

Introduction to MATLAB

Numerical Mathematics (TMA4215)

Eirik Hoel Høiseth

21 August, 2014



NTNU
Norwegian University of
Science and Technology

What is MATLAB?

- High level programming language.
- Simple to use, but significantly slower than lower level languages like C++ and Java.
- Initially a tool for matrix computations (MATrix LABoratory), that evolved into a platform for scientific computing.
- Huge scientific library.
- Expensive license.



Numbers

MATLAB supports all the usual **numbers**:

- **Integers**
- **Real numbers**
- **Complex numbers**

Can mix integers, complex numbers, and real numbers as we like.

Example:

```
>> 3.47*4 - (5+4i)
ans =
    8.8800 - 4.0000i
```

Note: Both **i** and **j** can be used for the imaginary unit.



Storage of numbers

- By default an integer is treated and stored as a real number.
- Real numbers are, by default, stored with **double precision** (roughly 16 digits accuracy).
- A complex number is stored as 2 real numbers (real and imaginary part).



Variables

Definition: Variable

A **variable** is a **reference** to an object. The **assignment operator**, **=**, is used to assign a value to a variable.

Note: Variables need not be declared before they are used.

Example: Assignment

```
>> x = 2;  
>> y = x % x and y refer to different objects  
y =  
    2
```



Arrays

Definition: Arrays

An **array** is a **data structure**. It consists of a collection of elements. In an array of **dimension** n the position of an element in memory is given by an ordered set of n **indices**.

Note: In MATLAB all variables refer to arrays.



Definition: Scalars

Definition: Scalar

A **scalar** in MATLAB is a **0-dimensional array** holding a single value.

Example: Creating numerical scalars

```
>> x = 3.4;  
>> y = 5+4i;
```



Vectors

Definition: Vector

A **vector** in MATLAB is a **1-dimensional numerical array**.

Arbitrary vectors can be created using the **square bracket []**

We distinguish between two types:

- **Row vector**: Elements separated by a **comma** or **space**
- **Column vector**: Elements separated by a **semicolon**.

Note: A column vector is often made by **transposing** a row vector.

Example: Creating vectors

```
>> x = [3,4,-1]; % Row vector of length 3
>> y = [3+i;2-2i]; % Column vector of length 2;
```



Generating Equally spaced vectors

The **colon operator**, `:`, is useful for creating equally spaced real vectors:

- `a:b` for $b \geq a$ creates $[a, a + 1, \dots, a + m]$, where $m = \lfloor b - a \rfloor$.
- `a:d:b` for real d creates $[a, a + d, \dots, a + md]$, where $m = \lfloor (b - a)/d \rfloor$, assuming $(b - a)/d \geq 0$.

An alternative is the built in MATLAB function `linspace`:

- `linspace(a,b,n)` creates a row vector of n equally spaced points between a and b .



Matrices

Definition: Matrix

A **matrix** in MATLAB is a **2-dimensional numerical array**.

Matrices can be created similarly to vectors using square brackets:

- Elements in a row are separated by a **comma** or **space**
- Rows are separated by a **semicolon**.

Note: Vectors and numeric scalars can be thought of as matrices.

Example: Creating matrices

```
>> A = [3 2 1; 4 7 -1]; % Matrix has 2 rows and 3 columns
```



Generating common matrices

Built in MATLAB functions used to generate some frequently used matrices:

- `zeros(n,m)` generates an $n \times m$ matrix of zeros.
- `ones(n,m)` generates an $n \times m$ matrix of ones.
- `eye(n)` generates an $n \times n$ identity matrix.
- `rand(n,m)` generates an $n \times m$ random matrix.

Note: Only need to state the length of one dimension if the matrix is square, e.g. `zeros(n)`.



Basic matrix operations

The most common operations are:

- + and − addition and subtraction.
- * matrix multiplication.
- ^ matrix power for square matrices.
- / and \ matrix right and left division
 - For matrices \mathbf{A} and \mathbf{b} , $\mathbf{A}\backslash\mathbf{b}$ solves the linear system $\mathbf{Ax} = \mathbf{b}$ for \mathbf{x} . Similarly \mathbf{b}/\mathbf{A} solves the linear system $\mathbf{xA} = \mathbf{b}$ for \mathbf{x} .
 - Preferable to multiplying by the inverse, e.g. $(\mathbf{A}^{-1})*\mathbf{b}$ and $\mathbf{b}*(\mathbf{A}^{-1})$ respectively.
- Putting a period before the operator gives the corresponding elementwise operations: `.*`, `./` and `.\`.



Useful vector operations

For a vector v

- `sum(v)` and `prod(v)` computes the **sum** and **product** respectively of the elements in v .
- `length(v)` gives the **length** of v .
- `max(v)` and `min(v)` finds the **maximum** and **minimum** of the elements in v .
- `norm` computes norms, e.g. `norm(v)` for the vector 2-norm.
- `diff(v)` computes a vector of differences of neighbouring elements in v .
- `sort(v)` **sorts** the elements in v in ascending order.

Note: Most of these are also applicable to matrices.



Useful matrix operations

For a matrix A

- A' and $A.'$ give the **conjugate transpose** and **transpose** respectively of A .
- `diag` can be used to generate diagonal matrices and extract diagonals from matrices.
- `size(A)` gives the size of A .
- `eig(A)` computes the **eigenvalues** and **eigenvectors** of A
- `cond(A)` computes the **condition number** of A

Note: Many others exist, and the listed ones often have additional optional features.



Indexing

Definition: Indexing

Indexing into an array is a means of **selecting a subset** of the elements.

MATLAB uses one-based indexing, which means the indices start at 1.

Note: Indexing is an important programming tool in MATLAB.



Indexing: Vectors

- To get the n -th element in a vector b , write $b(n)$
- Let a be a vector containing valid indices of the vector b . Then $b(a)$ and a have the same length, and element i of the vector $b(a)$ is element $a(i)$ of b . Special cases:
 - $b(1:n)$ gives the n first elements of b .
 - $b(\text{end}-n+1:\text{end})$ gives the n last elements of b
 - $b(m:n)$ with $m \leq n$ is the subvector from $b(m)$ and up to $b(n)$.

Example: Indexing vectors

```
>> x = [10 20 30]; % Vector to be indexed
>> x([3,1]) % New vector
ans =
    30    10
```



Indexing: Matrices

Definition: Subscript indexing

With **subscript indexing** for matrices, we give an index (or vector of indices) for each of the two dimensions, separated by comma, inside parentheses. The **first** index refers to the **rows**, and the **second** to the **columns**.

Note: This is the common way of indexing matrices.

Some specific cases:

- The element in the m -th row and n -th column of a matrix A is indexed as $A(m,n)$.
- $:$ is shorthand for $1:end$, and can be used to denote all rows or columns. E.g. $A(1:n,:)$ extracts the first n rows of A .



Indexing: Matrices

Definition: Linear indexing

With **linear indexing** we index as if the matrix was a vector. The matrix being indexed is treated as if its elements are strung out in a long **column vector**.

Note: The resulting subvector has the same form as the index vector.



Example: Indexing

```
>> A = [1 3 5; 7 9 11]
A =
     1     3     5
     7     9    11
>> A(2,2:3) % Subscript indexing
ans =
     9    11
>> A(3:5) % Linear indexing
ans =
     3     9     5
```



Characters and strings

Definition: Character

A **character** is an interpreted nonnegative integer.

A character is written by enclosing it in "single" quotation marks, e.g. `'a'`.

Definition: String

A **string** is a vector of characters.

A string is written in the same way as a single character, e.g. `'This is a string'`, and can be indexed just like a normal vector.



Logicals

Definition: Logical

A **logical** is a data type which can only have the value **true** or **false**.

- **true** (logical 1) and **false** (logical 0) evaluate to 1 and 0 respectively when used in computations.
- Any real nonzero number becomes **true** while 0 becomes **false**, when converted to or used as a logical.

Logical operators:

- **&** (**and**) and **|** (**or**) combine logical values.
- **~** (**not**) gives the logical negation of the logical value that follows.

Note: Also work elementwise with matrices.



NTNU
Norwegian University of
Science and Technology

Logical expression

Logicals frequently arise as the result of a **logical expression**

Definition: Logical expression

A **logical expression** is an expression that can only evaluate to true or false.

A logical expression in MATLAB often involve the **relational operators**:

- **==** (equal), **~=** (not equal)
- **>** (greater than), **<** (less than).
- **>=** (greater than or equal), **<=** (less than or equal).

Note: Also work elementwise with matrices.



Programming

To create useful code, at some point it is no longer enough to execute one command manually at a time.

- Can gather command sequences in their own **MATLAB files**, which **end with ".m"**. E.g. "program.m"
- We distinguish between two types of files: **scripts** and **functions**.



Editor

To create scripts and functions you'll need an **editor**. You may use any plain text editor you prefer.

- MATLAB has a built in editor. To create or edit a file called "filename.m" with this, just type `edit filename` in the command line.
- Typing `edit` opens a new script file called "Untitled.m".
- The file you wish to edit should be in the active MATLAB folder.



Scripts

Definition: Script

A **script** is a simple **sequence of MATLAB commands** without any input or output arguments.

- Useful for automating a series of MATLAB commands, e.g. ones you have to perform repeatedly.
- To run a script, type the filename in the command window without the ending.



Functions

Definition: Function

Functions define a specific action that (usually) takes in some data, does some processing, and (usually) returns a result, like most math functions.

Note: Functions are an essential programming tool.



Anatomy of a function

```
function [out1,out2,..., outM] = functionName(in1, in2, ..., inN)
% Comments that document the usage and purpose of the function
.
. MATLAB commands % Function body
.
out1 = . . .
.
.
outM = . . .
.
.
end % Optional
```



Anatomy of a function

- The function must begin with the keyword **"function"**.
- Variable number of input and output arguments supported.
 - Can drop square brackets if single output argument.
 - Can drop square brackets and equal sign if no output argument.
 - Can drop parentheses if no input argument.
- Input arguments are **passed by value**.
- The name of the function must match the filename.
- The data type of input and output arguments are not stated, and any type of data can be used.
- All output arguments should be given a value inside the function.
- No return statement is needed.
- **return** can be used to force a return before the end of the function is reached.



Calling functions

Calling syntax:

```
[out1, out2, ..., outM] = myFun(in1, in2, ..., inN);
```

- The number N of input values should match the number of input arguments.
- If $N = 0$ the parentheses can be skipped.
- M must be equal or smaller than the number of output arguments for the function.
 - If M smaller, only the first M output arguments of the function are returned.
 - For $M = 1$ the bracket can be skipped.
 - If no output is desired, the bracket and equality sign can be skipped.



Scripts vs. functions

- Both are plain text files containing Matlab commands, with ".m" file endings.
- **Scripts** run in the **same scope** as the command window. Created variables gets added to the workspace, and can replace and alter variables already there.
- **Functions** define a **separate scope** than the command window. Actions in the function do not impact the MATLAB workspace.



Anonymous functions

Definition: Anonymous function

An **anonymous function** is a function not stored in a program file. It must contain a single statement.

Syntax: `@(in1,in2, ..., inN) Statement;`

The anonymous function can be assigned to a variable and evaluated as a normal function using the variable name as the function name.



Example: Anonymous functions

```
>> CircleArea = @(r) pi*r.^2 % Create anonymous function
CircleArea =
    @(r)pi*r.^2
>> radius = 1:0.2:2; % Vector of radius values
>> A = CircleArea(radius) % Evaluate the area for all radii
A =
    3.1416    4.5239    6.1575    8.0425   10.1788   12.5664
```



Function handles

Definition: Function handle

A **function handle** is one of the standard MATLAB data types. It enables calling a function indirectly.

- You can pass a function to another function by passing a **function handle** to it.
- When an **anonymous function** is assigned to a variable, the variable holds a function handle. To pass the function, just pass the variable.
- For a **regular function**, a function handle must be created before it can be passed. This is done using `@`, e.g. `@MyFun` creates a handle for the function `MyFun`
- The passed function can be called as normal inside the other function.



Control structures

Definition: Control structure

A **control structure** allows the program to repeat code, take decisions and branch out.

- We will only consider the two most common control structures, the **if sentence** and **for loop**
- MATLAB also have the common control structures from other languages like **C++** and **Java**: **switch**, **while**, **continue**, **break**, **return**



If sentence

Definition: If sentence

An **if sentence** executes statements if an expression is **true**.

Syntax:

```
if expression1
    statements % Is performed if expression1 is true
elseif expression2 % Optional. Checked if expression1 is false
    statements % Is performed if expression1 is false, expression2 is true
else
    statements % Optional default statement.
end
```



For loop

Definition: For Loop

A **for loop** lets us repeat a block of code a specific number of times. An incrementing **index variable** keeps track of the iterations.

Syntax:

```
for variable = vector
    statements
end
```

If the vector has length n , the block is performed n times. In iteration i , **variable** has the value **vector(i)**.



Vectorization

Definition: Vectorization

Vectorization means replacing loop-based, scalar-oriented code with equivalent code using matrix and vector operations.

Note: Vectorization normally gives code which is faster, cleaner and less prone to errors.



Elementary functions

- The elementary scalar functions are already implemented in MATLAB. These include: `exp`, `log`, `cos`, `sin`, `tan`, `asin`, `acos`, `atan`, `sqrt`, `abs`, `sign`, `fix`, `floor`, `ceil`, `round`.
- These are all **vectorized functions**, meaning they operate elementwise on matrix and vector arguments.
- It is usually a good idea to try to vectorize a scalar function if possible.



Simple 2D plotting

- The basic MATLAB `plot`-command cannot directly plot a function.
- Rather than ask it to plot a function, you:
 1. Evaluate the function on a vector of input values to generate a vector of corresponding function values.
 2. Ask MATLAB to plot the two vectors against each other.

Example: Plot

```
>> x = -1:0.01:1; % Generate input values
>> y = sin(x); % Generate corresponding function values
>> plot(x,y); % Plot vectors against each other
```



Other plot commands

- `fplot` essentially lets you plot a function directly over a given interval.
- `surf` and `plot3` can be used to plot surfaces and lines in 3D.
- `loglog`, `semilogx` and `semilogy` can make 2D plots with logarithmic axes.
- Other useful commands to make plots or help modify your plots include: `xlabel`, `ylabel`, `zlabel`, `title`, `legend`, `text`, `grid`, `hold`, `box`, `mesh`, `bar`, `quiver`, `meshgrid`, `contour`, `shading`, `subplot`, `figure`, `set`, `axis`



Navigating the function library

A major advantage of MATLAB is the extensive function library. Using these functions is an important part of writing functional and efficient MATLAB code. To help you navigate, the following commands are particularly useful:

- The **help pages** of a function `func` can be accessed by typing `help func` in the command window.
- To **look for functions** related to the keyword "key" type `lookfor key` in the command window.
- To **display documentation** for the functionality specified by "name" type `doc name` in the command window.

