More Matrices

Lecture 15 (Mar 11)
CS100M – Spring 2008

Announcements

- Prelim 2 is onThursday, March 13
  - Time: 7:30-9:00 pm
  - Includes material through Wednesday, March 5
    - Includes functions and vectors
    - No matrices on prelim 2
    - There is an document on the website about *vectorized code*
- Section is in Lab this week
- Today's topics
  - More on matrices
  - Matrices and contour plotting

Review: Functions

- Functions "communicate" via
  - Parameters
  - Return value(s)
- Generally, the function body does not use
  - Built-in function input
  - Built-in function fprintf
- If you find yourself using input or fprintf within your function…
  - There is a good chance you're doing something wrong

```
function A = createExample(m,n)
for r = 1:m
    for c = 1:n
        A(r, c) = 10*r + c;
    end
end
```

Review: Functions

- Each function resides in a separate file
  - The name of the file matches the function name
  - Matlab finds the function by finding the file in the file system
  - Sometimes your function needs a "helper" function
    - The helper function (called a *subfunction* in Matlab) can reside in the same file

In the file "createExample":

```
function A = createExample(m,n)
for r = 1:m
    for c = 1:n
        A(r, c) = 10*r + c;
    end
end
```

To call createExample from another file:

```
B = createExample(5, 10)
```

Good Programming Style

- Don't use variable names that are the same as built-in function names

```
length = 15
% Length func won't work now
```

- Use meaningful variable names
  - But OK to use i, j, k, m, n for simple indices

```
s = 0;
for k= 1:n
        s = s + vect(k);
end
```

Functions and 2D Arrays

```
function  alpha = Ave(A)
% A is a 2D array
% alpha is the average of A's values
```

```
10 20 30
40 50 60
```

```
-> (10+20+30+40+50+60)/6
```

Need Built-In Function *size*

```
function  alpha = Ave(A)
[m,n] = size(A);

      Add up all the numbers in the array.
      Store in s.

alpha = s/(m*n);
```

Refine…

```
function  alpha = Ave(A)
 [m,n] = size(A);

 s = 0;
 for i=1:m
       for j = 1:m
             s = s + A(i,j);
       end
 end

 alpha = s/(m*n);
```

Now Some Less-Trivial Examples

Random Web

N web pages

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

Random Link Idea

$A(i,,j) = 1$ with probability $\dfrac{1}{1+|i-j|}$

Intuition:

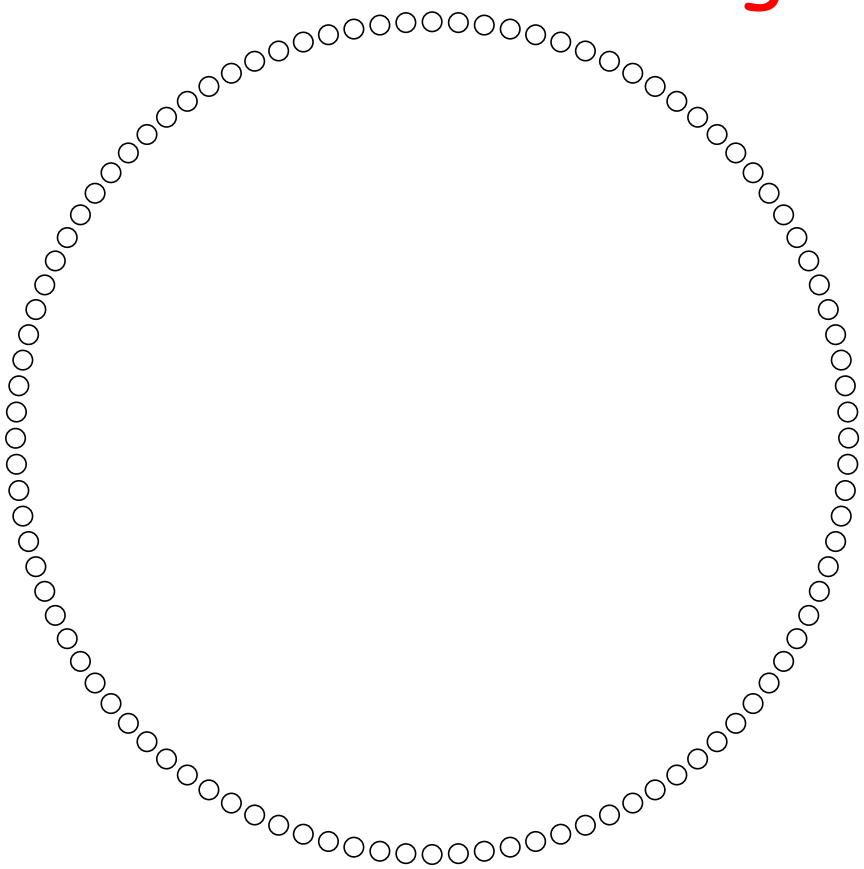
More likely to be a link if i is close to j

```
function A = RandomLinks(n)
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
```

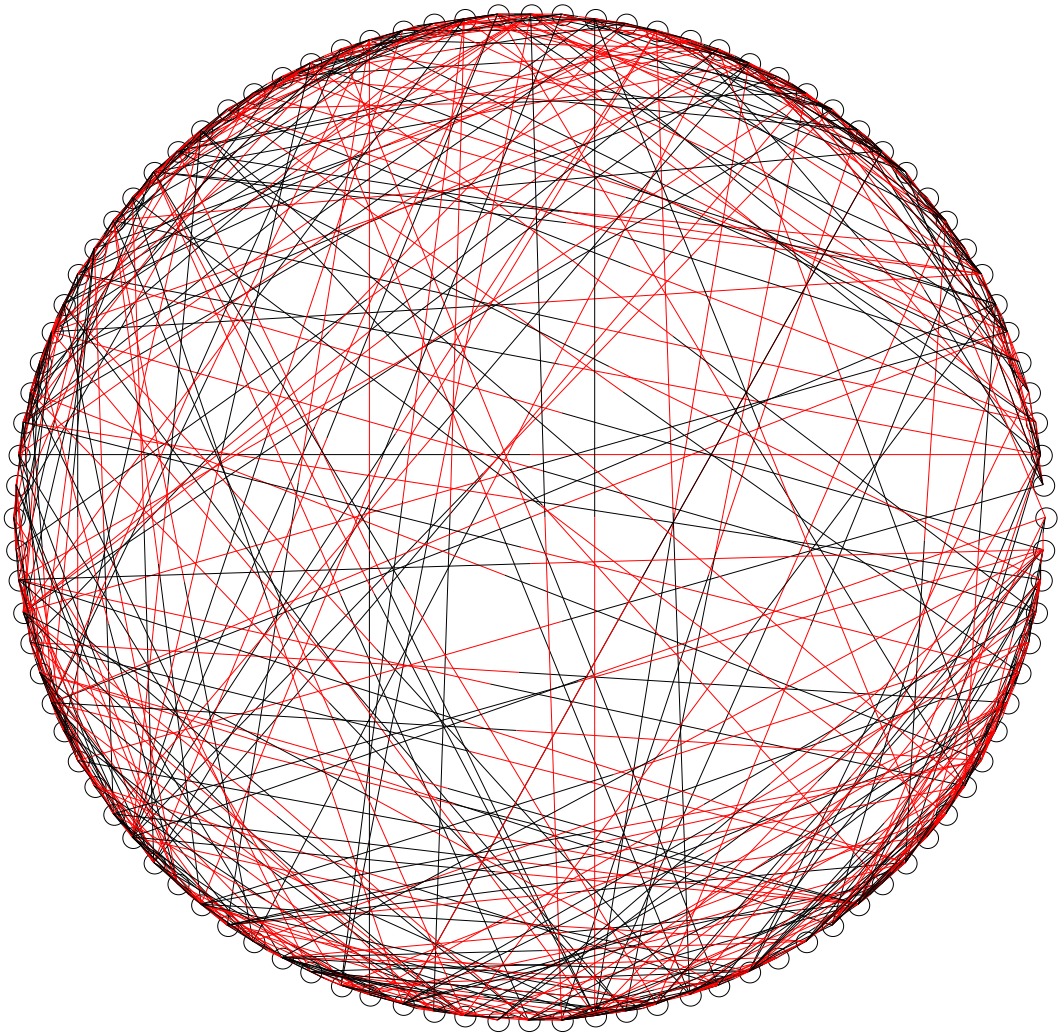
N = 20

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

100 Web Pages



Now display the links….



Each line is black as it leaves page j, red when it arrives at page i

New Problem

Visualizing a function of the form

$$z = f(x,y)$$

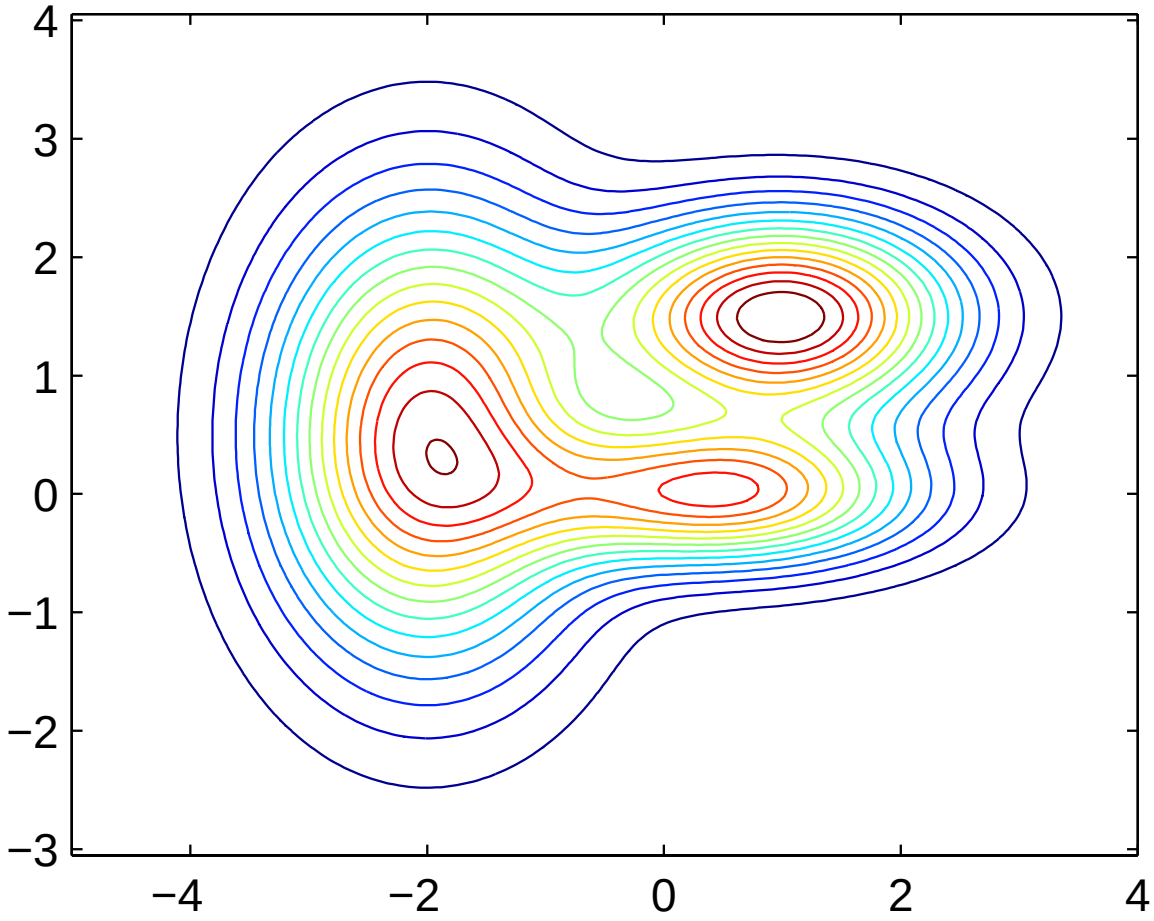
Think of z as an elevation which depends on the location; coordinates x and y describe the location

Sample Elevation Function

```
function z = Elev(x,y)
% A function with peaks at (1,1.5), (-2,.5), and (.5,0)
% Peak heights are 100, 90, and 80 resp.

 r1 = (x-1)^2 + 3*(y-1.5)^2;
 r2 = 2*(x+2)^2 + (y-.5)^2;
 r3 = (x-.5)^2 + 7*y^2;
 z = 100*exp(-.5*r1) + 90*exp(-.3*r2) + 80*exp(-.4*r3);
```

Its Contour Plot



Making a Contour Plot

```
x = linspace(-5,4,200);
y = linspace(-2.5,6.5,200);
A = zeros(200,200);
for i=1:200
  for j=1:200
     A(i,j) = Elev(x(j),y(i));
  end
end

contour(x,y,A,15)
```

- Set up a matrix of function evaluations
- Use the built-in function contour
  - The last argument (15) is the number of contour lines

General Set-Up

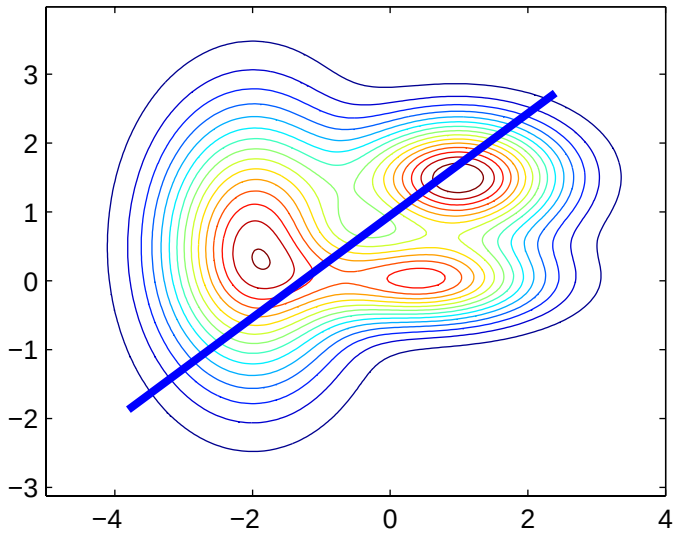
```
function A = SetUp(f,xVals,yVals)
Nx = length(xVals);
Ny = length(yVals);
A = zeros(Ny,Nx);
for i=1:Ny
  for j=1:Nx
     A(i,j) = f(xVals(j),yVals(i));
  end
end
```

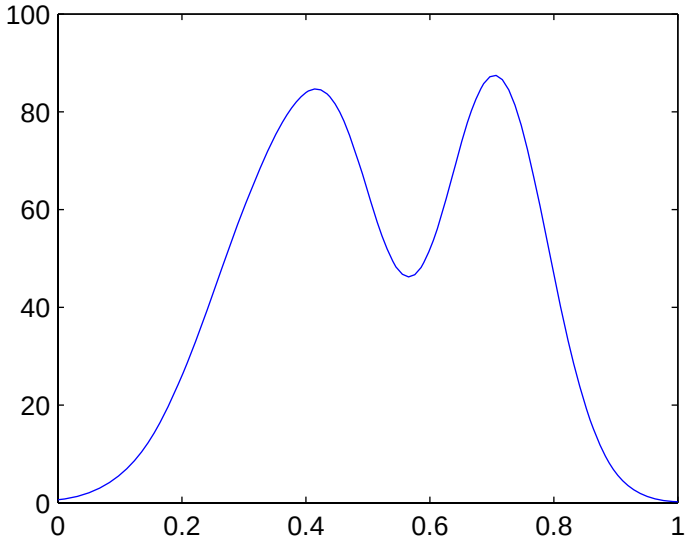
Calling SetUp

```
x = linspace(-5,4,200);
y = linspace(-2.5,6.5,200);
F = SetUp(@Elev,x,y);
```

- Not just 'Elev'
  - The @ is required for function parameters
  - Without @, Matlab attempts to call the function

Generating a Cross Section





Enter endpoints via **`ginput`**
Sample Elev(x,y) along the line segment

Mouse Input via ginput

```
[a,b] = ginput(2);
plot(a,b)
```

- [a,b] = ginput(n) puts the mouseclick coordinates in length-n arrays a and b.
- The plot statement draws the line segment connecting (a(1),b(1)) and (a(2),b(2))

Determining Elevations along the Line

```
n = 100;
t = linspace(0,1,n);
x = linspace(a(1),a(2),n);
y = linspace(b(1),b(2),n);
for i=1:n
        % At "time" t(i) we are at (x(i),y(i)).
        % Compute elevation at time t(i).
        f(i) = Elev(x(i),y(i));
end
figure
plot(t,f)
```

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 varies from factory to factory
- A customer submits a purchase order that is to be filled by a single factory
  - Find the cheapest way to do this

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

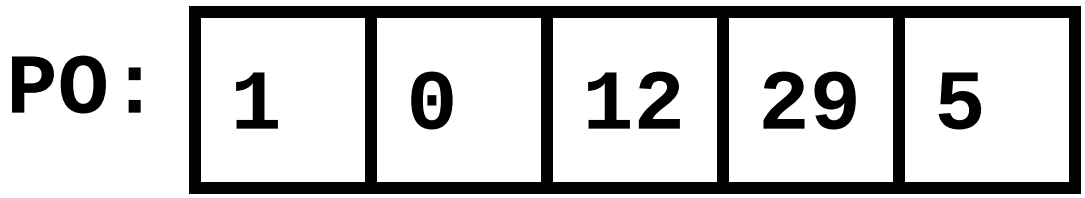
Inventory Array

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 at factory i of product j

Purchase Order



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

For factory 1:

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

C:

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

PO:

1	0	12	29	5

j = 1

For factory 1:

```
s = 0;
for j=1:5
   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

j = 2

For factory 1:

```
s = 0;
for j=1:5
   s = s + C(1,j) * PO(j)
end
```

C:

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

j = 3

PO:

1	0	12	29	5

For factory 1:

```
s = 0;
for j=1:5
   s = s + C(1,j) * PO(j)
end
```

C:

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

j = 4

PO:

1	0	12	29	5

For factory 1:

```
s = 0;
for j=1:5
   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

j = 5

For factory 1:

```
s = 0;
for j=1:5
   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

For factory 2:

```
s = 0;
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

For factory i:

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

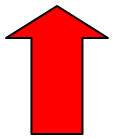
Encapsulate…

```
  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

	10	36	22	15	62	1019
C:	12	35	20	12	66	930
	13	37	21	16	59	1040

PO:	1	0	12	29	5



As computed by **iCost**

Finding Cheapest: Initialization

C:

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

PO:

1	0	12	29	5

1019

930

1040

Can we do better?

iBest: 0

minBill: inf

A Note on "`inf`"

A special value that can be regarded as
+ infinity.

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

Improvement at i = 1

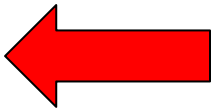
C:

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

1019

930

1040



PO:

1	0	12	29	5

iBest: 1

minBill: 1019

Improvement at i = 2

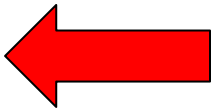
C:

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

1019

930

1040



PO:

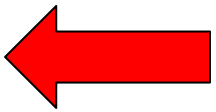
1	0	12	29	5

iBest: 2

minBill: 930

No Improvement at i = 3

C:	10	36	22	15	62	1019
	12	35	20	12	66	930
	13	37	21	16	59	1040



PO:	1	0	12	29	5

iBest: 2

minBill: 930

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
```

Inventory Considerations

- What if a factory lacks the inventory 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

Because 12 < 29

Wanted: A True/False Function

B is "true" if factory i can fill the order.

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

Boolean Operations in Matlab

When discussing expressions like

```
a <= x  && x <= b
```

```
abs(y) > 10
```

we say that an expression is either true or false

The 0-1 Secret

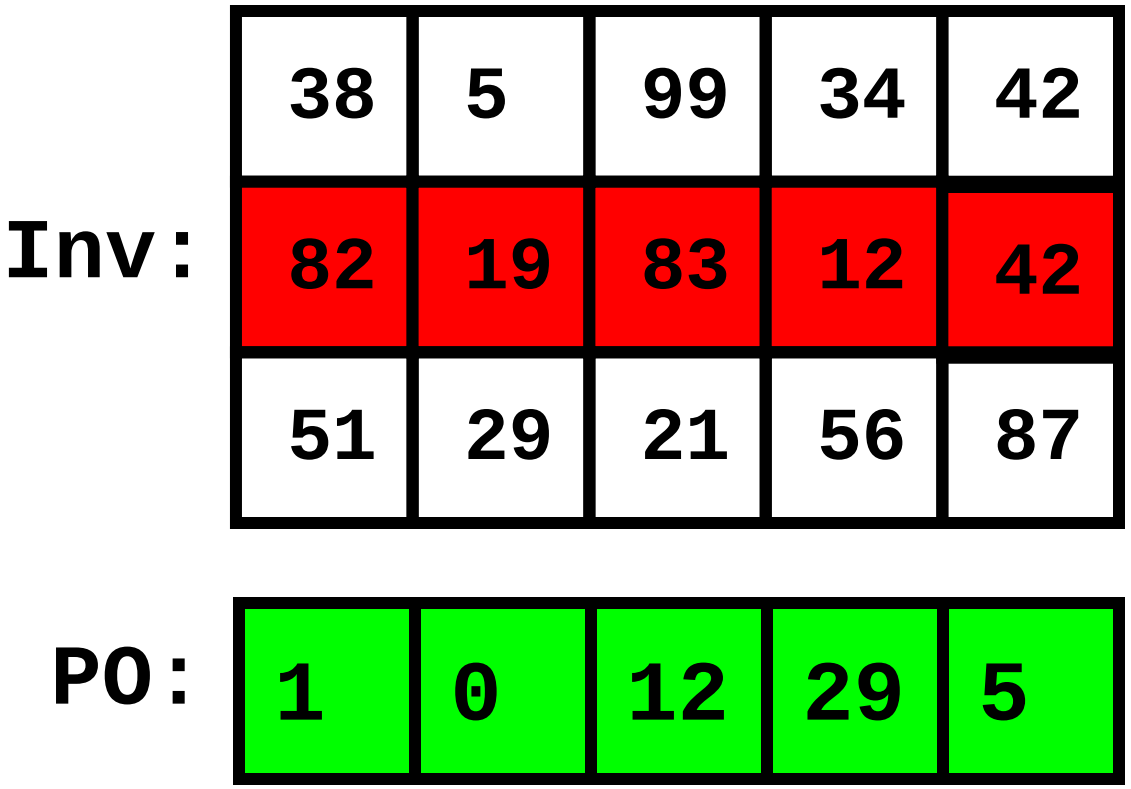
In reality, expressions like

```
a <= x  && x <= b
```

```
abs(y) > 10
```

render the value "1" if they are TRUE and "0" if they are FALSE

Back to Inventory Problem



Initialization

Inv:

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

B: 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

B: 1

PO:

1	0	12	29	5

```
B = B && ( Inv(2,1) >= PO(1) )
```

Still True…

Inv:

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

B: 1

PO:

1	0	12	29	5

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

Still True…

Inv:

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

B: 1

PO:

1	0	12	29	5

```
B = B && ( Inv(2,3) >= PO(3) )
```

No Longer True…

Inv:

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

B: 0

PO:

1	0	12	29	5

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

Encapsulate…

```
  function  B = iCanDo(i,Inv,PO)
% B is true if factory i can fill
% the purchase order. Otherwise, false
  nProd = length(PO);
  j = 1;
  B = 1;
  while j<=nProd && B
     B = B && ( Inv(i,j) >= PO(j) );
     j = j+1;
  end
```

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
```

Can't be "best" if insufficient inventory

Back To Finding the Cheapest

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

Finding the Cheapest