Arrays

- An **array** is a named collection of data (usually stored in contigious memory locations) with individual data elements being accessed via an **index** or **indices** (in the case two or more dimensions).
- 1D arrays are also called **vectors** and 2D arrays are sometimes referred to as **tables** or **matrices** (but a matrix can also be of higher dimension).
- A variable can be thought of as a 1D array with one element (and so the index need not be specified).
- 1D array example: `x=[2 4 6]` has 3 elements. `x(1)` is 2, `x(2)` is 4 and `x(3)` is 6.

- 2D array example: `y=[1 2 3; 4 5 6]` has 2 rows and 3 columns of data elements. `y(1,1)` is 1, `y(1,2)` is 2, `y(1,3)` is 3, `y(2,1)` is 4, `y(2,2)` is 5 and `y(2,3)` is 6.

- MatLab has both statically and dynamically allocated arrays. Static arrays are prefined before used (has computational efficiency benefits) while dynamic arrays are defined when using them.

```
>> x = [1 2 3]

x =

    1 2 3

>> y = [1; 2; 3]

y = 1

    2
```

```
     3

>> z=[2 1 0];

>> a = x+z

a =

    3  3  3

>> b=x+y

??? Error using ==> plus

Matrix dimensions must agree.


% You can multiply (or divide) the elements of two

% same-sized vectors term by term using array operators

% .*  or ./. Note that x.*x  and x.^2 both square each
```

```
% element of x.

% This is vectorization: [1 2 3].*[2 1 0]=[2 2 0]

% Vectorization is not matrix multiplication

>> a = x.*z

a =

   2  2  0

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

>> x=x.^2

x =

   1  4  9 16 25

>> b=2*a % 2*[2 2 0]

b =
```

```
   4  4  0

% Create a vector x with 5 elements linearly spaced

% between 0 and 10

>> x=linspace(0,10,5)

x =

   0 2.5000 5.0000 7.5000 10.0000

>> y = sin(x)

y =

         0    0.5985   -0.9589    0.9380   -0.5440

>> >> sqrt(x)

ans =

         0    1.5811    2.2361    2.7386    3.1623
```

```
>> z=sqrt(x).*y

z =

   0 0.9463 -2.1442 2.5688 -1.7203


% We can compute x^n, where x is a vector and
% n is an integer.
>> x=[1 2 3 4 5];
>> n=2;
>> x.^n
ans =
     1     4     9    16    25
```

```
% We can compute r^n, where scalar r is raised to
% the power of each element of n.
% Consider the series r^0+r^1+r+2+r^3+...+r^n.
% Note that r^0 is 1 and r^1 is r. To evaluate this
% sum for scalar r=0.5 we create a vector
% x=[1 r r^2 r^3 ... r^n] and then sum this vector.
% In the limit, as n --> infinity
% this limit (sum) approaches 1/(1-r), r < 1.
% For r=0.5 this limit is 2.0
>> n=0:10
n =
     0   1   2   3   4   5   6   7   8   9  10
```

```
>> r=0.5;
>> x=r.^n;
>> s1=sum(x);
>> s1 % not 2
s1 =
   1.9990
>> n=0:50;
>> x=r.^n;
>> s2=sum(x);
>> s2 % approximately 2
s2 =
   2.0000
```

```
>> n=0:100;

>> x=r.^n;

>> s3=sum(x);

>> s3 % much closer

s3 =

   2


>> % We can compute r^x and x^r, where both

>> % r and x are vectors.

>> r=[2 3 4]

r =

     2     3     4
```

```
>> x=[1 2 3]

x =

     1     2     3

>> r.^x

ans =


     2     9    64

>> x.^r

ans =

     1     8    81
```

- Arrays can be used statically or dynamically.

```
>> % set up a 1D array (row vector)
```

```
>> % and set its 6 elements
>> a(1)=1.0;
>> a(2)=2.0;
>> a(3)=3.0;
>> a(4)=4.0;
>> a(5)=5.0;
>> a(6)=6.0;
>> a % print the elements of 1
a=
     1     2     3     4     5     6
>> size(a) % print the size of a: 1 row with 6 elements
ans =
```

```
     1     6
>> % set up a 1D array by setting the last element
>> % all the elements before b(6) are set to zero
>> b(6)=6.0;
>> b
b =
     0     0     0     0     0     6
size(b)
ans =
     1     6
>> % extend b to have 16 elements by setting b(16)
>> % to 16: now b(6) and b(16) are set and all
```

```
>> % b elements have value 0
>> b(16)=16.0;
>> b
b =
 0  0  0  0  0  6  0  0  0  0  0  0  0  0  0 16
>> size(b)
ans =
     1    16
```

Structures and Cells

- **Structures** are like records with many fields of different types. If `S` is a structure variable then we might have:

```
S.char_stg = 'gauss';
S.matrix = [1 0; 0 1];
S.scalar = 3;


>> S.char_stg
ans =
    gauss
>> S.matrix
ans =
    1   0
    0   1
>> S.scalar
```

```
ans =

    3
```

- Of course, you can have arrays of structures/records.

- **Cells** are arrays where each element is of a different type. For example, one element could be a copy of a character vector, another a copy of a matrix and a third a copy of a scaler.

```
>> c = {'gauss', [1 0; 0 1], 3};
>> c{1}
ans =
    gauss
>> c{2}
```

```
ans =

     1   0

     0   1

>> c{3}

ans =

     3
```