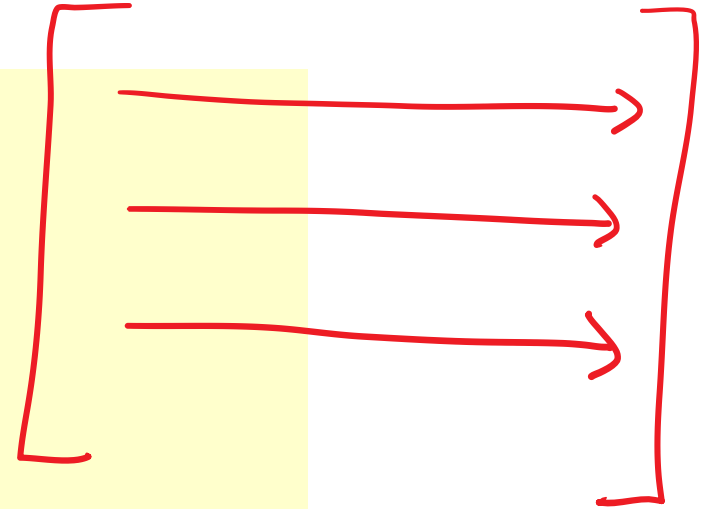


- Previous Lecture:
  - Vectorized operations
  - Introduction to 2-d array—*matrix*
- Today, Lecture 14:
  - Examples on computing with matrices
  - Pre-reading: 7.1 (transition probability matrices)
  - Post-reading: contour plots, PDEs (7.2, 7.3 in *Insight*)
- Announcements:
  - [Prelim 1](#) tonight at 6:30pm
    - Barton Hall, west entrance; see CMS for seat assignment
    - Online: see Canvas for Zoom, Gradescope links
  - Project 4 posted by Thurs, due April 8

## Pattern for traversing a matrix M

```
[nr, nc] = 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
```



## Exercise: what's different about this version?

```
function val = minInMatrix(M)

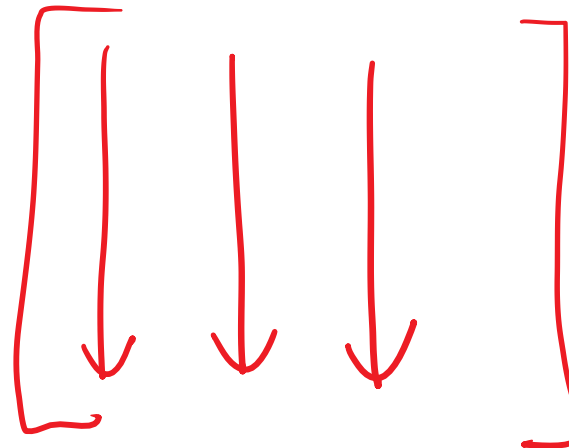
val= M(1,1);
[nr, nc] = size(M);
for c = 1:nc
    for r = 1:nr
        if M(r,c) < val
            val= M(r,c);
        end
    end
end
end
```

A: Nothing (identical to previous version)

B: Searches elements in a different order

C: Index-out-of-bounds if M is not square

D: Doesn't correctly return minimum



# 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

$C(i, j)$  is what it costs warehouse  $i$  to supply product  $j$

$Inv(i, j)$  is the inventory in warehouse  $i$  of product  $j$

$PO(j)$  is the number of product  $j$ 's that the client wants

$C$

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

$Inv$

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

$PO$

1	0	12	29	5
---	---	----	----	---

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5
---	---	----	----	---

Cost for  
warehouse 1:

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

*Work out  
specific  
example, then  
generalize*

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5
---	---	----	----	---

Cost for  
warehouse 1:

```
s = 0; %Sum of cost
for j=1:5 % n cols
    s = s + C(1,j)*PO(j)
end
```



C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5
---	---	----	----	---

Cost for  
warehouse 2:

```
s = 0; %Sum of cost
for j=1:5
    s = s + C(2,j)*PO(j)
end
```

C

10	36	22	15	62
12	35	20	12	66
13	37	21	16	59

PO

1	0	12	29	5
---	---	----	----	---

Cost for  
warehouse  $i$ :

```
s = 0; %Sum of cost
for j=1:5
    s = s + C(i,j)*PO(j)
end
```

## Encapsulate...

```
function theBill = iCost(i,C,PO)
% The cost when warehouse i fills the
% purchase order

nProd= length(PO); or size(c,2)
theBill= 0;
for j = 1:nProd
    theBill= theBill + C(i,j)*PO(j);
end
```

## Finding the Cheapest

Both which warehouse and how cheap

```
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?

Inv	38	5	99	34	42	Yes
	82	19	83	12	42	No
	51	29	21	56	87	Yes
PO	1	0	12	29	5	

## 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

	38	5	99	34	42
<b>Inv</b>	82	19	83	12	42
	51	29	21	56	87
<b>PO</b>	1	0	12	29	5

*Method 1: check the inventory for every product*

# Initialization

<b>Inv</b>	38	5	99	34	42
	82	19	83	12	42
	51	29	21	56	87

**DO** 1

<b>PO</b>	1	0	12	29	5
-----------	---	---	----	----	---

Still True...

	38	5	99	34	42
Inv	82	19	83	12	42
	51	29	21	56	87

DO 1

PO	1	0	12	29	5
----	---	---	----	----	---

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

Still True...

Inv

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

DO 1

PO

1	0	12	29	5
---	---	----	----	---

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

Still True...

**Inv**

38	5	99	34	42
82	19	83	12	42
51	29	21	56	87

**DO** 1

**PO**

1	0	12	29	5
---	---	----	----	---

**DO = DO && ( Inv(2,3) >= PO(3) )**

No Longer True...

	38	5	99	34	42
Inv	82	19	83	12	42
	51	29	21	56	87

DO 0

PO	1	0	12	29	5
----	---	---	----	----	---

`DO = DO && ( Inv(2,4) >= PO(4) )`

Stay False...

	38	5	99	34	42
Inv	82	19	83	12	42
	51	29	21	56	87

DO 0

PO	1	0	12	29	5
----	---	---	----	----	---

```
DO = DO && ( Inv(2,5) >= PO(5) )
```

## Encapsulate...

```
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...

```
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= _____;
```

*Method 2: stop as soon as you  
find one product for which there  
isn't enough inventory*

## Encapsulate...

```
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= _____;
```

DO should be true when...

- A  $j < nProd$
- B  $j == nProd$
- C  $j > nProd$



## Encapsulate...

```
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);
```

inv				
	8	9	5	4
PO	7	3	6	3

## Back To 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
```



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

## Back To Finding the Cheapest

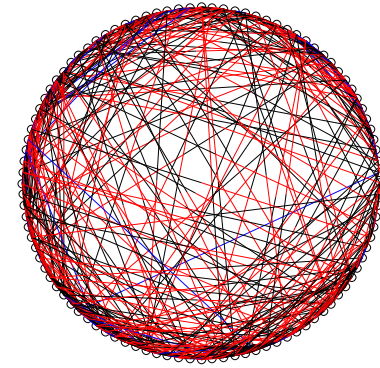
```
iBest= 0; minBill= inf;
for i = 1:nSuppliers
    if iCanDo(i, Inv, PO)
        iBill= iCost(i, C, PO);
        if iBill < minBill
            % Found an Improvement
            iBest= i; minBill= iBill;
        end
    end
end
```

See `Cheapest.m`  
for alternative implementation

# Finding the Cheapest

C	10	36	22	15	62	1019	Yes
	12	35	20	12	66	930	No
	13	37	21	16	59	1040	Yes
PO	1	0	12	29	5		
					As computed by <i>iCost</i>	As computed by <i>iCanDo</i>	

## Matrix example: Random Web



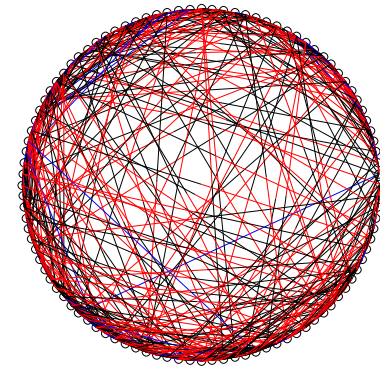
- N web pages can be represented by an N-by-N Link Array  $A$ .
- $A(i, j)$  is 1 if there is a link on webpage  $j$  to webpage  $i$

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	0	1	0	1	0	1
0	1	1	0	1	1	0

Page 4 links to  
Page 2

But Page 2 does  
not link back  
to Page 4

# 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:

				$j$
$i$	0			
		0		
			0	
				0

- There is no link from a page to itself

- If  $i \neq j$  then  $A(i,j) = 1$  with probability  $\frac{1}{1+|i-j|}$

➡ 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); % 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
```

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
```

```
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
```