

Matlab Tutorial: Basics

Comments and solutions.

Matrix manipulation... There was a typo here which is now fixed in the online version.

The H line on the 3rd page should read:

$H = D(1, :)$ means the first row of D, so $H = [1 \ 2]$

In general, when you access matrices, the syntax is (row,column). And : means everything. So, $D(1,:)$ means row 1, all columns. $D(:,1)$ means all rows, column 1.

Exercise 1:

There are a number of ways to switch two columns in a matrix. I personally think the most efficient way is to just create 1 extra variable rather than many. Here is my solution:

```
%filename = ex1.m
%create this file using File -> New -> M-file
% and then save the file as ex1
%Run this file by typing "ex1" in the command window
M = [1 2 3; 4 5 6; 7 8 9];
%switch columns 2 and 3
temp = M(:,2);
M(:,2) = M(:,3);
M(:,3) = temp;
%switch rows 1 and 3
temp = M(1,:);
M(1,:) = M(3,:);
M(3,:) = temp;
%transpose the matrix
M = M';
M(2,2)
```

The answer is... 6!

Exercise 2: Here are the results from my command window.

```
>> A = [1 2 3];
>> B = [1;2;3];
>> B-A
??? Error using ==> -
Matrix dimensions must agree.

>> A+A

ans =

     2     4     6

>>
```

Exercise 3: Here is the output from my command window

```
>> A = [0 2; 1 4];
```

```
>> B = [1 3; 2 6];
```

```
>> A*B %this is a matrix multiplication
```

```
ans =
```

```
     4     12
     9     27
```

```
>> A.*B %this is an element-by-element multiplication
```

```
ans =
```

```
     0     6
     2    24
```

```
>> A^-1 %this is the inverse of the matrix A
```

```
ans =
```

```
 -2.0000    1.0000
  0.5000         0
```

```
>> A.^-1 %this is 1/element for each element of the matrix A
```

```
ans =
```

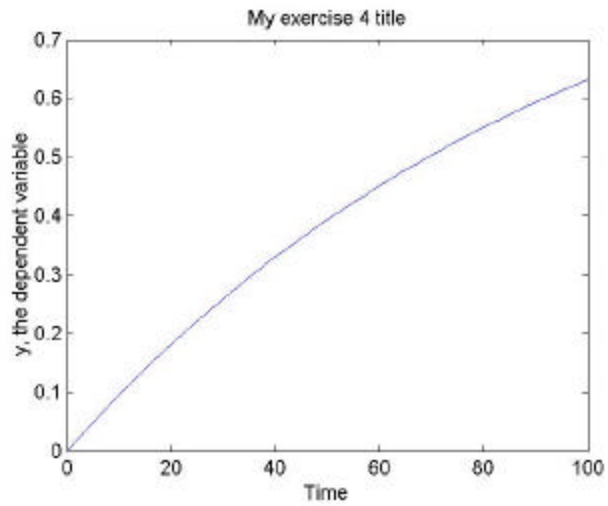
```
      Inf    0.5000
 1.0000    0.2500
```

For a review of matrix operations (and a reminder of how nice it is that you don't have to code up things like matrix-inversion because MATLAB will do it for you!):

<http://mathworld.wolfram.com/topics/MatrixOperations.html>

Exercise 4: This is my file ex4.m that I ran by typing “ex4”.

```
%ex4.m  
t = linspace(0, 100, 50);  
y = 1-exp(-t./100);  
plot(t,y);  
xlabel('Time');  
ylabel('y, the dependent variable');  
title('My exercise 4 title');
```



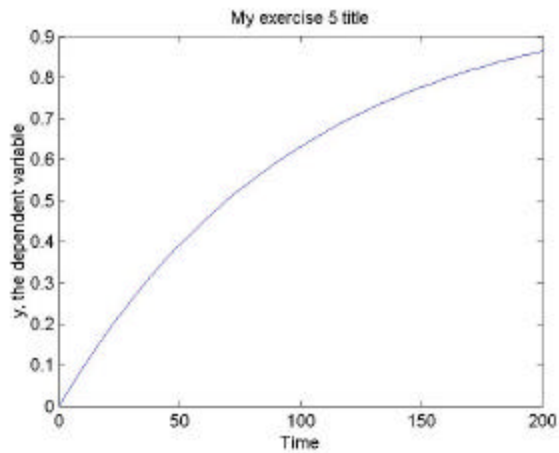
Exercise 5:

I again created a file using File -> New -> M-file and then saved it as “ex5function.m”
Note that since the first coding line of this file has the word “function” it can take inputs and give outputs, unlike a regular m-file.

```
function [t,y] = ex5function(start_time,end_time)
%this function takes in two numbers, start_time and end_time
%it creates a time vector with 100 points and then solves for y
%Both y and t are returned.
t = linspace(start_time,end_time,100);
y = 1-exp(-t./100);
```

I can run my function ex5function from the command window or from another m-file.

```
>> [time,y] = ex5function(0,200);
>> plot(time,y);
>> xlabel('Time');
>> ylabel('y, the dependent variable');
>> title('My exercise 5 title');
```



Exercise 6 (accidentally called Exercise 5 in the handout...):

Here is my m-file for this exercise.

```
%ex6.m
m = [1 1; 1 3];
%this is a 2x2 matrix, so I will make 2 for-loops to search the matrix
%I will also use an if-statement to test if an entry is greater than 1
total_greater_than_1 = 0;
for i=1:2
    for j=1:2
        if (m(i,j) > 1)
            %print that we found one and what the i,j indicies are
            'found an entry greater than one!' %output text
            i
            j
            %and, increment our counter
            total_greater_than_1 = total_greater_than_1 + 1;
        end
    end
end
total_greater_than_1
```

And here is the output from running ex6 in the command window.

```
>> ex6

ans =

found an entry greater than one!

i =

     2

j =

     2

total_greater_than_1 =

     1
```

Additional exercise 1:

Here is my m-file for this exercise, called add_ex1.m

```
%add_ex1
X = [1 2 5 -2 -1];
Y = [9 6 -3 5 7];
numpoints = 5;
refx = 0; refy = 0;
%loop over all x,y points to find the one closest to (0,0)
mindist = 999;
maxdist = -9999;
avgdist = 0;
for i=1:numpoints
    xd = X(i)-refx;
    yd = Y(i)-refy;
    dist = sqrt(xd*xd + yd*yd);
    avgdist = avgdist+dist;
    if (dist < mindist)
        mindist = dist;
    end
    if (dist > maxdist)
        maxdist = dist;
    end
end
mindist
maxdist
avgdist = avgdist/numpoints;
avgdist
```

And here is the output when I run “add_ex1”

```
mindist =
    5.3852
```

```
maxdist =
    9.0554
```

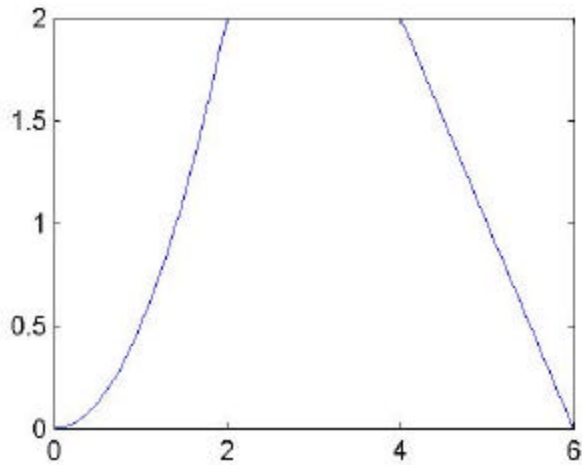
```
avgdist =
    6.7334
```

Additional exercise 2:

Here is my m-file for this exercise:

```
%add_ex2
x = linspace(0,6,100);
for i=1:length(x)
    if (x(i)<=2)
        y(i) = 0.5*x(i).^2;
    elseif ((x(i) > 2) & (x(i) <= 4))
        y(i) = 2;
    elseif (x(i) > 4)
        y(i) = 6-x(i);
    end
end
end
plot(x,y)
```

And here is the resulting plot:



Additional exercise 3:

Note, if you just use `sort(data)` to do this, it will sort all 3 columns so that the triplets of numbers are no longer associated with each other. So you need to do something else... you could write some complicated code to do this, but I used a method of using `sort` I found that saves the indices from the sort. I found this by looking at “help sort” and realizing that nothing there had the right functionality and then noticing the link at the bottom for “help cell/sort” and noticing this line:

`[Y,NDX] = SORT(X)` also returns an column vector of indices, `NDX`, such that `Y = X(NDX)`.

```
%add_ex3
data = [8 1.5 0; 5 -1 1; 1 -3 1; 11 2 1; 15 2 -1; 3 -3 0; 10 1 1; 14 1
-1; 2 -3 -1; 9 1.5 1; 6 -1 -1; 12 1.5 1; 4 -1 0; 7 -1 0; 13 1.5 -1];
plot(data);
title('Starting data');
[y,index] = sort(data(:,1));
for i=1:length(data(:,1))
    out(i,:) = data(index(i),:);
end
figure(2)
plot(out);
title('Data sorted on the first column');
```

Here are the output graphs:

