Previous class:
- Play with sound files
- Practice working with vectors

Now:
- Play with image files
- 2-dimensional array—matrix

2-d array: \textit{matrix}

- An array is a \textit{named} collection of \textit{like} data organized into rows and columns
- A 2-d array is a table, called a \textit{matrix}
- Two \textit{indices} identify the position of a value in a matrix, e.g., \texttt{\textit{mat}(r,c)} refers to component in row \( r \), column \( c \) of matrix \texttt{mat}
- Array index starts at 1
- Rectangular: all rows have the same \# of columns

Creating a matrix

- \textit{Built-in functions:} \texttt{ones}, \texttt{zeros}, \texttt{rand}
  - E.g., \texttt{\textit{zeros}(2,3)} gives a 2-by-3 matrix of 0s
  - “Build” a matrix using square brackets, \([ \ ]\), but the dimension must match up:
    - \([x~y]\) puts \( y \) to the right of \( x \)
    - \([x; y]\) puts \( y \) below \( x \)
    - \([4~0~3; 5~1~9]\) creates the matrix
    - \([4~0~3; \text{\textit{ones}}(1,3)]\) gives
    - \([4~0~3; \text{\textit{ones}}(3,1)]\) doesn’t work

% What will \texttt{M} be?
\texttt{M = [\textit{ones}(1,3); 1:4]}

\begin{verbatim}
A
1 1 1 0
1 2 3 4
B
1 1 1
1 2 3
C  Error – \texttt{M} not created
\end{verbatim}

What will \texttt{A} be?

\begin{verbatim}
A= [1 1]
A= [A' \text{\textit{ones}}(2,1)]
A= [1 1 1 1; A A]
\end{verbatim}

\begin{itemize}
  \item \texttt{A} 3-by-4 matrix
  \item \texttt{B} 4-by-3 matrix
  \item \texttt{C} vector of length 12
  \item \texttt{D} Error
\end{itemize}
Working with a matrix:

```matlab
size and individual components

Given a matrix M

\[
\begin{bmatrix}
2 & -1.5 & 0 & -3 \\
3 & 8 & 6 & 7 \\
5 & -3 & 8.5 & 9 & 10 \\
52 & 81 & .5 & 7 & 2
\end{bmatrix}
\]

\[\text{[nr, nc]} = \text{size}(M) \quad \% \text{ nr is # of rows,} \]
\[\% \text{ nc is # of columns} \]

```M(2,4) = 1;
disp(M(3,1));
M(1,nc)= 4;
```

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)

Grayness: a value in \([0..255]\)

\[0 = \text{black} \quad 255 = \text{white} \]

These are integer values
Type: `uint8`

```
\begin{array}{cccccccc}
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 \\
\end{array}
```

Let’s put a picture in a frame

- Read a grayscale jpeg file into a matrix P
  \[P = \text{imread('filename'.jpg')};\]
- See the image represented by P
  \[\text{imshow}(P)\]
- Change the “edge pixels” into the frame color (grayscale) you want
  ...

Problem: produce a negative

- “Negative” is what we say, but all color values
  are positive numbers!
- Think in terms of the extremes, 0 and 255. Then
  the “negative” just means the opposite side.
- So 0 is the opposite of 255;
  \[1 \quad \ldots \quad 254;\]
  \[5 \quad \ldots \quad 250;\]
  \[30 \quad \ldots \quad 225;\]
  \[x \quad \ldots \quad 255-x\]
A color picture is made up of RGB matrices

<table>
<thead>
<tr>
<th>Color image</th>
<th>3-d Array</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="RGB Matrix" /></td>
<td><img src="image" alt="Color Matrix" /></td>
</tr>
</tbody>
</table>

Operations on images amount to operations on matrices—good way to practice matrix manipulation!

Your multi-media project

- Create a Matlab program that involves image and sound manipulation
- You get to
  - Make your own design
  - Set the level of difficulty
- Finish by 11:30am and submit in CMS

Example: Mirror Image

- ![LawSchool.jpg](image)
- ![LawSchoolMirror.jpg](image)

Solution Framework

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.
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')

% Make mirror image of A
[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

% Make mirror image of A -- the whole thing
A= imread('LawSchool.jpg');
[nr,nc,np]= size(A);
B= zeros(nr,nc,np);
B= uint8(B); % Type for image color values
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
image(B)  % Show 3-d array data as an image
imwrite(B,'LawSchoolMirror.jpg')

Vectorized code simplifies things...
Work with a whole column at a time

A = imread('LawSchool.jpg')
[nr,nc,np] = size(A);
for c= 1:nc
    B(:,c,1) = A(:,nc+1-c,1)
    B(:,c,2) = A(:,nc+1-c,2)
    B(:,c,3) = A(:,nc+1-c,3)
end
imwrite(B,'LawSchoolMirror.jpg')
Example: color $\rightarrow$ black and white

Can “average” the three color values to get one gray value.

Averaging the RGB values to get a gray value

$$R \rightarrow \frac{3R}{3} + \frac{59G}{3} + \frac{11B}{3}$$

Averaging the RGB values to get a gray value

$$M = 0.3R + 0.59G + 0.11B$$

Here are 2 ways to calculate the average. Are gray value matrices $g$ and $h$ the same given image data $A$?

$$g(r,c) = \frac{A(r,c,1)}{3} + \frac{A(r,c,2)}{3} + \frac{A(r,c,3)}{3};$$

$$h(r,c) = \frac{(A(r,c,1) + A(r,c,2) + A(r,c,3))}{3};$$

Turn the white duck yellow!

- The duck’s body and the image’s background show some contrast. However, neither the duck’s body nor the background has a uniform color.
- Are the RGB values different enough for us to write a “rule” in the program to tell between the duck and the background?
- Check out the RGB values!