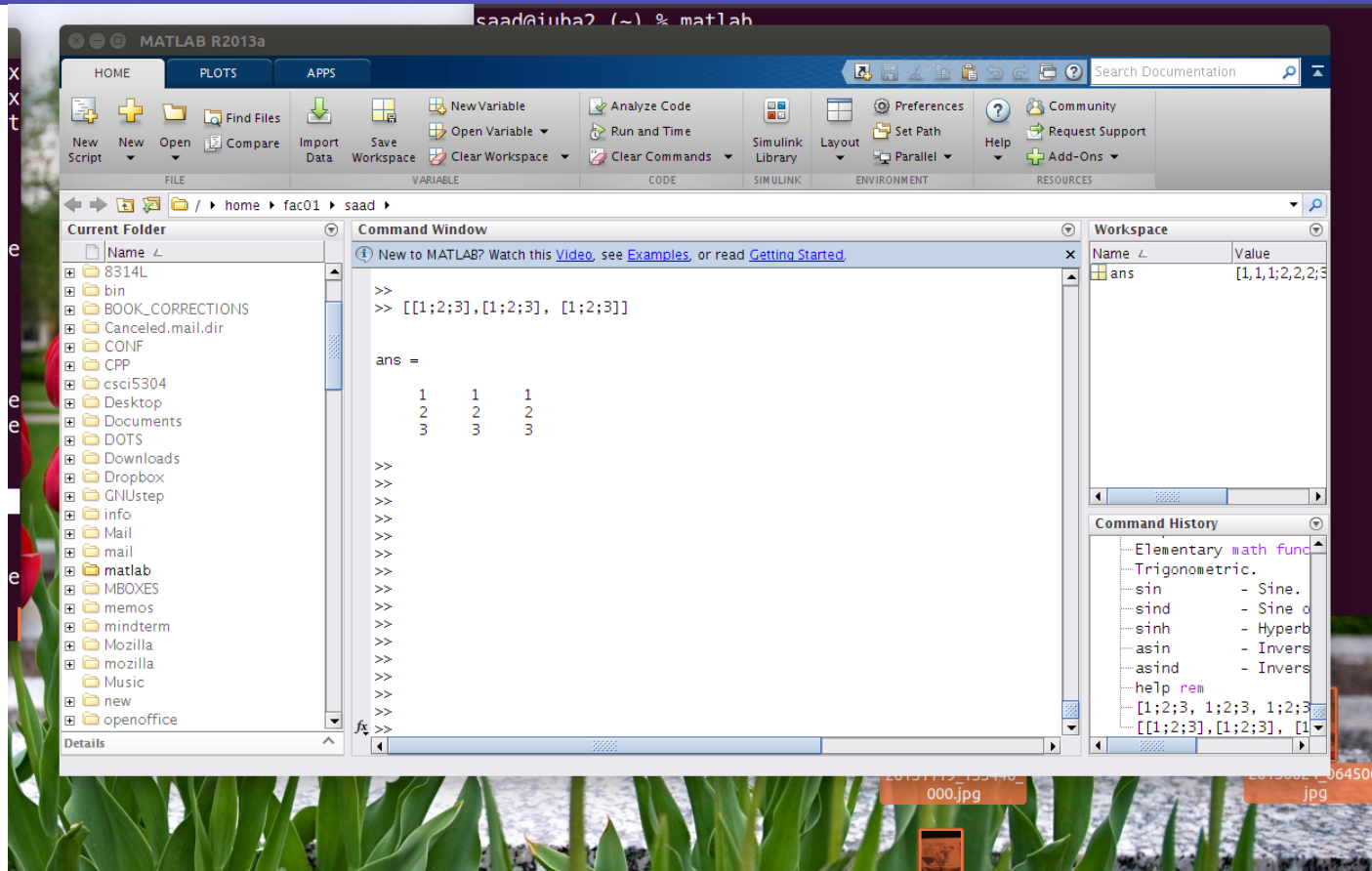


# A QUICK INTRODUCTION TO MATLAB

---

- **Very brief intro to matlab –**
- **Basic operations and a few illustrations**
- **This set is independent from rest of the class notes.**
- **Matlab will be covered in recitations and occasionally in class**

# Intro to matlab – getting started



**To start** type 'matlab' under a unix terminal (or click icon under windows). You will get a matlab GUI with a command window that has the prompt: >>.

➤ I prefer to use matlab without the GUI [especially for the demos given in class]. In linux or mac OS this is done by typing into a terminal the command

```
% matlab -nodesktop
```

instead of

```
% matlab
```

➤ To exit matlab use `exit` or `quit`

```
>> quit
```

## Getting Help

➤ Most of the help for matlab is online. In the GUI you can click on the '?' icon.

➤ Often it is faster to get help by typing into the matlab window

```
>> help topic
```

➤ Examples

```
>> help |
```

or

```
>> help rref
```

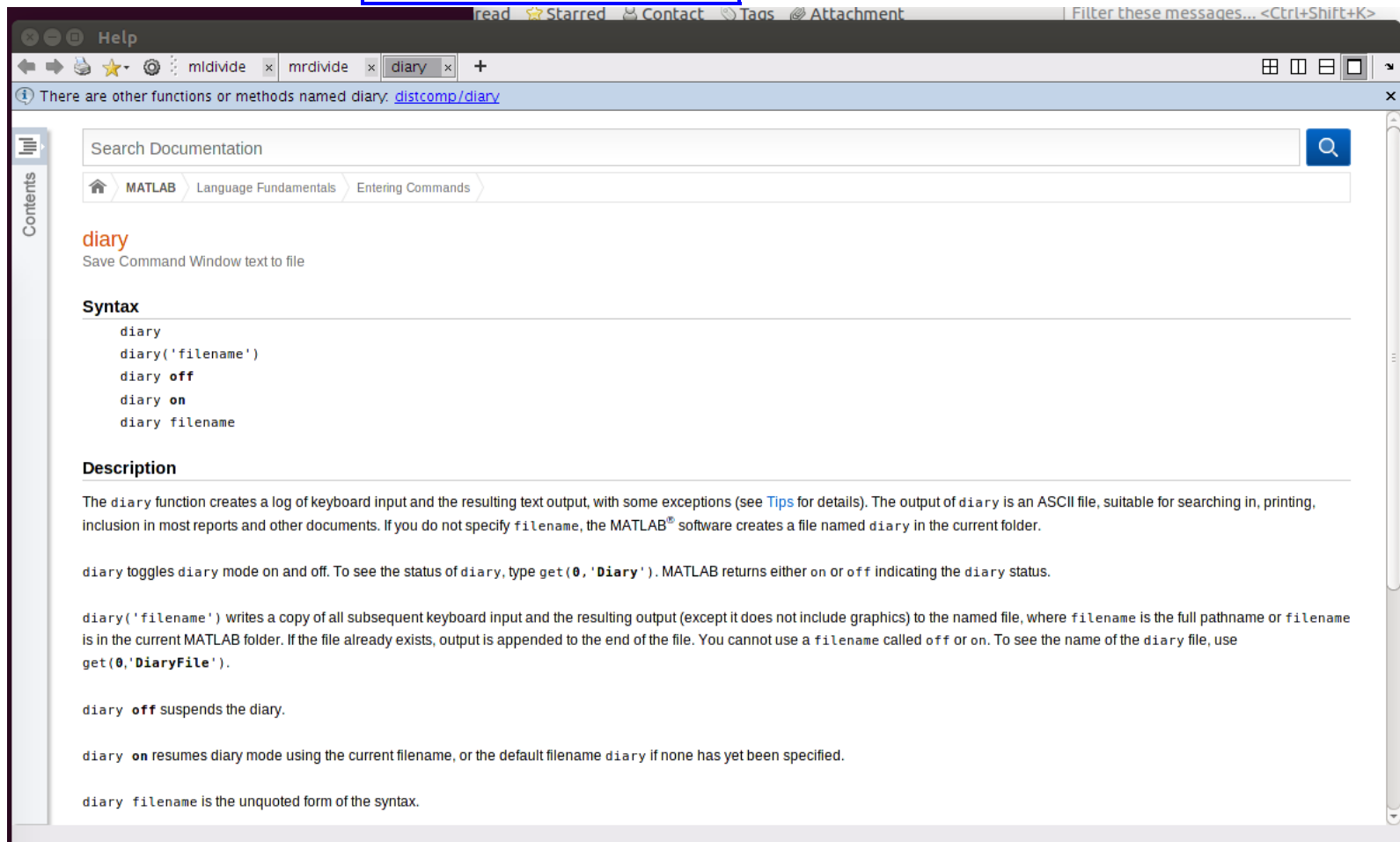
or

```
>> help punct
```

➤ Alternatively you can get the same info in a pop-out window by typing:

```
>> doc topic
```

➤ For example: `>> doc diary` gave this:



➤ '>>> help' or '>>> doc' by itself will list the help topics

➤ Same thing as clicking the '?' icon in the GUI.

## *Example:*

```
>> help mod
mod      modulus (signed remainder after division).
... followed by a few lines of explanation...
...
... then: shows related function (s):

See also REM.
```

```
>> a = 25; b = 3;
>> mod(a,b)
ans =
     1
>> mod1(a,5)
ans =
     0
>> mod0(25.2,2)

ans =
    1.2000
```

## *Basic Operations in Matlab*

➤ The following is on the basics of matlab. It starts with some basic operations and the help command.

➤ A useful command I used to generate some of these examples is

```
>>diary filename.
```

➤ This is equivalent to a **typescript**. Everything displayed on screen is saved in a file. [useful for homeworks]

➤ In what follows: Everything that starts with >> is what I typed into the matlab prompt.

## *Simple operations*

```
>> 4+6+3          |This is what I typed in
ans =             |These lines are matlab's
    13            |answer
>> 4*20+ 3*57 + exp(-0.1) |This is what I typed in
ans =             |These lines are matlab's
    251.9048      |answer
```

Note: ending versus not ending command with semi-colon.

```
>> a + 2          <----- do command + display result
ans =             <----- results of operation shown
    25
>> a+2;           <----- do command - do not display result
>>                <----- result not displayed
```



### *Squaring and powers:*

```
>> a = 12;  
>> a^2  
  
ans =  
    144  
  
>> a^4  
  
ans =  
    20736
```

### *Right/Left divide (/ and \)*

```
>> a = 12; b = 3;  
>> a/b  
  
ans =  
     4  
  
>> a\b  
  
ans =  
    0.2500  
  
>> b/a  
  
ans =  
    0.2500
```

➤ Important because these have their equivalent versions for matrices

## *more, disp, format*

```
>> more on
```

- `more on` allows you to scroll page by page
- `disp(x)` simply displays `x` without fillers
- `format` selects format for displaying results :

Options: `format short, long, rat, ...`

```
>> format short  
>> pi
```

```
ans =  
    3.1416
```

```
>> format long
>> pi

ans =
    3.141592653589793
```

```
>> format rat
>> pi
```

```
ans =
    355/113
```

➤ Also useful: `format compact` [avoids empty line feeds.. useful for homeworks]

- The command '`>> who`' lists the variable currently stored

```
>> who
```

```
your variables are:
```

```
a          ans          b
```

```
>>
```

- See also: '`>> whos`' which has more detail

- Earlier we invoked `exp` which is the exponential function.
- Get info by typing

```
>> help exp
```

```
exp      exponential.          | answer:
```

```
exp(x) is the exponential of the elements of x,  
e to the x. for complex z=x+i*y, exp(z) = ....  
+ a few more lines of explanation ending with
```

```
see also log, log10, expm, expint.  
overloaded methods  
help sym/exp.m
```

 Explore the other elementary functions:

```
>> help elfun
```

will list all the elementary functions used by matlab - A long list that starts like this :

```
elementary math functions.  
trigonometric.  
sin          - sine.  
sinh         - hyperbolic sine.  
asin        - inverse sine.  
asinh       - inverse hyperbolic sine.  
cos         - cosine.  
.  
.
```

## *Complex Numbers*

```
>> c = 1 - 2i  
c =  
1.0000 - 2.0000i  
>> conj(c)
```

```
ans =  
1.0000 + 2.0000i  
>> c*conj(c)
```

```
ans =  
5  
>> abs(c)
```

```
ans =  
2.2361
```

➤ Note:  $\text{abs}(c)$  is the modulus of  $c$

## Matrices

- To define a matrix enter entries row by row, separated by a “;”

```
>> A = [1 1 1 ; 2 2 2 ; 3 3 3 ]
```

```
A =
```

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

- Could use commas for separating columns (not required):

```
>> A = [ 1, 1, 1 ; 2, 2, 2 ; 3, 3, 3] ;
```



➤ So: ',' separates columns and ';' separates rows. The above matrix can also be defined as

```
>> A = [[1;2;3], [1;2;3], [1;2;3]]
```

➤ Can use matrices as blocks [very convenient!]

```
>> B = [A, A]
```

B =

1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3

 Show the result of the command: `>> C = [ A, -A; A*A, 2*A]`

➤ Two important special matrix functions

`eye(n)`

and

`zero(n)`

>> A = eye(5) | Identity matrix of size 5

A =

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

➤ It is enough to say `eye(5)` in this example but ...

➤ 'eye' is defined for rectangular matrices too

```
>> A = eye(6,3)
```

```
A =
```

```
    1     0     0
    0     1     0
    0     0     1
    0     0     0
    0     0     0
    0     0     0
```

➤ zeros(m) or zeros(m,n) is defined similarly:

```
>> A = zeros(3,4)
```

```
A =
```

```
    0     0     0     0
    0     0     0     0
    0     0     0     0
```

## *Defining a vector through loop constructs*

```
>> start=0; inc=2; last=12;
```

```
>> start:inc:last
```

```
ans =  
    0     2     4     6     8    10    12
```

```
>> 0:2:12
```

```
ans =  
    0     2     4     6     8    10    12
```

➤ Can also use real numbers

```
start = 0.0; inc = 0.15; last = 1.0;
```

```
>> start:inc:last
```

```
ans =  
    0  0.1500  0.3000  0.4500  0.6000  0.7500  0.9000
```

```
>> x = 0:0.15:1
```

```
x =  
    0    0.1500    0.3000    0.4500    0.6000    0.7500    0.9000
```

➤ Quite convenient for doing simple plots (see later)

➤ Can use loop constructs in matrices as well:

```
>> A = [1:4; 4:7] | 1st row = 1:4 = 1 2 3 4  
                | 2nd row = 4:7 = 4 5 6 7
```

```
A =  
    1    2    3    4  
    4    5    6    7
```

```
>> A = [0.0:0.1:0.5; 2.1:0.2:3.1] | Must have same  
                                   | number of entries  
                                   | in the 2 rows
```

```
A =  
          0    0.1000    0.2000    0.3000    0.4000    0.5000  
    2.1000    2.3000    2.5000    2.7000    2.9000    3.1000
```

## *The function 'size'*

Everything in matlab is considered a matrix. `size(x)` gives the dimensions of the object `x`

```
>> x = x = 0.0:0.1:0.8;    |0.0 0.1 ... 0.8 (9 entries)
>> size(x)
```

```
ans =
     1     9          <----- 1 row, 9 columns
```

```
>> A = [1:4; 4:7];
>> size(A)
```

```
ans =
     2     4          <----- 2 rows 4 columns
```

```
>>
>> size(pi)          | number pi = a scalar
```

```
ans =
     1     1          <----- 1 row 1 column
```

## Vector operations

```
>> x+y;           | adding 2 vectors of same shape
>> 0.15*x -.0*y; | linear comb. of x and y
>> y = exp(-x)   | point-wise exponential of -x
```

```
y =
 1.0000  0.8607  0.7408  0.6376  0.5488  0.4724  0.4066
```

➤ cannot square a vector:

```
>>
>> [1 2 3]^2
Error using ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

## *Pointwise (array) product:*

```
>> a = [2, 3 4] ; b = [ 0 5 6] ;  
>> c = a .* b
```

```
c =  
    0    15    24
```

Let us go back to  $z = x^2$ . To square the components of  $x$ , do:

```
>> y = x .^ 2
```

```
y =  
    0    0.0225    0.0900    0.2025    0.3600    0.5625    0.8100
```

or

```
>> y = x .* x
```

```
y =  
    0    0.0225    0.0900    0.2025    0.3600    0.5625    0.8100
```



## *Column/row access; submatrices*

[Try these!]

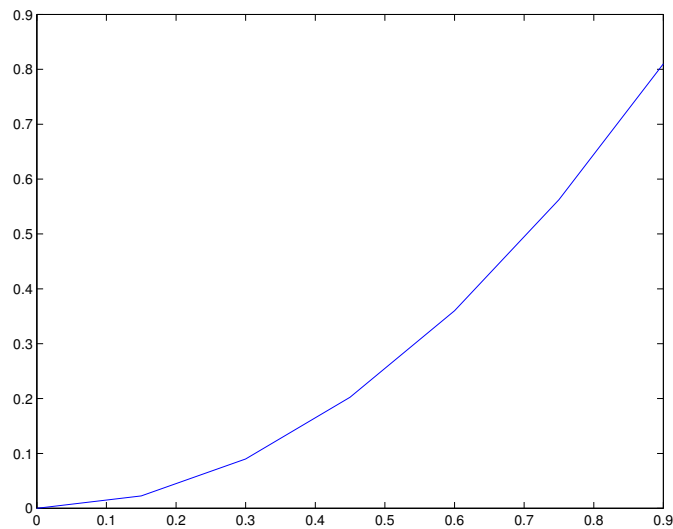
```
>> A = randn(5,10);      | generate 5x10 random matrix
>> B = A(2:5,5:10);     | subarray of rows 2 to 5
                        | and columns 5 to 10
>> B = A(1:2:5,2:2:10); | extract odd rows and even
                        | columns of A
>> r = A(1,:)           | 1st row of A
>> c = A(:,3:5)         | column 3 to 5 of A
>> A(:,3) = A(:,3)+0.5*A(:,1)
                        | Add 1/2 of col. 1 to col. 3
```

➤ Very useful:

```
r = r(:)                | forces r to be a column
                        | vector
```

## *Simple plotting*

- Matlab provides powerful graphics capabilities – 2D plots, 3D surfaces.
- The simplest command: `>> plot(x,y)` causes matlab to pop-out a window which has the following plot



 Try the following commands and explain what they do

```
x = [0:0.01:2*pi] ;  
y = x .* cos (x) ;  
plot(x,y);  
hold on  
z = 1 ./ ( 1/6 + y.^2);  
plot(x,z,'r--');  
plot([0, 2*pi],[0 0]);  
plot([0, 0],[0, 7]);  
axis([-1 7 -4 8])
```

## *Basic operators*

- Standard arithmetic operators:

$+$  ,  $-$  ,  $*$  ,  $/$

- Unary operations (for example  $-A$ ).
- Back-slash operator:

$$x = A \backslash b$$

where  $A$  is a matrix and  $b$  a vector (or matrix) then  $x = A^{-1}b$ .  
[to be seen later in the class.]

➤ Relational operators.

- Equal	==
- Not equal	~=
- Less than	<
- Greater than	>
- Less than or equal	<=
- Greater than or equal	>=

**Example:**

```
>> a = 1; b = 0; c = 2;  
>> a+b+c == c+a+b
```

```
ans =  
1
```

➤ Not to confuse with '=' :

```
>> a+b+c = c+a+b  
??? Error: Assignment statements cannot produce a result.
```

➤ Comparisons can be done on vectors and matrices:

```
>> a = 1:2:20
```

```
a =  
    1     3     5     7     9    11    13    15    17    19
```

```
>> b = 2:2:21
```

```
b =  
    2     4     6     8    10    12    14    16    18    20
```

```
>> a == b
```

```
ans =  
    0     0     0     0     0     0     0     0     0     0
```

```
>> a+1 == b
```

```
ans =  
    1     1     1     1     1     1     1     1     1     1
```

Note: 1 means “true”, 0 means “false”

# Conditionals

## *If statement*

➤ Simplest form:

```
if (logical-expression)
    :
    commands
    :
end
```

## *If statement*

More general form:

```
if (logical-expression)
    commands
elseif (logical-expression)
    commands
else
    commands
end
```

# Loops

## For loop

➤ Simplest form:

```
for j=1:m
    :
    commands
    :
end
```

Examples of other constructs

```
for j=0:3:31
```

```
for j=100:-1:0
```

```
for j=0.1:0.1:2.4
```



## *Example:*

Simple version of script to compute the square root of 5. [shown in class]

```
tol = 1.e-10;  
a = 5;  
x = a;  
for i=1:100  
    x = 0.5*(x+a/x);  
    if abs(x^2-a) < tol  
        break;  
    end  
end
```

## While loop

➤ Generic form:

```
while (logical)
    :
    commands
    :
end
```

➤ For the square example you can achieve the same result with a while loop

```
tol = 1.e-10;
a = 5;
x = a;
while abs(x^2 - a) > tol
    x = 0.5*(x+a/x);
end
```

 The above needs a fix [potential for infinite loop]