Dang_Kevin_Lab1

February 5, 2020

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

1 A Discrete Convolution Program (5 pts)

Write a discrete convolution function `myConv` that convolves two arrays $\{f_i, i = 0, \ldots, N_f - 1\}$ and $\{w_j, j = 0, \ldots, N_w - 1\}$ to obtain an output time series $\{g_n\}$. For simplicity, assume a fixed sampling interval $\Delta = 1$, and further, that f and w are 0 outside of their sampled regions.

1. How long is $\{g_n\}$? In other words, how many non-zero points can it have? Justify your answer.
2. Please copy and paste your function `g = myConv(f, w)` to the PDF report.
3. Provide a test to convince yourself (and me) that your function agrees with `numpy.convolve`. For example, generate two random timeseries f, w with $N_f = 50$, $N_w = 100$, drawing each element from $U[0, 1]$, and plot the difference between your function's output and numpy's. Include the code for your test in the PDF report.
4. Compare the speed of your `myConv` function to the NumPy function. Provide a plot of the comparison, and include your python code in the PDF report. Is your function faster or slower than the NumPy function? Can you suggest why that is the case?

Hint: For the speed test part, make up your own f_i and w_j time series, and for simplicity, study the cases of $N_f = N_w = 10, 100, 1000, 10000$. To accurately time each computation of the convolution function, import the time module and place calls to `time.time` around your code:

```
import time
t1 = time.time()
g = myConv(f, w)
t2 = time.time()
print(t2-t1)
```

Alternatively, use the `timeit` module:

```
import timeit
print(timeit.timeit('g = myConv(f, w)', number=10000))
```

1.0.1 Part 1

$\{g_n\}$ has a length of $N_f + N_w - 1$

The discrete convolution is given by $\{g_n\} = [\sum_{k=-\infty}^{\infty} w_k f_{n-k}]\Delta$

For a given n we have:

$$0 \leq n - k \leq N_f - 1 \Rightarrow n - N_f + 1 \leq k \leq n$$
$$0 \leq k \leq N_w - 1$$

This results in $max(0, n - N_f + 1) \leq k \leq min(n, N_w - 1)$, however if $max(0, n - N_f + 1) > min(n, N_w - 1) \Rightarrow n < 0, n > N_f + N_w - 2$ then $\{g_n\} = 0$.

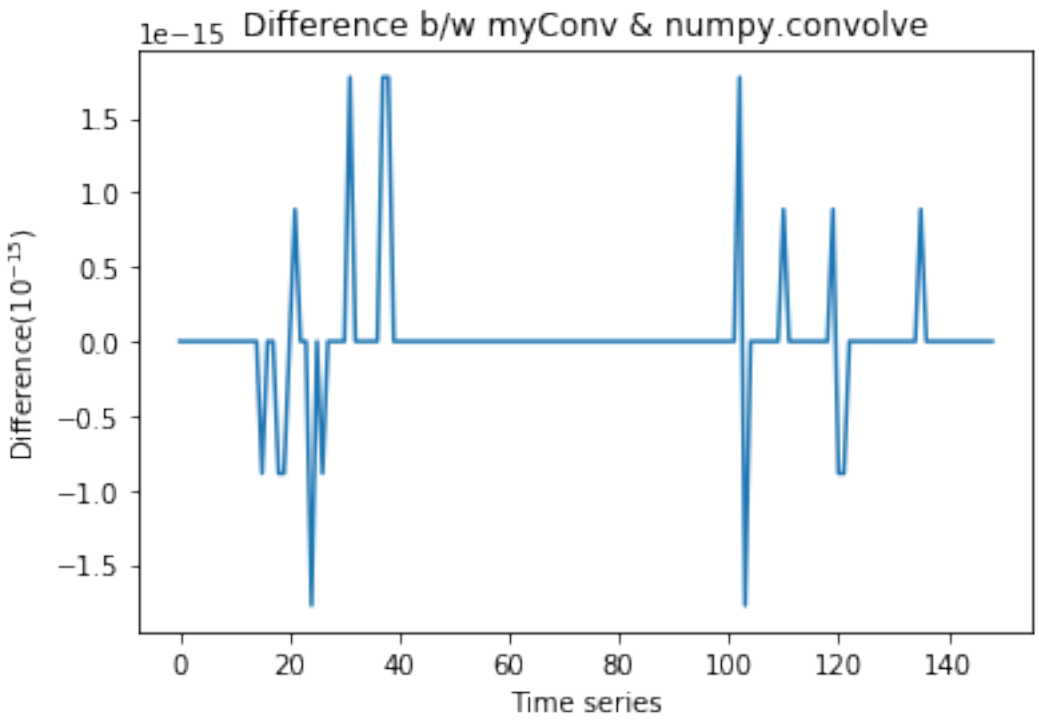
To obtain $\{g_n\} \neq 0$, we have $0 \leq n \leq N_f + N_w - 2 \Rightarrow$ the length of $\{g_n\}$ is $N_f + N_w - 1$

1.0.2 Part 2

[2]:
```
def myConv(f, w):
    # Assign the shorter list as f and reverse it
    if len(f) > len(w):
        f, w = w, f
    e = f[::-1]
    j = len(f)
    k = len(w)
    g = [] # for appending
    # Zero padding before and after to compute dot product correctly
    v = np.pad(w, (j-1, j-1), 'constant')
    for i in range(j+k-1):
        g.append(np.dot(e,v[i:i+j]))
    return np.array(g)
```

1.0.3 Part 3

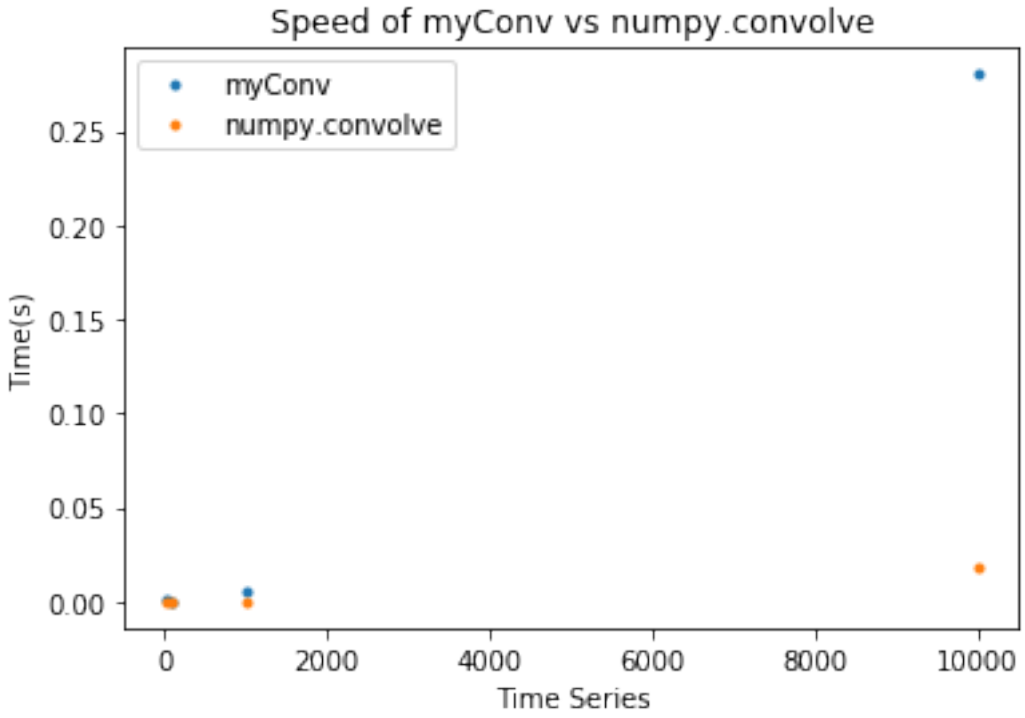
[3]:
```
f = np.random.uniform(0,1,50)
w = np.random.uniform(0,1,100)
g = myConv(f, w)
h = np.convolve(f, w)

plt.plot(g-h)
plt.xlabel('Time series')
plt.ylabel('Difference($10^{-15}$)')
plt.title('Difference b/w myConv & numpy.convolve')
plt.show()
```



1.0.4 Part 4

[4]:
```
import time

myfunc = []
numpy = []
N = [10, 100, 1000, 10000]
for n in N:
    f = np.random.uniform(0,1,n)
    w = np.random.uniform(0,1,n)

    t1 = time.time()
    g = myConv(f, w)
    t2 = time.time()
    myfunc.append(t2-t1)

    t3 = time.time()
    h = np.convolve(f, w)
    t4 = time.time()
    numpy.append(t4-t3)

plt.plot(N,myfunc,'.',label='myConv')
plt.plot(N,numpy,'.',label='numpy.convolve')
```

```
plt.xlabel('Time Series')
plt.ylabel('Time(s)')
plt.title('Speed of myConv vs numpy.convolve')
plt.legend()
plt.show()
```



From the above plot we can see that my function "myConv" is slower than the NumPy function. My function contains a for loop which takes a longer time to run, while the NumPy function might have a more efficient way of performing the convolution.

2 Simple Physical System: RL Circuit Response (8 pts)

Consider a simple physical system consisting of a resistor (with resistance `R`) and an inductor (with inductance `L`) in series. We apply an input voltage $a(t)$ across the pair in series, and measure the output voltage $b(t)$ across the inductor alone. For this linear system,

1. Show analytically that its step response (i.e., the $b(t)$ we obtain when the input voltage $a(t) = H(t)$, the Heaviside function) is given by

$$S(t) = e^{-Rt/L}H(t),$$

and its impulse response (i.e., the output voltage $b(t)$ when $a(t) = \delta(t)$) is given by

$$R(t) = \delta(t) - \frac{R}{L} e^{-Rt/L} H(t).$$

Hint: Construct and solve the ODE relating the voltages under consideration. Consider the two $b(t)$ choices to derive $S(t)$ and $R(t)$. Formulas $\frac{d}{dt}H(t) = \delta(t)$ and $\delta(t)f(t) = \delta(t)f(0)$ may help.

2. Discretize the impulse response $R(t)$ function, realizing that $H(t)$ should be discretized as

$$H = [0.5, 1, 1, \dots],$$

and $\delta(t)$ should be discretized as

$$D = [1/dt, 0, 0, \dots].$$

Take advantage of your `myConv` function, or the NumPy built-in function `convolve`, and write your own Python function `V_out = RLresponse(R,L,V_in,dt)` to take an input series V_{in} sampled at $\Delta = dt$, and calculate the output series V_{out} sampled by the same dt. Please paste your Python function here. (Hint: here Δ may not be 1, so remember to build the multiplication of Δ into your convolution function.)

3. Using $R = 850\Omega$, $L = 2H$, and sampling period $dt = 0.20$ ms, test your RL-response function with $\{H_n\}$ series (discretized $H(t)$) as input, and plot the output time series (as circles) on top of the theoretical curve $S(t)$ given by part 1 (as a solid line). Repeat this for $\{D_n\}$ (discretized $\delta(t)$) and $R(t)$. Make the time range of the plots 0 to at least 20 ms. Please list your Python code here.

2.0.1 Part 1

We have:
$V_R = RI(t)$ and $V_L = L\frac{dI}{dt}$
Then
$a(t) = V_R + V_L = RI(t) + L\frac{dI}{dt}$ and $b(t) = L\frac{dI}{dt} = a(t) - RI(t)$
This leads to:

$$\frac{dI}{dt} = \frac{a(t)}{L} - \frac{RI(t)}{L}$$

$$\frac{dI}{dt} + \frac{RI(t)}{L} = \frac{a(t)}{L} \qquad \text{integrating factor } \mu(t) = e^{\int R/L\, dt} = e^{Rt/L}$$

$$e^{Rt/L}\frac{dI}{dt} + e^{Rt/L}\frac{RI(t)}{L} = e^{Rt/L}\frac{a(t)}{L}$$

$$\int [e^{Rt/L} I(t)]' dt = \int e^{Rt/L}\frac{a(t)}{L} dt$$

$$e^{Rt/L} I(t) = \int e^{Rt/L}\frac{a(t)}{L} dt$$

$$I(t) = \int e^{R(t'-t)/L}\frac{a(t')}{L} dt' \quad \text{this is a convolution}$$

Plugging in a(t) = H(t) we get

$$\begin{aligned} I(t) &= \int_{-\infty}^{t} e^{R(t'-t)/L} \frac{H(t')}{L} dt' \\ &= \frac{1}{L} \int_{0}^{t} e^{R(t'-t)/L} dt' \\ &= \frac{1}{L} e^{-Rt/L} \left[\frac{L}{R} e^{Rt'/L} \right]_0^t \\ &= \frac{1}{L} e^{-Rt/L} \left[\frac{L}{R} e^{Rt/L} - \frac{L}{R} \right] \\ I(t) &= \frac{1}{R} - \frac{1}{R} e^{-Rt/L} \end{aligned}$$

So we have $b(t) = L\frac{dI}{dt} = e^{-Rt/L}$. But this is only true when t > 0, since b(t) = 0 for t < 0 so we need to multiply it by the Heaviside function. $\Rightarrow b(t) = e^{-Rt/L}H(t)$

Thus, the step response is given by $S(t) = e^{-Rt/L}H(t)$

For the case of a(t) = $\delta(t)$, we can differentiate S(t) to find R(t) since we know that $\frac{d}{dt}H(t) = \delta(t)$, then using product rule we have that $\frac{d}{dt}S(t) = \delta(t) - \frac{R}{L}e^{-Rt/L}H(t)$

Therefore, the impulse response is given by $R(t) = \delta(t) - \frac{R}{L}e^{-Rt/L}H(t)$

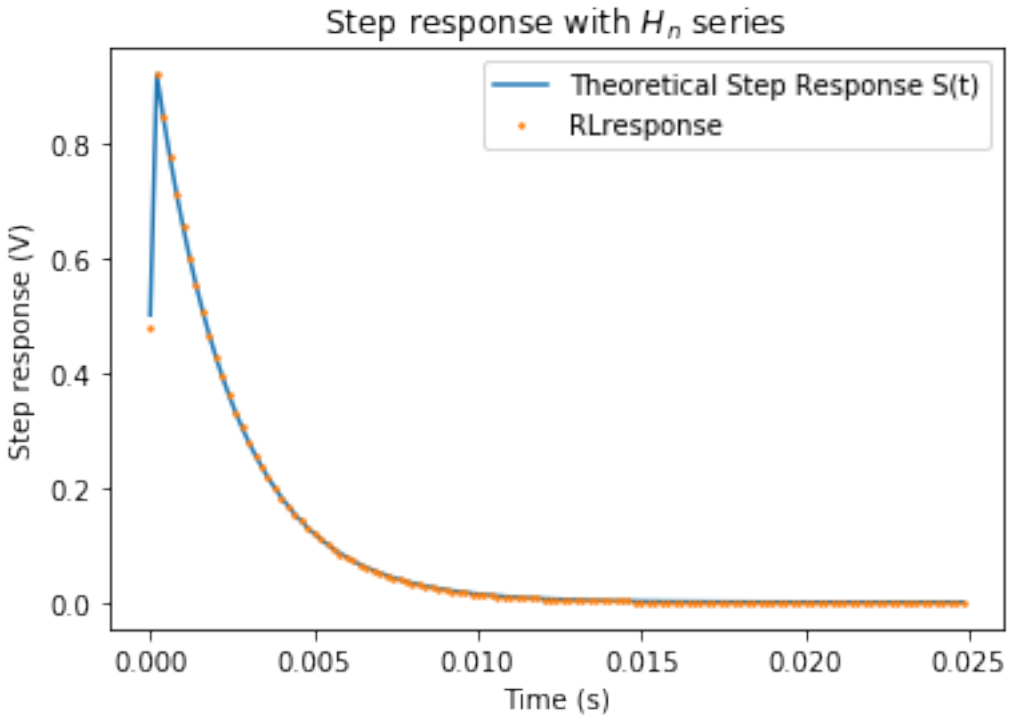
2.0.2 Part 2

[5]:
```
def RLresponse(R,L,V_in,dt):
    n = len(V_in)
    D = np.zeros(n)
    D[0] = 1/dt
    H = np.ones(n)
    H[0] = 0.5
    t = dt*np.arange(0,n)
    R_t = D - (R/L)*np.exp(-R*t/L)*H
    V_out = dt*np.convolve(V_in,R_t)
    return V_out[:n]
```

2.0.3 Part 3

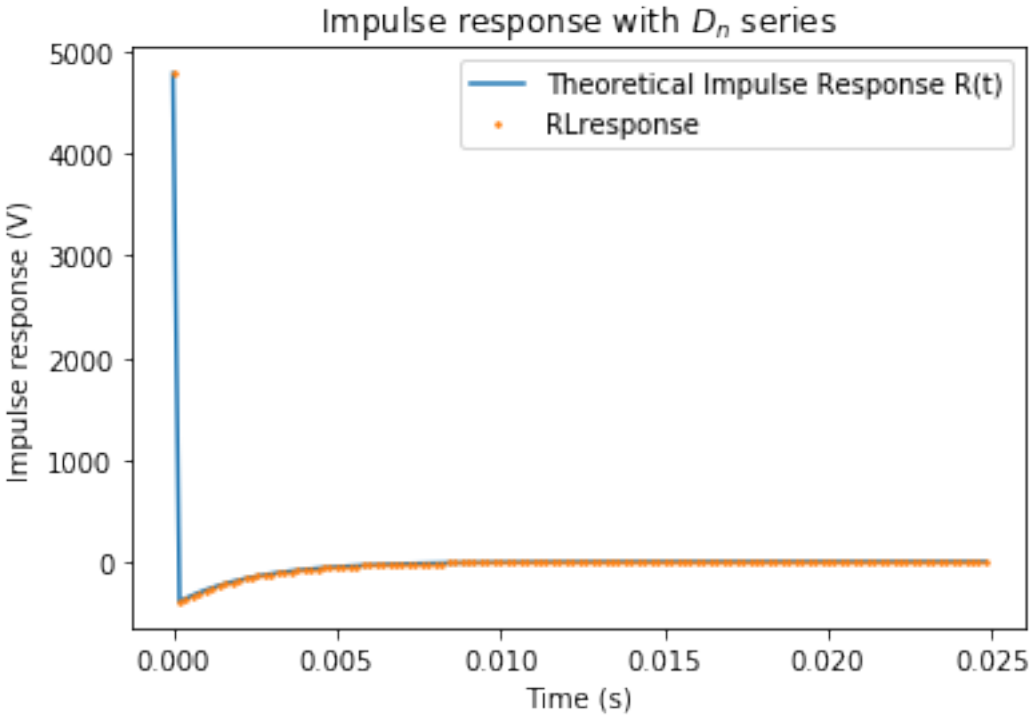
[6]:
```
def S_t(t,H,R,L):
    return np.exp(-R*t/L)*H

dt = 0.2/1000
t = np.arange(0,25/1000,dt)
H = np.heaviside(t,0.5)
R = 850
L = 2

S_t = S_t(t,H,R,L)
V_out = RLresponse(R,L,H,dt)
```

```
plt.plot(t,S_t,label='Theoretical Step Response S(t)')
plt.plot(t,V_out,'.',markersize='3',label='RLresponse')
plt.xlabel('Time (s)')
plt.ylabel('Step response (V)')
plt.legend()
plt.title('Step response with $H_n$ series')
plt.show()
```



[7]:
```
def R_t(t,D,H,R,L):
    return D - (R/L)*np.exp(-R*t/L)*H

dt = 0.2/1000
t = np.arange(0,25/1000,dt)

D = np.zeros(len(t))
D[0] = 1/dt
H = np.heaviside(t,0.5)
R = 850
L = 2

R_t = R_t(t,D,H,R,L)
V_out2 = RLresponse(R,L,D,dt)

plt.plot(t,R_t,label='Theoretical Impulse Response R(t)')
```

```
plt.plot(t,V_out2,'.',markersize='3',label='RLresponse')
plt.xlabel('Time (s)')
plt.ylabel('Impulse response (V)')
plt.legend()
plt.title('Impulse response with $D_n$ series')
plt.show()
```



3 Convolution of Synthetic Seismograms (5 pts)

Numerical simulations of seismic wave propagation can now be routinely done for global and regional earthquakes. For a recent southern Pakistan earthquake (Jan 18, 2011, 20:23:33 UTC), raw vertical synthetic seismogram (i.e., displacement field simulated at a seismic station) for station `RAYN` (Ar Rayn, Saudi Arabia) is provided (`RAYN.II.LHZ.sem`). A common practice in seismology is to convolve synthetic seismograms with a Gaussian function

$$g(t) = \frac{1}{\sqrt{\pi} t_H} e^{-(t/t_H)^2}$$

to reflect either the time duration of the event or the accuracy of the numerical simulation.

1. Provide two plots. Plot 1: the raw synthetic seismogram for station `RAYN` between 0 and 800 seconds. Plot 2: Gaussian functions with half duration $t_H = 10$ sec and $t_H = 20$ sec (include a legend). For the gaussians, use the same timestep dt as the seismogram data.

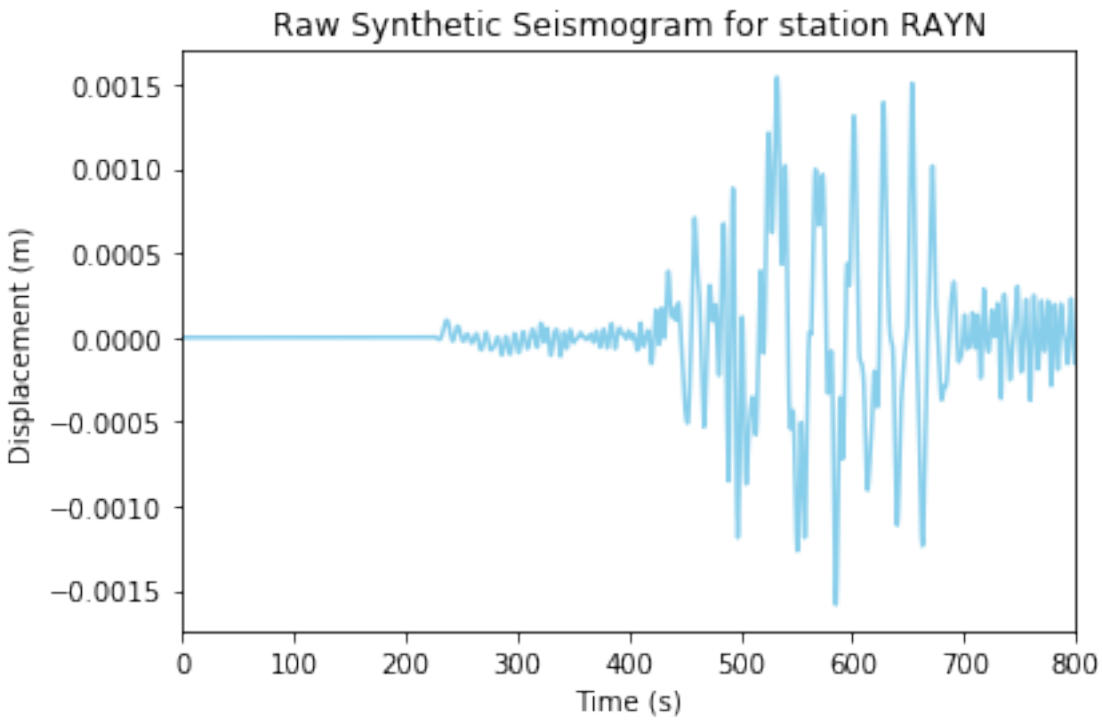
2. Use numpy's convolve function to convolve the raw timeseries with a Gaussian function (both $t_H = 10$ and $t_H = 20$ cases). Plot the raw data and the two convolved time series between 0 and 800 seconds on the same graph (include a legend) and comment on your results.

Hints

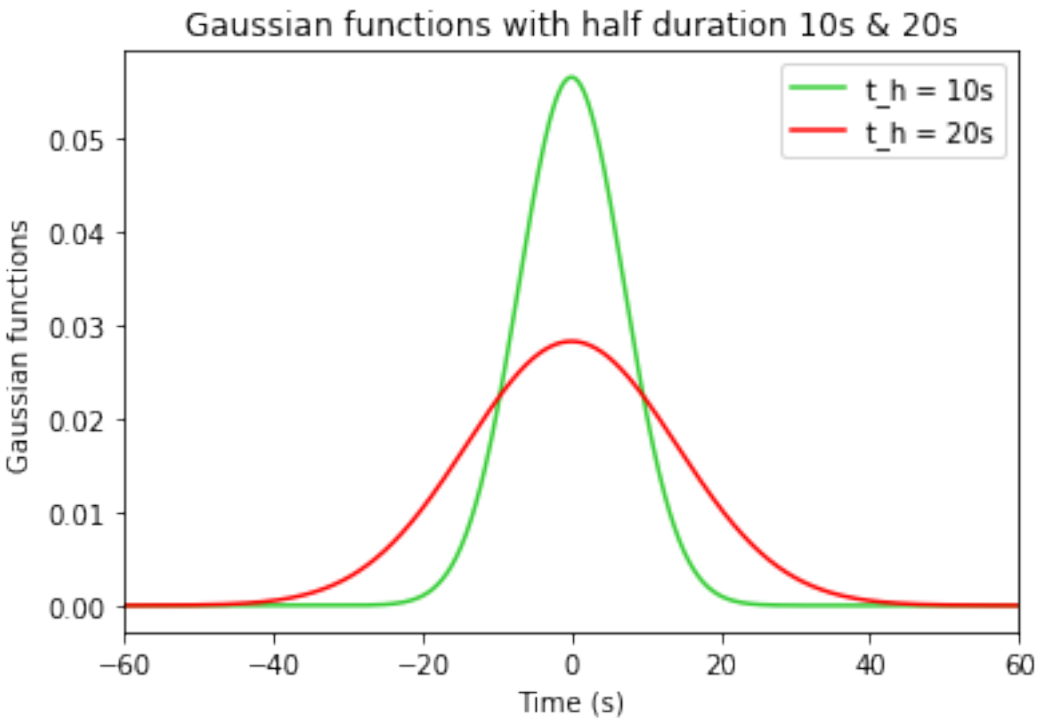
- The raw synthetics `RAYN.II.LHZ.sem` is given as a text file with two columns: time in seconds and displacement in meters.
- Gaussian functions quickly decay to zero beyond $[-3t_H, 3t_H]$, therefore it is sufficient to sample $g(t)$ within this interval.
- Use `mode='same'` when calling numpy convolve to truncate the convolution to the max of the supplied arrays (i.e. length of the raw timeseries in our case). This is convenient, since we want to compare the convolution output to the original timeseries. Alternatively, use the default mode (`'full'`) and truncate the output manually.
- As a check for part 2, ensure that your convolved timeseries is aligned with (or "overlaps") the raw data timeseries.

3.0.1 Part 1

[8]:
```
t, y = np.loadtxt('RAYN.II.LHZ.sem',unpack=True)

plt.plot(t,y,color='skyblue')
plt.xlim(0,800)
plt.xlabel('Time (s)')
plt.ylabel('Displacement (m)')
plt.title('Raw Synthetic Seismogram for station RAYN')
plt.show()
```

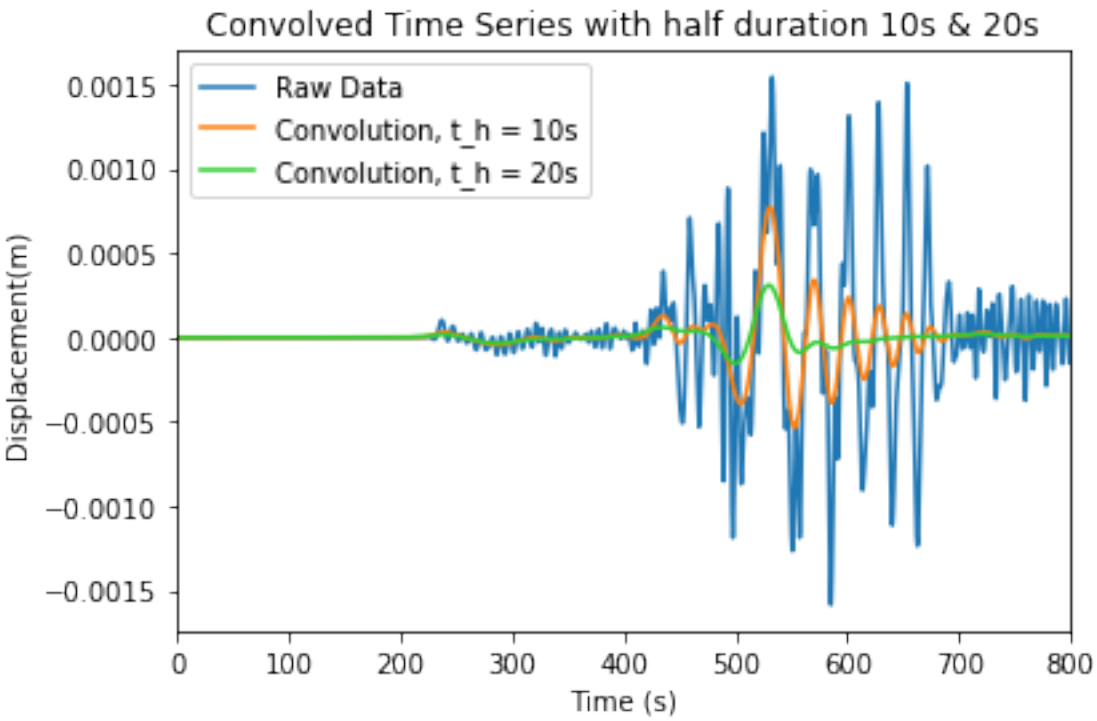


[9]:
```
def g(t,h):
    return np.exp(-(t/h)**2) / (np.sqrt(np.pi)*h)

dt = t[1]-t[0]
t2 = np.arange(-60,60,dt)

plt.plot(t2,g(t2,10),color='limegreen',label='t_h = 10s')
plt.plot(t2,g(t2,20),color='red',label='t_h = 20s')
plt.xlim(-60,60)
plt.xlabel('Time (s)')
plt.ylabel('Gaussian functions')
plt.title('Gaussian functions with half duration 10s & 20s')
plt.legend()
plt.show()
```



3.0.2 Part 2

[10]:
```
plt.plot(t,y,label='Raw Data')
plt.plot(t,dt*np.convolve(y,g(t2,10),mode='same'),label='Convolution, t_h =␣
↪10s')
plt.plot(t,dt*np.
↪convolve(y,g(t2,20),mode='same'),color='limegreen',label='Convolution, t_h =␣
↪20s')
plt.xlim(0,800)
plt.xlabel('Time (s)')
plt.ylabel('Displacement(m)')
plt.title('Convolved Time Series with half duration 10s & 20s')
plt.legend()
plt.show()
```



From the above plot we can see that the convolved times series for the half duration 10 seconds has smaller amplitudes than the raw data, and it is also much smoother. The same can be said for the convolved time series of half duration 20 seconds, it has an even smaller amplitude and has a smoother curve. This makes sense since the raw time series has been convolved with Gaussian functions which decay to 0 quickly.