6.003 Homework #5 Solutions

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:

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

Enter expressions for $y_i[n]$ for $n \geq 0$ and for $n < 0$ in the boxes below.

$y_a[n]$ for $n \geq 0$: $n + 1$

$y_a[n]$ for $n < 0$: 0

$$y_a[n] = \sum_{k=-\infty}^{\infty} u[k]u[n-k]$$

$u[k] = 0$ if $k < 0$ and $u[n-k] = 0$ if $n - k < 0$. Therefore

$$y_a[n] = \sum_{k=0}^{n} 1 = \begin{cases} n+1 & \text{if } n \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\boxed{y_a[n] = (n+1)\,u[n]}$$

Notice that convolving any function $g[n]$ with $u[n]$ results in the running sum of $g[n]$:

$$(g * u)[n] = \sum_{k=-\infty}^{\infty} g[k]u[n-k] = \sum_{k=-\infty}^{n} g[k]$$

Thus convolving $u[n]$ with $u[n]$ yields $\sum_{-\infty}^{n} u[k] = (n+1)u[n]$.

$y_b[n]$ for $n \ge 0$: $\boxed{2-\left(\frac{1}{2}\right)^n}$

$y_b[n]$ for $n < 0$: $\boxed{0}$

$$y_b[n]=\sum_{k=-\infty}^{\infty}u[k]\left(\frac{1}{2}\right)^{n-k}u[n-k]=\sum_{k=0}^{n}\left(\frac{1}{2}\right)^{n-k}=\sum_{m=n}^{0}\left(\frac{1}{2}\right)^{m}=\frac{1-\left(\frac{1}{2}\right)^{n+1}}{1-\frac{1}{2}}$$

$$\boxed{y_b[n]=\left(2-\left(\frac{1}{2}\right)^n\right)u[n]}$$

Notice that this is equivalent to the running sum of $\left(\frac{1}{2}\right)^n u[n]$.

$y_c[n]$ for $n \ge 0$: $\boxed{3\left(\frac{1}{2}\right)^n-2\left(\frac{1}{3}\right)^n}$

$y_c[n]$ for $n < 0$: $\boxed{0}$

$$y_c[n]=\sum_{k=-\infty}^{\infty}\left(\frac{1}{3}\right)^{k}u[k]\left(\frac{1}{2}\right)^{n-k}u[n-k]=\left(\frac{1}{2}\right)^{n}\sum_{k=0}^{n}\left(\frac{2}{3}\right)^{k}=\left(\frac{1}{2}\right)^{n}\frac{1-\left(\frac{2}{3}\right)^{n+1}}{1-\frac{2}{3}}$$

$$\boxed{y_c[n]=\left(3\left(\frac{1}{2}\right)^n-2\left(\frac{1}{3}\right)^n\right)u[n]}$$

Notice that this is the same as the unit-sample response of a system with poles at $z=\frac{1}{2}$ and $z=\frac{1}{3}$ (and a double zero at $z=0$).

$y_d[n]$ for $n \ge 0$: $\boxed{(n+1)\left(\frac{1}{2}\right)^n}$

$y_d[n]$ for $n < 0$: $\boxed{0}$

$$y_d[n]=\sum_{k=-\infty}^{\infty}\left(\frac{1}{2}\right)^{k}u[k]\left(\frac{1}{2}\right)^{n-k}u[n-k]=\left(\frac{1}{2}\right)^{n}\sum_{k=0}^{n}(1)^k=(n+1)\left(\frac{1}{2}\right)^{n}$$

$$\boxed{y_d[n]=(n+1)\left(\frac{1}{2}\right)^n u[n]}$$

Notice that this is the same as the unit-sample response of a system with a double pole at $z=\frac{1}{2}$ (and a double zero at $z=0$).

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:

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

Enter expressions for the non-zero parts of $y_i(t)$ and the range of t for which the expression apply in the boxes below.

$y_a(t)$ for $t \geq 0$:	t
$y_a(t)$ for $t < 0$:	0

$$y_a(t) = \int_{-\infty}^{\infty} u(\tau)u(t-\tau)d\tau = \int_0^t d\tau = tu(t)$$

$$\boxed{y_a(t) = tu(t)}$$

Notice that the convolution of any function $g(t)$ with $u(t)$ is equal to the indefinite integral of $g(t)$:

$$(g * u)(t) = \int_{-\infty}^{\infty} g(\tau)u(t-\tau)d\tau = \int_{-\infty}^{t} g(\tau)d\tau$$

Thus $(u * u)(t) = tu(t)$.

$y_b(t)$ for $t \geq 0$:	$1 - e^{-t}$
$y_b(t)$ for $t < 0$:	0

$$y_b(t) = \int_{-\infty}^{\infty} u(\tau)e^{-(t-\tau)}u(t-\tau)d\tau = e^{-t}\int_0^t e^{\tau}d\tau = e^{-t}(e^t - 1) = (1 - e^{-t})u(t)$$

$$\boxed{y_b(t) = (1 - e^{-t})u(t)}$$

Notice that this is the indefinite integral of $e^{-t}u(t)$.

$y_c(t)$ for $t \geq 0$:	$e^{-t} - e^{-2t}$
$y_c(t)$ for $t < 0$:	0

$$y_c(t) = \int_{-\infty}^{\infty} e^{-\tau}u(\tau)e^{-2(t-\tau)}u(t-\tau)d\tau = e^{-2t}\int_0^t e^{\tau}d\tau = e^{-2t}(e^t-1) = \left(e^{-t} - e^{-2t}\right)u(t)$$

$\boxed{y_c(t) = \left(e^{-t} - e^{-2t}\right)u(t)}$

Notice that this is equal to the impulse response of a system with poles at $s = -1$ and $s = -2$.

$y_d(t)$ for $t \geq 0$:	te^{-t}
$y_d(t)$ for $t < 0$:	0

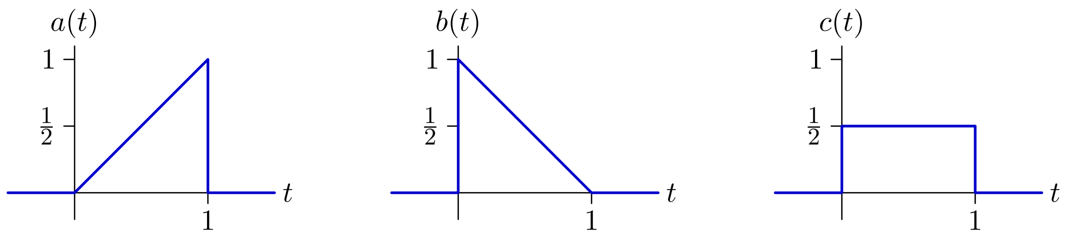
$$y_d(t) = \int_{-\infty}^{\infty} e^{-\tau}u(\tau)e^{-(t-\tau)}u(t-\tau)d\tau = e^{-t}\int_0^t d\tau = e^{-t}t = te^{-t}u(t)$$

$\boxed{y_d(t) = te^{-t}u(t)}$

Notice that this is equal to the impulse response of a system with a double pole at $s = -1$.

3. Convolutions

Consider the convolution of two of the following signals.

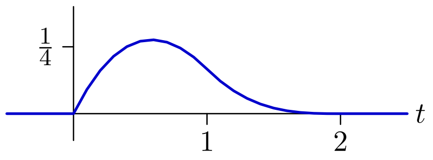


Determine if each of the following signals can be constructed by convolving (a or b or c) with (a or b or c). If it can, indicate which signals should be convolved. If it cannot, put an X in both boxes.

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.

$\boxed{b}$ $*$ $\boxed{b}$ $=$

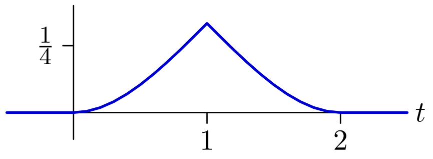
a or b or c or X a or b or c or X



Must be asymmetric with large output at early times and smaller output at later times.

$\boxed{\begin{matrix} a \\ b \end{matrix}}$ $*$ $\boxed{\begin{matrix} b \\ a \end{matrix}}$ $=$

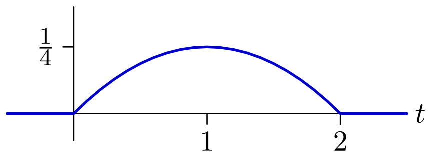
a or b or c or X a or b or c or X



The output is symmetric, which could happen if one of the inputs is a flipped-in-time version of the other input. There are only a few such options. One is $c * c$ — but that would result in a triangle-shaped output. Another symmetric option is $a*b$ (or equivalently $b * a$), which fits with the first interval being concave up.

$\boxed{X}$ $*$ $\boxed{X}$ $=$

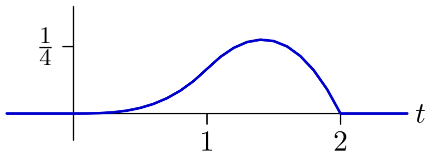
a or b or c or X a or b or c or X



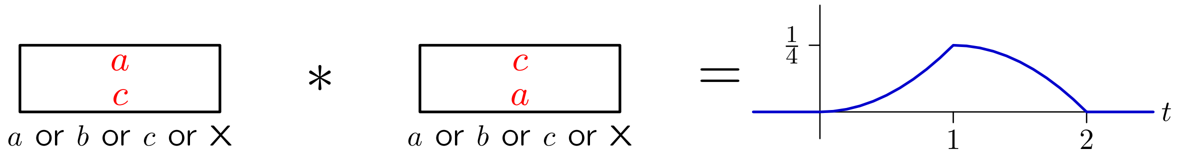
The output is symmetric, which could happen if one of the inputs is a flipped-in-time version of the other input. There are only a few such options. One is $c * c$ — but that would result in a triangle-shaped output. Another symmetric option is $a*b$ (or equivalently $b * a$) — but that would be concave up in first interval. None of the provided functions could result in this output.

$\boxed{a}$ $*$ $\boxed{a}$ $=$

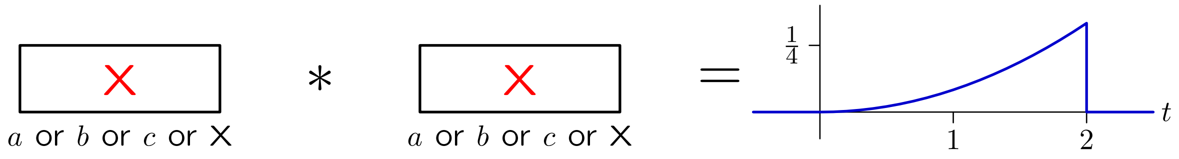
a or b or c or X a or b or c or X



Must be asymmetric with small output at early times and larger output at later times.



The first interval looks like the integral of a. The second interval looks like the first interval, but shifted and flipped in time. Thus the answer is $a * c$ or $c * a$.



The step discontinuity in this result could only result if one of the convolved functions contains an impulse.

Engineering Design Problems

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.

$$y[n] - 3y[n+1] = x[n]$$

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

Assume that the system is anti-causal and starts "at rest" so that $y[n] = 0$ for $n > 0$. Then we can solve the difference equation iteratively:

$$y[n] = x[n] + 3y[n+1]$$

$$\begin{aligned} y[0] &= 1 \\ y[-1] &= 3 \\ y[-2] &= 9 \\ y[-3] &= 27 \\ y[-4] &= 81 \\ &\dots \\ y[-n] &= 3^n \end{aligned}$$

$$y[n] = 3^{-n}u[-n]$$

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

The unit-sample response has the form $z_0^n = 3^{-n}$. Thus the pole is at $z_0 = \frac{1}{3}$.

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 $\mathcal{L}$.

Subsitute $\mathcal{L} \to z$ in the system functional. Reduce to a ratio of polynomials in z. Then the poles are the roots of the denominator.

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 $h_2[n]$.

We can express the functional for $h_2[n]$ as a polynomial in $\mathcal{R}$ and $\mathcal{L}$:

$$H_2 = 1 + \frac{1}{2}\mathcal{R} + \left(\frac{1}{2}\mathcal{R}\right)^2 + \left(\frac{1}{2}\mathcal{R}\right)^3 + \cdots + \frac{1}{2}\mathcal{L} + \left(\frac{1}{2}\mathcal{L}\right)^2 + \left(\frac{1}{2}\mathcal{L}\right)^3 + \cdots$$

We can close the form by recognizing two power series

$$H_2 = \sum_{n=0}^{\infty}\left(\frac{1}{2}\mathcal{R}\right)^n + \sum_{n=0}^{\infty}\left(\frac{1}{2}\mathcal{L}\right)^n - 1 = \frac{1}{1-\frac{1}{2}\mathcal{R}} + \frac{1}{1-\frac{1}{2}\mathcal{L}} - 1 = \frac{1-\frac{1}{4}\mathcal{R}\mathcal{L}}{(1-\frac{1}{2}\mathcal{R})(1-\frac{1}{2}\mathcal{L})}$$

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

Substitute $\mathcal{R} \to \frac{1}{z}$ and $\mathcal{L} \to z$ in the system functional:

$$H_2(z) = \frac{1-\frac{1}{4}\mathcal{R}\mathcal{L}}{(1-\frac{1}{2}\mathcal{R})(1-\frac{1}{2}\mathcal{L})} = \frac{1-\frac{1}{4}}{(1-\frac{1}{2}\frac{1}{z})(1-\frac{1}{2}z)} = \frac{\frac{3}{4}z}{(z-\frac{1}{2})(1-\frac{1}{2}z)}$$

The poles are the roots of the denominator: $z = \frac{1}{2}$ and 2.

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 $h_2[n]$.

Each term in the series expansion

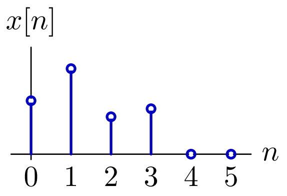
$$H_2 = 1 + \frac{1}{2}\mathcal{R} + \left(\frac{1}{2}\mathcal{R}\right)^2 + \left(\frac{1}{2}\mathcal{R}\right)^3 + \cdots + \frac{1}{2}\mathcal{L} + \left(\frac{1}{2}\mathcal{L}\right)^2 + \left(\frac{1}{2}\mathcal{L}\right)^3 + \cdots$$

corresponds to one point in the unit-sample response.

5. Upsampling

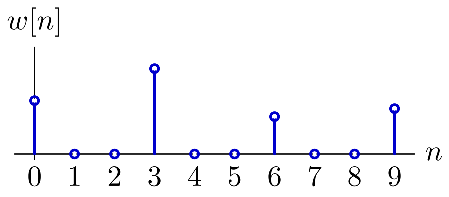
One way to enlarge an image is to upsample by linear interpolation, as follows.

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

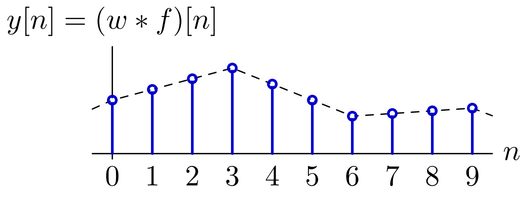


First, make a new signal $w[n]$ defined by

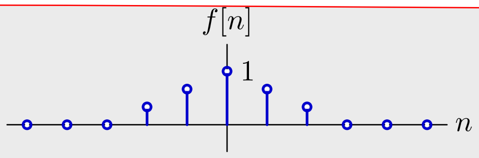
$$w[n] = \begin{cases} x[\frac{n}{3}] & n = 0, 3, 6, \ldots, 3N \\ 0 & \text{otherwise}\,. \end{cases}$$



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.

Program to enlarge image using bilinear interpolation.

```
import Image as im
xx = im.open('zebra.jpg')                            # original image
xpix = xx.load()
aa = im.new('L',[xx.size[0]+2,xx.size[1]+2],0)      # zero-pad edges
aa.paste(xx,(1,1))
apix = aa.load()
bb = im.new('L',[3*aa.size[0]-2,aa.size[1]],0)      # horiz stretch
bpix = bb.load()
cc = im.new('L',[3*aa.size[0]-2,aa.size[1]],0)      # horiz interpo-
lated
cpix = cc.load()
dd = im.new('L',[3*aa.size[0]-2,3*aa.size[1]-2],0)  # vertical stretch
dpix = dd.load()
yy = im.new('L',[3*aa.size[0]-2,3*aa.size[1]-2],0)  # final answer
ypix = yy.load()
for j in range(aa.size[1]):
    for i in range(aa.size[0]):           # stretch image horizontally
        bpix[3*i,j] = apix[i,j]
    for i in range(2,bb.size[0]-2):       # convolve with triangle
        cpix[i,j] = 0.333*bpix[i-2,j]+0.666*bpix[i-1,j]+bpix[i,j]
                 +0.666*bpix[i+1,j]+0.333*bpix[i+2,j]
for i in range(cc.size[0]):
    for j in range(cc.size[1]):           # stretch image vertically
        dpix[i,3*j] = cpix[i,j]
    for j in range(2,dd.size[1]-2):       # convolve with triangle
        ypix[i,j] = 0.333*dpix[i,j-2]+0.666*dpix[i,j-1]+dpix[i,j]
                 +0.666*dpix[i,j+1]+0.333*dpix[i,j+2]
yy.crop([3,3,yy.size[0]-3,yy.size[1]-3]).save('zebraBilinear.jpg')
```

Image produced by code provided in appendix.

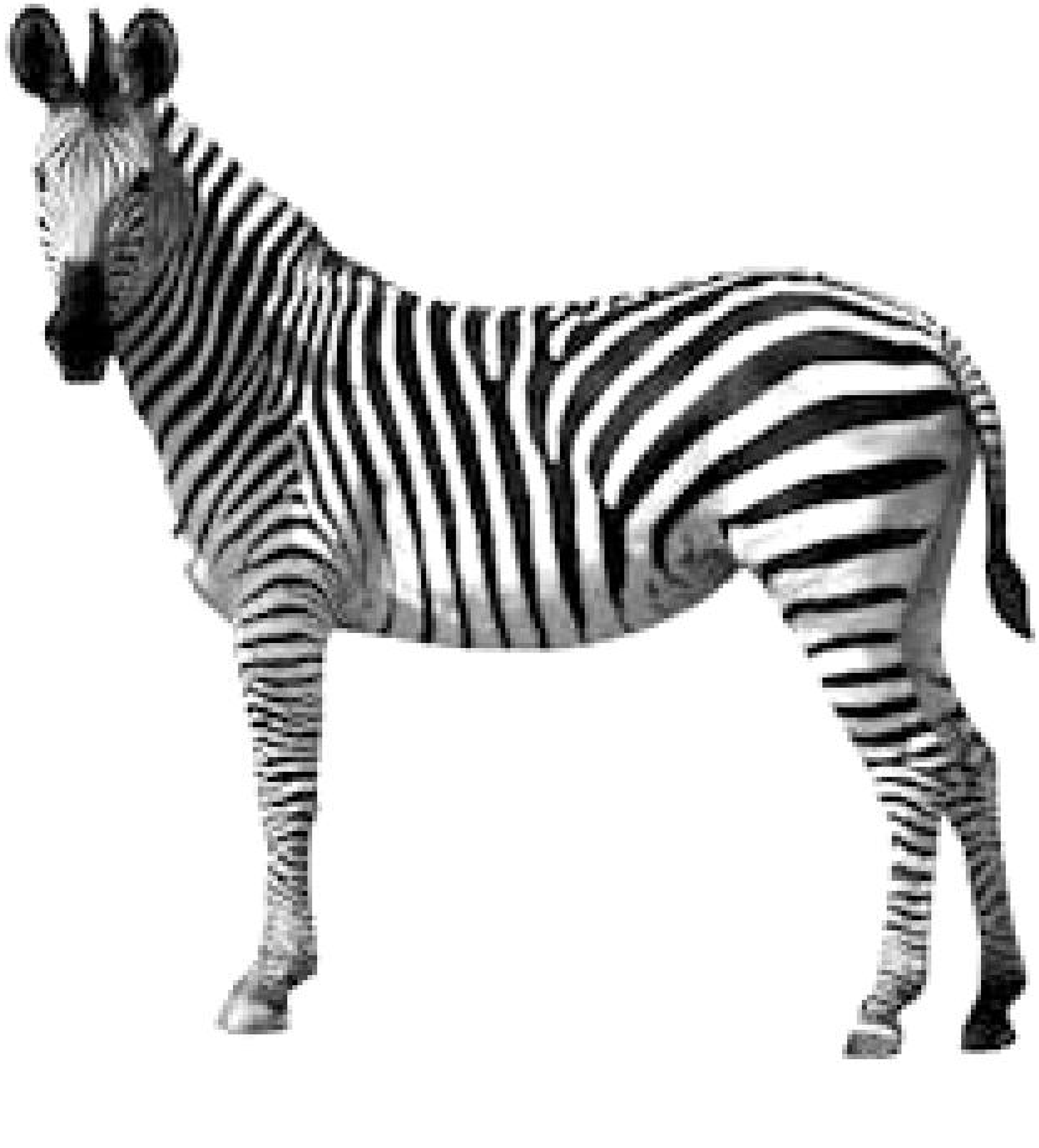
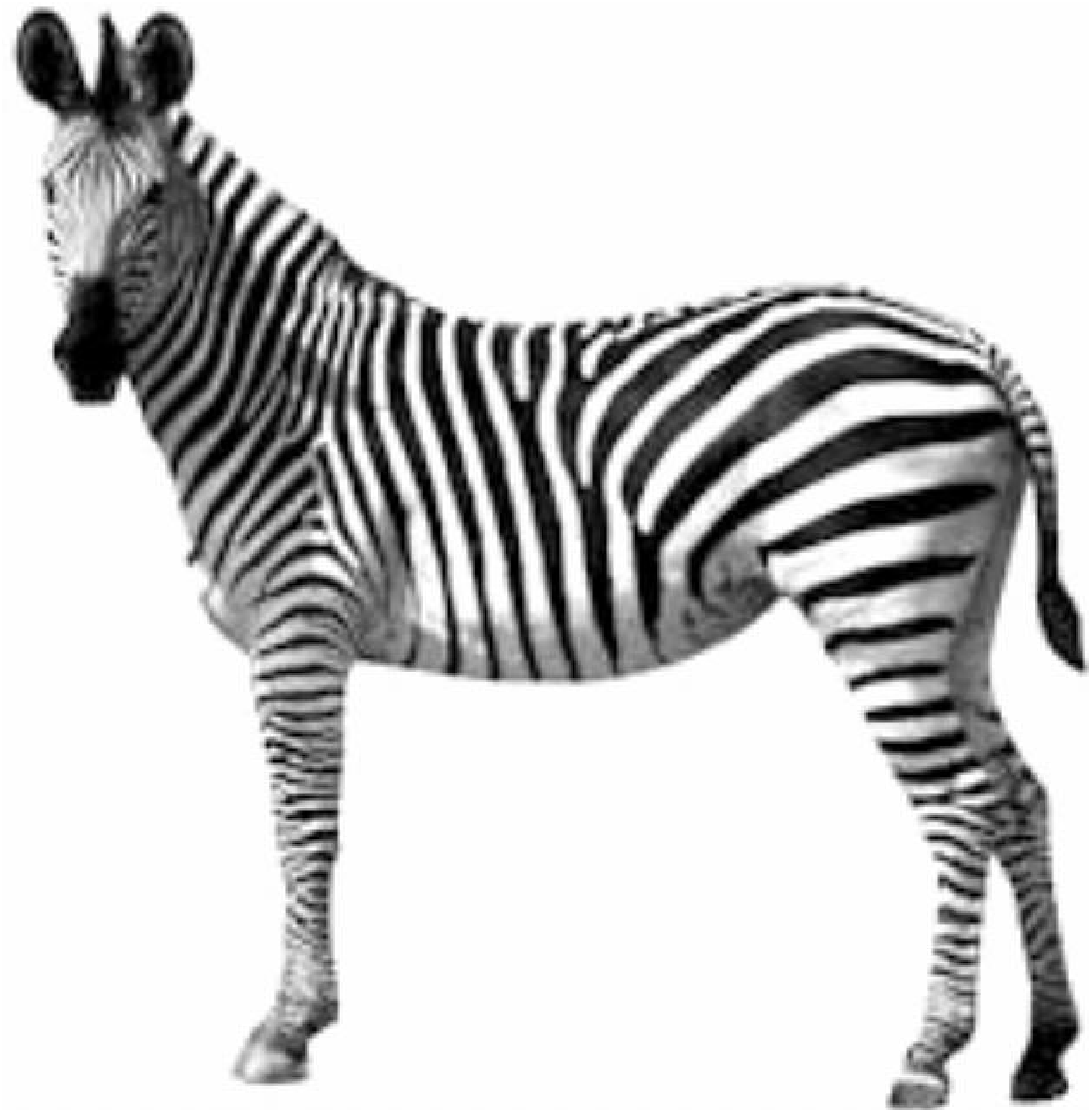


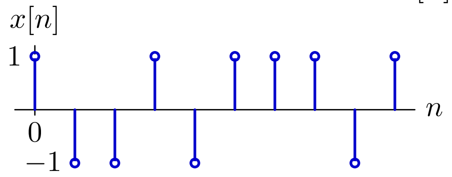
Image produced by bilinear interpolation.



This image has fewer jaggy edges.

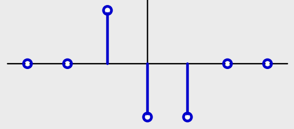
6. Smiley

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.

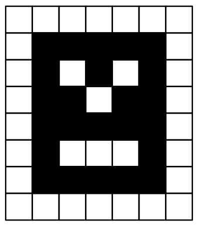
a. Determine a matched filter $p[n]$ that will find occurrences of the sequence: −1, −1, 1. Design $p[n]$ so that $(p * x)[n]$ has maxima at points that are centered on the desired pattern, i.e., at $n = 2$ for the sequence above.



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.

Program to locate nose.

I made an image file `smiley.jpg` to simplify the code. (By treating the desired target as a file, it was easy to debug the code with a different (simpler) image in a different file.)

Notice that 127.5 is subtracted from each pixel value before applying the convolution operation. This results in the following possible outputs (per pixel):

image pixel	smiley pixel	result
black	black	$(-127.5) * (-127.5) = +127.5^2$
white	white	$(127.5) * (127.5) = +127.5^2$
black	white	$(-127.5) * (127.5) = -127.5^2$
white	black	$(-127.5) * (127.5) = -127.5^2$

which were chosen so that both types of matches give the same output, both types of mismatches give the same output, and matches and mismatches give opposite signs.

Also, 127.5 was added back to the result, so that the result of the convolution could be viewed as an image `yy`.

```
import Image as im

xx = im.open('findsmiley.jpg')
xwidth,xheight = xx.size
ww = im.open('smiley.jpg')
wwidth,wheight = ww.size
yy = im.new('L',[xwidth-wwidth+1,xheight-wheight+1])
ywidth,yheight = yy.size
xpix = xx.load()
wpix = ww.load()
ypix = yy.load()

xmax = -10000
coords = [0,0]
for j in range(xheight-wheight+1):
    for i in range(xwidth-wwidth+1):
        ss = 0
        for jj in range(wheight):
            for ii in range(wwidth):
                ss += (xpix[i+ii,j+jj]-127.5)*(wpix[ii,jj]-127.5)/127.5+127.5
        ypix[i,j] = ss/wwidth/wheight
        if ss>xmax:
            xmax = ss
            coords = [i,j]
print coords[0]+3,coords[1]+3
yy.save('answer.jpg')
```

column = 765; row = 432.

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.

Program to add noise to image and then find nose:

```
import Image as im
import random

dev = 150

xx = im.open('findsmiley.jpg')
xwidth,xheight = xx.size
ww = im.open('smiley.jpg')
wwidth,wheight = ww.size
yy = im.new('L',[xwidth-wwidth+1,xheight-wheight+1])
ywidth,yheight = yy.size
xpix = xx.load()
wpix = ww.load()
ypix = yy.load()

for j in range(xx.size[1]):
    for i in range(xx.size[0]):
        xpix[i,j] += random.gauss(0,dev)

xmax = -10000
coords = [0,0]
for j in range(xheight-wheight+1):
    for i in range(xwidth-wwidth+1):
        ss = 0
        for jj in range(wheight):
            for ii in range(wwidth):
                ss += (xpix[i+ii,j+jj]-127.5)*(wpix[ii,jj]-127.5)/127.5+127.5
        ypix[i,j] = ss/wwidth/wheight
        if ss>xmax:
            xmax = ss
            coords = [i,j]
print coords[0]+3,coords[1]+3
yy.save('answer.jpg')
```

This program reliably locates smiley even if `dev` is as high as 150.[1] This noise level is HUGE: the standard deviation is greater than half the difference between white and black!

Matched filtering works very well in this application.

[1] A different noise sequence is generated each time this program is run. Therefore, repeated runs with the same value of `dev` may produce different results.