6.003 Homework 5

Due at the beginning of recitation on ***Wednesday, March 10, 2010****.*

Problems

1. DT convolution

Let y represent the DT signal that results when f is convolved with g, i.e.,

$$y[n] = (f * g)[n]$$

which is sometimes written as $y[n] = f[n] * g[n]$.
Determine closed-form expressions for each of the following:

$y[n]$	$f[n]$	$g[n]$
$y_a[n]$	$u[n]$	$u[n]$
$y_b[n]$	$u[n]$	$\left(\frac{1}{2}\right)^n u[n]$
$y_c[n]$	$\left(\frac{1}{2}\right)^n u[n]$	$\left(\frac{1}{3}\right)^n u[n]$
$y_d[n]$	$\left(\frac{1}{2}\right)^n u[n]$	$\left(\frac{1}{2}\right)^n u[n]$

2. CT convolution

Let y represent the CT signal that results when f is convolved with g, i.e.,

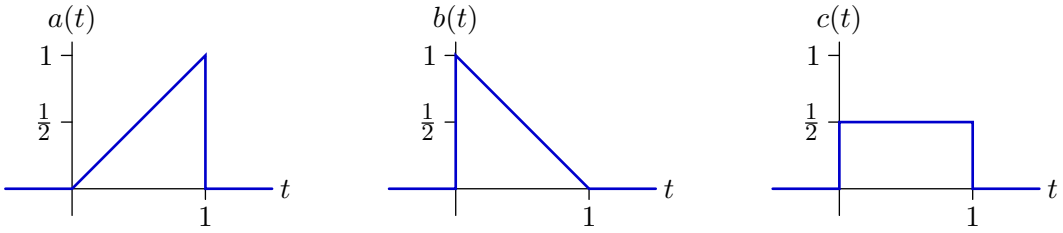
$$y(t) = (f * g)(t)$$

which is sometimes written as $y(t) = f(t) * g(t)$.
Determine closed-form expressions for each of the following:

$y(t)$	$f(t)$	$g(t)$
$y_a(t)$	$u(t)$	$u(t)$
$y_b(t)$	$u(t)$	$e^{-t}u(t)$
$y_c(t)$	$e^{-t}u(t)$	$e^{-2t}u(t)$
$y_d(t)$	$e^{-t}u(t)$	$e^{-t}u(t)$

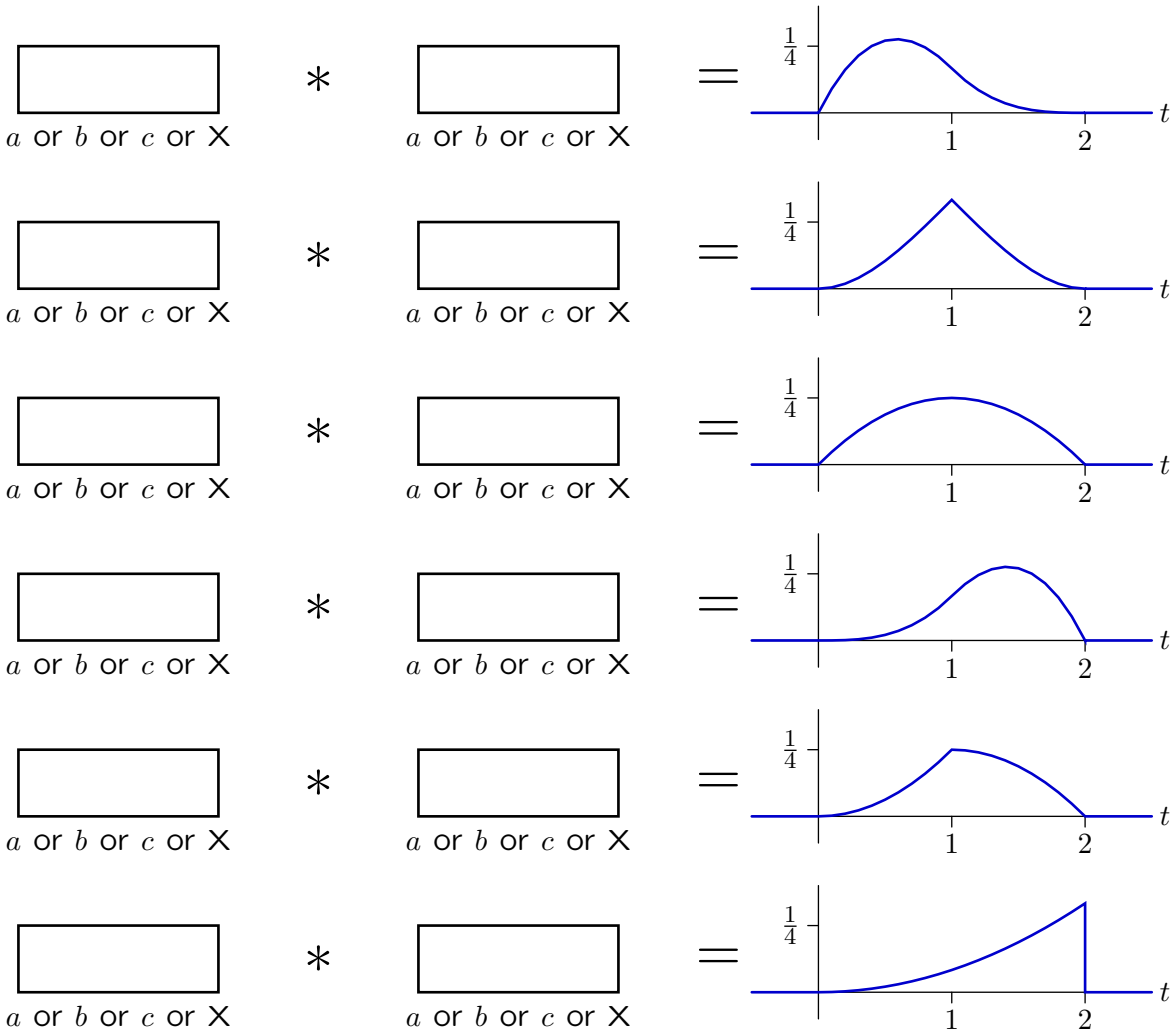
3. Convolutions

Consider the convolution of two of the following signals.



Determine if each of the following signals can be constructed by convolving (a or b or c) with (a or b or c). If it can, indicate which signals should be convolved. If it cannot, put an X in both boxes.

Notice that there are ten possible answers: $(a * a)$, $(a * b)$, $(a * c)$, $(b * a)$, $(b * b)$, $(b * c)$, $(c * a)$, $(c * b)$, $(c * c)$, or (X,X). Notice also that the answer may not be unique.



4. The $\mathcal{L}$ operator

The $\mathcal{R}$ (right-shift) operator can be used to express functional forms for any block diagram that is composed of adders, gains, and delays. Since the delay operation is causal (i.e., its output does not depend on future values of its input) functionals that build on $\mathcal{R}$ can only represent causal systems.

Define the $\mathcal{L}$ (left-shift) operator to represent "anticipators," which are the anti-causal counterparts to delay elements. If X is an input sequence, then $Y = \mathcal{L}X$ means that $y[n] = x[n+1]$ for all n.

Let H_1 represent the system described by the following functional:

$$H_1 = \frac{Y}{X} = \frac{1}{1-3\mathcal{L}}\,.$$

a. Determine a difference equation for this system.

b. Find the unit-sample response of this system.

c. Determine the pole(s) of this system.

d. If a system is represented as the ratio of polynomials in $\mathcal{R}$, then the poles of the system are the roots of the denominator polynomial after $\mathcal{R}$ is replaced by $\frac{1}{z}$. Determine an analogous rule for systems represented by a ratio of polynomials in $\mathcal{L}$.

Consider a system that is described by the following unit-sample response:

$$h_2[n] = \left(\frac{1}{2}\right)^{|n|} \qquad \text{for all } n\,.$$

e. Determine a closed-form functional representation for the system described by $h_2[n]$.

f. Determine the poles of the system in part e.

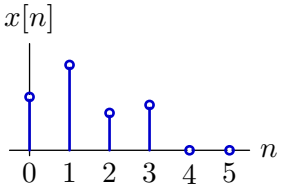
g. Do a series expansion of the system function in part e (it may be helpful to use partial fractions). Explain the relation between the series and the unit-sample response $h_2[n]$.

Engineering Design Problems

5. Upsampling

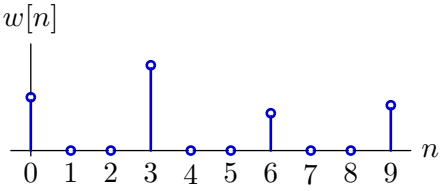
One way to enlarge an image is to upsample by linear interpolation, as follows.

a. Consider a one-dimensional signal $x[n]$ whose non-zero samples are indexed from 0 to N (shown here for $N = 3$).

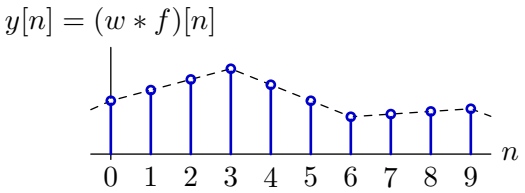


First, make a new signal $w[n]$ defined by

$$w[n] = \begin{cases} x[\frac{n}{3}] & n = 0, 3, 6, \ldots, 3N \\ 0 & \text{otherwise}. \end{cases}$$



Next, convolve $w[n]$ with $f[n]$ to produce $y[n] = (w * f)[n]$.



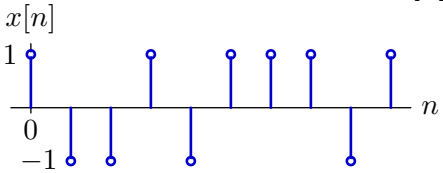
Determine $f[n]$.

b. Extend the linear interpolation method to two dimensions. Write a program that uses this method to enlarge the following image from its current dimensions (212×216) to three times those dimensions. Compare the result of linear interpolation with the result of replicating each pixel value 3×3 times (see appendix).

The digital form of this image is available on the 6.003 website.
The appendix contains sample Python code.

6. Smiley

Consider the sequence of 1's and −1's shown below as $x[n]$.



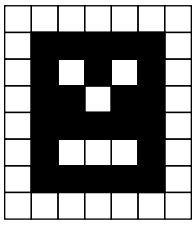
In this $x[n]$, there is a single occurrence of the pattern −1, −1, 1. It occurs starting at $n = 1$ and ending at $n = 3$. One method to automatically locate particular patterns of this type is called "matched filtering." Let $p[n]$ represent the pattern of interest flipped about $n = 0$. Then instances of the pattern can be found by finding the times when $y[n] = (p * x)[n]$ is maximized.

a. Determine a matched filter $p[n]$ that will find occurrences of the sequence: −1, −1, 1. Design $p[n]$ so that $(p * x)[n]$ has maxima at points that are centered on the desired pattern, i.e., at $n = 2$ for the sequence above.

The same approach can be used to find patterns in pictures by generalizing the convolution operator to two dimensions:

$$y[n,m] = (x * p)[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x[k,l]p[n-k,m-l]\,.$$

A file called `findsmiley.jpg` (which can be downloaded from the 6.003 website) contains a random pattern of white pixels (coded as 255) and black pixels (0) as well as a single instance of the following smiley face:



b. Find the row and column of `findsmiley` that corresponds to smiley's nose. Note: matched filtering will work best if matching white pixels AND matching black pixels contribute positively to the answer. For that reason, 255 and 0 may not be the optimum choices for the values associated with white and black pixels.

c. One advantage of the matched filter method is that it works even when the signal contains some noise. One can make a noisy version of `findsmiley` using Python's `gauss` function:

```
import random
...
pix[i,j] += random.gauss(0,dev)
```

where `dev` represents the standard deviation of the added noise.

Determine the largest value of `dev` for which your code still finds smiley.

Appendix

Python program to increase the size of the zebra image by replicating each pixel 3×3 times.

```
import Image as im                                # Python Imaging Library
xx = im.open('zebra.jpg')                         # open input picture
yy = im.new('L',[3*xx.size[0],3*xx.size[1]],0)  # create new picture 3x
xpix = xx.load()                                  # array-type access
ypix = yy.load()
for j in range(yy.size[1]):                       # loop over rows
    for i in range(yy.size[0]):                   #     loop over columns
        ypix[i,j] = xpix[int(i/3),int(j/3)]       #         copy pixels
yy.save('zebraZOH.jpg')                           # create output file
```

The functionality of this code was verified on the Athena linux workstations.