University of Diyala
Computer Science Department
Image Processing
3rd Class
Lecturer: Dr. Jumana Waleed Salih

- -

Image Processing

معالجة صور

3rd lecture

Converting between Data Classes

Converting between data classes is straightforward. The general syntax is

```
B = data_class_name(A)
```

Examples:

```
B = double(A)
D = uint8(C)
```

Converting between Image Classes and Types

Write the following array in the command window

```
>> v=[-0.5 0.5; 0.75 1.5]

v =

   -0.5000    0.5000
    0.7500    1.5000
```

Performing the conversion

```
>> w=im2uint8(v)
```

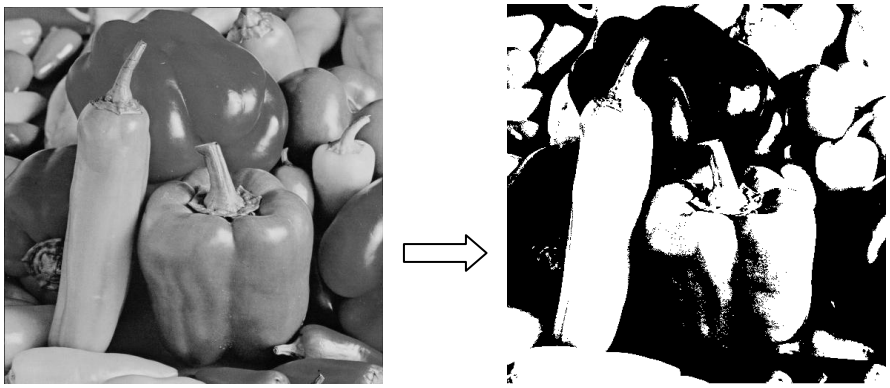
yields the result

```
w =

     0   128
   191   255
```

Other examples:

```
>> f=imread('E:\peppers.png');
>> z=im2bw(f);
>> imshow(z);
```



Name	Converts Input to:	Valid Input Image Data Classes
im2uint8	uint8	logical, uint8, uint16, and double
im2uint16	uint16	logical, uint8, uint16, and double
mat2gray	double (in range [0, 1])	double
im2double	double	logical, uint8, uint16, and double
im2bw	logical	uint8, uint16, and double

```
>> h = uint8([25 50; 128 200]);
```

Performing the conversion

```
>> g = im2double(h);
```

yields the result

```
g =

    0.0980    0.1961
    0.4706    0.7843
```

from which we infer that the conversion when the input is of class `uint8` is done simply by dividing each value of the input array by 255. If the input is of class `uint16` the division is by 65535.

Vector Indexing

```
>> v = [1 3 5 7 9]
v =
     1     3     5     7     9
>> v(2)
ans =
     3
```

A row vector is converted to a column vector using the *transpose operator* (.'):

```
>> w = v.'
w =
     1
     3
     5
     7
     9
```

To access *blocks* of elements, we use MATLAB's *colon* notation. For example, to access the first three elements of v we write

```
>> v(1:3)
ans =
     1     3     5
```

Similarly, we can access the second through the fourth elements

```
>> v(2:4)
ans =
     3     5     7
```

or all the elements from, say, the third through the last element:

```
>> v(3:end)
ans =
     5     7     9
```

where end signifies the last element in the vector. If v is a vector, writing

```
>> v(:)
```

produces a column vector, whereas writing

```
>> v(1:end)
```

produces a row vector.

Indexing is not restricted to contiguous elements. For example,

```
>> v(1:2:end)
ans =
    1    5    9
```

The notation `1:2:end` says to start at 1, count up by 2 and stop when the count reaches the last element. The steps can be negative:

```
>> v(end:-2:1)
ans =
    9    5    1
```

A vector can even be used as an index into another vector. For example, we can pick the first, fourth, and fifth elements of v using the command

```
>> v([1 4 5])
ans =
    1    7    9
```

Matrix Indexing

Matrices can be represented conveniently in MATLAB as a sequence of row vectors enclosed by square brackets and separated by semicolons. For example, typing

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

displays the 3×3 matrix

```
A =
     1    2    3
     4    5    6
     7    8    9
```

We select elements in a matrix just as we did for vectors, but now we need two indices: one to establish a row location and the other for the corresponding column. For example, to extract the element in the second row, third column, we write

```
>> A(2, 3)
ans =
    6
```

The colon operator is used in matrix indexing to select a two-dimensional block of elements out of a matrix. For example,

```
>> C3 = A(:, 3)
C3 =
     3
     6
     9
```

Here, use of the colon by itself is analogous to writing `A(1:3,3)`, which simply picks the third column of the matrix. Similarly, we extract the second row as follows:

```
>> R2 = A(2, :)
R2 =
     4     5     6
```

The following statement extracts the top two rows:

```
>> T2 = A(1:2, 1:3)
T2 =
     1     2     3
     4     5     6
```

To create a matrix `B` equal to `A` but with its last column set to 0s, we write

```
>> B = A;
>> B(:, 3) = 0
B =
     1     2     0
     4     5     0
     7     8     0

>> A(end, end)
ans =
     9
```

```
>> A(end, end - 2)
ans =
     7
>> A(2:end, end:-2:1)
ans =
     6     4
     9     7
```

Using vectors to index into a matrix provides a powerful approach for element selection. For example,

```
>> E = A([1 3], [2 3])
E =
     2     3
     8     9
```

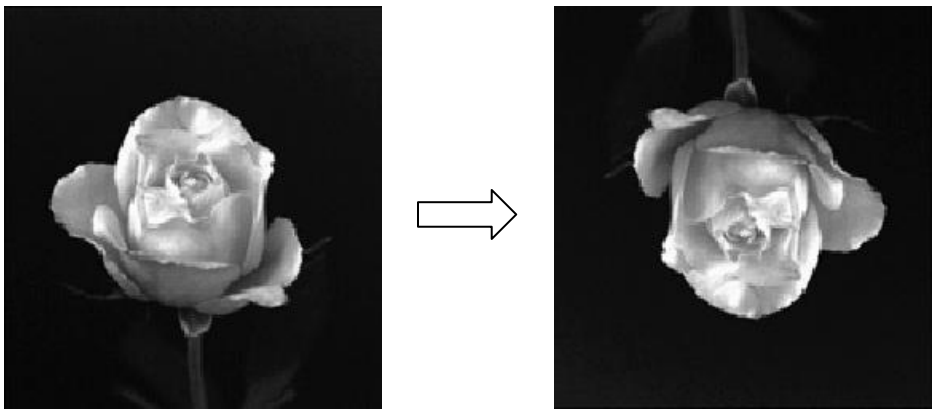
The notation `A([a b],[c d])` picks out the elements in `A` with coordinates (row a, column c), (row a, column d), (row b, column c), and (row b, column d). Thus, when we let `E = A([1 3], [2 3])` we are selecting the following elements in `A`: the element in row 1 column 2, the element in row 1 column 3, the element in row 3 column 2, and the element in row 3 column 3.

This use of the colon is helpful when, for example, we want to find the sum of all the elements of a matrix:

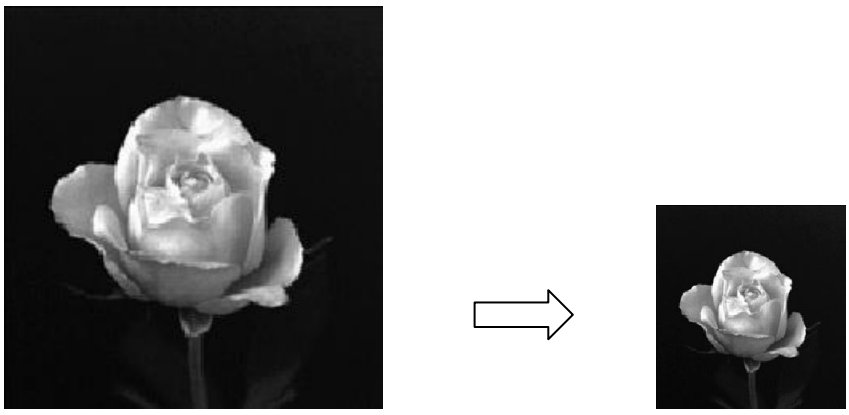
```
>> s = sum(A(:))
s =
    45
```

Examples:

```
>> f=imread('E:\rose.jpg');
>> fp=f(end:-1:1,:);
>> imshow(fp);
```



```
>> fc=f(1:2:end, 1:2:end);
>> imshow(fc);
```



Some Important Standard Arrays

- `zeros(M, N)` generates an `M` × `N` matrix of 0s of class `double`.
- `ones(M, N)` generates an `M` × `N` matrix of 1s of class `double`.
- `true(M, N)` generates an `M` × `N` `logical` matrix of 1s.
- `false(M, N)` generates an `M` × `N` `logical` matrix of 0s.
- `magic(M)` generates an `M` × `M` "magic square." This is a square array in which the sum along any row, column, or main diagonal, is the same. Magic squares are useful arrays for testing purposes because they are easy to generate and their numbers are integers.
- `rand(M, N)` generates an `M` × `N` matrix whose entries are uniformly distributed random numbers in the interval [0, 1].
- `randn(M, N)` generates an `M` × `N` matrix whose numbers are normally distributed (i.e., Gaussian) random numbers with mean 0 and variance 1.