \frac{dI(t)}{dt}L$. Dividing L on both sides we can construct the ODE:

$$\frac{dI(t)}{dt} + I(t)\frac{R}{L} = \frac{H(t)}{L}$$

The first order linear ODE can be solved by multiplying an integrating factor. In our case it is $e^{\frac{R}{L}t}$. So we get $I'(t)e^{\frac{R}{L}t} + I(t)\frac{R}{L}e^{\frac{R}{L}t} = \frac{H(t)}{L}e^{\frac{R}{L}t} \Rightarrow \frac{d}{dt}(I(t)e^{\frac{R}{L}t}) = \frac{H(t)}{L}e^{\frac{R}{L}t}$. Taking the integral of both sides: $I(t)e^{\frac{R}{L}t} = \frac{1}{L}\int H(t)e^{(\frac{R}{L})t}dt$. Then we can solve the integral on the right hand side by integration by parts:

$$\begin{aligned}\int H(t)e^{\frac{R}{L}t}dt &= \frac{L}{R}H(t)e^{\frac{R}{L}t} - \frac{L}{R}\int \delta(t)e^{\frac{R}{L}t}dt \\ &= \frac{L}{R}H(t)(e^{\frac{R}{L}t} - 1)\end{aligned}$$

This gives us $I(t) = \frac{1}{R}H(t)(1 - e^{-(\frac{R}{L})t})$, since the constant of integration can be determined by the boundary condition, where $Ce^{\frac{R}{L}0} = 1 \Rightarrow C = 1$. So the step response is then

$$\begin{aligned} S(t) = L\frac{dI(t)}{dt} &= \frac{L}{R}\frac{d}{dt}(H(t)(1 - e^{\frac{R}{L}t})) \\ &= \frac{L}{R}\delta(t)(1 - e^{-(\frac{R}{L})t}) + H(t)e^{-\frac{R}{L}t} \end{aligned}$$

When $t = 0$, $\frac{L}{R}\delta(t)(1 - e^{-(\frac{R}{L})t})$ is zero, thus we get the step response $S(t) = e^{-\frac{Rt}{L}}H(t)$. Similarly for the impulse response $a(t) = \delta(t)$, we construct the ODE $\frac{dI(t)}{dt} + \frac{R}{L}I(t) = \frac{H(t)}{L}$. Solving for $I(t)$ we get $I(t) = \frac{1}{L}H(t)e^{-(\frac{R}{L})t}$. Therefore,

$$\begin{aligned} R(t) = L\frac{dI(t)}{dt} &= \frac{d}{dt}(H(t)(e^{-\frac{R}{L}t})) \\ &= H'(t)e^{-\frac{R}{L}t} - \frac{R}{L}H(t)(e^{-\frac{R}{L}t}) \\ &= \delta(t) - \frac{R}{L}e^{-\frac{Rt}{L}}H(t) \end{aligned}$$

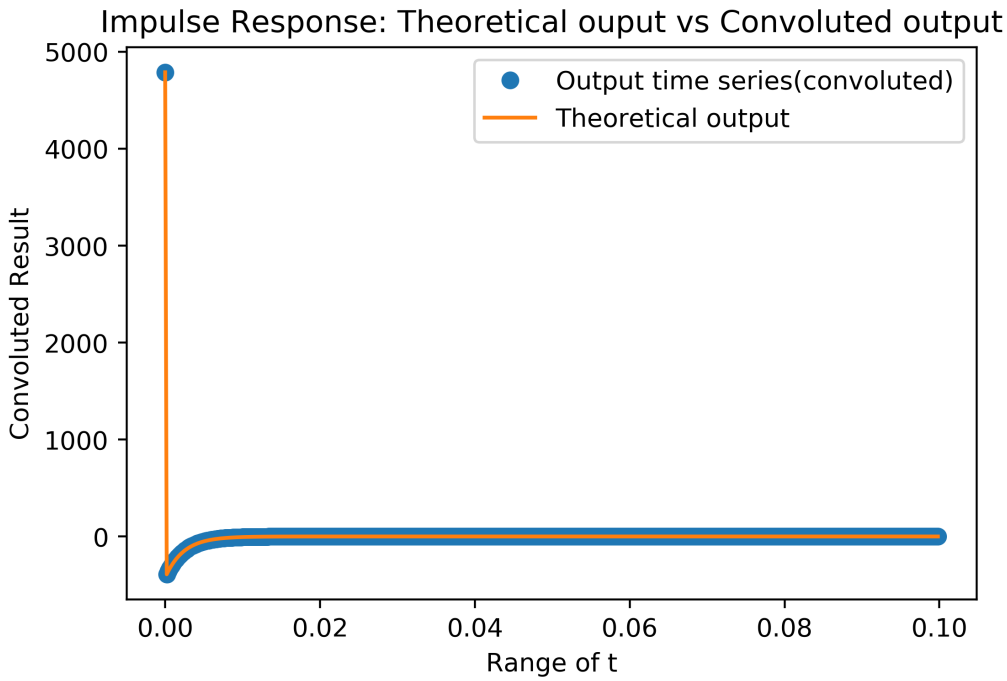
Thus we have showed $S(t) = e^{-\frac{Rt}{L}}H(t)$ and $R(t) = \delta(t) - \frac{R}{L}e^{-\frac{Rt}{L}}H(t)$

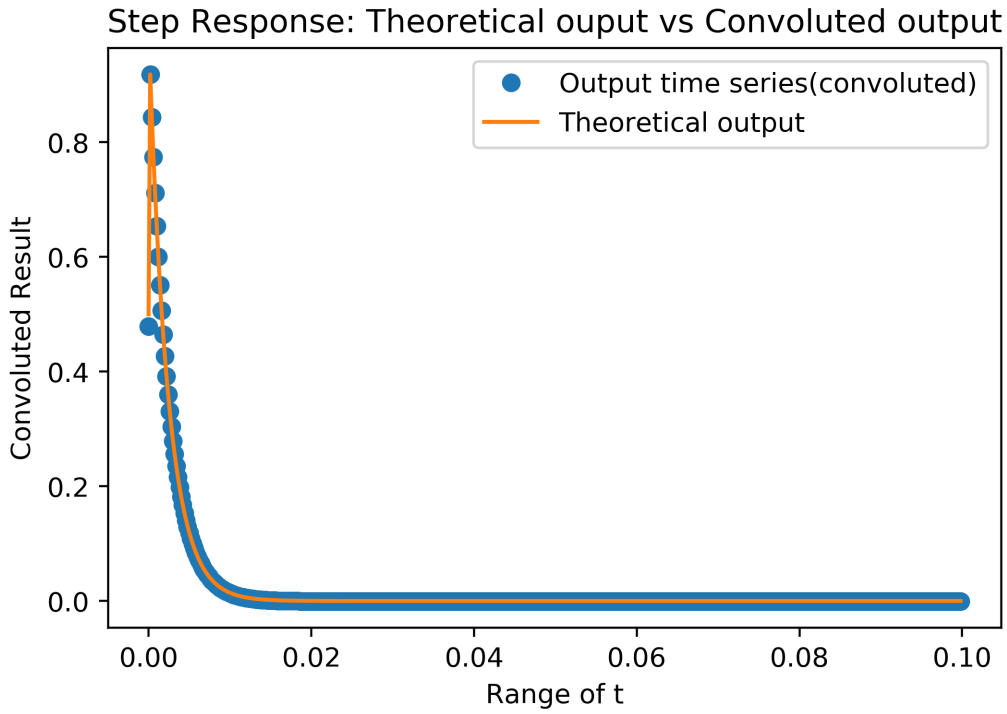
2. The RLresponse function:

```
#H,D function
def HD(n, dt):
    H= np.ones(n)
    D = np.zeros(n)
    H[0] = 0.5
    D[0] = 1 / dt
    return H,D


#RLresponse function
dt = 0.0002
t_range = np.arange(0, 0.1, dt)
n = len(t_range)
def RLresponse(R, L, V_in, dt):
```

```
R_t = HD(n, dt)[1] - (R/L)*np.e**(-R*t_range/L)*HD(n, dt)[0]
return np.convolve(V_in, R_t)*dt
```

3. Using $R = 850, L = 2$, $dt = 0.0002$ seconds and convolution range of 0 to $0.1ms$, the output plots are as follow:





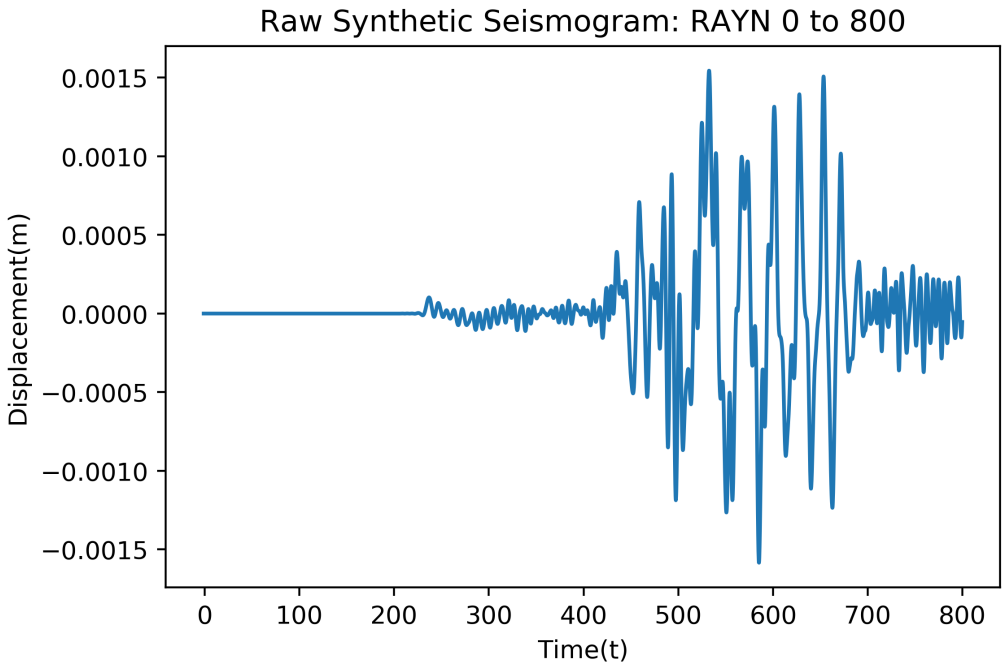
The code is included below:

```
R = 850
L = 2
theoretical_R = HD(n, dt)[1] - (R/L)*np.exp(-R*t_range/L)*HD(n, dt)[0]
theoretical_S = np.e**(-R*t_range/L)*HD(n, dt)[0]
V1, V2 = HD(n, dt)
V_impulse = RLresponse(R, L, V2, dt)
V_impulse = V_impulse[:n]
V_step = RLresponse(R, L, V1, dt)
V_step = V_step[:n]


plt.figure(3)
plt.plot(t_range, V_impulse, ls='',marker='o', label = 'Output time series(
plt.plot(t_range, theoretical_R, label = 'Theoretical output')
plt.xlabel('Range of t')
plt.ylabel('Convoluted Result')
plt.title('Impulse Response: Theoretical ouput vs Convoluted output')
plt.legend()
plt.savefig('p2q31', dpi = 400, bbox_inches='tight')

plt.figure(4)
plt.plot(t_range, V_step, ls='',marker='o', label = 'Output time series(cor
plt.plot(t_range, theoretical_S, label = 'Theoretical output')
plt.xlabel('Range of t')
plt.ylabel('Convoluted Result')
plt.title('Step Response: Theoretical ouput vs Convoluted output')
plt.legend()
plt.savefig('p2q32', dpi = 400, bbox_inches='tight')
```

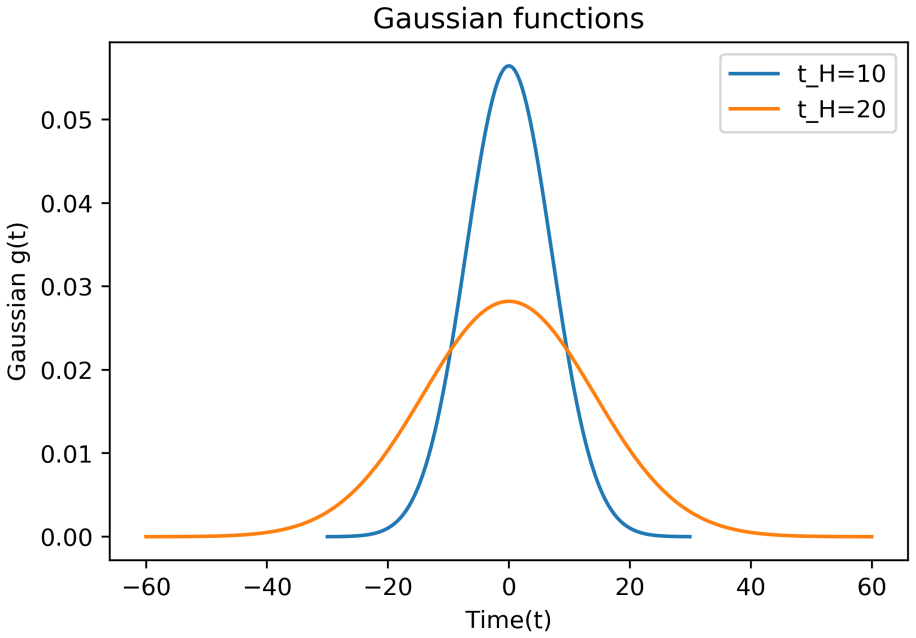
3 Convolution of Synthetic Seismograms

Collaborators: Name: Shenzhi Wang, Student Number: 1002894255

1. The plot of raw synthetic seismogram for station RAYN between 0 and 800 seconds:

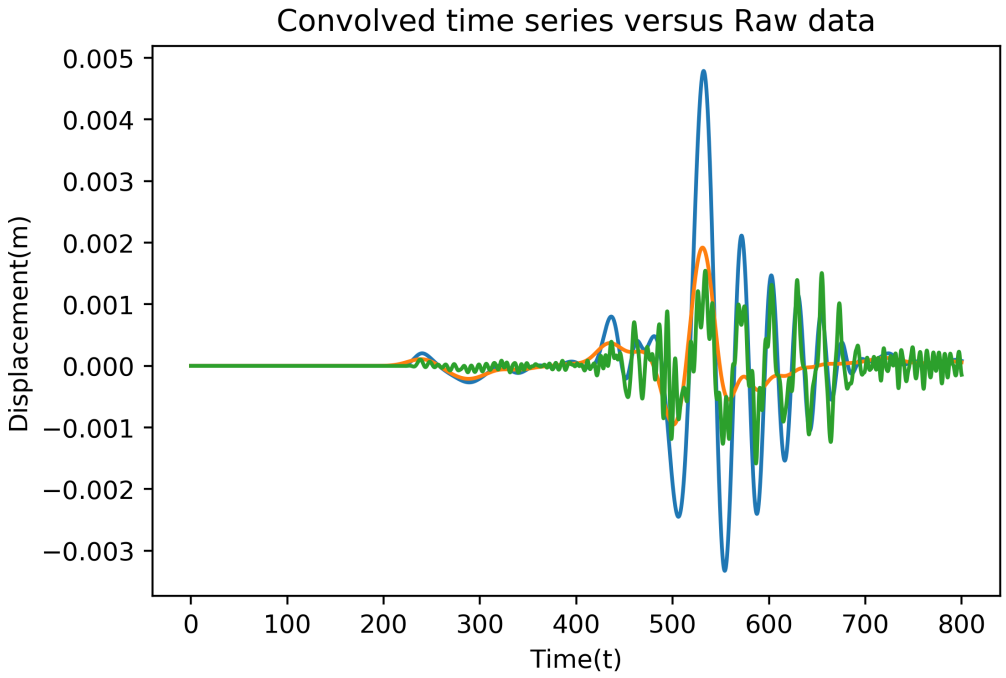


The plot of the Gaussian functions($t_H = 10, 20$):



I used the same time step dt as the seismogram data.

2. The raw data with two convoluted time series plot is included below:



As we can see, both time series somewhat shows resemblance to the raw data, which is good. However, when t_H is 10, it fluctuates a lot. When t_H is 20, the convoluted time series captures the data pattern much better. Therefore, if we want to improve the time series model, we could try to find a larger value of t_H to get the optimal representation of the data.