PROGRAMMING WITH MATLAB

WEEK 3

DATA TYPES - 2

ARRAY OPERATIONS

- Addition and substraction:

```
>> X = [-3, 7, 6; 2, 8, -5; 1, 4, -2]
X =
   -3    7    6
    2    8   -5
    1    4   -2
>> Y = [2, -3, 5; 3, -9, 1; -2, 2, -3]
Y =
    2   -3    5
    3   -9    1
   -2    2   -3
>> Z = X + Y
Z =
   -1    4   11
    5   -1   -4
   -1    6   -5
>> Z = X - Y
Z =
   -5   10    1
   -1   17   -6
    3    2    1
```

ARRAY OPERATIONS

- Multiplication of an array by a scalar:

```
>> X = [-3, 7, 6; 2, 8, -5; 1, 4, -2]
>> A = 5*X
A =
   -15    35    30
    10    40   -25
     5    20   -10
```

ARRAY OPERATIONS

- The built-in MATLAB functions:

```
>> x = [4, 36, 64]
x =
    4    36    64
>> sqrt(x)
ans =
    2     6     8
>> y = [1, 2, 3]
y =
    1     2     3
>> exp(y)
ans =
   2.7183    7.3891   20.0855
```

ARRAY OPERATIONS

- The built-in MATLAB functions:

```
>> x = [4, 36, 64]
x =
    4    36    64
>> sqrt(x)
ans =
    2     6     8
>> y = [1, 2, 3]
y =
    1     2     3
>> exp(y)
ans =
   2.7183    7.3891   20.0855
```

But

```
>> z = exp(y).*cos(x)
z =
  -1.7768   -0.9455    7.8707
```

ARRAY OPERATIONS

- Array division:

```
>> x = [4, 36, 64]
x =
    4    36    64
>> y = [1, 2, 3]
y =
    1     2     3
>> z = x./y
z =
   4.0000   18.0000   21.3333
>> X = [-3, 7, 6; 2, 8, -5; 1, 4, -2]
X =
   -3     7     6
    2     8    -5
    1     4    -2
>> Y = [2, -3, 5; 3, -9, 1; -2, 2, -3]
Y =
    2    -3     5
    3    -9     1
   -2     2    -3
>> Z = X./Y
Z =
  -1.5000   -2.3333    1.2000
   0.6667   -0.8889   -5.0000
  -0.5000    2.0000    0.6667
```

ARRAY OPERATIONS

- Array exponentiation:

Syntax: x.^y

```
>> x = [1.2, 3, 5]

x =

    1.2000    3.0000    5.0000

>> x.^3

ans =

    1.7280   27.0000  125.0000

>> 3.^x

ans =

    3.7372   27.0000  243.0000
```

ARRAY OPERATIONS

- Matrix multiplication:

Syntax: x*y (the number of columns in x must equal the number of rows in y)

```
>> x = [4, 36, 64]
x =
    4    36    64
>> y = [2; -3; 5]
y =
    2
   -3
    5
>> z = x*y
z =
  220
>> X = [-3, 7, 6; 2, 8, -5]
X =
   -3    7    6
    2    8   -5
>> Y = [2, -3, 5; 3, -9, 1; -2, 2, -3]
Y =
    2   -3    5
    3   -9    1
   -2    2   -3
>> Z = X*Y
Z =
    3  -42  -26
   38  -88   33
```

ARRAY OPERATIONS

- Polynomial Multiplication and Division:

Syntax: conv(x,y) (computes the product of the two polynomials described by the coefficient arrays x and y)

```
>> x = [4, -3, 2.1]
x =
    4.0000   -3.0000    2.1000
>> y = [1, 2, 3, 4]
y =
     1     2     3     4
>> conv(x,y)
ans =
    4.0000    5.0000    8.1000   11.2000   -5.7000    8.4000
```

Syntax: [q,r] = deconv(b,a) (The result of dividing b by a is quotient q and remainder r)

CELL ARRAYS

- cell: Creates cell array.

Syntax: cell(N) is an N-by-N cell array of empty matrices.

cell(M,N) or cell([M,N]) is an M-by-N cell array of empty matrices.

cell(M,N,P,...) or cell([M N P ...]) is an M-by-N-by-P-by-... cell array of empty matrices.

cell(size(A)) is a cell array the same size as A containing all empty matrices:

When you have data to put into a cell array, create the array using the cell array construction operator, {}:

```
>> myCell = {1, 2, 3; 'Cemil', [1, 2, 3;4, 5, 6], {11; 22; 33}}
myCell =
  2×3 cell array
    [    1]    [         2]    [       3]
    'Cemil'    [2×3 double]    {3×1 cell}
```

CELL ARRAYS

- Accessing data in a cell array:

There are two ways to refer to the elements of a cell array. Enclose indices in smooth parentheses, (), to refer to sets of cells:

```
>> C = {'Sunday', 'Monday', 'Tuesday';
     1, 2, 3}
C =
  2×3 cell array
    'Sunday'    'Monday'    'Tuesday'
    [     1]    [     2]    [     3]
>> Left = C(1:2,1)
Left =
  2×1 cell array
    'Sunday'
    [     1]
```

Enclose indices in curly braces, {}, to refer to the text, numbers, or other data within individual cells.

```
>> last = C{2,3}
last =
     3
```

STRUCTURE ARRAYS

- struct: creates structure array

Syntax: s = struct (creates a scalar (1-by-1) structure with no fields)

s = struct(field,value), creates a structure array with the specified field and values. The value input argument can be any data type, such as a numeric, logical, character, or cell array

```
>> field = 'f';
value = {'some text';
         [10, 20, 30];
         ones(5)};
s = struct(field,value)
s =
  3×1 struct array with fields:
    f
>> field1 = 'f1';  value1 = zeros(1,10);
field2 = 'f2';  value2 = {'a', 'b'};
>> s = struct(field1,value1,field2,value2)
s =
  1×2 struct array with fields:
    f1
    f2
```

STRUCTURE ARRAYS

- Accessing elements in a struct array

```
>> s.f2
ans =
a
ans =
b
```