MATLAB: A SIMPLE INTRODUCTION

2008

Matthew Lamb, BE, ME, Honours, Victoria University
MATRICES AND VECTORS

Single row vector: \( v = [1 \ 2 \ 3 \ 4] \) or \( v = [1,2,3,4] \) will return \( v = 1 \ 2 \ 3 \ 4 \)

To transform this row vector to a column vector: \( v = v' \)
This returns \( v = \begin{array} \hline \\ 1 \\ 2 \\ 3 \\ 4 \\ \end{array} \)

Single column vector: \( w = [1;2;3;4] \) will return \( w = 1 \\
2 \\
3 \\
4 \)

To transform this row vector to a column vector: \( w = w' \)
This returns \( w = 1 \ 2 \ 3 \ 4 \)

Multiple column and row matrices: \( m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \)
Will return \( m = \begin{array} \hline \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ \end{array} \)

Vector indexing: Used to find singular and multiple values at desired points

Index position given using rows then columns ie. Vector(rows, columns)

\[
m = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}
\]

For example: 1. \( m(3,2) = 8 \)

2. \( m(:,2) = 2 \)
Note that \( (colon) \) represents all the rows or all the columns
MULTIPLICATION, ADDITION, SUBTRACTION AND DIVISION

Multiplication and Division: Since MATLAB deals with matrices it needs to be made aware that it is dealing either with matrix or normal multiplication (we will only every use scalar)

To perform normal multiplication and division you must always have a dot preceding the multiplication or division sign as follows: .* and . /

Note this dot is also required for powers (.^) otherwise matrix operations will be preformed

Examples: 1. [1 2 3].*2 = 2 3 6
   2. [1 2 3].*[2 3 4] = 2 6 12
   3. [1 2 3]./[2 3 4] = 0.5000  0.6667  0.7500

Addition and Subtraction: Addition and subtraction is much simpler (no dots required) as shown below

Examples: 1. [1 2 3] + 2 = 3 4 5
   2. [1 2 3] + [2 3 4] = 3 5 7

COLON NOTATION

Creating vectors/arrays with equal steps: Vector = starting value : step : final value

For example a vector which contains all values 1 to 10 with an increment of 0.1 can be created by typing x = 1:0.1:10

This returns x = 1

x = 1.1  1.2  1.3  1.4 .... 9.8   9.9  10
PLOTTING IN MATLAB

2 Dimensional plots: The command for producing 2-D plots is simply plot.

Eg let \( x = 1:2:10 \) and \( y = 1:1:5 \) and plot a 2-D graph of \( x \) versus \( y \)

\[
x = \begin{array}{c}
1 \\
3 \\
5 \\
7 \\
9 \\
\end{array} \\
y = \begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}
\]

Type: plot(x,y)

Produces:
Improving using inbuilt commands such as:

- `title`
- `xlabel`
- `ylabel`
- `grid on`
- `set`

Can Produce:

![Graph](image)

Sample code and description:

```matlab
plot(x,y,'r-.','linewidth',3)
title('X versus Y','fontsize',14)
xlabel('X data')
ylabel('Y data')
grid on
set(1,'color','w')
```
USER INTERACTION IN MATLAB

Common functions for allowing users to input information:

- `input`
- `inputdlg`

**input command:**

Simple command which prompts users to input required information (in the form of strings or numbers) within the command window:

Sample Code:

```matlab
A = input('Enter the value of A: ')
```

This code will return the following in the command window:

*Enter the value of A:*

Now if A is a number simply type the number for example 5. However, if A happens to be a string for example `Area`, the following must be entered to represent A as a string:

`'Area'`

**inputdlg command:**

More sophisticated than the `input` command. The `inputdlg` command brings up a separate window for users to enter values in. An example of this is as follows:

```matlab
answer = inputdlg({'Question 1','Question 2'}, 'Title ' ,Number of lines per answer, {'Default answer 1','Default answer 2'});
```

The format of the `inputdlg` command is as follows:

```matlab
answer = inputdlg({'Question 1','Question 2'}, 'Title ' ,Number of lines per answer, {'Default answer 1','Default answer 2'});
```
An example of this is as follows:

```matlab
answer = inputdlg({'SAMPLE WINDOW SIZE (% OF DATA)','%OVERLAP (% OF WINDOW SIZE)'}, 'Details ',1, {'10','50'});
```

This code will produce the following window:

![Window Image]

To use the results from the window it is necessary to separate the variables:

1. **The variables as numbers (need to convert the strings entered into numbers), the following format applies:**

   ```matlab
   Question1 = str2num(answer{1});
   Question2 = str2num(answer{2});
   
   Etc, etc.
   ```

2. **The variables strings, the following format applies:**

   ```matlab
   Question1 = (answer{1});
   Question2 = (answer{2});
   
   Etc, etc.
   ```

Some command commands used to display answers from your Matlab codes include the following:

`sprintf`, `fprintf` and `msgbox`. These functions will be further illustrated during assignment exercises.
LOADING NON-BINARY DATA

Without Headings: When loading non-binary numerical data in MATLAB it is most simple to use the `load` command

For Example: given the following data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

Say you want x to equal all the data in column 1 and y to equal the data in column 2

Simply type: `data = load('filename');` – this will load all the data in the file

Then assign column 1 to x by typing: `x = data(:,1);`

Then assign column 2 to y by typing: `y = data(:,2);`

This results in:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

OR say you want x to equal the last 3 numbers in column 1 and you don’t know the overall length of column 1

Type: `data = load('data.txt');`

```
x = data(end-2:end,1);
```

Why minus 2 not 3?

```
x =

    1
    2
    3
    4
```

Minus 3

Misusing 3 puts you here giving the last 4 not the last 3

```
    4
```

end
With Headings: This requires the use of `fopen` and `fscanf`. Note `fscanf` also allows the use of strings

For Example: given the following data

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

Say you want x to equal all the data in column 1 and y to equal the data in column 2

First you need to open the file by typing: `fid = fopen('filename')`

To read the first heading type: `heading1 = fscanf(fid,'%s',1)`

Note `%s` is used to read strings and `%f` to read floating point numbers

This results in `heading1 = 'X'`  
Note: this is a string

To read the second heading type: `heading2 = fscanf(fid,'%s',1)`

This results in `heading2 = 'Y'`  
Note: this is also a string

If the headings won’t be used you can eliminate them in one step: `headings = fscanf(fid,'%s',2)`

To load all the data type: `data = fscanf(fid,'%f',[2, inf])`

This will produce the data in rows not columns:

```
data =
1  2  3  4
6  7  8  9
```

To convert to columns type: `data = data'`  
Note this step is not required if you don’t mind using rows instead of columns

This produces:

```
data =
1  6
2  7
3  8
4  9
```

Then the data can be assigned using the method we used for without headings
FOR LOOPS

Basic structure:

\[
\text{for } i = \text{min no.}:\text{step}:\text{max no.} \\
\quad \text{expressions} \\
\text{end}
\]

Example 1. Finding velocity given a change in acceleration every second

Incremental Velocity equation: \( v = v_0 + at \)

Data: \( v_0 = 2 \text{ m/s} \) acceleration = [1 2 7 0 5] m/s\(^2\)

Flow Diagram:

Sample Code:

\[
a = [1 \ 2 \ 7 \ 0 \ 5]; \\
V = 2; \\
t = 0; \\
\text{for } i = 1:\text{length(a)} \quad \text{Note: step = 1} \\
\text{plot(t,V,'ro')} \\
\text{hold on} \\
\quad t = t + 1; \\
\quad V = V + a(i);
\text{end}
\]
Results in:

<table>
<thead>
<tr>
<th>Time</th>
<th>Acceleration</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>17</td>
</tr>
</tbody>
</table>
Storing calculated data from for loops (single values only):

**Basic Structure:**

*Note: The variable must exist outside the loop*

```
Variable = [];  \textit{Note: can be a numerical value instead of \([\,]\);} \\
J = 0; \\
for \(i = \text{min no.}:\text{step}:\text{max no.}\) \\
    J = J + 1; \\
    Variable(J) = \text{expression} \\
end
```

**Example:** See previous example question, but now storing the values for velocity

```matlab
a = [1 2 7 0 5]; 
V(1) = 2;  \textit{Note: index values must be + non-zero values} 
t = 0; 
for \(i = 2:\text{length}(a)+1\)  \textit{Note: since the step is 1 the use of \(J\) is not required} 
    t = t + 1; 
    V(i) = V(i-1) + a(i-1); 
end 
plot(0:length(a),V,’ro’)  \textit{Note:0:length(a) represents time}
```

**Results in:**

<table>
<thead>
<tr>
<th>Time</th>
<th>Acceleration</th>
<th>Velocity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2 3</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>2 3 5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2 3 5 12</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2 3 5 12 12</td>
</tr>
</tbody>
</table>

Matthew Lamb, BE, ME, Honours: 11/15
Postgraduate Student in the School of Civil, Architectural and Mechanical Engineering, Victoria University.
This storage of V allows for future calculations based on velocity such as the average velocity, etc.
Using a step of more than one:

This is usually used when trying to analyse sub-groups of data

For example say you have a data record that contains 10,000 values and you wish to divide this up into smaller sub-records containing 100 values and to find the mean of each of these sub-records

(Total record variable = x)

Sample code:

```matlab
for i = 1:100:length(x) - 100 + 1;
    sub_record = x(i:i+100-1);
    mean_sr = mean(sub_record);
end
```

Say you want to store the mean values:

Sample code:

```matlab
mean_sr = [];
J = 0;
for i = 1:100:length(x) - 100 + 1;
    J = J + 1;
    sub_record = x(i:i+100-1);
    mean_sr(J) = mean(sub_record);
end
```
IF STATEMENTS

Basic format:
if statement
    expression
end

Nested format:
if statement-1
    expression-1
elseif statement-2
    expression-2
else statement-3
    expression-3
end

Example: Given a vector such a \( S = [9 \ 2 \ 3 \ 4 \ 5] \) which can change as the program is running display a message box to show whether the first value is less then, equal to or higher then the mean value of \( S \)

Sample code:
\[
X = \text{mean}(s)
\]
if \( S(1) < X \)
    \( \text{msgbox}('\text{less then}') \)
elseif \( S(1) = X \)
    \( \text{msgbox}('\text{equal to}') \)
else \( \text{msgbox}('\text{Greater then}') \)
end
Example 2: A vector filled with different angles has been generated some positive and some negative with absolute values ranging from 0 to 90 degrees. Make a program which converts all negative values to their positive equivalent by adding 180 degrees to the

Sample code – (Nested for loop with an if statement):

```matlab
for i = 1:length(angle);
    if sign(angle(i)) == -1
        angle(i) = angle(i) + 180;
    end
end
```

This represents negative values