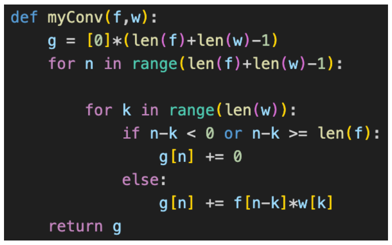
I would like to request to use the one-time free extension.
No collaborators for any exercises.
Exercise 1:

1. A discrete convolution is given by the following formula:

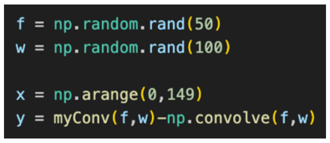
$$g_n = \left[\sum_{k=-\infty}^{\infty} f_{n-k} w_k \right] \Delta$$

   Assuming all indexes start from 0, this means for two arrays of length N_f and N_w, as long as $n < N_f + N_w$, there will be at least one value in the summation that could be non-zero. Taking $N_f \geq N_w$, or swapping f and w if the inverse, for $n = 0$, the $k = 0$ case could be non-zero. Then as n increases, while $n < N_f$, the $k = 0$ case continues to hold potential non-zero values. As soon as $n \geq N_f$, subtracting k from n by the minimum amount to make $n - k < N_f$, will also continue to hold potential non-zero values as long as $k < N_w$. Logically, this can continue all the way until $k = N_w - 1$.This means that for two arrays of length N_f and N_w, the resulting discrete convolution array has a maximum possible length of $N_g = N_f + N_w - 1$ non-zero values.

2. 

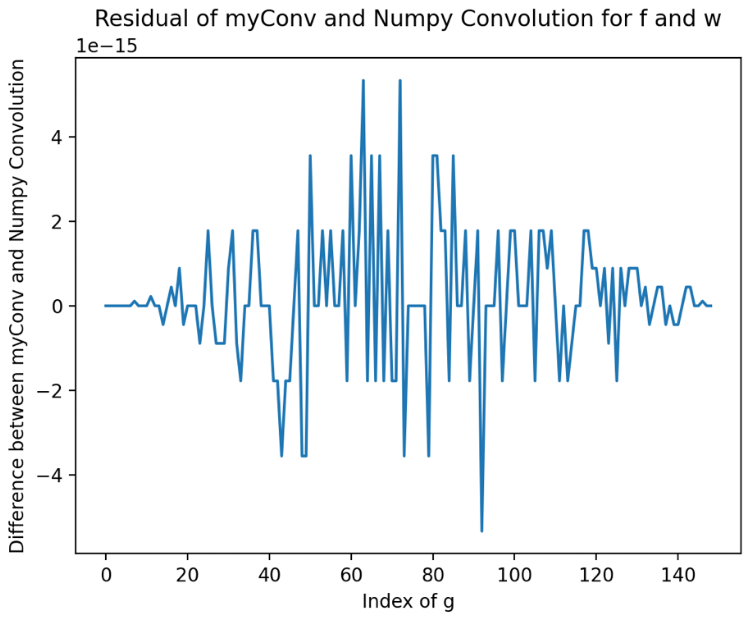
```
def myConv(f,w):
    g = [0]*(len(f)+len(w)-1)
    for n in range(len(f)+len(w)-1):

        for k in range(len(w)):
            if n-k < 0 or n-k >= len(f):
                g[n] += 0
            else:
                g[n] += f[n-k]*w[k]
    return g
```

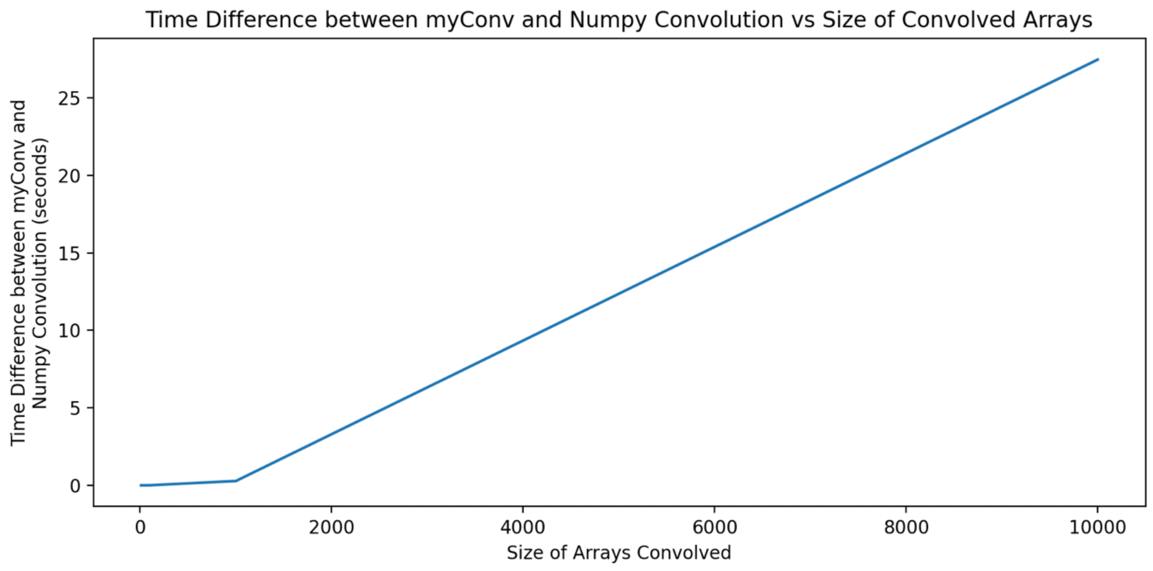
3. 

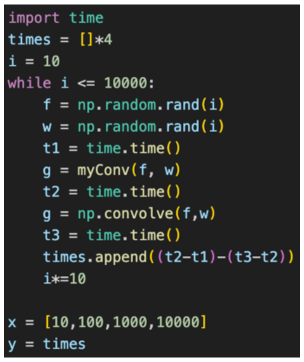
```
f = np.random.rand(50)
w = np.random.rand(100)

x = np.arange(0,149)
y = myConv(f,w)-np.convolve(f,w)
```



4.





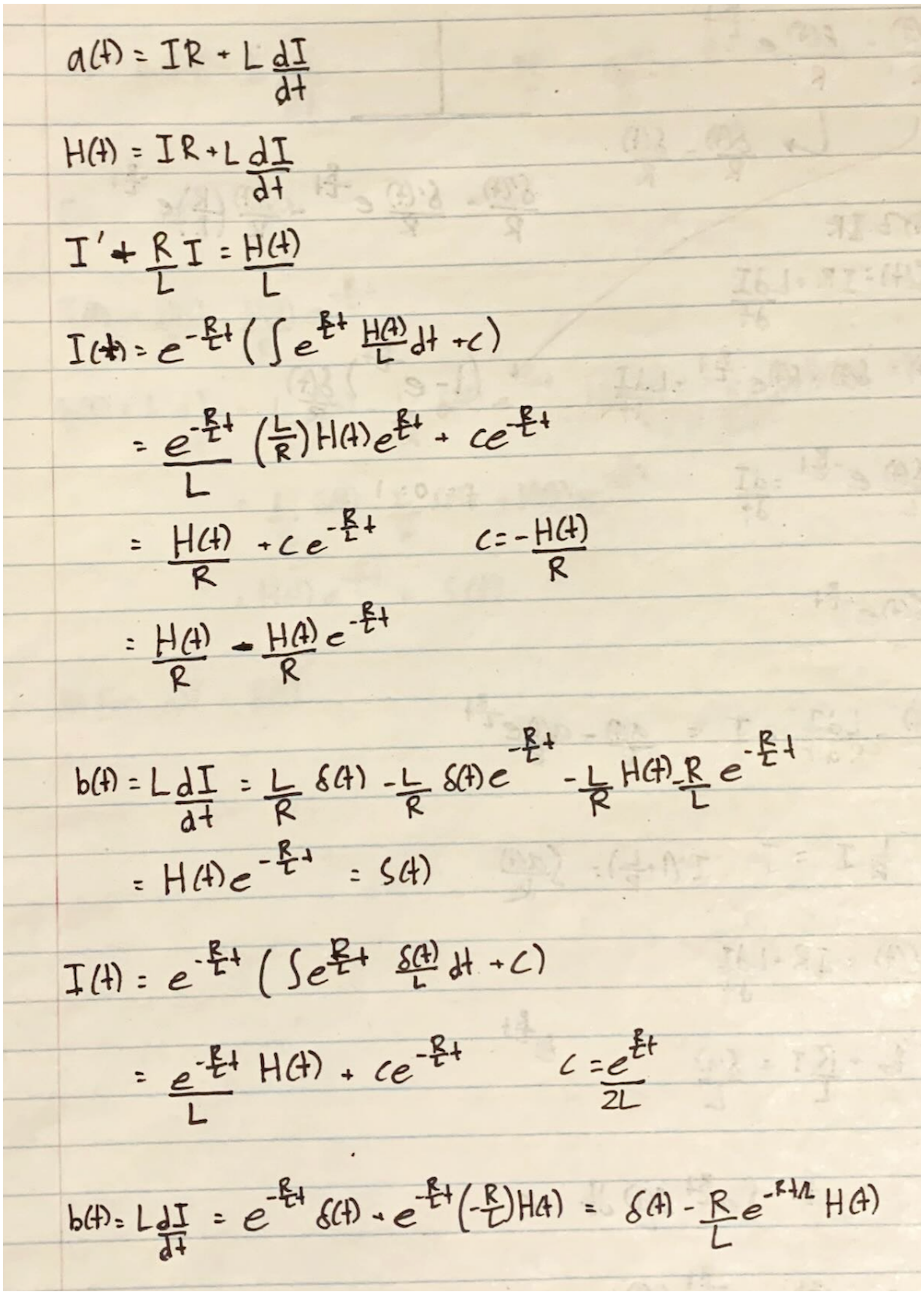
```
import time
times = []*4
i = 10
while i <= 10000:
    f = np.random.rand(i)
    w = np.random.rand(i)
    t1 = time.time()
    g = myConv(f, w)
    t2 = time.time()
    g = np.convolve(f,w)
    t3 = time.time()
    times.append((t2-t1)-(t3-t2))
    i*=10

x = [10,100,1000,10000]
y = times
```

For large arrays, the Numpy convolution is clearly faster than my own function. There are many possible explanations for this, but most generally it is likely the built into function of Numpy uses different methods of optimization as opposed to the most simplistic approach I used.

Exercise 2:

1.



$$a(t) = IR + L\frac{dI}{dt}$$

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

$$I' + \frac{R}{L}I = \frac{H(t)}{L}$$

$$I(t) = e^{-\frac{R}{L}t}\left(\int e^{\frac{R}{L}t}\,\frac{H(t)}{L}\,dt + c\right)$$

$$= \frac{e^{-\frac{R}{L}t}}{L}\left(\frac{L}{R}\right)H(t)e^{\frac{R}{L}t} + ce^{-\frac{R}{L}t}$$

$$= \frac{H(t)}{R} + ce^{-\frac{R}{L}t} \qquad c = -\frac{H(t)}{R}$$

$$= \frac{H(t)}{R} - \frac{H(t)}{R}e^{-\frac{R}{L}t}$$

$$b(t) = L\frac{dI}{dt} = \frac{L}{R}\delta(t) - \frac{L}{R}\delta(t)e^{-\frac{R}{L}t} - \frac{L}{R}H(t)\,{-\frac{R}{L}}\,e^{-\frac{R}{L}t}$$

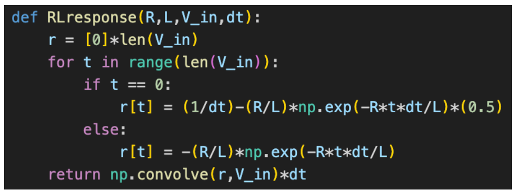
$$= H(t)e^{-\frac{R}{L}t} = S(t)$$

$$I(t) = e^{-\frac{R}{L}t}\left(\int e^{\frac{R}{L}t}\,\frac{\delta(t)}{L}\,dt + c\right)$$

$$= \frac{e^{-\frac{R}{L}t}}{L}H(t) + ce^{-\frac{R}{L}t} \qquad c = \frac{e^{\frac{R}{L}t}}{2L}$$

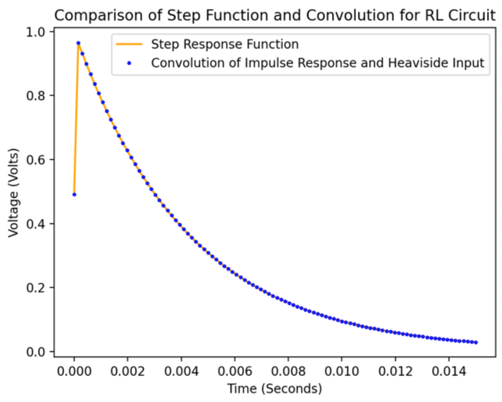
$$b(t) = L\frac{dI}{dt} = e^{-\frac{R}{L}t}\delta(t) + e^{-\frac{R}{L}t}\left(-\frac{R}{L}\right)H(t) = \delta(t) - \frac{R}{L}e^{-Rt/L}H(t)$$

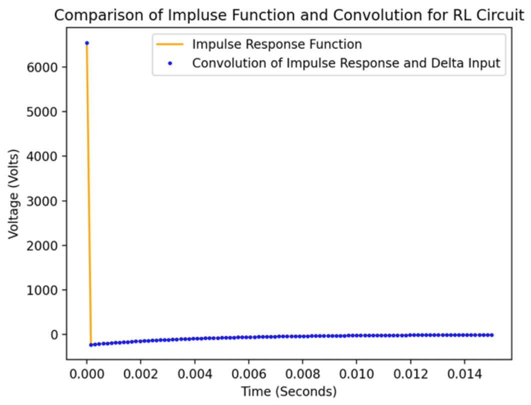
2.

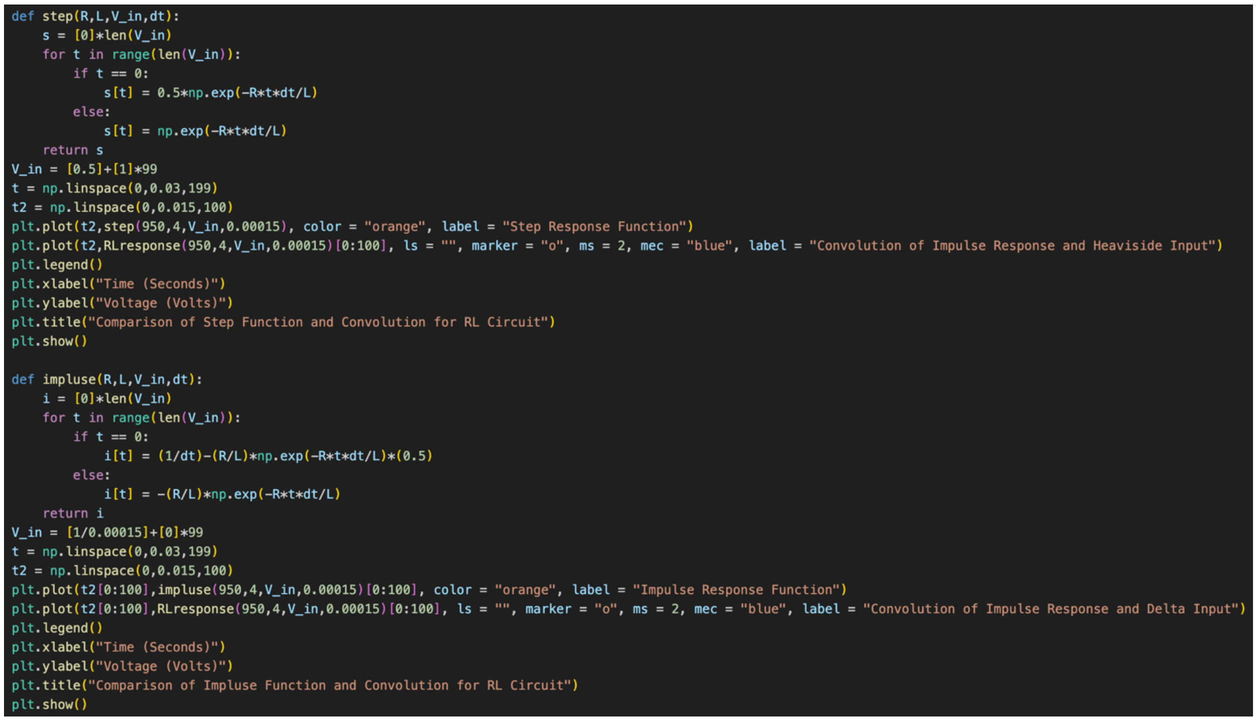


```python
def RLresponse(R,L,V_in,dt):
    r = [0]*len(V_in)
    for t in range(len(V_in)):
        if t == 0:
            r[t] = (1/dt)-(R/L)*np.exp(-R*t*dt/L)*(0.5)
        else:
            r[t] = -(R/L)*np.exp(-R*t*dt/L)
    return np.convolve(r,V_in)*dt
```

3.





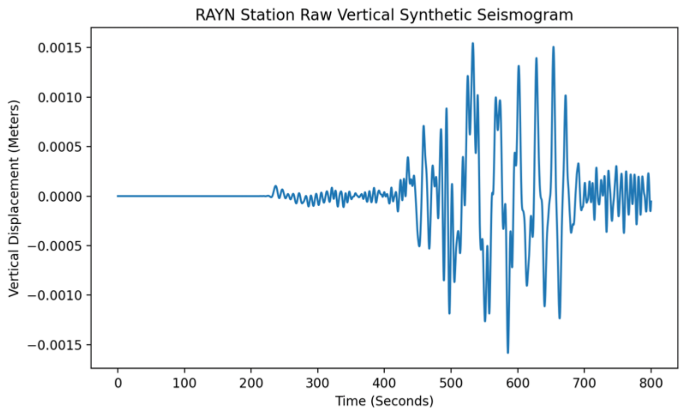


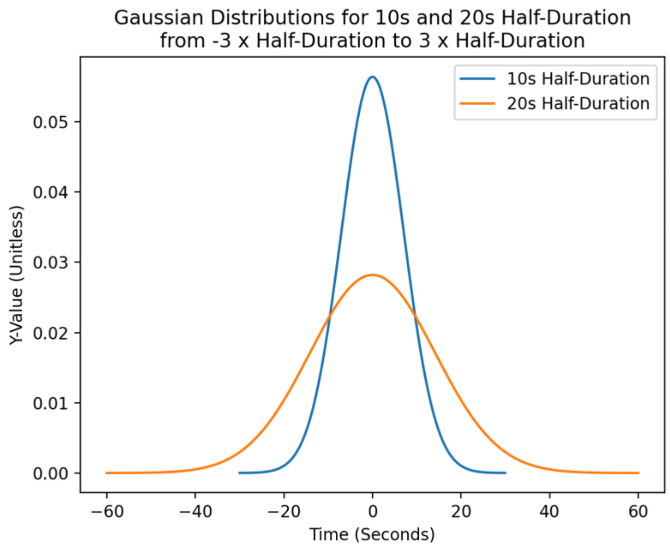
```python
def step(R,L,V_in,dt):
    s = [0]*len(V_in)
    for t in range(len(V_in)):
        if t == 0:
            s[t] = 0.5*np.exp(-R*t*dt/L)
        else:
            s[t] = np.exp(-R*t*dt/L)
    return s
V_in = [0.5]+[1]*99
t = np.linspace(0,0.03,199)
t2 = np.linspace(0,0.015,100)
plt.plot(t2,step(950,4,V_in,0.00015), color = "orange", label = "Step Response Function")
plt.plot(t2,RLresponse(950,4,V_in,0.00015)[0:100], ls = "", marker = "o", ms = 2, mec = "blue", label = "Convolution of Impulse Response and Heaviside Input")
plt.legend()
plt.xlabel("Time (Seconds)")
plt.ylabel("Voltage (Volts)")
plt.title("Comparison of Step Function and Convolution for RL Circuit")
plt.show()

def impluse(R,L,V_in,dt):
    i = [0]*len(V_in)
    for t in range(len(V_in)):
        if t == 0:
            i[t] = (1/dt)-(R/L)*np.exp(-R*t*dt/L)*(0.5)
        else:
            i[t] = -(R/L)*np.exp(-R*t*dt/L)
    return i
V_in = [1/0.00015]+[0]*99
t = np.linspace(0,0.03,199)
t2 = np.linspace(0,0.015,100)
plt.plot(t2[0:100],impluse(950,4,V_in,0.00015)[0:100], color = "orange", label = "Impulse Response Function")
plt.plot(t2[0:100],RLresponse(950,4,V_in,0.00015)[0:100], ls = "", marker = "o", ms = 2, mec = "blue", label = "Convolution of Impulse Response and Delta Input")
plt.legend()
plt.xlabel("Time (Seconds)")
plt.ylabel("Voltage (Volts)")
plt.title("Comparison of Impluse Function and Convolution for RL Circuit")
plt.show()
```

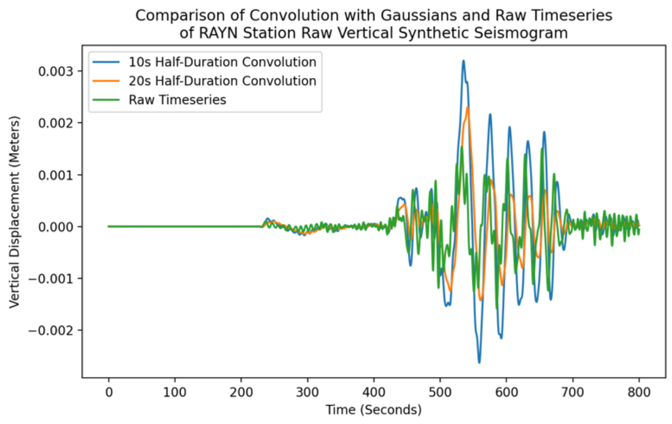
Exercise 3:

1.





2.



The most obvious difference between the 10s and 20s half-duration convolutions is that the 20s half-duration generally has a lower amplitude over the range of values. This makes sense as the 20s half-duration gaussian takes more from values surrounding the data at a specific point in time, while the 10s half-duration gaussian is more influenced by the closest values at a specific point.