Introduction to Digital Signals using MATLAB

I. Digital representation of audio signals

An audio signal can be represented digitally as a sequence of sample values, each corresponding to a voltage (or ultimately air pressure or velocity). These samples can be converted into a continuous voltage by using a device called a digital-to-analog converter (DAC). The voltages can then be converted to air pressure fluctuations by means of a loudspeaker, headphones, etc. Conversely, a continuous audio signal can be digitized using a device called an analog-to-digital converter (ADC). The digitized values can then be stored in the computer’s memory, on a storage device. It can then be played back through the use of a DAC. Analog-to-digital conversion and digital-to-analog conversion happen “behind the scenes” when you use a microphone to input sound to a computer, or play a sound out.

In a digitized signal, the time interval between each successive sample is referred to as the sampling period (T) of the system. The number of samples acquired (or output) in a second (i.e., 1/T) is referred to as the sampling rate (srate). Number of samples per second is usually referred to as Hertz (Hz). Common sampling rates for speech are 10000 Hz (sometimes written 10 KHz), 20000 Hz, and 22050 Hz. The standard sampling rate for music (CD) is 44100 Hz (the reason for this will be discussed later).

II. MATLAB and digital signals

MATLAB is a system that allows us to specify digital signals in a simple way, operate on them mathematically, plot them as time waveforms, calculate their spectra, and play them out over headphones or speakers. The system can be used interactively (you can just sit and type and the results of each operation will be displayed in sequence), or you can write “scripts” that are sequences of MATLAB statements, and constitute programs written in this language.

The basic unit of MATLAB is the matrix. A matrix is a set of numbers arranged into rows and columns. A matrix with a single row (or a single column) is called a vector. The power (and simplicity) of MATLAB stems from its treatment of matrices (and vectors) as primitives. A single number (“scalar”) is the limiting case of a matrix with a single row and a single column. In what we will be doing, we will make extensive use of vectors. A digital signal can be represented as a vector, (usually with a large number of elements) with the individual columns (or rows, depending on the orientation) of the vector corresponding to the successive signal samples.
III. Some basic MATLAB examples

In what follows lines beginning with » were typed by me. Other lines were typed back by MATLAB.

(1) To define a vector as a sequence of values, set it equal to the desired of sequence of values, enclosed in square brackets. For example, to define a vector x,

```plaintext
» X = [9 8 2 4 8 1 2 8 7]
X =
  9     8     2     4     8     1     2     8     7
```

(N.B. MATLAB is case sensitive, so x does not equal X).

(2) To plot the points of a vector in sequence across the page, use the `plot` command, with the name of the vector as an argument (in parentheses). If the only argument is the name of a single vector, the horizontal axis of the plot shows the number of each successive vector element, while the vertical axis shows the value of each element.

```plaintext
» plot (X)
```

![Graph of vector X](image_url)
(3) One a vector has been defined, it can be operated on by mathematical operators. For example, to take the vector \( \mathbf{X} \) and add 5 to every element in it:

\[
\text{»} \mathbf{X} = \mathbf{X} + 5
\]

\[
\mathbf{X} = \\
14 \quad 13 \quad 7 \quad 9 \quad 13 \quad 6 \quad 7 \quad 13 \quad 12
\]

\[
\text{»} \text{plot (X)}
\]

or to multiply the value of every element in the vector by .5:

\[
\text{»} \mathbf{X} = \mathbf{X} \ast .5
\]

\[
\mathbf{X} = \\
\text{Columns 1 through 7} \\
7.0000 \quad 6.5000 \quad 3.5000 \quad 4.5000 \quad 6.5000 \quad 3.0000 \quad 3.5000
\]

\[
\text{Columns 8 through 9} \\
6.5000 \quad 6.0000
\]
(4) A vector can also be filled with a progression of values from a starting value to an ending value. To do so, set the vector name equal to the `beginning_value : ending value`.

```matlab
» Z = 1:9
Z =
1  2  3  4  5  6  7  8  9
```

```matlab
» plot (Z)
```

(5) By default, the increment between beginning value and ending value is 1. However, this can be set to whatever you want by setting the vector equal to `beginning_value : increment : ending_value`.

```matlab
>> Z = 0:.1:.8

Z =

Columns 1 through 7
0    0.1000    0.2000    0.3000    0.4000    0.5000
0.6000

Columns 8 through 9
0.7000    0.8000

>> plot (Z)
```

(6) In the plots we have done so far, the successive points of the vector have been connected by a line. Sometimes it is useful to see just the successive points themselves, and not connect them with a line. To do this add an additional argument to the `plot` command, which shows symbol (in quotes) that you want to see represent each point (choices are `o`, `+`, `*`, `.`, `x`).

```matlab
>> plot (Z, 'o')
```
Another plot option is to give two vectors as arguments to the \texttt{plot} command. In this case, the x-axis values come from the first vector, while y-axis values come from the second. The two vectors must each have the same number of elements.

\texttt{>> plot(Z,X,'o')}
IV. Using MATLAB to generate the digital representation of a pure tone.

(1) Set the sampling rate.
(N.B. if you end a MATLAB line with a semicolon, then it will NOT respond by listing out all of the values of the vector that you are specifying. This is useful when we deal with fairly long vectors).

```matlab
sr = 10000;
```

(2) Fill a vector with the time values of the successive samples of a digital signal beginning at time 0, and going up to a tenth of a second. The increment will be the sampling period, or the reciprocal of the sampling rate.

```matlab
t = 0:1/sr:.1;
p = plot (t, 'o');
```

(3) Calculate the angular velocity in radians/sec (omega) that corresponds to a particular desired tone frequency. In this case, we choose to generate a 200 Hz tone. The corresponding angular velocity is then $2\pi$ times 200. pi is a built-in MATLAB variable whose value is an approximation to $\pi$.

```matlab
omega = 2*pi*200
omega =
1.2566e+03
```

(4) For each of the sample times (vector t), calculate the elapsed angle that would be traversed (theta) by that time (angular velocity times elapsed time).
\texttt{\textbackslash > theta = omega \times t;}

\texttt{\textbackslash > plot (theta)}

\texttt{\textbackslash > plot (t,theta)}
(5) To get the value of a pure tone signal (simple harmonic motion) at each sample point, take the sine of the elapsed angle:

```plaintext
»wave = cos (theta);
»plot (wave)
```

![Graph of wave](image1)

```plaintext
»soundsc (wave,srate)
Playing a mono buffer of length 1001 and sample rate 10000.
```

```plaintext
»plot (t,wave)
```

![Graph of plot](image2)
V. Complex Numbers

MATLAB makes it easy to represent and perform operations on complex numbers. $i$ and $j$ are both special characters that represent the square root of -1. Here are some simple examples.

(1) Define the argument ($\theta$) and magnitude ($\text{magnitude}$) of a complex number.

```
» theta = pi/6
theta =
    0.5236
» magnitude = 3
magnitude =
    3
```

(2) Define a complex number ($z_1$) using $\theta$ and $\text{magnitude}$. MATLAB returns its value in rectangular form.

```
» z1 = magnitude*exp(i*theta)
   z1 =
    2.5981 + 1.5000i
```

(3) The functions real and imag get the real and imaginary parts of a complex number.

```
» real(z1)
   ans =
    2.5981
» imag(z1)
   ans =
    1.5000
```

(4) We can also get the real ($x_1$) and imaginary ($y_1$) parts trigonometrically.

```
» x1 = magnitude*cos(theta)
   x1 =
    2.5981
y1 = magnitude*sin(theta)

y1 =
    1.5000

(5) Now define a complex number (z2) by specifying it real and imaginary parts.

» x2 = 5
x2 =
    5

» y2 = 5
y2 =
    5

» z2 = x2+i*y2
z2 =
    5.0000 + 5.0000i

(6) MATLAB functions abs and angle get the magnitude and argument (angle) of z2.

» abs (z2)
ans =
    7.0711

» angle (z2)
ans =
    0.7854

(7) Again, we can do the same thing trigonometrically.

» sqrt(x2^2 + y2^2)
ans =
    7.0711
>atan (y2/x2)

ans =

0.7854