Signals and Systems Lab

Variable Creation and Addition:

In this section, we just saw how to create different variables and how to perform arithmetic operations e.g addition. These assigned values can then be reassigned to other variables or even be replaced by new values.

```
a = 2;
b = 1;
c = a + b;
```

```
a = 5;
```

Matrix Creation:

We saw how we can create different matrices. Two 3x3 matrices have been created below by the name of 'a' and 'b'

```
a = [1,2,3 ; 4,5,6 ; 7,8,9];
b = [1,2,3 ; 4,5,6 ; 7,8,9];
```

Simple and Element-wise Operations:

Below is a demonstation specific to the multiplication operation. The '*' is simple multiplication or in case of matrices, matrix multiplication. The '.*' is the element-wise operator. Basically what it means is that the operation will only be performed in such a manner that the first element of the first matrix will only be multiplied with the first element of the second matrix. This operator accomodates for any size discrepancies in the matrices.

```
c = a * b;
```

```
d = a .* b;
```

Square Root:

This operation is quite self-explanatory

```
e = sqrt(25);
```

Creating vectors using linspace and colon operators:

When using the colon operator, the first element is the starting element. The second element is the spacing of the elements and the last is the end range of the values in the vector. Meanwhile, the linspace operator is used to create vectors with equally spaced elements. The first is starting element, second is ending element and last is the total number of elements in the vector.

```
x = 1:2:10;
```

```
y = linspace(1,20,5);
```

```
y = linspace(1,20);
```

Various Operations:

Z is formed by the horizontal concatenation of the matrix 'a' and matrix 'b'. A comma or space between the elements in the square brackets mean horizontal concatenation and a semi-colon refers to vertical concatenation.

```
z = [a , b];
a = a(2,2);
```

This is used to create a transpose of an element.

```
x = x';
```

The 'clc' command erases all the command history from the command window.

```
clc
```

The 'clear variables' or 'clearvars' operations deletes all the stored variables from the workspace.

```
clear variables
```

```
 f = [1 2  3 4];
 g = [5 6 , 7 8];
```

Matrix Addition:

If the dimensions of two matrices are equal, then they can added to each other. This is known as Matrix Addition. Each element is added to its corresponding element in the second matrix.

```
h = f + g;
```

Scalar Multiplication:

When a scalar is multiplied with a matrix, it results in all the elements being multiplied by that element.

```
d = 5 * h;
```

Ones, zeros and eye function:

The ones function creates a matrix with all ones. If we give only one numeric parameter n then it creates an nxn matrix. If we give two parameters such as n and m. Then it creates n rows and m columns. With ones, all the elements are ones. With zeros, all elements are zeros. And with eye, it creates an identity matrix.

```
i = zeros(3);
```

```
j = eye(3);
```

```
j = 3×3
     1     0     0
     0     1     0
     0     0     1
```

```
clearvars
```

```
a = [1,2,3,4 ; 5,6,7,8 ; 9,10,11,12 ; 13,14,15,16];

k = a(1:3,2);

j = a(1:2,4);

l = a(3,:);
```

```
clearvars
```

Task 1:

Part a:

```
A=[1 0 4 5 3 9 0 2]
```

```
A = 1×8
     1     0     4     5     3     9     0     2
```

```
a=[4 5 0 2 0 0 7 1]
```

```
a = 1×8
     4     5     0     2     0     0     7     1
```

Part b:

```
B=[A a]
```

```
B = 1×16
     1     0     4     5     3     9     0     2     4     5     0     2     0⋯
```

```
C=[a , A]
```

```
C = 1×16
     4     5     0     2     0     0     7     1     1     0     4     5     3⋯
```

Part c:

```
D = zeros(1,50)
```

```
D = 1×50
     0     0     0     0     0     0     0     0     0     0     0     0     0⋯
```

```
E = ones(1,100)
```

```
E = 1×100
     1     1     1     1     1     1     1     1     1     1     1     1     1⋯
```

Part d:

```
F=1:30
```

```
F = 1×30
     1     2     3     4     5     6     7     8     9    10    11    12    13⋯
```

```
G = 25:-3:1
```

```
G = 1×9
    25    22    19    16    13    10     7     4     1
```

```
H = 0:0.2:2.0
```

```
H = 1×11
         0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000⋯
```

Task 2:

Variable Creation:

```
V1 = [1:9,0];
V2 = [0.3 1.2 0.5 2.1 0.1 0.4 3.6 4.2 1.7 0.9];
V3 = [4 4 4 4 3 3 2 2 2 1];
```

Part a:

```
Addition = V1 + V2 + V3
```

```
Addition = 1×10
    5.3000    7.2000    7.5000   10.1000    8.1000    9.4000   12.6000   14.2000⋯
```

Part b:

```
V1fifth = V1(1,5)
```

```
V1fifth = 5
```

```
V2fifth = V2(1,5)
```

```
V2fifth = 0.1000
```

```
V3fifth = V3(1,5)
```

```
V3fifth = 3
```

```
%V1(0)
%V1(11)
```

Part c:

```
V4 = V2(1,1:5)
```

```
V4 = 1×5
    0.3000    1.2000    0.5000    2.1000    0.1000
```

```
V5 = V2(1,6:end)
```

```
V5 = 1×5
    0.4000    3.6000    4.2000    1.7000    0.9000
```

Part d:

```
V20 = V2
```

```
V20 = 1×10
    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    3.6000    4.2000⋯
```

```
V20(6) = []
```

```
V20 = 1×9
    0.3000    1.2000    0.5000    2.1000    0.1000    3.6000    4.2000    1.7000⋯
```

```
V6 = V20
```

```
V6 = 1×9
    0.3000    1.2000    0.5000    2.1000    0.1000    3.6000    4.2000    1.7000⋯
```

```
V7 = V2
```

```
V7 = 1×10
    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    3.6000    4.2000⋯
```

```
V7(1,7) = 1.4
```

```
V7 = 1×10
    0.3000    1.2000    0.5000    2.1000    0.1000    0.4000    1.4000    4.2000⋯
```

```
V8 = V2([1 3 5 7 9])
```

```
V8 = 1×5
    0.3000    0.5000    0.1000    3.6000    1.7000
```

Part e:

All elements are subtracted individually from 9 to create a new matrix.

```
9 - V1
```

```
ans = 1×10
     8     7     6     5     4     3     2     1     0     9
```

Each element is multiplied by 5.

```
V1*5
```

```
ans = 1×10
     5    10    15    20    25    30    35    40    45     0
```

Vector V1 is being added to Vector V2.

```
V1+V2
```

```
ans = 1×10
    1.3000    3.2000    3.5000    6.1000    5.1000    6.4000   10.6000   12.2000 ⋯
```

Vector V3 is subtracted from vector V1.

```
V1-V3
```

```
ans = 1×10
    -3    -2    -1     0     2     3     5     6     7    -1
```

Element-wise multiplication of vector V1 and vector V2.

```
V1.*V2
```

```
ans = 1×10
    0.3000    2.4000    1.5000    8.4000    0.5000    2.4000   25.2000   33.6000 ⋯
```

Due to dimension incompabilities, matrix multiplication cant take place so we commented it out.

```
%V1*V2
```

Each element is raised to the power of two.

```
V1.^2
```

```
ans = 1×10
     1     4     9    16    25    36    49    64    81     0
```

Element-wise the elements of vector V1 are raised to the power of their corresponding elements in V3.

```
V1.^V3
```

```
ans = 1×10
     1    16    81   256   125   216    49    64    81     0
```

A matrix power cannot be another matrix.

```

%V1^V3
```

This operation returns a vector with all elements which are equal to their corresponding element to return a 1 and those that are not to return a 0.

```
V1 == V3
```

```
ans = 1×10 logical array
   0   0   0   1   0   0   0   0   0   0
```

This element checks all elements in V1 and returns 1 for all those greater than 6. Otherwise it is a 0.

```
V1>6
```

```
ans = 1×10 logical array
   0   0   0   0   0   0   1   1   1   0
```

This element checks all elements in V1 and returns 1 for all those greater than their corresponding element in V3. Otherwise it is a 0.

```


V1>V3
```

```
ans = 1×10 logical array
   0   0   0   0   1   1   1   1   1   0
```

First, a vector is created with the answers of the operation (V1>2) and then they are subtracted from the vector V3.

```
V3-(V1>2)
```

```
ans = 1×10
     4     4     3     3     2     2     1     1     1     1
```

It checks the elements of V1 so that they are greater than 2 but less than 6.

```
(V1>2) & (V1<6)
```

```
ans = 1×10 logical array
   0   0   1   1   1   0   0   0   0   0
```

It checks the elements of V1 so that they are either greater than 2 or less than 6.

```
(V1>2) | (V1<6)
```

```
ans = 1×10 logical array
   1   1   1   1   1   1   1   1   1   1
```

'any' function checks whether there is even a single non-zero element in the vector or matrix. If yes, then its returns a single 1. Otherwise 0.

```
any(V1)
```

```
ans = logical
   1
```

'all' function checks whether all elements in a vector or matrix are non-zero. If yes, then its returns a single 1. Otherwise 0.

```
all(V1)
```

```
ans = logical
   0
```