Purdue Digital Signal Processing Labs (ECE 438)

By:
Charles A. Bouman

Online:
< http://cnx.org/content/col10593/1.4/ >

C O N N E X I O N S

Rice University, Houston, Texas

DISCRETE AND CONTINUOUS-TIME SIGNALS

Collection structure revised: September 14, 2009
PDF generated: October 26, 2012

2.1 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.

NOTE: Be sure to read the guidelines for the written reports.

2.2 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 click here[2] .

[1] This content is available online at <http://cnx.org/content/m18073/1.3/>.
[2] https://engineering.purdue.edu/ECN/Support/KB/Docs/MatlabCharlesBoumans

2.2.1 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 window.

2.2.2 Matrices and Operations

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

$a = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

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 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).

2.2.3 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 click here[3].

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

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

[3]See the file at <http://cnx.org/content/m18073/latest/script.pdf>

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 click here[4].

2.3 Continuous-Time Vs. Discrete-Time

The "Introduction" (Section 2.1: 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.

2.3.1 Analytical Calculation

Compute these two integrals. Do the computations manually.

1.
$$\int_0^{2\pi} sin^2\left(5t\right) dt \tag{2.1}$$

2.
$$\int_0^1 e^t dt \tag{2.2}$$

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

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

For help on the following topics, visit the corresponding link: Plot Function[5], Stem Command[6], and Subplot Command[7].

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 $sin(t/6)$.

[4]See the file at <http://cnx.org/content/m18073/latest/function.pdf>
[5]See the file at <http://cnx.org/content/m18073/latest/plot.pdf>
[6]See the file at <http://cnx.org/content/m18073/latest/stem.pdf>
[7]See the file at <http://cnx.org/content/m18073/latest/subplot.pdf>

```
    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.

2.3.3 Numerical Computation of Continuous-Time Signals

For help on the following topics, click the corresponding link: MatLab Scripts[8], MatLab Functions[9], and the Subplot Command[10].

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.

[8] See the file at <http://cnx.org/content/m18073/latest/script.pdf>
[9] See the file at <http://cnx.org/content/m18073/latest/function.pdf>
[10] See the file at <http://cnx.org/content/m18073/latest/subplot.pdf>

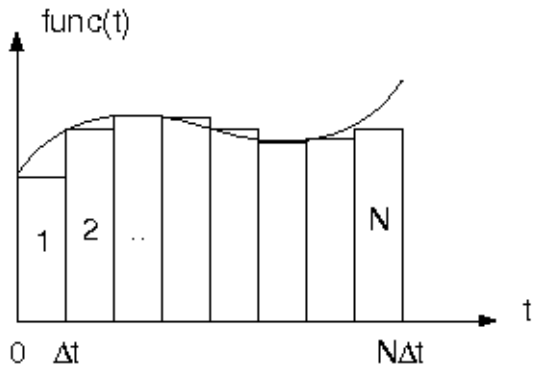

Figure 2.1: Illustration of the Riemann integral

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.

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 the "Analytical Calculation" (Section 2.3.1: Analytical Calculation) section. Explain why $I(5) = I(10 = 0$.

2.4 Processing of Speech Signals

For this section download the speech.au[11] file. For instructions on how to load and play audio signals click here[12].

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.

[11] See the file at <http://cnx.org/content/m18073/latest/speech.au>
[12] See the file at <http://cnx.org/content/m18073/latest/audio.pdf>

First download the speech audio file speech.au[13], 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 workstation with the Matlab `sound` function.

INLAB REPORT: Submit your plot of the speech signal.

2.5 Attributes of Continuous-Time Signals

For this section download the signal1.p[14] function.

In this section you will practice writing .m-files to calculate the basic attributes of continuous-time signals. Download the function signal1.p[15]. This is a pre-parsed pseudo-code file (P-file), which is a "pre-compiled" form of the Matlab function **signal1.m**. To evaluate this function, simply type `y = signal1(t)` where `t` is a vector containing values of time. Note that this Matlab function is valid for any real-valued time, `t` , so `y = signal1(t)` yields samples of a continuous-time function.

First plot the function using the `plot` command. Experiment with different values for the sampling period and the starting and ending times, and choose values that yield an accurate representation of the signal. Be sure to show the corresponding times in your plot using a command similar to `plot(t,y)`.

Next write individual Matlab functions to compute the minimum, maximum, and approximate energy of this particular signal. Each of these functions should just accept an input vector of times, `t`, and should call `signal1(t)` within the body of the function. You may use the built-in Matlab functions `min` and `max`. Again, you will need to experiment with the sampling period, and the starting and ending times so that your computations of the min, max, and energy are accurate.

Remember the definition of the energy is

$$\text{energy} = \int_{-\infty}^{\infty} |\text{signal1}\,(t)|^2 dt \ . \tag{2.3}$$

INLAB REPORT: Submit a plot of the function, and the computed values of the min, max, and energy. Explain your choice of the sampling period, and the starting and ending times. Also, submit the code for your energy function.

2.6 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.

- $\text{sinc}\,(t)$ 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)`.

[13] See the file at <http://cnx.org/content/m18073/latest/speech.au>
[14] See the file at <http://cnx.org/content/m18073/latest/signal1.p>
[15] See the file at <http://cnx.org/content/m18073/latest/signal1.p>

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.

2.7 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,

$$x(n) = f(T_s n) = sin(2\pi T_s n) \quad . \tag{2.4}$$

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 $T_s = 10/9$?

2.8 Random Signals

For help on the Matlab random function, click here[16].

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

[16] See the file at <http://cnx.org/content/m18073/latest/random.pdf>

`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[17] 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.

2.9 2-D Signals

For help on the following topics, click the corresponding link: Meshgrid Command[18], Mesh Command[19], and Displaying Images[20].

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

$$f(m,n) = 255|\text{sinc}(0.2m)\,sin(0.2n)| \tag{2.5}$$

   for $-50 \leq m \leq 50$ and $-50 \leq n \leq 50$. See the help on meshgrid[21] 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[22] 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?

[17] See the file at <http://cnx.org/content/m18073/latest/plot.pdf>
[18] See the file at <http://cnx.org/content/m18073/latest/meshgrid.pdf>
[19] See the file at <http://cnx.org/content/m18073/latest/mesh.pdf>
[20] See the file at <http://cnx.org/content/m18073/latest/image.pdf>
[21] See the file at <http://cnx.org/content/m18073/latest/meshgrid.pdf>
[22] See the file at <http://cnx.org/content/m18073/latest/image.pdf>