

# MATLAB Overview

## MATH1070: Numerical Mathematical Analysis

Location: [http://people.sc.fsu.edu/~jburkardt/classes/math1070\\_2019/matlab\\_overview/matlab\\_overview.pdf](http://people.sc.fsu.edu/~jburkardt/classes/math1070_2019/matlab_overview/matlab_overview.pdf)

---



*The interactive MATLAB language lets us do computational experiments.*

If you are completely new to MATLAB, then you should review the information at places like the MATHWORKS tutorial website: <https://www.mathworks.com/support/learn-with-matlab-tutorials.html>

## 1 The Very Basics

When you start MATLAB, you see a command window with a prompt sign

```
1 >>
```

This is where you type your commands.

You can give simple arithmetic commands like `98-53`, but most MATLAB commands will involve computing something and storing the result in a named variable. Once a variable has a value, it can be used in later computations:

```
1 >> x = 0.99
2 >> y = x * x % This could also be written as x^2
```

Arithmetic operators include `+`, `-`, `*`, `/`, `^`. Mathematical functions include `abs()`, `max()`, `min()`, `mod()`, `sqrt()`, and trig functions like `sin()`, `cos()`.

When a statement computes a value, that value will be printed to the command window. If this is not desired, simply terminate the statement with a semicolon.

## 2 M Files

While interactive access to MATLAB is useful, it is often necessary to enter long complicated instructions. In such a case, it is better to store these instructions as an M file. An M file can be created by any text editor, or you can use MATLAB's editor by choosing the **NEW** menu item and selecting **Script** or **Function**. You can type in your commands, and then save the file, giving it a name, such as *pde.m*. Once the file is saved, you can run the commands in the file by typing its name, as in

```
1 >> pde
```

or, if this is a function file, you might supply input and collect the resulting output:

```
1 result = lambert ( x )
```

### 3 User Functions

A user function can be created as an M file. It usually has inputs, outputs, and a function name. As a simple example, we might want a function that evaluates

$$y = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04} - 6.0$$

We can create a MATLAB function *hump.m*, so that we only have to type this formula correctly once. Note that division is done with a period-slash, squaring is done with a “period-caret”, and we have to be careful about the parentheses:

```
1 function y = hump ( x )
2   y = 1.0 ./ ( ( x - 0.3 ).^2 + 0.01 ) ...
3     + 1.0 ./ ( ( x - 0.9 ).^2 + 0.04 ) ...
4     - 6.0;
5   return
6 end
```

Once we have created this function, we can evaluate it at 2.0 by writing `y=hump(2.0)`, or get the value at a variable `x` by writing `y=hump(x)` and so on.

We can also pass in a whole vector of `x` values, and get back the corresponding list of `y` results, which is perfect for plotting, and other applications. The list of `x` values can be created using `linspace(first, last, how-many)`:

```
1 x = linspace ( 0.0, 2.0, 101 );
2 y = hump ( x );
3 plot ( x, y )
```

### 4 Control

If some statements are to be executed several times, use the `for` loop:

```
1 fact = 1;
2 for i = 1 : 10
3   fact = fact * i;
4 end
```

If some statements are to be executed only if a condition is true, use an `if()` statement:

```
1 if ( mod ( n, p ) == 0 )
2   n = n / p;
3 end
```

More elaborate cases can add the `elseif()` and `else` statements:

```
1 if ( x > 0 )
2   absx = x;
3 elseif ( x < 0 )
4   absx = -x;
5 else
6   absx = 0.0;
7 end
```

If some statements are to be repeated until a condition is true, use the `while()` statement:

```
1 while ( 1.0 < x )
2   x = x / 2.0;
3 end
```

Typical conditions include:

```
1  if ( a == 1 ) % Note the double equals sign here!  
2  if ( b ~= 2 ) % Note the "twiddle" for "not"  
3  if ( c < 3 )  
4  if ( d >= 4 )
```

## 5 Vectors

A numerical vector  $\vec{v}$  is stored as a MATLAB array. Examples of creating numerical vectors in MATLAB:

```
1  q = [] % Creates an empty vector  
2  r = rand ( 6, 1 ) % Column vector of random values  
3  s = randn ( 1, 5 ) % Row vector of normal random values  
4  t = ones ( 4, 1 ) % vector of 1's  
5  u = zeros ( 3, 1 ) % vector of 0's  
6  v = randperm ( 8 ) % random permutation of 12345678  
7  w = 10 : 20 % integers from 10 to 20  
8  x = linspace ( 0.0, 1.0, 7 )  
9  y = [ 1.1, 2.2, 3.3 ] % row vector of 3 values  
10 z = [ 4.4; 5.5; 6.6 ] % column vector of 3 values
```

Row and column vectors are different objects in MATLAB. A row vector has a shape (1,n), while a column vector has a shape (m,1). You can find the shape of any array with the **size()** command:

```
1  size ( x )
```

and you can save these values using a command like:

```
1  [ m, n ] = size ( x )
```

The **length()** command returns the number of elements of a vector; for a row vector, it returns n, for a column vector, m;

Each entry in a vector has an index; the first entry in the vector  $a$  is  $a(1)$ . You can print, use, or alter any element of a vector by indexing it.

```
1  u(2) = 20
```

A *colon* can be used to refer to a range of vector indices:

```
1  w(3:8)
```

We can specify a stepsize between successive indexes:

```
1  w(3:2:8)
```

Most arithmetic functions can be applied to a vector, giving a vector of results, such as **abs(x)**, **cos(x)**, **exp(x)**, **log(x)**, **sin(x)**, **sqrt(x)**. The **'** operator will transpose a vector from one form to the other.

```
1  rt = r'
```

Some functions return a single result based on all the vector values, such as **max(x)**, **mean(x)**, **min(x)**, **norm(x)**, **std(x)**, **var(x)**.

For pairs of vectors, the operator **'\***' requests a form of the vector dot product. To get the desired scalar result, the vectors must be written so that their dimensions have the form (1xn) \* (nx1), that is, row vector times column vector. If  $x$  and  $y$  are column vectors, as is common in mathematics convention, then the dot product is  $x' * y$ . If both are row vectors, then the express  $x*y'$  is necessary.

When applied to vectors, the operations **\***, **/** and **^** can be preceded by a period, indicating that the operation is to be applied element by element, returning a vector of results.

```

1 x .* y <— returns a vector of elementwise products .
2 x ./ y <— returns a vector of elementwise fractions .
3 x .^ 2 <— returns a vector of elementwise squares ;
4 x .^ y <— returns a vector of elementwise powers

```

## 6 Matrices

An  $m \times n$  numerical matrix can be stored in a MATLAB array. There are many commands to create such arrays:

```

1 O = pascal ( 5 )
2 Q = []
3 R = rand ( 3, 4 )
4 S = randi ( 5, 3 )
5 T = randn ( 2, 2 )
6 U = ones ( 5, 3 )
7 V = zeros ( 4, 3 )
8 W = eye ( 3, 3 )
9 X = [ 1, 2, 3; 4, 5, 6 ]
10 Y = [ 1, 2;
11      3, 4 ]
12 Z = magic ( 4 )

```

Note that MATLAB programmers typically use a capital letter to represent a matrix, The `size()` command returns the array dimensions:

```

1 [ m, n ] = size ( x )

```

Matrix values are accessed by specifying the row and column index, typically written as (i,j). You can print, use, or alter any element of a vector by indexing it.

```

1 U(2,1) = 20

```

The colon can be used to select a portion of the array:

```

1 R(2,2)      % a single entry
2 R(2,1:4)    % the second row
3 R(1:2,3)    % part of the third column
4 R(2:3,2:3)  % a 2x2 submatrix

```

Most arithmetic functions can be applied to an array, giving an array of results: `abs(X)`, `cos(X)`, `exp(X)`, `log(X)`, `norm(X)`, `sin(X)`, `sqrt(X)` The `'` operator will transpose an array from one form to the other.

```

1 Xt = X'

```

Some functions return a row vector of results, by applying the operation separately to each column, including `max(X)`, `mean(X)`, `min(X)`, `std(X)`, `var(X)`;

To multiply a matrix A times a vector x, we write

```

1 y=A*x

```

If A is  $m \times n$ , then x should be  $n \times 1$  and the result y will be  $m \times 1$ .

To multiply a matrix A times a matrix B, we write

```

1 C = A * B

```

If A is  $m \times n$ , B must be  $n \times o$ , and C will have size  $m \times o$ .

In some cases, it may be necessary to use the transpose operator, writing expressions like

```
1 u=A'*x
2 v=B*y'
3 w=C'*z'
```

When applied to matrices, the operations `*`, `/` and `^` can be prefaced by a dot to indicate that the operation is to be applied element by element, returning a matrix of results.

```
1 x .* y <- returns a matrix of elementwise products.
2 x ./ y <- returns a matrix of elementwise fractions.
3 x .^ 2 <- returns a matrix of elementwise squares;
4 x .^ y <- returns a matrix of elementwise powers
```

## 7 Linear Algebra

Scalar functions of a matrix include the condition number, determinant, and rank: `cond(A)`, `det(A)`, `rank(A)`, but only condition number is really useful and reliable;

Vector functions include returning the diagonal elements of a matrix: `diag(A)`.

Matrix functions include the inverse, the LU factorization, the pseudoinverse, and the QR factorization: `inv(A)`, `lu(A)`, `pinv(A)`, `qr(A)` Note that computing the inverse matrix `inv(A)` to solve a linear system is numerically unreliable and unnecessary.

The set of linear equations  $Ax=b$  is often posed, with the vector  $x$  unknown. If  $A$  is  $n \times n$  and nonsingular, while  $b$  is  $n \times 1$ , then a unique solution may be expected. MATLAB can provide that solution by the command

```
1 x = A \ b
```

## 8 Reading and Writing Data Files

Commands are available to save all of your data, or to restore it. To save a single item  $x$  item in a text file `my_x.txt`:

```
1 save ( 'my_x.txt', '-ascii', 'x' )
```

and then restore it by

```
1 x = load ( 'my_x.txt' )
```

Similarly, a text file `my_data.txt`, containing  $m$  rows and  $n$  columns of data, can be loaded into a MATLAB variable by

```
1 A = load ( 'my_data.txt' )
```