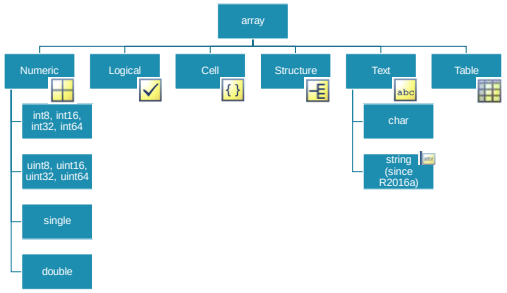
KU LEUVEN

MATLAB

Fundamental Data Types: basic

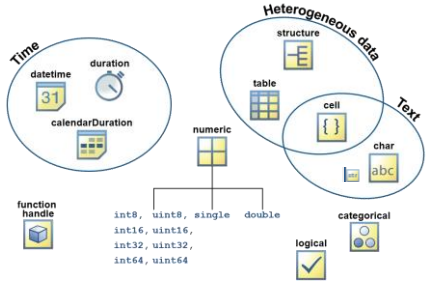
1

Overview data types (classes)



KU LEUVEN

3



Taken from MATLABacademy

KU LEUVEN

4

Data types

- By default, MATLAB treats everything as a matrix of double-precision floating-point numbers.
- Other kinds of data:
  - character strings,
  - heterogeneous lists whose items are different kinds of data,
  - etc.
- `help datatypes`

KU LEUVEN

5

Numeric data types

Class Name	Documentation	Intended Use
`double`, `single`	Floating-Point Numbers	•Required for fractional numeric data. •Double and Single precision. •Use realmin and realmax to show range of values. •Two-dimensional arrays can be sparse. •Default numeric type
`int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`	Integers	•Use for signed and unsigned whole numbers. •More efficient use of memory / speed. •Use intmin and intmax to show range of values. •Choose from 4 sizes (8, 16, 32, and 64 bits).

KU LEUVEN

6

Default data type

- Default data type: **double precision array**
- Automatically allocates required memory
- Arrays are resized dynamically
- Reuse a variable name, assigning a new value in an assignment statement
- FORTRAN roots:
  - Base-1 indexing
  - Column wise storage
- MATLAB displays results in scientific notation
  - Use **Home>Environment>Preferences** and/or **format** function to change default
  - format short (.4 digits), format long (.15 digits)

KU LEUVEN

7

floating point: double

- MATLAB represents floating-point numbers in:
  - double-precision (default)
  - single-precision format.
- Maximum and Minimum Double-Precision Values:
  - `realmax` and `realmin` return the maximum and minimum values that you can represent with the double data type
  - conversion function: `double`
- *File: demo_float*

KU LEUVEN

8

floating point: single

- MATLAB constructs the single data type according to IEEE Standard 754 for single precision. Maximum and Minimum Double-Precision Values:
  - `realmax('single')` and `realmin('single')` return the maximum and minimum values that you can represent with the double data type
  - conversion function: `single`

```
>> as = single(10)

as =

  single

    10
```

KU LEUVEN

10

Advanced: double or single?

- Most applications use double precision math for the following reasons:
  - To minimize the accumulation of round-off error,
  - For ill-conditioned problems that require higher precision,
  - The 8 bit exponent defined by the IEEE floating point standard for 32-bit arithmetic will not accommodate the calculation, or
  - There are critical sections in the code which require higher precision.
- *File: demo_single_double.m*
- http://www.hpcwire.com/hpcwire/2006-06-16/less_is_more_exploiting_single_precision_math_in_hpc-1.html

KU LEUVEN

11

integer

- 4 signed and 4 unsigned integer data types.
- Signed types:
  - enable negative integers as well as positive,
- Unsigned types
  - wider range of numbers
  - zero or positive.
- MATLAB supports 1-, 2-, 4-, and 8-byte storage for integer data.
  - save memory and execution time for your programs if you use the smallest integer type that accommodates your data.

KU LEUVEN

12

integer

Class	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to 2^7-1	`int8`
Signed 16-bit integer	-2^{15} to $2^{15}-1$	`int16`
Signed 32-bit integer	-2^{31} to $2^{31}-1$	`int32`
Signed 64-bit integer	-2^{63} to $2^{63}-1$	`int64`
Unsigned 8-bit integer	0 to 2^8-1	`uint8`
Unsigned 16-bit integer	0 to $2^{16}-1$	`uint16`
Unsigned 32-bit integer	0 to $2^{32}-1$	`uint32`
Unsigned 64-bit integer	0 to $2^{64}-1$	`uint64`

KU LEUVEN

13

integer

- MATLAB stores numeric data as double-precision floating point by default.
- Store data as an integer: use one of the conversion functions
  `x = int16(32501);`
- Use the `whos` function to show the dimensions, byte count, and data type of an array represented by a variable.
- ex. – *PAY ATTENTION!*

```
>> x = int8(60);
>> y = int8(70);
>> z = x + y

z =

  int8

   127
```

- *File: demo_integer*

KU LEUVEN

14

complex

- Complex numbers consist of two separate parts: a real part and an imaginary part.
- The basic imaginary unit is equal to the square root of -1.
  represented by: i or j
  ```
  x = 2 + 3i;
  ```
- Another way – use the `complex` function.
  ```
  x = rand(3) * 5;
  y = rand(3) * -8;
  z = complex(x, y)
  ```
- Separate a complex number into its real and imaginary parts using the real and imag functions:
  ```
  zr = real(z);
  zi = imag(z);
  ```

KU LEUVEN

15

Inf and NaN

- **Infinity**
  - special value `Inf`.
  - results from operations like division by zero and overflow.
  - Use the `isinf` function to verify that x is positive or negative infinity!
- **NaN**
  - values that are not real or complex numbers
  - NaN: Not a Number.
  - 0/0 and inf/inf result in NaN
  - Use the `isnan` function to verify that x is NaN!
  - Logical Operations on NaN.
    Because two NaNs are not equal to each other, logical operations involving NaN always return false, except for a test for inequality

```
x = 1/0
x =
     Inf
x = 1.e1000
x =
     Inf
x = exp(1000)
x =
     Inf
x = log(0)
x =
    -Inf

x = log(0); isinf(x)
ans = 1

x = 7i/0
x = NaN + Infi

x = log(0); isnan(x) ans = 1
```

KU LEUVEN

16

Logical data type

Class Name	Documentation	Intended Use
`logical`	Logical Operations	•Use in relational conditions or to test state. •Can have one of two values: true or false. •Also useful in array indexing. •Two-dimensional arrays can be sparse

KU LEUVEN

17

Logical data

- Logical arrays are usually the result of logical operations
  ```
  M = [true false true]
  M =
       1     0     1
  x = 1:5;
  y = [2 0 1 9 4];
  z = x>y
  z =
       0     1     1     0     1
  ```
- conversion function: `logical`
- Logical values can be used to extract data
- *File: demo_logical*

KU LEUVEN

18

Cell array

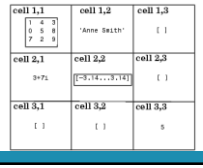
Class Name	Documentation	Intended Use
`cell`	Cell Arrays	• Cells store arrays of varying classes and sizes. • Allows freedom to package data as you want. • Manipulation of elements is similar to numeric or logical arrays. • Method of passing function arguments. • Use in comma-separated lists. • More memory required for overhead

KU LEUVEN

19

Cell Array

- Most *general* MATLAB data structure: '*spreadsheet*'
- Provides a storage mechanism for *dissimilar* kinds of data, for any type of data.
- A cell array is just like a matrix except each entry can be any data type, not just a number.



KU LEUVEN

20

Cell Array

- Cell arrays are created in the same way that data in an array is created and referenced, difference is the use of curly braces { }.
- Cell arrays are used by a lot of built in functions (ie `textscan`, …) and can be particularly useful within scripts.
- Cell arrays should be considered more as data "containers" and must be manipulated accordingly. *(Be careful with the notation when performing arithmetic computations like arrays can, e.g., + - * / ^)*

KU LEUVEN

21

Cell Array: Cell indexing ()

- Cell indexing allows you to access and manipulate the cells themselves. When accessing the cells themselves, you ignore the content of the cells and merely manipulate the cells.
- Enclose the cell subscripts in parentheses using standard array notation. Enclose the cell contents on the right side of the assignment statement in curly braces {}.

```
A(1,1) = {[1 4 3; 0 5 8; 7 2 9]};
A(1,2) = {'Anne Smith'};
A(2,1) = {3+7i};
A(2,2) = {-pi:pi/10:pi};
class(A(1,1))
ans =
    'cell'
```

KU LEUVEN

23

Cell Array: Content addressing {}

- Enclose the cell subscripts in curly braces using standard array notation. Specify the cell contents on the right side of the assignment statement:

```
A{1,1} = [1 4 3; 0 5 8; 7 2 9];
A{1,2} = 'Anne Smith';
A{2,1} = 3+7i;
A{2,2} = -pi:pi/10:pi;
class(A{1,1})
ans =
    'double'
```

KU LEUVEN

24

Creating Cell Arrays: {}

3 ways:

- using {} directly: {row stuff ; more row stuff ; etc }
  braces {} as cell constructors:

```
C = {'Jan',10,[1,2,3,4,5], [6, 7; 8, 9]}
C = {'Jan',10;[1,2,3,4,5], [6, 7; 8, 9]}
```

- cell indexing: array(indices) = {stuff}
  `C(i,j)={...}`
- content adressing: array{indices} = stuff
  `C{i,j}= ...`

all methods identical for results!

KU LEUVEN

25

Cell Array: Preallocation

- cell command:

`cell(m)`: m * m cell array

`cell(m, n)`: m * n cell array

```
D = cell(3);
```

once the cell array is created, assignment statement can be used to fill values into the cells

```
D{2,1}=1;
D{3,3}=[1, 2; 2, 6];
```

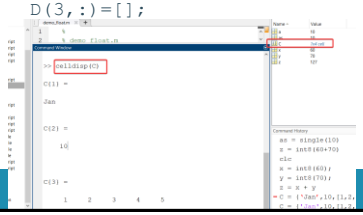
- *File: cell_ex_o.m*

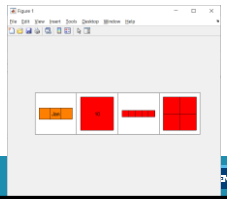
KU LEUVEN

26

Cell: Specific Commands

- `celldisp`: returns the content of a cell array
- `cellplot`: returns graphically the structure of a cell array
- extending a cell array: just by adding
- deleting elements from a cell array: assignment of an empty array
  `D(3,:)=[];`



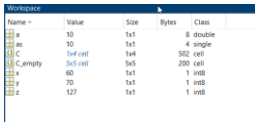


KU LEUVEN

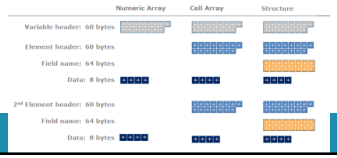
27

Memory requirements

- Cell arrays consume more memory!
- `C = {'Jan',10;[1,2,3,4,5], [6, 7; 8, 9]}`
- `C_empty = cell(5)`
- Check storage requirements for Ce



Container overhead



KU LEUVEN

28

Why Cell Array?

- **String arrays** hold text, not numbers.
  Avoid using string arrays! each element of a string array must be the same length.
  strings (a='hello' and b='bye') and try to put them into a string array (c=[a; b]) error because 'a' and 'b' are not the same length.
  The solution? Use cell arrays c = {a;b}
- necessary for lots of **MATLAB operations**. For example, most types of input to a program from the keyboard come into a cell array (so the input can be either a number or a string).

  demo_cell_array_textscan.m
- main thing: create them and 'unpack' them by using curly braces.

KU LEUVEN

30

Example

- *File: demo_cell_array_textscan_1.m*
- *File: cell_ex_1.m*

Type of Array	*Example*	*Stores*	*Might hold*
numeric scalar	nc	Number of Compounds	4
string matrix	cnms	Compound Names	ammonia nitrogen hydrogen argon
numeric vector	mw	molecular weights	17.03 28.013 2.016 39.948
numeric matrix	Aabc	Antoine Constants	15.494 13.45 12.78 13.915 2363.20 658.22 232.320 832.78 -22.62 -2.854 8.08 2.36

KU LEUVEN

31

Have a look at

- http://blogs.mathworks.com/loren/2006/06/21/cell-arrays-and-their-contents/

KU LEUVEN

33

text

Class Name	Documentation	Intended Use
`char`, `string`	Characters and Strings	• Data type for text. • Native or Unicode®. • Converts to/from numeric. • Use with regular expressions. • For multiple character arrays, use cell arrays. • Starting in R2017a, you also can store text in string arrays.

KU LEUVEN

34

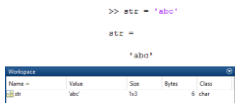
Text

- **character array** is a sequence of characters, just as a numeric array is a sequence of numbers.
  - A typical use is to store **short** pieces of text as character vectors.
  - `c = 'Hello World'`
- **string array** is a container for pieces of text.
  - String arrays provide a set of functions for working with text as data.
  - Starting in R2017a
  - `str = "Greetings friend"`

KU LEUVEN

35

Character arrays



- surrounded by single quotes
  - `str = 'abc'`

    is equivalent to str = ['a' 'b' 'c']
- each character requires 2 bytes
- All operations that apply to vectors and arrays can be used together with strings as well

```
» str(1) → 'a'
» str( [ 1 2 ] ) = 'XX' → s = 'XXc'
» str(end) → 'c'
```

KU LEUVEN

36

Character arrays

- Character arrays can be manipulated like numerical arrays.

```
» T = 'How about this character string?'
» u = T(16:24)
» u = T(24:-1:16)
» u = T(16:24)'
» v = 'I can''t find the manual!'  %  Note quote in string
» u ='If a woodchuck could chuck wood,';
» v = 'how much wood could a woodchuck chuck?';
» disp(u)   % works just like for arrays
```

KU LEUVEN

37

Character array conversion

- Conversion to numerical arrays: `double / int16`

```
» double( 'abc xyz' )
» ans =
     97    98    99    32   120   121   122
» double( 'ABC XYZ' )
» ans =
     65    66    67    32    88    89    90
```

- Conversion character array: `char`

```
» char( [ 72 101 108 108 111 33 ] )
» ans =
  Hello!
```

KU LEUVEN

38

Character Arrays

- Horizontal concatenation
  - `s1 = 'hello'`
  - `s2 = 'world'`
  - `s = [s1, ' ', s2]`
- Vertical concatenation (2-D character arrays)
  - `» s = [ 'my first string'; 'my second string' ]`
    `??? Error`
  - `» s = char( 'my first string', 'my second string' )`
    `» s =`
    `my first string`
    `my second string`
  - `» size(s)` → `[ 2 16 ]`
  - `» size( deblank( s(1,:) ) )` → `[ 1 15 ]`
  - char function automatically pads blancs

KU LEUVEN

39

Character Arrays

- You can use char to hold an m-by-n array of strings as long as each string in the array has the same length. (MATLAB arrays must be rectangular.)
- character arrays in multiple rows: must have the same number of columns as a matrix, use **`char`**

```
group = char('Jan', 'Arnoud',...
'Karel', 'Paul')
group1 = ['Jan    '; 'Arnoud';...
'Karel ', 'Paul  '];
```

- Advice: to hold an array of strings of unequal length, use a *cell array or string array*.
- *File: demo_chararray.m*

KU LEUVEN

40

Tests

- `ischar()` : returns 1 for a character array
  - `» ischar ( 'LU 1111' )`
    `ans =`
    `1`
- `isletter()` : returns 1 for letters of the alphabet
  - `» isletter( 'LU 1111' )`
    `ans =`
    `1  1  0  0  0  0  0`
- `isspace()` : returns 1 for whitespace (blank, tab, new line)
  - `» isspace( 'LU 1111' )`
    `ans =`
    `0  0  1  0  0  0  0`

KU LEUVEN

41

Comparison

- Comparing two characters
  - `» 'a' < 'e'`
    ```
    ans =
         1
    ```
- Comparing two arrays character by character
  - `» 'fate' == 'cake'`
    ```
    ans =
         0     1     0     1
    ```
  - `» 'fate' > 'cake'`
    ```
    ans =
         1     0     1     0
    ```

KU LEUVEN

42

String Comparison

- `strcmp()` : returns 1 if two strings are identical
  - `» a = 'ICTS';`
  - `» strcmp( a, 'ICTS' )`
    ```
    ans =
         1
    ```
  - `» strcmp( 'Hello', 'hello' )`
    ```
    ans =
         0
    ```
- `strcmpi()` : returns 1 if two strings are identical ignoring case
  - `» strcmpi( 'Hello', 'hello' )`
    ```
    ans =
         1
    ```

KU LEUVEN

43

String Case Conversion

- Lowercase-to-uppercase
  - `» a = upper( 'This is test 1!' )`
    ```
    a =
    THIS IS TEST 1!
    ```
- Uppercase-to-lowercase
  - `» a = lower( 'This is test 1!' )`
    ```
    a =
    this is test 1!
    ```

KU LEUVEN

44

Replacing in Strings

- `strrep()` : replaces one string with another
  - `s1 = 'This is a good example';`
  - `s2 = strrep( s1, 'good', 'great' )`
    ```
    s2 =
    This is a great example
    ```

KU LEUVEN

47

String Conversion

- `num2str()` for numeric-to-string conversion
  - `str = [ 'Plot for x = ' num2str( 10.3 ) ]`

```
str =
Plot for x = 10.3
```

- `str2num()` : converts strings containing numbers to numeric form
  - `x = str2num( '3.1415' )`

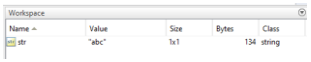
```
x =
    3.1415
```

KU LEUVEN

48

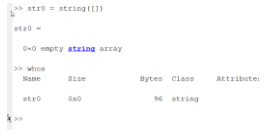
String arrays

- *File: demo_stringarrays.mlx*
- surrounded by double quotes
  - `str = "abc"`



- Concatenation

```
str1 = "hello"
str2 = "MATLAB"
sv = [str1 ; str2]
sh = [str1 , str2]
```



http://blogs.mathworks.com/loren/2016/12/22/singing-the-praises-of-strings/
http://blogs.mathworks.com/loren/2017/04/24/working-with-text-in-MATLAB/

KU LEUVEN

50

String arrays

- No individual characters in string array, treatment similar to cell arrays

```
str_a = ["Mercury","Gemini","Apollo";...
       "Skylab","Skylab B","ISS"]
stra1 = str_a(2,2)
stra2 = str_a(2,:)
stra1_c = str_a{2,2}
```

- MATLAB provides a set of functions to work with string arrays. For example, you can use the `split`, `join`, and `sort` functions

52

KU LEUVEN

52

Structure data type

Class Name	Documentation	Intended Use
`struct`	Structures	• Fields store arrays of varying classes and sizes. • Access one or all fields/indices in single operation. • Field names identify contents. • Method of passing function arguments. • Use in comma-separated lists. • More memory required for overhead

KU LEUVEN

54

Structure array

- Structures can store different types of data similar to cell arrays, but the data is stored by **name**, `fields` , rather than by index (hierarchy)
- Structures are similar to structures in C

```
A = 1:3;
B = ['abcdefg'];
C = single([1, 2, 3; 4, 5, 6]);
my_struct.numbers = A
my_struct.letters = B
my_struct.singlenumbers = C
```

KU LEUVEN

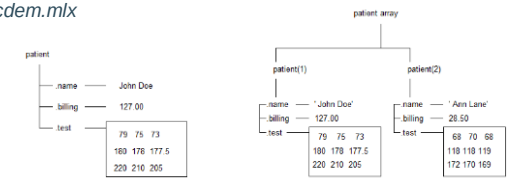
55

Structure array

- Can create structure array

```
my_struct(2).numbers = [2 3 6 8 9 10]
```

- *File: strucdem.mlx*

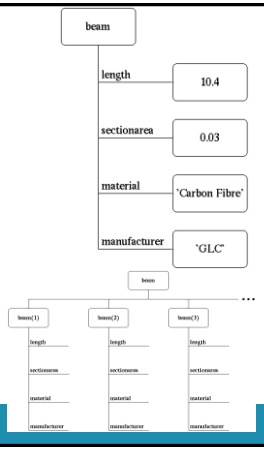


KU LEUVEN

56

Structures

- structures are inherently array oriented.
- Contents is addressable by name
- To access these fields, the dot "." notation is used.
- Each element of a structure can be of a different data type
- Multiple instances of a single structure: build up an array of structures.



KU LEUVEN

57

Structures

- Creating structures?
  1. a field at a time by assignment
  2. `struct` function
- Assignment

```
beam.length = 10,4;
beam.sectionarea = 0,03;
beam.material = 'Carbon Fibre';
beam.manufacturer = 'GLC';
beam(2).length = 15,48;
beam(2).sectionarea = 0,73;
beam(2).material = 'Steel';
beam(2).manufacturer = 'GLC';
```

- *File: structure_what.m*

KU LEUVEN

58

Structures

- preallocation using **`struct`**
- basic form is
  `strArray = struct('field1',val1,'field2',val2, ...)`
- where the arguments are field names and their corresponding values. A field value can be a single value, represented by any MATLAB data construct, or a cell array of values.
  `beam =struct('length', {}, 'sectionarea', {}, 'material',{}, 'manufacturer',{})`
- *Filles: struct_ex_1.m, struct_ex_3.m, struct_ex_4.m*

KU LEUVEN

59

Accessing

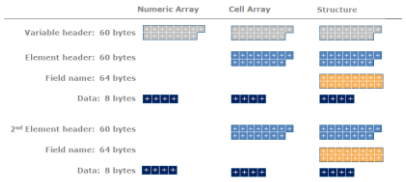
- Access the contents of the fields by typing
  `VariableName.FieldName`
- we can do
  `student.name`
  `student.street`
  `student.code`
- `student.code` is 1-by-4 double array.
  - access its 1st element.
    `student(1).code`
  - access its last element.
    `student(end).code`

KU LEUVEN

60

Memory requirements

Container overhead



KU LEUVEN

61

Structures

File: demo_structure_textscan_1.m

operations:

- **`rmfield`**: remove a field from a structure
  struct_new = rmfield(struct_old, 'veld')
- **`getfield`**: retrieving a value from a field
- **`setfield`**: putting a value in a field
- **`fieldnames`**: returns a list of fieldnames in a cell array of strings

File: struct_ex_2.m

KU LEUVEN

63