

## A Quick Introduction to Matlab

Tillman Weyde

## Create a Program

- Create a new **.m-file** with **File/New**  

```
function hello()
    'Hello World'
end
```
- Save** the file as **hello.m**
- Run** it from the editor. When asked, set the current Matlab directory to where the file is
- You should see: `ans = 'Hello World'`

## Vectors and Matrices

```
Row vectors octave:1> a = [ 1, 3, 2 ]
Column vectors octave:2> b = [ 3; 5; 1 ]
Matrices octave:3> A = [ 1,1,2;3,5,8;13,21,34 ]
A =
    1    1    2
    3    5    8
   13   21   34
```

## OCTAVE/MATLAB Programming

- Matlab** and **Octave**: programming languages for
  - **technical computing**
  - **prototyping**
  - **data analysis**, and other things
- Matlab** available in **labs** and for **student machines**  
<http://www.city.ac.uk/current-students/it-support/resources-and-facilities/matlab-nvivo-oxmetrics-spss#Matlab>
- Octave** is **open source** and can be obtained from  
<http://www.octave.org/>

## Data Types

- Variables** in Matlab are **not explicitly typed**, introduce them **as you use** them:  
`a = 1`
- Vectors** and **matrices** are the most **common data types** with excellent support.  
`b = [1,4,3,8,7,6,0,5,8,6];`
- The **value** of the **term** is **echoed** on the **terminal**. Ending a command with **semicolon** **suppresses** this behaviour.

## Matrix Arithmetic

- Matlab/Octave have **Matrix arithmetic built in**  
 let `a, b` be matrices and `x` a number
  - `a * b` matrix **multiplication**
  - `a / b` matrix **right division**
  - `a \ b` matrix **left division**
  - `a'` matrix **transposition** (`x'` is complex conjugate)
- Operators** can be applied **element-wise** by **prepending '.'**
  - `a ./ x` divide every element of matrix `a` by `x`
  - `a .+ x` add `x` to every element of `a`

## Getting Started

- Open **Matlab** (on lab machines search in Start menu)
- Type into the **command window**  
`1 + 1`
- You should see the following  
`ans =`  
`2`
- You can get **help** by clicking on the '?' icon

## Numbers

- All **numbers** are **double precision floating point**.
- Complex numbers** can be written like this:  
`3 + 4i` OR `3 + 4I` OR `3 + 4j` OR `3 + 4J`
- `abs()` gets the **absolute** value ( $\sqrt{a^2+b^2}$ ),  
`arg()` or `angle()` gets the **angle** (phase),  $\arctan(b/a)$   
`real()` gets the the **real** part,  
`imag()` gets the the **imaginary** part

## Ranges

- A **range** is defined by start and upper limit  
 – `1:5` creates 1,2,3,4,5
- Can have **step size**  
 – `1:2:5` creates 1,3,5
- Can be used as **vector**  
 – `[1:5]` creates [1,2,3,4,5]

## Defining a Function

- Write a **file** `some_name.m` and declare a **function** `some_name` in it.
  - A function is **declared** like this in the file
- ```
function val = some_name(var1, var2)
    .... % do something
end
```
- The **function** can be **called** using `some_name` **if** the file is in the **current directory** list of Matlab/Octave

## Reading a signal, more language features

- ```
[y,Fs] = audioread('filename')
```
- **reads** a signal and sampling Frequency
  - Functions can have **more than one return value**
  - create your own multi-valued functions
- ```
function [s,Fs]= readsignal(x)
    [s,Fs] = audioread('test.wav');
end
```
- **Older** versions of Matlab require at least one parameter for internal functions, therefore the `x`

## for-loop

- ```
for i = 1:m
    for j = 1:n
        H(i,j) = 1/(i+j);
    end
end
```
- **for** loop operates on **ranges**:  
`1:n = 1,2,3,...,n-1,n`  
`1:3:n = 1,3,6,...,k*3`, where  $0 \leq n-k*3 < 3$

## A signal

- Create a **signal** (i.e. a **vector**)
- Use a **range** to create a 'filled' **vector**  
`v = [1:10]`
- Use **zeros** to create a vector ( $1 \times n$  matrix) with 0s  
`z = zeros(1,10)`
- Fill the **vector** with a **sine wave**  
`s = sin(2*pi*250/8000*[1:2000]);`  
 creates 2000 samples, i.e. 0.25 sec, of a 250Hz sound at  $F_s = 8000\text{Hz}$

## Using Vectors

- Vector (and Matrix) **elements** are accessed giving the position in **round (!) brackets**.  
`[s,Fs] =readsignal(0);`  
`ws = s(1:10)`
- This gets the **first ten values** of the vector (as a new vector).

## while-loop

- ```
i=10
while i > 0
    i = i-1;
end
```
- **while** loops test for **logical** conditions (like if)

## Other output

- `;` at end of line **suppresses print output** (but it is **not necessary** like it is in Java or C)
- `plot(s)` **plots** the signal
- `sound(s,8000)` **plays** the signal at  $F_s = 8000\text{Hz}$
- `audiowrite('test.wav',s,8000)` **writes** the signal to **file test.wav** at  $F_s = 8000\text{Hz}$

## Control structures: conditionals

- ```
if rem(n,2) ~= 0
    M = odd_magic(n)
elseif rem(n,4) ~= 0
    M = single_even_magic(n)
else
    M = double_even_magic(n)
end
```
- **if** tests for logical value true or numeric value not 0
  - `~` is '**not**', like '**!**' in Java

## Complex numbers

- ```
S = fft([2,2,0,0])
```
- `S` is an array (vector) of complex numbers
  - `c = S(2)` gives a complex number  $2.0 - 2.0i$   
`real(c) = 2`  
`imag(c) = 2`  
`abs(c) = 2.8284` % comment:  $2 \sqrt{2}$   
`angle(c) = -0.7854` % comment:  $-\pi/4$