What is MATLAB and how to start it up?

- MATrix LABoratory
- Object-oriented high-level interactive software package for scientific and engineering numerical computations
- Enables easy manipulation of matrix and other computations
- Can handle symbolic computation, but is not specifically tailored to do that.

To start up MATLAB on CDF (teach.cs) type

`% matlab -nodesktop` or `% matlab &` or
`% matlab -nodesktop -softwareopengl`

at the Unix shell prompt.

The `matlab -nodesktop` command starts up MATLAB in your current window (most likely an `xterm`), and can work on any ASCII terminal; it gives you a simple and light ASCII interface, still including plots, assuming you are running Xwindows. The `-softwareopengl` option sets the software for viewing graphs.

The `matlab` command starts up MATLAB in a new window of the console you are on, assuming you are running Xwindows; it gives you a more powerful, but also complex and heavy interface, the **desktop**.

The standard MATLAB prompt is >> (while in Maple it is a single >).

In this document, whatever follows >> on a line starting with >> is typed by the user at the MATLAB prompt. To quit MATLAB, type

`>> quit` or `>> exit`

 © C. Christara, 2026-27

An example of MATLAB code

```
A = [1 2 3; 4 5 6];   % define the entries of a 2x3 matrix
x = [-1; 0; -2];  % define the entries of a 3x1 vector
b = A*x           % compute and output the product A*x
                  % and save it in b
```

The above piece of code sets the entries of a 2×3 matrix A and a 3×1 vector x, then computes the matrix vector product Ax and sets the result to (2×1 vector) b. Thus,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \; x = \begin{bmatrix} -1 \\ 0 \\ -2 \end{bmatrix}, \; b = Ax = \begin{bmatrix} -7 \\ -16 \end{bmatrix}$$

- The assignment operator is = (In Maple, it is `:=`).
- The semicolon (;) causes suppression of echoing the result of assignment/computations. (In Maple, the semicolon causes echoing the result and the colon (:) causes suppression of echoing). (See exception)
- The newline (<return> or <enter>) denotes the end of a statement. (The comma can also be used to denote the end of a statement.) If no semicolon precedes the newline, the result of the statement will be echoed. (In Maple, a statement ends with colon or semicolon. The newline does not denote anything specific.) (See exception)

 © C. Christara, 2026-27

- Exception: when the semicolon or the newline are inside the brackets' list of the matrix entries, they both denote end of a matrix row.
- MATLAB is case sensitive, i.e. variable A is different than variable a. (Maple is case sensitive too.)
- Whatever follows the % symbol on a line is a comment. (In Maple, the comment symbol is #)
- When a MATLAB statement is too long to fit on one line, it can be split in two (or more) lines using the ... continuation mark in each but the last line of the statement (see example on page after the next). (In Maple, if there is no colon or semicolon, it is assumed that the statement continues in the next line.)

 © C. Christara, 2026-27

Using MATLAB interactively

Recall that whatever follows >> on a line starting with >> is typed by the user at the MATLAB prompt. The rest is output echoed by MATLAB.

```
>> A = [1 2 3; 4 5 6];
>> x = [-1; 0; -2];
>> b = A*x
b =
    -7
   -16
>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
>> x = [-1; 0; -2]
x =
    -1
     0
    -2
```

 © C. Christara, 2026-27

```
>> 100 + (32-17)*5 + 2^3 + exp(1.1)
ans =
  186.0042
>> exp(1.1)
ans =
    3.0042
>> sqrt(-1)
ans =
        0 + 1.0000i
>> % statement split in two lines
>> avariablewithalongname = (32-17)*5 + sin(0.3*pi) ...
                          + 2^3 - log(10)/log(2)
avariablewithalongname =
  80.4871
```

- Exponentiation is carried by the hat symbol `^` (In Maple, it is carried by the hat or the double star `**` symbol.)

- MATLAB can be used as a simple calculator, i.e. it is not needed to assign the result of a computation to a variable. A default variable, named `ans`, is used to save the result of the latest computation or expression. (In Maple, the above is also true, but the name of the default variable is percent `%` in Maple 7 and doublequote `"` in Maple V.)
- MATLAB has all the standard mathematical functions, e.g. exp, log (stands for ln), log10, log2, sin, cos, tan, asin, acos, atan, sqrt, sinh, cosh, abs, max, min, etc, and many more built-in functions that compute a variety of mathematical results. (In Maple, the above is also true, but the names of some functions are different, e.g. ln (same as log), arcsin, arccos, etc.)
- Complex numbers are handled just as easily as real ones. Complex numbers are output in the form a + bi. The variables `i` and `j` are by default set to the value $\sqrt{-1}$. If, though, the user overwrites them, the value $\sqrt{-1}$ is lost. It can be recovered through `i = sqrt(-1)` and `j = sqrt(-1)`.
- The variable `pi` is by default set to the value of the number π. If, though, the user overwrites it, then the value of π is lost. It can be recovered through `pi = 4*atan(1)`. (In Maple, `pi` is `Pi`.)
- To clear the variable name `a` use `clear a`, while to clear all variable names use `clear`. (In Maple, use `unassign`.)

Matrices (arrays)

MATLAB requires neither the dimension of matrices nor the type of their entries to be specified. The simplest way to declare a matrix is to directly list its entries (row-by-row) enclosed in square brackets and assign the result to a variable. For example,

```
>> A = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]
```

will result in the output

```
A =
     1     2     3
     4     5     6
     7     8     9
```

Matrix entries in the same row are separated by one or more blank spaces (or by a comma), while rows are separated by a semicolon. If the matrix is large, we can put each row in one line. That is,

```
>> A = [ 1 2 3
         4 5 6
         7 8 9 ]
```

will give us the same result, as above.

To refer to a matrix entry that has already been assigned a value, type the matrix variable name followed by `(i, j)`, where `i` is the row index and `j` is the column index of the entry. For example, assuming the previous declaration of the matrix `A`,


```
>> A(2, 3)
```

results in

```
ans =
     6
```

The dimension of matrices in MATLAB is set dynamically. It can be changed on demand. Assuming the previous declaration of the matrix `A`, executing

```
>> A(4, 3) = 10
```

will give

```
A =
     1     2     3
     4     5     6
     7     8     9
     0     0    10
```

The size of the matrix is changed automatically to accommodate the new element, and the old values are kept. Any undefined elements are set to 0.

The built-in function `size` gives the size of a matrix. Assuming `A` is an already defined matrix, the multiple assignment statement

```
>> [m, n] = size(A)
```

assigns to `m` and `n` the number of rows and columns of `A`, respectively.

In MATLAB, it is easy to extract parts of the matrix. In general,

```
>> A(i:j, k:l)
```

gives the submatrix of `A` defined by rows `i` through `j` and columns `k` through `l` of matrix `A`. If the index is just a colon (`:`), without numbers that define a range, all rows or all columns of the matrix are included. For example,

```
>> A(:, k:l)
```

gives the submatrix of `A` defined by columns `k` through `l` of all rows of matrix `A`.

In general, the colon can be used to generate a sequence of numbers forming a vector (or one-dimensional matrix). Thus,

```
>> m:k:n
```

gives a list of numbers, starting from `m` and ending to `n` with stepsize `k` and

```
>> m:n
```

gives a list of numbers, starting from `m` and ending to `n` with stepsize 1.

Matrix concatenation can also be easily done. If `A` and `B` are two matrices of the same number of rows,

```
>> C = [A B]
```

creates a matrix `C`, with the same number of rows as `A` and `B`, and number of columns the sum of the numbers of columns of `A` and `B`. The columns of `C` consist of the columns of `A` (to the left) followed by the columns of `B` (to the right). On the other hand, if `A` and `B` are two matrices of the same number of columns,

```
C = [A; B]
```

creates a matrix `C`, with the same number of columns as `A` and `B`, and number of rows the sum of the numbers of rows of `A` and `B`. The rows of `C` consist of the rows of `A` (top) followed by the rows of `B` (bottom).

Examples:


```
>> A = [
     1     2     3
     4     5     6
     7     8     9
     0     0    10];
>> B = A(1:2, 2:3)
B =
     2     3
     5     6
>> C = [-1; -4]
C =
    -1
    -4
>> D = [B C]
D =
     2     3    -1
     5     6    -4
```



```
>> x = [11 12 13]
x =
    11    12    13
>> E = [x; D]
E =
    11    12    13
     2     3    -1
     5     6    -4
```

MATLAB has many built-in functions and operators for matrices. For example,

```
>> iA = inv(A);
```

computes the inverse of matrix `A` (if A is already defined as a square matrix and if the inverse exists), and saves it in variable `iA`. Also,

```
>> A'
```

displays the transpose of `A`, while

```
>> I3 = eye(3, 3);
```

saves the 3×3 identity matrix in variable `I3`.

MATLAB takes a high-level approach when asked to solve linear systems. If `A` is a solvable $n \times n$ matrix and `b` a $n \times 1$ vector,

```
>> x = A\b;
```

will compute the solution `x` to the system `Ax = b`.

Example:

```
>> A = [1 2 3; 5 4 6; 7 8 9]
A =
     1     2     3
     5     4     6
     7     8     9
>> b = [14; 31; 50]
b =
    14
    31
    50
>> x = A\b
x =
    1.0000
    2.0000
    3.0000
```


For loops

The most typical syntax of the for loop is of the form

```
for i = j:k:l
    statements
end
```

where i, the loop counter, starts from j, and proceeds to l, with stepsize k. If k is missing, i.e. the for statement looks like `for i = j:l`, the stepsize takes the default value 1. Note that none of the i, j, k and l variables needs to be integer.

Examples: What values does a take in the following loops?

```
for a = 0.1:0.1:0.5, statements, end
```

0.1, 0.2, 0.3, 0.4, 0.5

```
for a = 0.1:0.2:0.6, statements, end
```

0.1, 0.3, 0.5

```
for a = 0:-2:-10, statements, end
```

0, -2, -4, -6, -8, -10

```
for a = 0:-2:10, statements, end
```

None! To be precise, it takes the value of the empty matrix. Thus the statements of the loop (whatever they were), do not get executed at all.

A two-level nested loop: (assume n is already assigned a value)

```
for i = 1:n
    statements that possibly use i
    for j = 1:n
        statements that possibly use i and j
    end
end
```

The `break` statement can be used to terminate the loop prematurely.

If (conditional) statement, relational and logical operators

The general form of the if statement is

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

The statements following the (first) `if` are executed, if the (first) expression of the `if` statement is true. In MATLAB, any non-zero real expression is true and 0 corresponds to false. Thus, the statements following the (first) `if` are executed, if the real part of the (first) expression is non-zero (or is a matrix with all non-zero elements).
The statements following each `elseif` are executed, if the expression of the respective branch is true and none of the previous branches resulted in execution. The `else` branch is executed if none of the previous branches resulted in execution. The `else` and `elseif` parts are optional. Zero or more `elseif` parts can be used as well as nested `if`'s.

A simple form of the expression is `expr relop expr` where relop (relational operator) is `==, <, >, <=, >=`, or `~=`.
(In Maple, the respective relops are `=, <, >, <=, >=`, or `<>`.)
Example:

```
if I == J
    A(I, J) = 2;
elseif abs(I-J) == 1
    A(I, J) = -1;
else
    A(I, J) = 0;
end
```

What would the result of the above statement be, if it is inserted in a two-level nested for loop with both counters I and J starting at 1 and proceeding to 7 with stepsizes 1?

 © C. Christara, 2026-27

The standard logical operators are `&, |` and `~`.
(In Maple, they are `and, or` and `not`, respectively.)
With the help of logical operators, we can build more complex expressions for if statements.
Example:

```
if (I <= J) & (J-3 < I)
    A(I, J) = 7-J+I;
end
```

What would the result of the above statement be, if it is inserted in a two-level nested for loop with both counters I and J starting at 1 and proceeding to 7 with stepsizes 1?

Consider also the conditional statements

```
if (I <= J) & (J-3 < I)
    A(I, J) = 7-J+I;
elseif (J-4 < I)
    A(I, J) = -1;
end
```

```
if (J-4 < I)
    A(I, J) = -1;
elseif (I <= J) & (J-3 < I)
    A(I, J) = 7-J+I;
end
```

 © C. Christara, 2026-27

```
A =
     2    -1     0     0     0     0     0
    -1     2    -1     0     0     0     0
     0    -1     2    -1     0     0     0
     0     0    -1     2    -1     0     0
     0     0     0    -1     2    -1     0
     0     0     0     0    -1     2    -1
     0     0     0     0     0    -1     2
A =
     7     6     5     0     0     0     0
     0     7     6     5     0     0     0
     0     0     7     6     5     0     0
     0     0     0     7     6     5     0
     0     0     0     0     7     6     5
     0     0     0     0     0     7     6
     0     0     0     0     0     0     7
```

 © C. Christara, 2026-27

```
A =
     7     6     5    -1     0     0     0
    -1     7     6     5    -1     0     0
    -1    -1     7     6     5    -1     0
    -1    -1    -1     7     6     5    -1
    -1    -1    -1    -1     7     6     5
    -1    -1    -1    -1    -1     7     6
    -1    -1    -1    -1    -1    -1     7
A =
    -1    -1    -1    -1     0     0     0
    -1    -1    -1    -1    -1     0     0
    -1    -1    -1    -1    -1    -1     0
    -1    -1    -1    -1    -1    -1    -1
    -1    -1    -1    -1    -1    -1    -1
    -1    -1    -1    -1    -1    -1    -1
    -1    -1    -1    -1    -1    -1    -1
```

 © C. Christara, 2026-27

Functions and scripts in MATLAB

Functions in MATLAB can be either built-in or user-defined. Functions are saved in **M-files**, called so because the filename ends in `.m`. Functions may have one or more input parameters and may return one or more output variables. Currently, only one function (and its subfunctions, if any) can be defined per M-file.

A M-file does not always need to be a function. It can also be a (possibly long) sequence of MATLAB statements, i.e. a script. Scripts have neither formal input parameters, nor formal output variables.

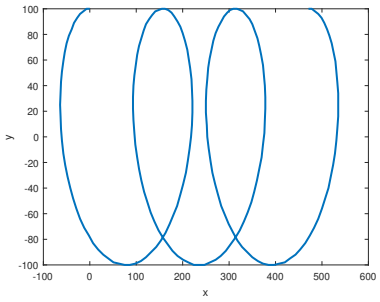
- To execute a function M-file, type (after the MATLAB prompt) the name of the file without the `.m` extension, followed by the input arguments in parentheses.
- Should you need to save the output variables, precede the call by the output variables (in square brackets if more than one) and the '=' sign.
- To execute a script M-file, just type (after the MATLAB prompt) the name of the file without the `.m` extension.

Examples:

Assume the following is in file `trochoid1.m` in the current directory

```
function [ix, iy] = trochoid1(a, r, U, n)

du = U/n;
x = 0;            % x = a*u0 - r*sin(u0);
y = r;            % y = r*cos(u0);
u = 0;            % u = u0;
ix(1) = round(x);
iy(1) = round(y);
for i = 2:n+1
    u = u + du;
    x = a*u - r*sin(u);
    y = r*cos(u);
    ix(i) = round(x);
    iy(i) = round(y);
end
```



Assume also the following is in file `script.m` in the current directory

```
a = 25; r = 100;
U = 6*pi; n = 200;
[ix1, iy1] = trochoid1(a, r, U, n);
plot(ix1, iy1, '-', 'LineWidth', 2); xlabel('x'); ylabel('y');
print -depsc trochoid.eps % print trochoid.ps
```

Assume now we start MATLAB, and type

```
>> script
```

What will happen?

- Variables `a, r, U` and `n` are set to the respective values.
- `pi` gets the default value of the number π.
- The trochoid1 function is called with input arguments `a, r, U, n`, and output arguments `ix1, iy1`.
- `ix1, iy1` get their value through the call to the function trochoid1.
- `a, r, U` and `n` keep the value they had before the call to trochoid1.
- All other variables used in the function trochoid1 are local by default (i.e. they do not keep their value outside the scope of trochoid1).

- `ix1, iy1` are vectors of size `n+1` and denote (integer) Cartesian coordinates of the pixels rendering a trochoid.
- The y-coordinates are plotted versus the x-coordinates with a solid line. The plot appears in an X-window.
- The plot is saved in file `trochoid.eps` in encapsulated postscript format, and can be included in Latex code. The .ps file can be printed on the CDF printer by the Unix command (after the user quits MATLAB)

```
% lpr trochoid.ps
```

On-line help

Type

```
>> help
```

to get all the help topics and

```
>> help topic
```

or

```
>> doc topic
```

to get information on any specific MATLAB topic. For example,

```
>> help inv
```

will tell you how to use the `inv` built-in function that gives the inverse of a matrix. To get on-line help through a html window environment, type

```
>> helpwin
```

Sparse matrices

Sparse storage schemes for matrices and sparse matrix computations are ubiquitous in high-performance scientific computing. In matlab, sparse matrices are easily created, and once a matrix is sparse, any computation with it will be carried by sparse matrix computation techniques.

The identity matrix is sparse, and can be created and stored in variable `II` (or any name you wish) by

```
>> II = speye(n, n);
```

where `n` already has the desired size value by some previous statement.

A type of sparse matrices that often arise in practical applications are banded matrices. Often (but not always) these have the same pattern along each diagonal.
Example:

```
n=7; e = ones(n,1); A = spdiags([1*e -2*e 1*e], [-1 0 1], n, n);
```

creates and stores in `A` a tridiagonal matrix of size `n`, with -2's on the main diagonal, and 1's in the first sub- and super-diagonal.

Although the function `sparse()` takes a matrix declared as regular (dense) and converts it to a sparse matrix, that form should never be used as it is very inefficient in memory usage. Matrices that are expected to be sparse should be created and stored as sparse from the very beginning.

For example, setting

```
>> II = eye(n, n);
>> II = sparse(II);
```

results in a sparse identity matrix `II`, but, temporarily at least, wastes n^2 positions (of which only n are nonzero).

To visualize the sparsity pattern of a sparse matrix `A`, the matlab function `spy(A)` can be used. Sparsity pattern gives where the nonzeros are, not their values.

Vectorization

While for loops are typical, and occasionally unavoidable, in many algorithms, whenever possible, we prefer a coding style refereed to as *vectorization* (or vectorized style) for more efficient computations and more elegant code.
Example: The piece of code

```
n = 100;
for i = 1:n
    v(i) = i^2;
end
for i = 1:n
    iv(i) = 1/v(i);
end
```

defines a vector `v` of 100 components with the ith component being the square of `i = 1, ..., n`, and the vector `iv` of the respective inverses.
This can be done more efficiently and elegantly by

```
n = 100; v = [1:n].^2; iv = 1./v;
```

Note the use of colon, instead of a loop.
Note the use of dot (.) before an operation, such as exponentiation or division, to denote the operation is applied componentwise.
What would the result of

```
v.*iv
```

be? And what of

```
sqrt(v).^2
```

Example with vectorization and sparse matrices

```
n = 1000; v = [1:n]'; v2 = v.^2; iv = 1./v;
A = spdiags([v v2 iv], [-1 0 1], n, n);
B = spdiags([v 2*v v2 2*iv iv], [-2 -1 0 1 2], n, n);
```