Matlab: What is it?

- Computing & programming environment
  - High-level, interpreted language
    - Rapid prototyping & integrated debugging
    - Built-in graphics & visualization tools
    - Built-in toolboxes for many applications

- Compared to other common languages
  - C/C++ is a low-level, compiled language
    - Better performance
    - Harder to write, difficult data visualization, harder to employ toolboxes, longer codes, less portable
  - Python
    - Somewhere in between

Matlab: Why learn it?

- Computation and data analysis are two fundamental components of virtually any discipline.
  - Math
  - Science
  - Engineering
  - Humanities
  - Social Sciences
- A little bit of time upfront to write a (good) Matlab code can yield a long-term tool to iteratively test & debug algorithms, explore & process data, automate calculations & analysis, visualize results, etc.

Lecture 1 Topics

- Basic concepts
  - Variables, expressions, & assignments
  - Operators
  - Built-in functions
  - Good programming practices
  - Data types
- Vectors and matrices
- Strings and cell arrays
- Getting help

Basic concepts

A variable is any symbolic representation for a "value"
An assignment gives the variable its value.

An expression is a mathematical statement which evaluates to a value

x, y, and z are **variables**

10 and 20 are values
the **expression** (x+y) is evaluated to a value (30)

Working on the command line

```
EDU>> x = 10;
EDU>> y = 20;
EDU>> z = x+y;
EDU>> z

z =

    30
```

Assignments can use values, previously defined variables, operators, functions, and parentheses

A semicolon at the end of a line represses output; no semicolon displays the evaluated result of the variable (or expression) on the line

Basic concepts

- Operators:

Addition: +	Subtraction: –
Multiplication: *	Division: /
Exponentiation: ^	

- Order of operations (including parentheses) as usual

- Common built in functions:
  - Trig: `sin(), cos(), tan(), sind(), cosd(), atan()`
  - Exponentials: `exp(), log()`
  - Complex: `abs(), conj(), imag(), real()`
  - Rounding: `round(), floor(), ceil(), mod()`

Basic concepts

- Variable names
  - Must begin with a letter, but can contain letters, numbers, and underscores
    - Apart from the underscore, no other special characters (including spaces) are allowed.
  - Case sensitive
  - Length limited (use the function `namelengthmax` in the command window to determine this value)
  - Cannot/should not use reserved words (such as names of built-in functions, such as `sin()`, or predefined values, such as `pi`)

Good programming practices

- Expressions & Assignments
  - If an assignment or expression is very long (i.e. longer than the command/editor window), it can be broken between several lines using an ellipsis (three periods in succession).

- Variables
  - Use meaningful/conventional variable names:

    ```
    force = jedi*midichlorian
    ```

    is going to be harder to understand when you look back on your physics homework code than

    ```
    force = mass*accel
    ```

  - Use relatively short names:

    ```
    root_mean_squared_velocity
    ```

    while descriptive, is going to be a pain to type; a simpler choice might be

    ```
    vel_rms
    ```

  - Don't make variable names too similar:

    `rcubed` and `rCubed` is asking for a debugging nightmare

  - Be consistent in case & capitalization style
    - underscore_case
    - camelCase

Basic concepts

- Data types (values that can be assigned to variables)
  - Numbers
    - Integers – entered without a decimal point
    - Real – use `realmin` and `realmax` to return the smallest and largest real numbers represented in Matlab
    - Complex – represented in rectangular form; imaginary unit $\sqrt{-1}$ defined as `i` and `j` in Matlab

```
EDU>> i
ans =
           0 + 1.0000i
```

    - "Non-numbers": `-Inf, Inf, NaN`
  - Vectors/matrices
  - Strings
  - Cell arrays
  - Structures, function handles, plot handles, …

Vectors and Matrices

5 + 6i

scalar
(complex)

4
−2
8

3x1 row vector
(integer)

8.2	NaN	92.9	-3.5

1x4 column vector
(real)

8	7	92	-3
17	5	-1	0
6	-8	7	10

3x4 matrix
(integer)

- Store values of the same type only

Creating Row Vectors

```
EDU>> v = [1, 2, 3, 4]
v =
     1     2     3     4


EDU>> v = 2:5
v =
     2     3     4     5


EDU>> v = 1.1:2.2:5.8
v =
    1.1000    3.3000    5.5000


EDU>> v = linspace(1.1, 5.8, 3)
v =
    1.1000    3.4500    5.8000
```

Use brackets with comma or simply space separated elements

Using the colon operator you can choose a beginning value, end value, and increment size (all of which must be integers or real numbers, not complex)

`linspace(start, end, n)`, where `n` is the number of equal increments in which to divide the range `start` to `end`

Creating Column Vectors

```
EDU>> v1 = [1; 2; 3; 4]

v1 =

     1
     2
     3
     4

EDU>> v2 = (2:5)'

v2 =

     2
     3
     4
     5

EDU>> v3 = [v1; v2]

v3 =

     1
     2
     3
     4
     2
     3
     4
     5
```

Use brackets with semicolon separated elements

Transpose operator acting on row vector (because operations within parentheses evaluated first, could use colon operator or `linspace` to form initial row vector)

Concatenating two column vectors. This can also be done with two row vectors using a comma or space separated elements within the brackets.

Accessing & Modifying Vector Elements

```
                1      2    3     4      5     6    7     8
EDU>> v = [1.2 |-3.4 |2.0 |10.0 |-12.5 |3.1 |72.9 |4.1];

EDU>> v(5)

ans =

  -12.5000


EDU>> v(6:8)

ans =

    3.1000   72.9000    4.1000


EDU>> v([1 3 4])

ans =

    1.2000    2.0000   10.0000


EDU>> v(v<0)

ans =

   -3.4000  -12.5000
```

Access elements using parentheses; here just a single element is selected

The colon operator accesses a range for which the start, end, and increment can all be defined, but must be integers

An index vector may be used to access specific elements of interest

Logical indexing can be used to identify elements that fulfill a specified condition

Creating Matrices

```
EDU>> M = [8 7 92 -3; ...
         NaN 5 -1 0;...
           6 -8 7 10]
M =

     8       7      92      -3
   NaN       5      -1       0
     6      -8       7      10
```

Commas or spaces between elements in rows, semicolons between rows. Note the use of an ellipsis to break up the assignment expression

```
EDU>> M = [1:4; 9:-1:6]
M =

     1       2       3       4
     9       8       7       6
```

Colon operators and linspace can also be used to create the rows or columns. Just make sure to use the proper separator between row vs column definitions!

```
EDU>> M = [(linspace(1,10,3))',(2:4)', (-1:2:4)’]
M =
    1.0000    2.0000   -1.0000
    5.5000    3.0000    1.0000
   10.0000    4.0000    3.0000
```

Creating Matrices

```
EDU>> M = [1 2 3; 4 5];
Error using vertcat
CAT arguments dimensions are not consistent.
```

Must have same number of elements in each row and same number of elements in each column

- Matlab also supplies a number of special matrix definitions, such as `zeros(), ones(), rand(), randi(), diag(), eye(), ...`

Accessing & Modifying Matrix Elements

Access element by
(row number, column number)

```
EDU>> M(2,3)

ans =

     1

EDU>> M(:,2)

ans =

    47
     7
    29

EDU>> M(2,:)

ans =

    27       7       1      40

EDU>> M(1:2, [2,4])

ans =

    47       9
     7      40
```

Access all elements in a row for specified column

Access all elements in a column specified row

For a specified range of rows, access specific columns

$$\begin{array}{c|c|c|c|c|}
 & \text{column 1} & \text{column 2} & \text{column 3} & \text{column 4} \\
\hline
\text{row 1} & 3 & 47 & 24 & 9 \\
\hline
\text{row 2} & 27 & 7 & 1 & 40 \\
\hline
\text{row 3} & 39 & 29 & 17 & 16 \\
\hline
\end{array}$$

Accessing & Modifying Matrix Elements

```
EDU>> M1 = [1 0; 2 3; 7 8];
M1 =

     1     0
     2     3
     7     8

EDU>> M2 = [10 11; 13 16];
M2 =

    10    11
    13    16

EDU>> M = cat(1, M1, M2)
M =

     1     0
     2     3
     7     8
    10    11
    13    16

EDU>> cat(2, M1, M2)
Error using cat
CAT arguments dimensions are not consistent.
```

`cat(dimension, A, B)`, concatenate two matrices, `A` and `B`, along `dimension`

Accessing & Modifying Matrix Elements

```
EDU>> M1

M1 =

     1     0
     2     3
     7     8

EDU>> M = repmat(M,[1,2])

M =

     1     0     1     0
     2     3     2     3
     7     8     7     8
```

`repmat(A, [n, m])` repeat matrix `A` `n` times in first dimension and `m` times in second dimension

Vector and Matrix Operations

- Matrix vs element-wise operations:

  - Matrix multiplication:

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}$$

  - Element-wise multiplication:

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} ae & bf \\ cg & dh \end{bmatrix}}$$

Vector and Matrix Operations

```
EDU>> A
A =

     0     1
     2     3

EDU>> B
B =

    -1     3
     4     8

EDU>> A*B
ans =

     4     8
    10    30

EDU>> A.*B
ans =

     0     3
     8    24
```

Matrix-wise operation

Element-wise operation

Vector and Matrix Operations

Operation	Matrix-wise	Element-wise
Addition	+	+
Subtraction	-	-
Multiplication	*	.*
Left division	\	.\
Right division	/	./
Exponentiation	^	.^

Left division: $a \backslash b \rightarrow \frac{b}{a}$

Right division: $a / b \rightarrow \frac{a}{b}$

Vector and Matrix Operations

- Most functions which act on a scalar can be given a vector/ matrix argument and will act element-wise:

```
EDU>> A = [4 16; 0 2];
EDU>> sqrt(A)

ans =

    2.0000    4.0000
         0    1.4142
```

- Functions of a matrix in the linear algebra sense are signified by names ending in m: `expm, funm, logm, sqrtm`

```
EDU>> B = sqrtm(A)
B =

    2.0000    4.6863
         0    1.4142

EDU>> B*B
ans =

    4.0000   16.0000
         0    2.0000
```

Vector and Matrix Operations

```
EDU>> v1 = [1 0 0];
EDU>> v2 = [0 1 0];
EDU>> dot_prod = dot(v1,v2)

dot_prod =

     0

EDU>> cross_prod = cross(v1, v2)

cross_prod =

     0     0     1
```

- Vector dot product and cross product

- Matrix transpose

```
EDU>> A = [1 2 3; 4 5 6];
EDU>> A

A =

     1     2     3
     4     5     6

EDU>> A'

ans =

     1     4
     2     5
     3     6

EDU>> transpose(A)

ans =

     1     4
     2     5
     3     6
```

Strings

- Strings are vectors of characters.

```
EDU>> fruit = 'apple’

fruit =
apple

EDU>> fruit(1)

ans =
a

EDU>> fruit = ['apple'; 'mango'; 'peach'];
EDU>> fruit

fruit =
apple
mango
peach


EDU>> fruit(2,1)

ans =
m

EDU>> fruit = ['apple'; 'mango'; 'kiwi'];
Error using vertcat
CAT arguments dimensions are not consistent.
```

Single quotation marks

Cell arrays

- Cell arrays generalize matrices to allow for arbitrary entries. Each element is called a "cell" and access of the cells is by numerical indexing

```
EDU>> fruit = {'apple', 'peach', 'mango'}
fruit =

    'apple'     'peach'     'mango'

EDU>> fruit(2)
ans =

    'peach'

EDU>> fruit = cat(1, fruit, {[2 0], 0, 'A'})
fruit =

    'apple'           'peach'     'mango'
    [1x2 double]      [     0]    'A'

EDU>> fruit(2,1)
ans =

    [1x2 double]

EDU>> fruit{2,1}
ans =

     2     0

EDU>> fruit{2,1}(1)
ans =

     2
```

Use curly braces to define a cell array

Parentheses access specific cells in the cell array

Cells don't have to be of the same type. Cell arrays can be concatenated like matrices

Again, parentheses access the cell, while curly braces access the contents of the cell

Access an element of the vector which is contained within a cell of the array

Getting help

- Information about Matlab can be found in Matlab's help facilities. Command line arguments include:
  - `helpbrowser:` opens a new window which contains different ways to obtain the correct information, like lists, a global index and a search function.
  - `help <function name>` gives a short description of the function, the syntax, and other closely related help functions. If more extensive results are needed, try the command `doc <function name>` which also opens the help browser window.
  - `lookfor 'topic'` gives the list of all possible function names which contain the specific search word.

- Online: http://www.mathworks.com/help/matlab/

http://www.mathworks.com/matlabcentral/?s_tid=gn_mlc