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:
- P4A has been posted; P4B will be posted by tomorrow
- Graded Prelim 1 will be available on Gradescope by Mon
- Read §12.4 of Insight—learn about arithmetic in type uint8
- As usual, office hours after lecture (4pm, Zoom)
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 $i$
Random web: $N=20$

$M(3, 2)$

$M(2, 3)$
Represent the connectivity of the web pages graphically

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.
% Connectivity given link array A
for i= 1:n
    for j= 1:n
        A(i,j) = 0;
    end
end
% Connectivity given link array A
for i = 1:n
    for j = 1:n
        if A(i,j) == 1 && A(j,i) == 1
            % blue line
        elseif A(i,j) == 1
            % red-black line
        end
    end
end

% Somewhat inefficient: each blue line gets drawn twice.
See ShowRandomLinks2.m
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
```

What should be $A, B, \ldots, F$ 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

\[
[nr, nc] = \text{size}(M);
\]
\[
\text{for } r = 1:nr \quad \% \text{Do something in every row}
\]
\[
\text{for } c = 1:r \quad \% \text{Start left, stop when row=col}
\]
\[
\text{disp}(M(r,c))
\]
\end
What’s different about this version?

\[[\text{nr}, \text{nc}] = \text{size}(\text{M});\]
\[\text{for } r = 1:\text{nr} \]
\[\quad \text{for } c = (r+1):\text{nc} \]
\[\quad \text{disp}(\text{M}(r,c)) \]
\[\text{end} \]
\[\text{end} \]
See ShowRandomLinks2.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 \([0..255]\)

New type: \texttt{uint8}

- \textit{Integer} value between 0 and 255
  - 0=dark, 255=bright
- Can see types of \textit{variables} in Workspace panel

\begin{array}{cccccc}
  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 \\
\end{array}
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
end
imshow(P)

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-1</td>
<td>.5</td>
<td>0</td>
<td>-3</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-3</td>
<td>8.5</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>52</td>
<td>81</td>
<td>.5</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>
Accessing a submatrix

- \( M \) 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

\[ P(1:w, 1:nc) \]

\[ P(1+w: nr-w, 1:w) \]
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 $\rightarrow$ 3-d array

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

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

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

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