Introduction to MATLAB: Basic Syntax

Eduardo Rossi
University of Pavia
erossi@eco.unipv.it

September 2014
When you start MATLAB, you get a multipaneled desktop:

- **Command Window**: Here you can give MATLAB commands typed at the prompt, `>>`.
- **Current Directory**: Directory where MATLAB looks for files
- **Workspace**: Shows what variable names are currently defined and some info about their contents
- **Command History**: History of your commands
MATLAB is huge! - there is no way to remember everything you will need to know.

- `help command` - shows in the Command Window all the ways in which you can use the command
- `doc command` - brings up more extensive help in a separate window
- Highlight the `command` then press F1
Objects in MATLAB

- MATLAB is a matrix-oriented environment and programming language.
- Vectors and matrices are the basic data structures.
- Functions and operators are available to deal with vectors and matrices directly.
- This means that MATLAB objects are intended as matrices or vectors.
- Note that MATLAB is case sensitive. This means that \( \Theta \) is different from \( \theta \).
Objects Names

- MATLAB does not require any memory management and variables can be input with no setup.
- This feature simplifies the definition of new objects.
- The memory is allocated when the objects are created.
- These can be used in subsequent calculations.

The generic form of an expression in MATLAB is (assignment)

\[ \text{VariableName} = \text{Expression}; \]

For instance,

\begin{verbatim}
x=1;
y=x;
z=y^2;
\end{verbatim}
Objects Names

- Values are assigned to variables.
- Each variable must be assigned a value before it is called.
- Note that assigning one variable to another assigns the value of that variable, not the variable itself.
- Legal names consist of any combination of letters and digits, starting with a letter.
- One often does not want to see the result of intermediate calculations - terminate the assignment statement or expression with semi-colon;
These are not allowable: Net-Cost, 2pay, %x, @sign

Special names: you should avoid using

\[
\begin{align*}
\text{eps} &= 2.2204e-16 \\
\text{pi} &= 3.14159...
\end{align*}
\]

The largest number such that \(1 + \text{eps}\) is indistinguishable from 1

If you wish to do arithmetic with complex numbers, both \(i\) and \(j\) have the value \(-1\) unless you change them.
Vectors and matrices are the basic data structures

All data in MATLAB are matrices by construction, even if they are $1 \times 1$ (scalar), $K \times 1$ or $1 \times K$ (vectors).

Vectors, both row ($1 \times K$) and column ($K \times 1$) can be entered directly into the command window.

Functions and operators are available to deal with vectors and matrices directly.

Vectors and matrices are lists of numbers separated by either commas or spaces.
You may enter row vectors as follows:

\[ \mathbf{vX} = [22, 5, 31]; \]

and column vectors as:

\[ \mathbf{vX} = [22; 5; 31]; \]

where the entries must be enclosed in square brackets.
You may enter also a function in a vector. For example

\[ vX = \begin{bmatrix} 22; 5; \sqrt{5} \end{bmatrix}; \]

inserts as last element of the vector the result of the square root of 5. Matrices are simply column vectors of row vectors. For instance, to input the $3 \times 3$ matrix

\[ X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \]

enter the matrix one row at a time, separating the rows with a semicolon (;):

\[ mX = \begin{bmatrix} 1 & 2 & 3 ; 4 & 5 & 6; 7 & 8 & 9 \end{bmatrix}; \]
Empty-matrix

There is one unusual matrix worth mentioning. The empty matrix is simply a matrix (or vector) with no elements, \( \mathbf{mX} = [ ] ; \). It serves to useful purposes.

- It can be used for lazy vector construction using repeated concatenation.
- When calling functions with multiple inputs, but only fewer are used.
Creating matrices

A = ones(m;n);
A = zeros(m;n);
A = eye(m);
A = a : b : c;

Note we can combine matrices too by horizontal concatenation:

C = [A B].
One Precaution: Spacing

Try typing

>> a2 = [1 + 2 3]

versus

>> a2 = [1 +2 3]
Accessing matrix elements

You may extract bits of a vector vectorising the parenthesis and the colon operator ::. For example, given the row vector

\[
\text{vY} = [1 \ 3 \ 5 \ -1 \ -3 \ -5 \ -7];
\]

To get the 3rd to 6th entries, we input

\[
\text{vY}(3:6)
\]

\[
\text{ans}=\begin{bmatrix}
5 \\
-1 \\
-3 \\
-5
\end{bmatrix}
\]

The colon operator can be interpreted as \textit{all elements in that dimension}, when we wish to select all the elements of a row or of a column.

For instance, \texttt{mX(:,1)} is interpreted as all elements from matrix \texttt{mX} in column 1. We can convert a row vector into a column vector (and vice versa) by transposing - denoted by ‘\’ .
Accessing matrix elements

Column 1 of matrix B

\[ B(:,1) \]

Third row of matrix B

\[ B(3,1:end) \]

\[ C = \begin{bmatrix} B_{12} \\ B_{32} \end{bmatrix} \]

\[ C = B([1 3],2) \]
Accessing matrix elements

Given $B(m \times n)$ to create a vectorized matrix:

\[ C = \text{vec}(B) = \begin{bmatrix} B.1 \\ B.2 \\ \vdots \\ B.n \end{bmatrix} \]

For instance,

\[ B = \begin{bmatrix} 0.8147 & 0.1270 \\ 0.9058 & 0.9134 \end{bmatrix} \]

\[ >> B(:) \]

\[ \text{ans} = \]
\[ \begin{array}{c} 0.8147 \\ 0.9058 \\ 0.1270 \\ 0.9134 \end{array} \]
The basic arithmetic operators are

\[ + \quad - \quad * \quad / \]

The symbol \(^\) is used to get exponents (powers)

\[ 2^4 = 16 \]

You should type in commands shown following the prompt \(>>\):

\[ >> \quad 2 + 3/4*5 \]

\[ \text{ans} = \quad 5.7500 \]

\[ >> \]

Is this calculation \(2 + 3/(4 \times 5)\) or \(2 + (3/4) \times 5\)?
Matlab works according to the priorities:

1. quantities in brackets,
2. powers $2 + 3^2 \rightarrow 2 + 9 = 11$,
3. $\times /$, working left to right $(3 \times 4/5 = 12/5)$
4. $+ -$, working left to right $(3 + 4 - 5 = 7 - 5)$

Thus, the earlier calculation was for $2 + (3/4) \times 5$ by priority 3.
Numbers and Formats

Matlab recognizes several different kinds of numbers

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1362; −217897</td>
</tr>
<tr>
<td>Real</td>
<td>1.234; −10.76</td>
</tr>
<tr>
<td>Complex</td>
<td>3.21 − 4.3i</td>
</tr>
<tr>
<td>Inf</td>
<td>Infinity (result of dividing by 0)</td>
</tr>
<tr>
<td>NaN</td>
<td>Not a Number, 0=0</td>
</tr>
</tbody>
</table>

The e notation is used for very large or very small numbers:

-1.3412e+03 = -1.3412 * 10^3 = -1341.2
-1.3412e-01 = -1.3412 * 10^{-1} = -0.13412

All computations in MATLAB are done in double precision, which means about 15 significant figures.
The format - how Matlab prints numbers - is controlled by the \texttt{format} command. Type \texttt{help format} for full list. Should you wish to switch back to the default format then \texttt{format} will suffice. The command \texttt{format compact} is also useful in that it suppresses blank lines in the output thus allowing more information to be displayed.

\begin{center}
\begin{tabular}{|l|l|}
\hline
>> \texttt{format short} & 31.4162 (4 decimal places) \\
>> \texttt{format short e} & 3.1416e+01 \\
>> \texttt{format long e} & 3.141592653589793e+01 \\
>> \texttt{format short} & 31.4162 (4 decimal places) \\
>> \texttt{format bank} & 31.42 (2 decimal places) \\
\hline
\end{tabular}
\end{center}
Multi-Dimensional Arrays

MATLAB is capable of working with $N$-dimensional arrays where $N$ can be a very large number (up to about 30, depending on the size of each matrix dimension). Unlike scalars, vectors and matrices, higher dimension arrays can only be constructed by calling functions and cannot be directly allocated. For example,

```matlab
mdX=zeros(3,3,3);
```

generates a multidimensional array, i.e. a $3 \times 3$ matrix of zeros, repeated over three planes.
Operations between multidimensional array require special commands.
Calling Functions

- Function calls have specific syntax;
- they require input and output;
- use square brackets for multiple outputs

Both built-in and user-created function are called as

\[
[\text{output1, output2, ..., outputN}] = \text{functionname(input1, input2, ..., inputN)};
\]

Examples:

\[
dM = \text{mean(vX)};
\]

or

\[
dM = \text{mean(vX)};
\]

\[
[\text{dy, index}] = \text{min(vX)};
\]
Small Exercise

- Find the function `magic` in the help;
- Call the function `magic` with different inputs;
- Store the output in the matrix $mX$;
- Try using a non-integer value;
- Try using a vector as input;
- Compute the sample average of the rows of $mX$;
- Compute the sample average of the columns of $mX$;
- Select the first 3 columns of $mX$;
- Select the rows 1,4,8 of $mX$;
A note on variable names

- Mathematicians have some conventional tricks to tell the “types” of their variables
  - You’d be surprised if something called $n$ was not an integer
  - And usually, $x$ is a real number, $\theta \in [0, 2\pi)$, $z$ a complex number, $\mathbf{v}$ or $\vec{v}$ a vector, $A$ or $\mathbf{A}$ a matrix, $f$ a function, etc

- Programming languages do not give us so much freedom in notation. MATLAB limits us to A-Z, a-z, 0-9, and _

- This is why Hungarian notation was developed
  - Use variable names consisting of multiple characters
  - The first one or two indicate the type of the variable
  - The remainder form a descriptive name
Example: suppose that you are estimating a linear model \( y = X\beta + \varepsilon \)

So, the input consists of a vector \( y \) and a matrix \( X \). Variable names could be \( \text{vY} \) and \( \text{mX} \)

The output is probably a vector of estimated regression coefficients, which you could call \( \text{vBeta} \) or \( \text{vBetaHat} \) or \( \text{vBetaOLS} \) or something. And perhaps there will also be a \( \text{vEpsilon} \) or \( \text{vEps} \) or . . .

MATLAB doesn’t care what you call your variables. As soon as a variable is given a value, MATLAB will figure out what type it is automatically

So, if you calculate \( (X'X)^{-1} X'y \) and want to call the result \( \text{apple} \), this is not technically wrong

. . . but it’s not particularly nice to people reading your code either (including yourself)
Hungarian notation for MATLAB

Differently from other programming software, MATLAB objects do not need to be declared. So, in principle we do not need to be strict on Hungarian notation. A suggestion could be

- Integers: \( i \), e.g. \( iN = 5 \);
- Double: \( d \), e.g. \( dX = 2.1 \);
- Vectors: \( v \), e.g. \( vX = [2.1, 2.7, 2] \);
- Matrices: \( m \), e.g. \( mX = [2.1, 2; 1, 2.3] \);
- Strings: \( str \), e.g. \( strX = 'HelloWorld' \);
- Function: \( f \), e.g. \( fOLS \)
Mathematical Operators

The mathematical operators allow to add, subtract, multiply and divide matrices or elements of matrices.

- The basic math commands follow the rules of linear algebra.
- For instance, to multiply two matrices together, they must be conformable along their inside dimensions;
- Attempting to multiply nonconforming matrices produces an error.
- To transpose a matrix the symbol is ‘’. 
Addition and subtraction in MATLAB require $x$ and $y$ to have the same dimensions or to be scalar.

- If $mX$ and $mY$ are matrices, they need to be conformable.
- Addition (or subtraction) are interpreted element-wise, so that

\[
mZ = mX + mY;
\]

will generate a new matrix $mZ$, with the same dimension of $mX$ and $mY$, such that $z(i,j) = x(i,j) + y(i,j)$.

- Analogously for the subtraction operator.
Matrix Multiplication

Multiplication between matrices requires the inner dimension of the matrices to be conformable (inner product).

- Suppose that the matrix $mX$ has dimension $L \times M$ and $mY$ to be a $N \times K$,
- The product $mX*mY$ will be allowed if and only if $M$ and $N$ are equal.
- Analogously, the product $mY*mX$ will be allowed if and only if $K$ and $L$ are equal.
- Note that, as for addition and subtraction, the product between a matrix and a scalar is intended element-wise, that is if $dX$ is scalar, then the operation

$$mZ=dX*mY$$

will produce a new matrix $mZ$, with the same dimension of $mY$, such that $z(i,j) = x * y(i,j)$. 
Matrix division, is intended in MATLAB as the solution of the linear system

\[ YZ = X \]

where \( X \) is an \((M \times N)\) matrix, and \( Y \) is an \((N \times L)\) matrix. The result of the division is the solution of the linear system, that is obtained by OLS, so that

\[ Z = (X'X)^{-1}X'y. \]

\[ \texttt{mZ} = \texttt{mX} \ \backslash \ \texttt{mY}; \]
The dot (.) operator transforms the usual operations between matrices in element-by-element operations. For example, suppose that \( mX \) and \( mY \) are \((M \times N)\) matrices, then

\[
mZ = mX .* mY
\]

will produce a new matrix \( mZ \), with the same dimension of \( mX \) and \( mY \), such that \( mZ(i,j) = mX(i,j) \times mY(i,j) \).

Analogous results are obtained with division (./) and exponentiation.

Note that the "dot" operator can not be used when matrices differ by size. This means that it can be useful to use the command \texttt{repmat},

\[
mX = \texttt{repmat}(mY,m,n)
\]

that creates a large matrix \( mX \) consisting of \((M \times N)\) tiling of copies of \( mY \).

Functions that operate on each element include \texttt{exp}, \texttt{ln}, \texttt{sqrt}, \texttt{cos}, \texttt{sin}, \texttt{tan}, \texttt{acos}, \texttt{asin}, \texttt{atan}, and \texttt{abs}.
Parenthesis and precedence

Parentheses can be used to control the order of mathematical expressions.
- Parentheses can be nested to create complex expressions.
- You can only use round brackets to control the order of the mathematical expressions.
- **Round** brackets are used to select
  1. select elements of matrices;
  2. calling functions;
  3. precedence
- **Square** brackets are used to form vectors and matrices;
- **Curly** brackets are used in cell array assignment statements.
Exercises

1. Define the vectors \( x = (1 \ 2 \ 3 \ 4) \); and \( x = (1 \ 7 \ 0 \ 4) \).

2. Compute the products \( z = x' y \), \( u = xy' \) and \( w = x \odot y \).

3. Compute the values \( (x + y)^2 \).

4. Which are the values of \( -x^2 \odot y \) and \( -x \odot y \)?

5. Suppose a command \( res = c + (b \ast a)' / c \). What restrictions on the dimensions of \( a, b \) and \( c \) must be true for this to be a valid statement, so that the result is a row vector? (Note that \( a, b \) and \( c \) cannot be scalar nor square matrices.)
Basic Functions

MATLAB contains a wide set of built-in functions. We will briefly describe the syntax of some of the most used functions.

1. **exp**: calculates the exponent function:
   
   ```matlab
dX = exp(3);
   ```

2. **sqrt**: calculates the square root function:
   
   ```matlab
dX = sqrt(3);
   ```

3. **log**: calculates the natural logarithm:
   
   ```matlab
dX = log(2);
   ```

4. **log10**: calculates the logarithm to the base 10:
   
   ```matlab
dX = log10(7);
   ```

5. **log2**: calculates the logarithm to the base 2:
   
   ```matlab
dX = log2(4);
   ```
Basic Functions

Suppose $A$ is a square matrix.
For the inverse of $A$:

\[ \text{inv}(A) \]

for determinant of $A$

\[ \text{det}(A) \]

for a vector equal to the diagonal elements of $A$

\[ \text{diag}(A) \]

for trace of $A$

\[ \text{trace}(A) \]
Basic Functions

- For rank of A

  \[ \text{rank}(A) \]

  With the exception of transposition all of these must be used with appropriate sized matrices.

- Kronecker product of matrices A and B: \[ C = A \otimes B \]

  \[ \text{kron}(A,B) \]

- \text{find} - finds indices of nonzero elements of matrix
Basic Functions

- **inv**: Computes the inverse of a square matrix. Example

  ```
  A = reshape([1 4 7 5], 2, 2);
  B = inv(A);
  ```

- **chol** - Cholesky factorization. Given $A$ a real-valued symmetric, positive-definite matrix there exists a lower triangular matrix $L$, with real and positive diagonal entries, such that

  $$A = LL'$$

  In MATLAB, the output is a upper triangular matrix,

  ```
  A = [0.6 0.4 ; 0.4 0.5];
  B = chol(A);
  B' * B
  ```

  The Cholesky decomposition is unique when $A$ is positive definite; there is only one lower triangular matrix $L$ with strictly positive diagonal entries such that $A = LL'$. However, the decomposition need not be unique when $A$ is positive semidefinite.
Basic Functions

- **qr** - QR factorization: QR factorization (also called a QR decomposition) of a matrix is a decomposition of a matrix $A$ into a product $A = QR$ of an orthogonal matrix $Q$ and an upper triangular matrix $R$.

- **lu** - LU factorization: factors a matrix as the product of a lower triangular matrix and an upper triangular matrix.
Basic functions

Eigenvalues and Roots

- **eig**: Computes the eigenvalues and eigenvectors of a square matrix. Example

  ```matlab
  A = reshape([1 4 7 5], 2, 2);
  [vv, va] = eig(A);
  ```

- **roots, poly**: `roots` computes the roots of a polynomial. The argument is a $N + 1 \times 1$ vector, $V$, that contains the coefficient of a polynomial, ordered in descending powers, such that

  $$v_1 s^N + v_2 s^{N-1} + \ldots + v_{N+1}$$

  Example

  ```matlab
  p = [0.5 1 -2];
  root(p);
  ```

- The inverse operator of `root` is `poly`. MATLAB displays polynomials as row vectors containing the coefficients ordered by descending powers. The characteristic equation of the matrix $A$, is returned in a row vector by `poly`.
Trigonometric Functions

The trigonometric functions operate element-wise on arrays and matrices. The most used trigonometric functions are $\sin$, $\cos$, $\tan$. For example, the command

$$dY = \tan(dX)$$

calculates the tangent to an angle, $dX$. For example, if $dX = \pi/6$, then $dY = 0.5774$. Note that the inverse of the trigonometric functions are called $\text{asin}$, $\text{acos}$, $\text{atan}$, and the results are in radians. For example,

```matlab
x = \sin(\pi/6)
\text{asin}(x)
ans = 0.5236
```

where $\pi/6 = 0.5236$. 
**Utility Functions**

1. **length**: calculates the size the largest dimension of a matrix.

   ```matlab
   A=[2 5 7; 3 2 1]
   length(A)
   ans= 3
   ```

2. **size**: calculates the size of a dimension (or more dimensions) of a matrix.

   ```matlab
   A=[2 5 7; 3 2 1]
   size(A,1)
   ans= 2
   size(A,2)
   ans= 3
   size(A)
   ans= 2 3
   ```

3. You can also assign the resulting number of rows and columns to new variables

   ```matlab
   [n,k]=size(A)
   ```

   The size operator works also with multi-dimensional arrays.
**Utility Functions**

1. **sum**: calculates the sum of the elements of a matrix (columnwise). Example

   ```
   vX=[1; 2; 3];
   sum(vX)= 6
   ```

   If applied to a \((N \times K)\) matrix, it calculates the sum of the \(N\) elements of each column, and reports the results in a row vector with \(K\) columns

   ```
   vX=[1 4 7; 2 5 8 ; 3 6 9];
   sum(vX)= 6 15 24
   ```

2. **mean**: returns the mean values of the elements along different dimensions of an array.

   ```
   vX=[1; 2; 3];
   mean(vX)= 2
   ```

   If applied to a \((N \times K)\) matrix, it computes the mean along the columns. It returns a row vector:

   ```
   vX=[1 4 7; 2 5 8 ; 3 6 9];
   mean(vX)
   ans= 2 5 8
   ```
Utility Functions

1. **min, max**: calculate respectively the minimum and the maximum of the elements of a column vector. If applied to a \((N \times K)\) matrix, they calculate the minimum and the maximum of the \(N\) elements of each column, and report the results in a row vector with \(K\) columns.

\[
A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix};
\]

min(A)
ans = 1 4 7
max(A)
ans = 3 6 9

2. **var, std**: calculate the sample variance (and standard deviation) of the elements of a column vector. Example

\[
vX = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix};
\]

var(vX)
ans = 2.333
std(vX)
ans = 1.5272
Basic functions

Sample Statistics

1. **cov**: calculate the covariance matrix. When applied to a \((N \times K)\) matrix, it calculates the covariance matrix, that is a \((K \times K)\) matrix with the variances on the main diagonal.

\[
A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 4 & 8 & 12 \end{bmatrix}
\]

\[
cov(A)
\]
\[
\text{ans} = \begin{bmatrix} 2.3333 & 3.1667 & 4.0000 \\ 3.1667 & 4.3333 & 5.5000 \\ 4.0000 & 5.5000 & 7.0000 \end{bmatrix}
\]

2. **skewness**: calculate the skewness of the elements of a column vector:

\[
vX = [1; 2; 4]
\]

\[
skewness(vX)
\]
\[
\text{ans} = 0.3818
\]

3. **kurtosis**: calculate the kurtosis of the elements of a column vector:

\[
vX = [2; 5; 4; 9]
\]

\[
kurtosis(vX)
\]
\[
\text{ans} = 2
\]


1. **sort**: sorts in ascending order the elements of a vector. Example

```matlab
vX=[5 3 9 1]
sort(vX)
ans= 1 3 5 9
```

2. To sort in descending order, just type the option ‘descend’. For example

```matlab
A=[2 5 7; 4 6 9; 17 21 13; 7 16 18];
sort(A,'descend')
ans= 17 21 18
    7 16 13
    4  6  9
    2  5  7
```
Exercises

Load the MATLAB file `returns.mat`.

1. Compute the mean for each column of the data.
2. Compute the covariance matrix of the data.
3. Find the maximum and the minimum value for each column of the data.
4. Sort the second column of the data and verify that the minimum corresponds to the first value of this sorted series and the maximum corresponds to the last.
5. Compute the correlation matrix of the data.
6. Compute variance, standard deviations, skewness and kurtosis of each column.
7. Standardize each column
8. Sort in ascending order the columns of the matrix and plot them.