Stanford University
EE 102A: Signal Processing and Linear Systems I
Summer 2022
Instructor: Ethan M. Liang

Homework 3, due Friday, July 15

DT LTI System Analysis

1. A savings account with fixed interest rate is described by a difference equation

$$y[n] = x[n] + a\,y[n-1],$$

where $x[n]$ and $y[n]$ are the deposit and balance in month *n*, and $a = 1+\alpha$, where α is the monthly interest rate. If \$1 is deposited in month 0 and no further deposits are made, $x[n] = \delta[n]$, the value of the account in month *n* is

$$h[n] = a^n u[n] = (1+\alpha)^n u[n],$$

which is the impulse response of the account. If \$1 is deposited each month, starting in month 0, $x[n] = u[n]$, the value of the account in month *n* (assuming $\alpha \neq 0$) is

$$s[n] = \frac{1-a^{n+1}}{1-a} u[n] = \frac{(1+\alpha)^{n+1}-1}{\alpha} u[n],$$

which is the step response of the account. Suppose the monthly interest rate is $\alpha = 0.005$ (6% annual rate), and you deposit \$100/month for 120 months (*n* = 0,…,119). What is the balance for *n* = 240? *Hint*: express the deposit $x[n]$ as a difference between two scaled, shifted step functions, allowing you to express the balance $y[n]$ as a difference between two scaled, shifted step responses.

Properties of Convolution

2. *Time-reversed signals*. This is true for both DT and CT convolution:

$$\text{If } y[n] = x[n] * h[n], \text{ then } y[-n] = x[-n] * h[-n]$$

$$\text{If } y(t) = x(t) * h(t), \text{ then } y(-t) = x(-t) * h(-t).$$

Prove it for the CT case.

Important CT LTI Systems

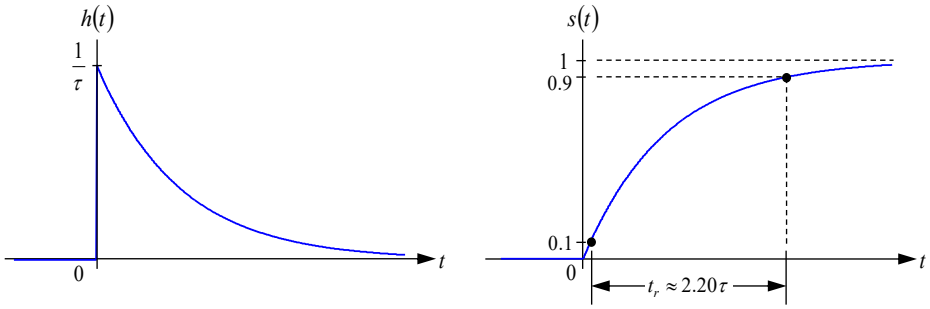
3. *First-order lowpass and highpass filters*. A *first-order lowpass filter* with input $x(t)$ and output $y(t)$ is described by a differential equation

$$\tau \frac{dy}{dt} + y(t) = x(t).$$

It has impulse and step responses

$$h(t)=\frac{1}{\tau}e^{-\frac{t}{\tau}}u(t) \text{ and } s(t)=\left(1-e^{-\frac{t}{\tau}}\right)u(t),$$

which are shown here.



a. Show that the *rise time* required for the step response to increase from 10% to 90% of its maximum value is $t_r \approx 2.2\tau$.

A *first-order highpass filter* is described by a differential equation

$$\frac{dy}{dt}+\frac{1}{\tau}y(t)=\frac{dx}{dt}.$$

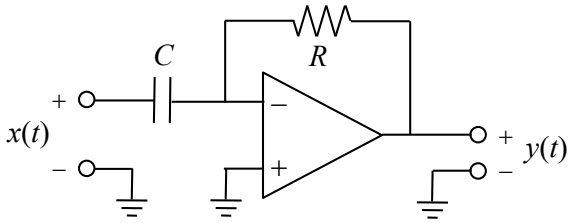
b. In lecture, we stated its impulse response to be

$$h(t)=\delta(t)-\frac{1}{\tau}e^{-\frac{t}{\tau}}u(t).$$

Verify that this $h(t)$ satisfies the differential equation with input $x(t)=\delta(t)$, output $y(t)=h(t)$, and zero initial condition $y(t)=0,\ t<0$. Sketch $h(t)$.

c. Derive an expression for the step response $s(t)$. Sketch $s(t)$.

4. *Differentiator and integrator*. If any of the following questions have been answered in the EE 102A lectures, feel free to state that and give the answer without proof. For a more detailed discussion of these systems, see *EE 102B Course Reader*, Chapter 5. A *differentiator* implemented using an operational amplifier (op amp) is shown here.



Given an input $x(t)$, ideally, the output is given by

$$y(t) = -RC\frac{dx}{dt}.$$

It is impossible to realize a perfect differentiator, as that would require the op amp to have infinite gain-bandwidth product and infinite slew rate. Here we assume ideality and ignore the factor $-RC$ so the output is given by

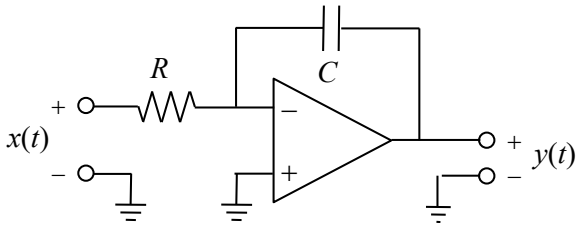
$$y(t) = \frac{dx}{dt}.$$

a. What is the impulse response $h(t)$?

b. What is the step response $s(t)$?

c. Explain how the result of part (b) allows us to determine whether the ideal differentiator is a bounded-input bounded-output (BIBO)-stable system.

An *integrator* implemented using an op amp is shown here.



Ideally, its output is given by

$$y(t) = -\frac{1}{RC}\int_{-\infty}^{t} x(t')dt'.$$

It is impossible to realize a perfect integrator, as that would require the op amp to have infinite d.c. gain and infinite output swing. Here we assume ideality and ignore the factor $-1/RC$ so the output becomes

$$y(t) = \int_{-\infty}^{t} x(t')dt'.$$

d. What is the impulse response $h(t)$?

e. What is the step response $s(t)$?

f. Explain how the result of part (e) allows us to determine whether the ideal integrator is a BIBO-stable system.

Representing CT LTI Systems by Convolutions

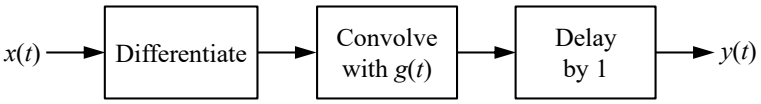
5. *Finite integration*. A CT LTI system H with input $x(t)$ yields an output

$$H[x(t)] = y(t) = \int_{t-1}^{t} x(t')dt'.$$

a. Find an impulse response $h(t)$ such that $H[x(t)] = y(t) = x(t) * h(t)$. Sketch $h(t)$. *Hint*: you can think of the finite integration as the difference between two infinite integrations with different upper limits, and express $h(t)$ as the difference between two shifted step functions.

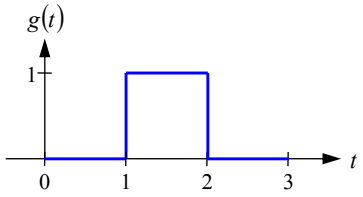
b. Is the system causal?

6. A CT system H with input $x(t)$ and output $y(t)$ is the cascade of three LTI systems, so the overall system is LTI.



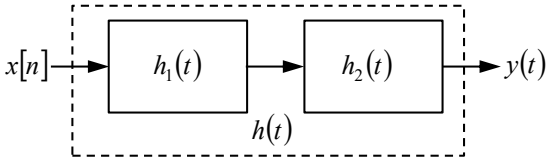
a. Find an impulse response $h(t)$ such that $H[x(t)] = y(t) = x(t) * h(t)$.

b. Suppose $g(t)$ is as shown. Give an expression for $h(t)$. Sketch $h(t)$.



Evaluating CT Convolution Integrals

7. LTI systems with impulse responses $h_1(t)$ and $h_2(t)$ are cascaded to form an LTI system with impulse response $h(t)$.



Both are first-order systems. Their impulse responses are

$$h_1(t) = \frac{1}{\tau_1} e^{-\frac{t}{\tau_1}} u(t)$$

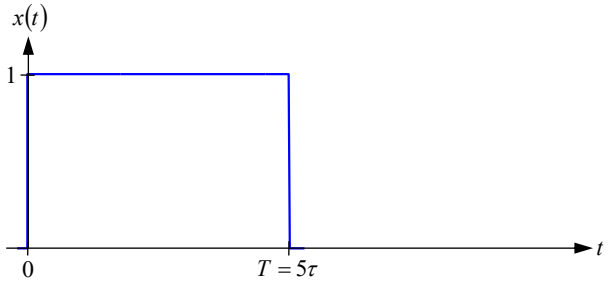
$$h_2(t) = \frac{1}{\tau_2} e^{-\frac{t}{\tau_2}} u(t),$$

where τ_1 and τ_2 are real and positive and $\tau_1 \neq \tau_2$.

a. Find an expression for $h(t) = h_1(t) * h_2(t)$. Simplify your expression for $h(t)$ so it is clearly a linear combination of $h_1(t)$ and $h_2(t)$. *Hint*: you can solve this by symbolic integration, without using "flip and drag".

b. Verify that $\int_{-\infty}^{\infty} h(t)dt = \left(\int_{-\infty}^{\infty} h_1(t)dt\right)\left(\int_{-\infty}^{\infty} h_2(t)dt\right)$, as expected from Homework 2 Problem 4.

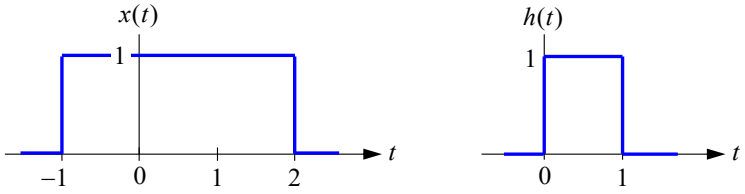
8. A rectangular pulse $x(t)$, shown below, is input to different LTI systems, each specified by an impulse response $h(t)$. For each system: (i) Find an expression for the output $y(t)$ by evaluating the convolution $x(t) * h(t)$. (ii) Make an approximate sketch of the output $y(t)$ without using a calculator or computer, assuming $T = 5$, $\tau = 1$. (iii) Verify that $\int_{-\infty}^{\infty} y(t)dt = \left(\int_{-\infty}^{\infty} x(t)dt\right)\left(\int_{-\infty}^{\infty} h(t)dt\right)$, as expected from Homework 2 Problem 4. You should compute $\int_{-\infty}^{\infty} x(t)dt$ and $\int_{-\infty}^{\infty} h(t)dt$ exactly. It is sufficient for you to estimate $\int_{-\infty}^{\infty} y(t)dt$ from your sketches.



a. First-order lowpass filter $h(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} u(t)$.

b. First-order highpass filter $h(t) = \delta(t) - \frac{1}{\tau} e^{-\frac{t}{\tau}} u(t)$.

9. A rectangular pulse $x(t)$ is input to a finite-duration integrator, which has an impulse response $h(t)$.



Evaluate the convolution $y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(t')h(t-t')dt'$ using the "flip and drag" method. *Hint*: first sketch $x(t')$ vs. t'. Then, for relevant choices of the time *t*, sketch $h(t-t')$ vs. t' and integrate $x(t')h(t-t')$ over t' to obtain $y(t)$.

Laboratory 3

In this homework, we will build on what you learned in Homework 2, where you wrote MATLAB functions and computed discrete-time (DT) convolutions, and we will extend these ideas to approximating continuous-time (CT) convolutions. Then, we will apply these techniques to processing signals in a RADAR or LIDAR ranging system.

1. Computing CT Convolutions

First, we will develop techniques to approximate CT convolution in MATLAB. While it is impossible to simulate general CT operations perfectly using DT computations on a computer, it is possible to emulate many CT operations, including CT convolution, to high accuracy.

To illustrate the CT convolution, we will start with an input signal $x(t)$ and a LTI system with impulse response $h(t)$. In MATLAB, we must first *discretize* CT signals in order to approximate them as DT signals. Throughout EE 102A, for clarity, we will use *sampling* to refer to the conversion of a CT signal to a DT signal, and *discretizing* to refer to the approximation of a CT signal by a DT signal.

When discretizing a signal for computer simulation, we choose some discretization interval Δt that is sufficiently small to produce an accurate result. We discretize time in increments of Δt. Let $t = n\Delta t$ and $t' = k\Delta t$. This allows us to represent CT signals by DT signals as follows:

$$x(t - t') \quad \text{by} \quad x[n - k] = x(n\Delta t - k\Delta t),$$
$$h(t') \quad \text{by} \quad h[k] = h(k\Delta t)$$
$$y(t) \quad \text{by} \quad y[n] = y(n\Delta t).$$

We write the CT convolution as

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(t - t')h(t')dt'$$

From our study of calculus, we know that a sufficiently well-behaved integral can be considered as the limiting case of a Riemann sum. Thus, we approximate the CT convolution integral by the Riemann sum

$$y(t) = \int_{-\infty}^{\infty} x(t - t')h(t')dt' = \lim_{\Delta t \to 0} \sum_{k=-\infty}^{\infty} x(n\Delta t - k\Delta t)h(k\Delta t)\Delta t \approx (x[n] * h[n])\Delta t$$

This tells us that if we want to compute the integral approximately, we can simply compute a sum with Δt sufficiently small, since we know that we achieve $y(t)$ exactly as $\Delta t \to 0$. In MATLAB, we can write this as

```
y = conv(x,h)*deltat;
```

and this yields a reasonable approximation of $y(t)$.

a. Rectangular Pulse

Now let us implement this in practice. Write and turn in a function that returns a unit rectangular pulse, as defined in lecture:

$$\Pi(x) = \begin{cases} 1, & |x| \leq \frac{1}{2} \\ 0, & \text{else} \end{cases}$$

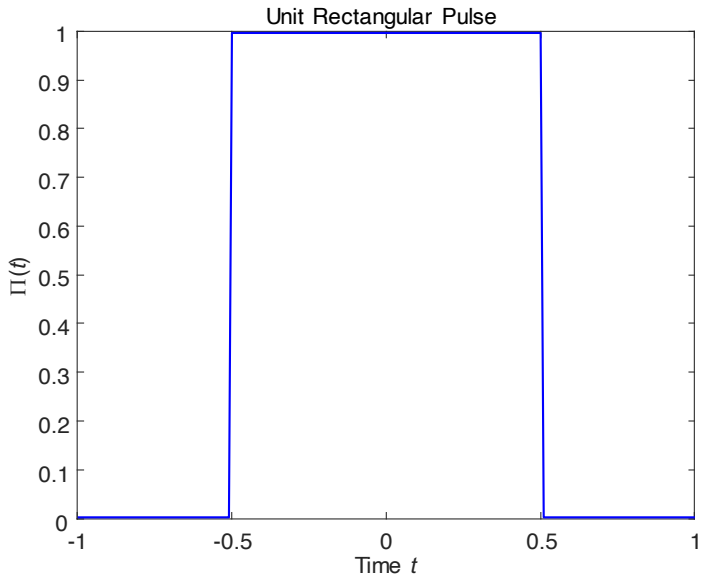
Recall from HW1, **`double(f(x))`** for some vector **`x`** in MATLAB evaluates to 1 when **`f(x)`** is true and 0 otherwise. For example, the unit step function $u(x-a)$ in MATLAB is given by **`double(x>=a)`**. Given this information, implement a rectangular pulse function described by the header below:

```
function y = Pi(x)
% Unit rectangular pulse
```

Make sure your function yields correct values at the endpoints, Π(–1/2) = Π(1/2) = 1. You can test your function with the following code in the Command Window

```
>> t = -1:.01:1;
>> figure; plot(t,Pi(t), 'LineWidth', 1.5);
>> set(gca,'FontName','arial','FontSize',14);
>> xlabel('Time \itt\rm (s)'); ylabel('\it\Pi(t)');
>> title('Unit Rectangular Pulse');
```

and you should see the following output:



b. First Order Lowpass Filter

Now, we use this signal as an input to a first order lowpass filter. It is described by a differential equation

$$\tau \frac{dy}{dt} + y(t) = x(t),$$

and has impulse response

$$h(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} u(t).$$

Assume a time constant $\tau = 0.2$. Assume an input signal $x(t) = \Pi\left(\frac{t-T/2}{T}\right)$, with pulse width $T = 1$. Define the time vector for $x(t)$ and $h(t)$ as follows

```
deltat = 0.01;                          % time increment
tau = 0.2;                              % FOLPF time constant
T = 1;                                  % rectangular pulse width
t1 = -0.5; t2 = 10*tau;
t = t1:deltat:t2;
```

We choose the time **t2 = 10*tau**, such that the impulse response **h** will decay by a factor $e^{-10} \approx 5 \times 10^{-5}$ before being truncated at **t2**. We choose the time vector for the output $y(t)$ to account for convolution:

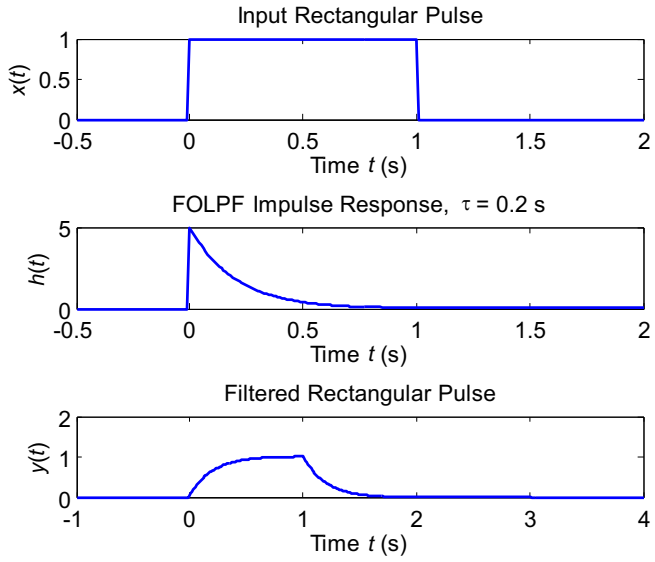
```
t1y = t1+t1; t2y = t2+t2;
ty = t1y:deltat:t2y;
```

We compute discretized versions of $x(t)$ and $h(t)$ using

```
x = Pi((t-T/2)/T);                      % x(t)
h = 1/tau * exp(-t/tau) .* double(t>=0); % h(t)
```

Write a MATLAB script to calculate the discretized version of $y(t)$. Report the number of samples in **x**, **h**, and **y**.

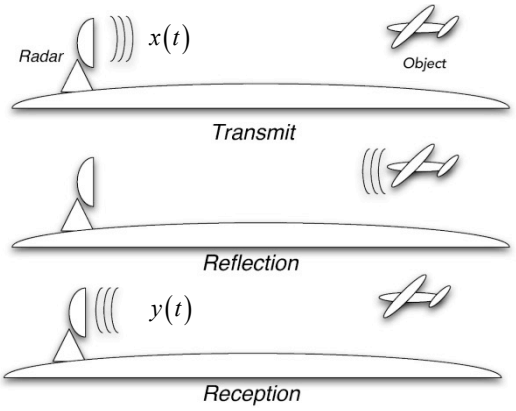
Finally, plot all three signals. Your final plot should look something like the one below. Note the different time scale of the third plot.



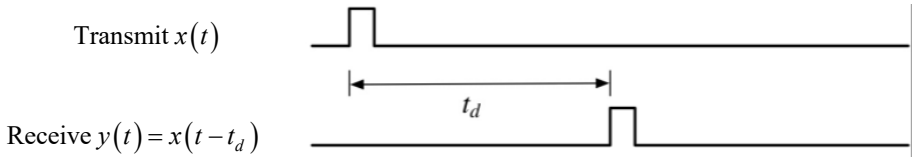
Briefly explain how we know what are the starting time, ending time, and number of samples in the time vector **ty**.

2. Ranging Systems, e.g., RADAR (RAdio Detection And Ranging)

The figures below illustrates the basic operating principle of a simple RADAR system. Similar principles are used in LIDAR (LIght Detection And Ranging) and SONAR (SOund Navigation And Ranging).



The antenna emits a pulse $x(t)$, which is reflected from an object and returns to the antenna after a round-trip propagation delay t_d, yielding a delayed signal $y(t) = x(t - t_d)$.

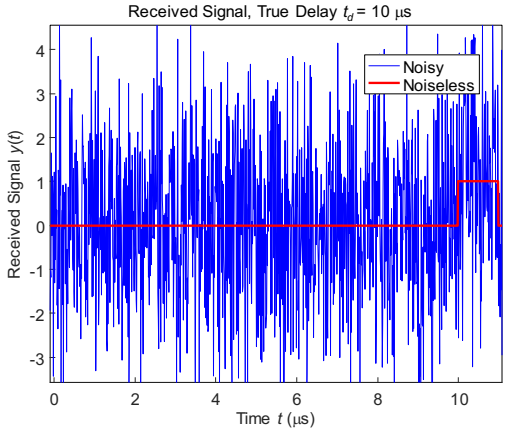


Since radio waves propagate at a known speed ($c = 2.998 \times 10^8$ m/s), we can calculate the distance d to the object of interest:

$$d = ct_d/2. \tag{1}$$

We divide by 2 because t_d corresponds to the round-trip time, and we are interested in the one-way distance d. In practice, there are complications (such as a Doppler shift for moving objects), which we ignore here.

This seems simple in theory; however, in practice we do not receive a clean signal, but rather a highly attenuated signal corrupted by substantial noise. This noise arises from various sources outside the receiver (interfering radio transmissions and thermal radiation in the sky) and inside the receiver (thermal noise in resistors and transistors). A noisy received signal is shown in blue. The reflected radar signal, shown in red, is completely obscured by noise.

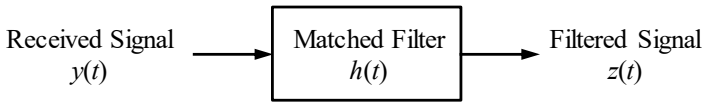


Fortunately, we can use techniques from EE 102A to extract the radar signal from the noise. In a ranging system, we know $x(t)$, the signal we transmitted, and can exploit this knowledge in designing the receiver.

For simplicity, we start by ignoring the noise, so the received signal is

$$y(t) = x(t - t_d). \tag{2}$$

To simplify the mathematics, we ignore attenuation of the signal. When we add noise later, we will scale up the noise accordingly. In our receiver, we filter $y(t)$ using a filter $h(t)$, obtaining an output $z(t)$.

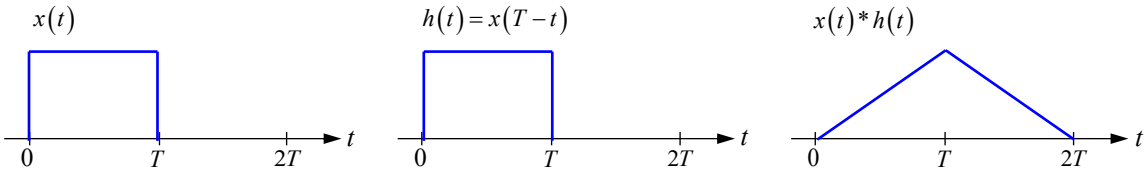


Since we know the signal we want to receive is a delayed version of $x(t)$, we will design the filter impulse response to match it, by choosing

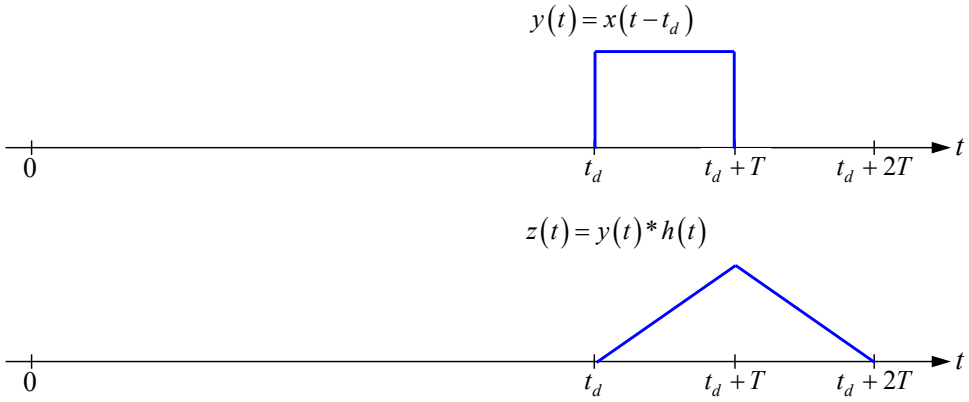
$$h(t) = x(T - t\,). \tag{3}$$

The filter impulse response is a time-reversed version of $x(t)$, delayed by T to make it a causal system. A filter $h(t)$ so designed for a signal $x(t)$ is called a *matched filter* for the signal $x(t)$. (In the more general case that $x(t)$ is complex-valued, (3) would become $h(t) = x^*(T - t\,)$.)

To keep things simple, we will choose $x(t)$ to be a real rectangular pulse, so the matched filter $h(t)$ is identical to $x(t)$, as shown here. If we convolve $x(t)$ with $h(t)$, as shown, we obtain a triangular pulse that reaches a peak at $t = T$. Even if the pulse shape $x(t)$ is not symmetric, the convolution of $x(t)$ with $h(t) = x(T - t\,)$ will peak at $t = T$. You can convince yourself of this by convolving any asymmetric pulse shape of duration T with its delayed time reversal.



Now suppose we receive $y(t) = x(t - t_d)$, given by (2). When we convolve $y(t)$ with the matched filter $h(t)$, the output $z(t) = y(t) * h(t)$ will peak at $t = t_d + T$, as shown below.



Thus, if we look for the time at which the matched filter output $z(t)$ reaches a peak, and subtract T from it, we will obtain the time delay t_d.

a. Noiseless Ranging System

Now that we have discussed the principle of matched filtering, we will simulate a noiseless system, assuming the received signal is $y(t) = x(t - t_d)$. First, define the parameters:

```
%% Task 2a (noiseless)

% All times are in microseconds
deltat = 0.01;                % time increment
T = 1;                        % rectangular pulse width
td = 10;                      % round-trip time delay
c = 2.9979e2;                 % speed of light (m/microsecond)
```

Now define the time vector for the transmitted signal $x(t)$:

```
% Transmitted signal x(t)
tx1 = 0; tx2 = T;
tx = tx1:deltat:tx2;          % time for x
```

Next, use your function **Pi.m** to generate a rectangular pulse **x** with unit amplitude over the interval $0 \le t \le T$:

$$x(t) = \Pi\left(\frac{t - T/2}{T}\right)$$

```
x = Pi((tx-T/2)/T);           % x(t)
```

Now define the time vector for $y(t)$. Use the same step **deltat**, but a start time of **tx1** and an end time of **tx2 + td**. Call this signal **ty**. Using your function **Pi.m**, write the MATLAB code to define the received signal $(t) = x(t - t_d)$:

$$y(t) = x(t - t_d) = \Pi\left(\frac{t - t_d - T/2}{T}\right).$$

Define your filter $h(t) = x(T - t)$ and its corresponding time vector, as given below.

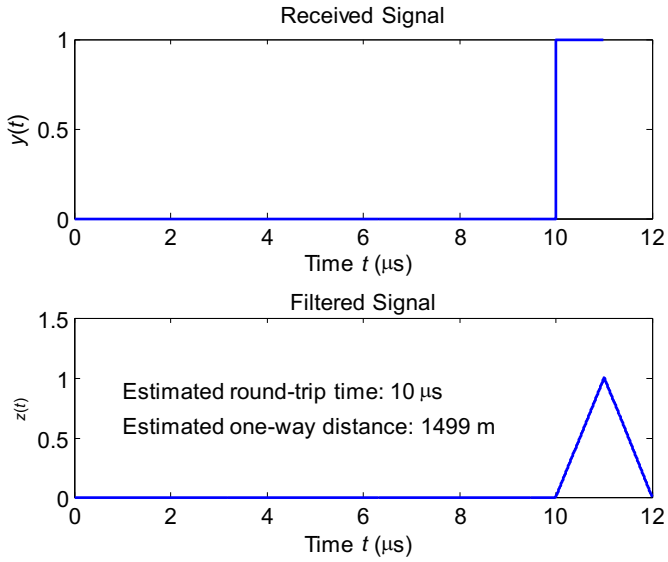
```
% Causal matched filter h(t)= x(T-t)
th1 = tx1; th2 = tx2;
th = th1:deltat:th2;          % time for h
h = fliplr(x);                % h(t)
```

Write the MATLAB code to compute the filtered signal $z(t)$ by approximating the CT convolution output $z(t) = y(t) * h(t)$ using the MATLAB **conv** command, as you did in Task 1. You will need to define the appropriate time vector **tz** (think about what **tz1** and **tz2** should be).

Now, we find the index of the maximum value of $z(t)$, which occurs at $t = T + t_d$. Then, we use this index to get the corresponding time, subtract T, and then compute the one-way propagation distance.

```
[zmax,index] = max(z);        % finding the peak in z(t)
td_est = tz(index) - T;       % estimated round-trip delay time
d_est = c*td_est/2;           % estimated one-way propagation distance
```

Finally, plot $y(t)$ and $z(t)$ using subplots. Your final plot should resemble the one below.



Report the estimated round-trip delay **`td_est`** and one-way distance **`d_est`** on the figure itself. You can put text on your figures with the following commands:

```
text(1,0.9*max(z),['Estimated round-trip time: ' num2str(td_est,4) '
\mus'], 'FontName','arial','FontSize',14);
text(1,0.6*max(z),['Estimated one-way distance: ' num2str(d_est,4) '
m'], 'FontName','arial','FontSize',14);
```

b. Noisy Ranging System

Now we include noise, so the received signal becomes

$$y(t) = x(t - t_d) + n(t), \tag{5}$$

where $n(t)$ represents the noise. Noise signals are *random processes*. They lie beyond the scope of EE 102A, and are discussed in courses such as EE 178. For now, we can just think of $n(t)$ as a signal that fluctuates randomly, and is not precisely predictable at any instant in time. If we input the noisy received signal (5) into the matched filter $h(t)$, the output is

$$z(t) = y(t) * h(t) = \int x(t - t_d - t')h(t')dt' + \int n(t - t')h(t')dt'. \tag{6}$$

On the right side of (6), the first term is the desired one, whose peak yields our estimate of the time delay t_d, while the second term represents noise that has been filtered by the matched filter $h(t)$. The noise observed in a ranging system or a communication system is often well-modeled as *white noise*, meaning it has equal power at all frequencies of interest. If white noise is filtered by a lowpass filter, such as the matched filter $h(t)$, the fluctuations are "smoothed out" and their impact is reduced. As you will learn in advanced courses, given a known signal $x(t)$ to be detected in the presence of white noise, a matched filter $h(t) = x(T - t)$ is optimal in that it maximizes the signal-to-noise ratio of the filtered signal.

Here we will give you a recipe for modeling CT white noise in a DT simulation. Do not worry about any of the technicalities, which are beyond the scope of EE 102A.

We first define the power spectral density of the noise (the power in a unit bandwidth at any frequency):

```
%% Task 2b (noise added)
Sn = 0.03;                    % power spectral density of noise
```

Then we describe the CT white noise by a DT random signal **n**. We use the command **randn**, so each sample of the random signal **n** is drawn independently from a normal (Gaussian) distribution with zero mean. The standard deviation **sigma** is scaled inversely with the square root of the discretization interval **deltat**, since a smaller **deltat** corresponds to a wider frequency bandwidth and thus a larger noise variance (square of the standard deviation).

```
% Noise n(t)
sigma = sqrt(Sn/deltat);      % standard deviation of noise
n = sigma*randn(size(ty));    % n(t)
```

Now, define your $x(t)$, $h(t)$, and $y(t)$ as you did before, with the only difference being the addition of noise in $y(t)$. Compute $z(t)$ and plot both $y(t)$ and $z(t)$ using subplots. Display the estimated round-trip time and estimated one-way distance. After including noise in $y(t)$, you should be able to use all the rest of the code you wrote for Task 2(a) above.