A SUMMARY OF BASIC OPERATIONS IN MATLAB

Electrical and Computer Engineering Department
Todd K. Moon
Version: 2.1

Introduction

This document provides an introduction to familiarize you with MATLAB. MATLAB is a widely available computing environment that allows programming, editing, plotting, and data manipulation. MATLAB provides excellent numeric capabilities. Symbolic processing is available as well. It is a full-blown computer language provides substantial capabilities and has become a standard tool for algorithm development in signal processing, controls, and many other areas of engineering.

This tutorial is merely an introduction to MATLAB. While this will not cover all of the features of MATLAB — a system so rich that it would take years to explore all of its possibilities — it provides a start so that you can begin the exploration on your own. A very valuable life skill for an engineer is the ability to acquire skill in new programming languages. This tutorial is written with this in mind.

In learning a new computer language, the most important thing is to use it. *Play around* with the language to learn how it works. Ask questions of the computer, by typing things in to see if they work. There are several built-in self-documentation features of the language that you should become familiar with. An important command to know is the `help` command. For any MATLAB command, such as `exp`, you can ask for information about it by typing

```
help exp
```

This provides a concise summary of the command. You can also get an HTML page of help using the `doc` command, as in

```
doc exp
```

In the tutorial that follows, you should type all of the indicated MATLAB examples into MATLAB, and make sure you can interpret and understand its results. There are also some exercises which you should do, again making sure you understand the results. When an exercise says "Study `help funct`," if the help provides some sample MATLAB code, you should run at least one of the examples to see what it does (for example, cut and paste).

You should turn in (as a programming assignment) a file containing the results of all numbered exercises. Please edit your file so that only the exercise results are included (not what is printed by the `help` command) and the file is easy to read and follow. To save your input/output interaction to a file, the `diary` command can be used. To save your results into a file called `filename`, use the command

```
diary filename
```

(You can learn more about this command with `help diary`.)

For plots, please either screenshot the plots or use the exporting functionality of MATLAB to export the plots and embed them into a file for upload.

Entering Data as Scalars, Vectors, and Matrices

MATLAB is primarily centered around matrices and vectors. These are entered delimited with square brackets. For example, to enter the vector

$$v = [1, 2, 3, 4]$$

you would type

```
v = [1,2,3,4]
```

or

```
v = [1 2 3 4]
```

That is, either commas or spaces can separate the elements. MATLAB distinguishes between column vectors and row vectors. To type the column vector

$$v = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

you would type

```
v = [1; 2; 3; 4]
```

or

```
v = [1
2
3
4]
```

Either the semicolon or a return separates rows.

In general, any time you want to see the value of a variable, simply type its name. So to see the vector `v` that you just typed in, you could type its name in MATLAB:

```
v
```

and MATLAB will print it. MATLAB uses a special format to display wide matrices and vectors.

To access elements of a vector, the index of the element you want is placed between parentheses. So `v(1)` is the first element of the array, `v(2)` is the last element. Conveniently, `v(end)` can be used to index the last element, `v(end-1)` can be used to index the penultimate element, and so forth.

The matrix

$$m = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 2 \\ 7 & 5 & 3 \end{bmatrix}$$

may be typed in either by

```
m = [1 2 3; 6 5 2; 7 5 3]
```

or

```
m = [1 2 3
6 5 2
7 5 3]
```

That is, either semicolons or returns may separate rows of the matrix.

Elements of a matrix can be accessed using parentheses with two numbers, `(row,column)`. So `m(1,1)` accesses the first row and the first column (element 1 in this example), `m(1,3)` is the first row third column (element 3 in this example. `m(end,end)` is 3.

Not everything in MATLAB has to be a matrix or vector. You can have scalar values as well. For example, you could enter

```
q = 1.602e-19
a = 6.023e23
mypi = 3.1415
```

as a few floating point numbers in scientific notation, or

```
numstates = 50
year = 2014
```

as a couple of integers.

The results of the most recent operation are displayed after you press Enter. If you do not want the results displayed, you can terminate the line with a semicolon. For example,

```
j2 = [5 6 7 8];
```

assigns the vector of numbers to j2, but does not display the result, because the line is terminated with a semicolon. For long computations, this can save a lot of screen space. You should also note that when the results of a computation are not assigned to any variable, the result is automatically assigned to the variable `ans`, which you can then use in the next computation. (Note that ans may be overwritten after any computation, so you should be careful in its use.)

The transpose operator is the single quote `'`. This converts column vectors to row vectors, and vice versa. If you enter the matrix `m` from above

```
m = [1 2 3
6 5 2
7 5 3]
```

then you can determine the transpose as

```
m'
```

which produces

```
ans =
     1     6     7
     2     5     5
     3     2     3
```

On variable names: Variable names in MATLAB generally follow the conventions used in other programming languages. You cannot begin a variable name with an integer (for example `1two` would not work, but `one2` would work). You can use upper and lower case letters, and also the underscore symbol _ (so `one_two` or `four_score_and_7` would be valid variable names).

Efficient keyboard interaction

At this point you should have typed enough using MATLAB that you may appreciate some efficiencies in the interaction. MATLAB allows you to use previously-typed results. This is helpful because often you want to do something again, with only minor modifications.

When you type the up arrow key ↑, the last line you typed appears. As you keep typing the up arrow, MATLAB will scroll through earlier lines you typed. Once a line appears, you can move around within it using the arrow keys to position the cursor, where you can edit the line in the usual way.

If you type the first letter (or more letters) of previous lines, and then press the up arrow, MATLAB will search through previously typed lines to find lines that match what you have typed in.

As you become familiar with these keyboard editing features, you will save a lot of time in interacting with MATLAB.

We now return to our discussion of entering data in MATLAB.

The colon : operator

It is helpful sometimes to be able to deal with a whole range of elements in a vector or matrix at once. Suppose you want to create a vector containing the numbers $[0, 1, 2, \ldots, 10]$. You could either type

```
j = [1,2,3,4,5,6,7,8,9,10];
```

(tedious), or you could use the colon operator. The colon specified a range (begin):(end). Thus, you could enter this range as

```
j = 1:10
```

Note that this makes a *row vector*. If you want to enter a column, you could use the transpose operator:

$$j = (1:10)'$$

Here, the `'` indicates the matrix transpose, which turns columns to rows and vice versa.

Note: Actually, `'` is the *conjugate transpose*, so that complex numbers are conjugated. If you want just the transpose without the conjugate, you should use the `.'` operator (dot-transpose). Most of the time, you can just use the `'` operator (because most of the time your numbers will be real, or most of the time you want the conjugate transpose).

If you want to use non-integer steps, you can specify a range using the colon operator in the form (begin):(step):(end). For example, the range `0:0.2:3` specifies a list starting at 0, ending at 3, and proceeding in steps of 0.2:

```
j2 = 0:0.2:3
```

produces the row vector with values

$$0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2, 2.2, 2.4, 2.6, 2.8, 3.$$

Exercise 1 Study `help colon`.

You can use the colon operator to access a range of values. For example, if you type `m(1,:)`, this returns the entire first row of the matrix `m`. If you type `m(:,1)`, this returns the entire first column. The form `m(1,1:2:end)` returns every other element of the first row.

Exercise 2 Create a 6×6 matrix `M` of random numbers, as `M = rand(6,6)`. Explore the following commands and make sure you understand why it does what it does.

```
M(1:2:end,1)
M(2:2:end,2)
M(1:2:end,:)
M(1:2:end,1:2:end)
M(1:2:end,2:2:end)
```

Other ways of entering vectors and matrices

There are other helpful ways of entering vectors and matrices.

- The command `zeros(m,n)` produces a matrix with `m` rows and `n` columns of all zeros. This is often a useful way of initializing a matrix or vector, or allocating space. For example,

```
z = zeros(3,4);
```

Exercise 3 Study `help zeros`

- The command `ones(m,n)` produces a matrix with `m` and `n` columns of all ones. This can be multiplied by a constant to set another value.

  ```
  f = 7 * ones(5,6);
  ```

  produces a 5×6 matrix whose elements are all set to 7.

  Exercise 4 Study `help ones`

- The command `rand(m,n)` produces a $m \times n$ matrix of independent random numbers, uniformly drawn from $[0, 1]$.

  Exercise 5 Study `help rand`

- The command `randn(m,n)` produces a $m \times n$ matrix of independent random numbers, drawn from a Gaussian (or normal) distribution with mean 0 and variance 1.

  Exercise 6 Study `help randn`

MATLAB is as adept at complex number operations as it is at real number operations. Complex numbers may be entered as `1-2i` or `1-2j` — either `i` or `j` may be used as $\sqrt{-1}$ in MATLAB. Here, `i` and `j` are actually variables in MATLAB so you can assign them to other values. You should be careful to not assign other values and then use them as if they were $\sqrt{-1}$. You can reset their default nature using one of the following methods:

```
clear i j       % clear assigned values (revert to default values)
i = sqrt(-1);   % assign what you want.
```

Note that MATLAB has no trouble computing $\sqrt{-1}$.

In these examples, the text after the percent % is *comments* — used to explain to human readers what is going on. The computer ignores the comments.

MATLAB has a variable with another useful value. The number π is represented using the variable `pi`. If you type `pi`, MATLAB responds with

```
ans =
    3.1416
```

While it is only showing only four decimal places of accuracy, internally numbers are represented with double precision accuracy. If you want to show more decimal places of accuracy, use the command

```
format long
```

Note that `pi`, like `i` and `j` is a variable. You can overwrite its value with anything you like, such as

```
pi = 3;  % probably not a good idea
```

Exercise 7 Study `help format`.

Computations

Arithmetic operations on real or complex numbers are typed as is typical, as shown in the following examples.

```
x = 3;
y = 7;
z1 = x*y;       % multiplication
z2 = x/y;       % division (real number)
z3 = x + y;     % addition
z4 = mod(7,3); % 7 mod 3 (remainder)

r=3;
vol = 4/3*pi*r^3;  % pi is built-in
```

MATLAB can also perform computations on vectors and matrices as a whole, as shown in the following examples.

```
v1 = [10 11 12 13];
v2 = [45 23 12 10];
vsum = v1 + v2;      % add the vectors

vsum2 = 5 + v1;      % add 5 to each element of v1
m = [1 2 3; 6 5 2; 7 5 3];
v3 = [20; 19; 18];   % a column vector
mv = m*v3;           % matrix times vector
vm = v1*m;           % vector times matrix
```

Mathematically, the operations of multiplication between the two row vectors `v1` and `v2` is not defined. But it may be convenient to multiply element-by-element. This is done using the `.*` operator.

```
vprod = v1 .* v2   % element-by-element multiplication
```

The ^ is the exponentiation operator. As applied to a scalar, it just computes a power:

```
j1 = 4;
e = 1/3;
j1sqrt = j1^e;  % compute the cube root of j1
```

If you want to exponentiate each element of a vector, you should use the `.^` operator, which applies the operator to each element separately.

```
j1 = 1:5;
j2 = j1 .^ 2;  % square each element of j1
```

Another example:

```
j3=1:5;
j4 = (3/2).^j;     % raise 3/2 to a series of different powers
```

MATLAB also has a rich set of mathematical functions that apply their operation across all the elements.

```
x = 2;
y = sin(x);       % compute the sin of a scalar
xv = [2:0.1:3];   % form a vector
yv = sin(xv);     % compute the sign of each element of xv
zv = exp(xv);     % compute exp of each element of xv
lv = log(xv);     % compute the log of each element of xv
```

Exercise 8 Study `help ops` and `help power` and `help mtimes` and `help times`. Make sure you understand difference between operators and their "dotted" versions, such as `/` and `./`.

Operations on matrices are very easy to do, since this forms the heart of MATLAB. The following exercises introduce some basic operations:

Exercise 9 Enter the following matrix (see page 1):

$$m = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 2 \\ 7 & 5 & 3 \end{bmatrix}$$

Find the eigenvalues and eigenvectors of m. (Study `help eig`.) Make sure you know which are eigenvalues and which are eigenvectors.

Exercise 10 Find the determinant and inverse of m. (Study `help det` and `help inv`).

Exercise 11 Solve the set of equations

$$\begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 2 \\ 7 & 5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$$

by entering

```
b = [4;2;1]
x = m\b
```

(See the operations $\backslash$ and `/` under `help slash`.)

Exercise 12 Compute

$$\frac{-\sqrt{16}-4}{2e^4+1}$$

by entering

```
e = exp(1)
(-sqrt(16)-4)/(2*e^4 + 1)
```

Note that e (the base of natural logarithms) is not a built-in number.

MATLAB also has some convenient functions for dealing with vectors. The `sum` command sums all of the elements in a vector.

Exercise 13 Compute $\sum_{k=1}^{20} k^2$. Enter

```
k=(1:20);
k = k.^2;
sum(k)
```

Related useful commands are `prod`, `cumsum`, and `diff`. Take a look at these (study their `help`).

Making plots

MATLAB has excellent plotting capability — perhaps the best of any computational tool around. In this exercise you will meet only the barest essentials of the plotting capability. For more information, see the `help` on the computer or the online manual.

Exercise 14 Study `help plot`

Here are some examples:

- Plot $\sin(2\pi j/50)$ (on the x-axis) versus $\cos(6\pi j/50)$ (on the y-axis) for $j = 1..51$:

  ```
  j=(1:51);              % make the index list
  x = sin(2*pi*j/50);    % x values
  y = cos(6*pi*j/50);    % y values
  plot(x,y)              % plot x vs. y
  ```

  Note that the statement `x = sin(2*pi*j/50)` produces an array at the output, with each component in the output array coming from a component in the input array.

- Plot $j = 1..51$ (on the x-axis) versus $\sin(2\pi j/50)$ (on the y-axis):

  ```
  plot(j,x)
  ```

- Plot $\sin(x)$ for $x \in [0, \pi]$.

  ```
  x = 0:0.1:pi;
  plot(x,sin(x));
  ```

- Frequently, it is important to plot more than one graph on a single pair of axes. This can be done at least two ways.

  In the first way, you simply provide a list of all the x, y data sets in a list.

  Plot the functions $y = x^2$ for $-2 \leq x \leq x$. Also, on the same axes, plot the function $z = x^3$ for $-3 \leq x \leq x$:

  ```
  x1 = -2:.01:2;    % set up the x range for the first plot; use .01 spacing
  y = x1.^2;        % set up the function y = x^2
  x2 = -3:.01:3;    % set up x range for second plot
  z = x2.^3;        % set up the function z = x^3
  plot(x1,y,x2,z); % plot both sets of data simultaneously
  ```

  If you want to control the plot color, marker, or the line style you can, specifying each individually. The colors are specified using letters. The following comes from `help plot`.

  ```
     COLORS               MARKERS                LINE STYLES
  b      blue          .      point            -      solid
  g      green         o      circle           :      dotted
  r      red           x      x-mark           -.     dashdot
  c      cyan          +      plus             --     dashed
  m      magenta       *      star
  ```

y	yellow	s	square
k	black	d	diamond
		v	triangle (down)
		^	triangle (up)
		<	triangle (left)
		>	triangle (right)
		p	pentagram
		h	hexagram

For example, redo the last plot, plotting the first line using a red dotted line, and the second line using a cyan dashed line:

```
plot(x1,y,'r:',x2,z,'c--');  % 'r:' means red dotted.
                             % 'c--' means cyan dashed
```

- Typically, every time you make a plot, MATLAB erases the old plot and puts on the new plot. However, you can tell MATLAB to *hold* the plot, so that new information can be overlaid, using the `hold` command. This provides another way to put more than one graph on a plot.

  Exercise 15 Study `help hold`

  To make the hold go away, you can use the command `hold off`. To turn the hold on, use `hold on`. To clear a graph, use `clf`.

  Repeat the previous plot, but put the plots on one at a time:

```
clf;    % clear the previous plot
plot(x1,y,'r:');   % plot the first function
hold on;           % turn on the hold
plot(x2,z,'c--');  % plot the second function.  They should both be there.
```

- Frequently it is useful to put more than one plot together. This can be done using the `subplot` command.

  Exercise 16 Study `help subplot`

  You invoke `subplot` with 3 arguments. The first two tell how many rows and columns of plots you want. The third column tells which subplot to plot into, numbered in row-dominant order.

- Plot the two functions into two separate plots, but in the same window:

```
clf;    % clear previous plot, if any
subplot(2,1,1);   % tell it you want 2 rows, 1 column, and the 1st plot
plot(x1,y,'r:');  % plot the first function
subplot(2,1,2);   % tell it you want 2 rows, 1 column and the 2nd plot
plot(x2,z,'c--'); % plot the second function
```

- You may want to have several plots on the screen at a time. This is done with the `figure` command. Study `help figure`.

You should know that it is possible to change just about everything in the plot. Several commands control the plots, or do plots of different kinds. These are explored in the following exercises.

Exercise 17 Study `help xlabel` (set the x-axis label on the plot)

Exercise 18 Study `help ylabel` (set the y-axis label on the plot)

Exercise 19 Study `help title` (set the title at the top of the plot)

Exercise 20 Study `help stem` (make a "stem" plot, which plots discrete time functions

Exercise 21 Study `help semilogy` (make a logarithmic plot on the y axis)

Exercise 22 Study `help semilogx` (make a logarithmic plot on the x axis)

Exercise 23 Study `help loglog` (make a log log plot)

Exercise 24 Study `help text` (put some text on a plot)

You should also play around with the icons on the plot screen. It is possible to zoom in on a plot and move around with the plot.

Exercise 25 Study `print` (save a plot to a file)

In addition to these commands, you can set any property of any line, point, text, or axis in a plot using the `get` and `set` commands. Take a look (but don't worry about internalizing) the `help get` and `help set`. Also, plots can be annotated and edited using the graphical user interface on the plot.

There is also a plot command that you can use for making plots of functions very easily, called `ezplot`. For example, you can simply type

```
ezplot('sin(x)')
```

Note that the function to be plotted is enclosed in quotes.

Exercise 26 Study `help ezplot`.

Getting help

MATLAB has an extensive help system built in. First, as seen many times in this document, there is the `help` command. For any of the commands, you may simply type `help commandname`. For example, type `help eig` to learn about how to compute eigenvectors and eigenvalues in MATLAB. You can even type `help help` to learn more about `help`. There is also an online manual. To access this, use the **Help** menu at the top of the MATLAB menu.

Exercise 27 Study `help help`.

You can learn what variables MATLAB has currently defined using the `who` command. If you want a more detailed list, including sizes of array variables and their types, use the `whos` command.

MATLAB is so extensive that there is reason to hope that just about any function you might want has probably already been implemented. But how to find it? MATLAB provides the `lookfor` command. This command searches through the MATLAB program files for all of the commands, looking at the comments in the program for a match.

Exercise 28 You are looking for a way to compute the binomial coefficient

$$\binom{n}{k}$$

You assume that MATLAB would have an implementation. See if you can find it using `lookfor binomial`.

As you are learning the language, you may find `lookfor` to be most powerful ally.

It may happen that you locate a function, and desire to know what directory it comes from. This can be found using `help which`.

The `doc` facility in MATLAB provides many, many pages of tutorial and example. Simply type

```
doc
```

It will bring up a menu. If you want to learn more about MATLAB itself, click under the MATLAB menu option, and go at it. You may enjoy examples of plotting, or examples of programming.

There are very many websites which provide MATLAB tutorials. Here are some places to start:

```
http://www.nt.uni-saarland.de/fileadmin/file_uploads/lectures/tc2/
unprotectedFiles/MatLabIntro.pdf

http://users.ece.gatech.edu/~bonnie/book/

https://www.mathworks.com/academia/student_center/tutorials/
signal-processing-tutorial-launchpad.html?confirmation_page#

http://www.mathworks.com/help/index.html
```

As a general rule, there is just all kinds of stuff at

```
http://www.mathworks.com
```

Writing programs in MATLAB

Besides a calculator of considerable value and elegance, MATLAB is also a programming language, with (almost) all of the programming constructs your heart could desire, such as `if-else`, `for`, `while`. If you have experience programming another language, you will find you can readily adapt to MATLAB's logical syntax.

One way to program is to simply type MATLAB commands into a file, just the way you would type them into the MATLAB working environment. This allows you to perform complicated computations and keep track of what you have done. You can edit your file if you find that you have typed something incorrectly, or run it multiple times. Such a way of programming is often called *scripting*.

All program files in MATLAB need to have the filename extension `.m`. For example, you could create a file named `myfile.m`. This becomes a new command that you can run in MATLAB, just like any of the other built-in commands in MATLAB. (You need to make sure that MATLAB knows where to find the file. That is, the file must be in the directory that MATLAB is working in, or in MATLAB's search path. See `help path`.) You can change directory using `cd`, and you can learn which directory MATLAB is in using `pwd`.

Exercise 29 Using a text editor, create a file called `myfile.m`. If you do not have have a text editor you call your own, you can always use MATLAB's built-in editor. You can do this by

```
edit myfile.m
```

This brings up an editor in its own window.

In that editor, type the following commands:

```
m = [1 2 3; 6 5 2; 7 5 3];
[u,v] = eig(m);
r = [1:.2:5];
vol = 4/3*pi*r.^3;
fprintf('Done!\n');
```

Save the file in a directory where MATLAB is working. You can see what is MATLAB's current directory with the `pwd` command.

Now, in MATLAB, type the name `myfile`. Verify that it has done what you expected.

Another way to program is to create new *functions*. Functions encapsulate some operation that you want to perform. Often they have input arguments and output arguments — things that go into the function, and things that come out of the function. Functions are also stored in files whose name has the extension `.m`.

As an example, the following is a simple recursive function that computes the factorial of a function

```
function f = fact(n)
% compute the factorial of a number recursively

if n==1
   f = 1;
else
   f = n*fact(n-1);
end
```

In this, you'll notice that a function is defined using the MATLAB keyword `function`. The first line

```
function f = fact(n)
```

creates a function called `fact`. It accepts a single argument called `n`, and has a single return value called `f`. After the function definition line, there is a comment that describes what the function does,

```
% compute the factorial of a number recursively
```

It is a highly recommended practice to put such a comment in. Not only does it help the reader understand what the program does, but it is this comment that is used by the `help` command.

To create this function, you would use a text editor to type the function in, in the directory where you are working.

Exercise 30

1. Enter the program into a file using your favorite text editor. If you do not have have a text editor you call your own, you can always use MATLAB's built-in editor. You can do this by

   ```
   edit fact.m
   ```

   This brings up an editor in its own window.

   Save the file into a file called `fact.m` in a directory. The name of the file *is* important – the name of the file should match the name of the function, and the filename should have the extension `.m`.

   (If you use MATLAB's editor, the default save directory will be the directory MATLAB is currently operating from.)

2. Make sure the file is in your MATLAB path (see `help path`), or in the directory that MATLAB is currently operating in. (You can find the current directory MATLAB is working in with the command `pwd`.

3. Compute the factorial of various numbers such as 4,5, and 10:

   ```
   fact(4)
   fact(5)
   fact(10)
   ```

4. Study `help if`

The factorial program could also be written using a `for` loop.

```
function f = fact2(n)
% return the factorial of a number using a for loop

f = 1;
for i=1:n
   f = i*f;
end;
```

The operation of this function may be easier to understand than the recursive function above.

The key operation here is the `for` statement. As in many other languages, the `for` statement successively assigns the variable 1, then 2, etc., up to 10. That is, it is as if we typed

```
f = 1;
i = 1;    f = i*f;
i = 2;    f = i*f;
i = 3;    f = i*f;
...
i = 10;   f = i*f;
```

The beauty of this, of course, is that you don't have to type a whole bunch of lines over and over. And you can change a variable, and change the number of times the line is repeated. (A third, much more efficient way of doing the factorial function, is also described below. It does not use an explicity `for` loop, and so it much faster.)

Exercise 31

1. Type this program using a text editor and save it in a file called `fact2.m`.

2. Test its operation on some numbers.

3. Study `help for`.

4. Another statement used for iteration is the while command. Study `help while`. (For those familiar with other programming languages, it might be interesting to note that there is nothing analogous to the `do/while` statement in MATLAB.

Exercise 32 Write a program to convert a base-10 number to binary. Return the binary number in an array. Call your function `dec2bin`. Thus, if you were to invoke

```
bina = dec2bin(37)
```

then `bina` should be `bina = [1 0 0 1 0 1]` (the LSB is on the right).

The following MATLAB operations may be useful to you: `rem, floor, for`.

Test your function on a variety of numbers.

MATLAB has another way of writing simple functions that you may find useful. This is using *anonymous functions*. Suppose you want to use the function $f(x) = \sin(x)/x$. You could, of course, create a function in a file that computes that function. Another way is to create one on the fly, inside MATLAB (or inside a MATLAB program). Suppose you want to call this function `f` (as in $f(x)$). You could enter

```
f = @(x) sin(x)./x;
```

The ampersand `@` tells MATLAB you are making a function (more literally, a function handle; you can read more about `@` under `help punct` and `help function_handle`). Having defined the function, you can use it:

```
f(1)
ans = 0.8415
f(2)
ans = 0.4546
x = 0.01:0.01:20;  plot(x,f(x))
```

Using these anonymous functions, we can provide yet another way of computing the factorial:

```
% Define the function (a nice 1-liner)
fact = @(n) prod([1:n])

% Then test it:
fact(4)
fact(100)
```

Elements of MATLAB style: Running fast

While MATLAB has full programming support for things like loops, conditionals, etc., it is often best to avoid explicit loops, using the capability that MATLAB has to deal with entire vectors all at once.

For example, suppose that we have two vectors `x` and `y`, and we want to compute the sum

$$s = \sum_{i=1}^{n} x(i)y(i)$$

This can be programmed using a `for` loop as follows:

```
n = length(x);  % how long is the vector?
s = 0;
for i=1:n
   s = s + x(i)*y(i);
end
```

However, this can be done using a single line:

$$s = x' * y;$$

(This assumes that `x` and `y` are column vectors.) As a general rule, working with entire vectors or matrices at a time, instead of step-by-step using `for` loops, operates considerable faster. Colon operators are often helpful for picking out entire rows or columns at a time to work with.

Moving on

MATLAB has a huge variety of tools that are useful for engineers. We touch on a few here.

MATLAB can print things to the screen, under your control. A common way to do this is using `disp`, or `fprintf`. Actually, `fprintf` can also write to files, so it is a nice command to know about.

MATLAB has full ability to read and write files and to save information. When dealing with files, typically you *open* the file using the `fopen` command, then write to the file using commands like `fprintf` or `fwrite`. To read from an file, use the command `fread`. After you are doing reading or writing to a file, close the file with `fclose`.

Exercise 33 Look at `help fprintf`, `help fopen`, `help fread`, `help fwrite`, and `help fclose`.

For just saving MATLAB variables to a file (for example to come back to later) then reading the variables in, the `save` and `load` commands are handy.

Exercise 34 Read `help save` and `help load`.

MATLAB can display matrices as images. As a simple example (which is a kind of useless image) type

```
M = rand(20,20);    % make a 20x20 random image
imagesc(M);         % display the image
```

`imagesc` scales the image so that the pixels are in a reasonable range. If you want to view without automatic scaling, use the `image` command.

MATLAB can play audio data. One way to get this data is to produce it inside MATLAB.

```
Fs = 4000;     % set the sampling frequency
Ts = 1/Fs;     % the sampling period
t = 0:Ts:5;    % 5 seconds of data
y = sin(2*pi*400*t);  % a pure sin at 400 Hertz
sound(y,Fs);  % play the sound
```

Another way to get sound into MATLAB is to read it from a file. MATLAB can also deal common audio formats. A common audio format is the `.wav` format. The commands `audioread` and `audiowrite` read and write audio files.

```
audiowrite('mysin400.wav',y,Fs)
```

produces a file name `mysin400`, which you should be able to play, for example, with a media player.

```
clear y    % get rid of the old y
y = audioread('mysin400.wav');   % read in the audio data again
sound(y,Fs);   % play the sound
```

This reads the data back in.

MATLAB can also solve for roots of polynomials. Enter the polynomial coefficients into a vector, then use the `roots` function. For example, to find the roots of

$$x^3 - 10x + 2,$$

form the vector

```
v = [1,0,-10,2]
```

(note that the coefficients go in *decreasing* powers of x, and that all coefficients, even if zero, must be included). Then use the function `roots(v)`.

```
rts = roots(v);
```

MATLAB also has a function that goes from the roots of a polynomial back to the polynomial. The function is called `poly`.

```
v1 = poly(rts)
```

This representation of the polynomial is (or should be very close to) the same as the original vector `v`.

MATLAB can also do numerical integration. For example, to compute

$$\int_0^5 \sin(t)dt,$$

enter

```
quad('sin',0,5)
```

The command `quad` is short for quadrature, which is the numerical analyst's word for integration.

Exercise 35 Study `help quad`

Due to the way the `quad function` is set up, the integrand must be a function that returns a vector of values for a vector of inputs. Practically what this means is that for most interesting problems you would have to create your own function. Thus, to integrate

$$\int_0^1 t\sin(2\pi t)dt$$

you would have to create your own function to compute $t\sin(2\pi t)$. This is not hard, but would take us too far afield on this introduction.

Symbolic computations

You may have noticed by now that all the computations return *numbers* as their answers. Numbers are good, but they are not all there is to mathematics. Your calculator could have done as well. Symbolic operations may also be done, however, using MATLAB. You can learn about symbolic functions by typing `help symbolic`, and by reading the online manual.

To get started, you need to tell MATLAB which variables are to be considered as "symbolic," that is, not taking on numeric values. This is done with the `syms` command. For example, to make the variables `n` and `k` be symbolic, use

```
syms k n;
```

1. To compute

   $$\sum_{k=1}^{n} k^2$$

   enter

   ```
   syms k n;
   s1 = symsum(k^2,k,0,n)
   ```

Now factor the result to put it in more conventional form

```
s2 = expand(s1)
```

(You may also find the `factor` command useful.) Now to find a particular numeric value, substitute a number in place of `n`.

```
subs(s2,n,20)
```

Exercise 36 Compare this result with the one obtained numerically previously.

Others symbolic operations that may be of interest are `simplify, factor, expand, collect, numden, subs, solve, fourier, laplace, ztrans, diff, int`. You can use the `help` function to learn more about these.

2. To compute the integral

$$\int_0^T t\cos(2\pi n t)dt$$

enter

```
syms t T n;
i1 = int(t*cos(2*pi*n*t),t,0,T)
```

This may be simplified by using

```
simplify(i1)
```

Exercise 37 Verify by integrating by hand that MATLAB has done its job correctly.

3. Try a derivative. To take the derivative of

$$e = \frac{2x^2 - 3x + 1}{x^3 + 2x^2 - 9x - 18}$$

enter

```
syms e x;
e = (2*x^2 -3*x +1)/(x^3 + 2*x^2 - 9*x -18)
de = diff(e,x)
```

You might also want to simplify the result by entering

```
simplify(de)
```

Exercise 38 Verify by doing the derivative by hand that MATLAB is doing it right.

4. To enter the matrix
$$m = \begin{bmatrix} d^2 & 2 & 7 \\ d & d^3 & 9 \\ 1 & 5 & \frac{1}{d} \end{bmatrix}$$
you enter
```
syms d m;
m = sym([d^2 2 7; d d^3 9; 1 5 1/d])
```
5. To take the determinant, enter
```
det(m)
```
6. To take the inverse of a symbolic matrix,
```
inv(m)
```

Another operation that will become important to us when we study Laplace transforms is partial fraction expansions (PFE), which you should have studied in Calculus. There are two ways of accomplishing this in MATLAB. The first is by using the `residue` command.

1. Let
$$f(x) = \frac{2x^2 - 3x + 1}{x^3 + 2x^2 - 9x - 18}$$
This may be represented by creating two vectors in MATLAB, with one vector representing the numerator polynomial and the other vector representing the denominator polynomial:
```
a = [1 2 -9 -18]     % denominator polynomial
b = [2 -3 1]         % numerator polynomial
```
Then the partial fraction expansion
$$\frac{b(s)}{a(s)} = \frac{r_1}{s - p_1} + \frac{r_2}{s - p_2} + \ldots + \frac{r_n}{s - p_n}$$
may be obtained using
```
[r,p] = residue(b,a)
```
Exercise 39 Try this using the expression above. Verify by hand computation that the partial fraction works as expected. Also, check the details on the command using `help residue`.

2. The other way to do partial fraction expansion is using the symbolic toolbox. There is no single function to do this (but you could write one!), but you could obtain the same effect by first integrating, then differentiating the function. (This works because MATLAB integrates the expression the way you would if you did it by hand: if first forms a partial fraction expansion, then integrates term-by-term. If you turn around and take the derivative, then you end up with the partial fraction expansion.) Enter the following:

```
syms x;
f = (2*x^2 - 3*x + 1)/(x^3 + 2*x^2 - 9*x - 18);
pfe = diff(int(f))
```

Exercise 40 Verify that this gives the partial fraction expansion correctly. You can turn around and undo the partial fraction expansion by

```
simplify(pfe)
```

3. MATLAB is able to handle partial fraction expansions even with non-integer coefficients.

   Exercise 41 Do a partial fraction expansion on the following expression using both numeric and symbolic techniques.

$$\frac{2x^2 - 3x + 1}{x^3 + 2x^2 - 9.4x - 18}$$

Even though MATLAB has symbolic capability, you should be aware that many of the kinds of operations you need to do on polynomials may be done using some of MATLAB's numeric commands. Explore the following commands:

1. `conv` (multiply two polynomials).
2. `deconv` (divide two polynomials).

You should also explore the use of the `solve` function.

A few more things

This tutorial only provides a start. There are many other things that it may be helpful to know about. Knowing that these things exist, you can start looking and learning for yourself.

- MATLAB can not only deal with numbers, but it can deal with strings (letters and characters). For example,

  ```
  nameprompt = 'Enter your name, por favor:'
  ```

- All the rows of a matrix must be the same length. Sometimes it is helpful to have data structures that can deal with different lengths. One way to do this is using *cells*.

  ```
  clear c;  % make sure this hasn't been used before
  c{1} = 'Name';   c{2} = 'Profession';
  c
  ```

- MATLAB has structures analogous to structures in languages like C or C++.

  ```
  clear d;
  d.name = 'Bob';
  d.title = 'jefe';
  d
  ```

- MATLAB can also be used for object oriented programming.
- MATLAB can make nice-looking 3D plots (`help plot3`, `help surf`, `help surf`).
- MATLAB can be used to build graphical user interfaces (GUIs), with slider bars, check boxes, and all the rest. See the `doc`.
- MATLAB has a built in debugger: you can set breakpoints, single step into or over functions, print variables, move around in the stack, and other things. Start with `help debug`.
- There is a MATLAB mode for emacs.
- Mathworks, MATLAB's producer, has a website where user contributed programs are archived.