Slide Set 2
MATLAB Basic Syntax

Maria Ptashkina Damian Romero

Barcelona Graduate School of Economics

September 2020

Agenda

Outline

Issuing Commands

To execute a command type it at the prompt ($>>$) in the command window and hit `Enter`

```
>> x=1+2
x =
     3
```

- We told MATLAB to add 1 to 2 and then assign the result to a variable called x
- This value will not change unless x is assigned a new value by later statement
- The definition of **variable** (in computer science, as opposed to mathematics) is a named location in *memory*
  - We will learn about different data types for variables later

To repeat the command hit the up-arrow key ($\uparrow$)

Issuing Commands

Unless you tell it otherwise, MATLAB will print the result. Use a semicolon (;) after the command if you don't want the result printed

```
>> x=5;
>> x
x =
     5
```

Continuing command to the next line (...)

```
>> x=2+...
8
x =
    10
```

Putting more that one command on a line

```
>> w = 4, a = 9.8; v = 4.5;
w =
     4
```

Variable Names

MATLAB allows a variable's name ("identifier" in computer science) to be a single letter, such as `x`, or a word, such as `weight`

MATLAB distinguishes between upper and lower case: `X` and `x` are two different variables, as are `hOmEr` and `HoMeR`

Additionally, any of the characters in the name, except the first one, may be a digit or the underscore (_)

Do not give names identical to build-in functions (e.g. `sum`)

Examples of legal identifiers:

- `phi007squared`
- `this_class_is_too_slow`
- `BEar_WiTh_mE`

Examples of illegal identifiers

- `45x`
- `Wat 4`

Comments

It's better you keep track of what you do for yourself and others. In MATLAB this is done using a percentage sign

Example 1:

```
% Author: me
% Session 1: Basic Syntax
% Date
```

Example 2:

```
>> number = 6 % number of eggs in basket
```

Example 3:

```
%{
Author: me
Session 1: Basic Syntax
Date
%}
```

Operators

Scalars (but also matrices and vectors, as we see in a bit) can be operated on by the following familiar operators: $+, -, *, /, \hat{}$

Arbitrarily complicated expressions can be formed by combining more than one operation, e.g.

```
>> x = a*b + c;
>> y = c + a*b;
>> z = a*(b + c);
```

Exercise 1

Consider two variables defined by two functions of `x`

$$y = x^3 - 100$$

$$z = log(x)$$

Find the value of $w = 2y + z^2$ when $x = 5$

Exercise 1: Solution

Consider two variables defined by two functions of `x`

$$y = x^3 - 100$$

$$z = log(x)$$

Find the value of $w = 2y + z^2$ when $x = 5$

Note that you can't assign values to `y` and `z` before assigning a value to `x`

```
>> x=5
x =
     5
>> y=x^3-100
y =
    25
>> z=log(x)
z =
    1.6094
>> w=2*y+z^2
w =
   52.5903
```

Outline

A Matrix

A **matrix** is a two-dimensional, rectangular arrangement of numbers

In MATLAB a matrix is a special case of an **array**, which can have more than two dimensions, and a scalar. In mathematics a scalar is a single number, but in MATLAB it is treated as a 1-by-1 matrix or array!

```
x =
     5
>> size(x)
ans =
     1     1
```

It is possible to make larger matrices by using brackets and semicolons (;)

```
>> X = [1 2 3; 3.4 pi -4]
X =
    1.0000    2.0000    3.0000
    3.4000    3.1416   -4.0000
```

- You can use an optional comma after an element
- You can use `Enter` instead of semicolon

Vectors

A vector in MATLAB is simply a matrix with exactly one column or exactly one row

A row vector

```
>> x = [1 4 7]
x =
     1     4     7
```

You can create the column vector by using semicolon or a transposition operator (')

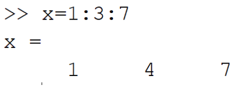
```
>> y = [1; 4; 7]
y =
     1
     4
     7
>> z=x'
z =
     1
     4
     7
```

Supplemental: Column vs. Row Major Arrays

The Colon Operator

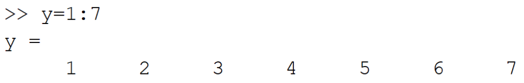
MATLAB provides a convenient way to produce regularly spaced vectors with a colon operator (:)

For example `x = 1:3:7` means "Assign `x` to be the vector whose elements begin with 1, increase by 3, and go no higher than 7"



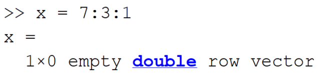
```
>> x=1:3:7
x =
     1     4     7
```

The most common spacing used with the colon operator is 1, which can be abbreviated



```
>> y=1:7
y =
     1     2     3     4     5     6     7
```

With caution!



```
>> x = 7:3:1
x =
  1×0 empty double row vector
```

MATLAB will not throw an error, but will create an empty matrix

`linspace` Function for Declaring Vectors

You can also generate a *row* vector by partitioning an interval in equal-sized sub-intervals using the function `linspace`

This is handy when you know how many intervals you need, but it's not obvious where the points are

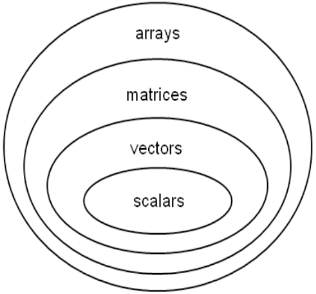
```
>> a = linspace(1, 25, 8)
a =
    1.0000    4.4286    7.8571   11.2857   14.7143   18.1429   21.5714   25.0000
```

We told MATLAB to create a vector with 8 numbers equally far from each other withing the interval from 1 to 25

Arrays, Matrices, Vectors, Scalars

It is possible to have three-(and more)-dimensional arrays in MATLAB

Example: the element `A(2,3,4)` is on the second row of the third column of the fourth page of the array `A`



Nerd side note: Although both “matrix” and “array” are often used interchangeably, strictly speaking in MATLAB the term “matrix” is appropriate only when the numbers represent the coefficients in a set of linear equations

Accessing Parts of a Matrix

Accessing a single element

```
>> X = [1 2 3; 3.4 pi -4];
>> X(2,3)
ans =
    -4
```

Accessing multiple elements

- A whole row `i:`
  `M(i,:)` or
  `M(i,1:end)`
- A whole column `j:`
  `M(:,j)` or
  `M(1:end,j)`
- Use `M(m:n, p:q)` to define sub-matrices

```
>> X(1,:)
ans =
     1     2     3
>> X(1:end,1)
ans =
    1.0000
    3.4000
>> X(1:2,1:2)
ans =
    1.0000    2.0000
    3.4000    3.1416
```

Assigning Matrix Elements

We can use this access method to assign a value to one element of a matrix (`X(2,3) = 7`), or the whole row (`X(1,:)   = 38`), or column (`X(:,2)= 54`)

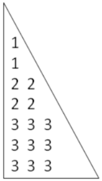
You can use an empty matrix `[]` to delete some elements

```
>> X=[1 2 3; 3.4 pi, -4]
X =
    1.0000    2.0000    3.0000
    3.4000    3.1416   -4.0000
>> X(:,1)=[]
X =
    2.0000    3.0000
    3.1416   -4.0000
```

Combining Matrices

You can combine (stack) matrices, but be careful with the dimensions!

```
>> B1 = [1;1];
>> B2 = [2 2; 2 2];
>> B3 = [3 3 3; 3 3 3];
>> [B1;B2;B3]
Error using vertcat
Dimensions of arrays being concatenated are not consistent.
```



Exercise 2

1. Generate a column vector A1 with 2 elements
2. Generate a 2×2 matrix B2
3. Generate a 2×3 matrix C3
4. Generate a 3×2 matrix D4
5. Generate a 3×4 matrix E5
6. Stack these matrices together into one matrix without transposing, call it whole
7. Access the bottom right element of whole
8. Access the top 3×3 matrix
9. Change all elements of the last row to 25

Exercise 2: Solution

1. `>> A1 = [1; 1];`
2. `>> B2 = [2 2; 2 2];`
3. `>> C3 = [3 3 3; 3 3 3];`
4. `>> D4 = [4 4; 4 4; 4 4];`
5. `>> E5 = [5 5 5 5; 5 5 5 5; 5 5 5 5];`
6. `>> whole = [A1 B2 C3; D4 E5]`
7. `>> whole(5,6)`
8. `>> whole(1:3,1:3)`
9. `>> whole(5,:)=25`

Outline

Matrix Arithmetic: Addition and subtraction

$Z = X + Y$ means that for each m and n, $Z(m,n) = X(m,n) + Y(m,n)$

```
>> X = [1 5 -2; 3 0 7]
X =
     1     5    -2
     3     0     7
>> Y = [6 0 6; 2 2 1]
Y =
     6     0     6
     2     2     1
>> Z = X + Y
Z =
     7     5     4
     5     2     8
```

Matrix Arithmetic: Multiplication

MATLAB has two types of multiplication

- Array multiplication: multiplies corresponding elements of the matrices (they must be same size).

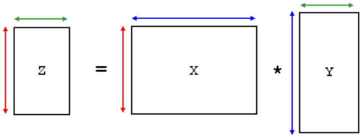
  `Z = X.*Y` (note a dot after X!) means that for each m and n, $Z(m,n) = X(m,n) * Y(m,n)$

- Matrix multiplication: multiplication operation used in linear algebra (number of columns in X be equal to the number of rows in Y).

  `Z = X*Y` means that for each row m and column n, $Z(m,n) = \sum_k X(m,k)Y(k,n)$

Matrix Arithmetic: Multiplication Example

```
>> X =[1 2 3; 4 5 6; 6 1 1; 0 1 -3]
X =
     1     2     3
     4     5     6
     6     1     1
     0     1    -3
>> Y = [2 -2; 3 8; 7 4]
Y =
     2    -2
     3     8
     7     4
>> Z = X*Y
Z =
    29    26
    65    56
    22     0
   -18    -4
```



1*2 + 2*3 + 3*7 → 29	1*(-2) + 2*8 + 3*4 → 26
4*2 + 5*3 + 6*7 → 65	4*(-2) + 5*8 + 6*4 → 56
6*2 + 1*3 + 1*7 → 22	6*(-2) + 1*8 + 1*4 → 0
0*2 + 1*3 + (-3)*7 → -18	0*(-2) + 1*8 + (-3)*4 → -4

Matrix Arithmetic: Division and Exponentiation

The same applies to division

- Array division: `Z = X./Y` means that for each m and n, $Z(m,n) = X(m,n)/Y(m,n)$
- Matrix "division": multiplying by the inverse in linear algebra

And similarly for exponentiation

- Array exponentiation: `X.^Y` means that for each m and n, $Z(m,n) = X(m,n)\hat{}\,Y(m,n)$ (must have same shape)
- Matrix exponentiation: If X is a square matrix and p is a scalar, `Z = X^p` means that $Z = X * X * X ... * X$ (p times)

Operations with Scalars

If one of the operands is a scalar, array and matrix operations are the same

```
>> Y = [1 2 3; -4 5 0]
Y =
     1     2     3
    -4     5     0
>> c = 6;
>> Z = c + Y
Z =
     7     8     9
     2    11     6
```

Subtraction and multiplication work similarly: `c.*Y`, `c*Y`, `Y.*c`, and `Y*c` each mean that every element of Y is multiplied by c

For division involving an array and a constant: `Y./c` and `Y/c` mean that every element of Y is divided by c, while `c./Y` means that c is divided by every element of Y (`c/Y` is not legal)

Operator Precedence

Programmer's Toolbox to Generate Matrices

Every programming language comes with a set of ready-to-use functions for frequently used operations (often referred to as 'libraries')

We've already seen some functions from the Toolbox (like `size` and `linspace`)

There are useful built-in functions to generate matrices

Function	Returns an n-by-m matrix...
zeros(n,m)	of zeros
ones(n,m)	of ones
eye(n,m)	identity matrix
rand(n,m)	of random numbers uniformly distributed in the range from 0 to 1
randn(n,m)	of random numbers from a standard normal distribution
NaN(n,m)	of Not a Number.

Note! If you type one argument instead of two (e.g. `rand(n)`) you will get a square matrix of dimension *n*

Programmer's Toolbox to Transform Matrices

A useful function is `repmat`, whose name means "replicate matrix"

```
>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
>> repmat(A,1,2)
ans =
     1     2     3     1     2     3
     4     5     6     4     5     6
```

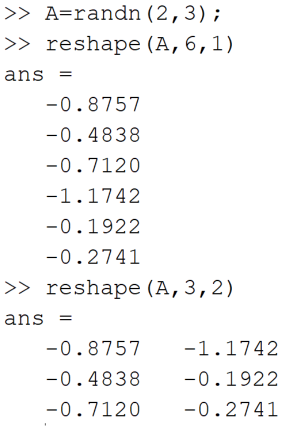
`repmat(A,m,n)` creates a matrix in which the elements in `A` are repeated on *m* rows and *n* columns

We can combine it with other transformation methods

```
>> B=ones(3,1);
>> C=[2;2;2];
>> D=zeros(3,1);
>> E = [C' D'; repmat(B,1,6)]
E =
     2     2     2     0     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
```

Programmer's Toolbox to Transform Matrices

Another useful function is `reshape`



```
>> A=randn(2,3);
>> reshape(A,6,1)
ans =
   -0.8757
   -0.4838
   -0.7120
   -1.1742
   -0.1922
   -0.2741
>> reshape(A,3,2)
ans =
   -0.8757   -1.1742
   -0.4838   -0.1922
   -0.7120   -0.2741
```

`reshape(A,m,n)` takes matrix A and puts into the matrix with m rows and n columns, using a column-major-order logic

Programmer's Toolbox to Work with Matrices

MATLAB Toolbox has plenty of different functions. No need to learn them by heart! Use help of search online if you need to perform an action. The tables that follow are just for illustration!

These are some important linear algebra instruments

Function	Returns
diag(A)	1) if A is a matrix - extracts the elements in the diagonal 2) if A is a vector - creates a diagonal matrix with elements of A in the diagonal
det(A)	Determinant of A
tril(A)	Compute the lower triangular matrix of A
triu(A)	Compute the upper triangular matrix of A
rank(A)	Rank of A
trace(A)	Trace of A
eig(A)	Eigenvalues of A

Programmer's Toolbox for Matrix Descriptions

FUNCTION	RETURNS A ROW VECTOR CONSISTING OF
max(M)	Largest element of each column
min(M)	Smallest element of each column
mean(M)	Mean of each column
median(M)	Median of each column
size(M)	Number of rows, number of columns
sort(M)	Sorted version, in ascending order, of each column
std(M)	Standard deviation of each column
sum(M)	Sum of the elements of each column

If you are working with vectors, the same functions will return scalars! (well, except `size`, as we've seen before, and `sort`, obviously)

Note! `sum(M,dim)` actually takes 2 arguments. But remember than MATLAB is column-major language? The default `dim=1` summing along columns. If you want to sum along rows, you tell MATLAB `sum(M,2)`

More on polymorphism

Exercise 3

1. Generate a 3-by-3 matrix M of random numbers from a uniform distribution
2. Generate a 2-by-2 identity matrix A
3. Calculate determinant, rank and eigenvalues of matrix M
4. Define a vector D containing the elements in the diagonal of M
5. Generate B, a 4-by-4 matrix generated by repeating the upper-leftmost 2-by-2 sub-matrix of M
6. Define a vector C containing the elements in the first column of M
7. Generate F, a 3-by-4 matrix which columns are all identical to vector C

Exercise 3: Solution

1. `M = rand (3, 3);`
2. `A = eye (2);`
3. `d = det (M), r = rank (M), eg = eig (M)`
4. `D = diag (M)`
5. `M1= M (1:2, 1:2); B = repmat (M1, 2, 2)`
6. `C = M (:, 1);`
7. `F = repmat (C, 1, 4);`

Outline

Column vs. Row Major Arrays

If you type a larger matrix in MATLAB

```
>> Y = [1 2 3 6 4 1 12; 3.4 -8 3 3 0 pi .2]
Y =
  Columns 1 through 4
    1.0000    2.0000    3.0000    6.0000
    3.4000   -8.0000    3.0000    3.0000
  Columns 5 through 7
    4.0000    1.0000   12.0000
         0    3.1416    0.2000
```

The default in MATLAB (like Julia and R, but unlike Python!) is that a matrix is printed in *column-major order*, meaning that all the elements of one column are processed before the elements of the next column

This is important for processing efficiency: in a 2D column-major array if you are using a nested loop, it is more efficient to process the columns in the outer loop and the rows in the inner loop

Back.

Operator Precedence

PRECEDENCE	OPERATOR
0	Parentheses: (...)
1	Exponentiation ^ and Transpose '
2	Unary +, Unary –, and logical negation: ~
3	Multiplication and Division (array and matrix)
4	Addition and Subtraction
5	Colon operator :

Polymorphism

In the general study of programming languages, when the *type of an argument* used in a function can vary (as for example, from a scalar to a vector to a matrix) from one call of the function to the next, the function is said to be **polymorphic**

`sum` is certainly polymorphic, since it accepts either a vector or a two=dimensional matrix as its argument

A function is also polymorphic if it accepts *varying numbers of arguments*. If a function can be called with one argument or with two arguments, it is polymorphic

Again, `sum` exhibits this second aspect of polymorphism as well (remember it can take also the dimension argument)! Back