GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 1999
Lab #5: Potpourri: A/D and D/A

Date: 28 Sept–4 Oct 1999

This is *the official* Lab#5 description.

Lab Quiz #2 will be given at the beginning of Lab #6 (next week).

Complete the survey either before or during lab, and have it signed off.

The Warm-up section of each lab must be completed in Lab and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by initialing on the **Instructor Verification** line. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the instructor.

The lab report for this week will an **Informal Lab Report.** Staple the **Instructor Verification** sheet to the end of your lab report as evidence that the appropriate steps were witnessed by the instructor.

The report will **due during the week of 5–11 Oct. at the start of your lab**.

---

1 Overview

There are three objectives/activities in this lab:

1. One objective of this lab is to introduce more complicated signals that are related to the basic sinusoid. These signals which implement frequency modulation (FM) and amplitude modulation (AM) are widely used in communication systems such as radio and television, but they also can be used to create interesting sounds that mimic musical instruments. There are a number of demonstrations on the CD-ROM that provide examples of these signals for many different conditions.

CD-ROM
FM Synthesis

2. Use linear-FM chirps to exhibit aliasing

3. Use digital images to illustrate the D-to-A reconstruction process. In this lab, we will show a commonly used method of reconstruction that gives "poor" results—a later lab will revisit this issue and do a better job.

1.1 Frequency Modulated Signals

We will also look at signals in which the frequency varies as a function of time. In the constant-frequency sinusoid $A\cos(2\pi f_0 t + \phi)$ the argument of the cosine is also the exponent of the complex exponential, so the phase of this signal is the exponent $(2\pi f_0 t + \phi)$. This phase function changes *linearly* versus time, and its time derivative is $2\pi f_0$ which equals the constant frequency of the cosine.

A generalization is available if we adopt the following notation for the class of signals with time-varying phase:

CD-ROM
FM Synthesis

$$x(t) = A\cos(\psi(t)) = \Re e\{Ae^{j\psi(t)}\} \quad (1)$$

The time derivative of the phase from (1) gives a frequency

$$\omega_i(t) = \frac{d}{dt}\psi(t) \qquad \text{(rad/sec)}$$

but we prefer units of hertz, so we divide by 2π to define the *instantaneous frequency:*

$$f_i(t) = \frac{1}{2\pi}\frac{d}{dt}\psi(t) \qquad \text{(Hz)} \tag{2}$$

1.2 Chirp, or Linearly Swept Frequency

CD-ROM

Spectrograms & Sounds: Wide-band FM

A *chirp* signal is a sinusoid whose frequency changes linearly from some starting value to an ending frequency. The formula for such a signal can be defined by creating a complex exponential signal with quadratic phase by defining $\psi(t)$ in (1) as

$$\psi(t) = 2\pi\alpha t^2 + 2\pi f_0 t + \phi$$

The derivative of $\psi(t)$ yields an instantaneous frequency (2) that changes *linearly* versus time.

$$f_i(t) = 2\alpha t + f_0$$

The slope of $f_i(t)$ is equal to 2α and its intercept is f_0. If the signal starts at $t = 0$, then f_0 is also the starting frequency. The frequency variation produced by the time-varying phase is called *frequency modulation,* or simply FM. Finally, since the linear variation of the frequency can produce an audible sound similar to a siren or a chirp, the linear-FM signals are also called "chirps." If $\alpha = 0$ there is no quadratic term in $\psi(t)$ and the instantaneous frequency $f_i(t)$ reduces to the constant frequency f_0.

1.3 Advanced Topic: Spectrograms

It is often useful to think of signals in terms of their spectra. A signal's spectrum is a representation of the frequencies present in the signal. For a constant frequency sinusoid the spectrum consists of two spikes, one at $2\pi f_0$, the other at $-2\pi f_0$. For more complicated signals, the spectra may be very interesting and, in the case of FM, the spectrum is considered to be time-varying. One way to represent the time-varying spectrum of a signal is the *spectrogram* (see Chapter 3 in the text). A spectrogram is found by estimating the frequency content in short sections of the signal. The magnitude of the spectrum over individual sections is plotted as intensity or color on a two-dimensional plot versus frequency and time.

CD-ROM

Sounds & Spectrograms

There are a few important things to know about spectrograms:

1. In MATLAB the functions `specgram` and `plotspec` will compute the spectrogram, as already explained in the previous lab. Type `help specgram` or `help plotspec` to learn more about these functions and their arguments *which are similar but not exactly the same*.

2. Spectrograms are numerically calculated and only provide a *numerical estimate* of the time-varying frequency content of a signal. There are theoretical limits on how well they can actually represent the frequency content of a signal. Also "estimate" means that the spectrogram image depends on the choice of parameters used in calling `specgram.` Lab 11 (in the book) treats this problem in the course of programming a method that uses the spectrogram to extract the frequencies of piano notes.

1.4 Digital Images

In this lab we introduce digital images as a signal type for studying the effect of sampling, aliasing and reconstruction. An image can be represented as a function $x(t_1, t_2)$ of two continuous variables representing the horizontal (t_2) and vertical (t_1) coordinates of a point in space.[1] For monochrome images, the $x(t_1, t_2)$ would be a scalar function of the two spatial variables, but for color images the function $x(\cdot, \cdot)$ would have

[1]The variables t_1 and t_2 are confusing since they *do not denote time*.

to be a vector-valued function of the two variables.[2] Moving images (such as TV) would add a time variable to the two spatial variables.

Monochrome images are displayed using black and white and shades of gray, so they are called *gray-scale* images. In this lab we will consider only sampled gray-scale still images. A sampled gray-scale still image would be represented as a two-dimensional array of numbers of the form

$$x[m,n] = x(mT_1, nT_2) \qquad\qquad 1 \le m \le M, \text{ and } 1 \le n \le N$$

where T_1 and T_2 are the sample spacings in the horizontal and vertical directions. Typical values of M and N are 256 or 512; e.g., a 512×512 image which has nearly the same resolution as a standard TV image. In MATLAB we can represent an image as a matrix consisting of M rows and N columns. The matrix entry at (m,n) is the sample value $x[m,n]$—called a *pixel* (short for picture element).

An important property of light images such as photographs and TV pictures is that their values are always non-negative and finite in magnitude; i.e.,

$$0 \le x[m,n] \le X_{\max}$$

This is because light images are formed by measuring the intensity of reflected or emitted light which must always be a positive finite quantity. When stored in a computer or displayed on a monitor, the values of $x[m,n]$ have to be scaled relative to the maximum value $X_{\max}$. Usually an eight-bit integer representation is used. With 8-bit integers, the maximum value (in the computer) would be $X_{\max} = 2^8 - 1 = 255$, and there are $2^8 = 256$ gray levels for the display, from 0 to 255.

1.5 Displaying Images

As you will discover, the correct display of an image on a gray-scale monitor can be tricky, especially after some processing has been performed on the image. We have provided the function `show_img.m` in the *DSP First Toolbox* to handle most of these problems,[3] but it will be helpful if the following points are noted:

CD-ROM
show_img.m

1. All image values must be non-negative for the purposes of display. Filtering may introduce negative values, especially if differencing is used (e.g., a high-pass filter).

2. The default format for most gray-scale displays is eight bits, so the pixel values $x[m,n]$ in the image must be converted to integers in the range $0 \le x[m,n] \le 255 = 2^8 - 1$.

3. The actual display on the monitor is created with the `show_img` function.[4] The `show_img` function will handle the color map and the "true" size of the image. The appearance of the image can be altered by running the pixel values through a "color map." In our case, we want grayscale so all three primary colors (red, green and blue, or RGB) are used equally, creating what is called a "gray map." In MATLAB the `gray` color map is set up via

   ```
   colormap(gray(256))
   ```

   which gives a 256×3 matrix where all 3 columns are equal. The function `colormap(gray(256))` creates a linear mapping, so that each input pixel amplitude is rendered with a screen intensity proportional to its value (assuming the monitor is calibrated). For our experiments, non-linear color mappings would introduce an extra level of complication, so we won't use them.

[2]For example, an RGB color system needs three values for red, green and blue at each spatial location.

[3]If you have the image processing toolbox, then the function `imshow.m` can be used.

[4]If the MATLAB function `imagesc.m` is used to display the image, two features will be missing: (1) the color map may be incorrect because it will not default to gray, and (2) the size of the image will not be a true pixel-for-pixel rendition of the image on the computer screen.

4. When the image values lie outside the range [0,255], or when the image is scaled so that it only occupies a small portion of the range [0,255], the display may have poor quality. In this lab, we will use `show_img.m` to *automatically rescale the image:* This requires a linear mapping of the pixel values:[5]

$$x_s[m,n] = \mu x[m,n] + \beta$$

The scaling constants μ and β can be derived from the min and max values of the image, so that all pixel values are recomputed via:

$$x_s[m,n] = \left\lfloor 255.999 \left(\frac{x[m,n] - x_{\min}}{x_{\max} - x_{\min}} \right) \right\rfloor$$

where $\lfloor x \rfloor$ is the floor function, i.e., the greatest integer less than or equal to x.

1.6 Printing Multiple Images

The phrase "what you see is what you get" can be elusive when dealing with images. It is VERY TRICKY to print images so that the hard copy matches exactly what is on the screen, because there is usually some interpolation being done by the printer or program that is handling the images. One way to think about this in signal processing terms is that the screen is one kind of D-to-A and the printer is another kind, but they use different (D-to-A) reconstruction methods to get the continuous-time output image that you see.

Furthermore, if you try to put two images of different sizes into subplots of the same MATLAB figure, it won't work. Therefore, you should display your images in separate figure windows. In order to get a printout with MULTIPLE IMAGES ON THE SAME PAGE, use the following procedure:

1. In MATLAB, use `show_img` and `trusize` to put your images into separate figure windows at the correct pixel resolution.
2. Use the Windows-95 program called `PAINT` to assemble the different images onto one page. This program can be found under `Accessories.`
3. For each MATLAB figure window, do `ALT-PRINT-SCREEN` which will copy the active window contents to the clipboard.
4. After each "window capture" in step 3, paste the clipboard contents into `PAINT.`
5. Arrange the images so that you can make a comparison for your lab report.
6. Print the assembled images from `PAINT` to a printer.

2 Warm-up

The instructor verification sheet may be found at the end of this lab.

2.1 Warm-up: Display of Images

CD-ROM
IMAGE DATA FILES

You can load the images needed for this lab from `*.mat` files. Any file with the extension `*.mat` is in MATLAB format and can be loaded via the `load` command. To find some of these files, look for `*.mat` in the *DSP First* toolbox or in the MATLAB directory called `toolbox/matlab/demos`. Some of the image files are named `lenna.mat`, `echart.mat` and `zone.mat`, but there are others within MATLAB's demos.

[5] The MATLAB function `show_img` has an option to perform this scaling while making the image display.

The default size is 256×256, but alternate versions are available as 512×512 images under names such as `lenna512.mat` and `zone512.mat`. After loading, use the command `whos` to determine the name of the variable that holds the image and its size.

Although MATLAB has several functions for displaying images on the CRT of the computer, we have written a special function `show_img()` for this lab. It is the visual equivalent of `soundsc()`, which we used when listening to speech and tones; i.e., `show_img()` is the "D-to-C" converter for images. This function handles the scaling of the image values and allows you to open up multiple image display windows. Here is the `help` on `show_img`:

CD-ROM
show_im.m

```
function [ph] = show_img(img, figno, scaled, map)
%SHOW_IMG    display an image with possible scaling
% usage:  ph = show_img(img, figno, scaled, map)
%    img = input image
%    figno = figure number to use for the plot
%             if 0, re-use the same figure
%             if omitted a new figure will be opened
% optional args:
%    scaled = 1 (TRUE) to do auto-scale (DEFAULT)
%           not equal to 1 (FALSE) to inhibit scaling
%    map = user-specified color map
%     ph = figure handle returned to caller
%----
```

Notice that unless the input parameter `figno` is specified, a new figure window will be opened.

2.2 Display Test

In order to probe your understanding of image display, do the following simple displays:

(a) Generate a simple test image in which all of the rows are identical by using the following outer product:

```
xpix = ones(256,1)*cos(2*pi/8*(1:256));
```

Display the image and explain the gray-scale pattern that you see. How wide are the bands in number of pixels? How can you predict that width from the formula for `xpix`?

(b) Now load and display the 256×256 "lenna" image from `lenna.mat`. The command `load lenna` will put the sampled image into the array `xx`. Use `whos` to check the size of `xx` after loading.

(c) Use the colon operator to extract the 200^{th} row of the "lenna" image, and make a plot of the 200^{th} row as a 1-D discrete-time signal. Observe that the range of signal values is between 0 and 255—which values represent white and which ones black?

Instructor Verification (separate page)

3 Lab Exercises: Sampling, Aliasing and Reconstruction

3.1 FM Signals: Chirps

The following MATLAB code will synthesize a chirp:

```
fsamp = 8000;
dt = 1/fsamp;
dur = 1.0;
```

```
tt = 0 : dt : dur;
psi = 2*pi*(-1200*tt.*tt + 3700*tt);
xx = real( exp(j*psi) );
soundsc( xx, fsamp );
```

(a) Determine the mathematical formula for the chirp signal $x(t)$ created by this function.

(b) Make a sketch by hand of the instantaneous frequency $f_i(t)$ (in Hz) versus time (t) in secs.

(c) Determine the range of frequencies (in Hz) that will be synthesized by this MATLAB script. What are the minimum and maximum frequencies that will be heard?

(d) Display the spectrogram of your chirp using the MATLAB function: `specgram(xx,[],fsamp)`.

(e) Listen to the signal and describe its frequency content (use `soundsc()`). Is it what you expect?

3.2 Chirps and Aliasing

Now we can use the chirp signal to illustrate aliasing and folding. We do this by trying to make the instantaneous frequency of a chirp signal cover a very wide range. In this part, you will investigate what happens when the instantaneous frequency tries to go well above the sampling frequency.

(a) Use the previous MATLAB code as a model to make an M-file that synthesizes the following "chirp" signal:

  (i) A total time duration of 2.5 secs.
  (ii) The *desired* instantaneous frequency starts at 0 Hz and ends at 15,000 Hz.
  (iii) Use a sampling rate and D/A conversion rate of $f_s = 8000$ Hz.
  (iv) Include the code with your lab report.

(b) Listen to the signal. Record your observations regarding the sound of the chirp. Does it chirp down, or chirp up, or both? How many times?

(c) Create a spectrogram of your chirp signal.

(d) Use the sampling theorem (from Chapter 4 in the text) and the concepts of aliasing and folding to help explain what you see in the spectrogram and how it relates to what you heard.

(e) In addition, make some theoretical calculations to support your explanation in part (c).

  (i) Make a sketch of the instantaneous frequency (in Hz) that you are *attempting* to synthesize by your MATLAB script. For this part ignore the effects of sampling.
  (ii) Make a sketch (by hand) of the *actual* instantaneous frequency versus time. This is for the signal out of the D/A converter—sampling makes an impact.
  (iii) In part (d) you explained how *aliasing* affects the instantaneous frequency that is actually heard. To show that you understand the role of the sampling theorem, explain what would happen if the sampling rate were increased to 11,025 Hz.
  (iv) Listen to the signal with $f_s = 11,025$ Hz to verify that it has the expected frequency content.

3.3 Sampling of Images

Images that are stored in digital form on a computer have to be sampled images because they are stored in an $M \times N$ array (i.e., a matrix). The sampling rate in the two spatial dimensions was chosen at the time the image was digitized (in units of samples per inch if the original was a photograph). For example, the image might have been "sampled" by a scanner where the resolution was chosen to be 300 dpi (dots per inch).[6] If we want a different sampling rate, we can simulate a *lower* sampling rate by simply throwing away samples in a periodic way. For example, if every other sample is removed, the sampling rate will be halved (in our example, the 300 dpi image would become a 150 dpi image). Usually this is called *sub-sampling* or *down-sampling*.

> *Down-sampling:* If the vector `x1` represents a signal such as a row of an image, we can reduce the sampling rate by a factor of 4 by simply taking every 4^{th} sample. In MATLAB this is easy with the colon operator, i.e., `xdown = x1(1:4:length(x1))`. The vector `xdown` will be one fourth the length of `x1` (use `whos` to verify this fact).

(a) *Down-sampling* throws away samples, so it will shrink the size of the image. This is what is done by the following scheme:

```
xp = xx(1:p:M,1:p:N);
```

when we are downsampling by a factor of `p`. One potential problem with down-sampling is that aliasing might occur. This can be illustrated in a dramatic fashion with the `barbara` image, and to a lesser degree with the `lenna` image.

Load the `barbara` image which has the image stored in a variable called `barb`. When you check the size of the image, you'll find that it is not square. Now down-sample the `barbara` image by a factor of 2. Notice the aliasing in the down-sampled image, which is surprising since no new values are being created by the down-sampling process. Describe how the aliasing appears visually.[7]

(b) This part is hard: explain why the aliasing happens by using a "frequency domain" explanation. In other words, estimate the frequency of the features that are being aliased. This frequency will be in cycles per pixel. Can you relate your frequency estimate to the Sampling Theorem?

You might try zooming in on a very small region of both the original and downsampled images.

(c) Down-sample the `lenna` image by a factor of 2, and also by factors of 4 and 8. Notice that this image seems to be relatively unaffected by the down-sampling by 2 process, but eventually the larger down-sampling factors do make a difference. What can you say about the frequency content of the `lenna` image as opposed to the `barbara` image?

3.4 Reconstruction of Images

When an image has been sampled, we can fill in the missing samples by doing interpolation. For images, this would be analogous to the examples shown in Chapter 4 for sine-wave interpolation which is part of the reconstruction process in a D-to-A converter. We could use a "square pulse" or a "triangular pulse" or other pulse shapes for the reconstruction.

---

[6]The Sampling Theorem applies to digital images, so there is a *Nyquist Rate* that depends on the maximum *spatial* frequency in the image.

[7]One difficulty with showing aliasing is that we must display the pixels of the image exactly. This almost never happens because most monitors and printers will perform some sort of interpolation to adjust the size of the image to match the resolution of the device. In MATLAB we can override these size changes by using the function `truesize` which is part of the Image Processing Toolbox. In the *DSP First Toolbox*, an equivalent function called `trusize.m` is provided.

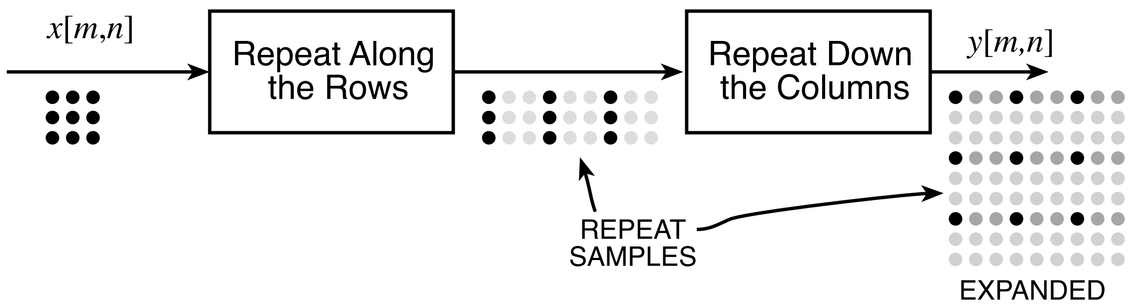

Figure 1: 2-D Interpolation broken down into row and column operations: the gray dots indicate repeated data values created by a zero-order hold; or, in the case of linear interpolation, they are the interpolated values.

For these reconstruction experiments, use the `lenna` image, down-sampled by 4 (Section 3.3). The objective will be to reconstruct the `lenna` image which is 256×256 from the 64×64 down-sampled image.

(a) The simplest interpolation would be reconstruction with a square pulse which produces a "zero-order hold." Here is a method that works for a one-dimensional signal (i.e., one row or one column of the image), assuming that we start with a row vector `x1`, and the result is the row vector `zz`.

```
x1 = (-2).^(0:4);
L = length(x1);
nn = floor( 1:(1/4):L+0.75 );   %<-- Truncate to the integer part
zz = x1(nn);
```

Plot the vector `zz` to verify that it is a zero-order hold version derived from `x1`. Explain what values are contained in the indexing vector `nn`. Your lab report should include the explanation for this part, but no plots are required.

(b) Now return to the down-sampled `lenna` image, and process all the rows of `xx` to fill in the missing points. Use the zero-order hold idea from part (a). Call the result `yhold`. Display `yhold` as an image, and compare it to the original image `xx`.

(c) Now process all the columns of `yhold` to fill in the missing points in each column and compare the result to the original image `xx`. Include your code for parts (b) and (c) in the lab report.

(d) *Linear interpolation* can be done in MATLAB using the `interp1` function. Its default mode is linear interpolation, which is equivalent to using the `'*linear'` option, but `interp1` can also do other types of polynomial interpolation. Here is an example on a 1-D signal:

```
n1 = 0:4;
x1 = (-2).^n1;
tti = 0:0.1:4;   %-- locations between the n1 indices
stem(tti,interp1(n1,x1,tti))
```

(e) Carry out a linear interpolation operation on both the rows and columns of the image. This requires two calls to the `interp1` function, because one call will only process all the columns of a matrix. Name the interpolated output image `ylin`. Compare `ylin` to the original image `xx` and to the square pulse interpolated image from part (c). Comment on the visual appearance of the two "reconstructed" images. Include your code for this part in the lab report.

When unsure about a command, use `help`.

(f) Compare the quality of the linear interpolation result to the zero-order hold result. Point out regions where they differ and try to justify this difference by estimating the local frequency content. In other words, look for regions of "low-frequency" content and "high-frequency" content and see how the interpolation quality is dependent on this factor.

Comment: You might use zooming to show the "zero-order hold" effect in a small patch of the output image. However, zooming does its own interpolation, probably a zero-order hold.

3.5 More about Images in MATLAB (Optional)

For more information on the image processing functions in MATLAB, try help:

```
help images
```

but keep in mind that the Image Processing Toolbox, which is available in the CoC labs, may not be on your computer.

3.5.1 Zooming in Software

If you have used an image editing program such as Adobe's "Photoshop," you might have observed how well or how poorly image zooming (i.e., interpolation) is done. For example, if you try to blow up a JPEG file that you've downloaded from the web, the result is usually disappointing. Since MATLAB has the capability to read lots of different formats, you can apply the image zooming via interpolation to any photograph that you can acquire. The MATLAB function for reading JPEG images is `imread( )` which would be invoked as follows:

```
xx = imread('foo.jpg','jpeg');
```

Since `imread( )` is part of the image processing toolbox, this test can be done in the CoC computer labs, but may not be possible on your home computer.

3.5.2 Warnings

Images obtained from JPEG files might come is many different formats. Two precautions are necessary:

1. If MATLAB loads the image and stores it as 8-bit integers, then MATLAB will use an internal data type called `uint8`. The function `show_img( )` cannot handle this format, but there is a conversion function called `double( )` that will convert the 8-bit integers to double-precision floating-point for use with filtering and processing programs.

   ```
   yy = double(xx);
   ```

   You can convert back to 8-bit values with the function `uint8()`.

2. If the image is a color photograph, then it is actually composed of three "image planes" and MATLAB will store it as a 3-D array. For example, the result of `whos` for a 545×668 color image would give:

   ```
   Name       Size          Bytes  Class
   xx       545x668x3    1092180  uint8 array
   ```

   In this case, you should use MATLAB's image display functions such as `imshow( )` to see the color image. Or you can convert the color image to gray-scale with the function `rgb2gray( )`. For more information on the image processing functions in MATLAB, try help:

   ```
   help images
   ```

Lab #5
ECE-2025
Fall-99
INSTRUCTOR VERIFICATION PAGE

Staple this page to the end of your Lab Report.

Name: ____ Date of Lab: __________

Complete the on-line survey in Web-CT:

Verified:____________________ Date/Time:__________

Part 2.2 Load and display a digital image; extract one row of the image. Explain the width of bands in the test image formed from the "outer product" of ones with a cosine.

Verified:____________________ Date/Time:__________