PHYS 361: Arrays and Matrices

Did you know that MATLAB stands for Matrix laboratory? It loves arrays and matrices!

Row vector, Column vector, and Matrices

Run the code below to see how you can build an array in row format, column format, or in multiple dimensions.

A 1 x 3 array (rows x columns) or row array:

```
clear all;

array1 = [3 4 7]
```

A 3 x 1 array or a column array:

```
array2 = [4; 8; 9]
```

A 2 x 3 array:

```
matrix1 = [1 2 3; 2 4 5]
```

Another way to define a 2 x 3 array:

```
matrix2 = [1 2 3
           2 4 5]
```

You can add arrays and matrices, but they have to be the same size!

This works:

```
clear all;

array1 = [3 4 7]
array2 = [4; 8; 9]
array2'

transpose(array2)

matrix1 = [1 2 3; 2 4 5]
matrix2 = [1 2 3
           2 4 5]
matrixsum=matrix1+matrix2
```

The code below doesn't, but you can fix it with a transpose.

```
arraysum=array1 + array2
arraysum=array1+array2'
```

Indexing Arrays and Matrics

You can refer to or change individual elements of an array or matrix with the following notation.

Just element 2:

```
clear all;

array1 = [3 4 7]
array2 = [4; 8; 9]

matrix1 = [1 2 3; 2 4 5]
matrix2 = [1 2 3
           2 4 5]
array1(2)
array2(2)
```

Element in row 1, column 2:

```
matrix1(1,2)
```

Replace element 1 with 10000:

```
array2 = [2; 4; 3]
array2(1)=10000*array2(1)
matrix2(2,2)=99
```

You can also examine or change whole rows, columns, or portions of a matrix.

Display the whole array:

```
clear all;

array1 = [3 4 7 10 9]
array2 = [4; 8; 9]

matrix1 = [1 2 3; 2 4 5]
matrix2 = [1 2 3
           2 4 5]

array1(3:5)
```

All of row 1:

```
matrix1(2,:)
```

All of column 2:

```
matrix1(:,2)
```

Replace all of column two with new values:

```
matrix1(:,2)=[100;100]
```

Create a random matrix:

```
randomatx=rand(6,6)
```

Look at a portion of it:

```
randomatx(3:5,4:6)
```

Creating evenly spaced arrays

You can use colon notation to create evenly spaced arrays.

1-10, spaced by 1:

```
clear all;

spacedbyone=3:10
```

1-10, spaced by 2:

```
spacedbytwo=1:2:10
```

Counting down:

```
reversed=10:-1:1
```

In two-dimensions

```
intwodim1=[10:-1:1; 1:10]
intwodim2=transpose([10:-1:1; 1:10])   %' takes the transpose
```

`Linspace`

If you want to specify then number of points over the range instead of the spacing, you can use the built-in function `linspace`.

To make an array that goes from 0 to 25 with 100 elements:

```
linexample=linspace(0,25,100)
```

Adding and Deleting Elements from Arrays and Matrices

The following code shows you how you can append data onto an array or matrix and remove elements of arrays or rows and columns of matrices.

Appened the new elements to slots 5:10:

```
clear all;

addtoarray=[1:4]'
addtoarray(5:10)=100:-1:95
```

Add a row:

```
add2matrix = [1 2 3 4; 5 6 7 8]
add2matrix(3,:)=[10:4:22]
```

Append two matrices:

```
newmat=eye(3)
addmattogether=[add2matrix newmat]
```

Delete element 6:

```
addtoarray
addtoarray(6)=[]
```

Delete column 3:

```
addmattogether
addmattogether(:,3)=[]
```

Element-by-Element Array Operations

THIS IS IMPORTANT! Sometimes you want to multiple or divide or exponentiate an array element by element. You can still do that, but you need to use **special notation (.* , .^ , ./) .**

$$a = \begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix}$$
$$b = \begin{pmatrix} b_1 & b_2 & b_3 \end{pmatrix}$$
$$a.*b = \begin{pmatrix} a_1 b_1 & a_2 b_2 & a_3 b_3 \end{pmatrix}$$
$$a./b = \begin{pmatrix} a_1/b_1 & a_2/b_2 & a_3/b_3 \end{pmatrix}$$
$$a.^b = \begin{pmatrix} a_1^{b_1} & a_2^{b_2} & a_3^{b_3} \end{pmatrix}$$

Examples:

```
clear all;

%Element-by-element multiplication, division, and exponentiation
A = [2 6 3; 5 8 4]
B = [1 4 10; 3 2 7]
A.*B
A./B
A.^B
```