VARIABLES

Convention

MATLAB uses variables that are used to define matrices. The matrix name can be of any group of alphanumeric symbols (max. length 19), given that the name starts with a letter.

MATLAB is case-sensitive; ‘MAT’ and ‘mat’ are two distinct names for variables.

Pre-defined variables

MATLAB provides for built-in variables, such as pi, eps, etc. You may learn their values from the MATLAB interpreter.

```matlab
pi
help pi
```

To learn of the active variables in your current session, use the keyword `who`:

```matlab
who
```

Sizes and description may be obtained by using `whos`:

```matlab
whos
```

The variables `ans` keeps track of the most recent output not assigned to a variable.

Removing variables

Use the keyword `clear`, followed by the variable name.

```matlab
clear var_name
```

VECTORS

Scalar values in MATLAB are considered as 1x1 matrices. Thus we could declare a variable `C` holding the scalar 2.54 as:

```matlab
C = 2.54  or  C = [ 2.54 ]
```

Creating and storing vectors

Vectors may be declared in several different ways. For instance, the following statement:

```matlab
0:2:8
```

creates a row vector of size 4 and initializes it as:
Vectors may also be assigned to variables for future reference:

\[ a = 0:0.1:10 \]

\[
\begin{align*}
\text{ans} & = 0 & 2 & 4 & 6 & 8 \\
\end{align*}
\]

To declare a row vector with arbitrary values, a variety of syntax may be used:

This declares a row vector \( a \) with the values \( a(1) = 1, a(2) = 2, a(3)=3 \) and \( a(4) = 4 \):

1. \( a = [1, 2, 3, 4] \)
2. \( a = [1 2 3 4] \)

We may also use prior matrices in the creation of a new one:

\[ D = [ a \ 6 \ 7 \ 8] \]

This creates the following row vector and stores it in \( D \):

\[
\begin{align*}
\text{ans} & = 1 & 2 & 3 & 4 & 6 & 7 & 8 \\
\end{align*}
\]

To create column vectors, we may use the \textit{transpose symbol}. This is the single apostrophe:

\[ a' \]

\[
\begin{align*}
\text{ans} & = 1 \\
2 & \\
3 & \\
4 & \\
\end{align*}
\]

Column vectors can also be declared using the semicolon to separate distinct rows. NOTE: \textit{The spaces are important}.

\[ a = [1; 2; 3; 4] \quad // \text{output same as above} \]

\textbf{Element access}

We may access single elements or sub-ranges of elements by using indices in a fashion similar to that in other programming languages. Remember that in MATLAB, indices start with 1. So:

\[ a(1) \]

would give:

\[
\begin{align*}
\text{ans} & = 1 \\
\end{align*}
\]

\textbf{MATRICES}

To declare matrices, we can use the conventions above to define rows/columns and use the semi-colon operator:
A = [0:2:8; 1:2:9]

>>A =
0 2 4 6 8
1 3 5 7 9

Another way of declaring the same matrix is:

A = [0:2:8
1:2:9]

>>A =
0 2 4 6 8
1 3 5 7 9

Remember to maintain size consistency when declaring matrices.

A = [0:2:8; 1:1:9] // produces an error

Arbitrary matrices are declared as:

A = [0 2 4 3 -2; 1 1 -1 1 1; 2 2 2 2 2]

>>A =
0 2 4 3 -2
1 1 -1 1 1
2 2 2 2 2

A'

>>A =
0 1 2
2 1 2
4 -1 2
3 1 2
-2 1 2

Accessing matrix entries is similar to that done in most programming languages. The term A(3,4) will return the entry in the third row and the fourth column of the matrix A. Again, there is no array bounds checking in MATLAB, so make sure you do no exceed matrix dimensions.

Built-in matrices

There are a number of useful matrices pre-defined in MATLAB. One is the rand(n) function, which takes in an integer as its argument and creates a n x n matrix with random values between 0 and 1. rand(m, n) creates a similar m x n matrix.

Another example: zeros(m, n) produces an m-by-n matrix of zeros and zeros(n) produces an n-by-n one; if A is a matrix, then zeros(A) produces a matrix of zeros of the same size as A.

If x is a vector, diag(x) is the diagonal matrix with x down the diagonal; if A is a square matrix, then diag(A) is a vector consisting of the diagonal of A. What is diag(diag(A))? Try it.
MATRIX OPERATIONS

The following operations are available in MATLAB: +(addition), - (subtraction), * (multiplication), ^ (power), ' (transpose), \ (left division) and / (right division).

For binary operations, make use that matrix sizes are compatible. If a scalar is used in conjunction with a matrix, the operation is applied to each element of the matrix.

For addition and subtraction, operations are obviously element-wise. To specify entry-level operations for the others, we use the "." operator:

\[
[1 \ 2 \ 3 \ 4] \cdot [1 \ 2 \ 3 \ 4] \quad \text{will yield} \quad [1 \ 4 \ 9 \ 16].
\]

Vectors and submatrices are often used in MATLAB to achieve fairly complex data manipulation effects. "Colon notation" (which is used both to generate vectors and reference submatrices) and subscipting by vectors are keys to efficient manipulation of these objects. Creative use of these features permits one to minimize the use of loops (which slows MATLAB) and to make code simple and readable. Special effort should be made to become familiar with them.

The following statements will, for example, generate a table of sines. Try it.

\[
x = \{0.0:0.1:2.0\}';
y = \sin(x);
[x y]
\]

Note that since \( \sin \) operates entry-wise, it produces a vector \( y \) from the vector \( x \).

The colon notation can be used to access submatrices of a matrix. For example,

\[
A(1:4,3)
\]

is the column vector consisting of the first four entries of the third column of \( A \). A colon by itself denotes an entire row or column:

\[
A(:,3)
\]

is the third column of \( A \), and \( A(1:4,:) \) is the first four rows. Arbitrary integral vectors can be used as subscripts:

\[
A(:,[2 \ 4])
\]

contains as columns, columns 2 and 4 of \( A \). Such subscripting can be used on both sides of an assignment statement:

\[
A(:,[2 \ 4 \ 5]) = B(:,1:3)
\]

replaces columns 2,4,5 of \( b \) with the first three columns of \( B \). Note that the \textit{entire} altered matrix \( A \) is printed and assigned. Try it.

Columns 2 and 4 of \( A \) can be multiplied on the right by the 2-by-2 matrix \([1 \ 2;3 \ 4]\):

\[
A(:,[2,4]) = A(:,[2,4]) *[1 \ 2;3 \ 4]
\]

Once again, the entire altered matrix is printed and assigned.

If \( x \) is an \textit{n}-vector, what is the effect of the statement \( x = x(n:-1:1) \)? Try it.

\textit{Cells}

We may also group different matrices together to form \textit{cell arrays} of matrices.
CELL(M,N) or CELL([M,N]) % is an M-by-N cell array of empty matrices.

For eg:
»celldat=cell(4,1)  <-- Making a empty, cell, column vector with place
  celldat =               for four "things".
    [ ]
    [ ]
    [ ]
    [ ]

»celldat(1,1)= 4 %scalar quantity---Note use of "{" and "}"
  celldat =
    [4]
    [ ]
    [ ]
    [ ]

»celldat{1,1}=[1;2;2;1]; <-- Adding cnms in the second place.
»celldat{3,1}=[1;4];
»celldat{4,1}=[1 2;2 5];
»celldat
  celldat =
    [ 4]    % first element
    [4x1 double] % column vector with 4 elements
    [2x1 double] % column vector with 2 elements
    [2x2 double] % 2-by-2 matrix

The use of braces ("{" and "}") to access the contents of a cell must be remembered. If you use parentheses instead you will find that it can lead to confusion:

»celldat(1,1)  % This produces a cell with the number 4 in it
  ans =
    [4]
»celldat{1,1}  % This gives us the number 4
  ans =
    4
»celldat(4,1)
  ans =
    [2x2 double]
»celldat{4,1}  % The numeric values of the coefficients
  ans =
    1  2
    2  5

NOTE: It is not always necessary for MATLAB to display the results of a command to the screen. If you do not want the matrix A displayed, put a semicolon after it: A; . When MATLAB is ready to proceed, the prompt >> will appear. Try this on a matrix right now.

GRAPH PLOTTING

plot(x,y)

This command will plot the elements of vector x on the horizontal axis of a figure, and the elements of the vector y on the vertical axis of the figure. The default is that each time the plot command is issued, the current figure will be erased; we will discuss how to override this below. If we wanted to plot the simple, linear formula:

\[ y = 3x \]

We use the code:
x = 0:0.1:100;
y = 3*x;
plot(x,y)

Try this.

One thing to keep in mind when using the plot command is that the vectors x and y must be the same length. The other dimension can vary. Matlab can plot a 1 x n vector versus a n x 1 vector, or a 1 x n vector versus a 2 x n matrix (you will get two lines), as long as n is the same for both vectors.

The plot command can also be used with just one input vector. In that case the vector columns are plotted versus their indices (the vector 1:1:n will be used for the horizontal axis). If the input vector contains complex numbers, Matlab plots the real part of each element (on the x-axis) versus the imaginary part (on the y-axis).

To investigate plot aesthetics (color and format), type in help plot.

You can plot more than one function on the same figure. Let's say you want to plot a sine wave and cosine wave on the same set of axes. The following m-file could be used to do this:

```matlab
x = linspace(0,2*pi,50);
y = sin(x);
z = cos(x);
plot(x,y,x,z)
```

When plotting many things on the same graph it is useful to differentiate the different functions based on color and point marker. This same effect can also be achieved using the hold on and hold off commands. The same plot shown above could be generated using the following m-file:

```matlab
x = linspace(0,2*pi,50); //linear region from 0 to 50 in intervals of 2*pi
y = sin(x);
plot(x,y)
z = cos(x);
hold on
plot(x,z)
hold off
```

Always remember that if you use the hold on command, all plots from then on will be generated on one set of axes, without erasing the previous plot, until the hold off command is issued.

**Subplotting**

More than one plot can be put on the same figure using the subplot command. The subplot command allows you to separate the figure into as many plots as desired, and put them all in one figure.

```matlab
subplot(m,n,p)
```

This command splits the figure into a matrix of m rows and n columns, thereby creating m*n plots on one figure. The p'th plot is selected as the currently active plot. For instance, suppose you want to see a sine wave, cosine wave, and tangent wave plotted on the same figure, but not on the same axis. The following m-file will accomplish this:

```matlab
x = linspace(0,2*pi,50);
y = sin(x);
z = cos(x);
w = tan(x);

subplot(2,2,1)
plot(x,y)
```
subplot(2,2,2)
plot(x,z)
subplot(2,2,3)
plot(x,w)

Try the above code.

A further subplot(2,2,4) could also be added to this figure.

**Changing the axis**
```
axis([xmin, xmax, ymin, ymax])
```

**Adding text**
- **Plot title**: use `title('title')`
- **x-axis**: use `xlabel('title')`
- **y-axis**: use `ylabel('title')`

All of the above commands are issued after the actual plot command has been issued.

**Text on plot:**

1. `text(xcor, ycor, 'string') //need to specify exact coordinates for text`
2. `gtext('string') //use the cross-hair to the desired location on the figure with the mouse and click to specify position.`

Other commands that can be used with the `plot` command are:

- `clf` (clears the current plot, so it is blank)
- `figure` (opens a new figure to plot on, so the previous figure is saved)
- `close` (closes the current figure window)

**M-FILES IN MATLAB**

An m-file, or script file, is a simple text file where you can place Matlab commands. When the file is run, Matlab reads the commands and executes them exactly as it would if you had typed each command sequentially at the Matlab prompt. All m-file names must end with the extension '.m' (e.g. plot.m). If you create a new m-file with the same name as an existing m-file, Matlab will choose the one which appears first in the path order (`help path` for more information). To make life easier, choose a name for your m-file which doesn't already exist. To see if a filename.m exists, type `help filename` at the Matlab prompt.

To create an m-file, choose **New** from the **File** menu and select **m-file**. This procedure brings up a text editor window in which you can enter Matlab commands.

To save the m-file, simply go to the **File** menu and choose **Save** (remember to save it with the '.m' extension). To open an existing m-file, go to the **File** menu and choose **Open**.

Files may also be saved at the prompt: type in `save filename.m` to save your file. The statement `save` without a filename argument saves the current file as `matlab.m`

By default, all files are saved in the directory **C:\MATLAB\bin**. To observe the path, type in `path` at the MATLAB prompt.

After the m-file is saved with the name `filename.m` in the Matlab folder or directory, you can execute the commands in the m-file by simply typing `filename` at the Matlab prompt. Another way is to go the **File** menu, click **Run Script** and browse to the directory where your file(s) are stored.
If you don't want to run the whole m-file, you can just copy the part of m-file that you want to run and paste it at the Matlab prompt.

FUNCTIONS

Built-in Functions

MATLAB has several built-in functions, several of which apply to both scalars and vectors.

\( \sin(x), \exp(x), \log(x), \text{abs}(x) \). Other functions include \text{round}(x), \text{fix}(x), \text{ceil}(x), \text{floor}(x), \text{sum}(x) \) and \text{prod}(x).

To build your own functions, use the keyword function

\[
\text{function } [\text{output1, output2, ...}] = \text{filename}(\text{input1, input2, input3, ...})
\]

Eq:

\[
\text{function } [\text{var3}] = \text{multiply}(\text{var1, var2})
\]

\% \text{multiply} is a function that multiplies two numbers

\text{var3} = \text{var1*var2};

Save this as “multiply.m” and invoke it as:

\[
Y = \text{multiply}(3,8);
\]

PROGRAMMING in MATLAB

Though programming will not be dealt with extensively, it is useful to be familiar with basic techniques in MATLAB. The code below reproduces the operation \text{add} for two matrices \( a \) and \( b \).

\[
\text{function } c = \text{add}(a, b)
\]

\% \text{c=add(a,b). This is the function which adds}
\% the matrices \( a \) and \( b \). It duplicates the MATLAB
\% function \( a+b \).

\[
[m,n]=\text{size}(a);
[k,l]=\text{size}(b);
\text{if } m\neq k \mid n\neq l,
\% negation is given by \'~\'
\text{r='ERROR using add: matrices are not the same size'};
\text{return},
\text{end}
\text{c=\text{zeros}(m,n); \% initialize an m-by-n matrix \'c\' with all zeroes.}
\text{for } i=1:m,
\text{for } j=1:n,
\text{c}(i,j)=a(i,j)+b(i,j);
\text{end}
\text{end}
\]

For further help, consult the tutorials on the web site.
A list of common built-in MATLAB functions/symbols is produced below:

```
intro  <       chol    end        function    lu        quit      sprintf
help   >       clc      eps       global     macro      quit      sqrt
demo   =       clear    error     grid       magic      rand      startup
{       &       conj     exit      hold       memory     real       subplot
(       |       clock    exist     hold       memory     real       subplot
)}      abs       contour   exp       ident      meta      rem       svd
.       all      cos       expm      if         min       return     tan
,       ans       cumprod  eye       imag      nan       round     text
;       any       cumsum   feval     inf        nargin    save      title
%       acos      delete    fft       input      norm      schur      type
:       asin      det       filter    inv        ones      script     what
;       atan2     diary     finite     keyboard   pause      semilogy    who
+       axis      dir       fix       load      pi        setstr     xlabel
-       balance   disp      floor     log        plot      shg       ylabel
*       break     echo      flops     loglog     polar     sign      zeros
\       casesen   eig       for       logop      prod      sin
/       ceil      else      format     ltitr      prtsc     size
^       chdir     elseif    fprintf    ltitr      qr        sort
```

```
acosh   demo     Hankel   membrane     print      table1
angle   demolist hds       menu       quad       table2
asinh   dft      hist      meshdemo    quaddemo   tanh
atanh   diff     hist      meshdom     quadstep    tek
bar     eigmovie histogram mkpp      rank       tek4100
bench   ergo     hp2647    movies      rat        terminal
bessel  etime    humps     nademo     ratmovie    toepol
bessela expm1    idft      nelder      readme     trace
besselh expm2    ieeex     neldstep    residue     translate
besseln expm3    ifft      nnls       retro      tril
blanks  feval    ifft2      null       roots      triu
cdf2rdf fft2     info       num2str    rot90      unmkpp
census  fftshift inquire ode23      rratref    vdpol
citoh   fitdemo   int2str    ode45       rratrefmovie versa
cla     fitfun    invhilb    ode45       rref       vt100
compan  flipx     isempty    orth       rfs2csf    vt240
computer flipy    kron       pinv       sc2dc      why
cond    funm      length     plotdemo    sg100      wow
conv     gallery   log10      poly       sg200      xterm
conv2   gamma     logm       polyfit     sinh       zerodemo
corr    getenv     logspace   polyline    spline     zeroin
cosh    ginput    matdemo    polymark    sqrtm
ctheorem gpp       matlab     polyval    square
dc2sc   graphon   mean       polyvalm    std
```