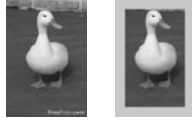


- Previous Lecture:
 - 2-d array examples



- Today's Lecture:
 - Complete matrix example from previous lecture
 - Image processing
 - Type `uint8`
 - Vectorized code for accessing subarrays
- Announcement:
 - Prelim 2 tonight 7:30-9pm in Kennedy Aud.

Available data

$C(i, j)$ is what it costs factory i to make product j

$Inv(i, j)$ is the inventory in factory i of product j

$PO(j)$ is the number of product j 's that the client wants

	10	36	22	15	62
C	12	35	20	12	66
	13	37	21	16	59

	38	5	99	34	42
Inv	82	19	83	12	42
	51	29	21	56	87

PO	1	0	12	29	5
------	---	---	----	----	---

Lecture 14

7

Who Can Fill the Order?

	38	5	99	34	42	Yes
Inv	82	19	83	12	42	No
	51	29	21	56	87	Yes

PO	1	0	12	29	5
------	---	---	----	----	---

Lecture 14

10

Wanted: A True/False Function



DO is "true" if factory i can fill the order.
DO is "false" if factory i cannot fill the order.

Lecture 14

11

Encapsulate...

```

function DO = iCanDo(i, Inv, PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false

nProd = length(PO);
DO = 1;
for j = 1:nProd
    DO = DO && ( Inv(i, j) >= PO(j) );
end
    
```

inv				
	8	9	5	4
PO	7	3	6	3

Lecture 14

12

Encapsulate...

```

function DO = iCanDo(i, Inv, PO)
% DO is true if factory i can fill
% the purchase order. Otherwise, false
nProd = length(PO);
j = 1;
while j <= nProd && Inv(i, j) >= PO(j)
    j = j+1;
end
DO = _____;
    
```

Lecture 14

13

Back To Finding the Cheapest

```

iBest = 0; minBill = inf;
for i=1:nFact
    iBill = iCost(i,C,PO);
    if iBill < minBill
        % Found an Improvement
        iBest = i; minBill = iBill;
    end
end

```

Don't bother with this unless there is sufficient inventory.

Lecture 14

16

Back To Finding the Cheapest

```

iBest = 0; minBill = inf;
for i=1:nFact
    if iCanDo(i,Inv,PO)
        iBill = iCost(i,C,PO);
        if iBill < minBill
            % Found an Improvement
            iBest = i; minBill = iBill;
        end
    end
end

```

See `Cheapest.m`
for alternative implementation

Lecture 14

17

Accessing a submatrix

M

2	-1	.5	0	-3
3	8	6	7	7
5	-3	8.5	9	10
52	81	.5	7	2

- **M** refers to the whole matrix
- **M(3,5)** refers to one component of **M**
- **M(2:3,3:5)** refers to a submatrix of **M**

row indices

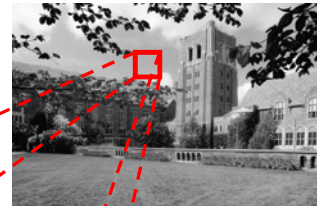
column indices

Lecture 15

20

A picture as a matrix

1458-by-2084



150	149	152	153	152	155
151	150	153	154	153	156
153	151	155	156	155	158
154	153	156	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162

Lecture 15

23

Images can be encoded in different ways

- Common formats include
 - JPEG: Joint Photographic Experts Group
 - GIF: Graphics Interchange Format
- Data are compressed
- We will work with jpeg files:
 - **imread**: read a .jpg file and convert it to a "normal numeric" array that we can work with
 - **imwrite**: write an array into a .jpg file (compressed data)

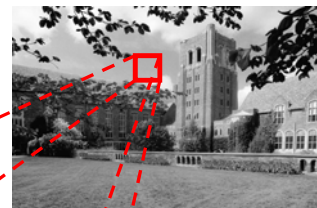
Lecture 15

24

Grayness: a value in [0..255]

0 = black
255 = white

These are *integer* values
Type: **uint8**

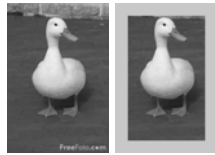


150	149	152	153	152	155
151	150	153	154	153	156
153	151	155	156	155	158
154	153	156	157	156	159
156	154	158	159	158	161
157	156	159	160	159	162

Lecture 15

25

Let's put a picture in a frame



Things to do:

1. Read `bwduck.jpg` from memory and convert it into an array
2. Show the original picture
3. Assign a gray value (frame color) to the "edge pixels"
4. Show the manipulated picture

Lecture 15

26

Reading a jpeg file and displaying the image

```
% Read jpeg image and convert to
% an array P
P = imread('bwduck.jpg');

% Show the data in array P as
% an image
imshow(P)
```

Lecture 15

27

```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200; % light gray
[nr,nc]= size(P);
for r= 1:nr
    for c= 1:nc
        % At pixel (r,c)

    end
end
imshow(P)
```

Lecture 15

30

```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200; % light gray
[nr,nc]= size(P);
for r= 1:nr
    for c= 1:nc
        % At pixel (r,c)
        if r<=width || r>nr-width || ...
            c<=width || c>nc-width
            P(r,c)= frameColor;
        end
    end
end
imshow(P)
```

Things to consider...

1. What is the type of the values in P?
2. Can we be more efficient?

Lecture 15

31

```
% Frame a grayscale picture

P= imread('bwduck.jpg');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 200; % light gray
[nr,nc]= size(P);
for r= 1:nr
    for c= 1:nc
        % At pixel (r,c)
        if r<=width || r>nr-width || ...
            c<=width || c>nc-width
            P(r,c)= frameColor;
        end
    end
end
imshow(P)
```

`P(r,c)= uint8(frameColor);`

P has type `uint8`: cannot change just one bin, `P(r,c)`, to type `double`. So Matlab did the type conversion to `uint8` for us behind the scenes.

Things to consider...

1. What is the type of the values in P?
2. Can we be more efficient?

See `pictureFrame*.m`

Lecture 15

32

A color picture is made up of **RGB** matrices → 3-d array



116	116	117	117	116	115	116	115	112	113
114	113	111	109	113	111	113	115	112	113
116	116	111	111	112	112	111	111	112	112
116	117	116	114	112	115	113	112	115	114
115	112	112	112	112	110	111	111	116	116
115	115	115	115	113	111	111	113	116	114
112	112	116	117	113	112	112	113	114	113
115	116	118	118	113	112	112	113	114	114
116	116	117	117	114	114	112	112	114	115

153	150	150	150	150	151	152	153	150	151
153	150	149	147	153	151	151	152	150	151
154	150	151	150	152	150	150	149	150	150
150	156	150	152	150	150	151	150	153	153
151	150	150	150	150	149	149	151	152	151
153	153	153	153	151	149	149	151	152	150
150	151	152	153	151	150	150	151	152	151
153	156	156	156	151	150	150	151	152	152
154	154	153	153	149	149	150	150	152	153

212	212	212	212	216	213	213	215	216	213
212	211	211	209	212	213	214	216	213	213
213	212	210	209	212	214	213	212	213	212
214	213	214	214	213	216	214	213	213	212
213	212	212	212	212	210	211	213	214	211
213	213	216	216	213	211	211	213	212	210
212	212	214	213	213	212	212	213	214	213
213	213	216	216	213	212	212	213	214	213
216	216	216	213	212	212	213	214	214	214
216	216	215	215	213	213	213	213	214	215

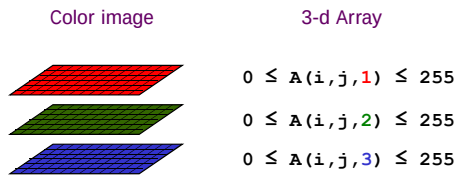
E.g., color image data is stored in a 3-d array A:

$$\begin{aligned}
 0 &\leq A(i, j, 1) \leq 255 \\
 0 &\leq A(i, j, 2) \leq 255 \\
 0 &\leq A(i, j, 3) \leq 255
 \end{aligned}$$

Lecture 15

34

A color picture is made up of **RGB** matrices $\rightarrow$ 3-d array



Operations on images amount to operations on **matrices!**

Lecture 15

35

Example: Mirror Image



1. Read **LawSchool.jpg** from memory and convert it into an array.
2. Manipulate the Array.
3. Convert the array to a jpg file and write it to memory.

Lecture 15

36

Reading and writing jpg files

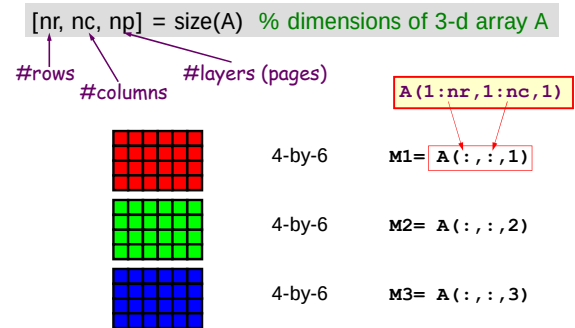
```
% Read jpg image and convert to
% a 3D array A
A = imread('LawSchool.jpg');

% Write 3D array B to memory as
% a jpg image
imwrite(B, 'LawSchoolMirror.jpg')
```

Lecture 15

37

A 3-d array as 3 matrices

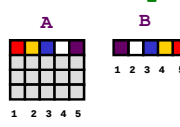


Lecture 15

38

%Store mirror image of A in array B

```
[nr, nc, np] = size(A);
for r = 1:nr
    for c = 1:nc
        B(r, c) = A(r, nc-c+1);
    end
end
```



Lecture 15

39

%Store mirror image of A in array B

```
[nr, nc, np] = size(A);
for r = 1:nr
    for c = 1:nc
        for p = 1:np
            B(r, c, p) = A(r, nc-c+1, p);
        end
    end
end
```

Lecture 15

40