Introduction to MATLAB

Markus Kuhn

Department of Computer Science and Technology
University of Cambridge

https://www.cl.cam.ac.uk/teaching/2223/TeX+MATLAB/
What is MATLAB

- high-level language (garbage collecting, var-len structures)
- BASIC-like syntax, with elements from C, GUI IDE
- basic data type: 2- or 3-dimensional floating-point matrix
- vectors are just 2D matrices with size 1 in one dimension
- all numbers are treated as floating-point values by default
- most operators and functions work on entire matrices
  ⇒ hardly ever necessary to write out loops
- uses internally highly optimized numerics libraries
  (BLAS, LAPACK, FFTW)
- comprehensive toolboxes for easy access to standard algorithms from
  many fields: statistics, machine learning, image processing, signal
  processing, neural networks, wavelets, communications systems
- very simple I/O for many data/multimedia file formats
- popular for experimental/rapid-prototype number crunching
- widely used as a visualization and teaching tool
What MATLAB is not

- not a computer algebra system
- not a strong general purpose programming language
  - limited support for other data structures
  - few software-engineering features;
    typical MATLAB programs are only a few lines long
  - not well-suited for teaching OOP
  - limited GUI features
- not a high-performance language (but fast matrix operators)
  got better since introduction of JIT compiler (JVM)
- not freely available (but local campus licence)
Open-source MATLAB alternatives

Similar to MATLAB, subset compatible: **GNU Octave, SciLab, FreeMat**

Other high-level languages for technical computing:

- **R** – focus on statistics and plotting
  
  [https://www.r-project.org/](https://www.r-project.org/)

- **Python** – a full-featured programming language. Modules:
  
  - *numpy* – numerical arrays, fast linear algebra
  - *matplotlib* – MATLAB-like plotting functions
    
    [https://matplotlib.org/](https://matplotlib.org/)
  
  - *SciPy* – scientific computing, *Pandas* – data analysis, etc.

- **Julia** – modern, fast, full-featured, compiled, interactive language
  
  [https://julialang.org/](https://julialang.org/)
  
  - MATLAB-inspired syntax (especially for arrays)
  
  - combines dynamic and static type systems, multiple dispatch
  
  - LLVM backend, can also call C, C++, Python, R, Fortran functions
  
  - LISP-like metaprogramming, rich flexible type system

- others: LuaJIT (SciLua), Perl Data Language (PDL), OCaml (Owl)

**Jupyter** is a popular browser-based notebook environment for recording and presenting experiments in Julia, Python, R, . . .
Local availability

MATLAB is installed and ready to use on

- Intel Lab, etc.: MCS Windows
- Intel Lab, etc.: MCS Linux (/usr/bin/matlab)
- CL MCS Linux server: ssh -X linux.cl.ds.cam.ac.uk
- MCS Linux server: ssh -X linux.ds.cam.ac.uk
- Computer Laboratory managed Linux PCs
  cl-matlab -> /usr/groups/matlab/current/bin/matlab

Campus license (666637) allows installation on staff/student home PCs
(Linux, macOS, Windows):

- Includes full suite of \( \approx 180 \) toolboxes: Statistics and Machine Learning, Signal Processing, Image Processing, Image Acquisition, Bioinformatics, Control Systems, Wavelets, Deep Learning, Audio System, GPU Coder, Instrument Control, RF, ...
- Installation requires setting up a [https://uk.mathworks.com](https://uk.mathworks.com) account with your @cam.ac.uk email address.
Full documentation built in: start `matlab` then type
- `doc` – to browse built-in hypertext manual
- `doc command` – to jump to a specific manual page (e.g. `plot`)
- `help command` – to show plain-text summary of a command

Read first: `doc → MATLAB → Getting Started`

Tutorial videos:

https://uk.mathworks.com/videos.html

Documentation also available online (HTML and PDF):
- https://uk.mathworks.com/help/matlab/
- https://uk.mathworks.com/help/ – toolboxes

Locally installed MATLAB may be behind the latest release. If you spot problems with the MCS MATLAB installation, please do let the lecturer know (→ mgk25@cl.cam.ac.uk).
MATLAB matrices (1)

Generate a “magic square” with equal row/column/diagonal sums and assign the resulting $3 \times 3$ matrix to variable `a`:

```matlab
>> a = magic(3)
a =
     8  1  6
     3  5  7
     4  9  2
```

Assignments and subroutine calls normally end with a semicolon. Without, MATLAB will print each result. Useful for debugging!

Results from functions not called inside an expression are assigned to the default variable `ans`.

Type `help magic` for the manual page of this library function.
Colon generates number sequence:

\[
\begin{align*}
\text{>> } & \quad 11:14 \\
\text{ans} &= \\
& \quad \begin{bmatrix} 11 & 12 & 13 & 14 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{>> } & \quad -1:1 \\
\text{ans} &= \\
& \quad \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{>> } & \quad 3:0 \\
\text{ans} &= \quad \text{Empty matrix: 1-by-0}
\end{align*}
\]

Specify step size with second colon:

\[
\begin{align*}
\text{>> } & \quad 1:3:12 \\
\text{ans} &= \\
& \quad \begin{bmatrix} 1 & 4 & 7 & 10 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{>> } & \quad 4:-1:1 \\
\text{ans} &= \\
& \quad \begin{bmatrix} 4 & 3 & 2 & 1 \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{>> } & \quad 3:-0.5:2 \\
\text{ans} &= \\
& \quad \begin{bmatrix} 3.0000 & 2.5000 & 2.0000 \end{bmatrix}
\end{align*}
\]

Access single matrix cell: \(a(2,3) = 7\). MATLAB uses 1-based indexing and column-major order (i.e., like FORTRAN/R/Julia, unlike C/C++).

Vectors as indices select several rows and columns. When used inside a matrix index, the variable \textit{end} provides the highest index value:
\(a(\text{end}, \text{end}-1) = 9\). Using just "::" is equivalent to "1:end" and can be used to select an entire row or column.
Select rows, columns and submatrices of a:

```matlab
>> a(1,:)  ans =
           8  1  6

>> a(:,1)  ans =
           8
           3
           4

>> a(2:3,1:2)  ans =
           3  5
           4  9
```

Matrices can also be accessed as a 1-dimensional vector:

```matlab
>> a(1:5)  ans =
           8  3  4  1  5

>> a(6:end)  ans =
            9  6  7  2

>> b = a(1:4:9)  ans =
             8  5  2

>> size(b)  ans =
            1  3
```
Use [ ] to build new matrices, where , or space as a delimiter joins submatrices horizontally and ; joins them vertically.

```matlab
>> c = [2 7; 3 1]
c =
    2    7
    3    1
>> d = [a(:,end) a(1,:)']
d =
    6    8
    7    1
    2    6
>> e = [zeros(1,3); a(2,:)]
e =
    0    0    0
    3    5    7
```

Mask matrix elements:

```matlab
>> find(a > 5)
ans =
    1
    6
    7
    8
>> a(find(a > 5)) = 0
a =
    0    1    0
    3    5    0
    4    0    2
```
Each element of the matrix product is the scalar product of the corresponding row in the first factor and the corresponding column in the second factor.
Each element of the matrix product is the scalar product of the corresponding row in the first factor and the corresponding column in the second factor.
Each element of the matrix product is the scalar product of the corresponding row in the first factor and the corresponding column in the second factor.
Review: matrix multiplication

Each element of the matrix product is the scalar product of the corresponding row in the first factor and the corresponding column in the second factor.
Each element of the matrix product is the scalar product of the corresponding row in the first factor and the corresponding column in the second factor.
Each element of the matrix product is the scalar product of the corresponding row in the first factor and the corresponding column in the second factor.
Review: inner and outer product of vectors

Special cases of matrix multiplication

\[
\begin{pmatrix}
\bullet & \bullet & \bullet & \bullet & \bullet
\end{pmatrix}
\cdot
\begin{pmatrix}
\bullet \\
\bullet \\
\bullet
\end{pmatrix}
=
\begin{pmatrix}
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet
\end{pmatrix}
\]

Row vector times column vector:
Review: inner and outer product of vectors

Special cases of matrix multiplication

\[
(\bullet \bullet \bullet \bullet \bullet) \cdot \begin{pmatrix}
\bullet \\
\bullet \\
\bullet \\
\bullet \\
\bullet
\end{pmatrix} = (\bullet) = \bullet
\]

Row vector times column vector:
Review: inner and outer product of vectors

Special cases of matrix multiplication

\[(\bullet \bullet \bullet \bullet) \cdot \begin{pmatrix} \bullet \end{pmatrix} = (\bullet) = \bullet\]

Row vector times column vector: scalar product, dot product
Review: inner and outer product of vectors

**Special cases of matrix multiplication**

\[
(\bullet \bullet \bullet \bullet \bullet) \cdot \begin{pmatrix} \bullet \\
\bullet \\
\bullet \\
\bullet \\
\bullet \\
\end{pmatrix} = (\bullet) = \bullet
\]

Row vector times column vector: **scalar product, dot product**

\[
\begin{pmatrix} \bullet \\
\bullet \\
\bullet \\
\end{pmatrix} \cdot (\bullet \bullet \bullet \bullet \bullet)
\]

Column vector times row vector:
Review: inner and outer product of vectors

**Special cases of matrix multiplication**

\[(\bullet \bullet \bullet \bullet \bullet) \cdot \begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix} = (\bullet) = \bullet\]

Row vector times column vector: **scalar product, dot product**

\[
\begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix} \cdot (\bullet \bullet \bullet \bullet \bullet) = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \end{pmatrix}
\]

Column vector times row vector:
Review: inner and outer product of vectors

Special cases of matrix multiplication

\[
\begin{pmatrix}
\cdots \\
\end{pmatrix}
\begin{pmatrix}
\bullet \\
\end{pmatrix}
= \begin{pmatrix}
\bullet \\
\end{pmatrix}
= \bullet
\]

Row vector times column vector: scalar product, dot product

\[
\begin{pmatrix}
\bullet \\
\end{pmatrix}
\begin{pmatrix}
\bullet \\
\cdots \\
\bullet \\
\end{pmatrix}
= \begin{pmatrix}
\bullet \\
\bullet \\
\bullet \\
\bullet \\
\end{pmatrix}
\]

Column vector times row vector: matrix of all pair-wise products
MATLAB matrices (5)

Operators on scalars and matrices:

```
>> [1 1; 1 0] * [2 3]'
ans =
   5
   2

>> [1 2 3] .* [10 10 15]
ans =
    10    20    45
```

Inner and outer vector product:

```
>> [2 3 5] * [1 7 11]'
ans =
    78
>> [2 3 5]' * [1 7 11]
ans =
    2    14    22
    3    21    33
    5    35    55
```

The imaginary unit vector $\sqrt{-1}$ is available as both $1i$ and $1j$, and matrices can be complex.

Related functions: real, imag, conj, exp, abs, angle
Plotting functions `plot`, `semilogx`, `semilogy`, and `loglog` all expect a pair of vectors for each curve, with $x$ and $y$ coordinates, respectively.

Use `saveas(gcf, 'plot2.eps')` to save current figure as graphics file.
2D plotting

```matlab
xl = -20:0.3:20;
yl = -20:0.3:20;
[x,y] = meshgrid(xl, yl);
r = sqrt(x.^2 + y.^2);
s = sin(r) ./ r; s(find(r==0)) = 1;
plot3(x, y, s);
grid on;

imagesc(xl, yl, s, [-1 1]);
colormap(gray);
set(gca, 'DataAspectRatio', [1 1 1]);
```
Some common functions and operators

*, ^
  — matrix multiplication, exponentiation
/, \, inv
  \(A/B = AB^{-1}, \ A\backslash B = A^{-1}B, \ A^{-1}\)
+, -, .*, ./, .^
  — element-wise add/sub/mul/div/exp
==, ~=, <, >, <=, >=
  — relations result in element-wise 0/1
length, size
  — size of vectors and matrices
zeros, ones, eye, diag
  — all-0, all-1, identity, diag. matrices
xlim, ylim, zlim
  — set plot axes ranges
xlabel, ylabel, zlabel
  — label plot axes
audioread, audiowrite, sound
  — audio I/O
csvread, csvwrite
  — comma-separated-value I/O
imread, imwrite, image,
imagec, colormap
  — bitmap image I/O
plot, semilog\{x,y\}, loglog
  — 2D curve plotting
conv, conv2, xcorr
  — 1D/2D convolution,
cross/auto-correlation sequence
fft, ifft, fft2
  — discrete Fourier transform
sum, prod, min, max
  — sum up rows or columns
cumsum, cumprod, diff
  — cumulative sum or product,
differentiate row/column
find
  — list non-zero indices
figure, saveas
  — open new figure, save figure
To define a new function, for example \( \text{decibel}(x) = 10^{x/20} \), write into a file `decibel.m` the lines

```matlab
function f = decibel(x)
  % DECIBEL(X) converts a decibel figure X into a factor
  f = 10 .^ (x ./ 20);
```

Only the function that has the same name as the m-file in which it is defined can be called from outside the file; all other functions are only visible inside the file. The `function` keyword sets the variable whose value will be returned and lists the parameter variables.

The m-file must be in the current directory (`cd`) or MATLAB’s search path (`path`) to become accessible.

Use `edit db` to edit the m-file, `help db` to show the first comment lines and `type db` to show its source text.

M-files can also contain just sequences of statements instead of a function definition. These are called simply by typing their name.
Example: generating an audio illusion

Generate an audio file with 12 sine tones of apparently continuously exponentially increasing frequency, which never leave the frequency range 300–3400 Hz. Do this by letting them wrap around the frequency interval and reduce their volume near the interval boundaries based on a raised-cosine curve applied to the logarithm of the frequency.

First produce a 2 s long waveform in which each tone raises 1/12 of the frequency range, then concatenate that into a 60 s long 16-bit WAV file, mono, with 16 kHz sampling rate. Avoid phase jumps.

Parameters:

```
fs   = 16000;  % sampling frequency [Hz]
d   = 2;   % time after which waveform repeats [s]
n   = 12;  % number of tones
fmin = 300;  % lowest frequency
fmax = 3400;  % highest frequency
```
Spectrogram of the first 6 s:

Time (s)
0
0.5
1
1.5
2
2.5
3
3.5

Frequency (kHz)
0
0.5
1
1.5
2
2.5
3
3.5

Power/frequency (dB/Hz)
-140
-120
-100
-80
-60
-40

Frequency (kHz)
Time (s)
Example solution:

\[ t = 0:1/\text{fs} : d-1/\text{fs}; \quad \% \text{timestamps for each sample point} \]

\[ \% \text{normalized logarithm of frequency of each tone (row)} \]
\[ \% \text{for each sample point (column), all rising linearly} \]
\[ \% \text{from 0 to 1, then wrap around back to 0} \]

\[ l = \text{mod}(((0:n-1)/n)' \* \text{ones}(1, \text{fs} \* d) + \text{ones}(n,1) \* (t/(\text{d} \* n)), 1); \]

\[ f = \text{fmin} \* (\text{fmax}/\text{fmin}) .^ l; \quad \% \text{freq. for each tone and sample} \]

\[ p = 2\pi \* \text{cumsum}(f, 2) / \text{fs}; \quad \% \text{phase for each tone and sample} \]

\[ \% \text{make last column a multiple of } 2\pi \text{ for phase continuity} \]

\[ p = \text{diag}((2\pi \* \text{floor}(p(:,\text{end})/(2\pi))) ./ p(:,\text{end})) \* p; \]

\[ s = \sin(p); \quad \% \text{sine value for each tone and sample} \]

\[ \% \text{mixing amplitudes from raised-cosine curve over frequency} \]

\[ a = 0.5 - 0.5 \* \cos(2\pi \* l); \]

\[ w = \frac{\text{sum}(s \* a)}{n}; \quad \% \text{mix tones together, normalize to } [-1, +1] \]

\[ w = \text{repmat}(w, 1, 3); \quad \% \text{repeat waveform 3x} \]

\[ \text{spectrogram}(w, 2048, 1800, 2048, \text{fs}, 'yaxis'); \quad \% \text{plot} \]

\[ w = \text{repmat}(w, 1, 20); \quad \% \text{repeat waveform 20x} \]

\[ \text{audiowrite('ladder.wav', w, fs, 'BitsPerSample', 16); \quad \% \text{make audio file} \]

A variant of this audio effect, where each tone is exactly one octave (factor 2 in frequency) from the next, is known as the *Shepard–Risset glissando*.

What changes to the parameters would produce that?
<table>
<thead>
<tr>
<th></th>
<th>MATLAB</th>
<th>Julia</th>
<th>NumPy</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector size (1,n)</td>
<td>[1 2 3]</td>
<td>[1 2 3]</td>
<td>np.array([[1, 2, 3]]).reshape(1, 3)</td>
</tr>
<tr>
<td>vector size (n,1)</td>
<td>[1; 2; 3]</td>
<td>[1 2 3]'</td>
<td>np.array([[1, 2, 3]]).reshape(3, 1)</td>
</tr>
<tr>
<td>vector size (n)</td>
<td>n/a</td>
<td>[1, 2, 3]</td>
<td>np.array([1, 2, 3])</td>
</tr>
<tr>
<td>j to n step k</td>
<td>j:k:n</td>
<td>j:k:n</td>
<td>np.arange(j, n+1, k)</td>
</tr>
<tr>
<td>matrix</td>
<td>[1 2; 3 4]</td>
<td>[1 2; 3 4]</td>
<td>np.array([[1, 2], [3, 4]])</td>
</tr>
<tr>
<td>0 matrix</td>
<td>zeros(2, 2)</td>
<td>zeros(2, 2)</td>
<td>np.zeros((2, 2))</td>
</tr>
<tr>
<td>1 matrix</td>
<td>ones(2, 2)</td>
<td>ones(2, 2)</td>
<td>np.ones((2, 2))</td>
</tr>
<tr>
<td>identity matrix</td>
<td>eye(2, 2)</td>
<td>I</td>
<td>np.eye(2)</td>
</tr>
<tr>
<td>diagonal matrix</td>
<td>diag([1 2 3])</td>
<td>Diagonal([1, 2, 3])</td>
<td>np.diag([1, 2, 3])</td>
</tr>
<tr>
<td>transpose</td>
<td>A.'</td>
<td>transpose(A)</td>
<td>A.T</td>
</tr>
<tr>
<td>complex conj. transpose</td>
<td>A'</td>
<td>A'</td>
<td>A.conj()</td>
</tr>
<tr>
<td>concat hor.</td>
<td>[[1 2] [1 2]]</td>
<td>[[1 2] [1 2]]</td>
<td>B = np.array([1, 2])</td>
</tr>
<tr>
<td>matrix to vector</td>
<td>A(:)</td>
<td>A[:]</td>
<td>np.hstack((B, B))</td>
</tr>
<tr>
<td>flip left/right</td>
<td>fliplr(A)</td>
<td>reverse(A,dims=2)</td>
<td>np.flip(A)</td>
</tr>
</tbody>
</table>
| broadcast a function   | f=@(x) x.^2; f(x)           | f(x)=x^2; f(x)               | def f(x):
|                         |                             |                               | return x**2
| element A_{2,2}        | A(2, 2)                     | A[2, 2]                      | A[1, 1]
| rows 1 to 4            | A(1:4, :)                   | A[1:4, :)                    | A[0:4, :]
| matrix multiplication  | A * B                       | A * B                        | A @ B

https://cheatsheets.quantecon.org/