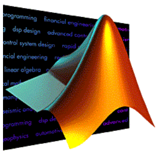
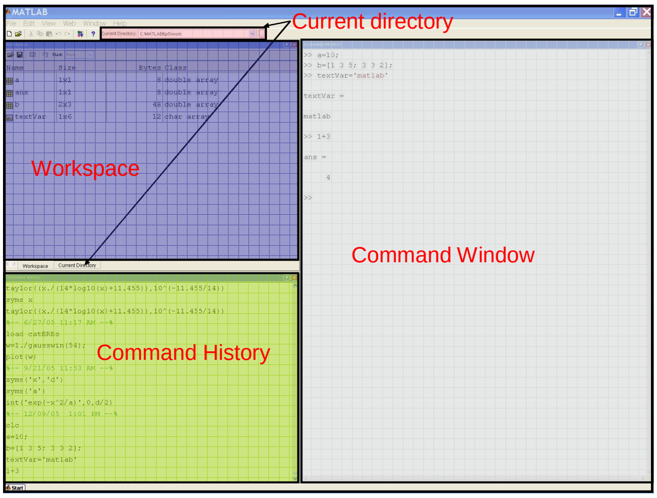
6.094
Introduction to Programming in MATLAB

Lecture 1: Variables, Operations, and Plotting

Danilo Šćepanović
Patrick Ho

IAP 2009





Current directory

Current Directory: C:\MATLAB6p5\work

Workspace

Name	Size	Bytes	Class
a	1x1	8	double array
ans	1x1	8	double array
b	2x3	48	double array
textVar	1x6	12	char array

Command Window

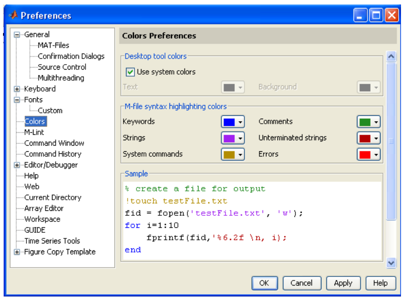
```
>> a=10;
>> b=[1 3 5; 3 3 2];
>> textVar='matlab'

textVar =

matlab

>> 1+3

ans =

     4

>>
```

Command History

```
taylor((x./(14*log10(x)+11.455)),10^(-11.455/14))
syms x
taylor((x./(14*log10(x)+11.455)),10^(-11.455/14))
%-- 6/27/05 11:17 AM --%
load catERBs
w=1./gausswin(54);
plot(w)
%-- 9/21/05 11:53 AM --%
syms('x','d')
syms('a')
int('exp(-x^2/a)',0,d/2)
%-- 12/09/05  1:01 PM --%
clc
a=10;
b=[1 3 5; 3 3 2];
textVar='matlab'
1+3
```

Customization

- File → Preferences
  - Allows you personalize your MATLAB experience



Preferences

Colors Preferences

Desktop tool colors

☑ Use system colors

Text ▨ Background ▨

M-file syntax highlighting colors

Keywords	🟦	Comments	🟩
Strings	🟪	Unterminated strings	🟥
System commands	🟨	Errors	🟥

Sample

```
% create a file for output
!touch testFile.txt
fid = fopen('testFile.txt', 'w');
for i=1:10
    fprintf(fid,'%6.2f \n, i);
end
```

MATLAB Basics

- MATLAB can be thought of as a super-powerful graphing calculator
  - Remember the TI-83 from calculus?
  - With many more buttons (built-in functions)
- In addition it is a programming language
  - MATLAB is an interpreted language, like Scheme
  - Commands executed line by line

Conversing with MATLAB

- **who**
  - MATLAB replies with the variables in your workspace
- **what**
  - MATLAB replies with the current directory and m-files in the directory
- **why**
- **help**
  - **The most** important function for learning MATLAB on your own
  - More on help later

Outline

(1) Getting Started
(2) Making Variables
(3) Manipulating Variables
(4) Basic Plotting

Variable Types

- MATLAB is a weakly typed language
  - ➢No need to initialize variables!
- MATLAB supports various types, the most often used are
  - » **3.84**
    - ➢64-bit double (default)
  - » **‘a’**
    - ➢16-bit char
- Most variables you’ll deal with will be arrays or matrices of doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc.

Naming variables

- To create a variable, simply assign a value to a name:
  - » **var1=3.14**
  - » **myString=‘hello world’**
- Variable names
  - ➢first character must be a LETTER
  - ➢after that, any combination of letters, numbers and _
  - ➢CASE SENSITIVE! (**var1** is different from **Var1**)
- Built-in variables
  - ➢**i** and **j** can be used to indicate complex numbers
  - ➢**pi** has the value 3.1415926…
  - ➢**ans** stores the last unassigned value (like on a calculator)
  - ➢**Inf** and **-Inf** are positive and negative infinity
  - ➢**NaN** represents ‘Not a Number’

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically
  - » **‘Hello 6.094’**
- To remove “ans =“, use **disp()**
  - » **disp('Hello 6.094')**
- **sprintf()** allows you to mix strings with variables
  - » **class=6.094;**
  - » **disp(sprintf('Hello %g', class))**
    - ➢The format is C-syntax

Scalars

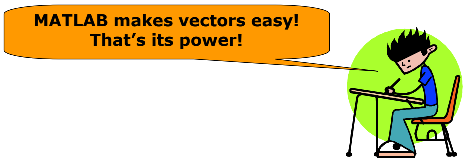
- A variable can be given a value explicitly
  - » **a = 10**
    - ➢shows up in workspace!
- Or as a function of explicit values and existing variables
  - » **c = 1.3*45-2*a**
- To suppress output, end the line with a semicolon
  - » **cooldude = 13/3;**

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays

(1) matrix of numbers (either double or complex)

(2) cell array of objects (more advanced data structure)

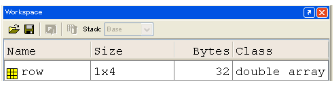


Row Vectors

- Row vector: comma or space separated values between brackets
  - » **row = [1 2 5.4 -6.6];**
  - » **row = [1, 2, 5.4, -6.6];**
- Command window:

```
>> row=[1 2 5.4 -6.6]

row =

    1.0000    2.0000    5.4000   -6.6000
```

- Workspace:



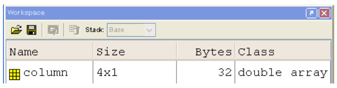
Name	Size	Bytes	Class
row	1x4	32	double array

Column Vectors

- Column vector: semicolon separated values between brackets
  » **column = [4;2;7;4];**
- Command window:

```
>> column=[4;2;7;4]

column =

     4
     2
     7
     4
```

- Workspace:



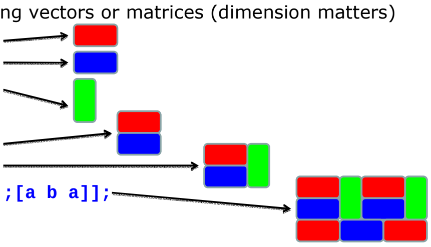
```
Name      Size      Bytes Class
column    4x1          32 double array
```

Matrices

- Make matrices like vectors
- Element by element
  » **a= [1 2;3 4];**

$$a=\begin{bmatrix}1&2\\3&4\end{bmatrix}$$

- By concatenating vectors or matrices (dimension matters)
  » **a = [1 2];**
  » **b = [3 4];**
  » **c = [5;6];**
  » **d = [a;b];**
  » **e = [d c];**
  » **f = [[e e];[a b a]];**



save/clear/load

- Use **save** to save variables to a file
  » **save myfile a b**
    - saves variables a and b to the file myfile.mat
    - myfile.mat file in the current directory
    - Default working directory is
  » **\MATLAB\work**
    - Create own folder and change working directory to it
  » **MyDocuments\6.094\day1**
- Use **clear** to remove variables from environment
  » **clear a b**
    - look at workspace, the variables a and b are gone
- Use **load** to load variable bindings into the environment
  » **load myfile**
    - look at workspace, the variables a and b are back
- Can do the same for entire environment
  » **save myenv; clear all; load myenv;**

Exercise: Variables

- Do the following 5 things:
  - Create the variable **r** as a row vector with values 1 4 7 10 13
  - Create the variable **c** as a column vector with values 13 10 7 4 1
  - Save these two variables to file **varEx**
  - clear the workspace
  - load the two variables you just created

```
» r=[1 4 7 10 13];
» c=[13; 10; 7; 4; 1];
» save varEx r c
» clear r c
» load varEx
```

Outline

(1) Getting Started
(2) Making Variables
(3) Manipulating Variables
(4) Basic Plotting

Basic Scalar Operations

- Arithmetic operations (**+**,**-**,*****,**/**)
  » **7/45**
  » **(1+i)*(2+i)**
  » **1 / 0**
  » **0 / 0**
- Exponentiation (**^**)
  » **4^2**
  » **(3+4*j)^2**
- Complicated expressions, use parentheses
  » **((2+3)*3)^0.1**
- Multiplication is NOT implicit given parentheses
  » **3(1+0.7) gives an error**
- To clear cluttered command window
  » **Clc**

Built-in Functions

- MATLAB has an **enormous** library of built-in functions

- Call using parentheses – passing parameter to function

```
» sqrt(2)
» log(2), log10(0.23)
» cos(1.2), atan(-.8)
» exp(2+4*i)
» round(1.4), floor(3.3), ceil(4.23)
» angle(i); abs(1+i);
```

Help/Docs

- To get info on how to use a function:

```
» help sin
```

  - ➢ Help contains related functions
- To get a nicer version of help with examples and easy-to-read descriptions:

```
» doc sin
```

- To search for a function by specifying keywords:

```
» doc  + Search tab
» lookfor hyperbolic
```

One-word description of what you're looking for

Exercise: Scalars

- Verify that e^(i*x) = cos(x) + i*sin(x) for a few values of x.

```
» x = pi/3;
» a = exp(i*x)
» b = cos(x)+ i*sin(x)
» a-b
```

size & length

- You can tell the difference between a row and a column vector by:
  - ➢ Looking in the workspace
  - ➢ Displaying the variable in the command window
  - ➢ Using the size function

```
>> size(row)              >> size(column)

ans =                     ans =

     1     4                   4     1
```

- To get a vector's length, use the length function

```
>> length(row)            >> length(column)

ans =                     ans =

     4                         4
```

transpose

- The transpose operators turns a column vector into a row vector and vice versa

```
» a = [1 2 3 4]
» transpose(a)
```

- Can use dot-apostrophe as short-cut

```
» a.'
```

- The apostrophe gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers

```
» a = [1+j 2+3*j]
» a'
» a.'
```

- For vectors of real numbers .' and ' give same result

Addition and Subtraction

- Addition and subtraction are element-wise; sizes must match (unless one is a scalar):

$$\begin{array}{r} [12 \quad 3 \quad 32 \quad -11] \\ +[2 \quad 11 \quad -30 \quad 32] \\ \hline =[14 \quad 14 \quad 2 \quad 21] \end{array}$$

$$\begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

- The following would give an error

```
» c = row + column
```

- Use the transpose to make sizes compatible

```
» c = row’ + column
» c = row + column’
```

- Can sum up or multiply elements of vector

```
» s=sum(row);
» p=prod(row);
```

Element-Wise Functions

- All the functions that work on scalars also work on vectors
  - `» t = [1 2 3];`
  - `» f = exp(t);`
    - ➢ is the same as
  - `» f = [exp(1) exp(2) exp(3)];`

- If in doubt, check a function's help file to see if it handles vectors elementwise

- Operators (*** / ^**) have two modes of operation
  - ➢ element-wise
  - ➢ standard

Operators: element-wise

- To do element-wise operations, use the dot. BOTH dimensions must match (unless one is scalar)!
  - `» a=[1 2 3];b=[4;2;1];`
  - `» a.*b, a./b, a.^b → all errors`
  - `» a.*b', a./b', a.^(b') → all valid`

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix}=ERROR$$

$$\begin{bmatrix}1\\2\\3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix}=\begin{bmatrix}4\\4\\3\end{bmatrix}$$

$3\times1.*3\times1=3\times1$

$$\begin{bmatrix}1&1&1\\2&2&2\\3&3&3\end{bmatrix}.*\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix}=\begin{bmatrix}1&2&3\\2&4&6\\3&6&9\end{bmatrix}$$

$3\times3.*3\times3=3\times3$

$$\begin{bmatrix}1&2\\3&4\end{bmatrix}.\wedge2=\begin{bmatrix}1^2&2^2\\3^2&4^2\end{bmatrix}$$

Can be any dimension

Operators: standard

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*****) is either a dot-product or an outer-product
  - ➢ Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (**^**) implicitly uses *****
  - ➢ Can only be done on square matrices or scalars
- Left and right division (**/ \**) is same as multiplying by inverse
  - ➢ Our recommendation: just multiply by inverse (more on this later)

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}*\begin{bmatrix}4\\2\\1\end{bmatrix}=11$$

$1\times3*3\times1=1\times1$

$$\begin{bmatrix}1&2\\3&4\end{bmatrix}\wedge2=\begin{bmatrix}1&2\\3&4\end{bmatrix}*\begin{bmatrix}1&2\\3&4\end{bmatrix}$$

Must be square to do powers

$$\begin{bmatrix}1&1&1\\2&2&2\\3&3&3\end{bmatrix}*\begin{bmatrix}1&2&3\\1&2&3\\1&2&3\end{bmatrix}=\begin{bmatrix}3&6&9\\6&12&18\\9&18&27\end{bmatrix}$$

$3\times3*3\times3=3\times3$

Exercise: Vector Operations

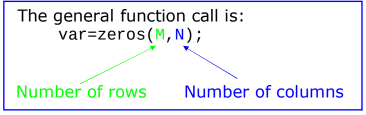
- Find the inner product between [1 2 3] and [3 5 4]
  - `» a=[1 2 3]*[3 5 4]'`

- Multiply the same two vectors element-wise
  - `» b=[1 2 3].*[3 5 4]`

- Calculate the natural log of each element of the resulting vector
  - `» c=log(b)`

Automatic Initialization

- Initialize a vector of **ones**, **zeros**, or **rand**om numbers
  - `» o=ones(1,10)`
    - ➢ row vector with 10 elements, all 1
  - `» z=zeros(23,1)`
    - ➢ column vector with 23 elements, all 0
  - `» r=rand(1,45)`
    - ➢ row vector with 45 elements (uniform [0,1])
  - `» n=nan(1,69)`
    - ➢ row vector of NaNs (useful for representing uninitialized variables)



The general function call is:

```
var=zeros(M,N);
```

Automatic Initialization

- To initialize a linear vector of values use **linspace**
  - `» a=linspace(0,10,5)`
    - ➢ starts at 0, ends at 10 (inclusive), 5 values

- Can also use colon operator (**:**)
  - `» b=0:2:10`
    - ➢ starts at 0, increments by 2, and ends at or before 10
    - ➢ increment can be decimal or negative
  - `» c=1:5`
    - ➢ if increment isn't specified, default is 1

- To initialize logarithmically spaced values use **logspace**
  - ➢ similar to **linspace**

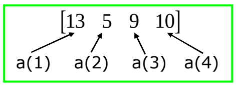
Exercise: Vector Functions

- Make a vector that has 10,000 samples of f(x) = e^{-x}*cos(x), for x between 0 and 10.

```
» x = linspace(0,10,10000);
» f = exp(-x).*cos(x);
```

Vector Indexing

- Matlab indexing starts with 1, not 0
  - We will not respond to any emails where this is the problem.
- a(n) returns the nth element

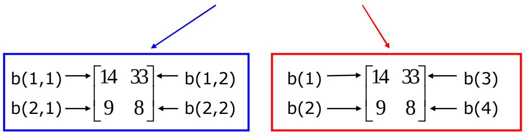


- The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.

```
» x=[12 13 5 8];
» a=x(2:3); ————————→ a=[13 5];
» b=x(1:end-1); —————→ b=[12 13 5];
```

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)
  - using linear indices (as if matrix is a vector)
- Matrix indexing: subscripts or linear indices



- Picking submatrices

```
» A = rand(5) % shorthand for 5x5 matrix
» A(1:3,1:2) % specify contiguous submatrix
» A([1 5 3], [1 4]) % specify rows and columns
```

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.

```
» a=[-1 10 3 -2];
» b=a([1 2 4;3 4 2]);
```

$$\longrightarrow b=\begin{bmatrix}-1 & 10 & -2\\ 3 & -2 & 10\end{bmatrix}$$

- To select rows or columns of a matrix, use the **:**

$$c=\begin{bmatrix}12 & 5\\ -2 & 13\end{bmatrix}$$

```
» d=c(1,:);————————→ d=[12 5];
» e=c(:,2);————————→ e=[5;13];
» c(2,:)=[3 6];  %replaces second row of c
```

Advanced Indexing 2

- MATLAB contains functions to help you find desired values within a vector or matrix

```
» vec = [1 5 3 9 7]
```

- To get the minimum value and its index:

```
» [minVal,minInd] = min(vec);
```

- To get the maximum value and its index:

```
» [maxVal,maxInd] = max(vec);
```

- To find any the indices of specific values or ranges

```
» ind = find(vec == 9);
» ind = find(vec > 2 & vec < 6);
```

  - **find** expressions can be very complex, more on this later
- To convert between subscripts and indices, use **ind2sub**, and **sub2ind**. Look up help to see how to use them.

Exercise: Vector Indexing

- Evaluate a sine wave at 1,000 points between 0 and 2*pi.
- What's the value at
  - Index 55
  - Indices 100 through 110
- Find the index of
  - the minimum value,
  - the maximum value, and
  - values between -0.001 and 0.001

```
» x = linspace(0,2*pi,1000);
» y=sin(x);
» y(55)
» y(100:110)
» [minVal,minInd]=min(y)
» [maxVal,maxInd]=max(y)
» inds=find(y>-0.001 & y<0.001)
```

BONUS Exercise: Matrices

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10

```
» mat=zeros(3,100);
» x=linspace(0,10,100);
```

- Replace the first row of the matrix with cos(x)

```
» mat(1,:)=cos(x);
```

- Replace the second row of the matrix with log((x+2)^2)

```
» mat(2,:)=log((x+2).^2);
```

- Replace the third row of the matrix with a random vector of the correct size

```
» mat(3,:)=rand(1,100);
```

- Use the sum function to compute row and column sums of mat (see help)

```
» rs = sum(mat,2);
» cs = sum(mat); % default dimension is 1
```

Outline

(1) Getting Started

(2) Making Variables

(3) Manipulating Variables

(4) Basic Plotting

Plotting Vectors

- Example

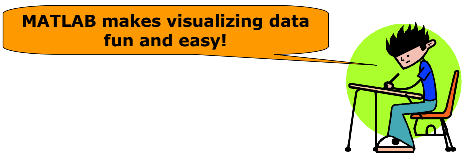
```
» x=linspace(0,4*pi,10);
» y=sin(x);
```

- Plot values against their index

```
» plot(y);
```

- Usually we want to plot y versus x

```
» plot(x,y);
```



What does plot do?

- **plot** generates dots at each (x,y) pair and then connects the dots with a line
- To make plot of a function look smoother, evaluate at more points

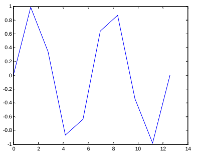
```
» x=linspace(0,4*pi,1000);
» plot(x,sin(x));
```

- x and y vectors must be same size or else you'll get an error

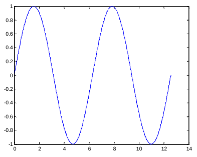
```
» plot([1 2], [1 2 3])
```

  - error!!

10 x values:



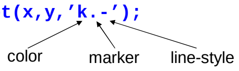
1000 x values:



Plot Options

- Can change the line color, marker style, and line style by adding a string argument

```
» plot(x,y,'k.-');
```



- Can plot without connecting the dots by omitting line style argument

```
» plot(x,y,'.')
```

- Look at **help plot** for a full list of colors, markers, and linestyles

Other Useful plot Commands

- Much more on this in Lecture 2, for now some simple commands
- To plot two lines on the same graph

```
» hold on;
```

- To plot on a new figure

```
» figure;
» plot(x,y);
```

- Play with the figure GUI to learn more
  - add axis labels
  - add a title
  - add a grid
  - zoom in/zoom out

Exercise: Plotting

- Plot f(x) = e^x*cos(x) on the interval x = [0 10]. Use a red solid line with a suitable number of points to get a good resolution.

```
» x=0:.01:10;
» plot(x,exp(x).*cos(x),’r’);
```

End of Lecture 1

(1) Getting Started
(2) Making Variables
(3) Manipulating Variables
(4) Basic Plotting