Pattern for traversing a matrix $M$

$$[nr, nc] = \text{size}(M)$$

for $r = 1:nr$

% At row r

for $c = 1:nc$

% At column c (in row r)

% Do something with $M(r,c)$ …

end

end

Storing and using data in tables

A merchant has 3 suppliers that stock 5 products with these costs:

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>36</th>
<th>22</th>
<th>15</th>
<th>62</th>
</tr>
</thead>
<tbody>
<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>

Connections between webpages

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

A Cost/Inventory Problem

- A merchant has 3 supplier warehouses that stock 5 different products
- The cost of a product varies from warehouse to warehouse
- The inventory varies from warehouse to warehouse

Problems

A customer submits a purchase order that is to be shipped from a single warehouse.

1. How much would it cost a warehouse to fill the order?
2. Does a warehouse have enough inventory to fill the order?
3. Among the warehouses that can fill the order, who can do it most cheaply?
Available data

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

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

| PO | 1 0 12 29 5 |

Cost for warehouse 1:

\[1 \times 10 + 0 \times 36 + 12 \times 22 + 29 \times 15 + 5 \times 62\]

Finding the Cheapest

```
iBest = 0; minBill = inf
for i = 1:nSuppliers
   iBill = iCost(i, C, PO);
   if iBill < minBill
      % Found an Improvement
      iBest = i; minBill = iBill;
   end
end
```

Aside: floating-point “bonus numbers”

- inf: Represents “infinity”
  - Both positive and negative versions
  - Larger (or smaller) than any other number
  - Generated on overflow or when dividing by zero
- nan: Not-a-number
  - Not equal to anything (even itself)
  - Not greater-than or less-than anything (even inf)
  - Generated from 0/0, inf*0, ...
  - If involved in mathematical operation, result will be nan
Inventory/Capacity Considerations

What if a warehouse lacks the inventory to fill the purchase order?

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

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></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>

Yes
No
Yes

Example: Check inventory of warehouse 2

<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></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>

Method 1: check the inventory for every product

Wanted: A True/False Function

DO is "true" if warehouse i can fill the order.
DO is "false" if warehouse i cannot fill the order.

Example: Check inventory of warehouse 2

DO = DO && ( Inv(2,1) >= PO(1) )

Initialization

<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></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>

Still True...

DO = DO && ( Inv(2,1) >= PO(1) )

DO = DO && ( Inv(2,1) >= PO(1) )
function DO = iCanDo(i, Inv, PO)
% DO is true if warehouse 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...
Encapsulate...

```matlab
function DO = iCanDo(i, Inv, PO)
% DO is true if warehouse 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);
end
```

Back To Finding the Cheapest

```matlab
iBest = 0; minBill = inf;
for i = 1:nSuppliers
    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

```
C
10 36 22 15 62
12 35 20 12 66
13 37 21 16 59
1019 Yes
930 No
1040 Yes
```

Finding the Cheapest

```
PO
1 0 12 29 5
```

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

```
0 0 1 0 1 0 0
1 0 0 1 1 1 0
0 1 0 1 1 1 1
1 0 1 1 0 1 0
0 0 1 1 0 1 1
0 1 0 1 0 1 0
0 1 1 0 1 1 0
```

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 \( \frac{1}{|i-j|} \)
  - 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);    % initialize to 0s
for i = 1:n
    for j = 1:n
        if A(i,j) not on diagonal,
        % assign 1 with some probability
    end
end
end

function A = RandomLinks(n)
% A is n-by-n matrix of 1s and 0s 
% representing n webpages
A= zeros(n,n);    % initialize to 0s
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
    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

Random web: N=20

Represent the web pages graphically...

100 Web pages arranged in a circle.
Next display the links....
Somewhat inefficient: each blue line gets drawn twice. See `ShowRandomLinks.m`.

```
for i = 1:n
    for j = 1:n
        if A(i,j) == 1 & & A(j,i) == 1
            disp('Blue link drawn twice')
        end
    end
end
```

Triangular traversal

```
[nr, nc] = size(M);
for r = 1:n
    for c = 1:n
        disp(M(r,c))
    end
end
```

Case 1

Case 2

What should be A, B, …, 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?