Lect 4

Introduction to M-function Programming

M-files : 1- script : simply execute a series of Matlab statements.

2- functions : accept arguments and can produce outputs.

function m-files components :

1. function definition line 2- H1 line
3. Help text 4. function body 5. Comments

function definition Form :

function [output] = name (input)

example :

function [s p] = sumprod (f , g)

- function : Keyword
- [s p] : the sum image, the product imag
- sumprod : function to Compute the sum and product
- (f , g) : input (2 images)

function Name up to 63 long, more than → ignored

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);
```

- H1 line :

Single Comment line that follows the definition line (no blank lines) between them

```
% SUMPROD Computes the sum and products
  of two images.
```

H1 line appears when user types

```
>> help function_name
```

H1 Provides information about the M-file.

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

- function body: Matlab code that performs Computation.

% - 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 $\Rightarrow$ c(i,j) = A(i,j) * B(i,j)

* addition and subtraction operations are the same for array and matrices.

.+ and .- are not used.

Matlab Example: $\Rightarrow$

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] $\Rightarrow$

* Using the Matlab text editor, write the following code:

```
function [P, Pmax, Pmin, Pn] = improd(f, g)
% IMPROD Computes the product 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)
```

$$P = \begin{bmatrix} 1 & 4 \\ 6 & 4 \end{bmatrix} \qquad Pmax = 6 \qquad Pmin = 1$$

$$Pn = \begin{bmatrix} 0 & 0.6000 \\ 1.000 & 0.6000 \end{bmatrix}$$

```
>> help improd
:
```

Relational operators:

Example:

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

both A and B must be the same size

operator	Name
<	less than
<=	less than or equal
>	Greater than
>=	Greater than or equal
==	Equal to
~=	Not Equal to

Relational operator

operation A == B produces a logical array of the same dimension as A and B with 1s in locations where the cooresponding elements of A and B match, and 0s elsewhere.

```
>> A >= B    % produces a logical array with
ans          % 1s where the elements of A
     1     1     0    % greater than or equal to the
     1     1     1    % cooresponding elements of B and 0s elsewhere
     1     1     1
```

* 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
```

operator	Name
&	AND
|	OR
~	NOT

Logical operators

MatLab Flow Control Structures:

1. if, else, and elseif:

Syntax:

(a) if expression
statements
end

(b) if expression1
statements1
elsif expression2
statements2
else
statements3
end.

(2) for: executes a group of statements a fixed number of times:

(a) for index = start : increment : end
statements
end.

(b) Nested for:

for index1 = start1 : increment1 : end
statements1
for index2 = start2 : increment2 : 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.

a) while expression
statements
end

b) while expression1
statement1
while expression2
statements
end
additional loop1 stat.
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_expression1, 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.
   detected

Examples:

1) write a function that computes the average intensity of an image.

Solution: (Hints)

1- 2-D array f can be converted to a column vector v, by letting v = f(:). (the function will be able to work with both vector and image inputs)

2- The program should produce an error if the input is not a one- or two-dimensional array.

3- to return the size of the longest dimension of an array, we use length(A).

4- Another way to obtain the number of elements in an array directly is to use function numel, with syntax: n = numel(A).

Solution: (code): ⇒

```
function av = average(A)
% AVERAGE Computes the average value of an array
% AV = AVERAGE(A) Computes the Average value
% of input array, A, which must be a 1-D or
% 2-D array
% Check the validity of the input
    if ndims(A) > 2
        error('The dimensions of the input cannot..
              exceed 2.')
    end
% Compute the Average
    av = sum(A(:)) / length(A(:));
  % or
  % av = sum(A(:)) / numel(A);
```

% ndims function returns the number of ~~dimensions~~, dimensions.

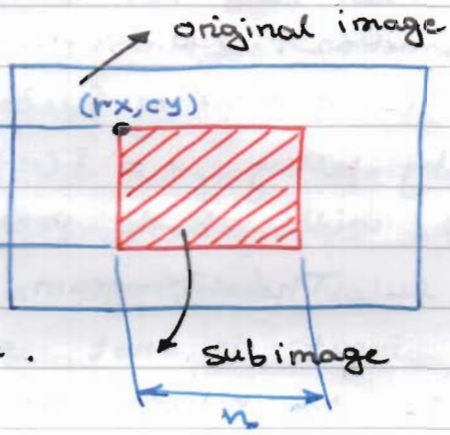
② write an M-function (based on for loop) to extract a rectangular subimage from an image.

Solution :

The inputs to the function :

* - original image
* - the size (number of rows and columns) of the subimage)
* - Coordinates of the top left corner of the extracted image.

* Keep in mind the image origin in Matlab is at (1,1).



Code :

```
function s = subim(f, m, n, rx, cy)
% SUBIM Extracts a subimage, s, from a
% given image, f.
% the subimage is of the size m-by-n. and
% the coordinates of its top, left corner are
% (rx, cy).
    s = zero(m, n); % preallocating matrix.
    rowhigh = rx + m - 1;
    colhigh = cy + n - 1;
    xcount = 0;
    for r = rx : rowhigh
        xcount = xcount + 1;
        ycount = 0;
        for c = cy : colhigh
            ycount = ycount + 1;
            s(xcount, ycount) = f(r, c);
        end
    end
```

(3) Color planes:

The green and red color plane of image rgbimage.jp are swapped:

```
f = imread('rgbimage.jpg');
red = f(:, :, 1);
g(:, :, 1) = f(:, :, 2);
g(:, :, 2) = red;
g(:, :, 3) = f(:, :, 3);
imshow(g);
```

(4) individual pixel Processing:

The intensity of the red color channel of rgbimage.jp

(4.1) is divided by 2. (low-level processing).

```
f = imread('rgbImage.jpg');
[M N d] = size(f);
g = uint8(zeros(M, N, 3));
for x = 1:M
    for y = 1:N
    g(x, y, 1) = f(x, y, 1)/2;
    g(x, y, 2) = f(x, y, 2);
    g(x, y, 3) = f(x, y, 3);
    end;
end;
imshow(g);
```

(4.2.) The intensity of the red color channel of rgbimage.jpg is divided by 2 using high-level processing (array notation - vectorization)

```
f = imread('rgbImage.JPG');
g = f;
g(:, :, 1) = g(:, :, 1)/2;
imshow(g);
```

Comments for examples (4.1 & 4.2) : ⇒

Matlab is a programming language specifically designed for array operations; we can whenever possible to increase the computational speed (code optimization) using Vectorizing Loops & preallocating arrays)

1- Vectorizing simply means converting for and while loops to equivalent vector or matrix operations The Vectorized code (example 4.2) runs on the order of 30 times faster than the implementation based on for loops.

2- Preallocating Arrays: to improve code execution time, preallocate the size of the array used in a program.

```
>> f = zeros(1024); % reduce memory fragmentation
>> g = zeros(1024);
>> img = uint8(zeros(512,1024) % Create a black image with width 1024 and height 512 of type uint8.
>> img = uint8(255 * ones(512,1024)) % white image
>> img = double(ones(512,1024)) % white image (double).
```

Example (5) : Vectorization of programs for extracting such regions (example 2).

```
rowhigh = rx + m - 1;
colhigh = cy + n - 1;
s = f(rx:rowhigh, cy:colhigh);
```

where f is the image from which the region is to be extracted.

Example (6) : Conversion the color image rgbimage.jpg to a grayscale image : (using the method of computing the mean of the three color channels and then store it in file grayimage.jpg;

6.1: Low-level processing using for loop:

```
f = imread ('rgbImage.JPG');
[M  N  d] =  size (f);
g =  uint8 (zeros (M, N));
for  x =  1 : M
     for  y = 1 : N
     g(x,y) = (f(x,y,1) + f(x,y,2) + f(x,y,3))/3 ;
     % The line above doesn't work.
     % overflow occurs, while processing uint8, because
     % the value range in the intermediate results
     % are limited to 255
     g(x,y) = ( f(x,y,1)/3 + f(x,y,2)/3 + f(x,y,3)/3) ;
end;
end;
imshow (g);
imwrite (g, 'grayimage.jpg', 'Quality', 100);
% The quality of the resulting image is set to
% 100 (no data loss);
```

6.2. High-level processing using Vectorization:

```
f = imread ('rgbimage.JPG');
g = uint8( mean(f, 3) ;
imshow (g)
imwrite (g, 'grayImage.jpg', 'Quality', 100);
```

Homework 1:

Write an M-function for grayscal image blurring Method (Compute the mean of a 3x3 pixel environment and by setting the resulting Center pixel to this value (mean). Using the Vectorization and for loop methods. the function also compares the Computational speed time between two methods (Use tic and toc Matlab function).

Interactive I/O :

To write interactive M-functions that display information and instructions to users and accept inputs from the keyboard

1- function disp is used to display information on the screen.

Syntax : disp (argument).

argument may be :

a) e. Array : >> A = [1 2 ; 3 4];
>> disp (A);

b) e. Variable contains string :

>> sc = 'Digital Image Processing';
>> disp (sc)

c) e. string :

>> disp ('This is another way to display text.')

2- function Input : is used for inputting data into an M-function.

syntax : a. t = input ('message')

This function outputs the words contained in message and waits for an input from the user, followed by a return, and stores the input in t.

The input can be

- single number
- character string (single quotes)
- vector (enclosed by square brackets)
- matrix (-11- 1- -11- -11- and rows separated by semicolons).

b. t = input ('message', 's') % outputs the contents of message and accepts a character string whose elements can be separated by commas.

n = str2num (t) is used to convert numbers of class double.