Introduction to MATLAB

Markus Kuhn

Computer Laboratory, University of Cambridge

https://www.cl.cam.ac.uk/teaching/2021/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
- most operators and functions work on entire matrices
  $\Rightarrow$ 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/
- *Python* – a full-featured programming language. Modules:
  - *numpy* – numerical arrays, fast linear algebra
  - *matplotlib* – MATLAB-like plotting functions
    https://matplotlib.org/
  - *SciPy* – scientific computing, *Pandas* – data analysis, etc.
- *Julia* – modern, fast, full-featured, compiled, interactive language
  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):

- `https://uk.mathworks.com/academia/tah-portal/the-university-of-cambridge-666637.html`
- Now (since 2018) includes *full suite* of ≈ 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/` account with your `@cam.ac.uk` email address.

The older Computer Laboratory license (136605) was replaced in 2018 by the University one.

Documentation

- 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 half a year 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 3×3 matrix to variable `a`:

```
>> 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.

MATLAB matrices (2)

Colon generates number sequence:

```
>> 11:14
ans =
    11   12   13   14

>> -1:1
ans =
    -1    0    1

>> 3:0
ans =
   Empty matrix: 1-by-0
```

Specify step size with second colon:

```
>> 1:3:12
ans =
     1    4    7   10

>> 4:-1:1
ans =
     4    3    2    1

>> 3:-0.5:2
ans =
    3.0000   2.5000   2.0000
```

Single matrix cell: `a(2,3) == 7`. Vectors as indices select several rows and columns. When used inside a matrix index, the variable `end` provides the highest index value: `a(end, end-1) == 9`. Using just ":" is equivalent to "`1:end`" and can be used to select an entire row or column.

MATLAB matrices (3)

Select rows, columns and submatrices of a:

```
>> 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:

```
>> 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
```


MATLAB matrices (4)

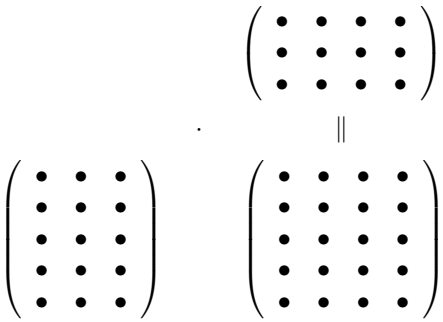
Use [ ] to build new matrices, where , or space as a delimiter joins submatrices horizontally and ; joins them vertically.

```
>> 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:

```
>> find(a > 5)
ans =
   1
   6
   7
   8
>>  a(find(a > 5)) = 0
a =
   0   1   0
   3   5   0
   4   0   2
```

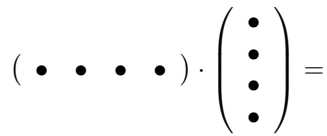

Review: matrix multiplication



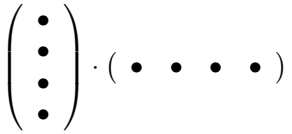
Each element of the matrix product is the scalar product of
the corresponding in the and
the corresponding in the

Review: inner and outer product of vectors

Special cases of matrix multiplication



Row vector times column vector:



Column vector times row vector:

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
```

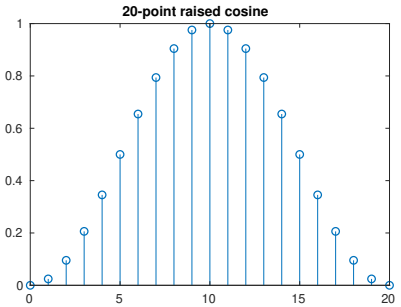
Inner and outer vector product:

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

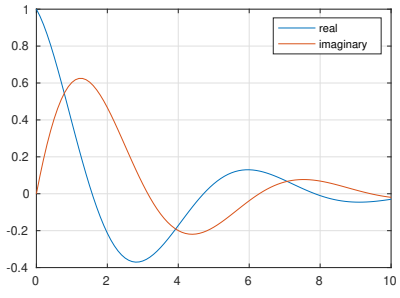
The imaginary unit vector $\sqrt{-1}$ is available as both `i` and `j`, and matrices can be complex.

Related functions: `real, imag, conj, exp, abs, angle`

Plotting




```
x = 0:20;
y = 0.5 - 0.5*cos(2*pi * x/20);
stem(x, y);
title('20-point raised cosine');
```

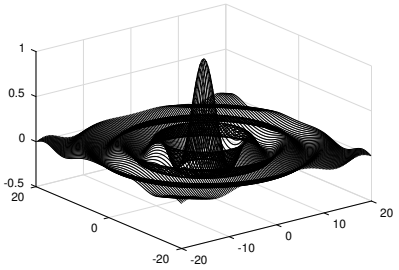



```
t = 0:0.1:10;
x = exp(t * (j - 1/3));
plot(t, real(x), t, imag(x));
grid; legend('real', 'imaginary')
```

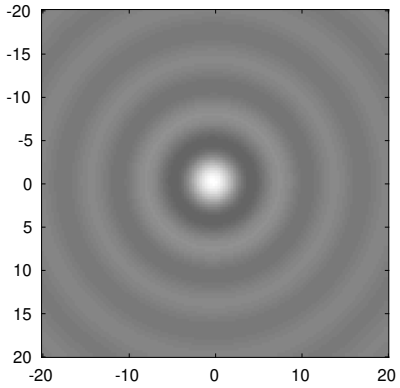
Plotting functions `plot`, `semilogx`, `semilogy`, `loglog` all expect a pair of vectors for each curve, with x and y coordinates, respectively.

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

2D plotting



```
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, imagesc, 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

Functions and m-files

To define a new function, for example `decibel`$(x) = 10^{x/20}$, write into a file `decibel.m` the lines

```
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

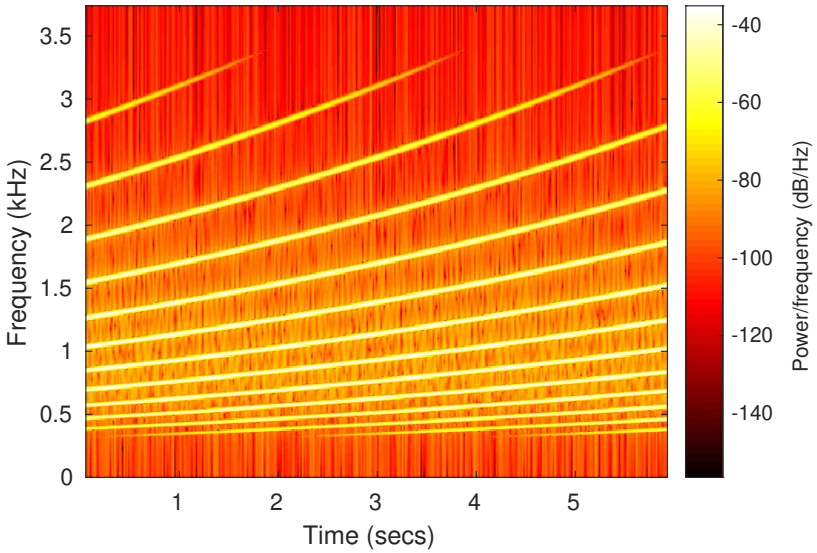
Generate an audio file with 12 sine tones of apparently continuously exponentially increasing frequency, which never leave the frequency range 300–3400 Hz. Do this by letting them wrap around the frequency interval and reduce their volume near the interval boundaries based on a raised-cosine curve applied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



Example solution:

```
t = 0:1/fs:d-1/fs;    % timestamps for each sample point
% normalized logarithm of frequency of each tone (row)
% for each sample point (column), all rising linearly
% from 0 to 1, then wrap around back to 0
l = mod(((0:n-1)/n)' * ones(1, fs*d) + ones(n,1) * (t/(d*n)), 1);
f = fmin * (fmax/fmin) .^ l;    % freq. for each tone and sample
p = 2*pi * cumsum(f, 2) / fs;   % phase for each tone and sample
% make last column a multiple of 2*pi for phase continuity
p = diag((2*pi*floor(p(:,end)/(2*pi))) ./ p(:,end)) * p;
s = sin(p);     % sine value for each tone and sample
% mixing amplitudes from raised-cosine curve over frequency
a = 0.5 - 0.5 * cos(2*pi * l);
w = sum(s .* a)/n; % mix tones together, normalize to [-1, +1]

w = repmat(w, 1, 3);   % repeat waveform 3x
spectrogram(w, 2048, 1800, 2048, fs, 'yaxis');  ylim([0 fmax/1e3*1.1]) % plot
w = repmat(w, 1, 20);  % repeat waveform 20x
audiowrite('ladder.wav', w, fs, 'BitsPerSample', 16);  % 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?

MATLAB, Julia, NumPy: comparative cheat sheet

	MATLAB	Julia	NumPy
vector size (1,n)	[1 2 3]	[1 2 3]	np.array([1, 2, 3]).reshape(1, 3)
vector size (n,1)	[1; 2; 3]	[1 2 3]'	np.array([1, 2, 3]).reshape(3, 1)
vector size (n)	n/a	[1, 2, 3]	np.array([1, 2, 3])
j to n step k	j:k:n	j:k:n	np.arange(j, n+1, k)
matrix	[1 2; 3 4]	[1 2; 3 4]	np.array([[1, 2], [3, 4]])
0 matrix	zeros(2, 2)	zeros(2, 2)	np.zeros((2, 2))
1 matrix	ones(2, 2)	ones(2, 2)	np.ones((2, 2))
identity matrix	eye(2, 2)	I	np.eye(2)
diagonal matrix	diag([1 2 3])	Diagonal([1, 2, 3])	np.diag([1, 2, 3])
transpose	A.'	transpose(A)	A.T
complex conj. transpose	A'	A'	A.conj()
concat hor.	[[1 2] [1 2]]	[[1 2] [1 2]]	B = np.array([1, 2]) np.hstack((B, B))
matrix to vector	A(:)	A[:]	A.flatten()
flip left/right	fliplr(A)	reverse(A,dims=2)	np.fliplr(A)
broadcast a function	f=@(x) x.^2; f(x)	f(x)=x^2; f.(x)	def f(x): return x**2 f(x)
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, :]
element-wise multipl.	A .* B	A .* B	A * B
matrix multiplication	A * B	A * B	A @ B
...			

https://cheatsheets.quantecon.org/