Introduction to MATLAB Software (1)

FuSen Lin

Department of Computer Science and Engineering
National Taiwan Ocean University

Scientific Computing, Fall 2011

§ 1.0 What is MATLAB? (1)

- Name came from: Matrix laboratory
- Originally created by Prof. **Cleve Moler**, U. of New Mexico, and written in **Fortran** in 1978.
- In 1984, **Jack (or John) Litttle** rewrote it in C language and created *MathWorks Inc.* and published it commercially.
- A high-level programming language with *interactive* environment–responding the results immediately
- A set of tools and facilities: the MATLAB desktop and Command Window, a command history, an editor and debugger, a code analyzer and other reports, and browsers for viewing help, the workspace and so forth. These help you use MATLAB functions and files.
- A full-featured scientific calculator–*numerical computation*

What is MATLAB? (2)

- Having programming and graphing capabilities with visualization tool, especially, using the *Handle Graphics* (握把式作圖, from V.4) with GUI (Graphic User Interface)
- A matrix-vector-oriented system and special data structures(from V. 5): *multidimentional arrays* (n-D arrays), *Cell Arrays* (like many drawers in a cabinet), and *structure arrays* (struc(field1, value1, field1, value1,...))
- A **mathematical function library:** a vast collection of computational algorithms ranging from *elementary functions* (like sum, sine, cosine) to *more sophisticated functions* (like matrix eigenvalues, Bessel functions, and fast Fourier transforms).

What is MATLAB? (3)

- Built-in many *intrinsic functions* and lots of intelligent *problem-solving tools* for particular applications (called *Toolboxes*) (more than 70 toolboxes so far)
- Having symbolic solutions by using *Symbolic Math Toolbox*), like **Maple** software
- A computational engine with dynamic linking to C, Fortran, and Maple for calling routines.
- MathWorks develops new products: **Simulink** with Blocksets (a real-time dynamic simulation) and **Stateflow** (Finite State Machines or Event-driven Systems)

Basic Features

- Basic math working in *command window* (see next page)
- display answers without semicolon
- Nothing display with semicolon (;)
- Remember the variables in *Workspace*
- Variables are not declared by the user but are created on a need-to-use basis by a memory manager
- Save the results as **.mat* (or **.tex*) files using ≫diary *name.mat* (or **.tex*)

Basic arithmetic operations

addition	+	5+3
subtraction	-	$23 - 12$
multiplication	*	$3.14 * 0.85$
division	$/$ or $\backslash$	$56/8 = 8\backslash 56$
power	$\wedge$	$5\wedge 2$

About Number Display Formats

- MATLAB use double-precision floating-point arithmetic, which is accurate to approximately 15 digits; however, it displays only 5 digits by default. To display more digits, type **format long**.
- The new version of MATLAB (7.0 or upper version) has the *variable precision arithmetic* with **vpa**. You can specify the number of digits as what you desire. (see examples)

Number Display Formats

Command	average_cost	Comments
format long	35.83333333333334	16 digits format
format short e	3.5833e+01	5 digits plus exponent
format long e	3.583333333333334e+01	16 digits plus exponent
format hex	4041eaaaaaaaaaab	hexadecimal
format bank	35.83	2 decimal digits
format +	+	positive, negative, or zero
format rat	215/6	rational approximation
format short	35.8333	default display

Variable Naming Rules

Rule	**Commonts**
Variable names are case sensitive (large and small letters are **different**).	fruit, Fruit, FrUiT, and FRUIT are all **different** MATLAB variables.
Variable names can be of **any length**.	Characters beyond the 63th are ignored.
Variable names must start with a letter, followed by any number of letters, digits, or underscores.	Punctuation characters are **not allowed** since many have special meaning to MATLAB.

Several Special Variables

Variable	Value
ans	Default variable name used for results
pi	Ratio of the circumference of a circle to its diameter
eps	**Smallest** number such that when added to 1 creates a floating-point number greater than 1 on the computer
inf	**Infinity**, e.g., $1/0$
NaN	Not-a-Number, e.g., 0/0
i and j	$i = j = \sqrt{-1}$
realmin	The **smallest** usable positive real number
realmax	The **largest** usable positive real number

Exercise 1.0

- 1. Calculate the value of the following functions at $x = 10$ using three different ways and also plot their graphs with "easy way":
- $\gg f =' \sin(2*x+5)*\cos(3*x)*(2*x^2-7)'$
- $\gg f = \mathrm{inline}('\sin(2*x+5)*\cos(3*x)*(2*x^2-7)')$

§1.1 Vectors and Plotting

- Row vectors: $\gg$ x=[10.1 20.2 30.3]
- Column vectors: $\gg$ x=[10.1; 20.2; 30.3]
- To change the orientation of a vector from row to column or column to row, use an apostrophe ('):
  $\gg$ x=[10.1 20.2 30.3]'
- Equal spacing vectors:
- $\gg$ x=linspace(<Left Endpoint>, <Right Endpoint>, <Number of Points>)

Vectors

- Row vectors: Use colon notation: $\gg$ x = 20:24
- Equivalent to $\gg$ x=[20 21 22 23 24]
- Using the stride: $\gg$ x = 20:2:29
- Equivalent to $\gg$ x=[20 22 24 26 28]
- ($<$Starting index$>$:$<$ Stride $>$:$<$Bounding index$>$)
- $\gg$ x=logspace($<$Left Endpoint$>$, $<$Right Endpoint$>$, $<$Number of Points$>$)

Examples

- $\gg$ x=linspace(a, b, n)
- The k-th point: $x_k = a + (k-1) * (b-a)/(n-1)$
- $\gg$ x=logspace(a ,b, n)
- The k-th point: $x_k = 10^{[a+(k-1)*(b-a)/(n-1)]}$

Elementary (built-in) functions

- $\gg$ y=abs(x)
- $\gg$ y=sin(x); y=cos(x); and so on.
- $\gg$ y=asin(x); y=acos(x); % the arcsine and arccosine
- $\gg$ y=log(x) % natural logarithmic function
- $\gg$ y=log10(x); y=log2(x); % the common logarithm and base 2 logarithm
- $\gg$ y=exp(x) % natural exponential function

Elementary functions

- $\gg$ y=min(x) % find the minimum number in vector x
- $\gg$ y=max(x); % find the maximum number in vector x
- $\gg$ y=mean(x); % find the average number in vector x
- $\gg$ y=sum(x) % find the sum of all numbers in vector x
- $\gg$ y=sort(x) % sorting the numbers in vector x

Matlab codes

```
n = 21;
h = 1/(n-1);
for k=1:n
  x(k) = (k-1)*h;
end

%%%%%%%%%%%%%%%%%%%%%%%%

n = 21;
h = 1/(n-1);
x = zeros(1, n);
for k=1:n
  x(k) = (k-1)*h;
end
```

Matlab codes

```
%Compute sin(x) for 21 points on [0, 1]
 n = 21;
 x = linspace(0,l,n);
 y = zeros(l,n);
 for k=1:n
   y(k) = sin(x(k));
 end

%%%%%%%%%%%%%%%%%%%%

% Compute sin(x) for 21 points on [0, 2pi]
 n = 21;
 x = linspace(0,l,n);
 y = sin(2*pi*x);
```

Advantages of Vectorization

- **Speed**: The build-in MATLAB functions provide results of several calls faster if called once with the corresponding vector argument(s).
- **Clarity**: Easier to read a vectorized MATLAB script than its scalar-level counterpart.
- **Education**: requiring to think at the vector level and fostering the style of algorithmic thinking.

Exploiting Symmetry

- Example of plotting $\sin(2\pi x)$ for $x \in [0, 1]$
- Using the properties:

$$\sin\left(\frac{\pi}{2} + x\right) = \sin\left(\frac{\pi}{2} - x\right)$$

- and

$$\sin(\pi + x) = -\sin(x)$$

Matlab codes

```
 function y = SinValue(n)

% Compute sin(x) for 21 points on [0, 2pi] with symmetry
% n must be a positive integer of multiple of 4

 m = (n+1)/4;  %  m = 5;  n = 4*m+1;
 x = linspace(0,1,n);
 a = x(1:m+1);
 y = zeros(1, n);
 y(1:m+1) = sin(2*pi*a);
 y(2*m+1:-1:m+2) = y(1:m);
 y(2*m+2:n) = -y(2:2*m+1);
```

Displaying Tables

- Creating a script file
- Use '%' notation to give *comments* for the command codes.
- **disp(' *** ')**: to display strings enclosed by single quotes.
- **sprintf**: to produce a string that includes the values of named variables.

  sprintf($<$ *String with Format Specification* $>$, $<$ *List of Variables* $>$)

- **disp(sprintf(' *** ', names of variables))**

Matlab codes

```
% Script File: SineTable
% Prints a short table of sine evaluations.
 clc     % clear the command window and home the cursor.
 n = 21;
 x = linspace(0,1,n);
 y = sin(2*pi*x);
 disp('                     ')
 disp('k    x(k)   sin(x(k))')
 disp('---------------------')
 for k=1:21
   degrees = (k-1)*360/(n-1);
   disp(sprintf('%2.0f %3.0f %6.3f', k, degrees, y(k)));
 end
 disp( '                     ');
 disp('x(k) is given in degrees.')
 disp(sprintf ('One Degree = %5.3e Radians', pi/180))
```

plotting $y = \sin(x)$

- Use '**plot**' command to create a *figure*.
- Use '**title**', '**xlabel**', and '**ylabel**' to *comment* the plot.
- Use '**pause(1)**' command permits a 1-second viewing of each plot.

Matlab codes

```
%1.1.5  A simple plot of y = sin(x)

 n = 21;
 x = linspace(0, 1, n);
 y = sin(2*pi*x);
 plot(x, y)
 title('The Function y = sin(2*pi*x)')
 xlabel('x   (in radians)')
 ylabel('y')
```

Matlab codes

```
n = 200;
x = linspace(0, 1, n);
y = sin(2*pi*x);
plot(x, y)
title('The function y = sin(2*pi*x)')
xlabel('x   (in radians)')
ylabel('y')
```

Matlab codes

```
% Script File: SinePlot
% Displays increasingly smooth plots of sin(2*pi*x).

 close all        % Close all windows.
 for n = [4 8 12 16 20 50 100 200 400]
   x = linspace(0, 1, n);
   y = sin(2*pi*x);
   plot(x,y)
   title(sprintf('Plot of sin(2*pi*x) based upon...
         n = \%3.0f points.', n))
   pause(1)
 end
```

Exercise 1.1

- 1. Calculate the length of the power cable in Lecture 1 by locating root function **fzero** and *arc length formula*.
- 2. plot $y = \cos(x)$ for $x \in [0, 2\pi]$ by applying vectorization and symmetry.
- 3. Display the results of problem 2 in a table.
- 4. Plot the function $y = \sin(x)$ for $x \in [-\pi, \pi]$ and their Taylor polynomials

$$S_1(x) = x, \quad S_2(x) = x - \frac{x^3}{3!}, \quad S_3(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!},$$

§ 1.2 More Vectors, More Plotting, and Matrices

Plotting the Rational Function

- Plotting the Rational Function

$$f(x) = \left(\frac{1 + \frac{x}{24}}{1 - \frac{x}{12} + \frac{x^2}{384}} \right)^8, \quad x \in [0, 1]$$

- An approximation to the function e^x.

Matlab codes of ExpPlot

```
% Script File: ExpPlot1
% Approximate exp(x) by the function:
% f(x)=((1+x/24)/(1-x/12+(x^2/384))^8 across [0, 1].
% Scalar operations--using for-loop.

 close all        % Close all windows.
 n = 200;
 x = linspace(0, 1, n);
 y = zeros(1,n);
 for k = 1:n
   y(k) = ((1+x(k)/24)/(1-x(k)/12+(x(k)/384*x(k))^8;
 end
 plot(x, y)
```

1.2.1 Vector Operations

- Operations of vector scale, vector add, vector subtract
- Operation of pointwise vector multiply '$.*$'
- Operation of pointwise vector divide '$./$'
- Operation of pointwise vector exponentiation '$.\wedge$'
- Write your programs with MATLAB using *vector* or *matrix* operations

Matlab codes of ExpPlot

```
% Script File: ExpPlot
% Approximate exp(x) by the function:
% f(x)=((1+x/24)/(1-x/12+(x^2/384))^8 across [0, 1].
% Vector operations--Using pointwise vector operations.

 close all        % Close all windows.
 x = linspace(0, 1, 200);
 num = 1 + x/24;
 denom = 1 - x/12 + (x/384).*x;
 quot = num ./ denom;
 y = (quot).^8;
 plot(x, y, x, exp(x))
```

Plotting the tan(x) Function

- Plot the graph of a function using **plot** with autoscaling feature.
- Example: Plot the **tan(x) function**

$$\tan(x) = \frac{\sin(x)}{\cos(x)}, \quad x \in [-\pi/2, 11\pi/2]$$

- Using the **axis** function to scale the axes manually.
- Syntax: $\gg$ axis([xmin xmax ymin ymax])

Matlab codes for TangentPlot

```
% Script File: TangentPlot1
% Plots the function tan(x), -pi/2 <= x <= 11pi/2

 close all
 x = linspace(-pi/2,11*pi/2,200);
 y = tan(x);
 plot(x, y)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 x = linspace(-pi/2,11*pi/2,200);
 y = tan(x);
 plot(x, y)
 axis([-pi/2 9*pi/2 -10 10])
```

Matlab codes for TagentPlot

```
% Script File: TangentPlot1
% Plots the function tan(x), -pi/2 <= x <= 11pi/2
 close all
 x = linspace(-pi/2, pi/2, 40);
 y = tan(x);  plot(x, y)
 ymax = 10;
 axis([-pi/2 9*pi/2 -ymax ymax])
 title ('The Tangent Function'),
 xlabel('x'),  ylabel('tan(x)')
 hold on
 for k=1:4
   xnew = x + k*pi;
   plot(xnew, y);
 end
 hold off
```

1.2.1 Superpositioning (Multi-Graph)

- **hold on:** to superimpose (疊置在上面 (使重疊); 加上去 = superpose) all subsequent plots on the current figure window.
- **hold off:** to shut down the superpositioning feature and set the stage for normal plotting thereafter.
- Another way to superimpose in the same plot is by calling **plot** with an extended parameter list.
- The syntax of multi-graph's **plot**:
  plot(<First graph specification>, ...,<Last graph specification>)
- The form of each graph specification:
  <Vector, Vector, String (optional)>

Matlab codes for Plotting with superposition

```
% Script File: SinAndCosPlot
% Plots the functions sin(2*pi*x) and cos(2*pi*x)
% across [0,1] and marks their intersection.

 close all
 x = linspace(0,1,200);
 y1 = sin(2*pi*x);
 y2 = cos(2*pi*x);
 plot(x,y1,x,y2,'--',[1/8,5/8],[1/sqrt(2),-1/sqrt(2)],'*')
```

Polygons with n Vertices

- If x and y are column vectors that contain the *coordinate values*, then **plot(**x**,** y**)** does not display the polygon because (x_n, y_n) is not connected to (x_1, y_1). Need to make a **concatenation** (連接; 連鎖) of vectors.
- If $r1, r2, ..., rm$ are row vectors, then

$$v = [r1r2 \cdots rm]$$

  is also a row vector.
- If $c1, c2, ..., cm$ are column vectors, then

$$v = [c1; c2; \cdots ; cm]$$

  is also a column vector.

Some Commands for Plotting

- The command **axis equal** ensures that the x-distance per pixel is the same as the y-distance per pixel.
- The command **axis off** does **not** display the coordinate axes.
- The command **subplot(**m, n, k**)** breaks up the current figure into a m-by-n array of sub-windows, and place the next plot in the kth one of these. They are indexed **left to right** and **top to bottom**.

Some 2D-Plotting

- The command **axis** ensures that the x-distance per pixel is the same as the y-distance per pixel.
- The command **axis** does **not** display the coordinate axes.
- The command **subplot(**m, n, k**)** breaks up the current figure into a m-by-n array of sub-windows, and place the next plot in the kth one of these. They are indexed **left to right** and **top to bottom**.

1.2.4 Some Matrix Computations

- Consider the problem of plotting the function

$$f(x) = 2\sin(x) + 3sin(2x) + 7\sin(3x) + 5\sin(4x)$$

  across the interval $[-10, 10]$.
- 1. Scalar-level script (see p.23)
- 2. Vector-level script
- 3. Matrix-level script
- Ideas: A linear combination of vectors is equivalent to matrix-vector multiplication.

Matrix-Vector Product

$$2\begin{bmatrix}3\\1\\4\\7\\2\\8\end{bmatrix}+3\begin{bmatrix}5\\0\\3\\8\\4\\2\end{bmatrix}+7\begin{bmatrix}8\\3\\3\\1\\1\\1\end{bmatrix}+5\begin{bmatrix}1\\6\\8\\7\\0\\9\end{bmatrix}=\begin{bmatrix}3&5&8&1\\1&0&3&6\\4&3&3&8\\7&8&1&7\\2&4&1&0\\8&2&1&9\end{bmatrix}\begin{bmatrix}2\\3\\7\\5\end{bmatrix}$$

Creating A Matrix

- 1. Typing all entries row by row.
- 2. Using two for-loops to initialize a matrix by creating a **zero matrix** first.
- 3. Aggregating its columns to form a matrix:

  $$[< Column1 >< Column2 > \cdots < Columnm >]$$

  Note: all column vectors must have the same length.
- 4. Using a single loop whereby each pass sets up a single column (test sumOfSines).
- Note: Creating a $m \times n$ zero matrix: $\gg A = \text{zeros}(m, n)$.

Another Example of Matrix Computations

- Consider the problem of plotting the two functions

$$\begin{aligned} f(x) &= 2\sin(x) + 3sin(2x) + 7\sin(3x) + 5\sin(4x) \\ g(x) &= 8\sin(x) + 2sin(2x) + 6\sin(3x) + 9\sin(4x) \end{aligned}$$

  over the interval $[-10, 10]$. (See SumOfSines2)
- In general, if the function

$$f(x) = \alpha_1 \sin(x) + \alpha_2 \sin(2x) + \alpha_3 \sin(3x) + \alpha_4 \sin(4x)$$

  We want to find α_1, α_2, α_3, α_4 so that $f(1) = -2$, $f(2) = 0$, $f(3) = 1$, and $f(4) = 5$.
- This arises the problem of solving a linear system (see p.27).

Exercise 1.2

- 1. Calculate the approximations of $e^{1/2}$ and e^8 by its Taylor polynomial with 8 terms and compare their absolute errors.
- 2. Calculate the approximations of $\ln 2$ by the two Taylor polynomials (shown in the Lecture Notes of Chapter 1) with 10 terms and compare their absolute errors.

§ 1.3 Building Exploratory Environments

An Example of Up/Down Sequence

- Suppose x_1 is given positive integer and that we define the sequence as

$$x_k = \begin{cases} x_k/2, & \text{if } x_k \text{ is even,} \\ 3x_k + 1, & \text{if } x_k \text{ is odd.} \end{cases}$$

- The sequence is: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, . . . , which is called *up/down* sequence.
- The **input** command
- The **while-loop** form
- The **if-then-else** structures
- The **switch-case** structures (see p.30)

while loops and *if-then-else* controls

```
k = 0;
while k <= 100
{command statements}
k = k +1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if A > B,
  'greater'
elseif A < B,
  'less'
elseif A == B,
  'equal'
else
  error('A and B are different data')
end
```

switch control

```
switch sign(A-B)
   case 0
     'A = B'
   case 1
     'A > B'
   case -1
     'A < B'
   otherwise
   error('A and B are different data type')
end
```

Relation Operators in MATLAB

- ' $<$ ' less than and ' $>$ ' greater than
- ' $<=$ ' less than or equal, ' $>=$ ' greater than or equal
- ' $==$ ' equal, ' $=$ ' not equal
- ' & ' and, ' | ' or, not

Some Build-in Functions

- The **rem** function: **rem(a, b)** returns the remainder when *b* is divided into *a*.
- In the command **disp(sprintf('%–5.0f', x))**, the **minus sign** left justifies the display of the value.
- The **max** function: **max(v)** returns the maximum value and the index where it occurs.
- The comparison $x <= x(1)$: returns a vector of **0's (false)** and **1's (true).**

Some Build-in Functions

- The **sort** function: **sort(v)** returns an ordered sequence which values are listed *from small to large.*
- The **find** function: **find(v)** returns a vector of subscripts that designate which entries are **nonzero**.
- The **diary *filename*** function: creates a file and starts to store everything that is now written in the command window.
- The **diary off** function: stops to store anything written in the command window.

The Build-in Random Functions

- Many simulations performed by computational scientists involve random processes.
- In MATLAB the build-in functions **rand** and **randn** work for random processes.
- The function **rand(n, 1)**: creates a length-n column vector of real numbers chosen randomly from the interval (0, 1).
- The function **randn(n, 1)**: creates a length-n column vector of real numbers chosen randomly and distributed normally.

The hist Function

- The histogram function: **hist** can be used in several ways and the script shows two possibilities:
- **hist(x, n)**: reports the distribution of the *x*-values according to where the "belong" with respect to *n* equally spaced bins spread across the interval [**min(x), max(x)**].
- **hist(x, linspace(-a, a, m))**: The bins locations can be specified by passing a *m*-vector for the histogram of normally distributed data on the interval [-a, a].

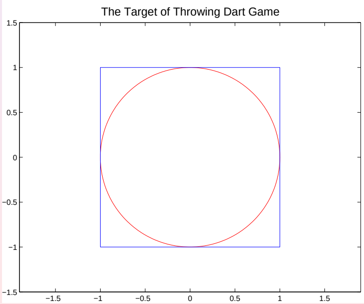
Generating Random Integers

- Using **rand** or **randn** functions plus the **floor** or **ceil** functions to generate random integers.
- An example–the **dice** problem: simulating 1000 rolls of a pair dice and displaying the outcome in histogram form.
- The command **z = floor(6*rand(n, 1) + 1)** computes a random vector of integers selected from $\{1, 2, 3, 4, 5, 6\}$ and assigns them to z.
- **Note:**

$$\text{floor}(x) + 1 = \text{ceil}(x) \quad \forall x \in (-\infty, \infty)$$

Solving Non-random Problems

- Random simulations can be used to answer **non-random** questions.
- The **throwing dart** problem: suppose we throw *n* darts at the circle-in-square target.



The Dart Problem

- Assume that the darts land anywhere on the square with *equal probability*.
- After a large number of throws, the fraction (probability) of the darts that land *inside the circle* should be approximately equal to $\pi/4$, the ratio of the circle area to the the square's area.
- By simulating the throwing of a large number of darts, we can produce an estimate of π:

$$\pi \approx 4 \cdot \frac{\text{Number of Throws Inside the Circle}}{\text{Total Number of Throws}}$$

Simulation of *Monte Carlo*

- Simulation in this spirit is called *Monte Carlo* method.
- The command **rand('seed', .123456)**: starts the random number sequence with a prescribed *seed*, which enables to repeated the random simulation with exactly the same sequence of underling random numbers.
- The **any** and **all** functions: indicates whether **any** or **all** the components of a vector are *nonzero*.

Polygon Smoothing

- Let x and y are $(n+1)$-vectors and $x_1 = x_{n+1}$ and $y_1 = y_{n+1}$ then
$$\text{plot}(x, y, x, y, '*')$$
display a polygon.
- If we compute
$$xnew = [(x(1:n) + x(2:n+1))/2; (x(1) + x(2))/2];$$
$$ynew = [(y(1:n) + y(2:n+1))/2; (y(1) + y(2))/2];$$
$$plot(xnew, ynew)$$
then a new polygon is displayed that is obtained by *connecting the side midpoints* of the original polygon.
- We wish to explore what happens when *this process is repeated*.

Polygon Smoothing

- How to specify the starting polygon?
- The **ginput** command supports mouse-click input.