Back to 'Lab 7 - MATLAB'

Started on	Friday, 30 November 2018, 10:05 AM
State	Finished
Completed on	Tuesday, 4 December 2018, 11:15 PM
Time taken	4 days 13 hours
Grade	Not yet graded

Question 1

Correct

Mark 1.00 out of 1.00

Enter your partner's UD email address. If you did not work with a partner enter the word "none".

Answer: bobbymcc@udel.edu

Correct

Marks for this submission: 1.00/1.00.

Question 2

Complete

Marked out of 5.00

Basic Linear Algebra: Matrices

Through the course of this exercise, you will create a text file called "MATLABsession.txt". Paste the contents of that text file into the space below.

Step 0: Video

Watch (again?) the MATLAB video, "Working with Arrays"

Step 1: Diary

Turn on the diary function which will record everything you are about to do in this MATLAB session in a file named *MATLABsession.txt*

>> diary MATLABsession.txt

Your entries and the results will be saved in the **MATLABsession.txt** diary.

Step 1: Introduction

This part will introduce you to some basic linear Algebra functions available to you in MATLAB. MATLAB allows for very easy computations with vectors and matrices. First perform the following matrix and vector operations in MATLAB and make sure you understand the output. m' represents the transpose of m. The ':' is a wildcard character that matches all values.

>> m = [1 2 5; 4 7 8; 2 4 6]
>> sum(m)
>> diag(m)
>> sum(m')
>> diag(m')
>> m(:,1)
>> m(1,:)
>> m(:,1) == 1
>> m(1,:) < 5
>> any(m(1,:)) < 5
>> all(m(:,1) == 1)
>> all(m(:,1) > 0)

Step 2: Basic Vector Manipulation.

Create the following matrices:

>> a = [1 3 5]
a =

1 3 5

>> b = [2 ; 4 ; 6]
b =

2
4
6

>> c = [5 3 6]
c =

5 3 6

Create a few other matrices of your own.

The function **length** returns the number of components of a vector

>> length (a)
ans =

3

The *dot operator* " **.**" in MATLAB is used for the component-wise application of the operator that follows the dot. In the following example, a .* c goes through the vectors a and c systematically multiplie the first value in a by the first value in c, the second by the second, etc. Try it:

>> a .* c
ans =

??

Apply the dot operator to the *power operator* **^** to the vector **a :**

>> a .^ 2
ans =

??

Consider multiplying the two vectors **a** times **b**: 1X3 * 3X1 results in a 1X1 vector (also called a *scalar*): a*b is 1*2+3*4+5*6. Try it:

>> a * b
ans =

44

Consider multiplying **b** times **a**: 3X1 * 1X3 results in a 3X3

>> b*a
ans =

2 6 10

4 12 20

6 18 30

Step 4: Matrices

We will now deal with operations on matrices. Create a 3-by-3 matrix:

>> A = [1 2 4; 5 6 7; 8 10 11]
A =

1 2 4
5 6 7
8 10 11

To extract a submatrix **B** consisting of rows 2 and 3 and columns 1 and 2 of the matrix **A** do the following (remember, while Python indices start with 0, MATLAB indices start with 1):

>> B = A([2 3],[1 2])
B =

5 6
8 10

Step 5: Dot Operator on Matrices

The dot operator **"."** works for matrices too. Let now create a 2x3 matrix:

>> A = [2 5 7; 3 5 6]
A =

2 5 7
3 5 6

>> B = [1 4 2; 2 3 5]

B=

1 4 2
2 3 5

The following command:

>> A.*B

ans =

2 20 14

6 15 30

computes the entry-by-entry product of **A** with **B**. This is called a “component-wise” operation. However, the following command generates an error message:

>> A*B

Error using *

Inner matrix dimensions must agree.

Why? Because A*B is attempting to do *matrix* multiplication, not *component-wise* multiplication! ***Make sure you understand this distinction!*** Matrix multiplication is a key aspect of Linear Algebra, but generally beyond the scope of this course.

Step 6: Matrix Transpose

To transpose a matrix is to reverse the rows and columns. The ' operator is MATLAB's transpose operator (it is standard notation to say that A' is the transpose of A):

>> A = [1 2 3;4 5 6;7 8 10]
A =

1 2 3
4 5 6
7 8 10

>> B = A’
B =

1 4 7
2 5 8
3 6 10

Step 7: Matrix Inverse and the Identity Matrix

Just as 1 is the multiplicative identity of the numbers so that 1*N=N for any number N, an *Identity Matrix* I is one such A*I=A or I*A=A. (In matrix multiplication, A*B may not equal B*A. In fact, it may not even be possible to multiply one way or the other!) Identity matrices are always square (i.e., they have the same number of rows as columns), and have ones on the diagonal and zeros elsewhere. MATLAB's function for generating an NxN identity matrix is eye(N).

>> I3 = eye(3)

I3 =

1 0 0

0 1 0

0 0 1

A matrix M is *invertable* if there's another matrix M^{-1} such that $M*M^{-1}=I$ for the appropriate identity matrix I, and M^{-1} is called the *inverse* of M. MATLAB's function **inv** is used to compute an inverse matrix.

Let the matrix **A** be defined as follows:

>> A = [1 2 4; 5 6 7; 8 10 11]

A =

```
    1     2     4
    5     6     7
    8    10    11
```

>> B=inv(A)

B =

```
  -0.6667    3.0000   -1.6667
   0.1667   -3.5000    2.1667
   0.3333    1.0000   -0.6667
```

To verify that B is the inverse matrix of A, one must show that **A*B = I_3** and **B*A = I_3**, where **I_3** is the 3-by-3 identity matrix. We have

>> **A*B**

ans =

```
1.0000   0        -0.0000
0        1.0000         0
0        0         1.0000
```

In a similar way, check that **B*A = I**.

Some matrices are not invertable: only square matrices have inverses, and not all of them!

>> A=ones(4)

A =

```
    1     1     1     1
    1     1     1     1
    1     1     1     1
    1     1     1     1
```

>> inv(A)

Warning: Matrix is singular to working precision.

ans =

```
   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
```

Step 8: Write everything to the file and close it with:

>> diary off

Don't forget to paste the contents of that file below! It will be attached to the assignment as soon as you "check" another question.

>> diary MATLABsession.txt

>> m = [1 2 5;4 7 8;2 4 6]

m =

```
    1     2     5
```

```
    4     7     8
    2     4     6

>> sum(m)

ans =

    7    13    19

>> diag(m)

ans =

    1
    7
    6

>> sum(m')

ans =

    8    19    12

>> diag(m')

ans =

    1
    7
    6

>> m(:,1)

ans =

    1
    4
    2

>> m(1,:)

ans =

    1     2     5

>> m(:,1) == 1
```

```
ans =

  3×1 logical array

   1
   0
   0

>> m(1,:) < 5

ans =

  1×3 logical array

   1   1   0

>> any(m(1,:)) < 5

ans =

  logical

   1

>> all(m(:,1) == 1)

ans =

  logical

   0

>> all(m(:,1) > 0)

ans =

  logical

   1

>> a = [1 3 5]

a =

     1     3     5

>> b = [2; 4; 6]
```

```
b =

    2
    4
    6

>> c = [5 3 6]

c =

    5    3    6

>> d = [0 1 4;8 9 8; 6 6 7]

d =

    0    1    4
    8    9    8
    6    6    7

>> e = [4 11 1999;2 2 1970]

e =

         4          11        1999
         2           2        1970

>> length(a)

ans =

    3

>> a.*c

ans =

    5    9   30

>> a.^2

ans =

    1    9   25

>> a*b
```

```
ans =

   44

>> b*a

ans =

    2     6    10
    4    12    20
    6    18    30

>> A = [1 2 4;5 6 7;8 10 11]

A =

    1     2     4
    5     6     7
    8    10    11

>> B = A([2 3],[1 2])

B =

    5     6
    8    10

>> A = [2 5 7;3 5 6]

A =

    2     5     7
    3     5     6

>> B = [1 4 2;2 3 5]

B =

    1     4     2
    2     3     5

>> A.*B

ans =

    2    20    14
```

```
    6    15    30

>> A*B
Error using  *
Incorrect dimensions for matrix multiplication.
Check that the number of columns in the first
matrix matches the number of rows in the second
matrix. To perform elementwise multiplication,
use '.*'.

>> A = [1 2 3;4 5 6;7 8 10]

A =

    1     2     3
    4     5     6
    7     8    10

>> B = A'

B =

    1     4     7
    2     5     8
    3     6    10

>> I3 = eye(3)

I3 =

    1     0     0
    0     1     0
    0     0     1

>> A = [1 2 4;5 6 7;8 10 11]

A =

    1     2     4
    5     6     7
    8    10    11

>> B = inv(A)

B =

   -0.6667    3.0000   -1.6667
```

```
    0.1667   -3.5000    2.1667
    0.3333    1.0000   -0.6667

>> A*B

ans =

    1.0000         0    0.0000
   -0.0000    1.0000    0.0000
   -0.0000   -0.0000    1.0000

>> B*A

ans =

    1.0000    0.0000   -0.0000
         0    1.0000   -0.0000
         0    0.0000    1.0000

>> A = ones(4)

A =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> inv(A)
Warning: Matrix is singular to working
precision.

ans =

   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf
   Inf   Inf   Inf   Inf

>> diary off
```

Question 3

Correct

Mark 1.00 out of 1.00

Write an *expression* for a row vector containing the values 1, 3, and 5.

Answer:

```
1  [1 3 5]
```

	Expected	Got	
✓	1 3 5	1 3 5	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 4

Correct

Mark 1.00 out of 1.00

Write an *expression* for a *column* vector containing the values 2, 4, and 6.

Answer:

```
1  [2; 4; 6]
```

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 5

Correct

Mark 1.00 out of 1.00

Write an expression for the length of an already-defined vector **v**.

Answer:

```
1  numel(v)
```

	Test	Expected	Got	
✓	v=[2 4 6 8]	4	4	✓
✓	v=1:100	100	100	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 6

Correct

Mark 1.00 out of 1.00

Write an expression that compares the elements of matrices A and B.

Answer:

```
1  A == B
```

	Test	Expected	Got	
✓	`A = [1,2,3;4,5,6;7,8,9];` `B = [1,2,3;8,7,6;9,8,6];`	1 1 1 0 0 1 0 1 0	1 1 1 0 0 1 0 1 0	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 7

Correct

Mark 1.00 out of 1.00

What is the value of **B** after these statements are executed?

B = [7, 2, 3, 2,0,1];

A = B > 2;

B(A) = 0;

Select one:

- a. an error
- b. B = [0 1 0 1 0 1]
- c. B = [1 0 1 0 0 0]
- d. B = [0 2 0 2 0 1] ✓

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question 8

Correct

Mark 1.00 out of 1.00

Given matrices **C** and **D** below, which statement will concatenate matrix **C** and **D** vertically?

C =

```
1   2   3
4   5   6
```

D =

```
 7   8   9
10  11  12
13  14  15
```

Select one:

- a. C;D
- b. C,D
- c. [C D]
- d. [C;D] ✓
- e. they can't be concatenated vertically because they don't have the same number of rows.

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question 9

Correct

Mark 1.00 out of 1.00

Given this 3 X 3 matrix called matA, which statement will cause an error in MATLAB?

matA =

```
1.2000   2.4000   7.5000
4.5000   3.4000   2.1000
9.1000   0.4000   6.2000
```

Select one:

- a. matA(6) = 5.6
- b. matA(4,2) = 5.6
- c. x = matA(6)
- d. x = matA(4,2) ✓

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question 10

Correct

Mark 1.00 out of 1.00

Given **A** and **B** are matrices with the same dimensions, what is the output of this expression?

A == B

Select one:

- a. an error message
- b. a logical array with **1**'s in any position where A and B are equal, and **0**'s in all other positions ✓
- c. **1** if all elements of A are equal to all elements of B, otherwise **0**
- d. **true** if all elements of A are equal to all elements of B, otherwise **false**

Your answer is correct.

Correct

Marks for this submission: 1.00/1.00.

Question 11

Correct

Mark 2.00 out of 2.00

Enter two correct assertEqual tests for the computeCircleArea function described below.

Note: to round a result to two decimal places in Matlab, use the **round** function like this:

```
round(answer * 100)/100
```

since Matlab's round function only works for integers.

Answer:

```
1  assertEqual(round(computeCircleArea(2) * 100) / 100,12.57)
2  assertEqual(round(computeCircleArea(1) * 100) / 100, 3.14)
```

	Expected	Got	
✓	`^SUCCESS: assertEqual\ (.*computeCircleArea.* ^SUCCESS: assertEqual\ (.*computeCircleArea.*`	`SUCCESS: assertEqual(round(computeCircleArea(2) * 100) / 100,12.57) SUCCESS: assertEqual(round(computeCircleArea(1) * 100) / 100, 3.14)`	✓

Passed all tests! ✓

Correct

Marks for this submission: 2.00/2.00.

Question 12

Correct

Mark 3.00 out of 3.00

Define a MATLAB function **computeCircleArea** with one parameter, the circle's radius. It should return the area of a circle having that radius.

Answer:

```
function [area_of_circle] = computeCircleArea(radius)
    area_of_circle = pi * radius ^ 2;
end
```

	Test	Expected	Got	
✓	`disp(computeCircleArea(5));`	78.5398	78.5398	✓
✓	`disp(computeCircleArea(2.2));`	15.2053	15.2053	✓

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 13

Correct

Mark 2.00 out of 2.00

Enter three correct assertEqual tests for the `computeCylinderVolume` function described below.

Answer:

```
assertEqual(round(computeCylinderVolume(1,1) * 100) / 100,3.14)
assertEqual(round(computeCylinderVolume(1,2) * 100) / 100,12.57)
assertEqual(round(computeCylinderVolume(2,2) * 100) / 100,25.13)
```

	Expected	Got

✓	`^SUCCESS: assertEqual\ (.*computeCylinderVolume.*`
	`^SUCCESS: assertEqual\ (.*computeCylinderVolume.*`
	`^SUCCESS: assertEqual\ (.*computeCylinderVolume.*`
		`SUCCESS: assertEqual(round(computeCylinderVolume(1,1) * 100) / 100,3`
		`SUCCESS: assertEqual(round(computeCylinderVolume(1,2) * 100) / 100,1`
		`SUCCESS: assertEqual(round(computeCylinderVolume(2,2) * 100) / 100,2`

Passed all tests! ✓

Correct

Marks for this submission: 2.00/2.00.

Question 14

Correct

Mark 3.00 out of 3.00

Write a function `computeCylinderVolume` which should call `computeCircleArea`. `computeCylinderVolume` should take as parameters the height and the radius of a cylinder (in that order), and should return the cylinder's volume. (If you do not recall the formula for a cylinder's volume, find it on the Internet.)

Answer:

```
function [volume_of_cylinder] = computeCylinderVolume(height,radius)
    volume_of_cylinder = computeCircleArea(radius) * height;
end
```

	Test	Expected	Got	

✓	`disp(computeCylinderVolume(4,2))`	`in circleArea`
		`2`
		`50.2655`
			`in circleArea`
			`2`
			`50.2655`
				✓

✓	`disp(computeCylinderVolume(2,4))`	`in circleArea`
		`4`
		`100.531`
			`in circleArea`
			`4`
			`100.531`
				✓

✓	`disp(computeCylinderVolume(0,0))`	`in circleArea`
		`0`
		`0`
			`in circleArea`
			`0`
			`0`
				✓

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 15

Correct

Mark 2.00 out of 2.00

Enter three correct assertEqual tests for the `sumOddInts` function described below.

Answer:

```
assertEqual(sumOddInts(1,10),25)
assertEqual(sumOddInts(5,7),12)
assertEqual(sumOddInts(-10,10),0)
```

	Expected	Got	

✓	`^SUCCESS: assertEqual\(sumOddInts\(.*`
	`^SUCCESS: assertEqual\(sumOddInts\(.*`
	`^SUCCESS: assertEqual\(sumOddInts\(.*`
		`SUCCESS: assertEqual(sumOddInts(1,10),25)`
		`SUCCESS: assertEqual(sumOddInts(5,7),12)`
		`SUCCESS: assertEqual(sumOddInts(-10,10),0)`
			✓

Passed all tests! ✓

Correct

Marks for this submission: 2.00/2.00.

Question 16

Correct

Mark 3.00 out of 3.00

Write a function `sumOddInts()`. `sumOddInts` should return the sum of all odd integers between the values provided to the two parameters, inclusive, e.g., `sumOddInts(5,9)` should return 21.

- `start` and `finish` do not have to be odd, e.g, `sumOddInts(4,8)` should return 12. \
- `start` and `finish` do not have to be positive, e.g., sumOddInts(-5,7) should return 7, However, `finish` may be assumed to be greater or equal to `start`.
- whereas Python has a modulo operator "%", MATLAB has a built-in **mod()** function that returns the remainder. For instance,

```
>> mod (13,5)
    ans = 3

>> mod (13,2)
    ans = 1
```

Answer:

```
function [sum_odd_integers] = sumOddInts(start,ending)
    sum_odd_integers = 0;
    while start <= ending
        if mod(start,2) == 1
            sum_odd_integers = sum_odd_integers + start;
        end
        start = start + 1;
    end
end


```

	Test	Expected	Got	
✓	`disp(sumOddInts(5,11))`	32	32	✓
✓	`disp(sumOddInts(4,11))`	32	32	✓
✓	`disp(sumOddInts(-5,-1))`	-9	-9	✓

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 17

Correct

Mark 2.00 out of 2.00

Enter three correct assertEqual tests for the `linearSearch` function described below.

Answer:

```
1  assertEqual(linearSearch([1 2 3 4 5],3),3)
2  assertEqual(linearSearch([3 3 3 5 6 7],3),1)
3  assertEqual(linearSearch([1 2 3 4 5],0),-1)
```

	Expected	Got	

✓	`^SUCCESS: assertEqual\(linearSearch\(.*`
	`^SUCCESS: assertEqual\(linearSearch\(.*`
	`^SUCCESS: assertEqual\(linearSearch\(.*`
		`SUCCESS: assertEqual(linearSearch([1 2 3 4 5],3),3)`
		`SUCCESS: assertEqual(linearSearch([3 3 3 5 6 7],3),1)`
		`SUCCESS: assertEqual(linearSearch([1 2 3 4 5],0),-1)`
			✓

Passed all tests! ✓

Correct

Marks for this submission: 2.00/2.00.

Question 18

Correct

Mark 4.00 out of 4.00

Below is a Python function definition for a linear search of a list **lst** for **element**. It returns the first index at which **element** can be found in the list (or **None**) if **element** is not in the list.

```
def linearSearch(lst, element):
    i = 0
    for item in lst:
        if element == item:
            return i
        else:
            i += 1
    return None
```

Write a similar MATLAB function ***linearSearch.m***. Your MATLAB function should take as parameters an unsorted list of non-negative integers, and an integer value, and returns the index where that value is **first** located in the list. (No argument testing is required.) Unlike the Python code which returns `None` if the value is not in the array, your MATLAB function should return the value `-1` if the value is not found.

Answer:

```
function [linear_index] = linearSearch(lst,element)
    logical_arrays = lst == element;
    if logical_arrays == zeros(size(lst))
        linear_index = -1;
    else
        linear_index = find(logical_arrays,1);
    end
end
```

	Test	Expected	Got	
✓	`x = [6 4 8 1 0 3 12 9 8 15]; disp(linearSearch(x,6))`	1	1	✓
✓	`x = [6 4 8 1 0 3 12 9 8 15]; disp(linearSearch(x,1))`	4	4	✓
✓	`x = [6 4 8 1 0 3 12 9 8 15]; disp(linearSearch(x,8))`	3	3	✓
✓	`x = [6 4 8 1 0 3 12 9 8 15]; disp(linearSearch(x,5))`	-1	-1	✓
✓	`x = [17 14 6 4 8 1 0 3 12 9 8 15]; disp(linearSearch(x,15))`	12	12	✓

Passed all tests! ✓

Correct

Marks for this submission: 4.00/4.00.

Question 19

Correct

Mark 3.00 out of 3.00

You are to create a plot of years vs home sales in the town of "Ourtown".

Assume the existence of two lists, **years** and **sales**. For every entry in **years**, the corresponding entry in **sales** indicates how many houses were sold that year.

Construct a Matlab **plot** to display a graph of **sales** vs **years**. Label the x-axis 'Year', the y-axis 'Sales', and title the complete plot 'Annual Sales'.

Answer:

```
plot(years,sales)
xlabel("Year")
ylabel("Sales")
title("Annual Sales")
```

	Test	Expected

✓	`p=get(gca);`
	`thePlot = get(p.children(1));`
	`disp(thePlot.xdata);`
	`disp(thePlot.ydata);`
		`1998 1999 2000 200`
		
		`111 112 113 114 11`

✓	`% Check X-axis label`
	`disp(get(get(get(ancestor(gca,'figure'),'CurrentAxes'),'XLabel'),'String'))`
	`% Check Y-axis label`
	`disp(get(get(get(ancestor(gca,'figure'),'CurrentAxes'),'YLabel'),'String'))`
		`Year`
		`Sales`

✓	`% Check the title` `disp(get(get(gca,'Title'),'String'))`	`Annual Sales`

✓	`p=get(gca);`
	`thePlot = get(p.children(1));`
	`disp(thePlot.xdata);`
	`disp(thePlot.ydata);`
		`1992 1993 1994 199`
		
		`100 101 102 103 10`

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 20

Correct

Mark 3.00 out of 3.00

You are to create a plot of years vs home sales in the towns of "Ourtown" and "Yourtown".

Assume the existence of two lists for each town, **years** and **sales**. For every entry in **years**, the corresponding entry in **sales** indicates how many houses were sold that year. The lists for "Ourtown" will be **yearsO** and **salesO**, while for **"**Yourtown", they will be **yearsY** and **salesY.**

Construct a Matlab **plot** of **salesO** vs **yearsO**. In the same plot, display **salesY** vs **yearsY**. Label the x-axis 'Year', the y-axis 'Sales', and title the complete plot 'Annual Sales'.

Note: the two graphs must each use a different color and line style!

Hint: Try plotting some data using Matlab first before submitting here.

Answer:

```
plot(yearsO,salesO,"r-")
xlabel("Year")
ylabel("Sales")
title("Annual Sales")
hold on
plot(yearsY,salesY,"go")

```

	Test	Expected

✓	`p = get(gca);`
	`pc1 = get(p.children(1));`
	`pc2 = get(p.children(2));`
	`disp(pc1.xdata);`
	`disp(pc1.ydata);`
	`disp(pc2.xdata);`
	`disp(pc2.ydata);`
		`1998 1999 2000 200`
		
		`135 131 170 202 17`
		
		`1998 1999 2000 200`
		
		`129 132 137 149 15`

✓	`% Check X-axis label`
	`disp(get(get(get(ancestor(gca,'figure'),'CurrentAxes'),'XLabel'),'String'))`
	`% Check Y-axis label`
	`disp(get(get(get(ancestor(gca,'figure'),'CurrentAxes'),'YLabel'),'String'))`
		`Year`
		`Sales`

✓	`% Check the title` `disp(get(get(gca,'Title'),'String'))`	`Annual Sales`

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 21

Correct

Mark 3.00 out of 3.00

Let's plot the equation $y=2x^3-12x+8$ and its first and second derivatives for $-2 \le x \le 4$, all in the same plot.

First create the vector x, holding the domain of the equation (from -2 to 4 by increments of .01):

x=[-2:0.01:4];

Then create the vector y with the equation value at each x value (Note: the **.^** means componentwise exponentiation)

y=(2*(x.^3))-(12*x)+8;

Note, since x is an array, when MATLAB multiplies by x, it creates an array in which each value in the x array is multiplied.So really the above line is shorthand notation for:

y[0] = 2*x[0]^3 - 12*x[0] + 8
y[1] = 2*x[1]^3 - 12*x[1] + 8
y[2] = 2*x[2]^3 - 12*x[2] + 8
…
y[n] = 2*x[n]^3 - 12*x[n] + 8

for each value in x)

Now create a vector yd with the values of the first derivative:

yd = 6*(x.^2) - 12;

Now create a vector ydd with the values of the second derivative:

ydd = 12 * x;

Plot these 3 lines in one plot using different colors and line styles.

Answer:

```
x = [-2: 0.01: 4];
y = (2*(x.^3)) - (12*x) + 8;
plot(x,y,"r-")
hold on

yd = 6*(x.^2) - 12;
plot(x,yd,"b--")
hold on
```

	Test	Expected	Got	

✓	`p = get(gca);`
	`pc3 = get(p.children(1));`
	`pc2 = get(p.children(2));`
	`pc1 = get(p.children(3));`
	`x=[-2:0.01:4];`
	`disp(all(pc1.xdata==x))`
	`disp(all(round(1000*pc1.ydata)==round(1000*(2*x.^3-12*x+8))));`
	`disp(all(pc2.xdata==[-2:0.01:4]))`
	`disp(all(round(1000*pc2.ydata)==round(1000*(6*x.^2 - 12))));`
	`disp(all(pc3.xdata==[-2:0.01:4]));`
	`disp(all(round(1000*pc3.ydata)==round(1000*12*x)));`
		1
		1
		1
		1
		1
		1
			1
			1
			1
			1
			1
			1
				✓

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 22

Correct

Mark 3.00 out of 3.00

The distance traveled by an object fired from an airplane towards the earth (ignoring air resistance) is given by the equation:

$\mathbf{y = v_0t + (1/2)at^2}$

where $\mathbf{v_0}$ is the object's initial velocity, **a** is the acceleration due to earth's gravity, and **t** is time. Use MATLAB to calculate and plot how far an object travels for times 0-6 at intervals of 0.1 seconds if $\mathbf{v_0 = 12m/s}$ and $\mathbf{a = 9.81m/second^2}$

Label the x and y axes and give the plot a title.

Answer:

```
x = [0: 0.1: 6];
y = 12 * x + ((1/2) * 9.81 * x.^2);
plot(x,y)
xlabel("Time - seconds")
ylabel("Distance - metre")
title("Distance vs Time")
```

	Test	Expected	Got	

✓	`p=get(gca);`
	`thePlot = get(p.children(1));`
	`t=0:.1:6;`
	`disp(all(thePlot.xdata==t));`
	`disp(all(round(1000*thePlot.ydata)==round(1000*(12*t+9.81*t.^2/2))));`
		1
		1
			1
			1
				✓

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 23

Correct

Mark 5.00 out of 5.00

You should define this function in MATLAB and when finished, copy and paste your function into this submission box.

Use the Design Recipe (Include a docstring!) to define a function named `plot_history(history)` which consumes a matrix containing history data for a simulation from the Soft Landing Project #2 and plots altitude (using squares markers), speed (using circle markers), and fuel (using star markers) over time, **in that order**. All three plots should be on the same figure and can use the colors of your choice.

Tip: To plot more than one line in a figure, after your first plot statement, add the statement:

hold on

The history data passed to this function will be *a matrix,* the first column holds the ***altitude*** values, the second column holds the ***speed*** values, the third column holds the ***fuel*** values and the fourth column holds the thrust values. Each row represents a single time-step of the simulation.

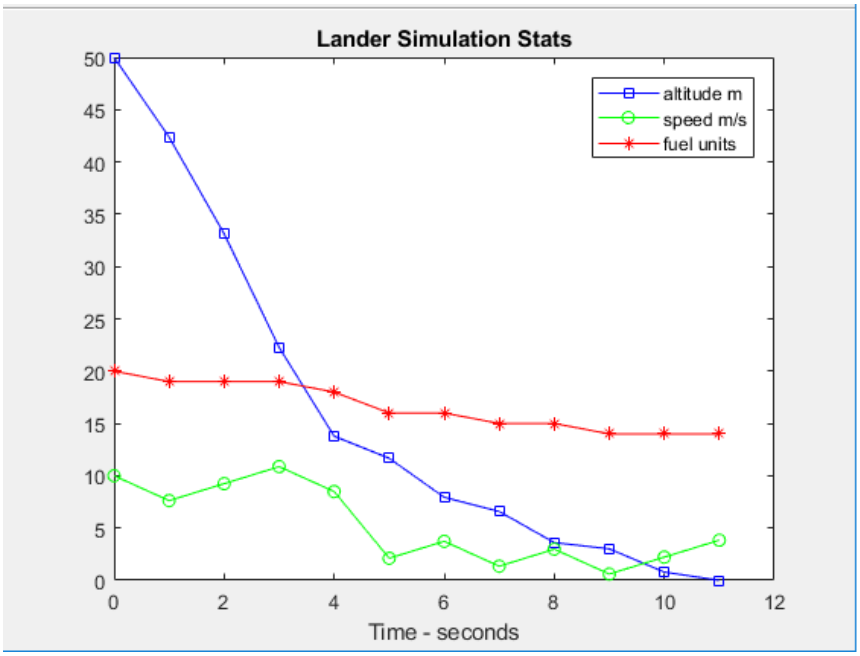
Include a title, an x label that displays **'*Time - seconds*',** and legends for the three plots:

- The units on the legend for **altitude** should be **m**
- The units on the legend for **speed** should be ***m/s***
- The legend for **fuel** should be ***fuel units.***

For example for this history matrix:

history = [50.0, 10.0, 20, 0;
42.38, 7.62, 19, 1;
33.13, 9.24, 19, 0;
22.27, 10.87, 19, 0;
13.78, 8.49, 18, 1;
11.67, 2.11, 16, 2;
7.94, 3.73, 16, 0;
6.58, 1.35, 15, 1;
3.61, 2.98, 15, 0;
3.01, 0.6, 14, 1;
0.79, 2.22, 14, 0;
0, 3.84, 14, 0];

Your plot should look like this:



Answer:

```
function [] = plot_history(history)
% Plots altitude, speed, and fuel (in that order) on the same figure given
% the history data in matrix
% Parameter:
%   history, a matrix of history data of simulation
% Return:
%   None
    altitude = history(:,1);
    speed = history(:,2);
    fuel = history(:,3);
    time = 0:1:size(history,1) - 1;
    plot(time,altitude',"rs")
    hold on
    plot(time,speed',"bo")
```

	Test	Expected

✓	`plot_history(history)`
	`p = get(gca);`
	`pc1 = get(p.children(1));`
	`pc2 = get(p.children(2));`
	`pc3 = get(p.children(3));`
	`disp(sum(pc1.xdata == [0 1 2 3 4 5 6 7 8 9 10 11]) );`
	`disp(sum(pc1.ydata == [20 19 19 19 18 16 16 15 15 14 14 14]));`
	`disp(sum(pc2.xdata == [0 1 2 3 4 5 6 7 8 9 10 11]) );`
	`disp(sum(pc2.ydata == [10 7.62 9.24 10.87 8.49 2.11 3.73 1.35 2.98 0.6 2.22 3.84]));`
	`disp(sum(pc3.xdata == [0 1 2 3 4 5 6 7 8 9 10 11]) );`
	`disp(sum(pc3.ydata == [50 42.38 33.13 22.27 13.78 11.67 7.94 6.58 3.61 3.01 0.79 0]));`
		12
		12
		12
		12
		12
		12

✓	`% Check X-axis label`
	`plot_history(history)`
	`disp(get(get(get(ancestor(gca,'figure'),'CurrentAxes'),'XLabel'),'String'))`
		Time - s

✓	`% Check the title`
	`plot_history(history)`
	`disp(get(get(gca,'Title'),'String'))`
		Lander S

Passed all tests! ✓

Correct
Marks for this submission: 5.00/5.00.

Question 24

Correct

Mark 3.00 out of 3.00

Write a function in MATLAB called **stats** that accepts an NxN matrix. This function should compute the following for each column and return 5 row vectors - mat_mean, mat_median, mat_min, mat_max and mat_std_dev which contain the mean, the median, the minimum, the maximum and the standard deviation values for each column in the NxN matrix.

For example:

Test	**Result**
A = [1 1 1; 3 4 5; 8 10 12]; [mean_A median_A min_A max_A std_dev_A] = stats(A); disp(mean_A) disp(median_A) disp(min_A) disp(max_A) disp(std_dev_A)	4 5 6 3 4 5 1 1 1 8 10 12 3.6056 4.5826 5.5678

Answer:

```
function [mat_mean mat_median mat_min mat_max mat_std_dev] = stats(A)
    mat_mean = mean(A);
    mat_median = median(A);
    mat_min = min(A);
    mat_max = max(A);
    mat_std_dev = std(A);
end
```

	Test	

✓	`A = [1 1 1; 3 4 5; 8 10 12];`
	`[mean_A median_A min_A max_A std_dev_A] = stats(A);`
	`if ~(all(mean_A == [ 4 5 6]) == 1)`
	`  disp("Mean values are not correct.")`
	`end`
	`if ~(all(median_A == [ 3 4 5]) == 1)`
	`disp("Median values are not correct.")`
	`end`
	`if ~(all(min_A == [1 1 1]) == 1)`
	`disp("Minimum values are not correct.")`
	`end`
	`if ~(all(max_A == [8 10 12]) == 1)`
	`disp("Maximum values are not correct.")`
	`end`
	`if ~(all(abs(std_dev_A -[3.6056  4.5826  5.5678]) > 1.0e-4) == 0)`
	`disp("Standard Deviation values are not correct.")`
	`end`
		✓

✓	`A =[1 1 1 1 1; 2 2 2 2 2; 3 3 3 3 3; 4 4 4 4 4; 5 5 5 5 5];`
	`[mean_A median_A min_A max_A std_dev_A] = stats(A);`
	`if ~(all(mean_A == [3 3 3 3 3]) == 1)`
		✓

```
   disp("Mean values are not correct.")
end
if ~(all(median_A == [3 3 3 3 3]) == 1)
 disp("Median values are not correct.")
end
if ~(all(min_A == [1 1 1 1 1]) == 1)
disp("Minimum values are not correct.")
end
if ~(all(max_A == [5 5 5 5 5]) == 1)
disp("Maximum values are not correct.")
end
if ~(all(abs(std_dev_A -[1.5811 1.5811 1.5811 1.5811 1.5811]) > 1.0e-4) == 0)
disp("Standard Deviation values are not correct.")
end
```

Passed all tests! ✓

Correct

Marks for this submission: 3.00/3.00.

Question 25

Correct

Not graded

Problem:

Alice buys four apples, a dozen bananas, and one cantaloupe for $6.00. Bob buys a dozen apples and two cantaloupes for $4.04. Carol buys two bananas and three cantaloupes for $5.84. You will use MATLAB to figure out how much single piece of each fruit cost.

Set up a system of 3 linear equations. Each equation has the form: ax + by + cz = cost, where a is the number of apples, b is the number of bananas, and c is the number of cantaloupes. That is, encode these 3 linear equations in one matrix for the quantities of fruit purchased, and one column vector for the cost of the purchase.

In the space below: assign a 3x3 matrix to the variable **bought** and a column vector to the variable **totals** so that **bought** and **totals** can be used to solve the system of linear equations.

Answer:

```
1  bought = [4 12 1;12 0 2;0 2 3];
2  totals = [6.00; 4.04; 5.84];
```

	Test	Expected	Got	

✓	`disp(bought(find(totals==5.84),:))`
	`disp(bought(find(totals==6),:))`
	`disp(bought(find(totals==4.04),:))`
		`0 2 3`
		
		`4 12 1`
		
		`12 0 2`
			`0 2 3`
			
			`4 12 1`
			
			`12 0 2`
				✓

Passed all tests! ✓

Question 26

Correct

Not graded

For the problem above, enter the augmented matrix that can be used to solve the problem. Do not assign the matrix to a variable.

Answer:

```
1  [4 12 1 6.00;12 0 2 4.04;0 2 3 5.84]
```

Passed all tests! ✓

Question 27

Correct

Not graded

For the above problem, solve for the 3 variables (a, b, c) which represent the price of each fruit. Your code should end with **ans** being a vector like **[a b c]** where **a** is the price of an apple, **b** is the price of a banana, and **c** is the price of a cantaloupe.

Note: it is not recommended that you try to solve the problem using Moodle. Solve it in Matlab, then just enter your final solution into Moodle.

Hint: Use Gaussian elimination manually manipulating the rows as shown in class, or learn about the matrix left-division operator in MATLAB.

Answer:

```
1  [0.05, 0.34, 1.72]
```

Passed all tests! ✓