- 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*)
- Announcements:
  - Fall Break Mon & Tues: no lec, dis, office/consulting hrs. Attendance at 10/16 (W) dis is optional, but the exercise is required. Attend any of the 10/16 dis sections if you wish. Location is Upson B7 lab.
  - *Optional* review sessions: W 5-6:30p, W 7-8:30p. Locations TBA

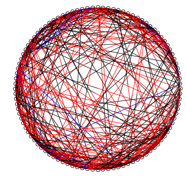
Storing and using data in *tables*

A company has 3 factories that make 5 products with these costs:

C

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

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



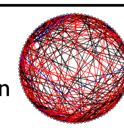
Connections between webpages

```
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
```


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
```


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 $\frac{1}{1+|i-j|}$

  There is more likely to be a link if i is close to j

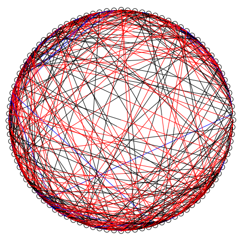

```
function A = RandomLinks(n)
% A is n-by-n matrix of 1s and 0s
% representing n webpages

A = zeros(n,n);
for i=1:n
  for j=1:n
    r = rand(1);
    if i~=j && r<= 1/(1 + abs(i-j));
       A(i,j) = 1;
    end
  end
end
```


Random web
N = 20

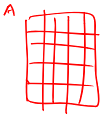
```
0 1 1 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0
0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0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

Represent the web pages graphically…



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

A



for i = 1:n
  for j = 1:n

  end
end

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?

Cost Array

C

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

The value of `C(i,j)` is what it costs factory i to make product j.

Inventory (or Capacity) Array

Inv

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

The value of `Inv(i,j)` is the inventory in factory i of product j.

Purchase Order

PO				
1	0	12	29	5

The value of PO(j) is the number of product j's that the customer wants

C

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

PO

1	0	12	29	5

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


inf – a special value that can be regarded as positive infinity

`x = 10/0`	assigns `inf` to `x`
`y = 1+x`	assigns `inf` to `y`
`z = 1/x`	assigns `0` to `z`
`w < inf`	is always true if `w` 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?

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 factory i can fill the order.

DO is "false" if factory i cannot fill the order.

Example: Check inventory of factory 2

Inv

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

PO

1	0	12	29	5

Method 1: check the inventory for every product

Initialization

Inv

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

DO 1

PO

1	0	12	29	5

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(2,1) >= PO(1) )
```


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 = _________;
```

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
```

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

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 `iCost`

As computed by `iCanDo`


```
% 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 A is M with the columns in reverse order

B A is M with the rows in reverse order

C A is the transpose of M

D A and M are the same