Chapter 2 Tutorial 4

The purpose of this tutorial is to teach you about arithmetic operations involving vectors and matrices

```
% Always clear workspace variables before a new tutorial or program.
clear
```

Edit the code below and update the variable named **name** with your name for this tutorial in the code below.

```
name="";
fprintf("Output for Tutorial_02_4 run by %s.\n", name)
```

```
Output for Tutorial_02_3 written by .
```

Addition and Subtraction

Let's create two simple vectors

```
vectorA=1:5          % A row vector 1 through 5
```

```
vectorA = 1×5
     1     2     3     4     5
```

```
vectorB=10:10:50    % A row vector 10 through 50 incrementing by 10s
```

```
vectorB = 1×5
    10    20    30    40    50
```

Addition and subtraction of vectors does not require any special notation. Notice that element (1,1) of `vectorA` is added to element(1,1) of `vectorB` and so on for each corresponding element.

```
vectorA+vectorB
```

```
ans = 1×5
    11    22    33    44    55
```

```
vectorA-vectorB
```

```
ans = 1×5
    -9   -18   -27   -36   -45
```

Multiplication and Division

Multiplication of vectors requires a dot `.` operator to signify that we want each element to be multiplied to the corresponding element from each vector. This is referred to as **element-wise arithmetic**. If we do not use the dot operator, MATLAB will perform a matrix multiplication which is a very different thing altogether, we will discuss matrix math later in the course.

```
vectorA.*vectorB
```

```
ans = 1×5
    10    40    90   160   250
```

```
vectorA./vectorB
```

```
ans = 1×5
    0.1000    0.1000    0.1000    0.1000    0.1000
```

There is an exception where matrix and element-wise math overlap. That is when we have a scalar and a matrix. When of the operands is a scalar, both element-wise and matrix multiplication and division will result in the same outcome.

```
% The outcome here will be the same for matrix and non-matrix math
vectorA*2
```

```
ans = 1×5
     2     4     6     8    10
```

```
vectorA/2
```

```
ans = 1×5
    0.5000    1.0000    1.5000    2.0000    2.5000
```

is the same as...

```
vectorA.*2  % Explicitly declare element-wise multiplication
```

```
ans = 1×5
     2     4     6     8    10
```

```
vectorA./2  % Explicitly declare element-wise division
```

```
ans = 1×5
    0.5000    1.0000    1.5000    2.0000    2.5000
```

Tip: It is always better to be safe than sorry, since adding the dot operator does not inhibit functionality, it not only ensures the right outcome but allows someone reading your code to determine your intentions. Additionally, using the dot operator when a scalar is *expected* will still function properly if a matrix was used instead for some reason.

Exponents

Exponentiation requires the dot operator for element-wise exponentiation.

```
vectorA.^2  % Square each element in vectorA
```

Example:

Let's create a time table giving the velocity and position of a falling object over time.

```
% Declare the initial data
initialVel=19.6;
initialPos=0;
GRAVITY=9.81;

% Let's plot the velocity and position every 1/4 second for 4 seconds
% starting at t=0
timeInc=0:0.25:4;
```

```
% Compute the velocity over the time increments.
velocityY=initialVel - GRAVITY*timeInc.^2;

% Compute the position over the time increments. Remember, we don't NEED
% the dot operator on the multiplication since initialVel and (1/2) are
% scalar values but hopefully you can see how it makes it clearer by using it.
position=initialPos + initialVel.*timeInc + (1/2).*timeInc.^2;

% Create a table for displaying the output (again, we could do this right
% in the disp() function later, but let's keep it separate until you become
% more proficient.
dispTable=[timeInc', velocityY', position'];

% Display a nice output
disp('Analysis of a falling object')    % Title for the table
```

```
Analysis of a falling object
```

```
disp(' ')                               % Prints a blank line
```

```
disp('   Time(s)  Vel(m/s)   Pos(m)')   % Column headings
```

```
   Time(s)  Vel(m/s)   Pos(m)
```

```
disp(dispTable)                         % Disp the table
```

```
         0   19.6000         0
    0.2500   18.9869    4.9313
    0.5000   17.1475    9.9250
    0.7500   14.0819   14.9813
    1.0000    9.7900   20.1000
    1.2500    4.2719   25.2812
    1.5000   -2.4725   30.5250
    1.7500  -10.4431   35.8313
    2.0000  -19.6400   41.2000
    2.2500  -30.0631   46.6313
    2.5000  -41.7125   52.1250
    2.7500  -54.5881   57.6813
    3.0000  -68.6900   63.3000
    3.2500  -84.0181   68.9813
    3.5000 -100.5725   74.7250
    3.7500 -118.3531   80.5312
    4.0000 -137.3600   86.4000
```

Additional Notes:

-