GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Spring 2025
Lab #4: Sampling of Signals and Aliasing

Date: 24 Feb. - 07 Mar.2025

Special Lab Instructions in Spring 2025: The labs will be held in person (or remotely through BlueJeans when needed). Students registered in a particular lab session are required to be present at the designated times. Attendances will be taken before each class. A total of six labs will be conducted. Each lab will typically last two weeks. The first week is devoted to students' **Q&As** and the second is for students' **demos** to the instructors for codes and lab results. Students will be grouped into teams of 4 to 5 by instructors before starting Lab 1. For each lab session, there will be two instructors administrating all activities. Each team will be given a breakout period of 15 minutes in each session for Q&As and verifications. Students are encouraged to discuss lab contents in teamwork. At the end of each lab, each student are required to turn in an individual lab report in a single pdf fill, containing answers to all lab questions, including codes and plots. Georgia Tech's **Honor Code** will be strictly enforced. See CANVAS Assignments for submission instructions.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students, but you cannot give or receive any written material or electronic files. In addition, you are not allowed to use or copy material from old lab reports from previous semesters. Your submitted work must be your own original work.

---

1 Overview

Please read through the information below prior to attending your lab. The 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. In addition, they 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. The resulting signal can be analyzed to show its time-frequency behavior by using the *spectrogram.*

CD-ROM
FM Synthesis

One objective in this lab is to study sampling and aliasing with simple signals: sinusoids and chirps. We will use a MATLAB GUI for sampling and aliasing, called `con2dis`, which tracks an input sinusoid and its spectrum through A/D and D/A converters. This demo is part of the *SP-First* Toolbox. Another objective in this lab is to introduce digital images as a second useful signal type. We will show how the A-to-D sampling and the D-to-A reconstruction processes are carried out for digital images.

1.1 Review: Theory of Sampling, A-to-D and D-to-A Conversion

In this lab, the short-duration sinusoids and music signals will be created with the intention of playing them out through a speaker. Therefore, it is necessary to take into account the fact that a conversion is needed from the digital samples, which are numbers stored in the computer memory to the actual continuous waveform that will be amplified and heard through the speakers (or headphones).

Chapter 4 treats sampling in detail, but this lab is usually done prior to lectures on sampling, so we provide a quick summary of essential facts here. The idealized process of sampling a signal and the subsequent reconstruction of the signal from its samples is depicted in Fig. 1. This figure shows a continuous-time input signal $x(t)$, which is sampled by the continuous-to-discrete (C-to-D) converter to produce a sequence of samples $x[n] = x(nT_s)$, where n is the integer sample index and T_s is the sampling period. The sampling

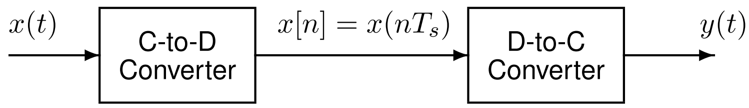

Figure 1: Sampling and reconstruction of a continuous-time signal.

rate (in samples per second) is $f_s = 1/T_s$. The discrete-to-continuous (D-to-C) converter creates a continuous output signal $y(t)$ from the samples $x[n]$. As described in Chapter 4 of the text, the ideal D-to-C converter takes the signal samples $x(nT_s)$ and interpolates a smooth curve through them. The *Sampling Theorem* tells us that when the input signal $x(t)$ is a sum of sine waves, the output $y(t)$ will be equal to the input $x(t)$ if the sampling rate is *more than twice the highest frequency* $(f_{\max})$ in the input, i.e., we need $f_s > 2f_{\max}$. In other words, if we *sample fast enough* then there will be no problems resynthesizing the continuous audio signal from $x[n]$.

Most computers have a built-in hardware for the analog-to-digital (A-to-D) converter and the digital-to-analog (D-to-A) converter (usually with a sound card or chip). These hardware systems are physical realizations of the idealized concepts of C-to-D and D-to-C converters, respectively, but for purposes of this lab we will assume that the hardware A/D and D/A are perfect realizations.

The digital-to-analog conversion process has many engineering design issues, but in its simplest form the only thing we need to worry about (in this lab) is that the time spacing (T_s) between the signal samples must correspond to the rate of the D-to-A hardware that is being used. From MATLAB, the sound output is done by the `soundsc(xx,fs)` function which does support a variable D-to-A sampling rate `(fs)` to the extent that the hardware on the machine has such variable rate capability. A convenient choice for the D-to-A conversion rate is 11025 samples per second,[1] so $T_s = 1/11025$ seconds; another common choice is 8000 samples/sec. Both of these rates satisfy the requirement of *sampling fast enough* as required by the Sampling Theorem. In fact, most piano notes have relatively low frequencies, so sampling rates much lower thatn 8000 samples/sec could be used. If you are using `soundsc()`, the vector `xx` will be *rescaled automatically* so that its maximum value equals the maximum allowed by the D-to-A converter, but if you are using `sound.m`, you must scale the vector `xx` so that it lies between ± 1. Consult `help sound.`

1.2 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.[2] For monochrome images, the signal $x(t_1, t_2)$ would be a scalar function of the two spatial variables, but for color images the function $x(\cdot, \cdot)$ would have to be a vector-valued function of the two variables. For example, an RGB color system needs three values at each spatial location: one for red, one for green and one for blue. Video or TV which consists of a sequence of images to show motion 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. which can be represented as a two-dimensional array of numbers of the form

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

[1]This sampling rate is one quarter of the 44,100-Hz rate used in audio CD players.

[2]The variables t_1 and t_2 *do not denote time, they represent spatial dimensions.* Thus, their units would be inches or some other unit of length.

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 frame. In MATLAB we can represent an image as a matrix, so it would consist 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 are also finite in magnitude; i.e.,

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

This is because light images are formed by measuring the intensity of reflected or emitted light, and intensity 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 a 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 would be $2^8 = 256$ gray levels for the display, from 0 to 255.

1.3 Displaying Images

As you will discover, the correct display of an image on a computer monitor can be tricky, especially if the processing performed on the image generates negative values. We have provided the function `show_img.m` in the *SP-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 when a first-difference 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 $0 \leq x[m, n] \leq 255 = 2^8 - 1$.

3. The actual display on the monitor created with the `show_img` function[4] 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 a "grayscale display" where 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 lab experiments, non-linear color mappings would introduce an extra level of complication, which we will avoid.

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 use `show_img.m` to *automatically rescale the image to use the full range of pixel value:* We can do this by applying a linear mapping of the pixel values:[5]

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

[3] If you have the MATLAB Image Processing Toolbox, then the function `imshow.m` can be used instead.

[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.

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

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.

Below is the `help` on `show_img`; notice that unless the input parameter `figno` is specified, a new figure window will be opened each time `show_img` is called.

```
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
%----
```

1.4 Digital Camera Color Imaging

Digital cameras are now ubiquitous because most cell phones come equipped with cameras pictures with digital cameras. When a digital image is recorded, the camera needs to perform a significant amount of processing in order to provide the user with a viewable image.[6] With a little knowledge of DSP, we can investigate some of that processing.

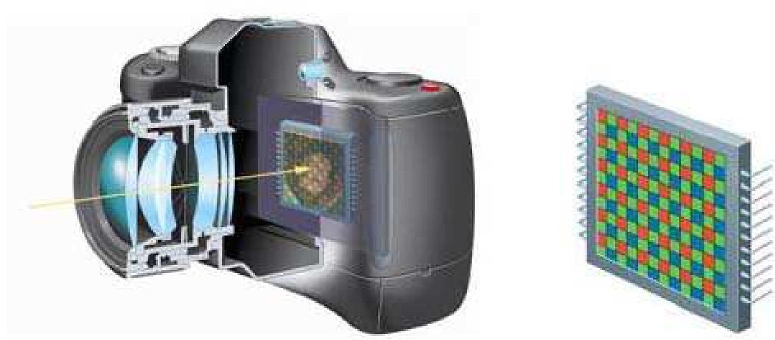

Figure 2: CFA in the digital camera (left), CFA module (right).

A color image requires at least three color values at each pixel location, usually red, green and blue for a computer image. Ideally, a camera would use three separate sensors in order to measure the red, green and blue intensities at every single pixel location. To reduce size and cost, many cameras use a single sensor

[6]Ref: B. K. Gunturk, J. Glotzbach, Y. Altunbasak, R. W. Schafer, R. M. Mersereau, "Demosaicking: Color Filter Array Interpolation in Single-Chip Digital Cameras," Center of Signal and Image Processing, Georgia Institute of Technology, Available at `http://users.ece.gatech.edu/ rmm/fall2003/ece6258/Demosaicking_SPM.pdf`

array in conjunction with a color filter array. The color filter array (CFA) will pass the red, the green or the blue component of light to a given pixel location on the sensor array. This means that the initially captured image matrix does not contain full color information at any pixel location; rather, any one pixel contains only red, or green, or blue intensity information for that pixel. Therefore, the camera must estimate the two missing color values at each pixel, and this estimation process is known as *demosaicking*.

1.4.1 Bayer CFA

Although several possible patterns exist for the CFA, the Bayer pattern is the one that is most commonly used.[7] As shown in Fig. 3, the Bayer CFA measures the green components of the image on a quincunx

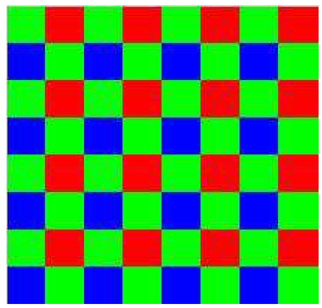

Figure 3: Bayer CFA pattern for an 8×8 image.

(checkerboard pattern) grid, while the red and blue components are measured on rectangular grids. The density of green pixels is twice that of either the red or blue pixels.

1.4.2 Image Sensors

Two technologies exist to manufacture imaging sensors: CCD (Charge Coupled Device) and CMOS (Complementary Metal Oxide Semiconductor). Most digital cameras use CCD sensors. The CCD is a collection of tiny light-sensitive diodes that convert photons (light) into electrons (electrical charge). These diodes are called photosites. As their name suggests, photosites are sensitive to light—the brighter the light that is incident on the photosite, the greater the electrical charge that will accumulate at that site. The amount of electrical charge that accumulated at each photosite (pixel) is read, and an ADC (analog-to-digital converter) turns each pixel's value into a digital value that is recorded. In the block diagram of the imaging system shown in Fig. 4, the 2D array of these digitized intensities is the output of the imaging array.
To summarize:

- A color image requires at least three color samples at each location. This would require three separate sensors at each location.
- For economic and size reasons, cameras are built with a single sensor in each location and a color filter array (CFA).
- At each pixel location, only one of the three color components is recorded.
- The 2D array of digitized intensities is the starting point for the demosaicking process.
- The DSP algorithm in the camera must estimate the two missing color values at each pixel; this is what will be implemented for the lab project.

[7]Note: The Bayer pattern is the exact CFA that should be referred to for the remainder of this lab.

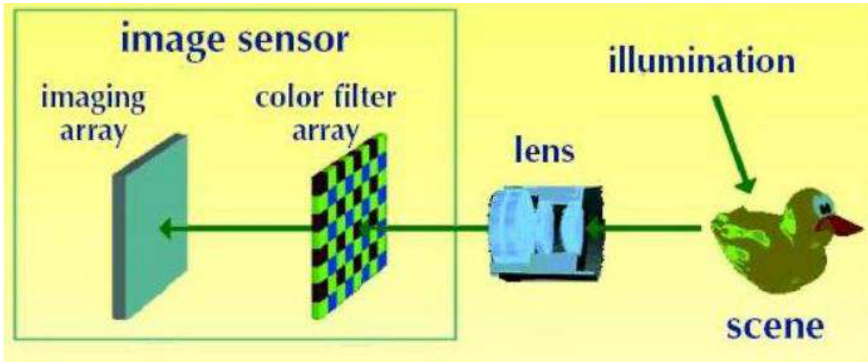

Figure 4: Digital still camera: image capture principle.

1.5 Demosaiking Process

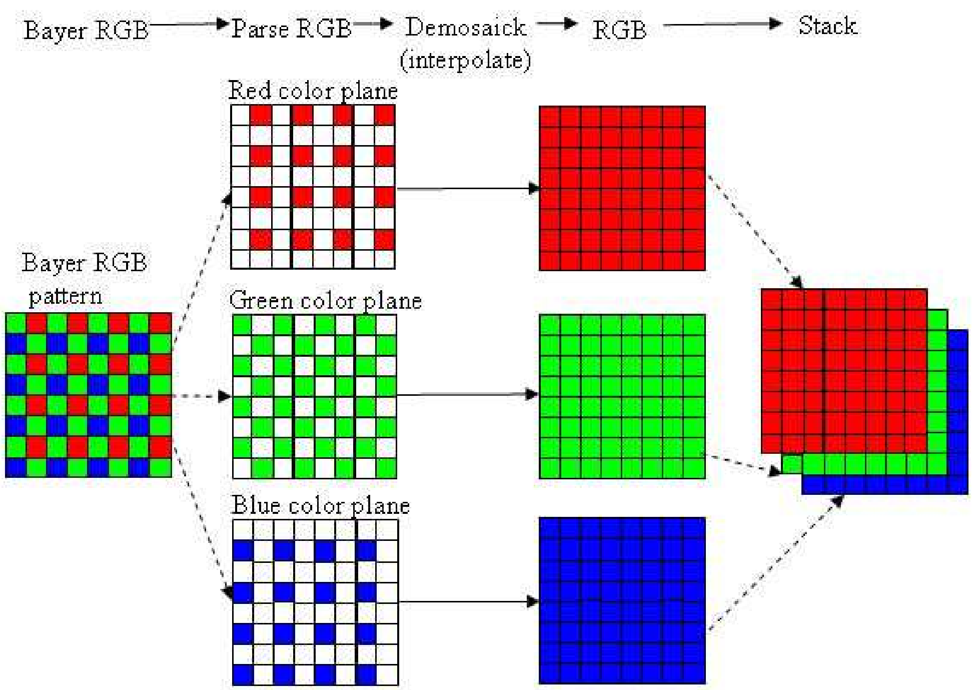

Figure 5: A demosaicking procedure that requires interpolation.

The demosaicking process is illustrated in Fig. 5; the steps in the procedure are as follows:

1. Read in the recorded Bayer CFA array.
2. Then parse the CFA array into three color planes, RGB.
3. Perform different interpolations (with FIR filters) on each of the color planes.
4. Stack the RGB color planes to form a 3D array that can be displayed via `show_img` or `imshow`.

Figure 6 shows the first step of taking the recorded Bayer CFA array (left image) and identifying the RGB color information which is shown in the right image.

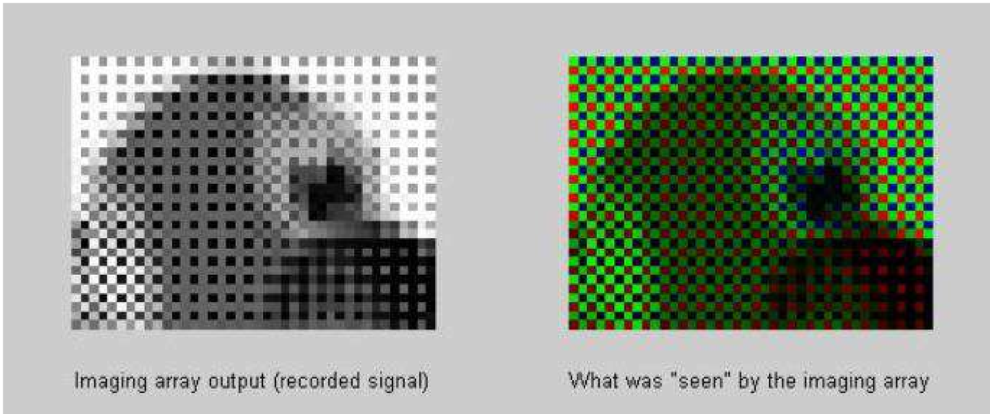

Figure 6: Images at two different stages in the image processing chain: (left) recorded CFA intensities, (right) color image after the RGB pixels have been separated.

2 Pre-lab

2.1 Sampling and Aliasing GUI

The first objective of this lab is to demonstrate usage of the `con2dis` GUI. If you have installed the *SP-First* Toolbox, you will already have this demo on the `matlabpath`.

In this MATLAB GUI, you can change the frequency of an input signal that is a sinusoid, and you can change the sampling frequency. Then the GUI will show the sampled signal, $x[n]$, its spectrum, and also the reconstructed output signal, $y(t)$ with its spectrum. Figure 7 shows the interface for the `con2dis` GUI. The top row shows plots in the time domain; the bottom row has the corresponding spectrum.

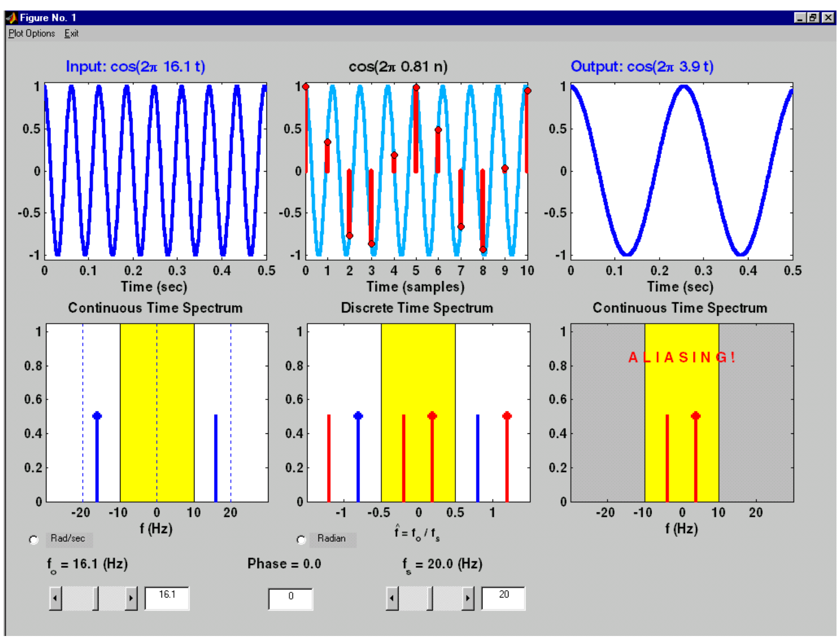

Figure 7: The `con2dis` MATLAB GUI interface. Input frequency is $\omega = 40\pi$ rad/s. With a sampling rate of $f_s = 24$ Hz, there is *aliasing*, so the output is not equal to the input. The spectrum of the discrete-time signal $x[n]$ has spectrum lines at $\hat{\omega} = \mp\pi/3$, which are aliases of $\hat{\omega} = \pm 40\pi/24 = \pm 5\pi/3$.

In the pre-Lab, you should perform the following steps with the `con2dis` GUI:

(a) Set the input to $x(t) = \cos(40\pi t + 0.5\pi)$, as in Fig. 7. Determine the Nyquist rate for sampling $x(t)$.

(b) Set the sampling rate to $f_s = 24$ samples/sec. Notice that this rate is too low to satisfy the Nyquist condition. Thus the output signal is not equal to the input, see Fig. 7.

(c) Determine the locations of the spectrum lines for the discrete-time signal, $x[n]$, found in the middle panels. Make sure that the `Radian` button is active so that the frequency axis for the discrete-time signal is $\hat{\omega}$.

(d) Determine the complex amplitudes for the spectral lines found in the yellow region of the bottom-middle panel. Notice that a $*$ on top of a spectral line indicates a line that was originally a negative frequency component in the input signal.

(e) Use the relationship $f_{\text{out}} = \hat{\omega} f_s$ to determine the formula for the output signal, $y(t)$. Apply the formula to the spectrum in the bottom-right panel in order to write the correct cosine formula for the time-domain signal shown in the top-right panel. What is the output frequency in Hz?

2.1.1 Relationships between the Frequency Domains: ω and $\hat{\omega}$

As you work through the following exercises, keep track of your observations by filling in the worksheet at the end of this assignment. Here are some issues to keep in mind:

- Is the question asking about a continuous-time signal $x(t)$, or a discrete-time signal, $x[n]$?
- Is the question about the time domain (top row of plots), or the frequency domain (bottom row)?
- The frequency axis (ω) for the spectrum of a continuous-time signal is different from the frequency axis $(\hat{\omega})$ of a discrete-time signal. The range of frequencies is also different: the important region of the $\hat{\omega}$-axis for the spectrum of a discrete-time signal always goes from $\hat{\omega} = -\pi$ to $\hat{\omega} = \pi$.
- The spectrum of a discrete-time sinusoidal signal will have many spectral lines separated by 2π.

Note: read Sections 4-1 and 4-2 in Chapter 4 for more information about the spectra of discrete-time signals, and for information about aliasing.

2.2 Aliasing the FM Signal

The Sampling Theorem tells us that the sampling rate f_s must be greater than twice the maximum frequency in the signal being sampled. An FM signal with a sinusoidal instantaneous frequency can be found as follows:

$$x(t) = A\cos(2\pi f_c t + \alpha\cos(2\pi\beta t + \gamma)) \tag{1}$$

where f_c centers the frequency plot, and α, β and γ control the frequency modulation.

Change the FM parameters to be $f_c = 3300$ Hz, $\alpha = 1200$ Hz, $\beta = 1.5$, $\gamma = -0.5\pi$ rads, and amplitude $A = 1$. Make the signal duration equal to 3.04 secs, starting at $t = 0$. Use $f_s = 8000$ Hz, make a spectrogram. In this case, there will be aliasing because the conditions of the Sampling Theorem are not obeyed: the maximum instantaneous frequency is 5100 Hz, and $f_s = 8000 < 2f_{\text{max}}$. Explain how the aliasing shows up in the spectrogram. In particular, observe the highest frequency shown in the spectrogram, and explain why it is equal to $\frac{1}{2}f_s$.

2.3 MATLAB Function to Display Images

CD-ROM
IMAGE DATA FILES

You can load the images needed for this lab from `*.mat` files, or from `*.png` files. Image files with the extension `*.png`, as well as other common formats like JPEG, can be read into MATLAB with the `imread` function. Any file with the extension `*.mat` is in MATLAB's binary format and must be loaded via the `load` command. After loading, use the command `whos` to determine the name of the variable that holds the image and its size.

CD-ROM
show_im.m

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.

2.4 Get Test Images

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

(a) Load and display the 428×642 "lighthouse" image[8] from `lighthouse.png`. The MATLAB command `ww = imread('lighthouse.png')` will put the sampled image into the array `ww`, Use `whos` to check the size and type of `ww` after loading. Notice that the array type for `ww` is `uint8`, so it would be necessary to convert `ww` to double precision floating-point with the MATLAB command `double` if calculations such as filtering are going to be done on `ww`. When you display the image it might be necessary to set the colormap via `colormap(gray(256))`.

(b) Use the colon operator to extract the 440^{th} row of the "lighthouse" image, and make a plot of that row as a 1-D discrete-time signal.

```
ww440 = ww(440,:);
```

Observe that the range of signal values is between 0 and 255. Which values represent white and which ones black? Can you identify the region where the 440^{th} row crosses the fence? Can you match up a black region between the image and the 1-D plot of the 440^{th} row?

2.5 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).[9] 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*.[10] One potential problem with down-sampling is that aliasing might occur because f_s is being changed—it's getting smaller by a factor of `p`. This can be illustrated in a dramatic fashion with the `lighthouse` image to be demonstrated later.

[8] The image size of 428×642 is the horizontal by vertical dimensions. When stored in a MATLAB matrix the `size` command will give the matrix dimensions, i.e., number of rows by number of columns, which is `[642 428]` for the lighthouse image.

[9] For this example, the sampling periods would be $T_1 = T_2 = 1/300$ inches.

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

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

```
wp = ww(1:p:end,1:p:end);
```

when we are downsampling by a factor of `p`.

2.6 Printing Multiple Images on One Page

The phrase "what you see is what you get" can be elusive when displaying and printing 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 by the program that is handling the images. One way to think about this in signal processing terms is to think of the screen as one kind of D-to-A and the printer as another kind; each one uses a different D-to-A reconstruction method to get the continuous-domain (analog) output image that you see.

Another problem occurs when you try to put two images of different sizes into subplots of the same MATLAB figure. It doesn't work because MATLAB wants to force them to be the same size. Therefore, you should display these different size images in separate MATLAB figure windows. In order to get a printout with multiple images on one 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 a Windows program such as `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`.[11]
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.

3 In-Lab Exercise

For the lab exercise, you will synthesize some signals, and then study their frequency content by using the spectrogram. The objective is to learn more about the connection between the *time-domain* definition of the signal and its *frequency-domain* content.

For the instructor verification, you will have to demonstrate that you understand concepts in a given subsection by answering questions from your lab instructor (or TA).

The lab verification requires that you write down your observations on the verification sheet when using the GUI. These written observations will be graded.

[11] An alternative is to use the free program called IRFANVIEW, which can do image editing and also has screen capture capability. It can be obtained from `www.irfanview.com`. Other alternatives are Photoshop, or "The GIMP" at `www.gimp.org/windows`.

3.1 Sampling and Aliasing

Use the `con2dis` GUI to do the following exercises. The parameters of the input signal are its frequency f_0 in Hz, and its phase φ in rads. The amplitude is one. The sampling rate for both the A/D converter and the D/A converter is f_s in samples/sec.

> ***In all cases, write a brief explanation of your answer. "Trial and error" is a poor justification, so try to write something better than that.***

(a) Set the input frequency to $f_0 = 15\,\text{Hz}$. Determine the *Nyquist Rate,* i.e., the lower bound for the sampling rate f_s so that no aliasing occurs. The units of f_s are samples per second. Justify your answer.

(b) Set the input frequency to $f_0 = 15\,\text{Hz}$ and the input phase to $\varphi = +\pi/4$. Determine the locations of the spectral lines in the spectrum of the discrete-time signal when the sampling rate is $f_s = 20\,\text{Hz}$. Do this for both spectral lines in the interval $[-\pi, \pi]$. Explain how you calculated the two values for $\hat{\omega}$.

(c) For the same parameters as the previous part, determine the complex amplitude for the spectral line that lies between 0 and π. Give the complex amplitude in polar form.

(d) Set the input frequency to $f_0 = 12\,\text{Hz}$ and the input phase to $\varphi = +\pi/4$. You can see many spectral lines in the spectrum of the discrete-time signal when the sampling rate is $f_s = 20\,\text{Hz}$. Find the locations of the two spectral lines in the interval $[-\pi, \pi]$. Explain how you calculated these two values for $\hat{\omega}$.

(e) For the same parameters as the previous part, determine the complex amplitude for the spectral line that lies between 0 and π. Give that complex amplitude in polar form.

(f) Set the sampling rate to $f_s = 20\,\text{Hz}$, and assume that the output signal has a frequency of $2\,\text{Hz}$, and a phase of $-3\pi/4$. Determine ***three different values of the input frequency*** that will give this output signal. In addition, determine the corresponding value of the input phase φ for each frequency. EXPLAIN.

(g) Set the input frequency to $f_0 = 15\,\text{Hz}$ and the input phase is $\varphi = +\pi/4$. ***Determine the sampling rate*** f_s so that the output signal has a frequency of $4\,\text{Hz}$, and a phase of $-\pi/4$.

Completion of Lab Results (on separate Report page)

3.2 Aliasing a Sinusoidal-FM Signal

Frequency modulated signals make good test cases for showing aliasing. Sketch the results of the following on the Report sheet, and also provide some explain that compares with the theory. An FM signal with a sinusoidal instantaneous frequency can be found as follows:

$$x(t) = A\cos(2\pi f_c t + \alpha\cos(2\pi\beta t + \gamma)) \tag{2}$$

where f_c centers the frequency plot, and α, β and γ control the frequency modulation.

(a) Create a sinusoidal-FM chirp using Eq. (2) with the parameters chosen to be $A = 2$, $f_c = 800\,\text{Hz}$, $\alpha = 1000\,\text{Hz}$, $\beta = 1.5$ and $\gamma = 0$. Set the sampling rate to $f_s = 4000\,\text{Hz}$, and the signal duration to be $2\,\text{s}$ starting at $t = 0\,\text{s}$. Make a spectrogram that shows both positive and negative frequencies, i.e.,

use `plotspec` and add a tiny imaginary value to the real FM signal. Use a relatively short window length.
Explain where the plot exhibits the correct value of the instantaneous frequency known from Eq. (2), and also *use aliasing to explain the positive frequency values* observed at $t = 0$, 0.5, 1.0, and 1.5 s..

Completion of Lab Results (on separate Report page)

Comment: It is also possible to "simulate" this behavior of the FM signal with the `con2dis` GUI by continually moving the frequency of the input signal up past $\frac{1}{2}f_s$, and simultaneously watching the frequency of the output signal. For example, set $f_s = 22\,\text{Hz}$ and try this in `con2dis`; you should see the frequency of the output signal go up and then down (when the input frequency passes $\frac{1}{2}f_s$).

3.3 Synthesizing a Test Image

In order to probe your understanding of the relationship between MATLAB matrices and image display, you can generate a synthetic image from a mathematical formula. Then you can use the theory of sampling and aliasing to explain how downsampling the cosine formula will provide surprising results.

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

```
xpix = ones(256,1)*cos(2*pi*(0:255)/32);
```

Display the image and explain the gray-scale pattern that you see. Count the number of black stripes across the image. Explain how you can predict that number from the period of the formula for `xpix`?

(b) In the previous part, which data value in `xpix` is represented by white? which one by black? Keep in mind that the cosine has values between ± 1.

Completion of Lab Results (on separate Report page)

(c) *Optional:* Explain how you would produce an image with bands that are horizontal. Give the formula that would create a 400×400 image with five horizontal black bands separated by white bands. Write the MATLAB code to make this image and display it.

3.4 Aliasing in a Test Image

The banding structure in the test images is controlled by the frequency of the cosine. In other words, we can rewrite the formula for the test image (above) as

```
wd = 2*pi*1/32; xpix = ones(256,1)*cos(wd*(0:255));
```

(a) Generate two test images with different frequencies, one with `wd = 2*pi*4/32` and the other with `wd = 2*pi*12/32`. Call these images `xpix4` and `xpix12`. Display the images, and explain why the image made from the higher frequency cosine has a shorter horizontal period.

(b) Now we apply down-sampling by two, i.e., `xpix4(1:2:end,1:2:end)` and `xpix12(1:2:end,1:2:end)`, to both images from the previous part. Use `subplot(2,2,n)` to make a four-panel display: put `xpix4` and `xpix12` in the top row, and put the two down-sampled images in the bottom row of the 2×2 subplot. Explain why the two down-sampled images look the same.

Completion of Lab Results (on separate Report page)

3.5 Observing Aliasing on an Image of Your Choice

Suppose that you wanted to illustrate the idea that sampling can cause aliasing. Images provide one way to show aliasing as a visual phenomenon.

Perform this operation on the `lighthouse.png` image.

(a) Read in the `lighthouse.png` file with the MATLAB function `imread`. When you check the size of the image, you'll find that it is not square. Now down-sample the `lighthouse` image by a factor of 2, and display the down-sampled image. What is the size of the down-sampled image?

(b) Notice the aliasing in the down-sampled image, which is surprising since no new values are being created by the down-sampling process. Identify specific parts of the image where you can see the effects of aliasing effects most dramatically. Describe how the aliasing appears visually. [12] Explain why the aliasing is happening by thinking about high frequencies in the image, i.e., look for features in the images that are *quasi-periodic* and can be described as having a frequency.

Next provide one (grayscale) images of your choice. You can get a high-resolution image from the web. No matter how you create the image, make sure that you work with a high-resolution image that is ***unique—no other student you know of should be using the same image.*** Furthermore, obtain an image that (more or less) fills your screen, e.g., with dimensions no smaller than 1024×768. Crop the image if necessary.

Apply down-sampling by a factor of three to your image. Then compare the original image to the down-sampled version and ***point out all regions where aliasing has occurred.*** In your lab report write a justification that uses theory to explain why some regions of the image aliased, and others did not.

In order to have a good example, you will have to choose the original image carefully. The original should have no aliasing, but must have features that will alias when the down-sample by three is applied. More than likely you will have to do a bit of trial and error with more than one image.

Aliasing can be illustrated with two images (before and after) along with a carefully written explanation that points out all regions where you see aliasing, a description of visual phenomenon that you are calling aliasing, and an explanation that connects the cause of the aliasing to the frequency content of the image in that region.

Completion of Lab Results (on separate Report page)

[12] 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.

ECE-2026 **Lab #4** **Spring-2025**

LAB COMPLETION REPORT

Name: ______________________________ gtLoginUserName: _______________ Date: _______

Part	Observe **and Justify**
3.1(a)	The Nyquist rate for sampling with no aliasing is $f_s =$ Why?
3.1(b)	Within the $[-\pi, \pi]$ interval, $\hat{\omega} = \pm$
3.1(c)	Complex amp at positive $\hat{\omega}$.
3.1(d)	Within the $[-\pi, \pi]$ interval, $\hat{\omega} = \pm$
3.1(e)	Complex amp at positive $\hat{\omega}$.
3.1(f)	Three input frequencies and phases.
3.1(g)	Determine f_s to get a specific output frequency and phase.
3.2	Sketch spectrogram below. Explain.

Part 3.3(a,b) Explain how to control the number of bands. Explain grayscale, i.e., which values correspond to black, white, or gray.

Part 3.4(b) Generate two test images with bands. Explain how downsampling two different images can give the same result.

Part 3.5 Describe the aliasing effects you have observed in the image of your choice.