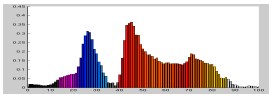
Dr. Qadri Hamarsheh



Digital Image Processing (750474)

Lecture 6

Outline of the Lecture

- Introduction to M-function programming
- Matlab Programming Example
- Relational operators
- Logical Operators
- Matlab Flow control structures

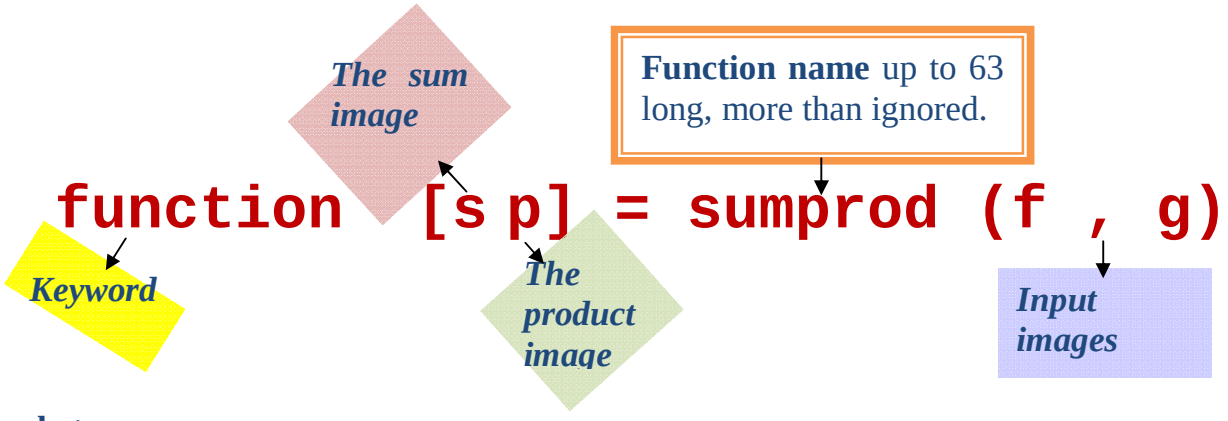
Introduction to M-function programming

- M-files:
  1. **Script**: simply execute a **series of Matlab statements**.
  2. **Function**: accept **arguments** and can product **outputs**.
- **Function m-files components:**
  - a) Function definition line.
  - b) H1 line.
  - c) Help text.
  - d) Function body.
  - e) Comments.

a) Function definition form:

function [output] = function_name (input)

Example: Function to compute the sum and the product of two images

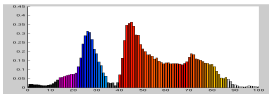


Brackets:

- Must be (multiple output).
- May be (one output).
- Without brackets or equal sign (no outputs).

- **Function calling:**

```
>> x= imread ('someimage1.tif');
>> y= imread ('someimage2.tif ');
```



```
>> [s , p] = sumprod (f , g);
```

b) H1 line:

- **Single comment line** that follows the definition line (no blank lines) between them.

```
% SUMPROD computes the sum and product of two input images.
```

- H1 line appears when user types

```
>> help function_name
```

- H1 provides **information** about the **M-file**.

c) Help text:

- **Text block** follows H1 (without any blank lines in between the two).
- Help text is used to provide comments and online help for the function, when user type:

```
>> help function_name:
```

- Matlab displays all comments lines that appear between the function name and first (executable or blank) line.

d) Function body:

- **Matlab code** that performs computation.

e) Comments.

- **%**- symbol used for **comments** declaration.

Operators:

- ***Arithmetic***: numeric computations
- ***Relational***: compare operands quantitively.
- ***Logic***: perform AND, OR and NOT.

Arithmetic operators:

- ***Matrix*** arithmetic.
- ***Array*** arithmetic.

Important: **.** (dot) operator is used for array manipulation.

```
>> A*B   % matrix multiplication.
>> A.*B  % array multiplication (A and B must be the same size).
>> C = A.*B   %  C(i,j) = A (i,j) * B (i,j)
```

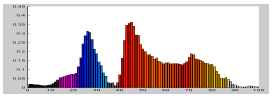
- Addition and subtraction operations are the **same** for arrays and matrices.

.+ and **.−** are not used.

Matlab Programming Example

Write an M-function, call it improd that multiplies two input images and outputs the product of the images, the maximum and the minimum values of the product, and a normalized product image whose values are in the range [0, 1].

using the Matlab text editor, write the following code



Example 6.1

```
function [p, pmax, pmin, pn] = improd (f , g)
% IMPROD computes the product and other parameters of two images.
% [ P, PMAX, PMIN, PN] = IMPROD (F , G) output the
% element-by-element product of two input images,
% F and G, the product maximum and minimum
% values, and a normalized product array with values
% in the range [0 , 1]. The input images must be
% of the same size. They can be of class uint8,
% uint16, or double. The outputs are of class double.
fd = double (f);
gd = double (g);
p = fd.*gd;
pmax = max (p (:));
pmin = min (p (:));
pn = mat2gray (p);
```

- **Note**: the input images were converted to double using the function **double** instead of **im2double**, because if the inputs were of type **uint8**, im2double would convert them to the range [0, 1]. (We want p to contain the products of the original values).

Using the function (calling)

```
>> f = [1   2  ;  3   4];
>> g = [1   2  ;  2   1];
>> [p , pmax, pmin, pn] = improd (f , g)

        p =
             1     4
             6     4

        pmax =
             6

        pmin =
             1

        pn =
             0    0.6000
         1.000    0.6000

>> help improd
```

Relational operators

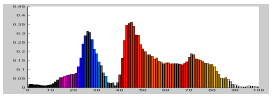
RELATIONAL OPERATOR	
OPERATOR	**NAME**
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal to
~=	Not Equal to

Example1:

Consider the following sequence of inputs and outputs:

```
>> A = [1   2   3; 4   5   6; 7   8   9];
>> B = [0   2   4; 3   5   6; 3   4   9];
```

Both A and B must be the same size



```
>>A==B
        ans =
        0    1    0
        0    1    1
        0    0    1
```

- Operation **A==B** produces a logical array of the same dimension as **A** and **B** with **1s** in locations where the corresponding elements of **A** and **B** **match**, and **0s** **elsewhere**.
- **Example2:**

```
>> A >= B
        ans
        1    1    0
        1    1    1
        1    1    1
```

- **A >= B** produces a logical array with **1s** where the elements of **A** greater than or equal to the corresponding elements of **B** and **0s** elsewhere.
- If one operand is a **scalar** ⇒matlab tests the scalar against every element of the other operand.

Logical operators

- ***Logical operators*** can operate on both logical and numeric data.

Matlab treats a **logical 1** or **nonzero** numeric quantity as **true**, and **logical 0** or numeric **0** as **false**.

```
>> A = [1  2 0 ;0 4 5];
>> B = [1 -2 3; 0 1 1];
>> A & B
        ans:
        1   1   0
        0   1   1
```

Operators Logical	
Operator	**Name**
&	AND
|	OR
~	NOT

Matlab Flow control structures

1.if, else, and elseif

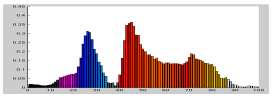
Syntax:

a) Single

```
if  expression
    statements
end
```

b) Multiple

```
if      expression1
        statements1
elsif   expression2
        statements2
else
        statements3
end
```



2. for

- executes a **group** of statements a fixed number of times:

```
for index = start: increment: end
     statements
end
```

Nested for

```
for index1 = start1: increment1: end
     statements1
     for index2 = start2: increments2:end
     statements2
     end
     additional loop1 statements
end
```

3. while

- Executes a group of statements an indefinite number of times, based on a specified logical condition.

```
while expression
     statements
end
```

Nested while

```
while expression1
     statement1
     while expression2
          statements2
     end
     additional loop1 statements
end
```

4. Break

- Terminates execution of a **for** or **while** loop.

5. Continue

- Passes control to the next iteration of a for or while loop, skipping any remaining statements in the body of the loop.

6. switch

- **switch**, together with **case** and **otherwise**, executes different groups of statements, depending on a specified value or string.

```
Switch   switch-expression
     case     case-expression
              statement(s)
     case     {case-expressoin1, case-expression2,...}
              statement(s)
    otherwise
              statement(s)
end
```

7. return:

- Causes execution to return to the invoking function.

8. try.. catch

- Changes flow control if an **error** is detected during execution.