EE102A Winter 2012-13 Pauly
Signal Processing and Linear Systems I

Problem Set #3 Solutions

Due: Wednesday, Jan 30, 2012 at 5 PM.

1. It is often useful to represent operations on signals as convolutions. For each of the following, find a function $h(t)$ such that $y(t) = (x * h)(t)$. These will turn up often later in the course.

   (a) $y(t) = \int_{-\infty}^{t} x(\tau)\, d\tau$

   Solution:

   There are two ways to do these problems. What we are looking for is the inpulse response of these systems, so you can simply replace $x(t)$ with $\delta(t)$, and see what you get. Many of these are pretty simple, then. The other way to solve these is to think about what the system does, and try to find a function that does this by convolution. That is what we'll do here.

   We want to find an $h(t)$ so that

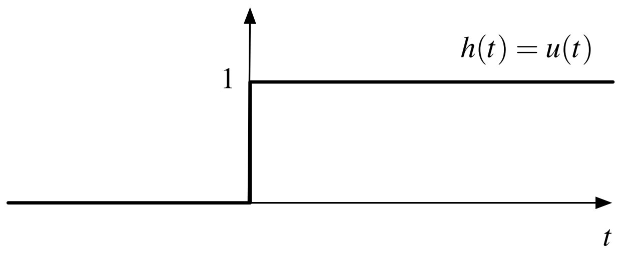
$$\begin{aligned} y(t) &= \int_{-\infty}^{t} x(\tau)\, d\tau \\ &= \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \end{aligned}$$

   From the lecture notes, if $h(t)$ was causal, the upper limit of the convolution would be t. In addition, we want $h(t)$ to be of unit amplitude, so that we get $x(\tau)$ when we multiply by $h(t-\tau)$. This means that $h(t) = u(t)$, since $h(t)$ is one for positive time, and zero for negative time. Then

$$\begin{aligned} y(t) &= \int_{-\infty}^{\infty} x(\tau)u(t-\tau)d\tau \\ &= (x * u)(t). \end{aligned}$$

   where $u(t)$ is the unit step. We conclude that

$$y(t) = (x * h)(t) \text{ with } h(t) = u(t).$$



(b) $y(t) = \int_{t-1}^{t} x(\tau)\, d\tau$

Solution:

$$\begin{aligned} y(t) &= \int_{t-1}^{t} x(\tau)\, d\tau \\ &= \int_{-\infty}^{t} x(\tau)\, d\tau - \int_{-\infty}^{t-1} x(\tau)\, d\tau \\ &= (x * h_0)(t) - (x * h_1)(t) \end{aligned}$$

where $h_0(t) = u(t)$ is the unit step, as in (a), and $h_1(t) = u(t-1)$ is the unit step delayed by 1. Then

$$\begin{aligned} y(t) &= (x * (h_0 - h_1))(t) \\ &= (x * h)(t) \end{aligned}$$

where $h(t) = h_0(t) - h_1(t) = u(t) - u(t-1)$. This is a square pulse of of amplitude 1, that goes from time zero to 1.

(c) $y(t) = x(t)$

Solution: Here we want a convolution system that simply returns the input

$$y(t) = \int_{-\infty}^{\infty} x(\tau) h(t-\tau) d\tau$$

If $h(t)$ is an an impulse function,

$$y(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t-\tau) d\tau = x(t)$$

by the sifting property. So

$$h(t) = \delta(t).$$

(d) $y(t) = x(t-1)$

Solution: This will be similar to (c), with an addition that the output should be delayed,

$$y(t) = \int_{-\infty}^{\infty} x(\tau) h(t-\tau) d\tau$$

If we let $h(t)$ be a delayed impulse $\delta(t-1)$, then

$$y(t) = \int_{-\infty}^{\infty} x(\tau)\delta((t-\tau)-1) d\tau = \int_{-\infty}^{\infty} x(\tau)\delta((t-1)-\tau)) d\tau = x(t-1)$$

again by the sifting property. So

$$h(t) = \delta(t-1).$$

(e) $y(t) = x'(t)$

Solution:

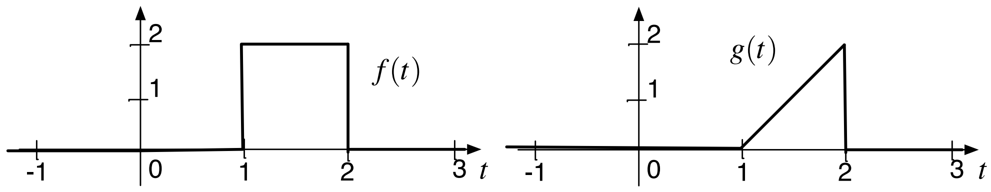
We want a function $h(t)$ such that

$$x'(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

In Lecture 3 we introduced the derivative of an impulse, which looks like a likely candidate. If we try it we get

$$\begin{aligned}\int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau &= \int_{-\infty}^{\infty} x(t-\tau)h(\tau)d\tau \\ &= \int_{-\infty}^{\infty} x(t-\tau)\delta'(\tau)d\tau \\ &= -\frac{d}{d\tau}x(t-\tau)\bigg|_{\tau=0} \\ &= -x'(t)(-1) \\ &= x'(t)\end{aligned}$$

2. Graphically compute the convolution of these two functions:



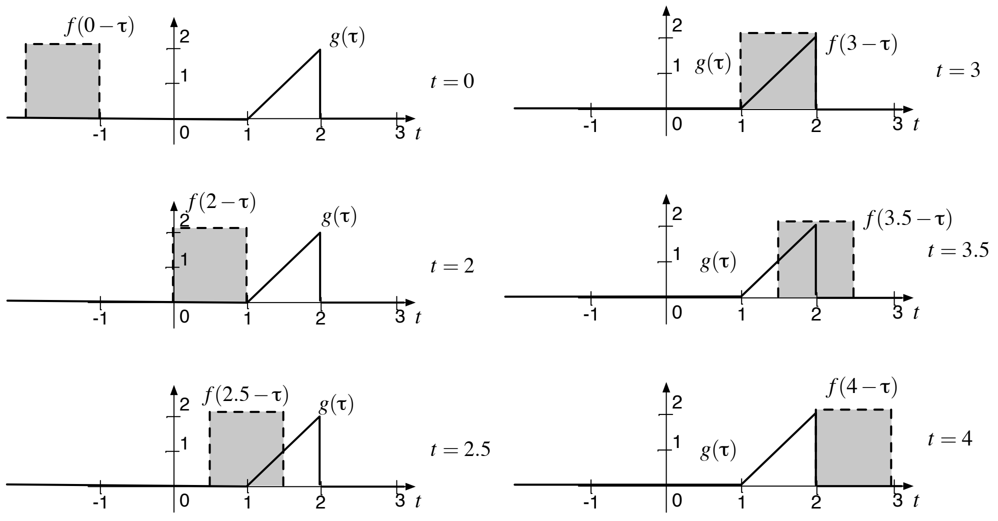
Solution:

The convolution can be graphically performed by flipping f about the origin, shifting it right by t, multiplying point by point with $g(t)$, and then evaluating the area. At time $t = 0$ there is no overlap, and the output is zero. The first overlap begins at $t = 2$, so this is the first output point. The area of the overlap is

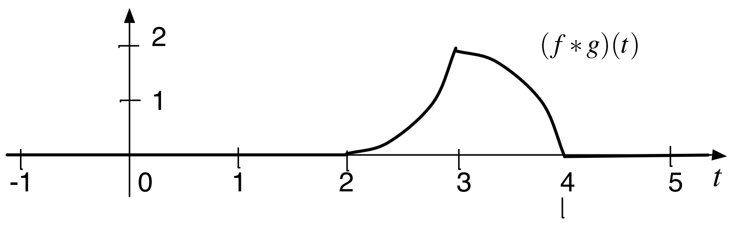
$$\int_{2}^{t} 4(\tau-2)d\tau = (1/2)(t-2)(4(t-2)) = 2(t-2)^2$$

for $2 < t < 3$. This is a smoothly increasing quadratic. over this interval, which reaches a peak amplitude of 2 at $t = 3$. From $t = 3$ to $t = 4$, the area of the overlap starts to decrease slowly, as the initial part of the triangle no longer overlaps the rectangle. The rate at which we loose area is exactly the same as the rate we gained area from $t = 2$ to $t = 3$, so the result is the same shape, but inverted. We get a smooth, quadratic signal, concave down, that starts at an amplitude of 2 at $t = 3$, and goes to 0 at $t = 4$, when the triangle and the rectangle no longer overlap.

These breakpoints are illustrated below:



The result is



3. Show the somewhat surprising result that the convolution of two impulse functions,

$$y(t) = \int_{-\infty}^{\infty} \delta(\tau)\delta(t-\tau)\ d\tau$$

is itself an impulse function.

Hint: To make sense of the integral, first replace one of the impulses with $\lim_{\epsilon\longrightarrow 0} g_\epsilon(t)$, where $g_\epsilon(t)$ is

$$g_\epsilon(t) = \begin{cases} 1/\epsilon & |t| < \epsilon/2 \\ 0 & \text{otherwise} \end{cases}$$

This is a model for an impulse function, as we discussed in class. After the convolution, show you get the same model function back.

Solution:
To figure out what

$$y(t) = \int_{-\infty}^{\infty} \delta(\tau)\delta(t-\tau)\ d\tau$$

means, we have to deal with the impulse functions carefully. Remember from class that one of the expressions that makes no sense with impulses is $\delta^2(t)$, and this is what we have here for $t=0$! To understand what this integral means we replace $\delta(\tau)$ with $\lim_{\epsilon\longrightarrow 0} g_\epsilon(t)$, and then evaluate the integral

$$\begin{aligned} y(t) &= \int_{-\infty}^{\infty} \delta(\tau)\delta(t-\tau)\, d\tau \\ &= \int_{-\infty}^{\infty} \left(\lim_{\epsilon\to 0} g_\epsilon(\tau)\right) \delta(t-\tau) d\tau \\ &= \lim_{\epsilon\to 0} \int_{-\infty}^{\infty} g_\epsilon(\tau)\delta(t-\tau)d\tau \\ &= \lim_{\epsilon\to 0} g_\epsilon(t) \\ &= \delta(t). \end{aligned}$$

where we can evaluate the last integral using the sifting property of impulse functions, since $g_\epsilon(\tau)$ is a conventional function for any given ϵ. Hence, we get the surprising result that

$$(\delta * \delta)(t) = \delta(t)$$

4. In problem 1(e) above, you showed that the differentiation operation can be represented as a convolution. Using this result, show that if $f * g = y$, then

$$f'' * g = f' * g' = f * g'' = y''.$$

Solution:
Convolution is commutative and associative, and so we can group the differentiation with either f, g, or $f * g = y$,

$$f' * g' = (f * \delta') * (g * \delta') = f * g * \delta' * \delta' = (f * g)' * \delta' = (f * g)'' = y''$$

5. Cross-correlation and convolution are closely related. In the radar class the cross-correlation operation was given by either of these two expressions,

$$(x \star y)(t) = \int_{-\infty}^{\infty} x(\tau - t)y(\tau)d\tau = \int_{-\infty}^{\infty} x(\tau)y(\tau + t)d\tau.$$

In this question we'll look at some of the properties of cross-correlation.

(a) Show that if $x(t)$ and $y(t)$ are two signals,

$$(x \star y)(t) = z(t)$$

then

$$(y \star x)(t) = z(-t)$$

meaning if we change the order of the signals in the cross correlation, we time reverse the resulting signal.

(b) Show that we can perform the cross correlation $(x \star y)(t)$ by a convolution operation. Express the cross correlation as a convolution involving $x(t)$ and $y_r(t) = y(-t)$.

Solution

As was mentioned in the radar notes, the cross correlation of $y(t)$ with $x(t)$ can be written with a positive lag of the second argument

$$(x \star y)(t) = \int_{-\infty}^{\infty} x(\tau) y(t+\tau) d\tau$$

Then

$$\begin{aligned}(y \star x)(t) &= \int_{-\infty}^{\infty} y(\tau) x(t+\tau) d\tau \\ &= \int_{-\infty}^{\infty} x(t+\tau) y(\tau) d\tau \\ &= \int_{-\infty}^{\infty} x(\tau - (-t)) y(\tau) d\tau \\ &= (x \star y)(-t)\end{aligned}$$

Hence interchanging the order of the correlation operators reverses the resultant correlation.

For (b), we start with the correlation expression, and replace $y(t)$ with $y_r(-t)$,

$$\int_{-\infty}^{\infty} x(\tau - t) y(\tau) d\tau = \int_{-\infty}^{\infty} x(\tau - t) y_r(-\tau) d\tau$$

Letting $\tau' = -\tau$, we get

$$\int_{-\infty}^{\infty} x(-\tau' - t) y_r(\tau') d\tau' = \int_{-\infty}^{\infty} x((-t) - \tau') y_r(\tau') d\tau' = (x * y_r)(-t).$$

so

$$(x \star y)(t) = (x * y_r)(-t)$$

6. Assume

$$f(t) = (g * h)(t).$$

Let the delayed versions of $g(t)$ and $h(t)$ be

$$\begin{aligned} g_1(t) &= g(t-1) \\ h_1(t) &= h(t-1). \end{aligned}$$

Find a simple expression for $(g_1 * h_1)(t)$ in terms of $f(t)$.

Solution

One way to solve this is write the delay as a convolution with a delayed impulse,

$$g_1(t) = g(t-1) = (g * \delta_1)(t)$$

where $\delta_1 = \delta(t-1)$. Then

$$g_1 * h_1 = (g * \delta_1) * (h * \delta_1) = (g * h) * (\delta_1 * \delta_1) = (g * h) * \delta_2 = f * \delta_2 = f(t-2)$$

where $\delta_2 = \delta(t-2)$.

7. If $y(t) = (x * h)(t)$, show that

$$\int_{-\infty}^{\infty} y(\tau)d\tau = \left(\int_{-\infty}^{\infty} x(\tau)d\tau\right)\left(\int_{-\infty}^{\infty} h(\tau)d\tau\right).$$

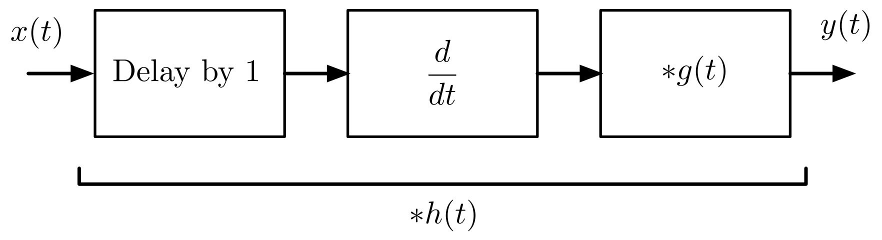
This means that the area of the convolution $y(t)$ is the product of the areas of the signals that are being convolved $x(t)$ and $h(t)$.

Solution

The easiest way to do this at this point in the course is to start with the expression for the convolution, and then directly integrate both sides

$$\begin{aligned}
y(t) &= \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \\
\int_{-\infty}^{\infty} y(t)dt &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau dt \\
&= \int_{-\infty}^{\infty} x(\tau)\left(\int_{-\infty}^{\infty} h(t-\tau)dt\right)d\tau \\
&= \int_{-\infty}^{\infty} x(\tau)\left(\int_{-\infty-\tau}^{\infty-\tau} h(t')dt'\right)d\tau \\
&= \int_{-\infty}^{\infty} x(\tau)\left(\int_{-\infty}^{\infty} h(t')dt'\right)d\tau \\
&= \left(\int_{-\infty}^{\infty} x(\tau)d\tau\right)\left(\int_{-\infty}^{\infty} h(t')dt'\right) \\
&= \left(\int_{-\infty}^{\infty} x(\tau)d\tau\right)\left(\int_{-\infty}^{\infty} h(\tau)d\tau\right)
\end{aligned}$$

8. A linear system has the block diagram



where

$$g(t) = \text{sinc}(t).$$

Since this is a linear time invariant system, we can represent it as a convolution with a single impulse response $h(t)$.

Find the impulse response $h(t)$. You don't need to explicitly differentiate, but do eliminate all convolutions from your answer.

Solution:

Each of these blocks can be written as a convolution. The delay is a convolution with $\delta(t-1)$, differentiation is convolutions with $\delta'(t)$, and the final block, convolution with sinc(t). The result is

$$h(t) = \delta(t-1) * \delta'(t) * \text{sinc}(t)$$

This was an acceptable answer. You can also simplify this to get

$$h(t) = \text{sinc}'(t-1)$$

Laboratory 3

This laboratory will be concerned with numerically evaluating continuous time convolution integrals. Matlab provides a function `conv()` that performs a discrete-time convolution of two discrete-time sequences. We will add a new function to matlab that uses `conv()` to numerically integrate the continuous time convolution. To do this, we'll need to learn about how to define new functions in matlab.

Matlab provides two ways of executing commands that you have programmed in a file. Both are stored in a ".m" file or m-file. The first is a *script*. Invoking the script at the command line causes the file to be treated as terminal input in your current matlab environment. Most of the examples in the book are matlab scripts. Variables created in the script are visible when it completes, and the script has access to all of the variables that have been defined in your work space. Because it can modify your workspace, this is not a good way to implement extentions to matlab.

The second option is to define a function in the m-file. In this case, the function operates in its own local context. It doesn't have access to variables in your workspace, only to the function arguments which are are passed by value. Only the returned values are available to the calling function, and these are also passed by value. Hence, function m-files are a much more robust way to encapsulate repeated operations.

As an example, lets say you want to be able to compute the average value of some vector. A function m-file that does this is

```
function a = avg(v)
%
% Anything you type here is printed when you enter "help avg"
% at the matlab prompt.  Good things to put here are a description
% what the function does,  the calling  arguments  and the
% return values.
%
%   a = avg(v)
%     Inputs:
%       v -- vector
%     outputs:
%       a -- average value of the elements in v
%

% at last, the actual code
% compute the mean
a = sum(v)/length(v);
```

Store this in a file "avg.m" either in your current directory, or somewhere in your matlab path ("help path" and "help addpath" for more information). Any time you enter `avg()` at the command prompt, this function will be invoked. Any other m-file can also call it. From then on, it will appear as though this function was actually a built-in component of matlab. In fact, many of the functions in matlab are function m-files.

We can also return more than one value. Here we return the mean and the median,

```
function [a, m] = avg_med(v)
%
% Anything you type here is printed when you enter "help avg"
% at the matlab prompt.  Good things to put here are a description
% what the function does, and the calling arguments and
% and return arguments.
%
%   a = avg_med(v)
%     Inputs:
%       v -- vector
%     outputs:
%       a -- average value of the elements in v
%       m -- median value of the elements in v
%

% the actual code
% compute the mean
a = sum(v)/length(v);

% sort the data, take the middle value
vs = sort(v);
m = v(floor((length(v)+1)/2));
```

If we invoke `avg_med` with one return argument

```
>> a = avg_med(v)
```

only the average is returned, as before. If we provide two return arguments, we get the average and the median.

```
>> [a, m] = avg_med(v)
```

In the first lab we talked about using vector operations to manipulate data. In general this is preferable, because it executes much more quickly. Occasionally, you will find operations that don’t vectorize easily. If looping is unavoidable matlab provide both a ”for” loop and a ”while” loop. An example of a for loop is

```
for n=1:2:length(v),
  a = a+ sqrt(v(n));
end;
```

This sums the square roots of the odd elements in `v`. One very useful trick to to build up a vector in a "for" loop. For example

```
a = [];
for n=1:length(v),
  a = [a new_vector(v(n))];
end;
```

initially defines `a` to be an empty vector. Each time through the loop, the `new_vector` function appends additional elements to `a`.

The purpose of the laboratory is to gain experience with continuous convolution by implementing it using simple numerical integration in matlab. Matlab provides a function `conv` which implements the convolution sum,

$$y[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m]$$

If we multiply the discrete time convolution sum by the sampling period T we get an approximation to the continuous time convolution

$$\sum_{m=-\infty}^{\infty} x[m]h[n-m]T \simeq \int_{-\infty}^{\infty} x(\tau)h(t-\tau)\, d\tau \bigg|_{t=nT}.$$

In this lab you will write a matlab m-file that takes two waveforms, each defined as vector of time points and a vector of sample values, and return the numerical approximation to the discrete time convolution. There are several complications that you will need to take care of. One is matlab that actually only returns the `L+M-1` nonzero values of the discrete time convolution, where `L` and `M` are the lengths of $x[n]$ and $h[n]$. Another complication is that matlab assumes both sequences start at 1. Your m-file will need to keep track of the times of the input and output samples. Finally, your m-file will have to include the multiplication by T, in order to get the right output amplitude.

The comment at the top of the file should read something like

```
function [y, ty] = nconv(x,tx,h,th)
%
%  nconv performs a numerical approximation to the
%  continuous time convolution using matlab's conv()
%  function
%
%   [y, ty] = nconv(x,tx,h,th)
%
%     Inputs:
```

```
%        x -- input signal vector
%        tx -- times of the samples in x
%        h -- impulse response vector
%        th -- times of the samples in
%
%   outputs:
%        y -- output signal vector,
%                 length(y)= length(x)+length(h)-1
%        ty -- times of the samples in y
%
%   The command plot(tx,h,th,h,ty,y) should properly
%   display your functions.
```

Task 1 Write a matlab m-file for `nconv`. Include a listing of the m-file in your diary file that you hand in (`type nconv` at the matlab prompt). Make sure that the time vector `ty` is correct. Check that the first and last samples of the convolution are where you would expect them to be, given the times of the first and last samples of the input vector and the impulse response.

Use your m-file to check your results for problem 2. Plot the input, impulse response, and output. Properly label the axes of the plots. Submit the plot with your report.

Solution:

To figure out the output time axis, you only need to realize that the time of the first output put is the sums of the times of the first points of the signals you are convolving. The sample spacing is the same, so we can the immediately compute the new time axis. The resutling m-file is then

```
function [y, ty] = nconv(x,tx,h,th)
%
%  nconv performs a numerical approximation to the
%  continuous time convolution using matlab's conv()
%  function
%
%   [y, ty] = nconv(x,tx,h,th)
%
%     Inputs:
%        x -- input signal vector
%        tx -- times of the samples in x
%        h -- impulse response vector
%        th -- times of the samples in
%
%   outputs:
%        y -- output signal vector,
%                 length(y)= length(x)+length(h)-1
%        ty -- times of the samples in y
%
%   The command plot(tx,h,th,h,ty,y) should properly
%   display your functions.

% first, calculate the time step
dt = tx(2)-tx(1);

% compute the convolution,
% multiplied by dt to approximate the continuous integral
y = conv(x,h)*dt;

% determine the time of the output samples
ty = (tx(1)+th(1)) + [0:(length(y)-1)]*dt;
```

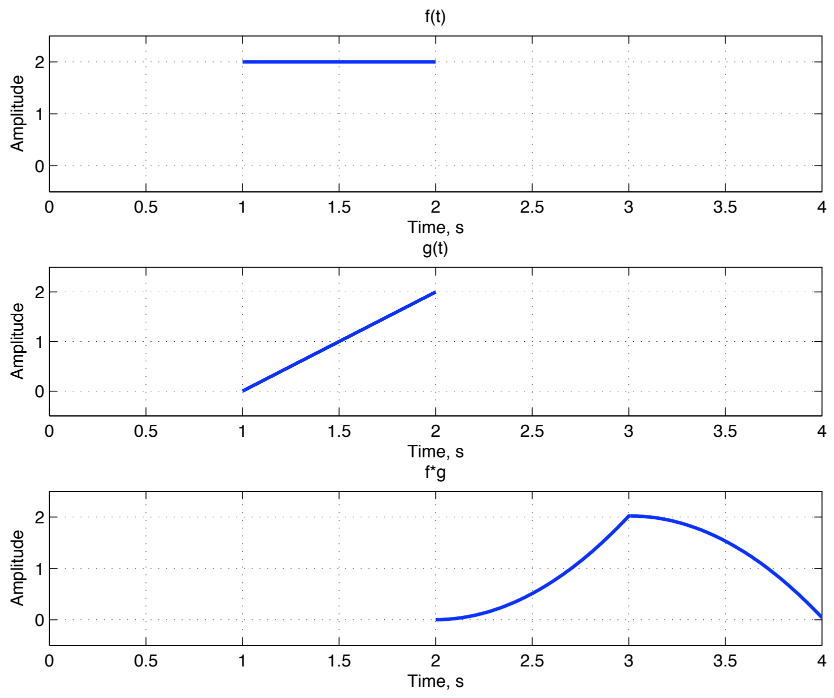
For the case of problem 2 we can define the input time and signal vectors as

```
>> tf = [1:0.01:1.99];
>> f = ones(1,100);
>> tg = tf;
>> g = [1:100]/50;
```

where we have only defined the signals where they are non-zero, since this is the only part that will contribute to the convolution. Then

```
>> [y ty] = nconv(f,tf,g,tg);
>> plot(ty,y);
```

The plots of the input signals, and the convolution are:



Task 2 Now that you have the function `nconv()`, let's use it to examine the properties of convolution a little further.

A) In class we saw that rect $*$ rect $(t) = \Delta(t)$. Perform this calculation explicitly in Matlab using the `nconv()` function that you just wrote. Define $x(t) = rect(t)$ in Matlab by the following

```
>> dt = 0.01;
>> t = -0.5:dt:0.5;
>> x = [ones(1,101)];
```

Now, let's also set the integral of $x(t) = 1$, so normalize `x` with the following

```
>> x = x/(sum(x)*dt); %Make the integral = 1
```

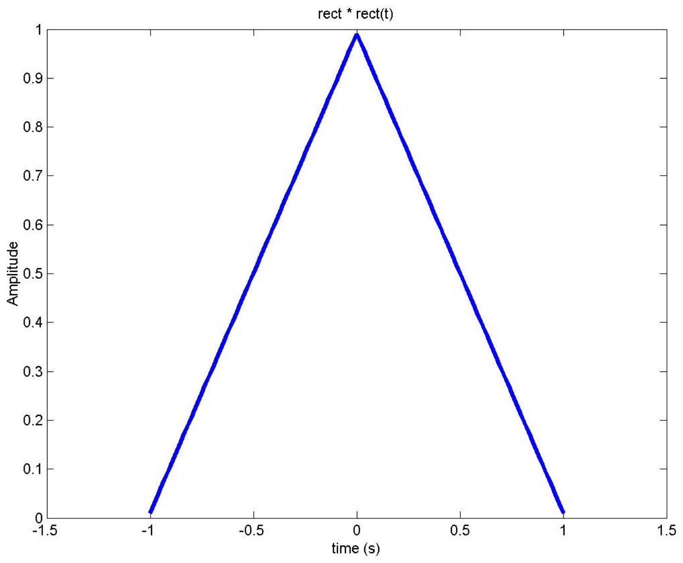
Calculate $y(t)$

```
>> [y, ty] = nconv(x,t,x,t);
```

Plot and label the results.
Solution:

```
x = [ones(1,101)];
x = dt*x/sum(x);
t = -.5:dt:.5;
[y,ty] = nconv(x,t,x,t);
```

```
figure;plot(ty,y);
axis([-1.5 1.5 0 1]);
xlabel('time (s)');
ylabel('Amplitude')
title('rect * rect(t)');
```

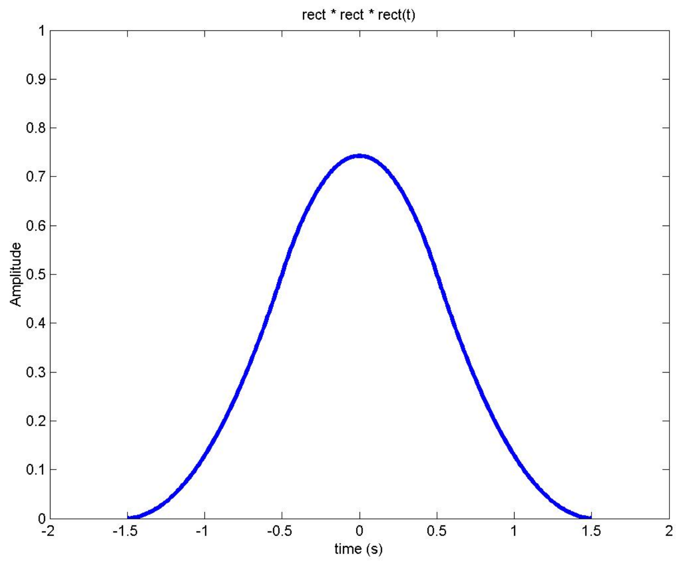


B) Using your `nconv()` function and the result of part A), calculate $y(t) = rect * rect * rect(t) = \Delta * rect(t)$.

```
>> [y, ty] = nconv(x,t,x,t);
>> [y, ty] = nconv(y,ty,x,t);
```

Plot and label the results.
Solution:

```
[y,ty] = nconv(x,t,x,t);
[y,ty] = nconv(y,ty,x,t);
figure;plot(ty,y);
axis([-2 2 0 1]);
xlabel('time (s)');
ylabel('Amplitude')
title('rect * rect * rect(t)');
```



C) Now, what happens if we consider $rect * rect * \cdots * rect(t) = rect^{(N)}(t)$? Using the matlab `for` loop, calculate the result of convolving N $rect(t)$ functions together.

```
>> y = x;
>> ty = t;
>> for n = 1:N
     [y,ty] = nconv(y,ty,x,t);
   end
```

Instead of plotting `y` and `ty`, scale and stretch them first (you'll see why soon). Specifically, calculate $y_2(t) = \sqrt{N}y(\sqrt{N}t)$ with the following.

```
>> y2 = sqrt(N)*y; %This scales the amplitude
>> ty = ty/sqrt(N); %This scales the axis
```

Now, on the same axes, plot `y2` for N = 3, 10, 20, and 40. As N increases, does `y2` seem to be converging to a specific function? On the same axes, plot the function

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

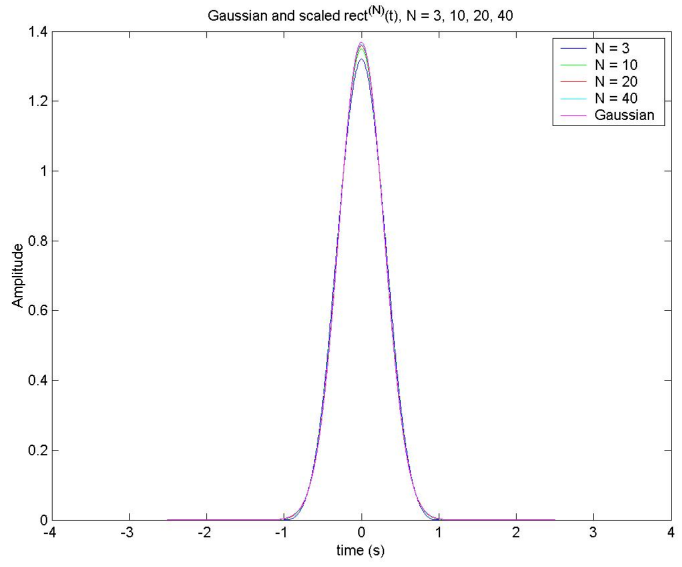
with σ = 0.2915. This is the Gaussian function, also known as the 'bell-curve', with $\mu = 0$.

```
>> sigma = 0.2915;
>> g = 1/(sqrt(2*pi)*sigma)*exp(-(ty.^2)/(2*sigma^2));
```

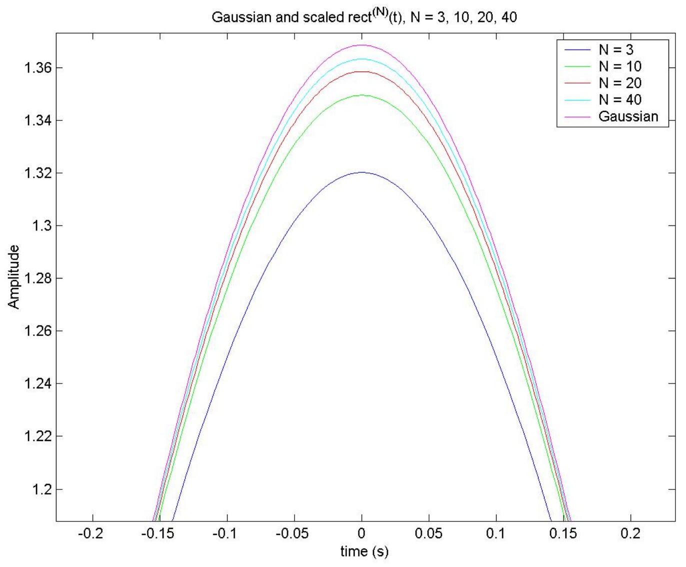
Solution:
Yes, `y2` seems to be converging to the Gaussian.

```
figure;

y = x;
ty = t;
for n = 1:3
    [y,ty] = nconv(x,t,y,ty);
end
y2 = sqrt(n+1)*y;
ty = ty/sqrt(n+1);
plot(ty,y2,'b');hold on;

y = x;
ty = t;
for n = 1:10
    [y,ty] = nconv(x,t,y,ty);
end
y2 = sqrt(n+1)*y;
ty = ty/sqrt(n+1);
plot(ty,y2,'g');hold on;

y = x;
ty = t;
for n = 1:20
    [y,ty] = nconv(x,t,y,ty);
end
y2 = sqrt(n+1)*y;
ty = ty/sqrt(n+1);
plot(ty,y2,'r');hold on;

y = x;
ty = t;
for n = 1:40
    [y,ty] = nconv(x,t,y,ty);
end
y2 = sqrt(n+1)*y;
ty = ty/sqrt(n+1);
plot(ty,y2,'c');hold on;

sigma = 0.2915;
g = 1/(sqrt(2*pi)*sigma)*exp(-(ty.^2)/(2*sigma^2));
```

```
plot(ty,g,'m');
title('Gaussian and scaled rect^(^N^)(t), N = 3, 10, 20, 40');
xlabel('time (s)');
ylabel('Amplitude');
legend('N = 3','N = 10', 'N = 20', 'N = 40','Gaussian');
```



And a zoomed in version



D) Let's repeat part C), except let's try using a different function for `x`. Use the following function instead

```
>> dt = 0.01;
>> t = -0.5:dt:0.5;
>> x = [ones(1,20) zeros(1,61) ones(1,20)];
>> x = x/(sum(x)*dt); %Make the integral = 1
```

Plot and label the function. Calculate `y` and `y2` the same way.

Does `y2` also seem to be converging? On the same set of axes plot

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

with $\sigma = 0.4091$

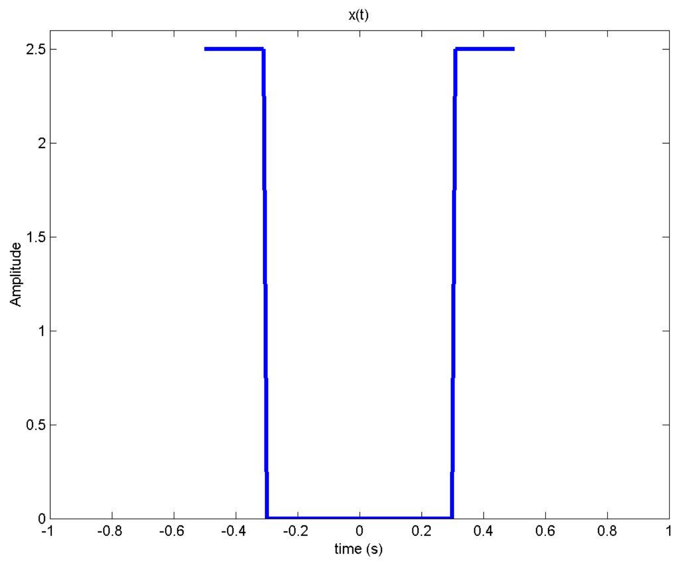
```
>> sigma = 0.4091;
>> g = 1/(sqrt(2*pi)*sigma)*exp(-(ty.^2)/(2*sigma^2));
```

`y2` should be converging to the function `g`. Calculate the minimum value of `N` so that the function differs by at most 0.01. In other words, make sure max(`y2` - `g`) < 0.01

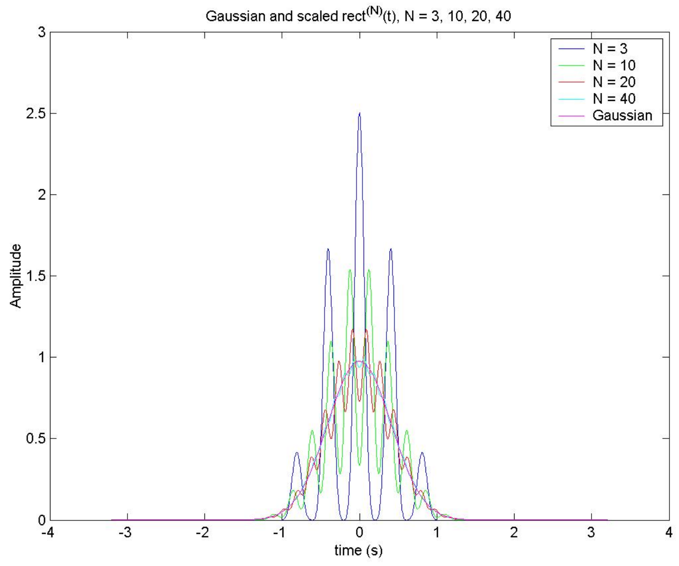
Solution:

The new x function is shown below

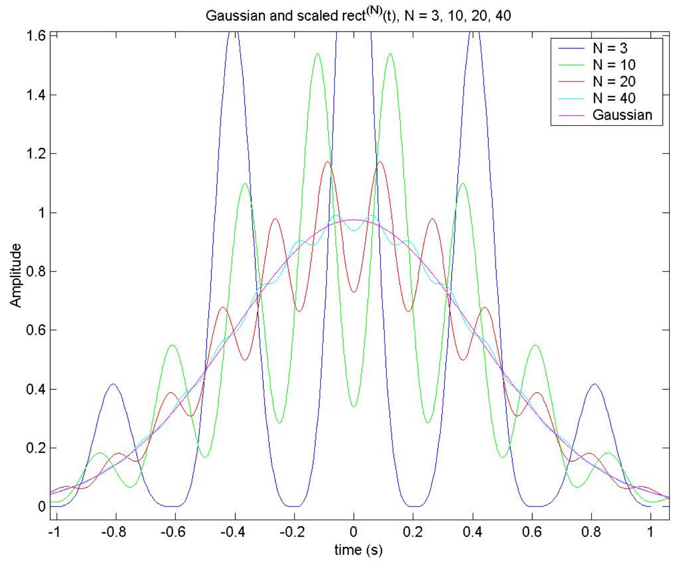
```
x = [ones(1,20) zeros(1,61) ones(1,20)];
x = dt*x/sum(x);
t = -.5:dt:.5;
figure;plot(t,x);
axis([-1 1 0 1.1]);
xlabel('time (s)');
ylabel('Amplitude')
title('x(t)');
```



Yes, `y2` is converging to the Gaussian. The matlab code is identical to part c), except with $\sigma = 0.4091$.



There are many more oscillations, but zooming in we see the plots still converge.



The error max(`y2` - `g`) $<$ 0.01 occurs for `N` = 50.

```
y = x;
ty = t;
for n = 1:100
    [y,ty] = nconv(x,t,y,ty);
    y2 = sqrt(n+1)*y;
    ty2 = ty/sqrt(n+1);
    g = 1/(sqrt(2*pi)*sigma)*exp(-(ty2.^2)/(2*sigma^2));
    if(max(y2 - g) < 0.01)
        break;
    end
end
n
```

In both parts C) and D), after many convolutions of $x(t)$, the results converge to Gaussian functions with different values for σ. This is true in general for any function $x(t)$. After enough convolutions with themselves, all signals (after scaling and stretching) approach the shape of a Gaussian! (This is easier to show with the cases we looked at when $\int x(t)dt = 1$, and $\mu = \int tx(t)dt = 0$.) Finally, this is related to an important theorem in Statistics and Statistical Signal Processing called 'The Central Limit Theorem'.