ELEC 324
Lab 1 – Sampling and Discrete Signals*

Fall 2022

1 Preface

The purpose of these ELEC 324 labs is to introduce you to Matlab for working with discrete-time signals. Basic issues associated with using computers to represent and process discrete-time signals are examined within the Matlab software environment.

Some knowledge of Matlab is assumed to have been acquired from the ELEC 323 labs. The built-in help system is a useful resource that can be accessed by typing `help <command name>` at the command prompt, or via the help desk available in the help menu or by typing F1. **Students are expected to use built-in help to glean additional information that may be needed to complete a lab. The student is encouraged to explore the rich Matlab environment and enjoy unexpected discoveries. To facilitate this goal, information may be deliberately omitted from the lab instructions. In addition, the student is encouraged to explore further (and perhaps finish the lab) beyond the scheduled lab hours.**

The ELEC 324 labs have been tested using recent Matlab release 2022a. It is possible, however, that Matlab code is not forward compatible with newer versions.

2 Introduction

Sampling is a very important subject in signal processing. Everywhere around you, there are digital signals which were either obtained by sampling analog signals, or synthesized as a sequence of samples. Each sample has a value which may be numerical or symbolic. Our focus is on numerical signals, for which a sample value may be a scalar or a vector.

Analog signals, mathematically represented as continuous-time (CT) expressions, permeate the physical world. A discrete-time (DT) signal can be obtained by sampling a CT signal at a discrete set of time points or sampling instances. The value of the CT signal at each sampling instant gives the corresponding value of the DT sample at that

instant. In this way, an analog signal can be represented as a sequence of samples. A sequence is a mathematical object used to represent a DT signal, or discrete signal for short. We usually prefer the DT signal to *faithfully* represent the analog signal. The sampling theorem gives a precise condition under which a DT signal can represent a CT signal with no loss of information.

Mathematically, we often treat signals as defined over an infinite-extent time domain. That is, we write $x(n)$ and say that it is defined for all possible integer values of n. In practice, physical signals have a finite time duration, so that we are really only interested in a finite number of sampling points. Nevertheless, to facilitate mathematical modeling and analysis, we maintain an infinite-time extent representation of signals. We need to keep this in mind in order to bridge theory with practice. For instance, the samples of a signal $x(n)$ could be stored in a Matlab vector variable v. The samples can be read from a file into v, using the `audioread` command. The number of samples in v is given by `length(v)`. v contains the values of only `length(v)` samples of $x(n)$. Very often the values of n for which the values of $x(n)$ are provided by v are implicit, e.g. $n \in \{0, 1, \ldots, length(v) - 1\}$. The rest of the sample values of $x(n)$ usually default to zero.

Using Matlab you will be able to visualize and analyze various aspects and consequences of the sampling theorem. In Section 3, we explore time interpolation as it relates to discrete signals and their representation in Matlab. In Section 4, you will be able to *see* how aliasing manifests itself in the time and frequency domains. In Section 6, you will be able to *hear* and *see* how it manifests itself in a sound file and in an image file, respectively. In Section 4.2 you will look in more detail at filtering, as applied to signal reconstruction.

2.1 Preparation

The first four lectures and relevant sections of the text references (see course topic outline) provide background material for the sampling theorem.

Questions below marked "Pre-lab" need to be answered *before* the lab. Bring your pre-lab answers to the lab with you.

You will need to bring a pair of headphones for listening to audio signals. Computer audio output jacks take only miniplugs.

2.2 Matlab Commands Used

Here are some of the Matlab commands you will be using for the first time in this lab. It is important to take time to familiarize yourself with them *before* the lab.

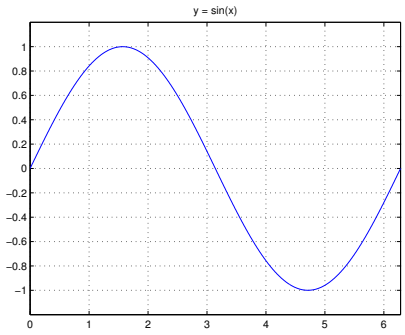
```
butter
decimate
downsample
filter
freqs
freqz
```

3 Time Interpolation

In this section we will explore one facet of how Matlab represents signals. In the classroom, you study both discrete-time (DT) and continuous-time (CT) signals. A digital computer is naturally suited to representing DT signals. It is also feasible to represent CT signals as computational algorithms that implement some mathematical equations. Many physical signals do not immediately lend themselves to such representation; these signals are represented on a digital computer as *samples*, i.e., values of the signal at discrete locations of the signal domain. We study sampled representations of signals below, beginning with signals that have closed form representations.

First, lets see how well DT signals can approximate CT signals. Consider the sine wave resulting from the following commands:

```
clf;
x = 0:2*pi/1000:2*pi;
y = sin(x);
plot(x,y);
```



While this signal appears to be a continuous-time signal on the graph, more careful scrutiny would suggest otherwise.

Questions:

3.1 Interpolation

Run the following code and observe the resulting graph:

```
% interpolate.m
clf;
steps = 10;
x = 0:2*pi/steps:2*pi;
y = sin(x);
plot(x,y);
grid;
axis([0 2*pi -1.2 1.2]);
```

Why are you not seeing a nice sine wave? Why are the points connected like they are?

Adding an `'o-'` argument to the `plot` command may give you some insight. What mathematical operation describes the manner by which Matlab generates the points in the graph between the 'o' points?

Change `steps` to 60 and re-run the code. How does the graph change?

Answer:

The triangular-wave looking reconstruction plotted for *steps*=3 below illustrates severe distortion of a sine wave by spectral images. The images act as the harmonic components in a Fourier series representation of a triangular wave.

3.2 Stem vs Plot

Re-run the code from Question 3.1 with `steps` set to 60, but change the `plot` command to `stem`. Is more or less information needed to generate the connected graph produced by `plot`, in comparison with the graph produced by `stem`?

Answer:

3.3 Varying the Time Resolution

Modify the above program `interpolate.m` to plot successively more refined representations of $y = \sin(x)$. Vary `steps` from 3 to 20, using the `pause` command inside a `for` loop to update the graph every half-second.

Answer:

4 Sampling and Reconstruction of a Sine Wave

4.1 Time Domain Analysis

In this section we will illustrate how a discrete-time signal is obtained by sampling a continuous-time signal, and how the original continuous-time signal can then (perhaps) be reconstructed from the discrete-time signal. In particular we will illustrate the effect of the sampling frequency and investigate what actually happens to the signal if it is not sampled at an appropriate frequency.

In order to understand the code below, it is important to realize that there is no simple way to represent an arbitrary CT signal on a digital computer. A CT signal may be represented by a computable formula or an algorithm, but it would be hard (and inconvenient) to find a formula or algorithm that would represent or regenerate any arbitrary CT signal with great accuracy. So what is the simplest way to represent an

arbitrary CT signal digitally? The answer is to sample it at sufficiently high frequency. But wait, within the Matlab environment, we are going to illustrate sampling a CT signal by sampling a DT signal! In the code below, for example, we have a "continuous" sine wave in the `x_t` array:

```
x_t = cos(2*pi*f*t)
```

We propose that this signal is roughly equivalent to $x(t) = \cos(2\pi f t)\,,\, t \epsilon \mathbb{R}$. The high resolution of the t variable is what makes the approximation valid. Mathematically, the condition is $\Delta t \ll \frac{1}{2\pi f}$ (sec/rad). In the code below we have $\Delta t = 0.0005$ and $\frac{1}{2\pi f} = 0.032$ (nearly two orders of magnitude greater).

While this may seem trivial, it's crucial to understand what's really going on; the validity of everything that follows depends on this. Now run the following code and look at the output:

```
% sampling1.m
clf;

% First plot
t = 0:0.0005:1;
f = 5;
x_t = cos(2*pi*f*t);
subplot (4,1,1);
plot (t,x_t);
grid;
xlabel ('Time [sec]');
ylabel ('Amplitude');
title ('Continuous-time signal x(t)');
axis([0 1 -1.2 1.2])

% Second plot
subplot(4,1,2);
T = 0.075;
n = 0:T:1;
x_n = cos(2*pi*f*n);
k = 0:length(n)-1;
stem (n,x_n);
grid;
xlabel ('Time [sec]');
ylabel ('Amplitude');
title ('Sampled x(t)');
axis ([0 1 -1.2 1.2]);

% Third plot
```

```
subplot(4,1,3);
k = 0:length(n)-1;
stem(k,x_n);
grid;
axis ([0 (length(n)-1) -1.2 1.2]);
xlabel ('Time index n');
ylabel ('Amplitude');
title ('Discrete-time signal x[n]');

% Fourth plot
subplot(4,1,4);
y = zeros(1,length(t));
for i = 1:length(n)
    y = y + x_n(i)*sinc(t/T - i + 1);
end
plot(t,y);
grid;
xlabel ('Time, sec');
ylabel ('Amplitude');
title ('Reconstructed continuous-time signal y(t)');
axis ([0 1 -1.2 1.2]);
```

Questions:

4.1.1 Pre-lab: Sampling

In the code above, we sample `x_t` to obtain `x_n`. Which variable controls the sampling frequency? What is the bound on its value if we don't want to have aliasing? What happens if the sampling frequency equals this bound? If it is above? If it is under?

Answer:

4.1.2 Sampling

Modify the `sampling1.m` program above to show three situations graphically: when the sampling frequency is below the bound given by the sampling theorem, when it is equal to the bound, and when it is above. That is, plot four signals: the original $x(t)$, and the three reconstructed CT signal in the above specified order.

Answer:

4.1.3 Aliased frequency

Determine the frequency of the reconstructed sinusoid in the second plot. Can you relate this frequency to the frequency of $x(t)$ and the sampling frequency you used, by applying the spectral image generation process embodied by the sampling theorem?

Answer:

4.1.4 Signal Reconstruction

Explain in words how the above code constructs the `y` signal.

Answer:

4.1.5 Reconstruction Quality

Examine the 3rd and 4th plots. Do the reconstructed signals match $x(t)$? Explain the discrepancies you observe, if any.

Answer:

4.2 Realistic Filters for Reconstruction

To reconstruct a signal from its sampled version, we use the samples to amplitude-modulate an impulse train and pass it through a low-pass filter (LPF). If this LPF has an ideal frequency response, and the passband edge is set to half the sampling frequency, the signal will be perfectly reconstructed – the filter output signal will be identical to the CT signal sampled. This is in fact what we closely approximated in Section 4.1, where we truncated the ideal LPF's impulse response.

A major problem is that ideal filters have noncausal impulse responses. The impulse response of an ideal LPF is the *sinc* function (refer to the lecture notes for more details on this). Ideal reconstruction, attained by convolution of the sample-weighted impulse train with the ideal filter impulse response, requires knowledge of present, past and future sample values. This is not at all possible in real-time signal processing (an ELEC 421 and 422 subject), although it can be closely approximated in non-real-time processing, where knowledge of *all* samples of the signal, not just present and past values, is available. Noncausal reconstruction is actually what we implemented in Section 4.1. There, the entire DT signal is already stored on the computer; "real-time" is not an issue.

In this section, we will explore what happens when a non-ideal, causal filter is used for signal reconstruction. We will use the most basic filter. Recall the first-order, lowpass

RC filter. It has a frequency response given by

$$H(F) = H(j\Omega)\,|_{\Omega=2\pi F} \;=\; \frac{1}{j2\pi FRC + 1},$$

and its impulse response is:

$$h(t) = \frac{e^{-t/RC}}{RC}u(t),$$

where $u(t)$ denotes the unit step function.

Questions:

4.2.1 Pre-lab: Cutoff Frequency

What is the 3dB cutoff frequency of the above filter?

Answer:

4.2.2 Reconstruction

Modify the `sampling1.m` code of Section 4.1 to use the above filter. Keep in mind that you are really implementing a discrete-time approximation of the above analog filtering using the densely sampled signal `t` and a similarly discretized impulse response. Don't forget the unit step function $u(t)$. This can be implemented in Matlab using the built-in function `heaviside`. Set sampling period $T = 1/(RC)$ to 0.005. Does the reconstructed signal resemble the original signal?

Answer:

4.2.3 Ideal vs Non-Ideal Reconstruction

Modify your program so that it plots the original signal, the reconstruction using the ideal *sinc* interpolation filter, and the reconstruction using the non-ideal RC filter, all together for easy comparison. Setting the time constant of the RC filter to $T = 1/(RC)$ and trying $T = 0.1$, 0.01, and 0.001, observe the results and comment on the performance of the non-ideal filter.

Answer:

5 Frequency Domain Analysis

Now we are going to look at what happens in the frequency domain when we vary the sampling frequency. Since the pure sine wave we were looking at is rather simple in the frequency domain (a sine wave has only a single frequency component), we will look at a signal which has (uncountably) many frequency components: a decaying exponential signal.

Run the following program:

```
% sampling2.m
clf;

% First plot:
subplot(2,2,1)
t = 0:0.005:10;
xa = 2*t.*exp(-t);
plot(t,xa);grid
xlabel('Time, sec');ylabel('Amplitude');
title('Continuous-time signal x_{a}(t)');

% Second plot:
subplot(2,2,2)
wa = 0:10/511:10;
ha = freqs(2,[1 2 1],wa);
plot(wa/(2*pi),abs(ha));grid;
xlabel('Frequency, Hz');ylabel('Magnitude');
title('|X_{a}(j\Omega)|');
axis([0 5/pi 0 2]);

% Third plot:
subplot(2,2,3)
T = 0.7;
n = 0:T:10;
xs = 2*n.*exp(-n);
k = 0:length(n)-1;
stem(k,xs);grid;
xlabel('Time index n');ylabel('Amplitude');
title('Discrete-time signal x[n]');

% Fourth plot:
subplot(2,2,4)
wd = 0:pi/255:pi;
hd = freqz(xs,1,wd);
plot(wd/(2*T*pi), T*abs(hd));grid;
xlabel('Frequency, Hz');ylabel('Magnitude');
title('|X_{\delta}(j\Omega)|');
axis([0 1/(2*T) 0 2])
```

Although the signals are different, the methods used to create, sample and plot them are the same as in Section 4.1. Note the use of `freqs` and `freqz` to create the frequency-domain representations. You are not expected at this time to understand how those

graphs are obtained (you would by the end of this course after learning the discrete-time Fourier transform (DTFT), but in the following questions you will have to interpret them and make some small modifications to them. Don't let yourself get confused with the notation used in the graph headings. Although this section may seem challenging, it's simple if you take it one step at a time.

Questions:

5.0.1 Signal Bandwidth

A lowpass (LP) signal has spectral energy distributed from 0 Hz to some maximum frequency F_m. F_m can also be called the "bandwidth" of a LP signal. (Note there exist other definitions of LP signals and bandwidth.)

A signal with no spectral energy above a finite frequency value is a bandlimited signal.

In this example, a mathematical expression for the Fourier transform is known and provided as parameters to the analog frequency response Matlab function `freqs`. What does the second graph, plot of $|X_a(j\Omega)|$, tell you about the bandwidth of $x_a(t)$? Is $x_a(t)$ bandlimited? What frequency should $x_a(t)$ be sampled at? **[Note: Ω is frequency in rad/s but the frequency axes in the plots are on Hz scale.]**

The code above plots $|X_a(j\Omega)|$ in the frequency domain from 0 to $\frac{5}{\pi}$ Hz because that is the domain over which the frequency transform was done: `wa` ranges from 0 to 10, and $\frac{10}{2\pi}$ is $\frac{5}{\pi}$. Increase this range further and observe the results (hint: you need to change two lines in the code for the "second plot": the `wa` assignment and the `axis` command, and don't forget the "magnifying glass" button is at your disposal).

Answer:

5.0.2 Sampling

The third and fourth plots show the sampled signal in the time and frequency domains, respectively. In the fourth plot, $|X_\delta(j\Omega)|$ is the magnitude of the Fourier transform of the amplitude modulated impulse train. Note that frequency points supplied to `freqz` range from 0 to π. You will understand why this is so after you have learned the discrete-time Fourier transform. For the time being, simply note that the frequency scale for `freqz` is 2π at the sampling frequency. That is, if sampling frequency is F_s, π corresponds to $F_s/2$ and 2π to F_s, and so on. This linear scale relation explains the scaling that is applied to generate the x-axis points and label in the code for the fourth plot.

The code above sets the sampling period to 0.7 sec., which results in some aliasing visible in the fourth plot. Based on your answer to Question 5.0.1, what would be a better sampling period? Modify the code to observe and describe the results.

Answers:

5.0.3 Aliasing

Prelab Question: Why is $|X_\delta(j\Omega)|$ periodic? What frequency range of $|X_\delta(j\Omega)|$ determines the reconstructed signal if reconstruction uses an ideal LPF?

How does aliasing manifest itself in the fourth plot in th above code? Why? Modifying the fourth plot's code to extend the x-axis to two or four times its current range may give you a better idea (hint: both the `wd` and `axis` lines need to be changed, as before). Based on this result, would you change your answer to Question 5.0.2 on the preceding page? Experiment with different sampling periods. Try T=1.0, 0.5 and 0.1 sec. and plot the results.

Answers:

6 Resampling and Downsampling

In this section, we perform experiment in order to hear and see the impact of aliasing distortions. Towards this end, we will make use of a process called sampling rate change, or resampling, i.e., process a given DT signal to produce another DT signal which may or may not have the same sampling rate as the given DT signal. Conceptually, we can think of first ideally reconstruct a CT signal from the given DT signal, and then (re)sample the reconstructed CT signal, using a different sampling clock, to produce a resampled DT signal. In practice, the CT signal reconstruction process is bypassed, so that resampling is entirely performed in the DT domain. In this section, we are going to make use of the conceptual equivalence of resampling in the DT domain and in the CT domain. Our goal is to produce signals distorted by aliasing distortions. Instead of sampling a CT signal with an insufficient sampling rate, we reduce the sampling rate of a DT signal without going into the CT domain.

In general, the primary motivation for resampling a DT signal is to calculate the value of the signal at non-integer time instances and/or change the sampling rate of the signal. Downsampling or subsampling means reducing the sampling rate of a DT signal. For instance, the 1080p high definition television (HDTV) picture format contains 1920x1080 pixels per frame. A standard-definition NTSC picture frame has 640x480 pixels. To display a HDTV frame as a standard definition picture involves subsampling by a factor of 3 horizontally, and a factor of 2.25 vertically.

6.1 Aliasing Distortions in Sounds

In this section we will subsample a sound file. To read a .wav audio file into Matlab, you can use the `audioread` command:

```
[x, fs] = audioread('input_file.wav');
```

Don't forget the semicolon or you will see thousands of sample values scroll down your Matlab window! This command stores the sample values into the `x` array; `fs` will get the sample rate (in hertz). The number of bits per sample is determined from the input audio file header. The function `audioinfo` can be used to read this header to determine parameters such as the number of bits per sample. The audio can be listened to by using the Matlab function `sound(x,fs)`.

You can now perform operations on the elements of `x` and use `audiowrite` to write your modified audio file back to disk. The syntax is simple:

```
audiowrite (y, fs, 'output_file.wav');
```

Here, `y` is the array containing the audio data; `fs` is defined as before. Depending on the manipulations you performed on the audio, they may not have the same values as that of the input audio file; you will need to figure out what these should be.

Questions:

6.1.1 Pre-lab: Reducing the sampling rate

According to the sampling theorem, a sampling frequency of F_s allows representing a signal whose spectral content has frequency no greater than $F_s/2$. Suppose we have a DT signal $x[n]$ that was originally sampled at 44.1 kHz. Suppose we wish to reduce the sampling rate of $x[n]$ by one-eighth, to produce a DT signal $y[n]$ whose sampling rate is 5.5125 kHz. What bandwidth of the CT signal can be represented by $x[n]$? What about $y[n]$?

[For your own interest: Have you listened to music through a telephone connection? Does it sound like CD music? Telephone signals are sampled at 8 kHz, whereas as CD music is sampled at 44.1 kHz. Now you know the signal bandwidth of telephone music versus CD music! By the way, since the maximum human-audible frequency is 20 kHz, why is CD music sampled at 44.1 kHz? Why not 40 kHz?]

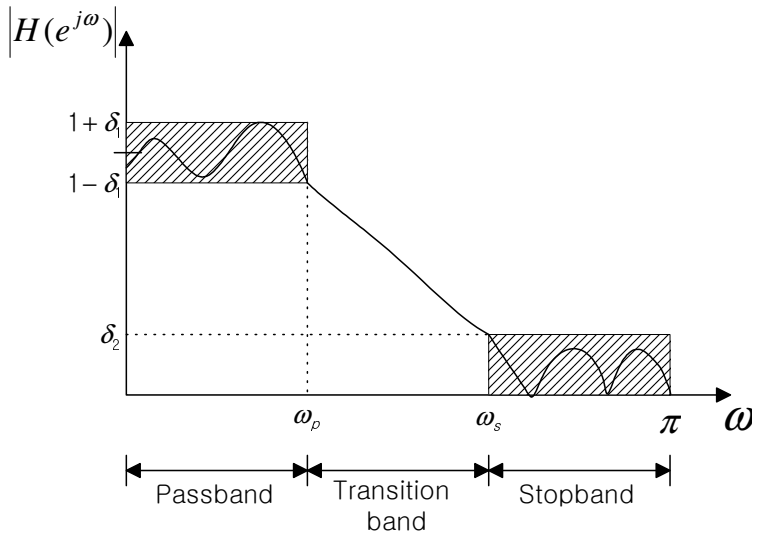
If one wishes to reduce the sampling rate of a DT signal, two steps are required: filtering (in DT domain) and downsampling (by dropping some samples). In what order must these steps be done? Why?

Answer:

6.1.2 Filtering

Matlab has built-in commands to generate certain types of filters; for example, the `butter` command implements a Butterworth filter. If you type "`help butter`", you will see that most forms of the command require three arguments: the order of the filter, its cutoff frequency, and its type (*e.g.* highpass, lowpass, or bandstop). Fortunately for you, there is a `buttord` command which you can use to obtain the order required to meet your specifications.

To reduce the sampling rate of a DT signal by a factor of 8, you need to filter the signal so that its bandwidth is one-eighth of its original bandwidth. What arguments would you give to `buttord`? Note that there is no single right answer, but that you must justify your choices. The figure below illustrates some relevant filter parameters. To decipher the frequency scale in the figure, refer to Section 5.0.2 concerning the frequency scale used by `freqz`. The Wp and Ws parameters of `buttord` use a different normalized frequency scale from `freqz` and the figure. For the parameters, a normalized value of 1 corresponds to half the sampling frequency (of the DT signal that is to be filtered), and a value of 0 corresponds to 0 Hz.



Note that any of the following actions increases the order of a filter: reduce the transition bandwidth, reduce the stopband amplitude (i.e. increase the stopband attenuation), and reduce the passband ripple amplitude. Increased filter order means more computation.

Answer:

6.1.3 Subsampling

Using only the `butter`, `downsample`, `filter`, `audioread` and `audiowrite` commands, write a Matlab program which will read the supplied `bass.wav` audio file, reduce its sampling rate by a factor of 8 without introducing any aliasing, and write the result to disk. Use Matlab's built-in help system to learn about the `butter, downsample`, and `filter` commands. Note that your program only needs five lines: one for each of the above commands (but not in the above listed order). Listen to the resulting file and comment on any subjective quality differences with the original file. You may need to fine-tune the `buttord` parameters based on your results here. Another audio file

`flute.wav` is also supplied for your exploration.

Answer:

A probable pitfall here is to get this warning when using `audiowrite`:

```
Warning:  Data clipped during write to file
```

This is due to inappropriate selection of the filter parameters. The solution is to increase the distance between the passband and stopband frequencies.

6.1.4 Aliasing

Remove the filtering commands from your code so that downsampling is performed without filtering. Compare the results. How does aliasing manifest itself? Comment subjectively on the differences between the results with and without filtering.

Answer:

6.1.5 Decimating

Use the `decimate` command to consolidate the `butter, downsample`, and `filter` commands into a single step (assume for now that the `cheby1` filter is roughly equivalent to the `butter` filter). Is the end result the same?

Answer:

6.2 Image Aliasing

In this section we will subsample an image file. Although image processing is not covered in this course, the same principles apply: treat an image as a function defined over a two-dimensional domain, or a function with two independent variables. The Matlab command `imresize` performs image subsampling, but in this section you will be guided to a more manual approach in order to gain a better understanding of what happens to an image when it is subsampled.

You will work with the supplied `NYcity.jpg` image. An image can be read into Matlab with the `imread` command:

```
A = imread('NYcity.jpg');
```

Here, `A` will be a matrix with dimensions corresponding to the size of the image, with each entry of the matrix containing the grayscale value of the corresponding pixel. This is because we are working with a grayscale image; things would be a little more complicated with a colour image (and you will see this in another lab).

As was the case with sound, there are two components to resampling: filtering and downsampling. Filtering here is done via two-dimensional convolution of the image with

an appropriate matrix. The matrix contains the impulse response samples of a two-dimensional filter. You are not expected to understand the details of this; you just need to know that this code accomplishes filtering and use it in your program:

```
M = [ 1 4 6 4 1;
4 16 24 16 4;
6 24 36 24 6;
4 16 24 16 4;
1 4 6 4 1 ] / 256;
B = uint8(conv2(double(A),M));
```

Questions:

6.2.1 Downsampling

You have used the `downsample` command on a one-dimensional signal before. Use it to downsample the following 4 by 4 matrix. Your result should be a 2 by 2 matrix, with every other entry of the original matrix missing.

```
M = [ 1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16];
```

Answer:

6.2.2 Resampling

Using a combination of downsampling and filtering (in the correct order), resample the given image by a factor of two. Compare this to the result obtained when only downsampling is done (i.e. no filtering). Describe the differences observed.

Answer:

6.2.3 Filtering

Could the given filter code be used to subsample by an integer factor greater than 2? Why or why not? *Hint:* think in terms of downsampling an audio signal. What filter cutoff frequency would you need for downsampling by a factor of k, where k is a positive integer?

Answer:

6.2.4 Filter Type

For your own exploration, you can compare the above subsampled image with the subsampled images generated by `imresize`. `imresize` provides several types of subsampling filters to choose from. Can you tell which filter, including the 2-D filter supplied above, provides the best visual quality?