Phys 361: Introduction to Array Assignment

Problem 1: Basic calculation with an array.

For the function $y = x^2 - e^{0.5x} + x$, calculate the value of y for the following values of x using an row array for x with the following elements: -3, -2, -1, 0, 1, 2, 3.

```
x = -3:3;
y = x.^2 - exp(1).^(0.5 .* x) + x
```

```
y = 1×7
    5.7769    1.6321   -0.6065   -1.0000    0.3513    3.2817    7.5183
```

Problem 2: Defining functions.

Exploring Built-in Functions for Arrays and Matrices

Many, dare I say most, built-in functions have no problem working with arrays and matrices. Depending on the function, it might act element-by-element or on the whole array. Below are a few examples. Run the code and replace the "What does.." line with some a short description of what the function does. Try to figure it out without looking at the help.

Declaring and shaping arrays:

What does linspace do?

Linspace creates a row vector starting at the first argument, ending at the second argument, with the number of entries specified by the third argument.

```
array1 = linspace(1,20,200)
```

```
array1 = 1×200
    1.0000    1.0955    1.1910    1.2864    1.3819    1.4774    1.5729    1.6683 ⋯
```

What does length do?

Length returns the number of entries in an array.

```
length(array1)
```

```
ans = 200
```

What does rand do?

rand generates an $m \times n$ matrix of random numbers.

```
randmat=rand(4,6)
```

```
randmat = 4×6
    0.8147    0.6324    0.9575    0.9572    0.4218    0.6557
    0.9058    0.0975    0.9649    0.4854    0.9157    0.0357
    0.1270    0.2785    0.1576    0.8003    0.7922    0.8491
    0.9134    0.5469    0.9706    0.1419    0.9595    0.9340
```

What does size do?

size returns the shape of an $m \times n$ array as a vector of two numbers $[m \ \ n]$.

```
size(randmat)
```

```
ans = 1×2
     4     6
```

What does reshape do?

Reshape takes input of an $m \times n$ array and returns a new $a \times b$ array. Reshape will preserve entries wherever possible.

```
reshape(randmat, 8,3)
```

```
ans = 8×3
    0.8147    0.9575    0.4218
    0.9058    0.9649    0.9157
    0.1270    0.1576    0.7922
    0.9134    0.9706    0.9595
    0.6324    0.9572    0.6557
    0.0975    0.4854    0.0357
    0.2785    0.8003    0.8491
    0.5469    0.1419    0.9340
```

What does diag do?

diag creates an appropriately sized matrix with entries specified by a vector argument.

```
diagval=[10:-2:4]
```

```
diagval = 1×4
    10     8     6     4
```

```
newdiagmat=diag(diagval)
```

```
newdiagmat = 4×4
    10     0     0     0
     0     8     0     0
     0     0     6     0
     0     0     0     4
```

What does zero do?

zeros creates an $m \times n$ array with all 0 entrys.

```
zeros(3,2)
```

```
ans = 3×2
     0     0
     0     0
     0     0
```

What does ones do?

ones creates an $m \times n$ array with all 1 entrys.

```
ones(3,2)
```

```
ans = 3×2
     1     1
     1     1
     1     1
```

What does eye do?

eye creates an identity matrix of size n.

```
eye(3)
```

```
ans = 3×3
     1     0     0
     0     1     0
     0     0     1
```

Basic Statistics:

How does mean work?

mean returns the average of the passed array. Mean defaults to taking the mean of each column and returning a row vector. This seems to disagree with the comments but looking at the output the row vector is 3 entries long and the matrix is a 2 by 3 matrix.

```
newrando=rand(2,3)   %Generate a test matrix
```

```
newrando = 2×3
    0.6787    0.7431    0.6555
    0.7577    0.3922    0.1712
```

```
mean(newrando)       %Defualts to taking the mean of row
```

```
ans = 1×3
    0.7182    0.5677    0.4133
```

```
mean(newrando,1)     %Mean of rows
```

```
ans = 1×3
    0.7182    0.5677    0.4133
```

```
mean(newrando,2)     %Mean of columns
```

```
ans = 2×1
    0.6924
    0.4404
```

```
mean(mean(newrando)) %Doing it twice will give you the mean of all values
```

```
ans = 0.5664
```

How does max work? (Note: min is a function that works in a similar way.)

max returns the maximum value of an array and the index of that value.

```
randarray=rand(1,6)'
```

```
randarray = 6×1
    0.7060
    0.0318
    0.2769
    0.0462
    0.0971
    0.8235
```

```
ans=max(randarray)
```

```
ans = 0.8235
```

```
[d,n]=max(randarray)   %Optional output format
```

```
d = 0.8235
n = 6
```

What does sum do?

sum returns the sum of all numbers in an array. If the array is a matrix then sum returns the sum of columns in the matrix.

```
sum(newrando)
```

```
ans = 1×3
    1.4365    1.1354    0.8267
```

```
sum(randarray)
```

```
ans = 1.9816
```

What does sort do?

sort sorts the entries of an array. The default behavior is to sort the entries least to greatest.

```
sort(randarray)
```

```
ans = 6×1
    0.0318
    0.0462
    0.0971
    0.2769
    0.7060
    0.8235
```

What does median do?

median returns the median of a list of numbers.

```
median(randarray)
```

```
ans = 0.1870
```

What does std do?

std returns the standard deviation of a list of numbers.

```
std(randarray)
```

```
ans = 0.3497
```

Advanced: What does det do?

det returns the determinant of a matrix.

```
squaremat=rand(3,3)
```

```
squaremat = 3×3
    0.6948    0.0344    0.7655
    0.3171    0.4387    0.7952
    0.9502    0.3816    0.1869
```

```
det(squaremat)
```

```
ans = -0.3564
```

Advanced: What do dot and cross do?

dot and cross return the dot and cross product of vectors respectively.

```
a=[1 2 3];
b=[3 4 5];
dot(a,b)
```

```
ans = 26
```

```
cross(a,b)
```

```
ans = 1×3
    -2     4    -2
```

Advanced: What does inv do?

inv returns the inversion of a matrix.

```
inv(squaremat)
```

```
ans = 3×3
    0.6213   -0.8015    0.8655
   -1.9539    1.6767    0.8692
    0.8303    0.6521   -0.8247
```

```
inv(squaremat)*squaremat
```

```
ans = 3×3
    1.0000    0.0000    0.0000
         0    1.0000   -0.0000
         0         0    1.0000
```

Note: There are many more useful functions than than what is included here. A pdf of useful functions can be found on the Canvas webpage.

Optional Advanced Exercises:

Exercise 1: Explore table arrays

Use the Help window or `doc table` to learn how to use the table command. Look up information on at least 4 elementary particles (name, charge, mass, spin, etc.)

Follow the example in the documentation and make a table containing the information you gathered.

```
% define the properties of four elementary particles as vectors
% electron-neutrino, muon-neutrino, tau-neutrino, sterile-neutrino (hypothetical)
% name = ["\nu_{\text{e}}" "\nu_{\mu}" "\nu_{\tau}" "\nu_\text{s}"]
name = ["electron-neutrino", "muon-neutrino", "tau-neutrino", "sterile-neutrino"];
% charge in units of the elementary charge e
charge = zeros(1,4);
% mass in units of eV/c^2
% since neutrino mass is not well measured mass is reported as the maximum theoretical
% the sterile neutrino is not well constrained so it will be random, for fun
mass = [1.0, 0.17, 18.2, 1*10.^(rand*15)];
spin = ones(1,4).*(1/2);

% define the variable names
var_names = ["flavour", "charge", "mass", "spin"]
```

```
var_names = 1×4 string
"flavour"    "charge"     "mass"       "spin"
```

```
% define the table
neutrino_table = table(name', charge', mass', spin', 'VariableNames', var_names)
```

```
neutrino_table = 4×4 table
```

	flavour	charge	mass	spin
1	"electron-neutrino"	0	1	0.5000
2	"muon-neutrino"	0	0.1700	0.5000
3	"tau-neutrino"	0	18.2000	0.5000
4	"sterile-neutrino"	0	3.2910e+08	0.5000

Exercise 2: Explore the cell data type

The cell data type allows you to collect lists of any type of data. For example, you can create a cell of data that contains both numbers and strings. Look up the documentation on `cell and cell2table` (or watch the short video on Canvas). Define a cell data type for elementary particles, storing the name, mass, charge, and spin for each partile. Next write some code to display the information in a table.

```
% neutrino cell defined such that columns refer to a particular neutrino
neutrino_cell = {
    "electron-neutrino", "muon-neutrino", "tau-neutrino", "sterile-neutrino";
    1, 1, 1, 1;
    1.0, 0.17, 18.2, 1*10.^(rand*15);
    1/2, 1/2, 1/2, 1/2};
```

```
cell2table(neutrino_cell, 'RowNames', var_names)
```

```
ans = 4×4 table
```

	neutrino_cell1	neutrino_cell2	neutrino_cell3	neutrino_cell4
1 flavour	"electron-neutrino"	"muon-neutrino"	"tau-neutrino"	"sterile-neutrino"
2 charge	"1"	"1"	"1"	"1"
3 mass	"1"	"0.17"	"18.2"	"13.7345"
4 spin	"0.5"	"0.5"	"0.5"	"0.5"

Exercise 3: Explore the structure data type

The `stucture` data type is another useful way to store data. Look up the documentation on `structures` and `cell2struct.` Define a `structure` data type for elementary particles, storing the name, mass, charge, and spin for each partile. Demonstrate your structure works as intended with some output.

```
neutrino_struct.name = name'
```

```
neutrino_struct = struct with fields:
      name: [4×1 string]
    charge: [4×1 double]
      mass: [4×1 double]
      spin: [4×1 double]
```

```
neutrino_struct.charge = charge'
```

```
neutrino_struct = struct with fields:
      name: [4×1 string]
    charge: [4×1 double]
      mass: [4×1 double]
      spin: [4×1 double]
```

```
neutrino_struct.mass = mass'
```

```
neutrino_struct = struct with fields:
      name: [4×1 string]
    charge: [4×1 double]
      mass: [4×1 double]
      spin: [4×1 double]
```

```
neutrino_struct.spin = spin'
```

```
neutrino_struct = struct with fields:
      name: [4×1 string]
    charge: [4×1 double]
      mass: [4×1 double]
      spin: [4×1 double]
```

```

struct2table(neutrino_struct)
```

```
ans = 4×4 table
```

	name	charge	mass	spin
1	"electron-neutrino"	0	1	0.5000
2	"muon-neutrino"	0	0.1700	0.5000

	name	charge	mass	spin
3	"tau-neutrino"	0	18.2000	0.5000
4	"sterile-neutrino"	0	3.2910e+08	0.5000