EE323 Digital Signal Processing
Lab 1 – Discrete and Continuous-Time Signals
25 Sept. 2019 ~ 8 Oct. 2019

1.1 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.

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.

1.2 Review of Matlab

Practically all lab tasks in the EE323 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.

1.2.1 Starting Matlab and Getting Help

You can start Matlab on your computer by clicking the icon

on your desktop. 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

Here, replace "topic" by a word of topic that you are interested.

You can get an interactive help window using the function

doc

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

1.2.2. Matrices and Operations

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

$a = [1\ 2\ 3]$

creates a matrix named "a" with dimensions of 1×3. 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 $14 = 1*1 + 2*2 + 3*3$.

Often, you may want to apply an operation to each element of a vector. For example, you may 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).

In addition, you may also add a ";" after a command, e.g.,

$a = [1\ 2\ 3];$

$b = a.';$

$c = a * b;$

c

$c = a. * a;$

c

Compare the difference between the commands with or without ";".

1.2.3 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 in that they accept inputs and return outputs, and variables defined within a function are generally local to that function.

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, see script.pdf.

To create a function nameed **func**, you first create a text file named 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 within 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 see function.pdf.

1.3 Continuous-Time Vs. Discrete-Time

The "Introduction" (Section 1.1: Introduction) 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.

1.3.1 Analytical Calculation

Compute these two integrals.   Do the computation manually.

1.

$$\int_0^{2\pi}(\sin 5t)^2\,dt$$

2.

$$\int_0^1 e^t\,dt$$

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

1.3.2 Displaying Continuous- and Discrete-Time Signals in Matlab

To get help on the following topics, see plot.pdf, stem.pdf, and subplot.pdf

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).

```
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 functions. Label them with the **title** command, and include your names. Comment on the accuracy of each of the continuous time plots.

1.3.3 Numerical Computation of Continuous-Time Signals

To get help on the following topics, see script.pdf, function.pdf, and subplot.pdf.

Background on Numerical Integration
One common calculation on continuous-time signals is integration. Figure 2.1 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 $\sin^2(5t)$ 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.
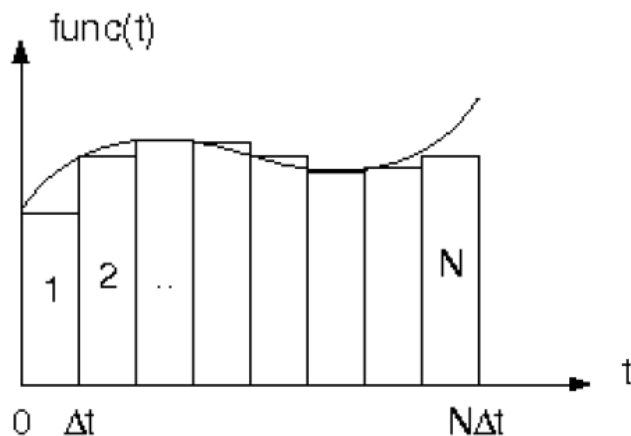
Figure 2.1: Illustration of the Riemann integral

Next write an m-file script that evaluates $I(N)$ for $1 \le N \le 100$, store the result in a vector and plot the resulting vector as a function of N. This m-file script may contain for loops.
Repeat this procedure for a second function $J = \text{integ2}(N)$ 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 the "Analytical Calculation" (Section 2.1: Analytical Calculation) section. Explain why $I(5) = I(10) = 0$.

1.4 Processing of Speech Signals (Optional)

For this section download the speech.au file. For instructions on how to load and play audio signals see 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 **auread** 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 computer with the Matlab **sound** function.

> **INLAB REPORT**: Submit your plot of the speech signal.

1.5 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.

🔹 sinc (t) for t in $[-10\pi, 10\pi]$

🔹 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 time indices.

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

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

1.7 Random Signals

For help on the Matlab random function, see **random.pdf**.

The objective of this section is to show how two signals that "look" similar can be distinguished by computing their average over a large interval. This type of technique is used in signal demodulators to distinguish between the digits "1" and "0".

Generate two discrete-time signals called "sig1" and "sig2" of length 1,000. The samples of "sig1" should be independent, Gaussian random variables with mean 0 and variance 1. The samples of "sig2" should be independent, Gaussian random variables with mean 0.2 and variance 1. Use the Matlab command **random** or **randn** to generate these signals, and then plot them on a single figure using the **subplot** command. (Recall that an alternative name for a Gaussian random variable is a normal random variable.)

Next form a new signal "ave1(n)" of length 1,000 such that "ave1(n)" is the average of the vector "sig1(1:n)" (the expression sig1(1:n) returns a vector containing the first *n* elements of "sig1"). Similarly, compute "ave2(n)" as the average of "sig2(1:n)". Plot the signals "ave1(n)" and "ave2(n)" versus "n" on a single plot. Refer to help on the Matlab **plot** command for information on plotting multiple signals.

INLAB REPORT: Submit your plot of the two signals "sig1" and "sig2". Also submit your plot of the two signals "ave1" and "ave2". Comment on how the average values changes with *n*. Also comment on how the average values can be used to distinguish between random noise with different means.

1.8 2-D Signals (Optional)

For help on the following topics, see meshgrid.pdf, mesh.pdf and image.pdf.

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 of meshgrid.pdf 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 in image.pdf 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?

1.9 Exercises:

1. Write a Matlab program to generate the conjugate-symmetric and conjugate-antisymmetric parts of a finite length complex sequence. Use this program to verify the results of Example 1 on Slide Page 49 of Lecture 2.

2. Write a program to generate and plot the real and imaginary parts, respectively, of the complex exponential sequences
(1) $e^{(-1/12+j\pi/6)n}$, for $0 \le n \le 40$,
(2) $-2.7e^{(-0.4+j\pi/6)n}$, for $0 \le n \le 82$.

3. Write a Matlab program to generate a sinusoidal sequence $x[n] = A\sin(\omega_0 n + \varphi)$, and plot the sequence using the **stem** function. The input data specified by the user are the desired length L, the amplitude A, the angular frequency ω_0, and the phase φ, where $0 < \omega_0 < \pi$ and $0 \le \varphi \le 2\pi$. Using this program, generate the sinusoidal sequences shown in the figures on Slide Pages 60 and 61 of Lecture 2.

4. Write a Matlab program to verify the results on Slide Page 62 of Lecture 2.

INLAB REPORT: Submit the hardcopy of matlab code of the above programs and functions, as well as the plots of each exercise.