Data Types in MATLAB

Lecture 7

Dr. Görkem Saygılı

Department of Biomedical Engineering
Ankara University

Introduction to MATLAB, 2017-2018 Spring

Outline:

- Numerical Data
- Categorical Data
- Struct
- Cell
- built-in functions for different data types

Numerical Data:

Numerical data is composed of numbers that can be used in mathematical calculations.

Some examples are:

- int8
- uint8
- int16
- uint16
- int32
- uint32
- int64
- uint64
- single
- double

Difference between Data Types:

There are several differences between the numerical data types. Some of the main differences are:

- The size of memory to be used to store them.
- The minimum and maximum (the range) of numbers that can be stored.
- Whether or not we can store floating point numbers.

Single-Double Precision:

Single precision corresponds to float in C/C++ where MATLAB uses 32 bits to store the number in its memory.

Double precision corresponds to double in C/C++ where MATLAB uses 64 bits to store the number in its memory.

single precision: -3.7×10^38 to 3.7×10^38

double precision: -1.79×10^308 to 1.79×10^308

Checking the Type of the Data:

There is a built-in function in MATLAB to check the type of data: class().

```
>> x=5;
>> class(x)

ans =

double

>> y = 'Matlab';
>> class(y)

ans =

char

>>
```

Mixed-mode Arithmetic:

While calculations are done on the same variable type, such as multiplying two doubles, the result is also a variable of the same type, double.

MATLAB also allows calculations on different variable types, such as one uint8 multiplied with a double. In such cases, the resulting variable is the same type with the smallest-sized input variable (multiplication of uint8 with double results uint8).

This is for memory efficiency.

Variable Casting - To Smaller Size:

In MATLAB, it is possible to convert a variable from one type to another. This process is called variable casting. This is not specific for MATLAB, you can also cast variables in C/C++.

```
>> x = 5.5;
>> class(x)

ans =

    'double'

>> b = uint8(x);
>> class(b)

ans =

    'uint8'

>> disp(b)
   6
```

Variable Casting - To Larger Size:

There are no issues while casting a variable to a larger-sized type. The only difference is that now MATLAB uses a larger memory size to store the casted variable.

```
>> x=5;
>> x = uint8(x);
>> class(x)

ans =

    'uint8'

>> y = single(x);
>> class(y)

ans =

    'single'
```

String Type:

All data types are not related to numeric values in MATLAB. A different type of data in terms of content is char arrays, strings.

```
>> s = 'Matlab';
>> t = "MATLAB";
>> class(s)

ans =

    'char'

>> class(t)

ans =

    'string'
```

Comparing Strings:

In MATLAB, there is a built-in function to compare strings, strcmp():

```
>> t = "MATLAB";
>> strcmp(t, 'matlab')

ans =

  logical

   0

>> strcmp(t, 'MATLAB')

ans =

  logical

   1
```

Char Arrays:

Character arrays are similar to strings and also have similar characteristics as arrays (vectors). Character arrays have a length and can be modified similar to arrays.

```
>> s = 'Matlab';
>> class(s)

ans =

    'char'

>> length(s)

ans =

     6

>> s(1) = 'm';
>> s

s =

    'matlab'
```

Cascading Char Arrays:

Similar to cascading for numerical arrays, MATLAB allows us to cascade char arrays:

```
>> s1 = 'Matlab';
>> s2 = ' is fun';
>> s = [s1, s2];
>> disp(s)
Matlab is fun
>>
```

ASCII Encoding Scheme:

Strings of type char are stored as numerical values using corresponding ASCII codes for that specific character:

```
>> disp(s)
Matlab is fun
>> disp(class(s))
char
>> disp(int8(s))
  Columns 1 through 7

    77    97   116   108    97    98    32

  Columns 8 through 13

   105   115    32   102   117   110
```

Ceasar's Cipher:

As perhaps one of the simplest encryption techniques today has its roots from the time of Julius Ceasar. This technique relies on shifting each character of the original string by a predefined amount (known only by trusted people).

Original Characters: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher Characters: XYZABCDEFGHIJKLMNOPQRSTUVW

Arithmetic Operations on ASCII Characters:

MATLAB allows us to apply arithmetic operations on character arrays:

```
>> disp(s)
Matlab is fun
>> s_ = s+3;
>> disp(char(s_))
Pdwode#lv#ixq
>> s_ = s_-3;
>> disp(char(s_))
Matlab is fun
>>
```

Arrays with Unique Data Types:

Normally MATLAB arrays are composed of unique types:

```
>> x = [3 2 4 5 3 2];
>> x(1) = 'Matlab'
In an assignment  A(:) = B, the number of
elements in A and B must be the same.

>> x(2) = 'M'

x =

     3    77     4     5     3     2

>> x(2) = "M"

x =

     3   NaN     4     5     3     2

>>
```

Struct:

MATLAB allows us to store different data types in one data format, which is called struct.

```
>> s = struct()

s =

  struct with no fields.

>> s.x = x

s =

  struct with fields:

    x: [3 NaN 4 5 3 2]

>> s.t = t

s =

  struct with fields:

    x: [3 NaN 4 5 3 2]
    t: "MATLAB"
```

Dynamically Assigning Values to Struct:

Rather than assigning values statically, we can also create elements of struct dynamically using the following format:

```
>> st.b = [3, 4; 5 6];
>> a = 'st_a';
>> st.(a) = a

st =

  struct with fields:

      b: [2×2 double]
   st_a: 'st_a'
```

Built-in Function: setfield():

struct data can be created with struct command and its content can be created using setfield() built-in function:

```
>> b = st.b;
>> st_a = st.st_a;
>> st2 = struct('b', b);
>> st2=setfield(st2, st_a, st_a)

st2 =

  struct with fields:

      b: [2×2 double]
    st_a: 'st_a'
```

Cell Data Type:

Elements of struct data is accessed by their names. In comparison, we can create arrays inside arrays using cell data type in MATLAB:

```
>> A = {logical(1), 'Matlab'; int8(7),...
[5 4; 2 1]};
>> disp(A)
    [1]    'Matlab'
    [7]    [2×2 double]

>> A{1,1}

ans =

  logical

   1

>> A{1,2}

ans =

    'Matlab'
```