FF505

Computational Science

MATLAB Section - Introduction 1
Matrix Algebra

Marco Chiarandini (marco@imada.sdu.dk)

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark

Outline

Outline

1. Getting Started
2. More on Matrix Calculations
3. Math Functions

MATLAB Desktop

- Command window
- Workspace
- Command history
- Current folder browser

- Variable editor
- MATLAB program editor

- Help
- Desktop menu

Command line programming

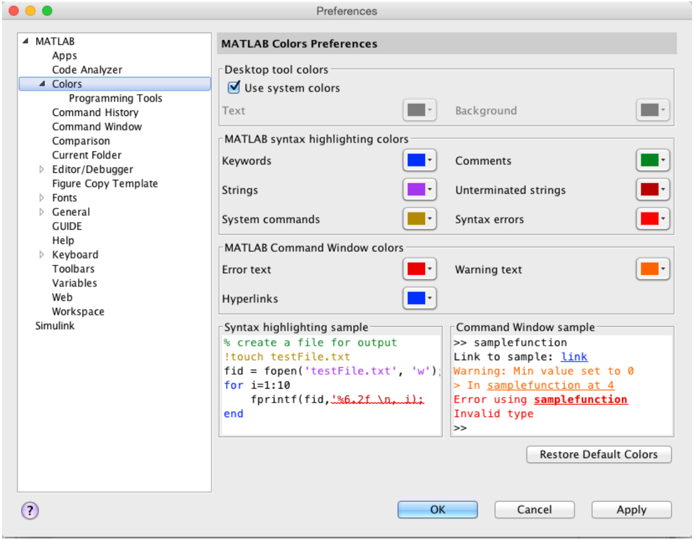
```
%%% elementary operations
5+6
3-2
5*8
1/2
2^6
1 == 2 % false
1 ~= 2 % true. note, not "!="
1 && 0
1 || 0
xor(1,0)
```

- Docking/Undocking, maximize by double click
- Current folder
- Search path (`File menu -> set path`)
- Documentation: Press ? → MATLAB → Getting Started

Customization

`MATLAB -> preferences`

Allows you personalize your MATLAB experience



Variable Assignment

The = sign in MATLAB represents the assignment or replacement operator. It has a different meaning than in mathematics.

Compare:

`x = x + 3` In math it implies 0=2, which is an invalid statement
In MATLAB it adds 2 to the current value of the variable

```
%% variable assignment
a = 3; % semicolon suppresses output
b = 'hi';
c = 3>=1;

% Displaying them:
a = pi
disp(sprintf('2 decimals: %0.2f', a))
disp(sprintf('6 decimals: %0.6f', a))
format long % 16 decimal digits
a
format short % 4 decimal digits +
      scientific notation
a
```

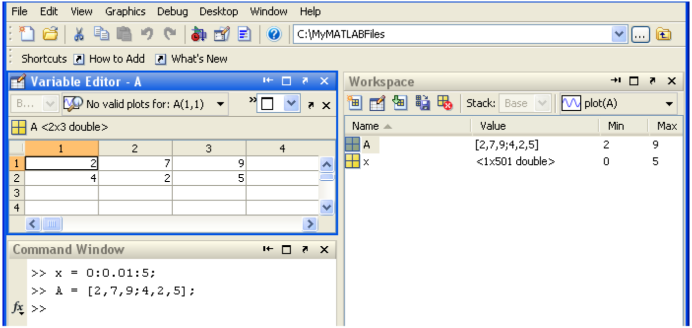
```
x + 2 = 20 % wrong statement
x = 5 + y % wrong if y unassigned
```

Variables are visible in the workspace

Names:

- [a-z] [A-Z] [0-9] _
- case sensitive
- max 63 chars

Variable Editor



Managing the Work Session

```
who % lists variables currently in memory
whos % lists current variables and sizes
clear v % clear w/ no argt clears all
edit filename % edit a script file
clc % clears theCommand window
... % ellipsis; continues a line
help rand % returns help of a function
quit % stops MATLAB
```

Predefined variables

```
pi
Inf % 5/0
NaN % 0/0
eps % accuracy of computations
i,j % immaginary unit i=j=sqrt(-1)
3+8i % a complex number (no *)
Complex(1,-2)
```

Working with Files

MATLAB handles three types of files:

- M-files `.m`: Function and program files
- MAT-files `.mat`: binary files with name and values of variables
- data file `.dat`: ASCII files

```
%% loading data
load q1y.dat
load q1x.dat
save hello v; % save variable v into file
      hello.mat
save hello.txt v -ascii; % save as ascii
% fopen, fprintf, fscanf also work
% ls %% cd, pwd & other unix commands
       work in matlab;
% to access shell, preface with "!"
```

Files are stored and searched in current directory and search path

Directories and paths

If we type `problem1`

1. seeks if it is a variable and displays its value
2. checks if it is one of its own programs and executes it
3. looks in the current directory for file `program1.m` and executes the file
4. looks in the search path for file `program1.m` and executes it

```
addpath dirname % adds the directory dirname to the search path
cd dirname % changes the current directory to dirname
dir % lists all files in the current directory
dir dirname % lists all files in dirname
path % displays the MATLAB search path
pathtool % starts the Set Path tool
pwd % displays the current directory
rmpath dirname % removes the directory dirname from the search path
what % lists MATLAB specific files in the current directory
what dirname % lists MATLAB specific files in dirname
which item % displays the path name of item
```

Getting Help

- `help funcname`: Displays in the Command window a description of the specified function `funcname`.
- `lookfor topic`: Looks for the string topic in the first comment line (the H1 line) of the HELP text of all M-files found on MATLABPATH (including private directories), and displays the H1 line for all files in which a match occurs.
  Try: `lookfor imaginary`
- `doc funcname`: Opens the Help Browser to the reference page for the specified function `funcname`, providing a description, additional remarks, and examples.

Scripts, M-files
Overview

Scripts are

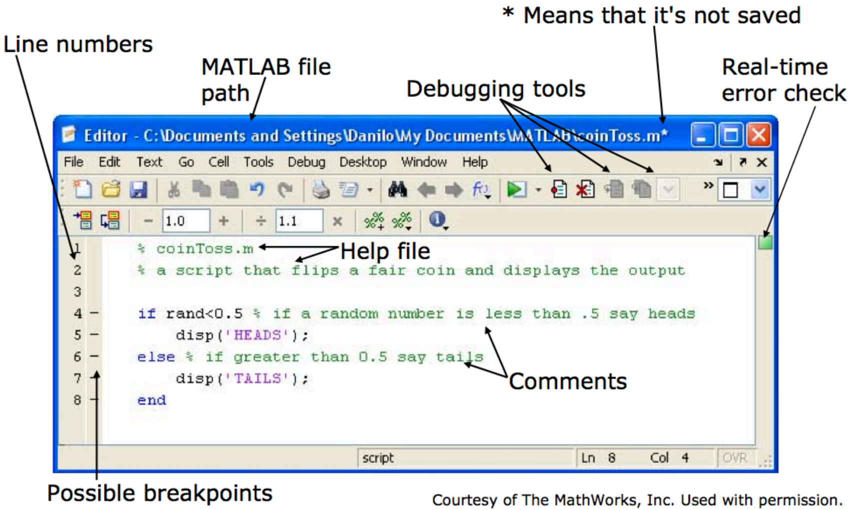
- collection of commands executed in sequence
- written in the MATLAB editor
- saved as MATLAB files (.m extension)

To create an MATLAB file from command-line

```
edit helloWorld.m
```

or from Menu on the top

Script: the Editor



Exercise: Scripts

- Make an initial script `Gravity` and save it.
- When run, the script should display the following text:

```
This is my first script! Yuhuu!
```

Hint: use `disp` to display strings. Strings are written between single quotes, like `'This is a string'`

1-D Arrays

Vectors: To create a row vector, separate the elements by commas. Use square brackets. For example,

```
>> p = [3,7,9]
p =
   3 7 9
```

You can create a column vector by using the transpose notation (').

```
>> p = [3,7,9]'
p =
   3
   7
   9
```

You can also create a column vector by separating the elements by semicolons. For example,

```
>> g = [3;7;9]
g =
   3
   7
   9
```

Appending vectors:

```
r = [2,4,20];
w = [9,-6,3];
u = [r,w]
u =
    2 4 20 9 -6 3
```

```
r = [2,4,20];
w = [9,-6,3];
u = [r;w]
u =
    2 4 20
    9 -6 3
```

2-D Arrays

Matrices: spaces or commas separate elements in different columns, whereas semicolons separate elements in different rows.

```
>> A = [2,4,10;16,3,7]
A =
     2 4 10
    16 3 7

>>c = [a b]
c =
   1 3 5 7 9 11

>>D = [a ; b]
D =
   1 3 5
   7 9 11
```

Arrays

Arrays are the basic data structures of MATLAB (weakly typed language - no need to declare the type)
Types of arrays:
numeric • character • logical • cell • structure • function handle

```
%% vectors and matrices
A = [1 2; 3 4; 5 6]

v = [1 2 3]
v = [1; 2; 3]
v = [1:0.1:2] % from 1 to 2, with stepsize of 0.1. Useful for plot axes
v = 1:6 % from 1 to 6, assumes stepsize of 1

C = 2*ones(2,3) % same as C = [2 2 2; 2 2 2]
w = ones(1,3) % 1x3 vector of ones
w = zeros(1,3)
w = rand(1,3) % drawn from a uniform distribution
w = randn(1,3) % drawn from a normal distribution (mean=0, var=1)
w = -6 + sqrt(10)*(randn(1,10000)) % (mean = 1, var = 2)
hist(w) % histogram
e = []; % empty vector
I = eye(4) % 4x4 identity matrix
A = linspace(5,8,31) % equivalent to 5:0.1:8
```

Indexing

```
%% indexing
A(3,2) % indexing is (row,col)
A(2,:) % get the 2nd row. %% ":" means every elt along that dimension
A(:,2) % get the 2nd col
A(1,end) % 1st row, last elt. Indexing starts from 1.
A(end,:) % last row

A([1 3],:) = [] % deletes 1st and 3rd rows
A(:,2) = [10 11 12]' % change second column
A = [A, [100; 101; 102]]; % append column vec
% A = [ones(size(A,1),1), A]; % e.g bias term in linear regression
A(:) % Select all elements as a column vector.
```

```
%% dimensions
sz = size(A)
size(A,1) % number of rows
size(A,2) % number of cols
length(v) % size of longest dimension
```

Plots

```
%% plotting
t = [0:0.01:0.98];
y1 = sin(2*pi*4*t);
plot(t,y1);
y2 = cos(2*pi*4*t);
hold on; % "hold off" to turn off
plot(t,y2,'r--');
xlabel('time');
ylabel('value');
legend('sin','cos');
title('my plot');
close; % or, "close all" to close all figs
```

```
figure(2), clf; % can specify the figure number
subplot(1,2,1); % Divide plot into 1x2 grid, access 1st element
plot(t,y1);
subplot(1,2,2); % Divide plot into 1x2 grid, access 2nd element
plot(t,y2);
axis([0.5 1 -1 1]); % change axis scale
```

`help graph2D`

Rapid Code Iteration

- Rapid code iterations using cells in the editor
- cells are small sections of code performing specific tasks
- they are separated by double %
- they can be executed independently, eg, CTRL+Enter and their parameters adjusted
- navigate by CTRL+SHIFT+Enter or by jumping
- publish in HTML or PDF or Latex (menu publish on the top).

Outline

Order of Operations

1. parenthesis, from innermost
2. exponentiation, from left to right
3. multiplication and division with equal precedence, from left to right
4. addition and subtraction with equal precedence, from left to right

```
>>4^2-12-8/4*2
ans =
    0
>>4^2-12-8/(4*2)
ans =
    3
>> 3*4^2 + 5
ans =
    53
>>(3*4)^2 + 5
ans =
    149
```

```
>>27^(1/3) + 32^(0.2)
ans =
    5
>>27^(1/3) + 32^0.2
ans =
    5
>>27^1/3 + 32^0.2
ans =
    11
```

Creating Matrices

```
eye(4) % identity matrix
zeros(4) % matrix of zero elements
ones(4) % matrix of one elements
```

```
A=rand(8)
triu(A) % upper triangular matrix
tril(A)
diag(A) % diagonal
```

```
>> [ eye(2), ones(2,3); zeros(2),
     [1:3;3:-1:1] ]

ans =

    1 0 1 1 1
    0 1 1 1 1
    0 0 1 2 3
    0 0 3 2 1
```

Can you create this matrix in one line of code?

```
-5   0   0   0   0   0   0   1   1   1   1
 0  -4   0   0   0   0   0   0   1   1   1
 0   0  -3   0   0   0   0   0   0   1   1
 0   0   0  -2   0   0   0   0   0   0   1
 0   0   0   0  -1   0   0   0   0   0   0
 0   0   0   0   0   0   0   0   0   0   0
 0   0   0   0   0   0   1   0   0   0   0
 1   0   0   0   0   0   0   2   0   0   0
 1   1   0   0   0   0   0   0   3   0   0
 1   1   1   0   0   0   0   0   0   4   0
 1   1   1   1   0   0   0   0   0   0   5
```

Matrix-Matrix Multiplication

In the product of two matrices A * B,
the number of columns in A must equal the number of rows in B.

The product AB has the same number of rows as A and the same number of columns as B. For example

```
>> A=randi(10,3,2) % returns a 3-by-2 matrix containing pseudorandom integer values
     drawn from the discrete uniform distribution on 1:10
A =
    6 10
   10 4
    5 8

>> C=randi(10,2,3)*100
C =
       1000 900 400
        200 700 200
>> A*C % matrix multiplication
ans =
       8000 12400 4400
      10800 11800 4800
       6600 10100 3600
```

Remark:
Matrix multiplication does not have the commutative property; that is, in general, $AB \neq BA$. Make a simple example to demonstrate this fact.

Matrix Operations

```
%% matrix operations
A * C % matrix multiplication
B = [5 6; 7 8; 9 10] * 100 % same dims as A
A .* B % element-wise multiplcation
% A .* C or A * B gives error - wrong dimensions
A .^ 2
1./B
log(B) % functions like this operate element-wise on vecs or matrices
exp(B) % overflow
abs(B)
v = [-3:3] % = [-3 -2 -1 0 1 2 3]
-v % -1*v

v + ones(1,length(v))
% v + 1 % same

A' % (conjuate) transpose
```

Multidimensional Arrays

Consist of two-dimensional matrices layered to produce a third dimension. Each layer is called a page.

```
cat(2,A,B) % is the same as [A,B].
cat(1,A,B) % is the same as [A;B].
```

```
>> A = magic(3); B = pascal(3);
>> C = cat(4,A,B) %concatenate matrices along DIM

C(:,:,1,1) =


    8 1 6
    3 5 7
    4 9 2

C(:,:,1,2) =
    1 1 1
    1 2 3
    1 3 6
```

Array Operations

- **Addition/Subtraction**: trivial
- **Multiplication**:
  - of an array by a scalar is easily defined and easily carried out.
  - of two arrays is not so straightforward:
    MATLAB uses two definitions of multiplication:
    - array multiplication (also called element-by-element multiplication)
    - matrix multiplication
- **Division and exponentiation** MATLAB has two forms on arrays.
  - element-by-element operations
  - matrix operations

Element-by-Element Operations

Symbol	Operation	Form	Examples
+	Scalar-array addition	A + b	[6,3]+2=[8,5]
-	Scalar-array subtraction	A - b	[8,3]-5=[3,-2]
+	Array addition	A + B	[6,5]+[4,8]=[10,13]
-	Array subtraction	A - B	[6,5]-[4,8]=[2,-3]
.*	Array multiplication	A.*B	[3,5].*[4,8]=[12,40]
./	Array right division	A./B	[2,5]./[4,8]=[2/4,5/8]
.\	Array left division	A.\B	[2,5].\[4,8]=[2\4,5\8]
.^	Array exponentiation	A.^B	[3,5].^2=[3^2,5^2]
			2.^[3,5]=[2^3,2^5]
			[3,5].^[2,4]=[3^2,5^4]

Backslash or Matrix Left Division

`A\B` is roughly like `INV(A)*B` except that it is computed in a different way:
`X = A\B` is the solution to the equation `A*X = B` computed by Gaussian elimination.

Slash or right matrix division:
`A/B` is the matrix division of `B` into `A`, which is roughly the same as `A*INV(B)`, except it is computed in a different way. More precisely, `A/B = (B'\A')'`.

Dot and Cross Products

`dot(A,B)` scalar product: computes the projection of a vector on the other.
eg. `dot(Fr,r)` computes component of force $\mathbf{F}$ along direction $\mathbf{r}$
Inner product, generalization of dot product

```
v=1:10
u=11:20
u*v' % inner or scalar product
ui=u+i
ui'
v*ui' % inner product of C^n
norm(v,2)
sqrt(v*v')
```

`cross(A,B)` cross product: eg: moment $\mathbf{M} = \mathbf{r} \times \mathbf{F}$

Exercise: Projectile trajectory

$\boldsymbol{p}$ position vector

$$\boldsymbol{p}_t = \boldsymbol{p}_0 + \boldsymbol{u}_t s_m t + \frac{\boldsymbol{g}t^2}{2}$$

s_m muzzle velocity (speed at which the projectile left the weapon)
u_t is the direction the weapon was fired
$g = -9.81\mathrm{ms}^{-1}$

Predict the landing spot

$$t_i = \frac{-u_i s_m \pm \sqrt{u_y^2 s_m^2 - 2g_y(p_{y0} - p_{yt})}}{g_y} \qquad \boldsymbol{p}_E = \begin{bmatrix} p_{x0} + u_x s_m t_i \\ p_{y0} \\ p_{z0} + u_z s_m t_i \end{bmatrix}$$

Plot the trajectory in 2D.

Exercise: Projectile trajectory

Given a firing point $\boldsymbol{S}$ and s_m and a target point $\boldsymbol{E}$, we want to know the firing direction $\boldsymbol{u}$, $|\boldsymbol{u}| = 1$.

$$
\begin{aligned}
E_x &= S_x + u_x s_m t_i + \frac{1}{2} g_x t_i^2 \\
E_y &= S_y + u_y s_m t_i + \frac{1}{2} g_y t_i^2 \\
E_z &= S_z + u_z s_m t_i + \frac{1}{2} g_z t_i^2 \\
1 &= u_x^2 + u_y^2 + u_z^2
\end{aligned}
$$

four eq. in four unknowns, leads to:

$$
|\boldsymbol{g}|^2 t_i^4 - 4(\boldsymbol{g} \cdot \boldsymbol{\Delta} + s_m^2) t_i^2 + 4|\boldsymbol{\Delta}|^2 = 0, \qquad \boldsymbol{\Delta} = \boldsymbol{E} - \boldsymbol{S}
$$

solve in t, and interpret the solution.

Useful Functions

```
% max (or min)
a = [1 15 2 0.5]
val = max(a)
[val,ind] = max(a)

% find
find(a < 3)
A = magic(3) %N-by-N matrix
      constructed from the integers 1
      through N^2 with equal row, column,
       and diagonal sums.
[r,c] = find(A>=7)

% sum, prod
sum(a)
prod(a)
floor(a) % or ceil(a)
max(rand(3),rand(3))
max(A,[],1)
min(A,[],2)
A = magic(9)
sum(A,1)
sum(A,2)
```

```
% pseudo-inverse
pinv(A) % inv(A'*A)*A'

% check empty e=[]
isempty(e)
numel(A)
size(A)
prod(size(A))
```

```
sort(4:-1:1)
sort(A) % sorts the columns
```

Useful Functions

Working with polynomials:

$$f(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \ldots + a_{n-1} x^2 + a_n x + a_{n+1}$$

is represented in MATLAB by the vector

$$[a_1, a_2, a_3, \ldots, a_{n-1}, a_n, a_{n+1}]$$

```
help polyfun
r=roots([1,-7,40,-34]) % x^3-7x^2+40x-34
poly(r) % returns the polynomial whose roots are r
roots(poly(1:20))
poly(A) % coefficients of the characteristic polynomial, det(lambda*EYE(SIZE(A)) - A)
```

Reshaping

```
%% reshape and replication
A = magic(3) % magic square
A = [A [0;1;2]]
reshape(A,[4 3]) % columnwise
reshape(A,[2 6])
v = [100;0;0]
A+v
A + repmat(v,[1 4])
```

Outline

Common Mathematical Functions

Exponential	
`exp(x)`	Exponential; e^x.
`sqrt(x)`	Square root; $\sqrt{x}$.
Logarithmic	
`log(x)`	Natural logarithm; $\ln x$.
`log10(x)`	Common (base-10) logarithm; $\log x = \log_{10} x$.
Complex	
`abs(x)`	Absolute value; x.
`angle(x)`	Angle of a complex number x.
`conj(x)`	Complex conjugate.
`imag(x)`	Imaginary part of a complex number x.
`real(x)`	Real part of a complex number x.
Numeric	
`ceil(x)`	Round to the nearest integer toward ∞.
`fix(x)`	Round to the nearest integer toward zero.
`floor(x)`	Round to the nearest integer toward $-\infty$.
`round(x)`	Round toward the nearest integer.
`sign(x)`	Signum function: $+1$ if $x > 0$; 0 if $x = 0$; -1 if $x < 0$.

Common Mathematical Functions

Trigonometric*	
`cos(x)`	Cosine; $\cos x$.
`cot(x)`	Cotangent; $\cot x$.
`csc(x)`	Cosecant; $\csc x$.
`sec(x)`	Secant; $\sec x$.
`sin(x)`	Sine; $\sin x$.
`tan(x)`	Tangent; $\tan x$.
Inverse trigonometric†	
`acos(x)`	Inverse cosine; $\arccos x = \cos^{-1} x$.
`acot(x)`	Inverse cotangent; $\text{arccot}\, x = \cot^{-1} x$.
`acsc(x)`	Inverse cosecant; $\text{arccsc}\, x = \csc^{-1} x$.
`asec(x)`	Inverse secant; $\text{arcsec}\, x = \sec^{-1} x$.
`asin(x)`	Inverse sine; $\arcsin x = \sin^{-1} x$.
`atan(x)`	Inverse tangent; $\arctan x = \tan^{-1} x$.
`atan2(y,x)`	Four-quadrant inverse tangent.

*These functions accept x in radians.
†These functions return a value in radians.

Common Mathematical Functions

Hyperbolic	
`cosh(x)`	Hyperbolic cosine; $\cosh x = (e^x + e^{-x})/2$.
`coth(x)`	Hyperbolic cotangent; $\cosh x/\sinh x$.
`csch(x)`	Hyperbolic cosecant; $1/\sinh x$.
`sech(x)`	Hyperbolic secant; $1/\cosh x$.
`sinh(x)`	Hyperbolic sine; $\sinh x = (e^x - e^{-x})/2$.
`tanh(x)`	Hyperbolic tangent; $\sinh x/\cosh x$.
Inverse hyperbolic	
`acosh(x)`	Inverse hyperbolic cosine
`acoth(x)`	Inverse hyperbolic cotangent
`acsch(x)`	Inverse hyperbolic cosecant
`asech(x)`	Inverse hyperbolic secant
`asinh(x)`	Inverse hyperbolic sine
`atanh(x)`	Inverse hyperbolic tangent