C.1 Arrays and sound

The purpose of this lab is to explore arrays in Matlab and to use them to construct sound signals. The lab is designed to help you become familiar with the fundamentals of Matlab. It is self contained, in the sense that no additional documentation for Matlab is needed. Instead, we rely on the on-line help facilities. Some people, however, much prefer to sit down with a tutorial text about a piece of software, rather than relying on on-line help. There are many excellent books that introduce Matlab. Check your local bookstore or The MathWorks' website (http://www.mathworks.com/).

Note that there is some potential confusion because Matlab uses the term "function" somewhat more loosely than we do when we refer to mathematical functions. Any Matlab command that takes arguments in parentheses is called a function. And most have a well-defined domain and range, and do, in fact, define a mapping from the domain to the range. These can be viewed formally as a (mathematical) functions. Some, however, such as `plot` and `sound` are a bit harder to view this way. The last exercise here explores this relationship.

C.1.1 In-lab section

To run Matlab simply double click on the Matlab icon on the desktop, or find the Matlab command in the start menu. This will open a Matlab command window, which displays a prompt "`>>`". You type commands at the prompt. Explore the built-in demos by typing `demo`.

Matlab provides an on-line help system accessible by using the `help` command. For example, to get information about the function `size`, enter the following:

```
>> help size
```

There also is a help desk (formatted in HTML for viewing from a web browser) with useful introductory material. It is accessed from the Help menu. If you have no prior experience with Matlab, see the topic "Getting Started" in the help desk. Spend some time with this. You can find in the help desk all the information you need to carry out the following exercises.

1. A variable in Matlab is an array. An array has dimensions $N \times M$, where N and M are in *Naturals*. N is the number of **rows** and M is the number of **columns**. If $N = M = 1$, the variable is a **scalar**. If $N = 1$ and $M > 1$, then the variable is a **row vector**. If $N > 1$ and $M = 1$, then the variable is a **column vector**. If both N and M are greater than one, then the variable is a **matrix**, and if $N = M$ then the variable is a **square matrix**. The coefficients of an array are real or complex numbers.

   (a) Each of the following is an assignment of a value to a variable called `array`. For each, identify the dimensions of the array (M and N), and identify whether the variable is a scalar, row vector, column vector, or matrix.

   ```
   array = [1 2 3 4 5]
   array = [1:5]
   ```

```
array = 1:5
array = [1:1:5]
array = [1:-1:-5]
array = [1 2; 3 4]
array = [1; 2; 3; 4]
```

(b) Create a 2×3 matrix containing arbitrary data. Explore using the Matlab functions `zeros`, `ones`, `eye`, and `rand` to create the matrix. Find a way to use the square matrix `eye(2)` as part of your 2×3 matrix. Verify the sizes of your arrays using `size`.

(c) Use the Matlab commands `size` and `length` to determine the length of the arrays given by `1:0.3:10` and `1:1:-1`. Consider more generally the array constructor pattern

```
array = start : step : stop
```

where `start`, `stop`, and `step` are scalar variables or real numbers. How many elements are there in `array`? Give an expression in Matlab in terms of the variables `start`, `stop`, and `step`. That is, we should be able to do the following:

```
>> start = 1;
>> stop = 5;
>> step = 1;
>> array = start:step:stop;
```

and then evaluate your expression and have it equal `length(array)`. (Notice that the semicolons at the end of each command above suppress Matlab's response to each command.) Hint: to get a general expression, you will need something like the `floor` function. Verify your answer for the arrays `1:0.3:10` and `1:1:-1`.

2. Matlab can be used as a general-purpose programming language. Unlike a general-purpose programming language, however, it has special features for operating on arrays that make it especially convenient for modeling signals and systems.

(a) In this exercise, we will use Matlab to compute

$$\sum_{k=0}^{25} k.$$

Use a `for` loop (try `help for`) to specify each individual addition in the summation.

(b) Use the `sum` function to give a more compact, one-line specification of the sum in part (a). The difference between these two approaches illustrates the difference between using Matlab and using a more traditional programming language. The for loop is closer to the style one would use with C++ or Java. The `sum` function illustrates what Matlab does best: compact operations on entire arrays.

(c) In Matlab, any built-in function that operates on a scalar can also operate on an array. For example,

```
>> sin(pi/4)

ans =

    0.7071

>> sin([0 pi/4 pi/2 3*pi/4 pi])

ans =

         0    0.7071    1.0000    0.7071    0.0000
```

This feature is called **vectorization**. Use vectorization to construct a vector that tabulates the values of the sin function for the set $\{0, \pi/10, 2\pi/10, \cdots, \pi\}$. Use the colon notation explored in the previous exercise.

(d) Given two arrays `A` and `B` that have the same dimensions, Matlab can multiply the elements pointwise using the `.*` operator. For example,

```
>> [1 2 3 4].*[1 2 3 4]

ans =

     1     4     9    16
```

Use this pointwise multiply to tabulate the values of sin^2 for the set

$$\{0, \pi/10, 2\pi/10, \cdots, \pi\}.$$

3. A discrete-time signal may be approximated in Matlab by a vector (either a row or a column vector). In this exercise, you build a few such vectors and plot them.

(a) Create an array that is a row vector of length 36, with zeros everywhere except in the 18th position, which has value 1. (Hint: try `help zeros` to find a way to create a row vector with just zeros, and then assign the 18-th element of this vector the value one.) This array approximates a discrete-time **impulse**, which is a signal that is zero everywhere except at one sample point. We will use impulses to study linear systems. Plot the impulse signal, using both `plot` and `stem`.

(b) Sketch by hand the sine wave $x : [-1,1] \rightarrow \mathit{Reals}$, given by

$$\forall t \in [-1,1], \quad x(t) = \sin(2\pi \times 5t + \pi/6).$$

In your sketch carefully plot the value at time 0. Assume the domain represents time in seconds. What is the frequency of this sine wave in Hertz and in radians per second, what is its period in seconds, and how many complete cycles are there in the interval $[-1,1]$?

(c) Sample the function x from the previous part at 8 kHz, and using Matlab, plot the samples for the entire interval $[-1,1]$. How many samples are there?

(d) Change the frequency of the sinewave from the previous section to 440 Hz and plot the signal for the interval $[-1,1]$. Why is the plot hard to read? Plot the samples that lie in the interval $[0,0.01]$ instead (this is a 10 msec interval).

(e) The Matlab function `sound` (see `help sound`) with syntax

```
sound(sampledSignal, frequency)
```

sends the one-dimensional array or vector `sampledSignal` to the audio card in your PC. The second argument specifies the sampling frequency in Hertz. The values in `sampledSignal` are assumed to be real numbers in the range $[-1.0,1.0]$. Values outside this range are clipped to -1.0 or 1.0. Use this function to listen to the signal you created in the previous part. Listen to both a 10 msec interval and 2 second interval. Describe what you hear.

(f) Listen to

```
sound(0.5*sampledSignal,frequency)
```

and

```
sound(2*sampledSignal,frequency)
```

where `sampledSignal` is the signal you created in part (d) above. Explain in what way are these different from what you heard in the previous part. Listen to

```
sound(sampledSignal,frequency/2)
```

and

```
sound(sampledSignal,frequency*2)
```

Explain how these are different.

C.1.2 Independent section

1. Use Matlab to plot the following continuous-time functions $f:[-0.1,0.1] \rightarrow \textit{Reals}$:

$$\begin{array}{lll} \forall t \in [-0.1,0.1], & f(t) & = \sin(2\pi \times 100t) \\ \forall t \in [-0.1,0.1], & f(t) & = \exp(-10t)\sin(2\pi \times 100t) \\ \forall t \in [-0.1,0.1], & f(t) & = \exp(10t)\sin(2\pi \times 100t) \end{array}$$

The first of these is a familiar sinusoidal signal. The second is a sinusoidal signal with a decaying exponential envelope. The third is a sinusoidal signal with a growing exponential envelope. Choose a sampling period so that the plots closely resemble the continuous-time functions. Explain your choice of the sampling period. Use `subplot` to plot all three functions in one tiled figure. Include the figure in your lab report.

2. Use Matlab to listen to a one-second sinusoidal waveform scaled by a decaying exponential given by

$$\forall t \in [0,1], \quad f(t) = \exp(-5t)\sin(2\pi \times 440t).$$

Use a sample rate of 8,000 samples/second. Describe how this sound is different from sinusoidal sounds that you listened to in the in-lab section.

3. Construct a sound signal that consists of a sequence of half-second sinusoids with exponentially decaying envelopes, as in the previous part, but with a sequence of frequencies: 494, 440, 392, 440, 494, 494, and 494. Listen to the sound. Can you identify the tune? In your lab report, give the Matlab commands that produce the sound. When the next lab meets, play the sound for your instructor.

4. This exercise explores the relationship between Matlab functions and mathematical functions.

   (a) The `sound` function in Matlab returns no value, as you can see from the following:

   ```
   >> x = sound(n)
   ??? Error using ==> sound
   Too many output arguments.
   ```

   Nonetheless, `sound` can be viewed as a function, with its range being the set of sounds. Read the help information on the `sound` function carefully and give a precise characterization of it as a mathematical function (define its domain and range). You may assume that the elements of Matlab vectors are members of the set *Doubles*, double-precision floating-point numbers, and you may, for simplicity, consider only the one-argument version of the function, and model only monophonic (not stereo) sound.

   (b) Give a similar characterization of the `soundsc` Matlab function, again considering only the one-argument version and monophonic sound.

   (c) Give a similar characterization of the `plot` Matlab function, considering the one argument version with a vector argument.