L20. More on 2D Arrays

Operations

Subscripting

Functions* & 2D Arrays

*Will see two new things.

- Boolean-valued functions
- Functions that have a function as a parameter.

Two Applications

A commercial setting that involves cost arrays, inventory arrays, and purchase orders.

A setting that requires the visualization of a function of two variables f(x,y) via contour plotting.

A Cost/Inventory Setting

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.

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 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 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 product j's that the customer wants

How Much Does it Cost for Each Factory to Process a Purchase order?

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

j = 1

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 = 2

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 = 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

j = 5

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

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

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

As computed by **`iCost`**

Finding Cheapest: Initialization

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

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	1019
12	35	20	12	66	930
13	37	21	16	59	1040

PO:

1	0	12	29	5

iBest: 1

minBill: 1019

Improvement at i = 2

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

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

SO FAR we have indicated that expressions like

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

abs(y) > 10
```

are 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.

Example

```
>> x = 8; y = 7;

>> B = x<y
B =
     0

>> B = x>y
B =
     1
```

A Boolean-Valued Function

```
  function  B = Overlap(a,b,c,d)
% B is true if intervals [a,b]
% and [c,d] intersect.
% Otherwise B is false.
% Assume a<b and c<d.

  abToLeft  = b < c;
  abToRight = d < a;
  B = ~(abToLeft || abToRight);
```

Using Overlap

```
S = 0;
for k=1:100
   a = rand; b = a + rand;
   c = rand; d = c + rand;
   if Overlap(a,b,c,d)
      s = s+1;
   end
end
probOverlap = s/100
```

Back to Inventory Problem

Inv:

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

PO:

1	0	12	29	5

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
```

Don't bother with this unless 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
```

```
function [iBest,minBill] = …
                   Cheapest(C,Inv,PO)
[nFact,nProd] = size(C);
iBest = 0; minBill = inf;
for i=1:nFact
   if iCanDo(i,Inv,PO)
      iBill = iCost(i,C,PO);
      if iBill < minBill
         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

PO:

1	0	12	29	5

$\boxed{1019}$	$\boxed{\text{Yes}}$
930	No
1040	Yes

As computed by **iCost**

As computed by **iCanDo**

New Problem

Visualizing a function of the form

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

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

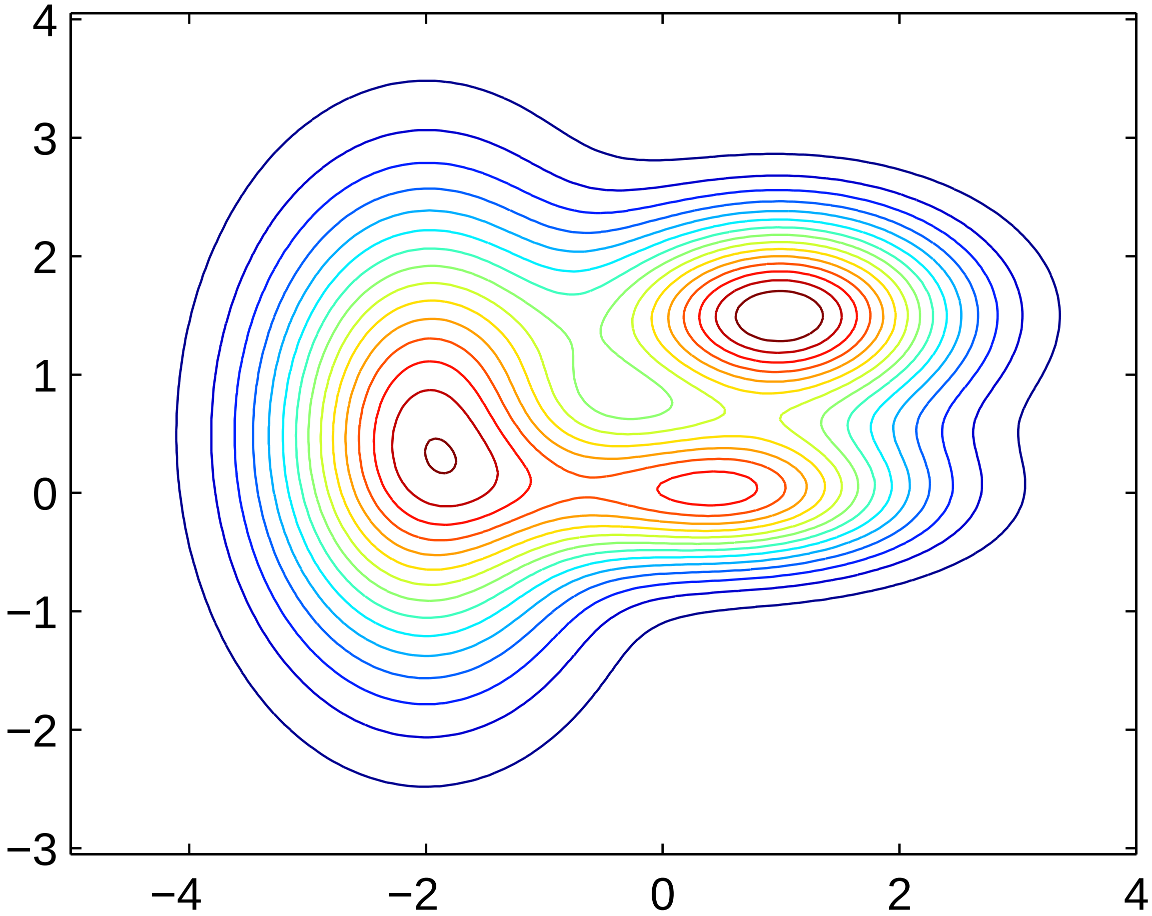
Sample Elevation Function

```
function z = Elev(x,y)

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);
```

Three Hills at (1,1.5),(-2,.5), (.5,0)

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 evals

General Set-Up

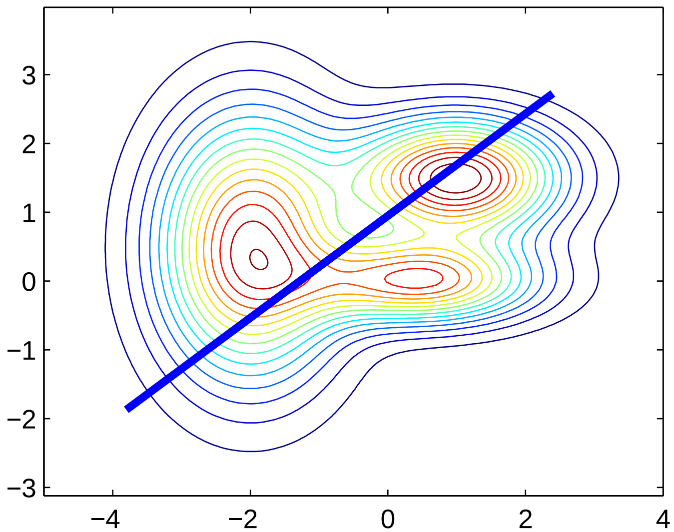
```
function A = SetUp(f,xVals,yVals)
Nx = length(xVals);
Ny = length(yVals);
A = zeros(Ny,Nx);
for i=1:Ny
  for j=1:Nx
     A(i,j) = f(xVals(j),yVals(i));
  end
end
```

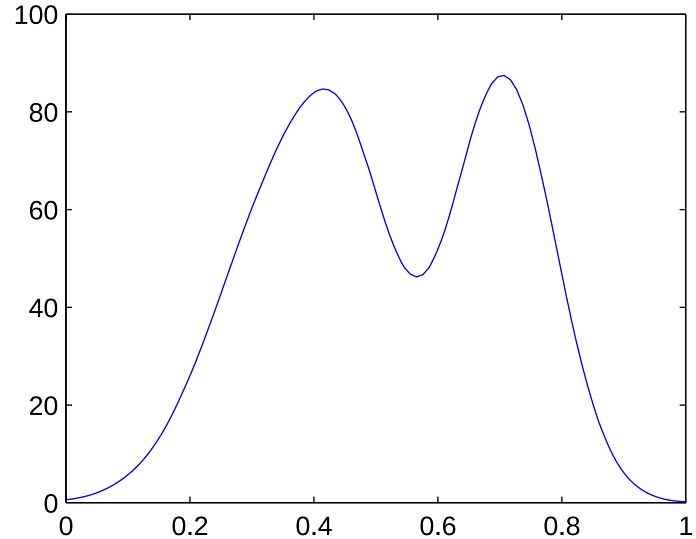
Calling SetUp

```
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.

Generating a Cross Section





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

Mouse Input Via `ginput`

To draw a line segment connecting (a(1),b(1)) and (a(2),b(2)):

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

`[a,b] = ginput(n)` puts the mouseclick coords in length-n arrays **a** and **b**.

```
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)
```