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

\[
\begin{array}{ccccccc}
10 & 36 & 22 & 15 & 62 \\
12 & 35 & 20 & 12 & 66 \\
13 & 37 & 21 & 16 & 59 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
38 & 5 & 99 & 34 & 42 \\
82 & 19 & 83 & 12 & 42 \\
51 & 29 & 21 & 56 & 87 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
1 & 0 & 12 & 29 & 5 \\
\end{array}
\]

Who Can Fill the Order?

<table>
<thead>
<tr>
<th>Inv</th>
<th>Yes</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
</tr>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
</tr>
<tr>
<td>PO</td>
<td>1</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Wanted: A True/False Function

\[
\begin{align*}
&i \quad \text{CanDo} \\
&\text{Inv} \quad \text{DO} \\
&\text{PO} \\
\end{align*}
\]

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

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

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 = _______ ;
Back To Finding the Cheapest

```matlab
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.

Back To Finding the Cheapest

```matlab
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

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**

A picture as a matrix

```
1458-by-2084
```

Grayness: a value in [0..255]

```
0 = black
255 = white
```

These are integer values

Type: `uint8`

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)
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

Reading a jpeg file and displaying the image

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

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

% Frame a grayscale picture
```matlab
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
end
imshow(P)
```

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

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

```
0 ≤ A(i,j,1) ≤ 255
0 ≤ A(i,j,2) ≤ 255
0 ≤ A(i,j,3) ≤ 255
```
A color picture is made up of RGB matrices → 3-d array

<table>
<thead>
<tr>
<th>Color image</th>
<th>3-d Array</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Color Image" /></td>
<td><img src="image2.png" alt="3-d Array" /></td>
</tr>
</tbody>
</table>

Operations on images amount to operations on matrices!

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

<table>
<thead>
<tr>
<th>Color image</th>
<th>3-d Array</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Color Image" /></td>
<td><img src="image2.png" alt="3-d Array" /></td>
</tr>
</tbody>
</table>

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.

Reading and writing jpg files

```matlab
% 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')
```

A 3-d array as 3 matrices

```matlab
[nr, nc, np] = size(A) % dimensions of 3-d array A

% Store mirror image of A in array B

```matlab
for r = 1:nr
    for c = 1:nc
        B(r, c, p) = A(r, nc-c+1, p);
    end
end
```

% Store mirror image of A in array B

```matlab
[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
```