Previous Lecture:
- 2-d array examples

Today’s Lecture:
- Complete matrix example from previous lecture
- Image processing
  - Type \texttt{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 $j$.
- $\text{Inv}(i,j)$ is the inventory in factory $i$ of product $j$.
- $\text{PO}(j)$ is the number of product $j$'s that the client wants.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>36</td>
<td>22</td>
<td>15</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>35</td>
<td>20</td>
<td>12</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>37</td>
<td>21</td>
<td>16</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
function TheBill = iCost(i,C,PO)
% The cost when factory i fills the
% purchase order

nProd = length(PO);
TheBill = 0;
for j=1:nProd
    TheBill = TheBill + C(i,j)*PO(j);
end
Finding the Cheapest

\[
iBest = 0; \quad \text{minBill} = \text{inf}; \\
\text{for } i = 1:\text{nFact} \\
\quad \text{iBill} = \text{iCost}(i,C,PO); \\
\quad \text{if } \text{iBill} < \text{minBill} \\
\quad \quad \% \text{ Found an Improvement} \\
\quad \quad \text{iBest} = i; \quad \text{minBill} = \text{iBill}; \\
\quad \end \\
\end for \\
\text{end}
\]
## Who Can Fill the Order?

<table>
<thead>
<tr>
<th>Inv</th>
<th>38</th>
<th>5</th>
<th>99</th>
<th>34</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

- **PO**: 1
- **PO**: 0
- **PO**: 12
- **PO**: 29 (green)
- **PO**: 5

- **Inv**: Yes
- **Inv**: No
- **Inv**: Yes

- **PO**: Yes
- **PO**: No
- **PO**: Yes
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.
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
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 = ________;
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 = (j>nProd);
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

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

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

\[
\begin{array}{cccc}
2 & -1 & .5 & 0 & -3 \\
3 & 8 & 6 & 7 & 7 \\
5 & -3 & 8.5 & 9 & 10 \\
52 & 81 & .5 & 7 & 2 \\
\end{array}
\]
A picture as a matrix

1458-by-2084

<table>
<thead>
<tr>
<th>150</th>
<th>149</th>
<th>152</th>
<th>153</th>
<th>152</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>150</td>
<td>153</td>
<td>154</td>
<td>153</td>
<td>156</td>
</tr>
<tr>
<td>153</td>
<td>151</td>
<td>155</td>
<td>156</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>154</td>
<td>153</td>
<td>156</td>
<td>157</td>
<td>156</td>
<td>159</td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>158</td>
<td>159</td>
<td>158</td>
<td>161</td>
</tr>
<tr>
<td>157</td>
<td>156</td>
<td>159</td>
<td>160</td>
<td>159</td>
<td>162</td>
</tr>
</tbody>
</table>
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 = 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 |
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

% 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

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

% Change the "frame" color

imshow(P)
% Frame a grayscale picture

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

% Change the "frame" color
width = 50;
frameColor = 200; % light gray

imshow(P)
% 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
end
imshow(P)
% 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?
% 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?

See pictureFrame*.m