Introduction to MATLAB

Markus Kuhn

Computer Laboratory

`http://www.cl.cam.ac.uk/Teaching/2003/DigSigProc/`

Easter 2004 – Part II

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
  ⇒ 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, 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 is MATLAB 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 suited for teaching OOP
  - limited GUI features
- ⟶ not a high-performance language (but fast matrix operators)
- ⟶ not freely available

Some of these limitations have been reduced in recent releases, e.g. release 13 replaced slow interpreter with a JIT compiler (JVM).

GNU Octave is a mostly compatible free reimplementation of a MATLAB subset: `http://www.octave.org/`

Availability and documentation

- ⟶ Installed on
  - Intel Lab PWF Windows
  - Intel Lab PWF Linux (`/usr/bin/matlab`)
  - PWF servers `linux{2,3}.pwf.cl.cam.ac.uk`
  - Computer Laboratory Windows and Linux PCs
- ⟶ Full documentation available online in HTML and PDF
  - Start `matlab` then type `helpdesk`
  - `http://www.mathworks.com/access/helpdesk/help/helpdesk.html`
- ⟶ Read "Getting Started" section of the MATLAB manual
- ⟶ Use the command `help` *`function-name`*

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)`. 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
```

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
```

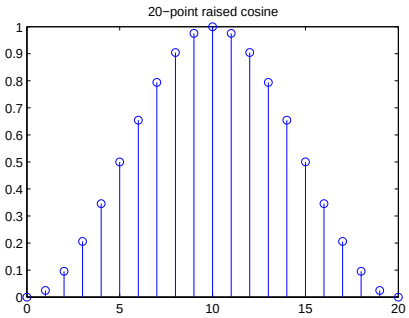
Inner and outer vector product:

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

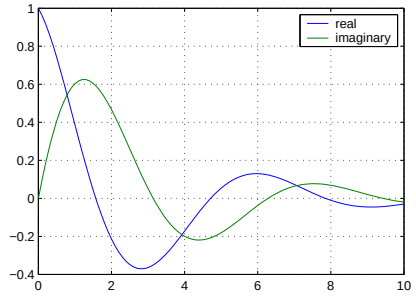
The imaginary unit vector $\sqrt{-1}$ is available as both `i` and `j`, and matrices can be complex.

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.

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

`wavread`, `wavwrite`, `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 $\mathrm{db}(x) = 10^{x/20}$, write into a file `db.m` the lines

```
function f = db(x)
% DB(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

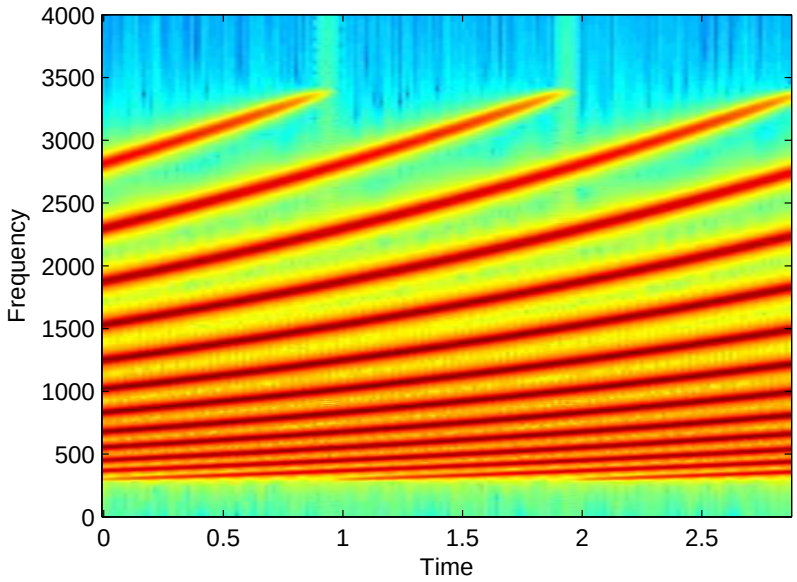
Generate an audio file with 12 sine tones of apparently continuously exponentially increasing frequency, which never leave the frequency range 300–3400 Hz. Do this by letting them wrap around the frequency interval and reduce their volume near the interval boundaries based on a raised-cosine curve applied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

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

Parameters:

```
fs   = 16000;  % sampling frequency [Hz]
d    = 1;      % time after which waveform repeats [s]
fmin = 300;    % lowest frequency
fmax = 3400;   % highest frequency
n    = 12;     % number of tones
```


Spectrogram of the first 3 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
specgram(w, 2048, fs, 2048, 1800);  ylim([0 4000])  % plot
w = repmat(w, 1, 20);  % repeat waveform 20x
wavwrite(w, fs, 16, 'ladder.wav');  % make audio file
```


Exercise 1 (a) Write down the function $g(t)$ that has the shape of a sine wave that increases linearly in frequency from 0 Hz at $t = 0$ s to 5 Hz at $t = 10$ s.

(b) Plot the graph of this function using MATLAB's `plot` command.

(c) Add to the same figure (this can be achieved using the `hold` command) in a different colour a graph of the same function sampled at 5 Hz, using the `stem` command.

(d) Plot the graph from (c) separately. Try to explain its symmetry (hint: sampling theorem, aliasing).

Exercise 2 Use MATLAB to write an audio waveform (8 kHz sampling frequency) that contains a sequence of nine tones with frequencies 659, 622, 659, 622, 659, 494, 587, 523, and 440 Hz. Then add to this waveform a copy of itself in which every other sample has been multiplied by -1. Play the waveform, write it to a WAV file, and use the `specgram` command to plot its spectrogram with correctly labelled time and frequency axis.