Issued February 16, 2005
Part I of Pre-Labs are due February 28, 2005 by the **beginning** of your section
Part II of Pre-Labs are due March 7, 2005 by the **beginning** of your section
Lab reports are due March 14, 2005 by the **beginning** of your section

Lab 5b: Convolution and Deconvolution and Deblurring Images [1]

1. Introduction

Suppose you went for a holiday, saw beautiful scenery, and came back with a snapshot that looked like this:

Figure 1: A blurred picture.

[1] This lab is based upon portions of labs originally developed by Professor Peter Mathys in Spring 2002 and Professor John Hauser in Fall 2004. Modifications were made by Weerawat Khawsuk and Lucy Pao in Fall 2002. Further modifications have been made by Harish Venkatachari and Lucy Pao in February 2005.

What does that have to do with convolution and deconvolution? If the blurred image can be modeled as the result of a perfect image convolved with the impulse response of an LTI system (*e.g.*, an imperfect or defocused lens), then deconvolution with the right parameters might actually result in the recovery of the original sharp image. More generally, a situation that is often encountered in practice is that one is interested in a DT sequence $x[n]$, *e.g.*, a sequence of temperature measurements, or a sequence of grey values from a line of a scanned image, but $x[n]$ can only be observed indirectly after having passed through a sensor. Thus, rather than observing $x[n]$ directly, only $y[n] = h[n] * x[n]$ can be obtained in the best case when the sensor is (reasonably well) modeled as a LTI system with unit impulse response $h[n]$. The usual approach in such situations is to try to compute an estimate $\hat{x}[n]$ from $y[n]$ using some form of inverse system with unit impulse response $h_{inv}[n]$ so that

$$\hat{x}[n] = h_{inv}[n] * y[n] = h_{inv}[n] * h[n] * x[n] \, .$$

This is shown graphically in the following block diagram.

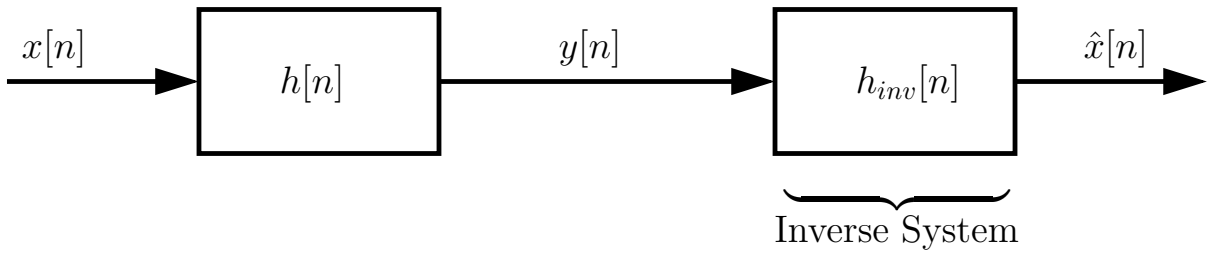

Figure 2: Estimation of $\hat{x}[n]$ from $y[n]$ using an inverse system.

Ideally (if it exists), $h_{inv}[n]$ is computed such that

$$h_{inv}[n] * h[n] = \delta[n] \qquad \text{or} \qquad h_{inv}[n] * h[n] = \delta[n - N] \, ,$$

where in the first case $\hat{x}[n] = x[n]$, and in the second case $\hat{x}[n] = x[n - N]$ for some positive integer N. Because of noise, nonlinearities, and other imperfections it is usually not possible in real applications to make $\hat{x}[n]$ exactly equal to the original $x[n]$ or a delayed version of it, but most of the time appropriate processing can yield substantially better results than using $y[n]$ directly.

A simple model for blurring that works quite well in the simplified case of one-dimensional bitmap image processing (*e.g.*, processing an image line by line) is in the form of a (non-causal) DT LTI system with unit impulse response

$$h[n] = K \, a^{|n|} \, , \quad -\infty < n < \infty \, , \quad |a| < 1 \, .$$

When the impulse response $h[n] \neq 0$ for an infinite number of indices n, it is called an Infinite Impulse Response or IIR. When there are only a finite number of n where $h[n] \neq 0$, then

$h[n]$ is called a Finite Impulse Response or FIR. While the above $h[n]$ is IIR, in practice it can be implemented approximately as an FIR.

The unit step response $g[n]$ (corresponding to a transition from black to white in an image) is easily obtained using direct convolution as

$$g[n] = h[n] * u[n] = \sum_{m=-\infty}^{\infty} h[m]\, u[n-m] = \sum_{m=-\infty}^{n} h[m] = K \sum_{m=-\infty}^{n} a^{|m|} \,.$$

Thus, if $n \leq 0$

$$g[n] = K \sum_{m=-\infty}^{n} a^{-m} = K \left(a^{-n} + a^{-n+1} + a^{-n+2} + \ldots\right) = K\, a^{-n} \sum_{i=0}^{\infty} a^{i} = K \frac{a^{-n}}{1-a} \,.$$

For $n \geq 0$

$$g[n] = K \left[\sum_{m=-\infty}^{-1} a^{-m} + \sum_{m=0}^{n} a^{m}\right] = K \left[\frac{a}{1-a} + \frac{1-a^{n+1}}{1-a}\right] = K \frac{1+a-a^{n+1}}{1-a} \,.$$

As $n \to \infty$

$$\lim_{n\to\infty} g[n] = K \frac{1+a}{1-a} \,,$$

assuming $|a| < 1$. Thus, setting $K = (1-a)/(1+a)$ preserves the final value of a transition from black to white. Note that

$$g[n] - g[n-1] = K\, a^{|n|} \,, \quad \text{for all } n \,; \quad \text{and} \quad \frac{g[n]}{g[n-1]} = a^{-1} \,, \quad \text{for } n \leq 0 \,.$$

Thus, a strategy to determine K and a from a (black and white) image that was blurred using the $h[n]$ given above is to find a transition from black to white and determine the location of the maximum difference between two adjacent pixels. This corresponds to $|g[n] - g[n-1]|$ at $n = 0$, and thus the difference is equal to K. Taking the ratio $g[n]/g[n-1]$ at $n = 0$ then yields $1/a$. An example where $K = 1/3$ and $a = 1/2$ is shown in Figure 3, both in the form of images and in the form of sequences of pixel values that make up a line in the image.

The script file that was used in Matlab to produce the images and plots in Figure 3 is given below and serves as an example of how to deal with images in Matlab.

```
%blurx01  Image Blur, Example 1
% One-dimensional (line) unit step response of system with
%       1 - a
% h_n = ----- * a^(|n|) ,    -N <= n <= +N ,   |a| < 1
%       1 + a

% Originally written by P. Mathys, April 7, 2002.

a = 0.5;                                % Decay parameter
```

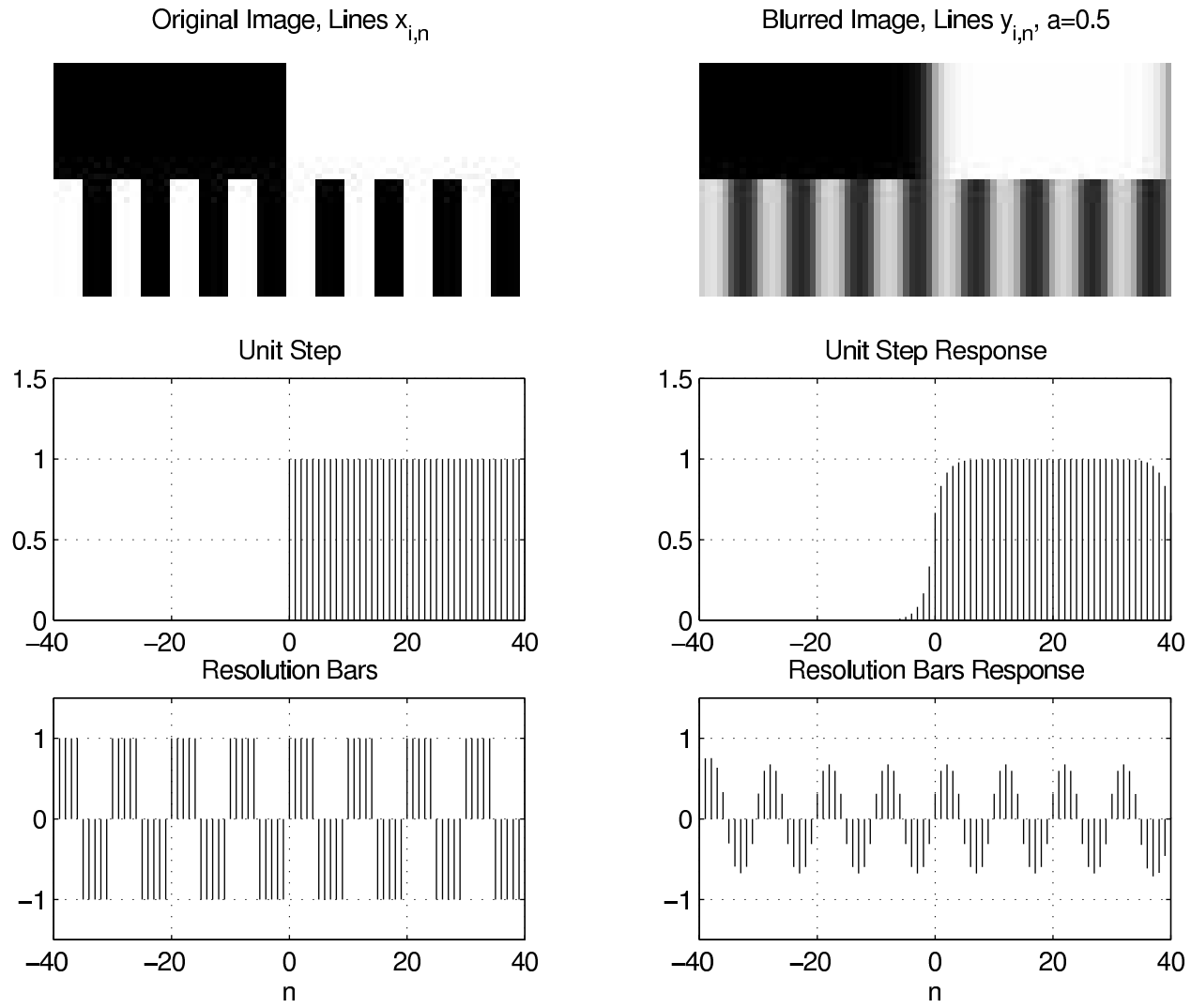

Figure 3: Demonstration of blurring an image using $K = 1/3$ and $a = 1/2$.

```
N = 20;                                 % Truncation index for h_n
nmin = -40;                             % Left end index of image
nmax = 40;                              % Right end index of image
nlin = 20;                              % Number of lines per image section
F0 = 0.1;                               % Frequency for resolution bars

n = [nmin:nmax];                        % Image index axis
ustep = zeros(size(n));                 % Prepare unit step
ix = find(n>=0);
ustep(ix) = ones(size(ix));             % Unit step
xn = sign(sin(2*pi*F0*n+1e-3));         % Resolution bars

IMGx = 256*ones(nlin,1)*ustep;          % Unit step image
```

```
IMGx = [IMGx;128*ones(nlin,1)*(xn+1)]; % Resolution bar image

nn = [-N:N];                           % hn index axis
hn = ((1-a)/(1+a))*(a.^(abs(nn)));     % Unit impulse response
gn = conv(ustep,hn);                   % Unit step response
gn = gn(N+1:end-N);                    % gn shifted and truncated
yn = conv(xn,hn);                      % Resolution bar response
yn = yn(N+1:end-N);                    % yn shifted and truncated

IMGy = 256*ones(nlin,1)*gn;            % Unit step response image
IMGy = [IMGy;128*ones(nlin,1)*(yn+1)]; % Resolution bar response

bwmap=[0:255]'/255*ones(1,3);          % Colormap for black and white image
colormap(bwmap);                       % Invoke gray scale colormap
subplot(321); image(IMGx);
axis off; axis equal;
title('Original Image, Lines x_{i,n}');

subplot(322); image(IMGy);
axis off; axis equal;
title(['Blurred Image, Lines y_{i,n}, a=' num2str(a)]);

subplot(323); stem(n,ustep,'.-b');
grid; title('Unit Step'); ylim([0 1.5]);

subplot(324); stem(n,gn,'.-b');
grid; title('Unit Step Response'); ylim([0 1.5]);

subplot(325); stem(n,xn,'.-b');
grid; title('Resolution Bars'); xlabel('n'); ylim([-1.5 1.5]);

subplot(326); stem(n,yn,'.-b');
grid; title('Resolution Bars Response');
xlabel('n'); ylim([-1.5 1.5]);
```

Basically, a bitmap image is represented in Matlab in the form of a matrix, with rows corresponding to lines in the image. A matrix element with a given row and column index corresponds to a pixel in the image. The value of each pixel, which is either a single number or a triplet of numbers, determines the luminance and/or the color of the pixel. If a triplet of numbers is given then this corresponds directly to the R,G,B (red, green, blue) values of the pixel. If only a single number per pixel is given then this corresponds to an index number (usually in the range $[0 \ldots 255]$) and the corresponding R,G,B triplet is stored in a separate `colormap` matrix (usually of size 256×3). The default colormap (in other contexts colormaps are often called palettes) for black and white images with 256 grey values is constructed using the command

```
bwmap=[0:255]'/255*ones(1,3);
```

and then invoked using the `colormap(bwmap)` command. The bitmap image itself is displayed using the `image(X)` command, where `X` is the matrix that holds the image. Use the following Matlab script to test whether you are using the right colormap for black and white images:

```
% colormap_test.m
% Tests colormap currently in use for displaying images

% Originally written by P. Mathys, April 10, 2002.

X = [0:255];              % Gray scale, 0: black, 255: white
subplot(211);
image(X);                 % Display grayscale in X
title(['"colormap" Test, Should Show Gray Scale from',...
' Black (0) to White (255)']);
```

The result should look the same as in the following figure.

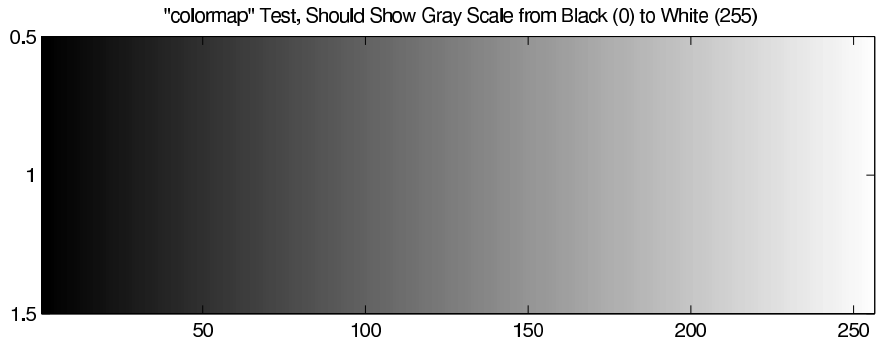

Figure 4: Grayscale colormap.

Images can be read from files and stored to files in various formats using the `imread` and `imwrite` commands. For details see the Matlab help files. One word of caution when dealing with the processing of images that should be stated here is that in most cases image data is non-negative, whereas many filtering algorithms use both negative and positive numbers, internally at least. This can lead to unexpected results if not dealt with properly.

The main question, however, that still remains to be answered is how an image that was blurred using $h[n] = K\,a^{|n|}$, for all n, can be "unblurred" or recovered. For this one has to to compute $H_{inv}(z)$. Once $H_{inv}(z)$ is known $h_{inv}[n]$ can be computed and the operation of deblurring reduces to one of convolving $h_{inv}[n]$ with every row of pixels. In this lab, you will compute $H_{inv}(z)$ and complete a number of experiments on convolution and deconvolution and image deblurring.

2. Lab Experiments

You should complete experiments **E1(b)**, **E1(c)** in your lab section on February 28, and experiments **E2(b)**, **E2(c)**, and **E4** should be completed in your lab section on March 7. Because a full lab report is required for this in-depth lab, there are no TA check offs. The TAs will of course walk around during the lab sections to make sure that everyone is hopefully on the right track. Beware that the full lab report will take much longer to compile and write up compared to the brief lab reports that you have been doing for the basic labs so far. The lab reports for Lab 5 are due at the beginning of your section on March 14.

Pre-Lab:
Part I: Read carefully through the lab handout and complete experiment **E1(a)**. Turn in all handwritten calculations, Matlab code, and result plots for **E1(a)** in your pre-lab. Your TA will discuss this lab in section on February 23. This part of the pre-lab must be turned in to your TA at the beginning of your lab section on February 28. Make a copy of your pre-lab before turning it in if you will need it to complete the lab during your lab section. If you turn in your pre-lab in your TAs mailbox by noon on February 27 (email your TA so he knows to look for it), your TA will grade your pre-lab and return it to you at the beginning of your lab section on February 28. To ensure that you are progressing correctly, show your work and results on **E1(a)** to your TA at the beginning of section on February 28 before you proceed with the rest of **E1**.

Part II: Complete experiments **E2(a)** and **E3**. Turn in all handwritten calculations, Matlab code, and result plots for **E2(a)** and **E3** in your pre-lab. This part of the pre-lab must be turned in to your TA at the beginning of your lab section on March 7. Make a copy of your pre-lab before turning it in if you will need it to complete the lab during your lab section. If you turn in your pre-lab in your TA's mailbox by noon on March 6 (email your TA so he knows to look for it), your TA will grade your pre-lab and return it to you at the beginning of your lab section on March 7.

E1. Continuous-Time Convolution in Matlab. Write a Matlab function that takes two signals:

- an impulse response $h(t)$
- an input signal $x(t)$

and produces the corresponding output repsonse $y(t) = h(t) * x(t)$.

The function should begin with

```
function [ty, y] = convolution(th, h, tx, x)
%
% function [ty, y] = convolution(th, h, tx, x)
%
% calculate the continuous time convolution y = h*x
%
```

```
% inputs:
%
% th  time sequence for impulse response
% h  impulse response signal
%
% tx  time sequence for input signal
% x  input signal
%
% output:
%
% ty  time sequence for output signal
% y  output signal
%
% Notes:
%
% 1. The signal specification, e.g., tx, x, is such that
%
%    figure(1), plot(tx, x), grid on, zoom on
%
%    will produce a nice plot of the signal.
%
% 2. The time sequences for each signal should be equally spaced, e.g.,
%
%    tx = (0:0.01:10)'; x = sin(2*pi*tx/10);
%    % note that tx is a column vector
%
%    Assume equal sampling rate for both x(t) and h(t).
%
% 3. Each signal is identically zero outside of the specified region.
```

Now develop different versions of the convolution function that approximate the convolution integral using

(a) Piecewise constant functions: Since Matlab can store $x(t)$ and $h(t)$ only at discrete values of t, you can think of the arrays `x` and `h` in Matlab as containing "samples" of $x(t)$ and $h(t)$.

Assume that the sampled functions $x(t)$ and $h(t)$ are constant (= the sample value) over a sampling interval.

Write a numerical integration scheme in Matlab for computing the convolution integral exactly when this assumption holds. Provide your well-documented Matlab code and explain your numerical integration method in your pre-lab. Test your convolution function out on several pairs of $h(t)$ and $x(t)$ (which may or may not satisfy the above assumption), including the following:

(i)

```
>> th = (0:0.01:5)';
>> h = 3*exp(-3*th);
>> figure(1),plot(th,h), grid on,zoom on
>> title('h(t)'), xlabel('t'), ylabel('h')
>> tx = (0:0.01:20)';
>> x = sin(2*tx);
>> [ty, y] = convolution(th, h, tx, x);
>> figure(2), plot(tx,x,ty,y), grid on, zoom on,
>> title('x(t) and y(t)'), xlabel('t'), ylabel('x and y')
```

(ii) $x(t) = u(t) - u(t-3)$ and $h(t) = u(t) - u(t-3)$

(iii) $x(t) = u(t) - u(t-3)$ and $h(t) = 3e^{-3t}u(t)$

Work each convolution out by hand as well and compare. Provide your hand calculations as part of your pre-lab.

Devise a number of tests to verify that your system ($h(t)$) is linear and time invariant. Explain these tests and provide results from some experiments verifying that your test works in your pre-lab. Also, determine and explain how to test whether an LTI system is causal or not. Explain this test and test it out on some systems, such as

```
>> th = (-10:.01:10)';
>> h = sinc(th);
>> figure(1), plot(th,h), grid on, zoom on
>> title('h(t)'), xlabel('t'), ylabel('h')
```

which is the truncated impulse response of an ideal low pass filter. This is an important impulse response that will be discussed more in class later in the semester. The sinc function is defined as:

$$\text{sinc}(t) = \frac{\sin(\pi t)}{\pi t}.$$

(b) Trapezoid rule: Write a convolution function that makes a straight-line approximation of the samples of the integrand in the convolution integral. That is, the samples in the function to be integrated are interpolated through straight lines and the area under each of the intervals is computed.

(c) Exact integration for piecewise linear functions: Write a convolution function that computes the convolution integral "exactly" for piecewise linear functions as described below and in Figure 5.

The following equations for $x(\tau)$ and $h(t-\tau)$ represent the lines joining x_i to x_{i+1}

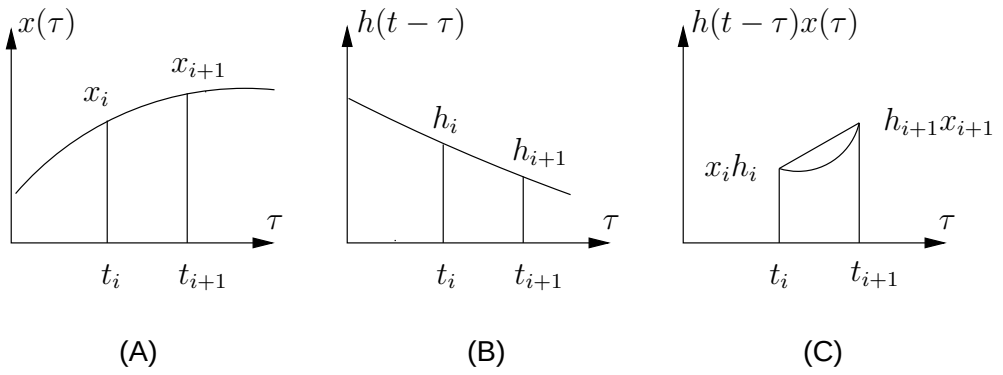

Figure 5: Diagrams illustrating exact integration method. The trapezoid rule yields the area under the straight line in (C) while the "exact" integration method yields the area under the curve in (C).

and h_i to h_{i+1} over the interval t_i to t_{i+1}, as shown in Figure 5:

$$x(\tau) = \left(\frac{x_{i+1}-x_i}{\Delta t}\right)(\tau - t_i) + x_i, \qquad \text{for } t_i \le \tau < t_{i+1}$$

$$h(t-\tau) = \left(\frac{h_{i+1}-h_i}{\Delta t}\right)(\tau - t_i) + h_i, \qquad \text{for } t_i \le \tau < t_{i+1}$$

where $\Delta t = t_{i+1} - t_i$. Letting $s = \tau - t_i$, we now have that

$$\int_{t_i}^{t_{i+1}} x(\tau)h(t-\tau)d\tau = \int_0^{\Delta t}\left[\left(\frac{x_{i+1}-x_i}{\Delta t}\right)s + x_i\right]\left[\left(\frac{h_{i+1}-h_i}{\Delta t}\right)s + h_i\right]ds$$

$$= \frac{1}{3}(x_{i+1}-x_i)(h_{i+1}-h_i)\Delta t + \frac{1}{2}x_i(h_{i+1}-h_i)\Delta t + \frac{1}{2}h_i(x_{i+1}-x_i)\Delta t + h_ix_i\Delta t$$

The area of each of interval is given by the formula above. Use this formula to implement the "exact" integration method.

As part of your lab report,

- Give a step-by-step discussion of each convolution function and its merits (as compared to the others) for each of the methods explored:
  - Piece-wise constant
  - Trapezoid rule
  - Exact Integration
- How does the time interval (`th` and `tx`) affect the results?
- Provide plots of $y(t)$ for the test cases given in part (a) above and give arguments as to why it is correct/incorrect. Compare your results with your hand calculations.

- How do you verify that the systems ($h(t)$) involved are linear, time-invariant, and/or causal? Explain and include the results of these tests in your lab report.

E2. Discrete-Time Convolution in Matlab. The command for performing (one dimensional) DT convolution between two vectors `v1` and `v2` in Matlab is `conv(v1,v2)`. You have already used this command to multiply numerator or denominator polynomials of system functions together. In signal processing applications the `conv` command is used to convolve an input signal `xn` with the unit impulse response `hn` of a DT system to produce the output signal `yn=conv(hn,xn)`. An important issue in this case is to keep track of the time indexes associated with the input and output signals. For example, because the length of `yn` is equal to the length of `xn` plus the length of `hn` minus one, it is not entirely trivial to make a plot that shows both `xn` and `yn` simultaneously.

(a) The goal here is to convolve two DT rectangular pulses defined by

$$f[n] = \begin{cases} A\,, & M_1 \le n \le M_2, \\ 0\,, & \text{otherwise}, \end{cases} \qquad g[n] = \begin{cases} B\,, & N_1 \le n \le N_2, \\ 0\,, & \text{otherwise}, \end{cases}$$

to obtain $x[n] = f[n] * g[n]$, and to display all three DT signals on a common time index axis n. Write a Matlab script file using the `conv` command and make plots and check them for correctness for the following sets of parameters:

Parameters for Convolutions $x[n] = f[n] * g[n]$						
Case	M_1	M_2	A	N_1	N_2	B
(i)	13	7	1/5	−9	−3	1/3
(ii)	−8	0	1/4	1	6	1/2
(iii)	−4	3	1/3	0	9	1/5
(iv)	−3	3	1	−12	12	1/10

You should obtain similar plots as shown in Figure 6.

(b) Suppose you take a rectangular DT pulse

$$f[n] = \begin{cases} A\,, & -M \le n \le M, \\ 0\,, & \text{otherwise}, \end{cases} \qquad M \ge 0,$$

centered at $n = 0$ and convolve it N times with itself to obtain $x[n]$, *i.e.*,

$$x[n] = \underbrace{f[n] * f[n] * \ldots * f[n]}_{N \text{ times}}\,.$$

Would the result converge in some sense to a known shape? Write a Matlab script file and make and interpret plots for $N = 2, 3, 10, 100$, and $M = 0, 1, 2$.

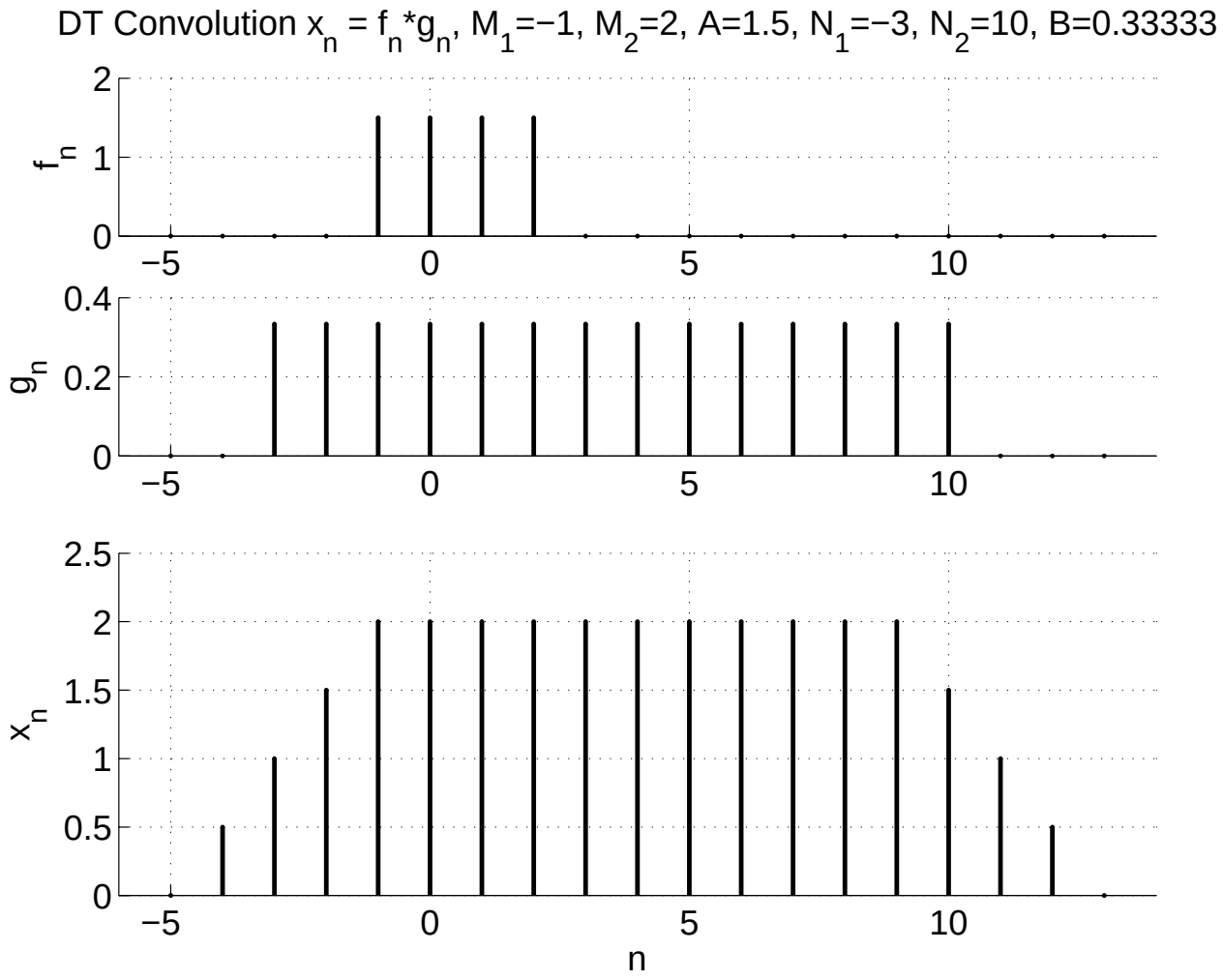

Figure 6: A DT convolution example for $M_1 = -1$, $M_2 = 2$, $A = 1.5$, $N_1 = -3$, $N_2 = 10$, $B = 1/3$.

How does $A > 0$ have to be chosen such that $\sum_n x[n] = 1$, independent of N? A characteristic sample plot for $M = 2$ and $N = 10$ is shown in Figure 7. (Hint: This experiment demonstrates one of the fundamental theorems of probability. What is it?)

(c) Repeat part **(b)**, but instead of a fixed amplitude A for $f[n]$, use individual uniformly distributed amplitude values in the range $0 \ldots 1$ for each of the nonzero components of $f[n]$. That is, use the following Matlab command to generate `fn`

$$\texttt{fn} = \texttt{rand}(1, 2 * \texttt{M} + 1);$$

and then convolve `fn` N times with itself to obtain $x[n]$. How does that affect the shape of $x[n]$? What conclusions can you draw?

E3. Compute $H_{inv}(z)$ and $h_{inv}[n]$ for the deblurring system. Is it an FIR or an IIR? How will you use it to deblur your image? Make sure you are on the right track by showing your work and results for this part to your TA before you proceed further with the lab.

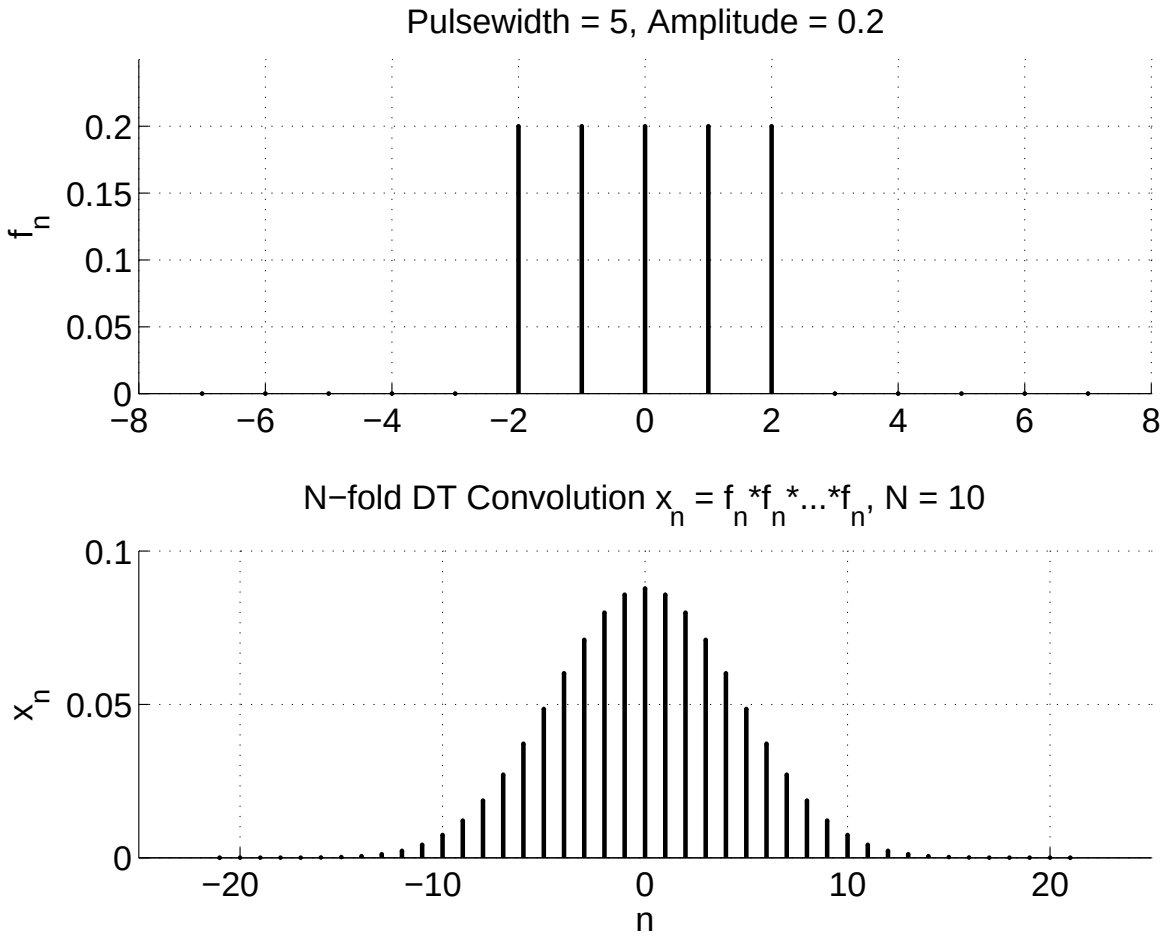

Figure 7: N-fold DT convolution of rectangular pulses when $M = 2$, $N = 10$.

E4. Deconvolution of Blurred Images.

(a) The file `Blur01.mat` (posted on the course website) contains the black and white image in Figure 8 that has been blurred line by line using a DT LTI system with unit impulse response

$$h[n] = K\, a^{|n|}\,, \quad -\infty < n < \infty\,, \quad |a| < 1\,.$$

Download the `Blur01.mat` file and read it into Matlab. The array of a blurred image is accessible via the `IMGy` variable. To display it as shown in Figure 8, try the following

```
load('Blur01.mat');
image(uint8(IMGy));
bwmap = [0:255]'/255*ones(1,3);
colormap(bwmap);
```

The `uint8` command converts the `double` array of `IMGy` into unsigned 8-bit integers, which has values from 0 (corresponds to black) to 255 (corresponds to white).

Figure 8: A black and white image that has been blurred line by line. Deblur this image and find out how taking ECEN 3300 might make you famous.

In practice, images are stored in many formats such as `.bmp, .jpg, .tif,` and `.png` files. For images in these formats, you need to convert unsigned integers to real numbers using `double` before you do any processing.

Examine a line with a characteristic black to white (or dark gray to light gray) transition and estimate K and a from it. Then write a Matlab script file for deblurring the image. The unit impulse response $h_{inv}[n]$ of the inverse system for deblurring will contain positive and negative numbers. Before you display the deblurred image, make sure to convert real numbers back into `uint8` integers. Give your estimates for the parameters a and K as well as the unblurred image in your report. Explain the process of how you obtained these parameter estimates.

(b) The image in the file `Blur02.mat` shown in Figure 9 contains a "secret" message that was made unreadable using the same blurring function but for different a and K parameters. Can you make the message readable?

Figure 9: A black and white image of a secret message that has been blurred line by line.