Introduction to MATLAB

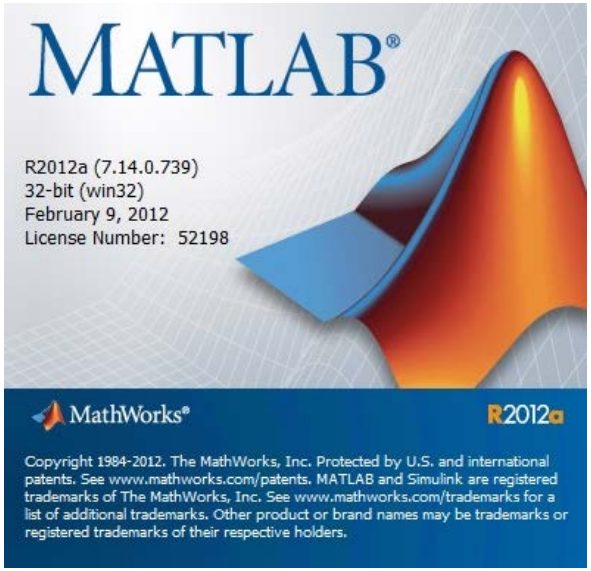


Image Source: http://www.mathworks.com/

This lecture is modified from a web-based tutorial at UBC
http://www.cs.ubc.ca/spider/cavers/MatlabGuide/guide.html

Reference:
[Gonzalez, Woods and Eddins] Rafael C. Gonzalez, Richard E. Woods and Steven L. Eddins, First Edition, Digital Image Processing using MATLAB, Prentice-Hall, Inc.

http://www.cs.ubc.ca/~mitchell/matlabResources.html

Introduction

1. MATLAB provides a powerful interactive computing environment for numeric computation, visualisation, and data analysis.
2. It has a wide range of commands and functions.
3. On-line help and manuals are available on-line.
4. Getting started: enter matlab <rtn> or press the MATLAB icon
5. For demo: enter demo <rtn>. It is one of the best ways to learn MATLAB from numerous examples.
6. To exit: enter exit <rtn>

Variables, Expressions and Statements

1. MATLAB statements typically take one of the following two forms:
   variable = expression or
   expression
   e.g., `a = 3;` or `5`

2. All variable (and function) names consist of a letter followed by any number of numbers, letters and underscores, e.g., ab12, myvariable1, myvariable2, ...

3. MATLAB is case sensitive.

4. Expressions are composed from operators, function calls and variable names. >> is the command prompt.
   e.g., >> a = min(2,7);

5. A carriage ("Enter") return normally signifies the end of a statement, causing MATLAB to interpret the command and print its result. If the last character of a statement is a ; (semicolon), however display of the result is suppressed.
   e.g.,

```
>> a = min(3,8)
a =
     3
>>
```

```
>> a = min(3,8);
>>
```

6. This feature is especially useful when the result of a computation is a large matrix. Several statements separated by commas may be placed on a single line.

7. When an expression is not explicitly assigned to a variable with the assignment operator (=), MATLAB automatically stores the result in the special variable ans.

e.g.,

```
>> min(3,8)
ans =
     3
>>
```

8. During a MATLAB session you may forget the names of variables stored in your *workspace*. The command who lists the names of all your variables. If you want to know their size as well, use the command whos.

```
>> who
Your variables are:
a    ans
```

```
>> whos
  Name      Size            Bytes  Class
  a         1x1                 8  double array
  ans       1x1                 8  double array
Grand total is 2 elements using 16 bytes
```

9. By default MATLAB stores all variables until the session is terminated. To remove a variable from the workspace, we can use the command clear var_name. WARNING: clear with no arguments removes all variables from the workspace.
   e.g., >> clear a
   >> a
   ??? Undefined function or variable 'a'.

10. At any time you can interrupt the computation of a MATLAB statement with Cntrl-C. It is very useful if you want to stop a program immediately.

Matrices and MATLAB

1. Essentially, the only data objects in MATLAB are rectangular numerical matrices.

2. There are no restrictions placed on the dimensions of a matrix (except by system resources). But special meaning is sometimes attached to 1x1 matrices (scalars) and matrices with only one row or column (vectors).
   e.g., `a` = 1x1 matrix, `[14 8 2]` = 1x3 vector with one row and three columns.

3. The memory required for the storage of each matrix is *automatically allocated* by MATLAB.

4. The easiest way to enter a matrix into MATLAB is to provide an explicit list of elements enclosed in square brackets [ ]. MATLAB uses the following conventions:
   - A matrix element can be any valid MATLAB expression.
   - Matrix elements are separated by spaces or commas.
   - A semicolon or carriage return is used to indicate the end of a row.

5. For example, entering the assignment statement

```
>> A = [1 2 4.5; 8/2.0 6 5]

A =
    1.0000 2.0000 4.5000
    4.0000 6.0000 5.0000
```

6. The 2x3 matrix is saved in variable A for future reference. If you want to see the contents of this or any other variable, simply entering its name as a command. To reference individual elements, we can enclose their subscripts in parentheses after the variable name in the usual fashion.

   e.g., `>> A(2,3)`

   ```
   ans =
         5
   ```

7. It is important to realize that MATLAB distinguishes between row and column vectors. [1 2 3] is a row vector, while [1; 2; 3] is a column vector. Column vectors can also be created by applying MATLAB's transpose operator ' (prime) to a row vector.

8. For example,

```
>> [1 2 3]'

ans =
     1
     2
     3
```

9. The transpose operator can be applied to matrices of any valid dimension.

Reading, Displaying and Saving Images

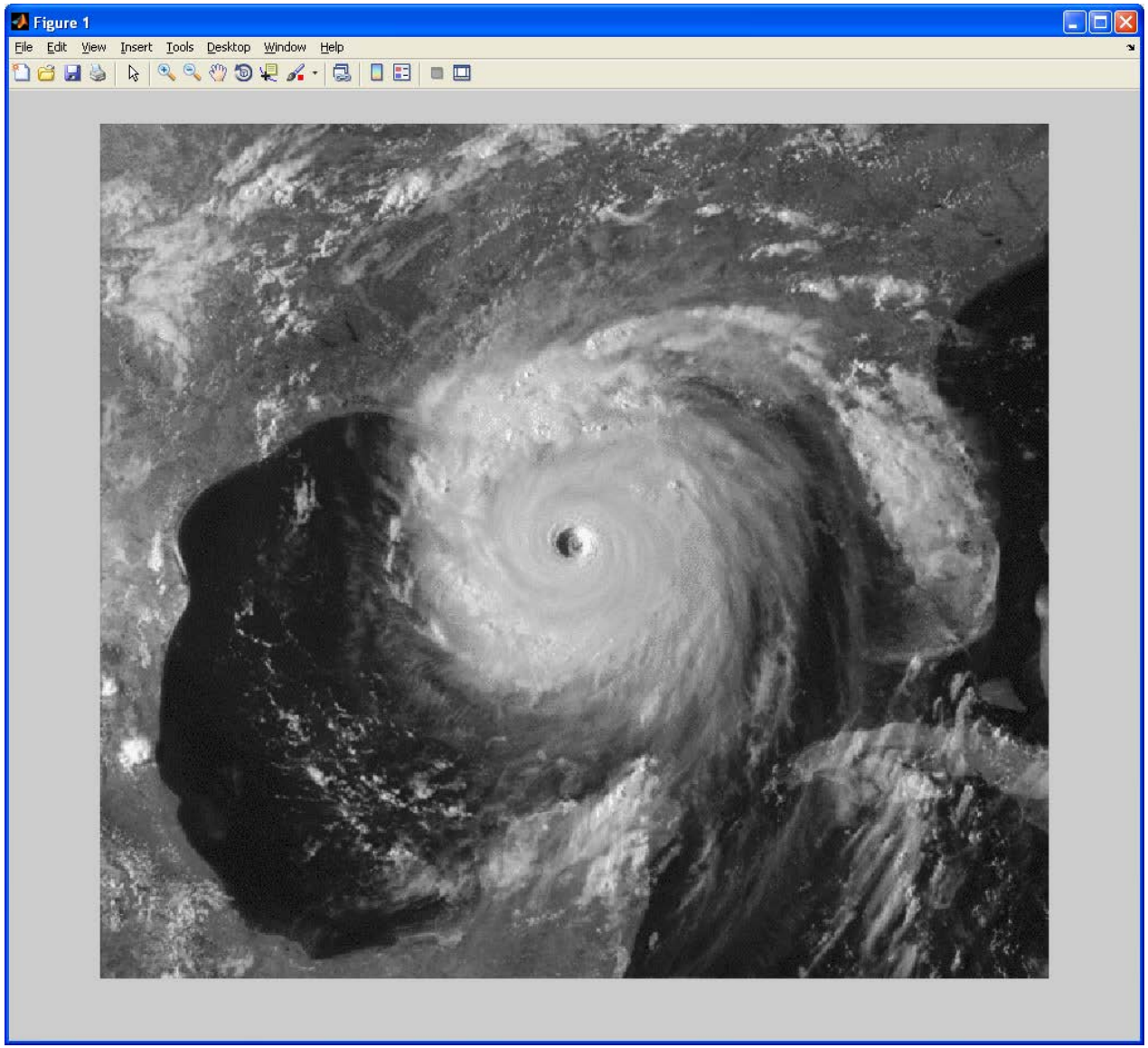
1. Images are read into the MATLAB using function imread

```
>> f=imread('Fig0111(katrina_2005_08_29_NOAA).tif');
>> size(f)
>> imshow(f);
```

2. Image formats can be tif, jpg, gif, bmp, png or xwd.
3. The variable f is a two dimensional (2D) matrix.
4. The 2D matrix can be manipulated by using arithmetic and logical operators, e.g., addition for increasing image brightness.
5. The 2D matrix can be displayed by using imshow(f).
6. The modified image can be saved by using imwrite(f, 'filename').

[Gonzalez, Woods and Eddins]

Reading, Displaying and Saving Images



[Gonzalez, Woods and Eddins]

Numbers and Arithmetic Operators

1. MATLAB uses conventional decimal notation to enter numbers. The leading minus sign and decimal point are optional, and numbers may be specified using scientific notation. The following examples are valid numbers in MATLAB.

   e.g.,

8	-1298	.3508
3.6e10	5.4E-10	0.000002

2. To build expressions, MATLAB provides the usual arithmetic operators.

   + addition; - subtraction; * multiplication;
   \ left division; / right division; ^ power.

MATLAB Functions

1. In addition to the standard arithmetic operators, MATLAB provides an extensive collection of built-in functions. For example, the most elementary mathematical functions (e.g. sin, cos, log, sqrt, ...) are available. Assume that the "short e" format has been chosen.

```
>> cos(pi/4)
ans =
        7.0711e-001
```

2. pi is an example of a function that does not require parameters and simply returns a commonly used constant.

```
>> pi
ans =
      3.1416e+000
```

3. Other functions are available in libraries of *M-files* grouped into *toolboxes*.

4. So far we have only seen functions that return a single matrix, but some functions return two or more matrices. To save these matrices, we surround the output variables by square brackets [ ] and separate them by commas. For example, [V,D] = eig(A) returns the eigenvectors and eigenvalues of A in matrices V and D.

```
>> [V,D]=eig([2 3; 4 5])
V =
-7.9681e-001 -4.9437e-001
 6.0423e-001 -8.6925e-001
D =
 -2.7492e-001                  0
                0   7.2749e+000
```


http://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors

Matrix Operations

1. The arithmetic operators presented in "Numbers and Arithmetic Expressions" Section also operate on matrices. In each case the operator behaves in a manner consistent with standard linear algebra practises.

2. The operators + and - permit the addition and subtraction of matrices, and are defined whenever the matrices have the same dimension. For example,

```
>> [1 2 3; 4 5 6] + [3 2 1; 1 1 1];

ans =
4    4    4
5    6    7
```

Matrix Operations

3. The exception to this rule is the addition (subtraction) of a scalar to (from) a matrix. In this case the scalar is added to or subtracted from each element of the matrix individually.

```
>> [1 2 3] + 1

ans =
     2  3  4
```

4. The multiplication of two matrices, denoted by A*B, is defined whenever the inner dimensions of the operands A and B are equal.

   For example, if

   ```
   C=[1 2 3; 4 5 6];
   D=[1 1 1; 2 2 2];
   x=[1 1 1]';
   ```

   then C*x, x'*x (an inner product), x*x' (an outer product) and C*D' are defined, but C*D is not defined (Matrix dimensions must agree).

5. In the special case when one of the operands is a scalar, each element of the matrix is multiplied by the scalar, e.g., C*2.

MATLAB Program output

```
C=[1 2 3; 4 5 6];
D=[1 1 1; 2 2 2];
x=[1 1 1]';

C*x
x'*x
x*x'
C*D'
C*D

ans =
     6
    15
ans =
     3
ans =
     1     1     1
     1     1     1
     1     1     1
ans =
     6    12
    15    30
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

6. MATLAB's division operator. If A is a square non-singular matrix then A\B (backslash) formally corresponds to the left multiplication of B by A^{-1} (in MATLAB, A^{-1} is inv(A)). This expression is used to solve the following type of systems of equations.

   left division:

   X = A\B solves A*X = B

7. **Left division**: If A is a square non-singular matrix and B is a vector with as many rows as A, MATLAB evaluates the expression X = A\B by finding the solution to A*X = B.

MATLAB Program output

```
A=[2 5 4; 8 4 2; 1 6 7];
B=[5; 2; 7];
X = inv(A)*B
A\B

X =
   -0.2581
    0.8710
    0.2903

ans =
   -0.2581
    0.8710
    0.2903
```

8. The expression A^p raises A to the p^{th} power. This operation is only defined if A is square and p is an scalar. For example, A^2 is equivalent to A*A, although MATLAB does not always compute powers with simple matrix multiplication.

```
>> [3 5; 4 9]^2

ans =

    29    60
    48   101
```

Array Operations

1. The "**Matrix Operations**" section described standard linear algebra matrix operations. Alternatively, element-by-element matrix arithmetic is provided by *array operations*.
2. An array operator is formed by preceding one of the symbols +, -, *, \, or / by a period (.). Of course, the matrix and array operators for addition and subtraction are equivalent, and + and - are used in either case.
   e.g.,
   ```
   >> A=[1 2 3; 4 5 6] ; B = [2 2 2; 3 3 5]; C=A.*B
   ```
   results in
   ```
   C =  2   4   6
       12 15 30
   ```
3. A.\B and A./B provide the left and right element-by-element divisions. Raising each element of a matrix to the same power is accomplished by the .^ operator.

MATLAB Program output

```
>> A=[1 2 3; 4 5 6] ; B = [2 2 2; 3 3 5]; C=A.\B
C =
  2.0000e+000  1.0000e+000  6.6667e-001
  7.5000e-001  6.0000e-001  8.3333e-001

>> A=[1 2 3; 4 5 6] ; B = [2 2 2; 3 3 5]; C=A./B
C =
  5.0000e-001  1.0000e+000  1.5000e+000
  1.3333e+000  1.6667e+000  1.2000e+000
```

4. Most standard MATLAB functions operate on a matrix element-by-element. For example,

```
>> cos(C)
ans =
                -4.1615e-001 -6.5364e-001 9.6017e-001
                8.4385e-001  -7.5969e-001 1.5425e-001
```

Vector and Matrix Manipulation

1. Vectors are easily generated with MATLAB's colon ":" notation. For example, the expression `1:5` creates the following row vector.
   `1 2 3 4 5`
   You can also create a vector using an increment other than one. For example, `1:2:7` results in the vector
   `1 3 5 7`

2. The increment may be negative and need not be an integer.

3. It is very easy to create a table using the colon notation. Experiment with the commands
   `>> x=(0:pi/4:pi)'; y=cos(x); AA=[x y]`

Vector and Matrix Manipulation

MATLAB Program output

```
>> x=(0:pi/4:pi)'; y=cos(x); AA=[x y]

AA =

                0   1.0000e+000
   7.8540e-001   7.0711e-001
   1.5708e+000   6.1232e-017
   2.3562e+000   -7.0711e-001
   3.1416e+000   -1.0000e+000
```

Vector and Matrix Manipulation

4. MATLAB permits users to easily manipulate the rows, columns, sub-matrices and individual elements of a matrix.

5. The subscripts of matrices can be vectors themselves.

6. If x and v are vectors then x(v) is equivalent to the vector [x(v(1)), x(v(2)), ...] .

```
>> x=[10 20 30 40 50 60 70];
>> v=[3 2 6];
>> x(v)
ans =
    30    20    60
```

Vector and Matrix Manipulation

7. Subscripting a matrix with vectors extracts a sub-matrix from it. For example, suppose A is an 8X8 matrix. A(1:4, 6:8) is the 4x3 sub-matrix extracted from the first 4 rows and last 3 columns of the matrix.

MATLAB Program output

```
>> A =[1:8; 2:9; 3:10; 4:11; 5:12; 6:13; 7:14; 8:15]
A =
     1     2     3     4     5     6     7     8
     2     3     4     5     6     7     8     9
     3     4     5     6     7     8     9    10
     4     5     6     7     8     9    10    11
     5     6     7     8     9    10    11    12
     6     7     8     9    10    11    12    13
     7     8     9    10    11    12    13    14
     8     9    10    11    12    13    14    15

>> size(A)
ans =
     8     8

>> A(1:4, 6:8)
ans =
     6     7     8
     7     8     9
     8     9    10
     9    10    11
```

8. When the colon operator is used by itself, it denotes all of the rows or columns of a matrix. Using the result of the table above

```
>> AA(:,1)
```

produces the first column of matrix AA.

```
ans =
     0
     0.7854
     1.5708
     2.3562
     3.1416
```

For Loops, While Loops, and Conditional Statements

1. MATLAB also provides a programming language that includes looping statements (for and while), conditional statements (if), and relational and logical operators for constructing conditions.

2. The execution of an if or while depends upon the evaluation of a *condition*. To construct conditions, MATLAB provides six relational operators

<	less than	≤	less than or equal	==	equal
>	greater than	≥	greater than or equal	~=	not equal

3. and three logical operators.

&	and	|	or	~	not

4. Note that the relational operator == compares two arguments, while = is used to assign a value to a variable.

5. When a condition compares scalar quantities, it evaluates to 1 if true and 0 if false.

6. Relational operators can also be applied to matrices of the same dimension. In this case a condition evaluates to a matrix of 0s and 1s representing the result of applying the operator to the individual elements of the matrix operands.

7. A condition evaluating to a matrix is interpreted by if or while to be true if each element of the matrix is nonzero.

8. The simplest form of an if statement is

```
if condition
    statements
end
```

9. For example,

```
A = [6 3 9];
n = 3;
if n ~= 0
    A = A/n;
end
A
```

results in the output

```
A =
     2     1     3
```

10. Each semicolon in the previous example suppresses the output of a statement's result.

11. As in other programming languages, if statements can also be used to choose between 2 or more alternatives. For example, the statement

```
x = [1 3 8]; n=3;
if n < 0
    x = [x,abs(n)];
else
    if (rem(n,2) == 0) & (n ~= 0)
        x = [x,n/2];
    else
        x = [x,n+1];
    end
end
```

adds one of three possible elements to the end of the existing row vector x, depending on the value of scalar n.

MATLAB Program output

```
>> x = [1 3 8]; n=3;
     if n < 0
          x = [x,abs(n)];
      else
          if (rem(n,2) == 0) & (n ~= 0)
              x = [x,n/2];
          else
             x = [x,n+1];
          end
      end
>>
>> x

x =

     1     3     8     4
```

12. The columns of *expression* are assigned to *variable* one-by-one and then the statements are executed. As an example, the following statements construct a vector containing the squares of the integers 2 through 5.

```
x = [];
low = 2;
hi = 5;
for i = low:hi,
        x = [x, i*i]
end
```

13. The columns of *expression* are assigned to *variable* one-by-one and then the statements are executed. As an example, the following statements construct a vector containing the squares of the integers 2 through 5.

```
x = [];
low = 2;
hi = 5;
for i = low:hi,
        x = [x, i*i]
end
```

MATLAB Program output

```
>> x = [];
 low = 2;
 hi = 5;
 for i = low:hi
      x = [x, i*i]
 end

x =
     4

x =
     4     9

x =
     4     9    16

x =
     4     9    16    25
```

14. The following statement produces a vector with the same elements as x, but they are arranged in the *reverse order*.

```
y = [];
for i = hi:-1:low
      y = [y, i*i]
end
```

15. You can also nest the for statements. The following statements create an m x n Hilbert matrix, H.

```
m=2;
n=3;
for i = 1:m
   for j = 1:n
       H(i,j) = 1/(i+j-1)
   end
end
```

MATLAB Program output

```
>> y = [];
   for i = hi:-1:low
          y = [y, i*i]
   end

y =

    25

y =

    25    16

y =

    25    16     9

y =

    25    16     9     4
```

```
>> m=2;
n=3;
for i = 1:m
   for j = 1:n
       H(i,j) = 1/(i+j-1)
   end
end

H =
     1

H =
  1.0000e+000  5.0000e-001

H =
  1.0000e+000  5.0000e-001  3.3333e-001

H =
  1.0000e+000  5.0000e-001  3.3333e-001
  5.0000e-001            0            0

H =
  1.0000e+000  5.0000e-001  3.3333e-001
  5.0000e-001  3.3333e-001            0

H =
  1.0000e+000  5.0000e-001  3.3333e-001
  5.0000e-001  3.3333e-001  2.5000e-001
```

16. Finally, MATLAB also has its own version of the while loop, which repeatedly executes a group of statements while a condition remains true.

```
while condition
        statements
end
```

17. Given a positive number val, the following statements compute and display the even powers of 2 less than or equal to val.

```
n = 0; val=10;
while 2^n <= val
    if rem(n,2) == 0
        2^n
        n = n + 1;
    else
        n = n + 1;
    end
end
```

MATLAB Program output

```
>> n = 0; val=10;
     while 2^n <= val
         if rem(n,2) == 0
              2^n
              n = n + 1;
         else
              n = n + 1;
         end
     end

ans =
     1

ans =
     4
```

M-Files: Creating Your Own Scripts and Functions

1. Users are able to tailor MATLAB by creating their own functions and scripts based on the MATLAB commands and functions.

2. Both scripts and functions are *ordinary ASCII* text files external to MATLAB. The name of each file must end in ".m".

3. It is easiest to start MATLAB from the directory containing your M-files.

4. A script may contain *any sequence of MATLAB statements*, including references to other M-files. It is invoked like any other command without arguments.

5. The first line of a function M-file starts with the word *function* and declares the name of the function and its input and output parameters.

6. For example,

```
function y = mymean(x)
% returns average or mean value
% If x is a vector, then returns a mean value
% if x is a matrix, then y is a row vector contains the mean
%       value of each column
[m,n] = size(x);
if (m == 1) & (n == 1)
            m = n;
end
y = sum(x)/n;
```

7. If we type this function into a file called mymean.m, then we can call mymean like any other MATLAB function.
   >> mymean(1:99)
   ans = 50