SEAS Short Course on MATLAB
Tutorial 1: An Introduction to MATLAB

Peter J. Ramadge

9/12/2007 v3.3

1 Summary

MATLAB is an integrated numerical computing environment. The name, MATLAB, stands for *Matrix Laboratory*, a reflection of the programs origins. Indeed, one of the most distinctive feature of MATLAB is its integrated use of matrices and the operations of linear algebra. However, it has grown to be widely applicable in all aspects of scientific computing.

There are two objectives to this tutorial: (a) to learn basic MATLAB commands, and (b) to understand how to write and execute simple MATLAB programs to generate, manipulate, display and save data.

The tutorial cannot cover all aspects of MATLAB. We will concentrate on the basics and show how you can use the MATLAB help facility to learn more. Please read this material before coming to the first class meeting.

2 Preliminaries

- **Where on campus is MATLAB available?** MATLAB is available on all OIT computer clusters.
- **Starting MATLAB.** MATLAB can be started by double clicking on the MATLAB icon. Once MATLAB is started, you can type commands at the MATLAB prompt `>>`. Alternatively, you can write a program (called an M-file) containing MATLAB commands and type the name of the M-file at the MATLAB prompt. More on this later.
- **Ending MATLAB:** When you are finished, exit the MATLAB program by typing the command `exit` at the MATLAB prompt.
- **Working directory:** The command `pwd` will print the name of the current working directory and the command `dir` will list the contents of the working directory.
- **Changing the working directory:** When you first start MATLAB it is a good idea to change the working directory to a subdirectory within your home directory. This way any files that you create will be saved in your home directory structure. Clicking on the three horizontal dots in the upper right corner of the MATLAB window will allow you to select the working directory. If you need to create a new subdirectory, do this first using your operating system. Then use MATLAB to select it as the working directory.

- **Previous commands:** The up-arrow and down-arrow keys allow you to cycle through previously issued commands.

- **Help:** The command `help` followed by the name of any MATLAB command will provide on-line help for that command. For example, to learn about plotting type `help plot`. To learn more about the built-in elementary functions type `help elfun`. The command `help` will list all the help topics.

3 MATLAB Basics

3.1 Variables, assignment statements and mathematical operations

- **Basic data structure:** The primary data structure in MATLAB is a matrix and almost all commands operate on matrices. Scalars are just 1×1 matrices. Recall that a matrix is a rectangular array of numbers with each entry indexed by integer row and column numbers. A square matrix has the same number of columns as rows. A row vector is a matrix with just one row and a column vector is a matrix with just one column. Some examples:

$$A = \begin{pmatrix} 1 & 2 \\ 3.1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & 3.1 \\ 3.1 & 2 & 1 \end{pmatrix} \quad x = \begin{pmatrix} 1 & 2 & 3.1 \end{pmatrix} \quad y = \begin{pmatrix} 1 \\ 2 \\ 3.1 \end{pmatrix} \tag{1}$$

- **Scalar assignment statements:** A scalar variable is created and assigned a value in MATLAB with a simple statement of the form:

```
a=2.35;
b=2*a+pi*a^2;
```

  The first statement creates a variable called `a` and assigns it the real number value 2.35. A real number is represented in MATLAB in double precision floating point format. The second statement creates a variable `b` and assigns it the value computed using the formula on the right hand side of the statement. Only variables that have already been created can appear on the RHS of assignment statements. The variable `pi` is a predefined constant with value π.

  MATLAB variable names must begin with a letter. This can be followed by any combination of letters, digits, and underscores. MATLAB distinguishes between uppercase and lowercase characters; `A` and `a` are different variables.

- **The ending semicolon:** If you omit the semi-colon at the end of a MATLAB assignment command you get a echo response:

```
>> x=1.5

x =

    1.5000
```

This echo is useful for checking the value of a scalar variable or small matrix (just type in the variable name without a semicolon) but is undesirable most of the time.

- **short, long, bank format:** Notice that the value of `x` was only printed to five significant digits. This is called `short` output format. If you issue the command `format long` and then repeat the above, you get the output:

```
x =

   1.50000000000000
```

The command `format bank` will result in variable values being printed with only two decimal places and `format` without any argument will revert to the default format. Values are represented internally in double precision floating point format and the `format` command controls how they are displayed.

- **Scalar mathematical operations:** If `a` and `b` are scalar variables, then the standard mathematical operations are denoted as follows:

```
addition:           a+b
subtraction:        a-b
multiplication:     a*b
division:           a/b
exponentiation:     a^b
```

As usual, exponentiation takes precedence over multiplication and division which take precedence over addition and subtraction. In complex formulas it's a good idea to use parentheses for clarity:

```
a=(2+b^2)*(1-(2-b)^3);
```

- **Matrix assignment statements:** A matrix variable is defined with the following syntax:

```
A=[1 2; 3.1 0];
B=[1 2 3.1; 3.1 2 1];
x=[1 2 3.1];
y=[1;2;3.1];
```

These commands define the matrices in equation (1). The general pattern is to use the delimiters `[ ]` to indicate a matrix and within these delimiters, list each row separated by a semicolon. Elements of each row can be separated by either spaces or a comma.

- **Arithmetic progressions:** MATLAB has a simple syntax to define a row vector with entries that form an arithmetic progression:

```
m=[1:1:100];
t=[0:0.1:10];
```

  The first command creates a row vector `m` of size 1×100 by setting the first entry to be 1, the next entry $1 + 1 = 2$, then $2 + 1 = 3$, and so on up to the last multiple of 1 that is less than or equal to 100. The second statement creates a row vector `t` with first entry 0, second entry $0 + 0.1 = 0.1$, next entry $0.1 + 0.1 = 0.2$ and so on until the largest multiple of 0.1 that is less than or equal to 10. In this case the length of the row vector is 101. The general pattern for this command is: `x = [b:a:s];` where `b, a, s` are scalar variables giving the first value of the vector, the additive increment, and the stopping value respectively. The delimiters `[ ]` are optional since it is clear that a row vector is intended. In addition, when the increment is 1, it can be omitted. So it is common to write `m=1:100;` and `t=0:0.1:10;`.

- **Commonly occurring matrices:** There are also built in commands for creating commonly occurring matrices. Here are a few of the most useful:

```
u=zeros(10,10);
v=ones(10,20);
I=eye(10);
```

  The first two of these commands should be self explanatory. The third command creates an identity matrix of size 10×10.

- **Referencing matrix values:** The values in a matrix variable can be referenced in several ways. First, if `x` is a row or column vector of length N, then `x(j)` for $j = 1, \ldots, N$, refers to the value of the jth entry of `x`. This can be used in assignment statements and formulas thus:

```
x(1)=2*pi+1;
x(3)=x(1)/2;
```

  For a general matrix `A`, the elements of `A` can be referenced by using row and column indices:

```
A(1,1)=pi;
A(1,2)=x(1)/2 +A(1,1);
```

Note that MATLAB indices start at 1 and matrix elements are indexed by row number first and column number second.

So far this is all very standard. Now fasten your seat belt. Suppose `A` is a 3×4 matrix. Then the command `A([1 3],[2 3])` will select the submatrix of `A` formed by the intersection of the first and third rows with the second and third columns. For example:

```
>> A=[1 2 3 4; 5 6 7 8; 9 10 11 12];
>> B=A([1 3],[2 3])

B =

     2     3
    10    11
```

In general, `A(M,N)` is the matrix consisting of the intersection of the rows in the vector `M` and the columns in the vector `N` in the order in which the indices appear in `M` and `N`. A colon in place of a vector index indicates all values of the index. So `B=A(1:3,2:4)` is the submatrix of `A` formed by the intersection of the first three rows with columns two through four; `C=A(:,2)` is the second column of `A`; `D=A(2,:)` is the second row of `A` and `E=A([3 1 4 2],:)` permutes the rows of `A` - row three becomes row one, row one becomes row two, row two becomes row four and row four becomes row three.

- **The end of a matrix dimension:** The variable `end` when used to index a matrix dimension, refers to the largest index in that dimension. For example, if `x` is a row vector, then `x(end)` is the last entry in `x`, `x(end-2)` is the third last entry in `x`, and so on. If `A` is a matrix, then `A(j,end)` is the last entry on row `j` and `A(end,end)` is the right bottom most entry of the matrix.

- **Building matrices:** Suppose `A` is $m \times n$ and `B` is $m \times k$. Then `C=[A B];` is an $m \times (n+k)$ matrix formed by joining the rows of `B` to the end of the rows of `A`. Similarly, if `A` is $m \times n$ and `B` is $k \times n$, then `C=[A; B];` is an $(m+k) \times n$ matrix formed by joining the columns of `B` to the end of the columns of `A`. Here is an example:

```
x=ones(1,100); y=zeros(size(x));
u=[x y x];
v=[x;y;x];
```

  `u` is a row vector consisting of 100 ones followed by 100 zeros followed by 100 ones and `v` is 3×100 matrix with first row all ones, next row all zeros and last row all ones.

- **An Example:** A circulant matrix is a square matrix with the property that each column after the first, is a downwards circular rotation of the column that precedes it. For example:

$$C = \begin{pmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

Suppose we want to construct a circulant matrix given its first column. Let the first column be stored as the MATLAB column vector v. Then the first two columns of C are constructed by the code:

```
C=v;
v=v([end 1:end-1]);
C=[C v];
```

Repeating the last two commands a second time will add the third column to C. If we put the last two commands in a loop and go through this loop length(v)-1 times, then the final value of C will be the desired circulant matrix.

3.2 Mathematical operations with matrices

- **Adding and scaling matrices:** Matrices *of the same size* can be added: A+B denotes the matrix sum and A-B the matrix difference of A and B. If a is a scalar valued variable, the command a*A yields the matrix obtained by multiplying each entry of the matrix A by the value of a. Multiplication takes precedence over addition. So 3*A+B is computed as (3*A)+B.

- **Matrix multiplication:** Since the basic data structure in MATLAB is a matrix, multiplication of variables is by default matrix multiplication. This means that the variables (i.e., matrices) being multiplied must be *conformable*. If A is $m \times n$ and B is $n \times k$, then A*B is defined and yields the matrix product of the matrices A and B.

- **An example:** A theorem of linear algebra states that any square matrix satisfies its own characteristic polynomial. So if the $n \times n$ matrix A has characteristic polynomial $\lambda^n + a_1\lambda^{n-1} + a_2\lambda^{n-2} + \ldots, +a_{n-1}\lambda + a_n$, then

$$A^n + a_1 A^{n-1} + a_2 A^{n-2} + \cdots + a_{n-1} A + a_n I = 0$$

where I is the $n \times n$ identity matrix. We can use MATLAB to test this for a 3×3 matrix as follows:

```
A=[1 2 3;4 5 6;7 8 9];
a=poly(A);
PA= a(1)*A*A*A + a(2)*A*A + a(3)*A + a(4)*eye(3)
```

The first line defines the matrix we are going to test. The command poly(A) finds the coefficients of the characteristic polynomial of A and returns them in a row vector ordered from highest to lowest powers, i.e., $1, a_1, a_2, a_3$. The commands yield the following result:

```
PA =

  1.0e-011 *
```

```
    0.0312    0.0320    0.0313
    0.0639    0.0607    0.0853
    0.0838    0.1052    0.1260
```

The answer has very small entries (less than 10^{-11}) but it is not the zero matrix. This is a consequence of the floating point numerical representation and the associated computational roundoff error. More on this in a later tutorial.

- **Matrix transpose:** MATLAB denotes the transpose of matrix `A` by `A'`. If `x` is a row vector, then `x'` is a column vector and visa versa. Suppose `A` is a $n \times n$ matrix and `x, y` are column vectors of length n. Then:

  - `z=A*x` is a column vector formed by the matrix `A` acting on the vector `x`.
  - `z=x'*A` is a row vector.
  - `p=x'*y` is a scalar. It is the inner product (or dot product) of `x` and `y`.
  - `B=x*y'` is an $n \times n$ matrix called the outer product of `x` and `y`.

  Notice that transpose takes precedence over multiplication.

- **Element-by-element matrix multiplication, division and exponentiation:** There are many instances where instead of a matrix product you want an element-by-element product. As the name implies, the element-by-element product is a matrix of the same size as `A` and `B` with the $(j,k)^{th}$ entry the product of the $(j,k)^{th}$ entries of `A` and `B`. So if `C` stores the result, then for every (j,k): `C(j,k)=A(j,k)*B(j,k)`.

  MATLAB provides a built-in command `A.*B` for the element-by-element product. It is defined whenever `A` and `B` have the same size. It is worth reemphasizing that `A.*B` is not the same as `A*B`. Here is an example:

```
>> A=[1 2 ;3 4]; B=[1 0; 0 1];
>> C=A*B

C =

     1     2
     3     4

>> D=A.*B

D =

     1     0
     0     4
```

There are also commands for element-by-element division: `A./B`, and element-by-element exponentiation: `A.^s` where `s` is a scalar variable.

3.3 Loops and conditionals

- **Loops:** MATLAB provides two basic loop structures: `for` and `while`. A simple for loop on variable `k` is created with the command structure:

```
for k=1:1:N
    list of MATLAB statements
end
```

where `N` is an integer valued variable with value n. This will execute the list of MATLAB statements for $\texttt{k} = 1, 2, 3, ..., n$. More generally, if `J` is a row vector with length n, then the loop

```
for k=J
    list of MATLAB statements
end
```

will execute the list of MATLAB statements n-times for `k=J(1),J(2),...,J(n)`. Some examples:

```
X=[1 3 6 9 10 10.5];
for k=X            (loop 6 times for k=1,3,6,9,10,10.5)
for x=0:0.1:0.95   (loop 10 times for x=0,0.1,0.2,...,0.9)
for z=10:-2:-10    (loop 11 times for z=10,8,...,2,0,-2,...,-8,-10)
```

Caveat: if the loop variable will be used as an index into a matrix, it must be integer valued and ≥ 1.

- **Conditionals:** MATLAB provides two basic conditional execution commands `if` and `switch`. The `if` conditional structure takes the form:

```
if <logical value>
    list of MATLAB commands
end
```

The logical value is usually the result of a logical test such as `x==10`, `x>=2`, `log(x)+y > pi`, and so on. Use `help ==` to obtain more information on logical operators and tests. There is also a variation that includes an `else` option. Here are some examples:

```
if (x==2)&(y<3)
    z=x^2+y;
end
if (x+2*y+3*z>=1)
    phi=(1+sqrt(5))/2;
else
    phi=1.0;
end
```

The `switch` command provides an alternative flow control structure. To learn more, use `help switch`.

4 Built-in MATLAB Functions

MATLAB has many built-in functions that operate on variables. These can be roughly divided into two classes: (a) 'management' functions that save, load, clear, plot, display or return properties of variables and (b) mathematical functions that compute functions of the variable values.

- **Some useful management functions:**
  - **size and length:** For a variable `A`, the command `size(A)` returns the dimensions of `A` (number of rows, number of columns) as a 1×2 matrix. The command `length(A)` returns the largest of the two dimensions of `A`.
  - **who and whos:** If you issue the command `who`, MATLAB will respond with a list of all the variables you have defined. The command `whos` will list all the variables currently defined, their sizes, the number of elements and some other useful information.
  - **clear:** The command `clear` will clear ALL variables from memory, `clear x` will just clear the variable `x`.
  - **save and load:** The command `save filename x y A` saves the values of the variables `x`, `y` and `A` in a binary file named `filename.mat` in the working directory. The command `load filename` recreates the variables stored in `filename.mat`. Use `help load` and `help save` for more information.
  - **plot:** The command `plot(y)` plots the values of a row or column vector `y` versus its integer index (row or column number). `plot(x,y)` will plot the values of `y` against the corresponding values of `x`. More on plotting later.
- **Scalar mathematical functions:** A host of common scalar mathematical functions are provided as built-in MATLAB commands. These include trigonometric functions (`sin, cos, ...`), exponential functions (`exp, log, log10, log2, sqrt,...`) and rounding and remainder functions (`fix, round, mod, rem, floor, ceil,...`) Type `help`

`elfun` to see a full list of the built-in elementary (mathematical) functions. The functions are applied to a scalar variable `s` using the standard syntax:

```
a=sin(s);
b=exp(-2*s);
```

- **Matrix mathematical functions:** MATLAB also has many built-in matrix functions. Here are few interesting examples. For a general matrix `A`:
  - `sum(A)` is a row vector containing the sums of the columns of `A`.
  - `max(A)` (resp. `min(A)`) is a row vector containing the maximum (resp. minimum) of each column of `A`.
  - `diag(A)` is a row vector containing the main diagonal of `A`.
  - `eig(A)` is a row vector containing the eigenvalues of `A`.
  - `poly(A)` is a row vector containing the coefficients of the characteristic polynomial of `A`.

  For a row or column vector `x`:
  - `sum(x)` is the sum of the elements of `x`.
  - `max(x)` (resp. `min(x)`) is the maximum (resp. minimum) of the elements of `x`.
  - `norm(x)` is the Euclidean norm of `x` (i.e., $\sqrt{\sum_k x(k)^2}$).

  For matrices `A` and `B`:
  - `isequal(A,B)` tests if `A` and `B` are equal, returning 1 if true, 0 otherwise.

  These descriptions just scratch the surface. Use `help <functionname>` to learn more.

4.1 Scalar functions applied to matrices

- **Vectorized computations:** An important property of MATLAB is that it permits you to apply a scalar function to a matrix variable with the understanding that the function is to be applied to each element of the matrix. For example:

```
A=[pi -pi/2; pi/3 -2*pi/3];
B=sin(A);
```

`B` is the 2×2 matrix that results by taking the sine function of every element of `A`.

- **An Example:** Suppose we want to compute the signal

$$y(t) = 2e^{-2t}\sin(\pi t) + 3e^{-2t}\cos(\pi t) = e^{-2t}(2\sin(\pi t) + 3\cos(\pi t))$$

over the range $[0, 2]$ for t. To do so we can evaluate the signal at N uniformly spaced sample times in the interval $[0, 2]$. One obvious way to do this is to use a loop to compute the value of the function for each sample time and save the results as a row vector. Here is MATLAB code to do this:

```
T=2; N=1000; dt=T/N;
for j=1:1:N
    t=(j-1)*dt;
    y(j)=exp(-2*t)*(2*sin(pi*t) + 3*cos(pi*t));
end
```

However, MATLAB's matrix variables together with the convention for the application of scalar functions also allows us to *vectorize* the computation as follows:

```
T=2; N=1000; dt=T/N;
t=(0:1:N-1)*dt;
y=exp(-2*t).*(2*sin(pi*t) + 3*cos(pi*t));
```

The second line sets up a row vector of sample times with 1000 entries starting from 0 in steps of 2/1000. The next line computes the exponential and trigonometric parts of the function and takes their element-by-element product. As general rule of thumb, vectorized computations are efficient. That said, loops should be used if they aid the readability of complex code.

5 Writing a MATLAB Program: M-files

We are now ready to write some MATLAB programs. Writing a program, called an M-file, allows you to make changes, fix bugs, etc., in an efficient manner and gives you a textual record of the steps in your computation. There are two types of M-files: *scripts* and *functions*. A script is a program that operates on variables in the workspace and the variables that it creates or modifies remain in the workspace after the script has completed. A function M-file, on the other hand, is passed values and returns values but all variables used in the course of execution are erased when the function completes. A function M-file is thus analogous to a subroutine.

5.1 Mechanics of writing a program

- **Creating an M-file:** To write a M-file you use a text editor to create a file containing the desired MATLAB commands and then store the file as an ASCII file with a .m extension. MATLAB comes with its own built-in M-file editor which has some nice features that help you write correct code. It is highly recommended that you use this editor. You can start the MATLAB editor directly from the file menu tab in MATLAB (file/new/M-file).

- **Storage:** Your M-files must be stored in a location where MATLAB knows to look for them. By default, MATLAB always looks first in the working directory. So that's a natural place to store them. Caveat: make sure you have changed your working directory to a directory in your home directory structure.
- **Program Comments:** If MATLAB encounters a `%` symbol on any input line, it ignores the rest of that line. So `%` can be used to add comments to your program.

5.2 An example MATLAB script

Here is a simple script program that computes and plots the function $y(t) = e^{-8t}\sin(2\pi f t)$ for a given value of f and for t ranging over the interval $[0, 0.25]$ seconds.

```
% plotexpsin
% A MATLAB script to compute a decaying exponential signal
% Written by PJR 8/5/2005

clear           % clear memory

% Define constants
T=0.25;         % signal duration in seconds
Fs=8000;        % number of samples per second
t=0:1/Fs:T;     % time axis 0 to T in steps of 1/Fs seconds
f=75;           % sinusoid frequency in Hertz

% Compute the signal
x=exp(-8*t).*sin(2*pi*f*t);

% Plot the result
plot(t,x,'r');
grid;
xlabel('time-secs');
ylabel('signal value - volts');
title('A Plot of a Decaying Sinusoid');
```

A file is created in the working directory called `plotexpsin.m` that contains the above commands. Then the script name `plotexpsin` is entered at the MATLAB prompt. This executes the script and results in the plot shown in Figure 1.

5.3 An example MATLAB function

Here is a simple function M-file that computes the absolute value of the eigenvalue with largest magnitude of a symmetric matrix.

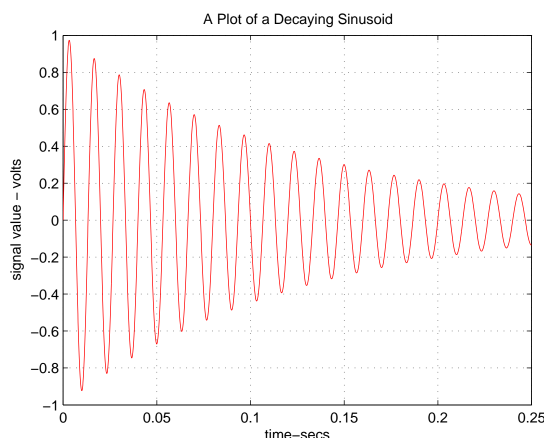

Figure 1: The plot produced by the script plotexpsin.

```
function lambda=findlargeig(A)
% FINDLARGEIG function
% findlargeig(A) computes the largest |eigenvalue| of symmetric matrix A
% Written by PJR 8/5/2005

if isequal(A,A')                        % is A symmetric?
    x=eig(A);                           % Yes, compute max absolute eigenvalue
    lambda=max(abs(x));
else
    error('Matrix is not symmetric')    % No, print error message
end
```

The function M-file begins with a special command that identifies the M-file as a function and indicates the input and output arguments:

```
function lambda=findlargeig(A)
```

`findlargeig` is the name of the function and the M-file should be stored as `findlargeig.m`. `A` is the input argument and `lambda` is the output argument. The remainder of the M-file contains standard MATLAB commands. Somewhere in the file a value should be assigned to the variable `lambda`. This ensures that a value will be returned. Other variables that are created or assigned values within the M-file are discarded when the function completes execution. In the above example, the vector variable `x` temporarily stores the eigenvalues of `A` but is discarded when the function completes execution.

You can call your function M-file directly from the MATLAB prompt or you can call it from within another MATLAB script or function M-file. You can think of your function M-file as adding a new function to those already available.

6 Plotting in MATLAB

The graphical facilities of MATLAB are one of its key conveniences. Here we just summarize the most basic commands.

6.1 Plotting the values of a row or column vector

- **simple x-y plot:** If `y` is a row or column vector, the command `plot(y)` plots the values of `y` versus its integer index. The graph can be augmented using the commands `grid`, `xlabel`, `ylabel`, `title` and `axis`.
- **Plotting variable `y` versus variable `x`:** `plot(x,y)` will plot the values of `y` against the corresponding values of `x`. Of course, `x` and `y` must be vectors of the same length.
- **Several graphs on one plot, changing line styles, etc:** The command

  ```
  plot(t,x,'r-',t,y,'b--');
  ```

  will plot the vectors `x` and `y` versus `t` on the same graph. The first vector will be plotted with a solid red line and the second signal will be plotted in a dashed blue line. Use `help plot` to learn more about color and line type controls.

  Other commands of interest include `figure`, `clf`, `hold`, `axis`, `text`, `subplot`, and so on. Use `help plot` to explore more.

7 Exercises

1. **Generating and plotting a signal.**

   The sinusoidal signal $x(t) = A\sin(2\pi ft + \phi)$ models a pure tone. Here A is the amplitude of the tone, f is the frequency in Hz, $2\pi f$ is the frequency in rad/sec, and ϕ is the phase. To represent this signal in MATLAB we compute the value of the signal at N equally spaced times over the time interval of interest. If the time interval is $[0, T]$, then the sample times are usually $t_k = kT/N$ for $k = 0, 1, \ldots, N-1$.

   Write an M-file script called `tone.m` to generate and plot the sinusoidal signal of frequency 262 Hz (middle C), amplitude 2 and zero phase over the interval $[0, 1]$. Use a sampling frequency of 16 KHz, i.e. $16,000$ samples each second. Your plot should display a grid, have the axes correctly labelled, and be given a title. Need better resolution? Try plotting the first M signal and time values (say, $M = 200$).

   A skeleton outline of your script should look like something like this:

   ```
   clear
   Fs=...;              % sampling frequency in Hz
   M=...;               % no. of samples to be plotted
   ```

```
T=...;              % duration of signal
fre=...;            % tone frequency Hz
amp=...;            % amplitude of tone
pha=...;            % phase of tone

w=2*pi*fre;       % tone frequency rad/sec
t= [...];         % vector of sample times
x= ...  ;         % the vector of signal values

plot(t,x)
...
...
```

2. **Generating sound.** MATLAB has a command for sending signals to the computer sound system. Check `help sound`. Note that the signal values need to be scaled so that they lie in the range $[-1, 1]$ otherwise they will be clipped.

   (a) Write a function M-file `rowscale.m` that takes as its input argument a row vector and returns the row vector scaled to the interval $[-1, 1]$. The commands `abs` and `max` may be useful for accomplishing this. An outline of your function will have the form:

   ```
   function y=rowscale(x)
   set y=x
   find the largest absolute value MV in y
   if MV is positive, divide each element of y by MV
   ```

   (b) Now modify your script `tone.m` to scale the signal generated to the range $[-1, 1]$ and play the signal through the computer sound system.

3. **Adding signals.** One the most fundamental operations on signals is superposition. This simply corresponds to combining signals by adding their values at each time. For example, a musical chord results when several (appropriate) notes are played concurrently.

   Copy your script `tone.m` to a new script `chord.m` and modify the `chord` script so that it will generate 3 single tone signals each with its own frequency, amplitude and phase. This is simple to do: just change the assignment statement for `amp`, `ph`, and `fre` to be vector assignments instead of scalars. You can generate the signals using a loop. Store the three signals in a single matrix. Then form their weighted sum. Your program should plot the sum of the signals (the first N=500 samples) versus time.

   To play your chord you first need to scale the signal values so that they lie in the range $[-1, 1]$. Your summed signal may not have this property even when the component signals do. Use a loop to play the three sinusoids in succession. Then play the summed signal. You cannot call `sound` again until the previous call completes. Pausing for the duration of the signal before preceding is a good idea. Check `help pause`. Try `amp=[1 1 1]`, `ph=[0 0 0]`, `f=[262 330 392]`. This is the C Major chord.

4. **A musical note function.** When a particular note is played on a musical instrument not only is the fundamental frequency (tone) generated but also higher harmonics of the fundamental, i.e., tones at the frequencies $2f, 3f, 4f, \ldots$. These harmonics give the instrument a richer sound. In addition, the signal has an envelope representing the fact that the note quickly rises in intensity and then gradually fades away. Mathematically we can represent this as $s(t) = e(t)h(t)$ where $h(t)$ is the signal containing the sum of the desired harmonics and $e(t)$ is the time varying envelope function. In general, the envelope function can be quite complicated but let's try something very simple: $e(t) = e^{-3t}, t \geq 0$.

   Write a MATLAB function that takes three scalar parameters, note frequency, duration and sampling frequency, as arguments and returns a row vector containing the note signal with six harmonics modulated by the note envelope. The signal returned should be normalized to the range $[-1, 1]$. Proceed as follows:

   (a) Copy your M-file `chord.m` into a new file `fnote.m` and modify `fnote.m` so that it is a function M-file with the opening line:

   ```
   function z=fnote(f,T,Fs)
   ```

   The function takes the fundamental frequency `f` in Hz, the note duration `T` in seconds, and the sampling frequency `Fs` in Hz as parameters and returns the sampled signal as a row vector `z` scaled so that its maximum absolute value is 1. Clean up your code to reflect this change and test your function by calling it from the MATLAB command line.

   (b) Now modify your function so that the signal generated is the sum of the tone at the specified frequency and its harmonics from 2 through 6. Allow for the possibility that the harmonics have different amplitudes and phases. Changing the amplitudes of the harmonics will change the character of the sound. You can test your new function by calling it with a simple script such as

   ```
   clear
   Fs=16000;
   fre=262; dur=1; amp=1;
   sig=amp*fnote(fre,dur,Fs);
   sound(sig,Fs)
   ```

   (c) Now modify your function `fnote.m` to include the note envelope. This will require an element-by-element matrix product.

   If you are ambitious you can write a simple script to call your fnote function to generate and sound a sequence of notes (a.k.a. music).

8 References

(1) Cleve Moler, Numerical Computing with MATLAB, SIAM, 2004.
See also `www.mathworks.com/moler/chapters` .