Previous Lecture:
- Vectorized code
- 2-d array—matrix

Today's Lecture:
- More examples on matrices
- Optional reading: contour plot (7.2, 7.3 in Insight)

Announcement:
- Fall Break next Mon & Tues: no lec, dis, office/consulting hrs. Attendance at 10/12 (W) discussion is optional, but do the exercise even if you don’t attend. Attend any of the 10/13 dis sections if you wish. Location is Hollister 464
- Optional review sessions: W7:30-9p, W5:30-7p (tentative). Location: Olin 255

Storing and using data in tables
A company has 3 factories that make 5 products with these costs:

<table>
<thead>
<tr>
<th></th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
</tbody>
</table>

What is the best way to fill a given purchase order?

Pattern for traversing a matrix M

```
[rn,nc] = size(M)
for r = 1:rn
    for c = 1:nc
        % Do something with M(r,c) ...
    end
end
```

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
- Generate a random link array and display the connectivity:
  - There is no link from a page to itself
  - If i ≠ j then A(i,j) = 1 with probability 1/(1+||i-j||)
  - There is more likely to be a link if i is close to j

```
function A = RandomLinks(n)
    % A is n-by-n matrix of 1s and 0s representing n webpages
    A = zeros(n,n);
    for i = 1:n
        for j = 1:n
            r = rand;
            if i ~= j & & r <= 1/(1 + abs(i-j))
                A(i,j) = 1;
            end
        end
    end
end
```

An event happens with probability p, 0<p<1

```
% Flip a fair coin
r = rand;
if r <= 5/6
    disp('heads')
else
    disp('tails')
end
```

```
% Unfair coin: shows heads twice as often as tails
r = rand;
if r <= 2/3
    disp('heads')
else
    disp('tails')
end
```

```
% Event X happens with probability p
r = rand;
if r <= p
    % Code for event X
end
```

```
% Given an nr-by-nc matrix M.
% What is A?
for r = 1 : nr
  for c = 1 : nc
    A(c,r) = M(r,c);
  end
end

A
B
C
D

A is M with the columns in reverse order
A is M with the rows in reverse order
A is the transpose of M
A and M are the same

Bidirectional links are blue. Unidirectional link is black as it leaves page j, red when it arrives at page i.

for i = 1 : n
  for j = 1 : n
    A(i,j) = M(i,j);
  end
end

Transpose—like switching row and column indices
A Cost/Inventory Problem

- A company has 3 factories that make 5 different products
- The cost of making a product varies from factory to factory
- The inventory/capacity varies from factory to factory

Problems

A customer submits a purchase order that is to be filled by a single factory.

1. How much would it cost a factory to fill the order?
2. Does a factory have enough inventory/capacity to fill the order?
3. Among the factories that can fill the order, who can do it most cheaply?

Available data

C(1, j) is what it costs factory i to make product j
Inv(i, j) is the inventory in factory i of product j
PO(j) is the number of product j's that the client wants

<table>
<thead>
<tr>
<th>C</th>
<th>10</th>
<th>36</th>
<th>22</th>
<th>15</th>
<th>62</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>35</td>
<td>20</td>
<td>12</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>37</td>
<td>21</td>
<td>16</td>
<td>59</td>
</tr>
<tr>
<td>Inv</td>
<td>38</td>
<td>5</td>
<td>99</td>
<td>34</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>82</td>
<td>19</td>
<td>83</td>
<td>12</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>29</td>
<td>21</td>
<td>56</td>
<td>87</td>
</tr>
<tr>
<td>PO</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>29</td>
<td>5</td>
</tr>
</tbody>
</table>

Cost for factory 1:

1*10 + 0*36 + 12*22 + 29* 15 + 5*62

Encapsulate...

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

```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
```

\( \text{inf} \) – a special value that can be regarded as positive infinity

\[ x = \frac{10}{0} \text{ assigns } \text{inf} \text{ to } x \]
\[ y = 1 + x \text{ assigns } \text{inf} \text{ to } y \]
\[ z = \frac{1}{x} \text{ assigns } 0 \text{ to } z \]
\[ w < \text{inf} \text{ is always true if } w \text{ is numeric} \]

Inventory/Capacity Considerations

What if a factory lacks the inventory/capacity to fill the purchase order?

Such a factory should be excluded from the find-the-cheapest computation.

Who Can Fill the Order?

```
<table>
<thead>
<tr>
<th>Inv</th>
<th>Po</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>83</td>
<td>12</td>
</tr>
<tr>
<td>51</td>
<td>29</td>
</tr>
<tr>
<td>21</td>
<td>56</td>
</tr>
<tr>
<td>56</td>
<td>87</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>PO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
```

Wanted: A True/False Function

```
i
Inv
PO
```

```
iCanDo
```

```
DO
```

DO is “true” if factory \( i \) can fill the order.

DO is “false” if factory \( i \) cannot fill the order.

Example: Check inventory of factory 2

```
<table>
<thead>
<tr>
<th>Inv</th>
<th>Po</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>83</td>
<td>12</td>
</tr>
<tr>
<td>51</td>
<td>29</td>
</tr>
<tr>
<td>21</td>
<td>56</td>
</tr>
<tr>
<td>56</td>
<td>87</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>PO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
```

Method 1: check the inventory for every product
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 = _________;

DO should be true when...
A. j < nProd
B. j == nProd
C. j > nProd

Back To Finding the Cheapest

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

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

Finding the Cheapest

As computed by iCost
As computed by iCanDo

Don't bother with this unless there is sufficient inventory.