Lab 2: Sampling, Aliasing, and Reconstruction

1 Overview
This laboratory covers the topics of sampling, aliasing, and reconstruction. Sampling is a critical step in nearly all signal processing applications. Moreover, sampling must be properly applied to avoid aliasing and allow appropriate reconstruction of the continuous time signal. Sampling is investigated here by considering first a rotating disk illuminated by a strobe light. After applying sampling to this mechanical system, we consider one and two-dimensional signals. In the one-dimensional case, we employ chirp signals that have time varying frequency. Such signals can be visualized in the time and frequency domains, as well as listened to by outputting them through speakers. Lastly, we utilize images as two-dimensional signals to study the effects of sampling as well as reconstruction. In all cases, the results will be investigated and compared to the results predicted by the Sampling Theorem.

2 Procedures

2.1 Strobe Sampling of a Rotating Disk
The effects of sampling and aliasing can be demonstrated through the use of a rotating disk and strobe light. Strobe lights, in addition to entertainment, can be utilized to determine the frequency of mechanical movements. In this experiment, a disk is affixed onto a motor that rotates at a constant speed. The disk is marked with an arrow representing a phaser. Thus, the system represents a rotating phaser. A strobe light can be used to illuminate the rotating phaser at a fixed frequency. Using this setup, complete the following and include the results in your report:

a) Vary the frequency of the strobe light to determine the frequency of rotation of the phaser and motor. What should the apparent motion of the phaser be when the strobe light and phaser are at the same frequency? Determine what other strobe frequencies give the same result. What is the relation between the frequencies?

b) After determining the frequency of the phaser, increase the frequency of the strobe light so that it is slightly greater than the frequency of the phaser. Observe the apparent motion of the phaser. Decrease the frequency of the strobe light so that is slightly less than the frequency of the phaser. Observe the apparent motion again. Can you explain this behavior? Is aliasing observed? Are aliasing and folding observed?

c) Derive an expression for the complex phaser $p[n]$ that gives the position of the phaser at the $n^{th}$ flash, assuming that the phaser is initially pointing straight up at $n=0$. Draw a two sided spectrum for the cases: (1) strobe frequency equals phaser frequency, (2) strobe frequency is slightly greater than phaser frequency, and (3) strobe frequency is slightly less than phaser frequency. Explain the observed apparent rotation (direction and speed) of the phaser based on the spectrums.

2.2 Chirp Signals and Aliasing
Chirp signals, by definition, have time varying frequency. The time varying frequency can, for instance, be used to transmit information, which is the approach adopted in Frequency Modulation (FM) transmission. Here, we will use the frequency variations in such signals to further investigate sampling and aliasing.
2.2.1 Frequency Modulated Signals

We will look at signals in which the frequency varies as a function of time. In the constant-
frequency sinusoid

\[ x(t) = A \cos(2\pi f_0 t + \phi) = \Re\{Ae^{j\phi}e^{j2\pi f_0 t}\} \]  

(1)

the argument of the cosine is also the exponent of the complex exponential, so the angle of this
signal is the exponent \((2\pi f_0 t + \phi)\). This angle function changes \textit{linearly} versus time, and its
time derivative is \(2\pi f_0\), which equals the constant frequency of the cosine in rad/sec.

A generalization is available if we adopt the following notation for the class of signals
represented by a cosine function with a time-varying angle:

\[ x(t) = A \cos(\psi(t)) = \Re\{Ae^{j\psi(t)}\} \]  

(2)

The time derivative of the angle from (2) gives a frequency in rad/sec

\[ \omega_i(t) = \frac{d}{dt} \psi(t) \quad \text{(rad/sec)} \]

but we prefer units of hertz, so we divide by \(2\pi\) to define the \textit{instantaneous frequency}:

\[ f_i(t) = \frac{1}{2\pi} \frac{d}{dt} \psi(t) \quad \text{(Hz)} \]  

(3)

2.2.2 Chirp, or Linearly Swept Frequency

A \textit{chirp} signal is a sinusoid whose frequency changes linearly from a starting value to an ending
one. The formula for such a signal can be defined by creating a complex exponential signal with
quadratic angle by defining \(\psi(t)\) in (2) as

\[ \psi(t) = 2\pi \mu t^2 + 2\pi f_0 t + \phi \]

The (scaled) derivative of \(\psi(t)\) yields an instantaneous frequency, equation (3), that changes
\textit{linearly} versus time:

\[ f_i(t) = 2\mu t + f_0 \]

The slope of \(f_i(t)\) is equal to \(2\mu\) and its intercept is equal to \(f_0\). If the signal starts at time
\(t = 0\) secs., then \(f_0\) is also the starting frequency. The frequency variation produced by such a
time-varying angle is called \textit{frequency modulation}. This kind of signal is an example of a
frequency modulated (FM) signal. More generally, we often consider them to be part of a larger
class called \textit{angle modulation} signals. 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.”

The following MATLAB code will synthesize a chirp:

```matlab
fsamp = 11025;
dt = 1/fsamp;
dur = 1.8;
.tt = 0 : dt : dur;
psi = 2*pi*(0.25 + 200*tt + 500*tt.*tt);
```
xx = real( 7.7*exp(j*psi) );
soundsc( xx, fsamp );

(a) Determine the total duration of the synthesized signal in seconds, and also the length of the \( tt \) vector (number of samples).

(b) In MATLAB, signals can only be synthesized by evaluating the signal’s defining formula at discrete instants of time. These are called samples of the signal. For the chirp we do the following:

\[
x(t_n) = A \cos(2\pi\mu t_n^2 + 2\pi f_0 t_n + \phi)
\]

where \( t_n = nT_s \) represents discrete time instants. In the MATLAB code above, what is the value for \( t_n \)? What are the values of \( A \), \( \mu \), \( f_0 \), and \( \phi \)?

(c) Determine the range of frequencies (in hertz) that will be synthesized by the MATLAB script above. Make a sketch/plot by hand or using Matlab of the instantaneous frequency versus time. What are the minimum and maximum frequencies that will be heard?

(d) Listen to the signal to determine whether the signal’s frequency content is increasing or decreasing (use \( \text{soundsc}() \)). Notice that \( \text{soundsc}() \) needs to know the sampling rate at which the signal samples were created. For more information do \text{help sound} and \text{help soundsc}.

Use the code above as a starting point in order to write a MATLAB function that will synthesize a “chirp” signal according to the following comments:

```matlab
function [xx,tt] = mychirp( f1, f2, dur, fsamp )
%MYCHIRP generate a linear-FM chirp signal
%
% usage: xx = mychirp( f1, f2, dur, fsamp )
%
% f1 = starting frequency
% f2 = ending frequency
% dur = total time duration
% fsamp = sampling frequency (OPTIONAL: default is 11025)
%
% xx = (vector of) samples of the chirp signal
% tt = vector of time instants for t=0 to t=dur
%
if( nargin < 4 ) %-- Allow optional input argument
    fsamp = 11025;
end
```

As a test case, generate a chirp sound whose frequency starts at 2500 Hz and ends at 500 Hz; its duration should be 1.5 sec. Listen to the chirp using the \text{soundsc} function. Include in your report a listing of the \text{mychirp.m} function that you wrote.

**2.2.3 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 as in (1)
the spectrum consists of two components, one at $2\pi f_0$, the other at $-2\pi f_0$. For more complicated signals, the spectrum 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.

When unsure about a command, use `help`.

There are a few important things to know about spectrograms:

1. In MATLAB the function `specgram` will compute the spectrogram. Type `help specgram` to learn more about this function and its arguments.

2. Spectrograms are numerically calculated and only provide an estimate of the time-varying frequency content of a signal. There are theoretical limits on how well spectrograms can actually represent the frequency content of a signal.

3. A common call to the function is `specgram(xx,1024,fsamp)`. The second argument is the window length which could be varied to get different looking spectrograms. The spectrogram is able to “see” the separate spectrum lines with a longer window length, e.g., 1024 or 2048.

### 2.2.4 Chirps and Aliasing

Use your MATLAB function `mychirp` to synthesize a “chirp” signal with the following parameters:

a) A total time duration of 2.5 secs, where the desired instantaneous frequency starts at 13,000 Hz and ends at 200 Hz.

b) Use a sampling rate of $f_{\text{samp}} = 8000$ Hz. Listen to the signal. What comments can you make regarding the sound of the chirp (e.g., is it linear)? Does it chirp down, or chirp up, or both? Create a spectrogram of your chirp signal. Experiment with the spectrogram window length to determine an appropriate window length value. Use the sampling theorem to help explain what you hear and see. In addition, make some theoretical calculations by hand: Determine the range of frequencies (in hertz) that will be synthesized by this MATLAB script. Make a sketch/plot by hand or with Matlab of the instantaneous frequency versus time. Explain how aliasing affects the instantaneous frequency that is actually heard. Listen to the signal again to verify that it has the expected frequency content.

### 2.3 Digital Image: Sampling, Aliasing and Reconstruction

Digital images are also signals that can be used to investigate 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 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.
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. 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 grayscale images. Here, 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

\[
x[m, n] = x(mT_1, nT_2) \quad 1 \leq m \leq M, \text{ and } 1 \leq n \leq N
\]

where \( T_1 \) and \( T_2 \) give the sample spacing in the horizontal and vertical directions. Typical values of \( M \) and \( N \) are 256 or 512; e.g., a \( 512 \times 512 \) image that has nearly the same resolution as a standard TV image. 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 finite in magnitude; i.e.,

\[
0 \leq x[m, n] \leq X_{\text{max}} < \infty
\]

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 a maximum value \( X_{\text{max}} \). Usually an eight-bit integer representation is used. With 8-bit integers, the maximum value (in the computer) is \( X_{\text{max}} = 2^8 - 1 = 255 \), and there are \( 2^8 = 256 \) different gray levels for the display, from 0 to 255. **NOTE:** The \([0, X_{\text{max}}]\) range is often scaled to \([0,1]\) by dividing all values by \( X_{\text{max}} \).

### 2.3.1 Displaying Images

As you will discover, the correct display of an image in a gray-scale format can be tricky, especially after some processing has been performed on the image. Fortunately, the function `imshow` in the *Image Processing Toolbox* handles most of these problems. Still, it is helpful if the following points are noted:

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 \( 0 \leq x[m, n] \leq 255 = 2^8 - 1 \).
3. The actual display on the monitor is created with the `imshow` function. The 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 display” where all three primary colors (red, green and blue, or RGB) are equally used, creating what is called a “gray map.” In MATLAB the `gray` color map is set up via

\[
\text{colormap(gray(256))}
\]

\(^2\) For example, an RGB color system needs three values at each spatial location: one for red, one for green and one for blue.
which gives a $256 \times 3$ 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, so they will not be used.

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 `imshow` to display images, which automatically rescales the image to the appropriate range.

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

a) Load and display the $326 \times 426$ “lighthouse” image from `lighthouse.mat`. The command `load lighthouse` will put the sampled image into the array `xx`. Use `whos` to check the size of `xx` after loading. 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 200th row of the “lighthouse” image, and make a plot of that row as a 1-D discrete-time signal.

\[
xx200 = xx(200,:) ;
\]

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 200th row crosses the fence?

### 2.3.2 Down-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)$^3$. 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$^4$.

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

\[
xxp = xx(1:p:end,1:p:end);
\]

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

a) One potential problem with down-sampling is that aliasing might occur. This can be illustrated in a dramatic fashion with the `lighthouse` image. Down-sample the `lighthouse` image by a factor of 2. What is the size of the down-sampled image?

b) Describe how the aliasing appears visually. Compare the original to the downsampled image. Which parts of the image show the aliasing effects most dramatically?

---

$^3$ For this example, the sampling periods would be $T_x = T_y = 1/300$ inches.

$^4$ The Sampling Theorem applies to digital images, so there is a Nyquist Rate that depends on the maximum spatial frequency in the image.
c) This part is challenging: explain why the aliasing happens in the lighthouse image by using a “frequency domain” explanation. In other words, estimate the frequency of the features that are being aliased. Give this frequency as a number in cycles per pixel. (Note that the fence provides a sort of “spatial chirp” where the spatial frequency increases from left to right.) 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.

### 2.3.3 Reconstruction of Images

When an image has been sampled, we can fill in the missing samples by doing interpolation. For images, this is analogous to the examples shown in Chapter 4 for sine-wave interpolation that 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.

![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.](image)

For these reconstruction experiments, use the lighthouse image, down-sampled by a factor of 3. You will have to generate this by loading in the image from lighthouse.mat to get the image which is in the array called **xx**. A down-sampled lighthouse image should be created and stored in the variable **xx3**. The objective will be to reconstruct an approximation to the original lighthouse image, which is 256 × 256, from the smaller 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 **xr1**, and the result is the row vector **xr1hold**.

\[
xr1 = (-2).^(0:6);
L = length(xr1);
nn = ceil((0.999:1:4*L)/4); %<-- Round up to the %integer part
xr1hold = xr1(nn);
\]

Plot the vector **xr1hold** to verify that it is a zero-order hold version derived from **xr1**. Explain what values are contained in the indexing vector **nn**. If **xr1hold** is treated as an interpolated version of **xr1**, then what is the interpolation factor? Your lab report should include an explanation for this part, but plots are optional—use them if they simplify the explanation.

#### b) Now return to the down-sampled lighthouse image, and process all the rows of **xx3** to fill in the missing points. Use the zero-order hold idea from part (a), but do it for an interpolation factor of 3. Call the result **xholdrows**. Display **xholdrows** as an image, and compare it to the downsampled image **xx3**; compare the size of the images as well as their content.
c) Now process all the columns of xhold\_rows to fill in the missing points in each column and call the result xhold. Compare the result (xhold) to the original image lighthouse. Include your code for parts (b) and (c) in the lab report.

d) **Linear interpolation** can be done in MATLAB using the interp1 function (that’s “interp-one”). When unsure about a command, use help. 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:

```matlab
n1 = 0:6;  
xr1 = (-2).^n1;  
tti = 0:0.1:6;  %-- locations between the n1 indices  
xr1linear = interp1(n1,xr1,tti);  %-- function is 'linear'  
stem(tti,xr1linear)
```

For the example above, what is the interpolation factor when converting xr1 to xr1linear?

e) In the case of the lighthouse image, you need to carry out a linear interpolation operation on both the rows and columns of the down-sampled image xx3. 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 xxlinear. Include your code for this part in the lab report.

f) Compare xxlinear to the original image lighthouse. Comment on the visual appearance of the “reconstructed” image versus the original; point out differences and similarities. Can the reconstruction (i.e., zooming) process remove the aliasing effects from the down-sampled lighthouse image?

g) 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. A couple of questions to think about: Are edges low frequency or high frequency features? Are the fence posts low frequency or high frequency features? Is the background a low frequency or high frequency feature?

*Comment:* You might use MATLAB’s zooming feature to show details in small patches of the output image. However, be careful because zooming does its own interpolation, probably a zero-order hold.

### 3 Report Components

Your report should be neatly prepared and have a cover sheet, short overview, discussion section, and conclusions. The body of the report should answer all questions asked and include necessary plots and figures as well as MATLAB code. Please make sure all plots, figures, and code are appropriately referenced in the body of the report.

---

5 Use a matrix transpose in between the interpolation calls. The transpose will turn rows into columns.