ECE438 - Laboratory 1:
Discrete and Continuous-Time Signals

By Prof. Charles Bouman and Prof. Mireille Boutin
Fall 2016

1 Introduction

The purpose of this lab is to illustrate the properties of continuous and discrete-time signals using digital computers and the Matlab software environment. A continuous-time signal takes on a value at every point in time, whereas a discrete-time signal is only defined at integer values of the "time" variable. However, while discrete-time signals can be easily stored and processed on a computer, it is impossible to store the values of a continuous-time signal for all points along a segment of the real line. In later labs, we will see that digital computers are actually restricted to the storage of quantized discrete-time signals. Such signals are appropriately known as *digital* signals.

How then do we process continuous-time signals? In this lab, we will show that continuous-time signals may be processed by first approximating them by discrete-time signals using a process known as sampling. We will see that proper selection of the spacing between samples is crucial for an efficient and accurate approximation of a continuous-time signal. Excessively close spacing will lead to too much data, whereas excessively distant spacing will lead to a poor approximation of the continuous-time signal. Sampling will be an important topic in future labs, but for now we will use sampling to approximately compute some simple attributes of both real and synthetic signals.

2 Laboratory Ethics

Students are expected to behave ethically both in and out of the lab. Unethical behavior includes, but is not limited to, the following:

- Possession of another person's laboratory solutions from the current or previous years.
- Reference to, or use of another person's laboratory solutions from the current or previous years.

Questions or comments concerning this laboratory should be directed to Prof. Mireille Boutin, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907

- Submission of work that is not done by your laboratory group.
- Allowing another person to copy your laboratory solutions or work.
- Cheating on quizzes.

The ECE438 laboratory experience is meant to strengthen and deepen the student's understanding of basic concepts taught in the ECE438 lectures and to help the student develop practical skills in applying the concepts taught in the ECE438 course. The rules of laboratory ethics are designed to facilitate these goals. We emphasize that laboratory teaching assistants are available throughout the week to help the student both understand the basic concepts and answer the questions being asked in the laboratory exercises. By performing the laboratories independently, students will likely learn more and improve their performance in the course as a whole.

Please note that it is the responsibility of the student to make sure that the content of their graded laboratories is not distributed to other students. If there is any question as to whether a given action might be considered unethical, please see the professor or the TA before you engage in such actions.

INLAB REPORT:
Each student on the team must *write by hand* the following statement in the lab report, sign and date.
"I have read and understood the Laboratory Ethics section (Section 2) of Laboratory 1. I pledge to behave ethically and with honesty in ECE438 this semester. The reports I will hand in will be the product of original work by myself and my teammate, and no one else. I will not look at other people's laboratory. I will not use other people's code. I will not make my labs available to other students beyond my teammates, even after the semester is over. In particular, I will not post my labs on the Internet or make my files available to other people. I will not be a cheater. " Name, Signature, Date.

3 Matlab Review

Practically all lab tasks in the ECE438 lab will be performed using Matlab. Matlab (MATrix LABoratory) is a technical computing environment for numerical analysis, matrix computation, signal processing, and graphics. In this section, we will review some of its basic functions. For a short tutorial and some Matlab examples see

`https://engineering.purdue.edu/ECN/Support/KB/Docs/MatlabCharlesBoumans/index_html`

3.1 Starting Matlab and Getting Help

You can start Matlab (version 7.0) on your workstation by typing the command

```
matlab
```

in a command window. After starting up, you will get a Matlab window. To get help on any specific command, such as "plot", you can type the following

```
help plot
```

in the "Command Window" portion of the Matlab window. You can do a keyword search for commands related to a topic by using the following

```
lookfor topic
```

You can get an interactive help window using the function

```
helpdesk
```

or by following the Help menu near the top of the Matlab window.

3.2 Matrices and Operations

Every element in Matlab is a matrix. So, for example, the Matlab command

```
a = [7 3 4]
```

creates a matrix named "a" with dimensions of 1×3. To access a specific entry inside the matrix, one uses the index representing the position of the desired entry in the matrix. For example $a[2]$ corresponds to the number 3 in our previous example.

The variable "a" is stored in what is called the Matlab workspace. The operation

```
b = a.'
```

stores the transpose of "a" into the vector "b". In this case, "b" is a 3×1 vector.

Since each element in Matlab is a matrix, the operation

```
c = a*b
```

computes the **matrix** product of "a" and "b" to generate a **scalar** value for "c" of $74 = 7 \times 7 \ + \ 3 \times 3 \ + \ 4 \times 4$.

Often, you may want to apply an operation to each element of a vector. For example, you many want to square each value of "a". In this case, you may use the following command.

```
c = a.*a
```

The dot before the * tells Matlab that the multiplication should be applied to each corresponding element of "a". Therefore the `.*` operation is *not* a matrix operation. The dot convention works with many other Matlab commands such as divide `./`, and power `.^`. An error results if you try to perform element-wise operations on matrices that aren't the same size.

Note also that while the operation `a.'` performs a transpose on the matrix "a", the operation `a'` performs a *conjugate* transpose on "a" (transposes the matrix and conjugates each number in the matrix).

3.3 Matlab Scripts and Functions

Matlab has two methods for saving sequences of commands as standard files. These two methods are called *scripts* and *functions*. Scripts execute a sequence of Matlab commands just as if you typed them directly into the Matlab command window. Functions differ from scripts because they take inputs and return outputs.

A script-file is a text file with the filename extension ".m" . The file should contain a sequence of Matlab commands. The script-file can be run by typing its name at the Matlab prompt without the .m extension. This is equivalent to typing in the commands at the

prompt. Within the script-file, you can access variables you defined earlier in Matlab. All variables in the script-file are global, i.e. after the execution of the script-file, you can access its variables at the Matlab prompt. For more help on scripts, please refer to the following file

`https://engineering.purdue.edu/VISE/ee438L/matlab/help/pdf/script.pdf`

To create a function call "func", you first create a file called "func.m". The first line of the file must be

```
function output = func(input)
```

where "input" designates the set of input variables, and "output" are your output variables. The rest of the function file then contains the desired operations. All variables in the function are local; that means the function cannot access Matlab workspace variables that you don't pass as inputs. After the execution of the function, you cannot access internal variables of the function. For more help on functions please refer to the following file

`https://engineering.purdue.edu/VISE/ee438L/matlab/help/pdf/function.pdf`

4 Continuous-Time Vs. Discrete-Time

The introduction in Section 1 mentioned the important issue of representing continuous-time signals on a computer. In the following sections, we will illustrate the process of *sampling*, and demonstrate the importance of the *sampling interval* to the precision of numerical computations.

4.1 Displaying Continuous-Time and Discrete-Time Signals in Matlab

It is common to graph a discrete-time signal as dots in a Cartesian coordinate system. This can be done in the Matlab environment by using the *stem* command. We will also use the *subplot* command to put multiple plots on a single figure.

Start Matlab on your workstation and type the following sequence of commands.

```
n = 0:2:60;
y = sin(n/6);
subplot(3,1,1)
stem(n,y)
```

This plot shows the discrete-time signal formed by computing the values of the function $\sin(t/6)$ at points which are uniformly spaced at intervals of size 2. Notice that while $\sin(t/6)$ is a continuous-time function, the sampled version of the signal, $\sin(n/6)$, is a discrete-time function.

A digital computer cannot store all points of a continuous-time signal since this would require an infinite amount of memory. It is, however, possible to plot a signal which *looks like* a continuous-time signal, by computing the value of the signal at closely spaced points in time, and then connecting the plotted points with lines. The Matlab *plot* function may

be used to generate such plots.

Use the following sequence of commands to generate two continuous-time plots of the signal $\sin(t/6)$. [1]

```
n1 = 0:2:60;
z = sin(n1/6);
subplot(3,1,2)
plot(n1,z)
n2 = 0:10:60;
w = sin(n2/6);
subplot(3,1,3)
plot(n2,w)
```

As you can see, it is important to have many points to make the signal appear smooth. But how many points are enough for numerical calculations? In the following sections we will examine the effect of the sampling interval on the accuracy of computations.

INLAB REPORT:
Submit a hard copy of the plots of the discrete-time function and two continuous-time "looking" functions. Label them with the *title* command, and include your names. Comment on the accuracy of each of the continuous time plots.

4.2 Vector Index versus Time

In Matlab, the possible indices of a vector are the natural integers, starting from one (i.e., $1, 2, 3, 4, 5, \ldots$). Vector indices can neither be negative nor zero. For example, there is no vector entry corresponding to $a[0]$ or $a[-1]$: referring to such entries would yield an error message.

We saw in 4.1 that the samples of a continuous-time signal, say $x(t)$, can be stored in a vector in Matlab. It is common practice to use the same variable for the vector and the signal. So one often denotes the samples of $x(t)$ by $x[n]$, even though this is an abuse of notation and lacks rigor.

It is important not to confuse the index of a vector $x[n]$ with the value of the independent variable of a function $x(t)$. For example, Matlab can be used to represent the function $x(t) = \sin(t)$ by sampling t at small intervals. The resulting samples may be stored in a vector called "x" in your program. However, it is important to realize that the function "x" and the vector "x" in the program are not the same thing. The following code illustrates this.

```
t1=-10:0.1:10;
x=sin(t1);
```

[1] If you have trouble printing, make sure LPDEST is defined. Refer to the printing information on the VISE home page.

```
subplot(3,1,1)
plot(x)
subplot(3,1,2)
plot(t1,x)
subplot(3,1,3)
t2=0:0.1:20;
plot(t2,x)
```

INLAB REPORT:
Print the three subplots and explain the difference between the three signals represented. Write Matlab command(s) that would print the graph of sin(t) for the values of t on the interval [3.5,4.5]. (Pick a suitable increment for t.)

4.3 Analytical Calculation

Compute these two integrals. Do the computations manually.

1.
$$\int_0^{2\pi} sin^2(7t)dt \tag{1}$$

2.
$$\int_0^1 e^t dt \tag{2}$$

INLAB REPORT:
Hand in your calculations of these two integrals. Show all work.

4.4 Numerical Computation of Continuous-Time Signals

Background on Numerical Integration

One common calculation on continuous-time signals is integration. Figure 1 illustrates a method used for computing the widely used Riemann integral. The Riemann integral approximates the area under a curve by breaking the region into many rectangles and summing their areas. Each rectangle is chosen to have the same width Δt, and the height of each rectangle is the value of the function at the start of the rectangle's interval.

To see the effects of using a different number of points to represent a continuous-time signal, write a Matlab function for numerically computing the integral of the function $sin^2(7t)$ over the interval $[0, 2\pi]$. The syntax of the function should be `I=integ1(N)` where I is the result and N is the number of rectangles used to approximate the integral. This function should use the *sum* command and it should *not* contain any *for* loops!

Note: Since Matlab is an *interpreted* language, *for loops* are relatively slow. Therefore, we will avoid using loops whenever possible.

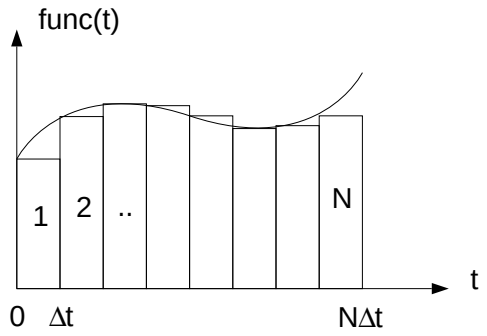

Figure 1: Illustration of the Riemann integral

Next write an m-file script that evaluates $I(N)$ for $1 \leq N \leq 100$, stores the result in a vector and plots the resulting vector as a function of N. This m-file script may contain *for* loops.

Repeat this procedure for a second function `J=integ2(N)` which numerically computes the integral of $\exp(t)$ on the interval $[0, 1]$.

INLAB REPORT:
Submit plots of $I(N)$ and $J(N)$ versus N. **Use the subplot command to put both plots on a single sheet of paper.** Also submit your Matlab code for each function. Compare your results to the analytical solutions from Section 4.3. Explain why $I(7) = I(14) = 0$.

5 Processing of Speech Signals

Download speech.au file
`https://engineering.purdue.edu/VISE/ee438L/lab1/data/speech.zip`
How to load and play audio signals:
`https://engineering.purdue.edu/VISE/ee438L/matlab/help/pdf/audio.pdf`

Digital signal processing is widely used in speech processing for applications ranging from speech compression and transmission, to speech recognition and speaker identification. This exercise will introduce the process of reading and manipulating a speech signal.

First download the speech audio file speech.au , and then do the following:

1. Use the *audioread* command to load the file *speech.au* into Matlab.

2. Plot the signal on the screen as if it were a continuous-time signal (i.e. use the *plot* command).

3. Play the signal via the digital-to-analog converter in your workstation with the Matlab *sound* function.

INLAB REPORT:
Submit your plot of the speech signal.

6 Special Functions

Plot the following two continuous-time functions over the specified intervals. Write separate script files if you prefer. Use the *subplot* command to put both plots in a single figure, and be sure to label the time axes.

- $\begin{cases} \frac{\sin \pi t}{\pi t}, & t \neq 0 \\ 1. & t = 0 \end{cases}$ for $t \in [-10\pi, 10\pi]$
- $\text{rect}(t)$ for $t \in [-2, 2]$

Hint: The function *rect(t)* may be computed in Matlab by using a Boolean expression. For example, if `t=-10:0.1:10`, then $y = rect(t)$ may be computed using the Matlab command `y = (abs(t)<=0.5)`.

Write an .m-script file to stem the following discrete-time function for $a = 0.8$, $a = 1.0$ and $a = 1.5$. Use the *subplot* command to put all three plots in a single figure. Issue the command `orient('tall')` just prior to printing to prevent crowding of the subplots.

- $a^n(u[n] - u[n-10])$ for $n \in [-20, 20]$

Repeat this procedure for the function

- $\cos(\omega n)a^n u[n]$ for $\omega = \pi/4$, and $n \in [-1, 10]$

Hint: The unit step function $y = u[n]$ may be computed in Matlab using the command `y = (n>=0)` , where `n` is a vector of values of time indices.

INLAB REPORT:
Submit all three figures, for a total of 8 plots. Also submit the printouts of your Matlab .m-files.

7 Sampling

The word *sampling* refers to the conversion of a continuous-time signal into a discrete-time signal. The signal is converted by taking its value, or sample, at uniformly spaced points in time. The time between two consecutive samples is called the *sampling period*. For example, a sampling period of 0.1 seconds implies that the value of the signal is stored every 0.1 seconds.

Consider the signal $f(t) = \sin(2\pi t)$. We may form a discrete-time signal, $x[n]$, by sampling this signal with a period of T_s. In this case,

$$x(n) = f(T_s n) = \sin(2\pi T_s n) \ .$$

Use the *stem* command to plot the function $f(T_s n)$ defined above for the following values of T_s and n. Use the *subplot* command to put all the plots in a single figure, and scale the plots properly with the *axis* command.

1. $T_s = 1/10$, $0 \leq n \leq 100$; axis([0,100,-1,1])
2. $T_s = 1/3$, $0 \leq n \leq 30$; axis([0,30,-1,1])
3. $T_s = 1/2$, $0 \leq n \leq 20$; axis([0,20,-1,1])
4. $T_s = 10/9$, $0 \leq n \leq 9$; axis([0,9,-1,1])

INLAB REPORT:
Submit a hardcopy of the figure containing all four subplots. Discuss your results. How does the sampled version of the signal with $T_s = 1/10$ compare to those with $T_s = 1/3$, $T_s = 1/2$ and $T_s = 10/9$?

8 2-D Signals

So far we have only considered 1-D signals such as speech signals. However, 2-D signals are also very important in digital signal processing. For example, the elevation at each point on a map, or the color at each point on a photograph are examples of important 2-D signals. As in the 1-D case, we may distinguish between continuous-space and discrete-space signals. However in this section, we will restrict attention to discrete-space 2-D signals.

When working with 2-D signals, we may choose to visualize them as images or as 2-D surfaces in a 3-D space. To demonstrate the differences between these two approaches, we will use two different display techniques in Matlab. Do the following:

1. Use the *meshgrid* command to generate the discrete-space 2-D signal

   $$f[m,n] = 255|\text{sinc}(0.2m)\sin(0.2n)|$$

   for $-50 \leq m \leq 50$ and $-50 \leq n \leq 50$. See the help on **meshgrid** if you're unfamiliar with its usage.

2. Use the *mesh* command to display the signal as a surface plot.

3. Display the signal as an image. Use the command `colormap(gray(256))` just after issuing the *image* command to obtain a grayscale image. Read the help on **image** for more information.

INLAB REPORT:
Hand in hardcopies of your mesh plot and image. For which applications do you think the surface plot works better? When would you prefer the image?

9 2D Random Signals- Optional Exercise

Help on random function:
`https://engineering.purdue.edu/VISE/ee438L/matlab/help/pdf/random.pdf`

The objective of this section is to show how to recover a signal from noisy observations of that signal.

Generate one 100x100 image with a 10x10 white square in the middle (pixel value 1) on a black background (pixel value 0). Add a random number to each pixel value of the image. The random number should be generated independently following a uniform distribution on the interval $[0, 1]$. Use the Matlab command *rand* to generate these random numbers. This will create a noisy observation of the image. Display the resulting 2D signal as an image. Can you distinguish the square in the center of the noisy image?

Repeat this procedure to generate 99 additional (different) noisy observations of the image of the square. Then obtain a new image by averaging the pixel values of each of these 100 images. Plot the resulting new image. Can you distinguish the square in the center of the new image?