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

Problem Set #3

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

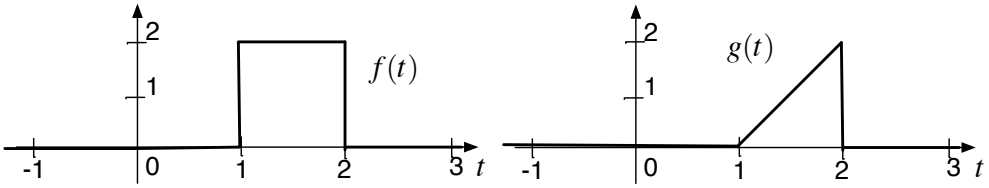
   (b) $y(t) = \int_{t-1}^{t} x(\tau)\, d\tau$

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

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

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

2. Graphically compute the convolution of these two functions:



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 $\lim_{\epsilon \to 0} g_\epsilon(t)$, 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.

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

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

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 $y_r(t) = y(-t)$.

6. Assume

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

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

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

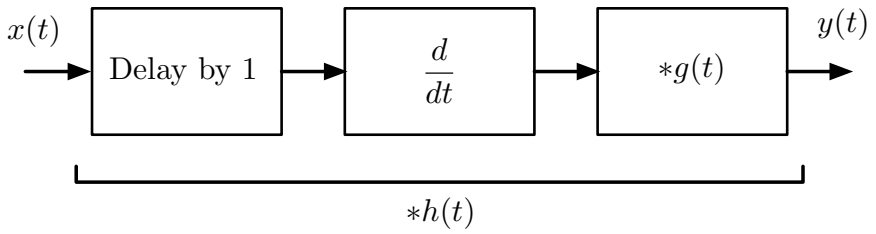
Find a simple expression for $(g_1 * h_1)(t)$ in terms of $f(t)$.

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 $h(t)$.

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.

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 com-

mand 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

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

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.

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.

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.

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));
```

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 σ = 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