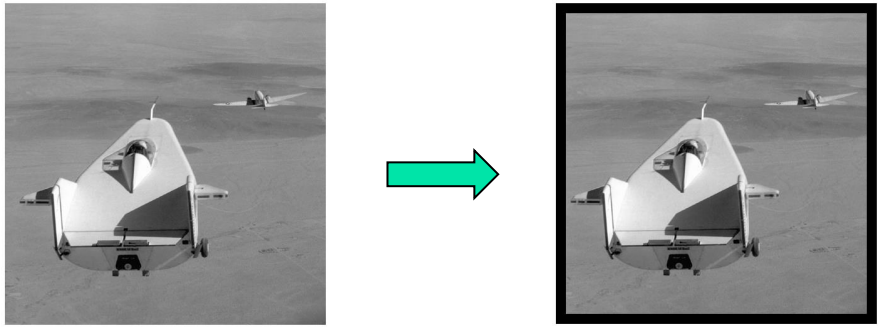
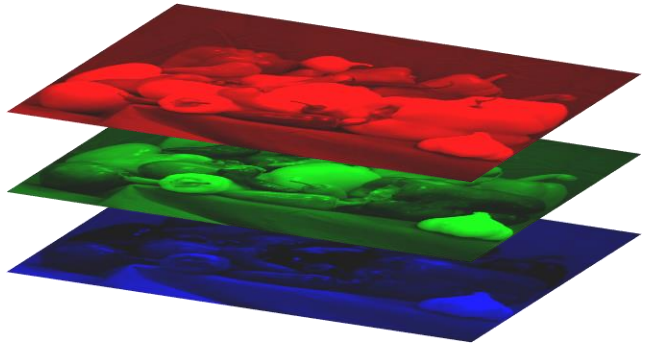
- Previous Lecture:
  - 2-d array examples



- Today's Lecture:
  - Complete matrix example from previous lecture
  - Image processing
    - Type `uint8`
    - Vectorized code for accessing subarrays

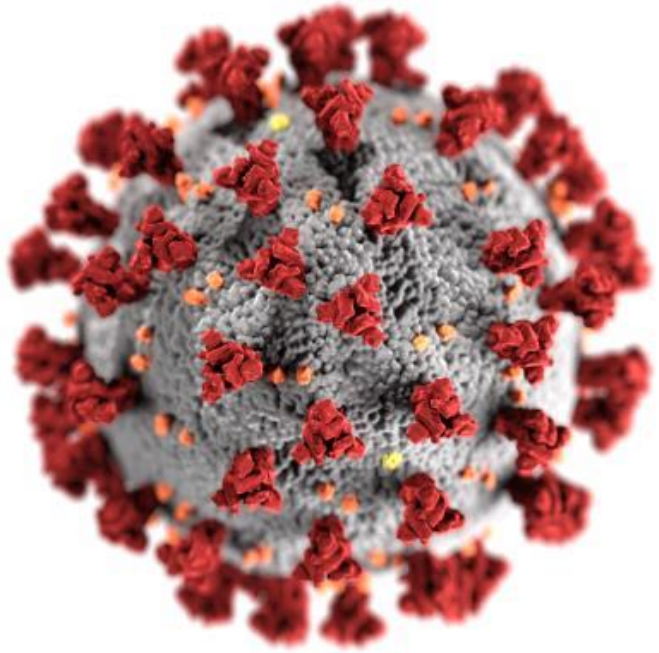


- Announcements:
  - P4 to be posted after today's lectures
  - Graded Prelim 1 will be available on *Gradescope* next week
  - Read §12.4 of Insight—learn about arithmetic in type `uint8`

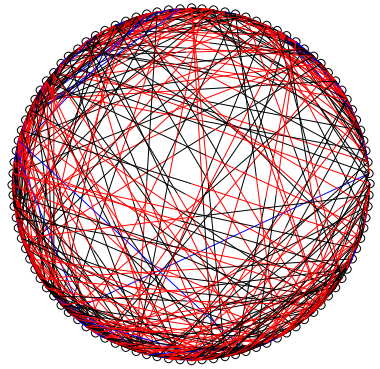
No curtain today –
spread out as much as you want

Transition to virtual instruction

- *Gradual transition: in-person course activities continue for now*
- Sign up for **Piazza**, participate in discussions
- Prepare to watch video lectures
- Expect to see more **Canvas**
- Explore videoconferencing options for discussion, office hours
- Exams TBD (online or scanned)



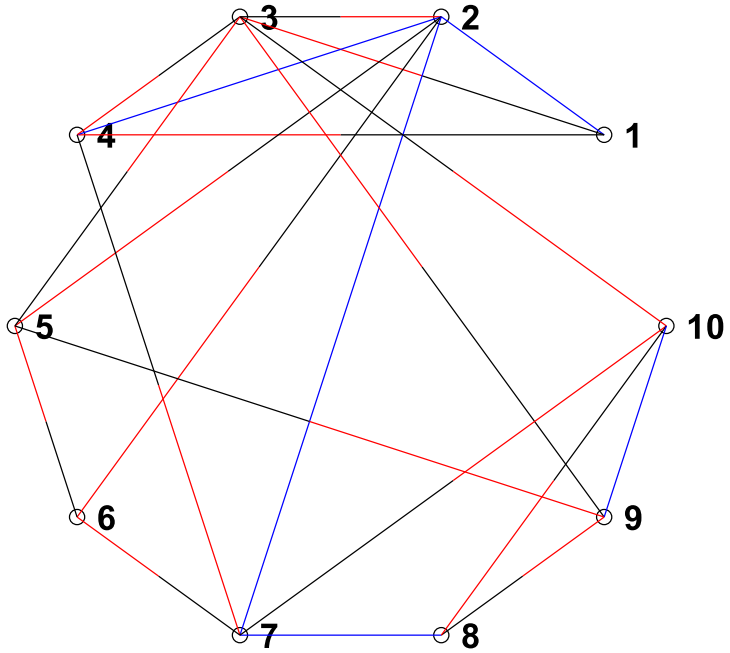
Matrix example: Random Web



- N web pages can be represented by an N-by-N Link Array A.
- A(i,j) is 1 if there is a link on webpage j to webpage i

0	0	1	0	1	0	0
1	0	0	1	1	1	0
0	1	0	1	1	1	1
1	0	1	1	0	1	0
0	0	1	1	0	1	1
0	0	1	0	1	0	1
0	1	1	0	1	1	0

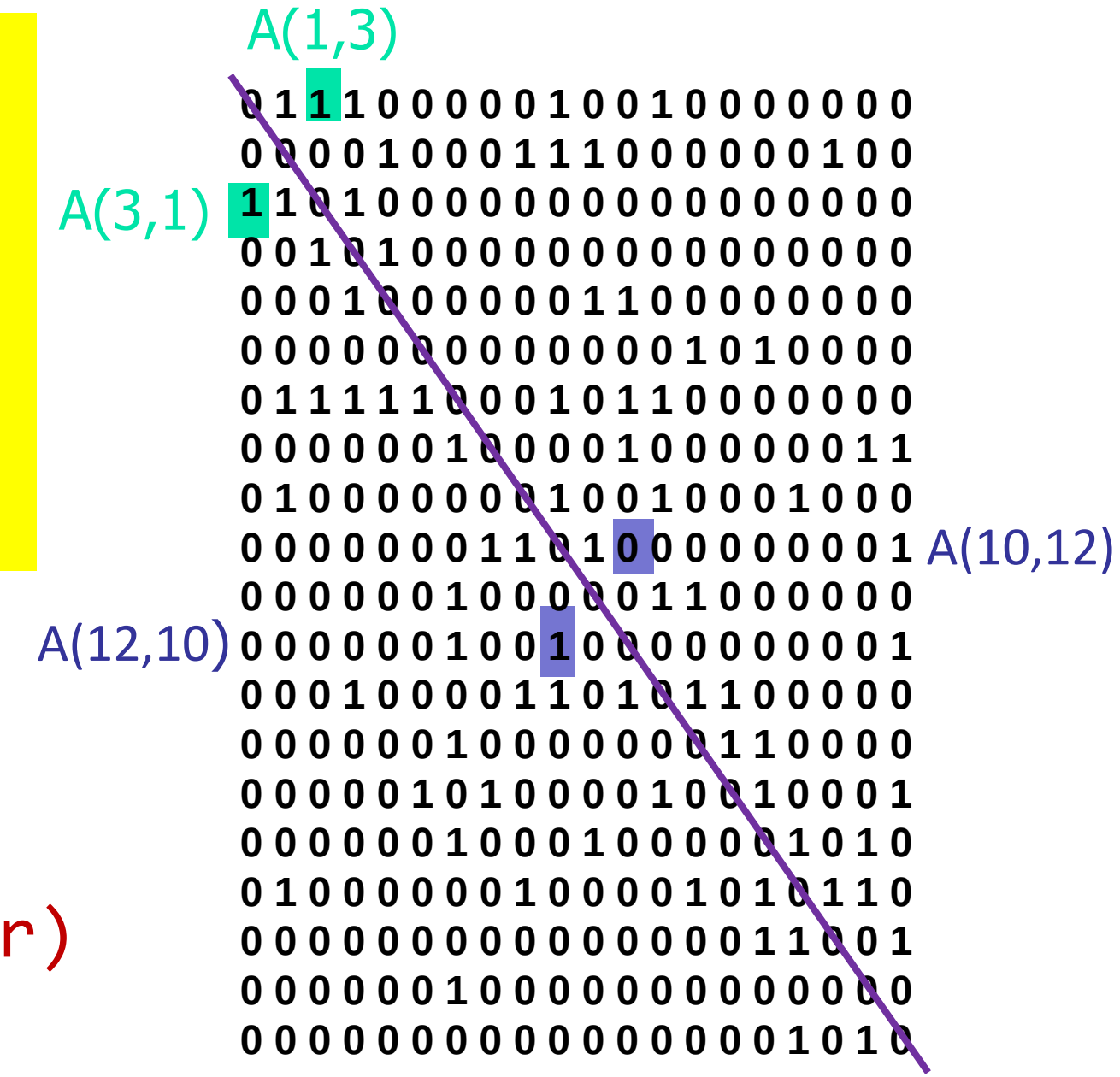
Represent the connectivity of the web pages graphically



	1	2	3	4	5	6	7	8	9	10
1	0	**1**	0	0	0	0	0	0	0	0
2	**1**	0	1	**1**	0	0	**1**	0	0	0
3	1	0	0	0	1	0	0	0	1	0
4	1	**1**	1	0	0	0	0	0	0	0
5	0	1	0	0	0	1	0	0	0	0
6	0	1	0	0	0	0	1	0	0	0
7	0	**1**	0	1	0	0	0	**1**	0	0
8	0	0	0	0	0	0	**1**	0	0	1
9	0	0	0	0	1	0	0	1	0	**1**
10	0	0	1	0	0	0	1	0	**1**	0

Web pages arranged in a circle. Bidirectional links are blue. Unidirectional link is black as it leaves page j, red when it arrives at page i.

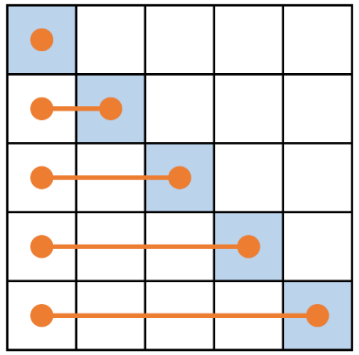
Take any element, its "*transpose element*" is the element at the transposed indices.




```
A'(r,c) == A(c,r)
```

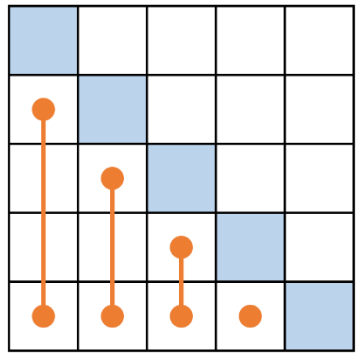
Triangular traversal

```
[nr, nc] = size(M);
for A = B:C
    for D = E:F
        disp(M(r,c))
    end
end
```



Case 1

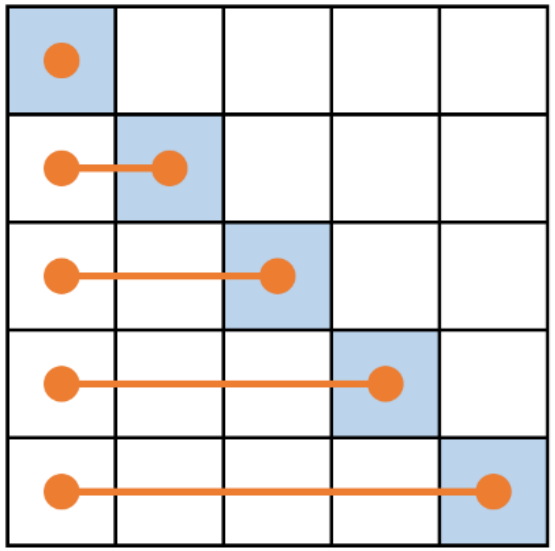
Case 2



What should be A, B, …, F in order to traverse the “triangular part” of a square matrix row-wise as in Case 1? How about traversing column-wise as in Case 2?

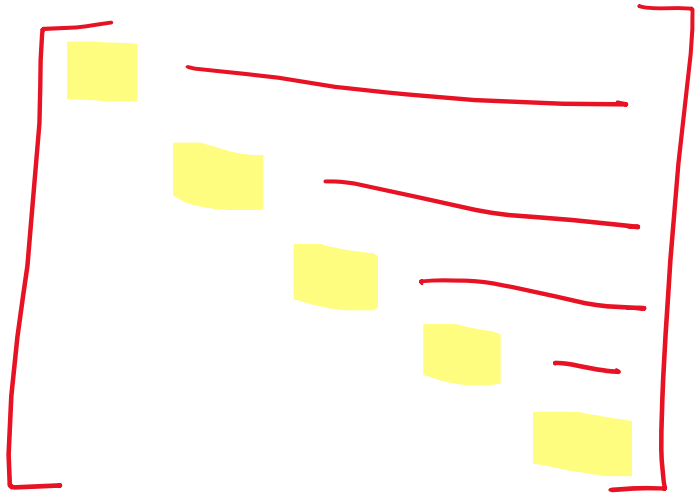
Row-major, lower-triangle

```
[nr, nc] = size(M);
for r = 1:nr                % Do something in every row
    for c = 1:r             % Start left, stop when row=col
        disp(M(r,c))
    end
end
```



What's different about this version?

```
[nr, nc] = size(M);
for r = 1:nr
    for c = (r+1):nc
        disp(M(r,c))
    end
end
```

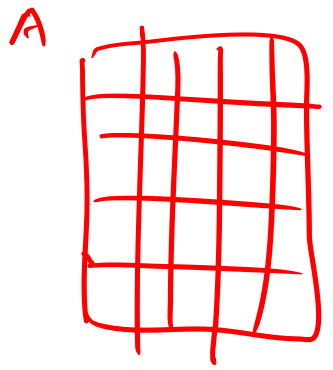


A: Nothing (identical to previous version)

B: Same as before, but without diagonal

C: Iterates in *column*-major order

D: Displays exactly those elements that were *not* displayed before




```
for i = 1:n
    for j = 1:n
        if A(i,j) == 1 && A(j,i) == 1
            % Blue

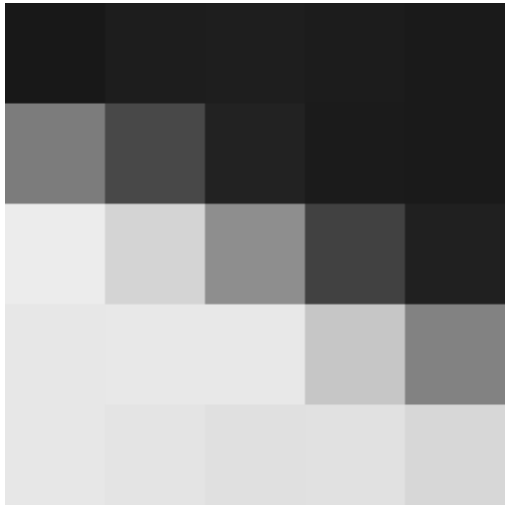
        elseif A(i,j) == 1
            % Black - Red
            % j -> mid   mid -> i

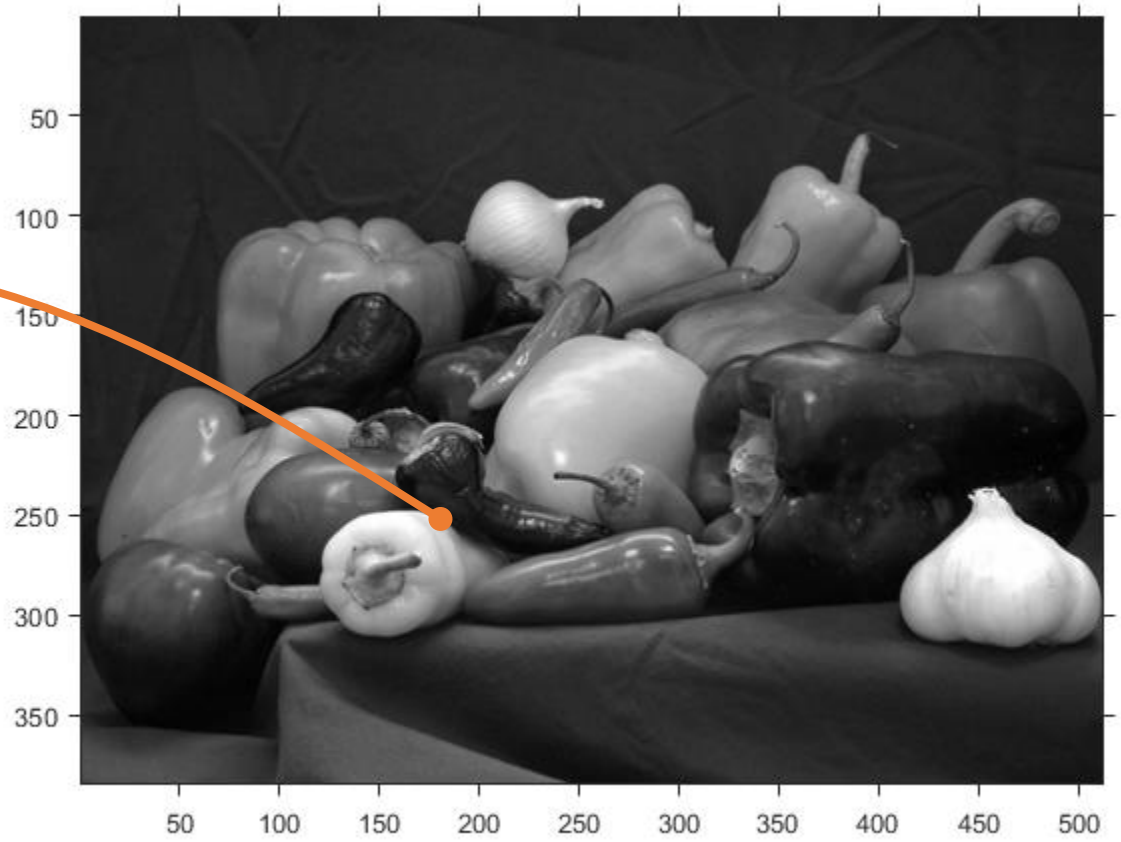
        end
    end
end
```

Somewhat inefficient: each blue line gets drawn twice.
See **`ShowRandomLinks.m`**

New topic: Image processing

Pictures as matrices





24	29	30	28	26
124	72	34	27	26
236	212	142	65	32
231	232	232	198	130
231	228	224	225	215

“512 x 384” image ⇒ 384 x 512 array

Image files & raster data

File formats

- **JPEG**: Photographs, lossy
- **PNG**: Graphics, lossless
- TIFF: Technical

Others

- WebP, GIF, DNG, OpenEXR, …

Properties

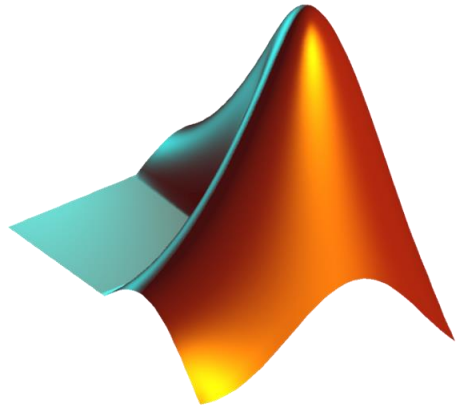
- Channels
  - **Greyscale**, **RGB**(A), YCbCr
- Bit depth, range
  - **8-bit**, 10-bit, HDR
- Color space, “gamma”
  - sRGB, DCI-P3, raw
- Subsampling
  - 4:4:4, 4:2:0

MATLAB image features

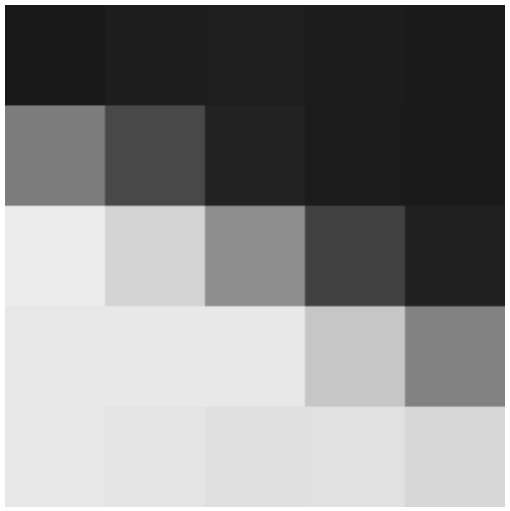
- ```
  % Read image file into matrix
  mat = imread('filename')
  ```

- ```
  % Plot matrix as image
  imshow(mat)
  ```

- ```
  % Write matrix to image file
  imwrite(mat, 'filename')
  ```


  ```

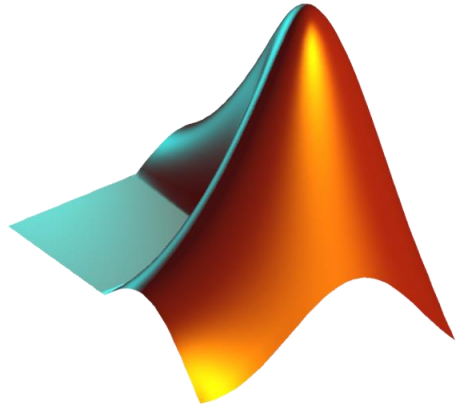
Greyness: a value in [0..255]



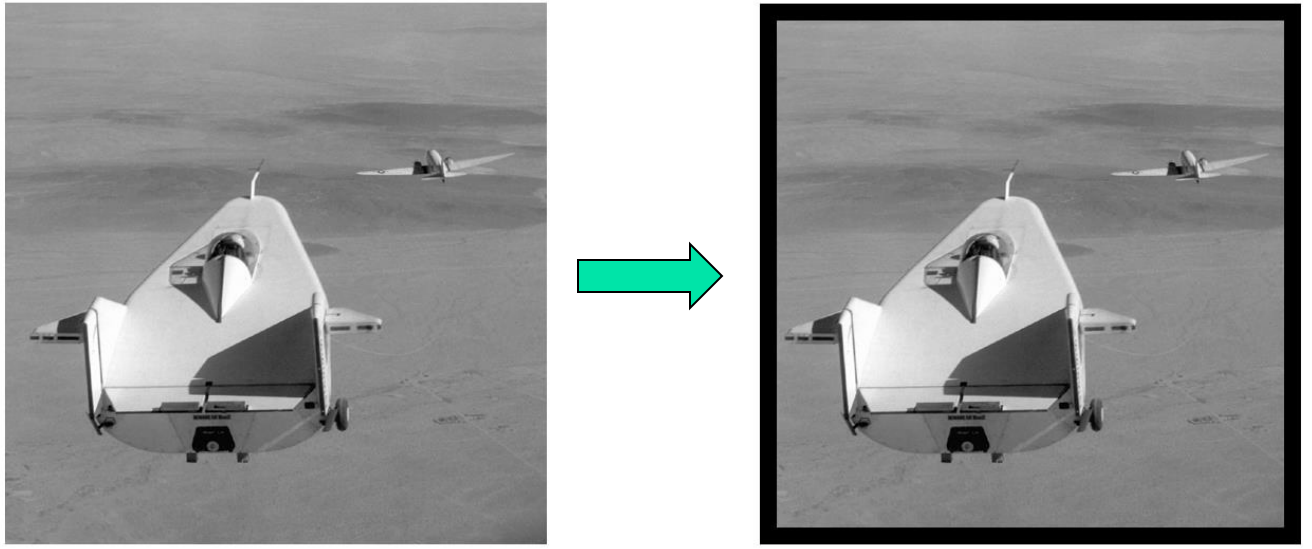
```
 24   29   30   28   26
124   72   34   27   26
236  212  142   65   32
231  232  232  198  130
231  228  224  225  215
```

New type: `uint8`

- *Integer* value between 0 and 255
  - 0=dark, 255=bright
- Can see types of *variables* in Workspace panel



Let's put a picture in a frame



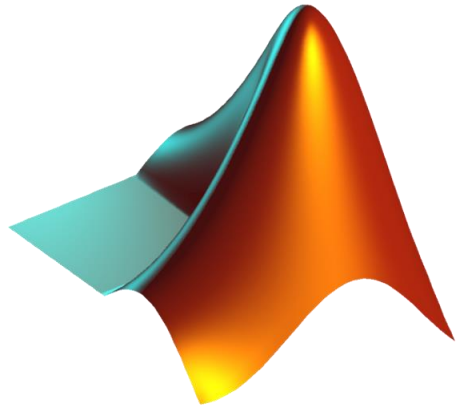
Things to do:

1. Read `liftingbody.png` from disk and convert it into an array
2. Show the original picture
3. Assign a black value (frame color) to the "edge pixels"
4. Show the manipulated picture

Reading a PNG file and displaying the image

```
% Read jpg image and convert to
% a type uint8 array P
   P = imread('liftingbody.png');

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



```
% Frame a grayscale picture

P= imread('liftingbody.png');
imshow(P)

% Change the ”frame” color




imshow(P)
```

```
% Frame a grayscale picture

P= imread('liftingbody.png');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 20;  % dark gray









imshow(P)
```

```
% Frame a grayscale picture

P= imread('liftingbody.png');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 20;  % dark gray
[nr,nc]= size(P);
for r = 1:nr
  for c = 1:nc
    % At pixel (r,c)




  end
end
imshow(P)
```

```
% Frame a grayscale picture

P= imread('liftingbody.png');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 20;  % dark 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?

```
% Frame a grayscale picture

P= imread('liftingbody.png');
imshow(P)

% Change the "frame" color
width= 50;
frameColor= 20;   % dark 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`

Type conversions

```
P= imread('liftingbody.png');
% (all of) P has type uint8

framecolor= 20;
% framecolor has type double

P(r,c)= framecolor;
% RHS value is implicitly converted to type of LHS var

P(r,c)= uint8(framecolor);
% RHS value is explicitly converted to uint8
```

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

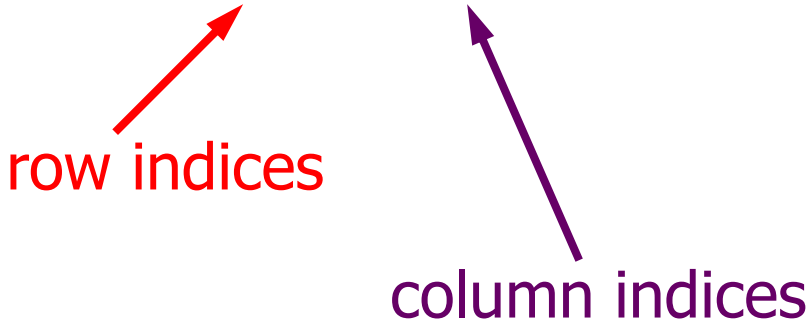
- `M` refers to the whole matrix
- `M(3,5)` refers to one component of `M`

Accessing a submatrix

`M`

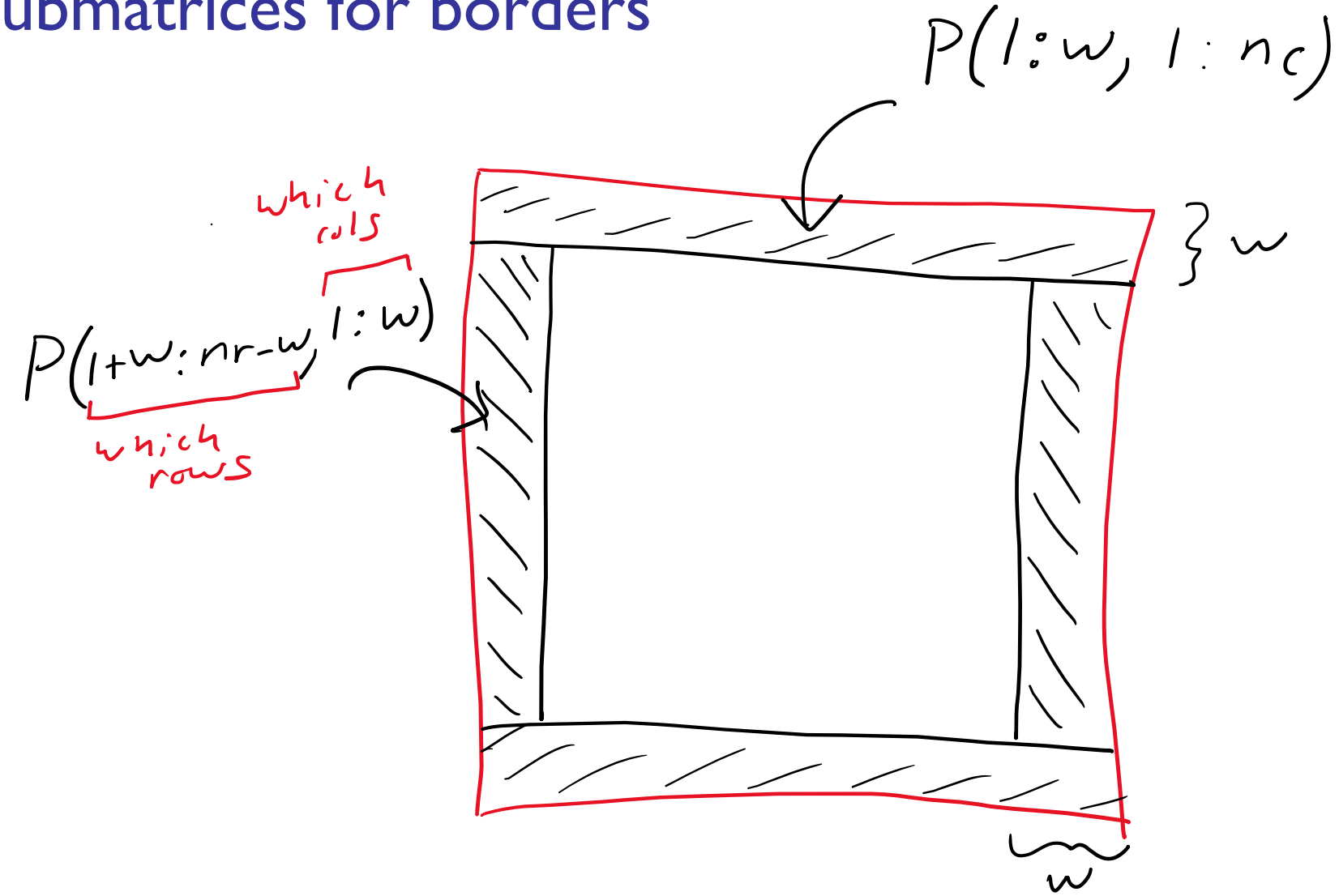
2	-1	.5	0	-3
3	8	6	7	7
5	-3	8.5	9	10
52	81	.5	7	2

- `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`

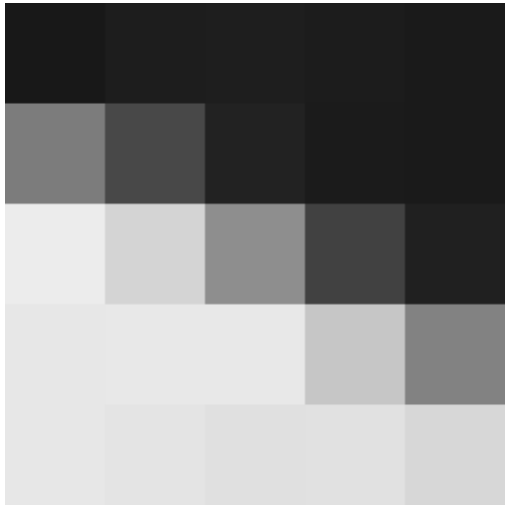


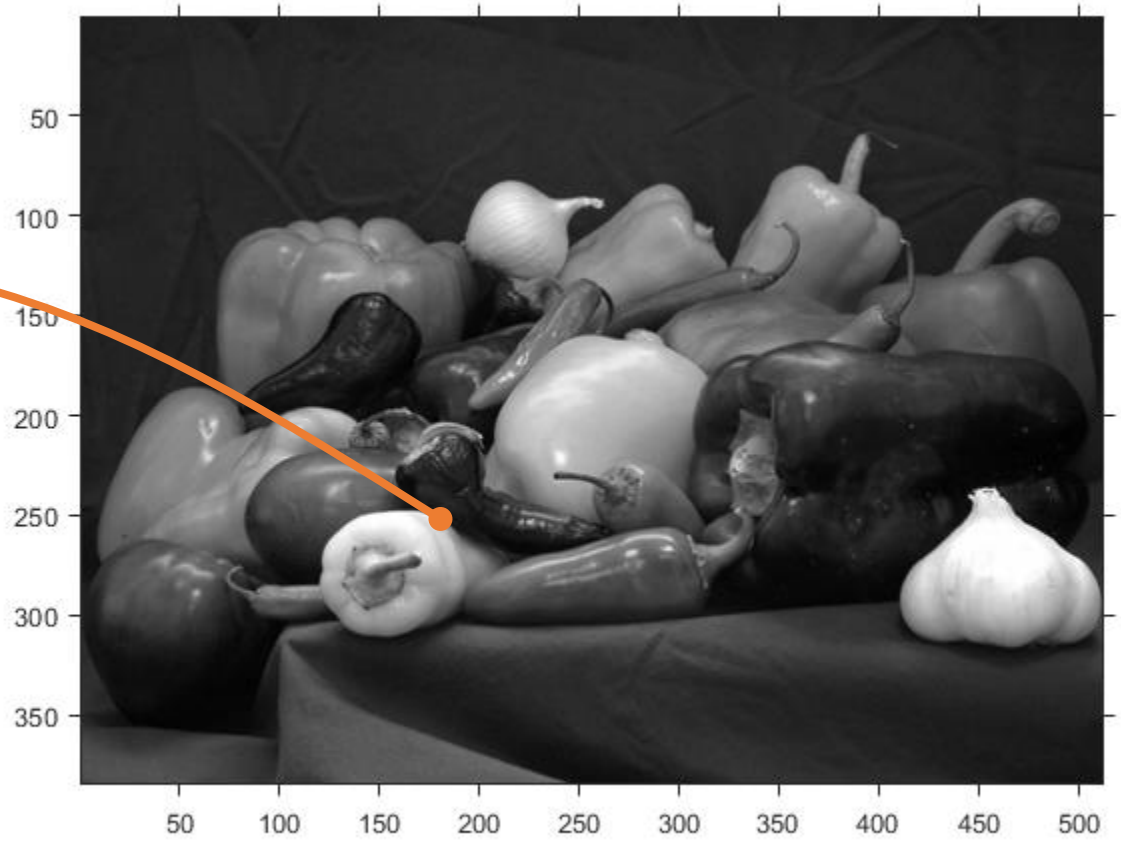
See `pictureFrameV.m`

Submatrices for borders



Pictures as matrices



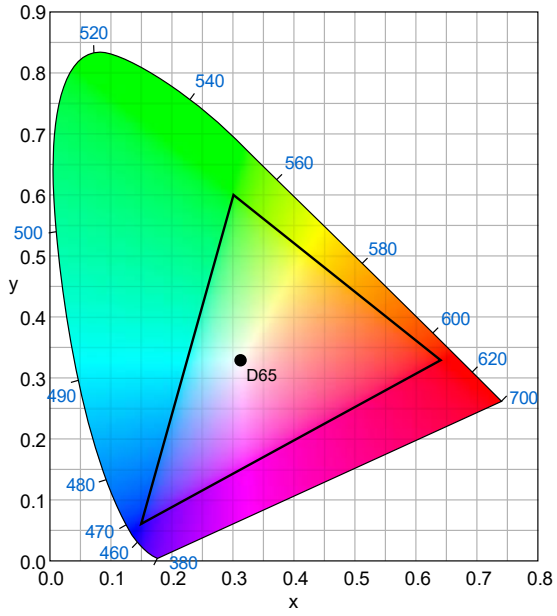


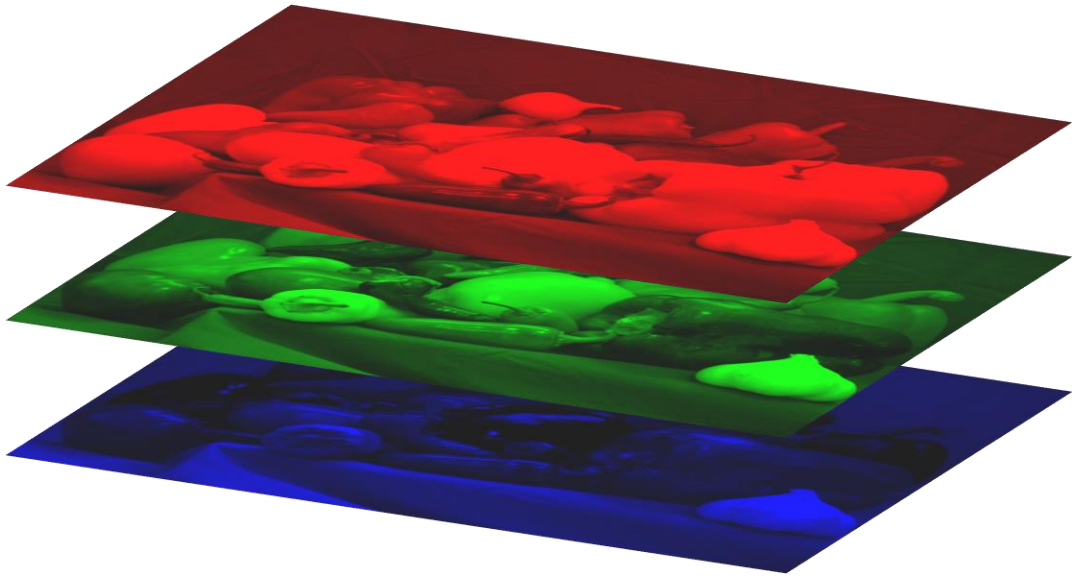
24	29	30	28	26
124	72	34	27	26
236	212	142	65	32
231	232	232	198	130
231	228	224	225	215

“512 x 384” image ⇒ 384 x 512 array

Color

- 3 different cone cells in eye means color can be represented by 3 numbers (channels)
  - Cameras, displays work with Red, Green, and Blue light: RGB
- Each channel (color) represented by its own matrix “plane”
- MATLAB: `pic(row, col, ch)`
  - `pic(:,:,1)`: Red
  - `pic(:,:,2)`: Green
  - `pic(:,:,3)`: Blue





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

114	114	112	112	114	111	114	115	112	113
114	113	111	109	113	111	113	115	112	113
115	114	112	111	111	112	112	111	112	112
116	117	116	114	112	115	113	112	115	114
113	112	112	112	112	110	111	113	116	115
115	115	115	115	113	111	111	113	116	114
112	113	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	153	150	150	154	151	152	153	150	151
153	152	149	147	153	151	151	153	150	151
154	153	151	150	151	152	150	149	150	150
155	156	155	152	152	155	151	150	153	153
151	150	150	150	150	148	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	154	154	154	151	150	150	151	152	152
154	154	153	153	149	149	150	150	152	153

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

$$0 \le A(i,j,1) \le 255$$

$$0 \le A(i,j,2) \le 255$$

$$0 \le A(i,j,3) \le 255$$