ملخصات مدار

د. ليندا الحمود

ماتلاب

زيد ابراهيم

Matlab Commands and functions:

- (*) clc: clears the command window
- (*) clf: clears the graphics window
- (*) clear: clears the work space (i.e. removes variables or some specific ones)
- (*) who: displays the names of your variables
- (*) whos: display the names, information about each variable size, number of elements, number of bytes, density wether the variable is complex.
- (*) format short: 4 decimals
- (*) format: default (same as short)
- (*) format long: 15 decimals
- (*) format short e: scientific notation with 4 decimals
- (*) format long e: scientific notation 15 decimals
- (*) format bank: fixed format for currency
- (*) format compact: suppress extra line feeds
- (*) format loose: puts the extra line feeds in
- (*) format rat: fraction

* Functions:

1) exp(): e^
2) log(): ln()
3) log10: $\log_{10}$
4) real(): real part
5) imag(): imaginary part
6) conj: complex conjugate
7) round: round to nearest integer
8) fix(): round toward zero
9) ceil: round to plus inf.
10) floor: round to negative inf.

(*) linspace(x, y, z) → z elements
from x, to y

(*) z : y : x
from z, y steps, to x

~~transpose~~ transpose operator ('):
x = [1 2 3] , x' = [1; 2; 3]
1×3 size, 3×1 size

---

a × b = 1st row × 1st column
matrices

(*) size(): _ _ rows, columns

(*) variable name (_ , _) R C , R = M([_ , _], :) all column

(*) x = w([2,3], [2,4]) row 2 and 3, column 2 and 4

(*) disp("____") or (var)

(*) disp(['__' blanks(n) '__']) فراغ

(*) disp(blanks(n)) سطر فاضي

(*) input('text')

text =

Solving system of ~~sys~~ linear eqns.

[*] rearrange : unknowns = known

$AX = B$

[*] A = [; ; --] — coeff. of equ. 1 ; equ2...

[*] B = [known1 ; known2 ; ---]

$X = A \backslash B$

[*] check: $C = A * X$ should equal B.

Func. polynomials :

[*] polyval(y, z) — name of the function ; z → x =

[*] roots(_) — function "polynomial" — "أصفار الدالة" ص.ع.

[*] Polyfit (y, z, 2) — must have the same size
- y → عوض من قيم y
- z → قيم الـ fund z
- 2 → أعلى رتبة 2nd order

[*] poly(_) — roots of a function
- (roots عوامل) → find the function ~~ф~~ that has _ roots

[*] polyder(_) "مشتقة"

Matlab :

* plot (x, y): plots y against x.
* grid on :  (use (show plot tools on matlab))
* xlabel ('~~'): writes ~~ on the x axis
* ylabel ('~~'): writes ~~ on the y axis (use {} to show the power on the plot)
* title ('A'): gives (A) as a title for the plot.
* We can use the plot function for two or more curves as follows:

plot (x, y, x, z) ⟶ will plot y vs. x and z vs. x

(1st curve, 2nd curve)

* legend ('y vs. x', 'z vs. x') : this function gives some notation for the user to know which is which.

(1st curve, 2nd curve)

* You can specify line style and color within the plot command

e.g.: plot(x, y, 'b-', x, z, 'r--')

means plot y vs. x with a blue solid line

means plot (z vs. x) with a red dashed line.

-	solid line (default)
--	dashed line
:	Dotted line
-.	Dash-Dot line
.	Dotted (more space)

r	red
g	green
b	blue (default)
w	white
k	black

* subplot (_, _, _)
  - rows
  - columns
  - what element

E.g. subplot(2, 2, 1), plot (x1, y1)

will give 4 plots

2 rows
2 columns

* plot3 (x, y, z)
* ~~title~~ zlabel : same as x and y label.
* take 100 elements or more.

end of lec. 4

M-files:

Why do we use them?

1. if u wanna change a var. value more easily (U don't have to write the command again)
2. if u wrote a long program contains lots of commands and u wanted to use it some other time
3. if there is a problem with such program U don't have to rewrite the whole thing; instead U can easily modify it using m-files

* edit ~~ ! let's u edit the M file that has a name of (~~) or makes a new file with (~~) name.
* M files should be named as the way u name a variable

Script file:

(*) statments that begin with '%' are comments

(*) help ______ (name of a script file) : will show the comment of the script file at the beginning of the file.

(*) Varibles in the workspace can be used by the script files

(*) Variable created by ~~the~~ script files can be used after the script file has been run.

[*] For a function returning one Variable
→ function dependent_variable = function_name(independent_var, ...)

Endfunction

(*) for a function returning more than one Value (let's say two)

function [y, P] = Function_Name(dep. Variable1, ...)
endfunction

* the name of the m-file must be the same as the name of the function.

* function can be used in the command window or in another script file.

(*) a function may have no input arguments and no output list:

function Function_Name
Endfunction

| date is a predefined Variable

[*] Important notes

Script	function
Var. are global : U can use the vars. defined in the script file after running it	Vars. are local: the Vars. defined in the function are used only inside it and can't be used inside the command

Decision making with matlab

less than	<	Not	~
less than or equal	<=	and	&
equal to	==	or	|
doesn't equal	~=		

[*] The general form of the if statement

```
if expression, ~~command~~ commands
else if expression
        commands
else
        commands
end
```

θ: الدوران
r = ?
x = ~~r cos θ~~ r cos θ
x = r sin θ
plot(x,y), axes equal

loops:

① for loop: Used if the number of passes is known ahead of time

② While loop: Used when the looping process must terminate when a specific condition is satisfied, and thus the number of passes is not known in advance

notes on for loops:

① S could be negative   m:s:n

② if s is omitted, s=1 by default

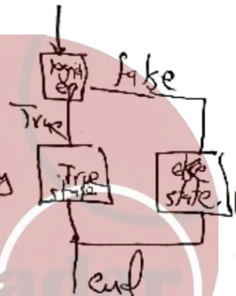
③ if s is +ve and m>n the loop will not be executed

④ if s is -ve, and m<n the loop will not be executed

⑤ if m=n, the loop will be executed only once

x = [4, -9, 25];
if x<0
    a
~~else~~ else
    b

although ... it display b for true



* Matlab will look at the vector as if it were a single thing
if x<0

Matlab: Solving systems of nonlinear equation

[*] Inline : a command that can be used for simple, one line function:

f = inline('x^2 - 5*x + 3')

تكتب معادلة

[*] two ways to write a function:

1. Inline for single functions
2. Matlab editor (scrips files)

[*] fzero: a function that can be used to solve a single variable nonlinear eqn. of the form $f(x)=0$

→ this function needs an inital guess (x0)

X = fzero(fun, x0) or X = fzero(@fun, x0)

معنى ال @ انه بيعرف فانكشن fun موجودة بملف الماتلاب.

[*] roots: a command used to find all the roots of a function:

F = [a b c d] roots(F)

coeffi. ← 3rd order poly

[*] fsolve: Used to solve sets of nonlinear algebric equations

* نعرف ال X قيم: X = [ ]'

fsolve(@fun, X)

1. Numerical Integration

1. quad('fun', a, b, tol) = fun تكامل

- fun: name of the function
- a: from
- b: to
- tol: optional, indicates the specific error tolerance

2. trapz(X, Y): computes the integral (area) of y with respect to x.

- X: points of f
- Y: contains the function values

[*] Note: we can define a fun. in the command window using the 'inline' command then we can use it in the 'quad' func. like: quad(f, 0, 5) without (' ')

[*] when the integral is specified by a set of points use 'trapz'. (although quad is more accurate)

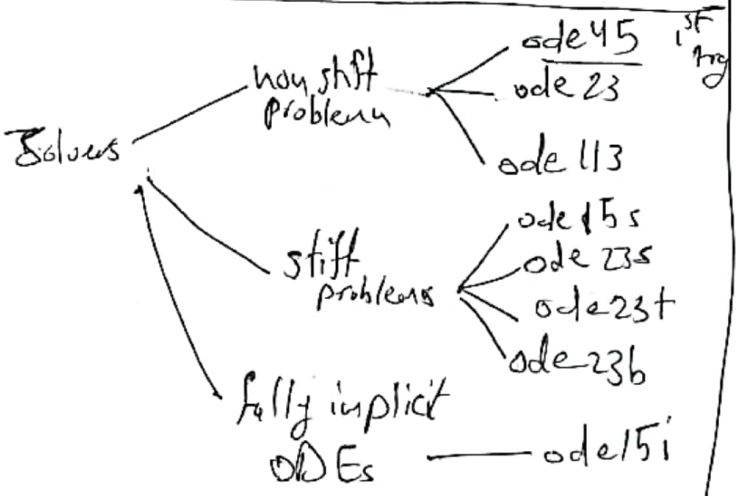
[*] كلما زدت عدد ال x زادت دقة ال trapz

Ordinary Differential Equations Solvers

(*) A stiff system is one involving rapidly changing components together with slowly changing ones.

ex: $\frac{dy}{dt} = \underbrace{-1000y}_{\text{slowly changing}} + 3000 - \underbrace{2000e^{-t}}_{\text{rapidly changing}}$

"stiff"



(*) Matlab solvers handle the following types:

- Explicit ODE: $y' = f(y,t)$
- Linearly implicit ODEs: $M(t,y)\,y' = f(t,y)$ (M: matrix)
- Fully implicit ODEs: $f(y',y,t) = 0$ (ode15i only)

(*) We can use inline to solve ode: `f = inline('...')`

`[t,y] = ode_(f, tspan, y0)`

(2) Solver syntax:

all of the solvers share the same syntax except ode15i:

`[t,y] = solver(odefun, tspan, y0, options)`

- $\frac{dy}{dt} = odefun(t,y)$
- odefun: column vectors
- (t,y): scalar
- tspan: t from to
- y0: Initial condition of the problem
- options: optional

(*) Important note: Matlab doesn't solve 2nd derivatives, so u must assume say $y_2 = y_1'$

(*) u can evaluate the approx. solution at any point in the interval of integration (tspan) using the function deval,

```
sol = ode_(...);
Xint = 1:5;
Sxint = deval(sol, Xint)   % 5 x 5
```

ode15i syntax:

`[t,y] = ode15i(odefun, tspan, y0, yp0)`

(*) if the condition of yp isn't given you should find it yrself:

```
t0 = 1;
y0 = sqrt(3/2);
yp0 = 0;
[y0,yp0] = decic(odefun,t0,y0,fixed_y0,yp0,fixed_yp0)
        = decic(@func,t0,y0,[],yp0,[])
```