- Previous Lecture:
  - 2-d array—matrix
- Today's Lecture:
  - More examples on matrices
  - Contour plot (see 7.2, 7.3 in *Insight*)
- Announcements:
  - Project 3 due tonight at 11pm
  - See website for announcement on review session
  - Prelim 2 on Thurs, Oct 21st, 7:30-9pm. Email Randy Hess (rbhess@cs.cornell.edu) NOW if you have an exam conflict

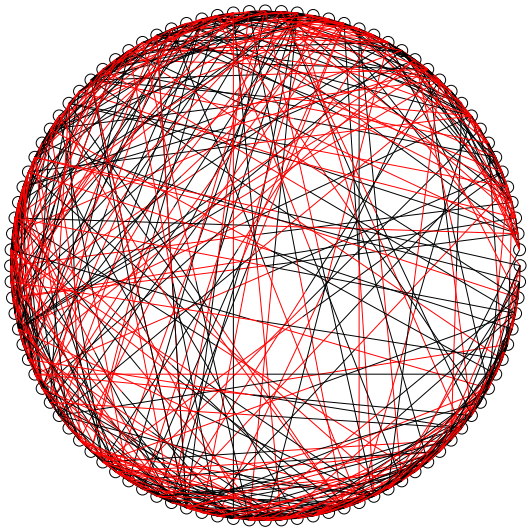
Storing and using data in *tables*

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

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

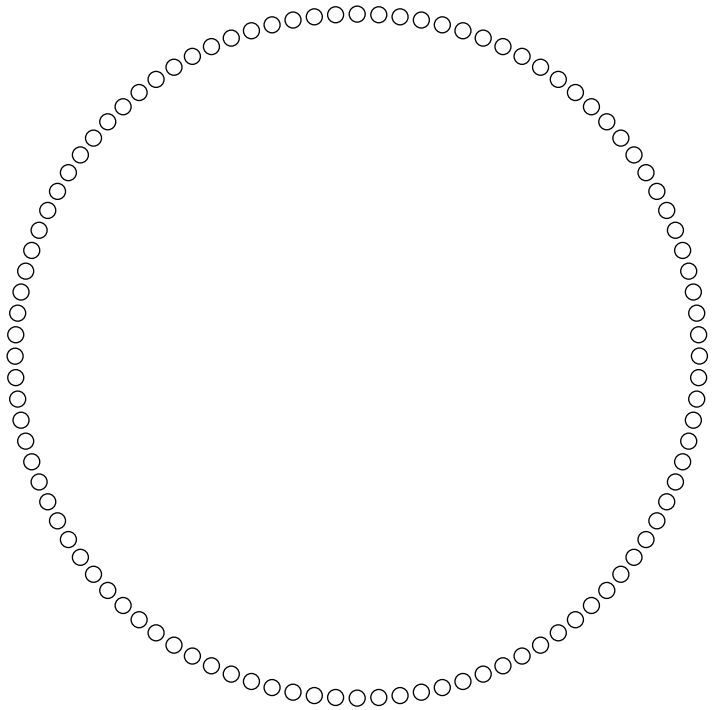
```
function A = RandomLinks(n)
% A is n-by-n matrix of 1s and 0s
% representing n webpages

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

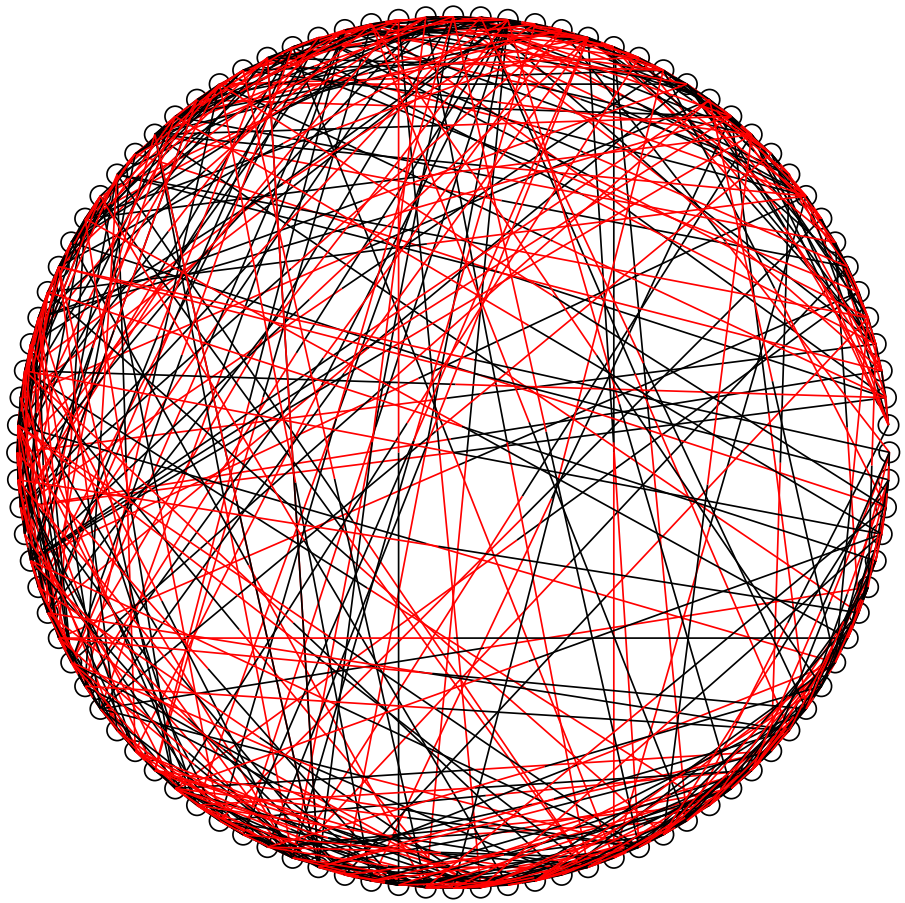
0	1	1	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
0	1	1	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	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…



100 Web pages arranged in a circle.
Next display the links….

Represent the web pages graphically…



Line black as it leaves page j, red when it arrives at page i

ShowRandomLinks.m

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?

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
```

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?

	38	5	99	34	42	Yes
Inv	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

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

P0	1	0	12	29	5

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) )
```

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…

Inv

38	5	99	34	42
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…

Inv

38	5	99	34	42
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 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 = _______;
```

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`

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 = (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
```

Cheapest.m

Finding the Cheapest

	10	36	22	15	62	1019	Yes
C	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**

Initialize vectors/matrices if dimensions are known

…instead of "building" the array one component at a time

```
% Initialize y
x=linspace(a,b,n);
y=zeros(1,n);
for k=1:n
   y(k)=myF(x(k));
end
```

```
% Build y on the fly
x=linspace(a,b,n);

for k=1:n
   y(k)=myF(x(k));
end
```

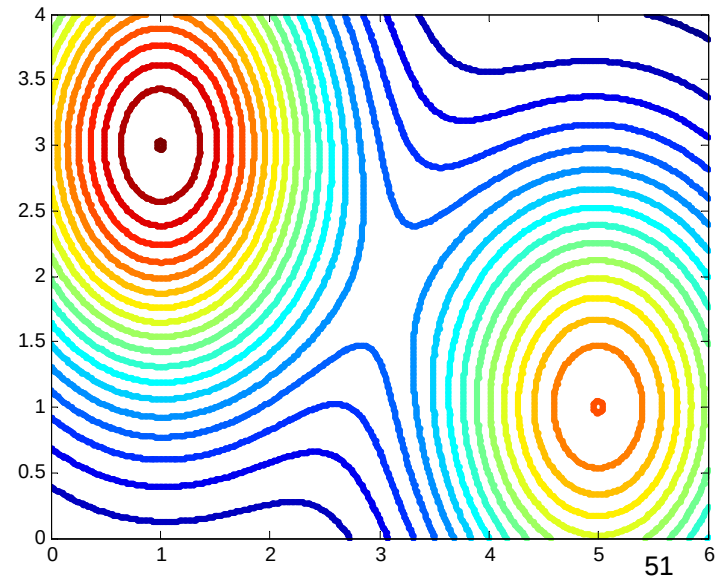
Much faster for large n!

Contour Plot

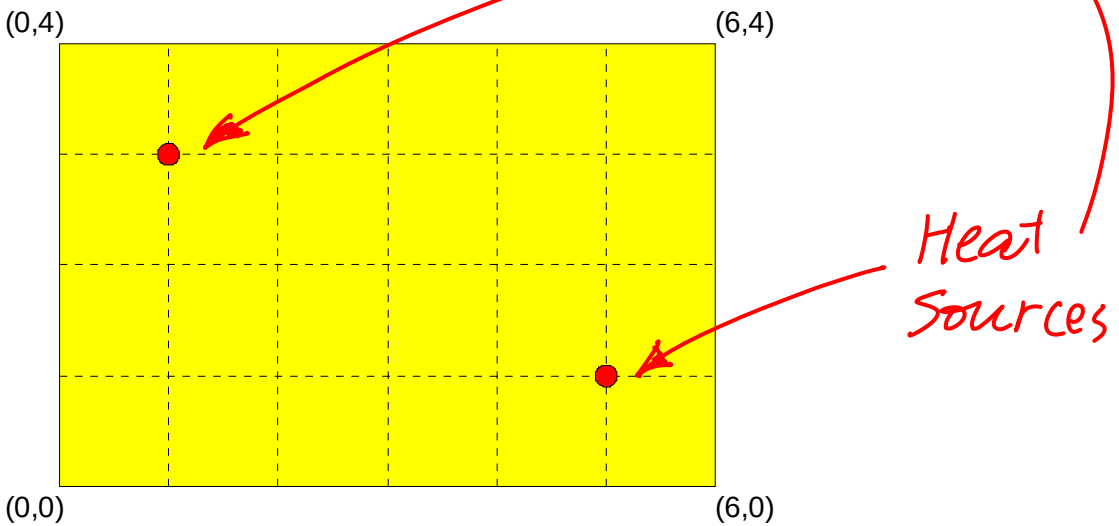
Visualize a function of the form

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

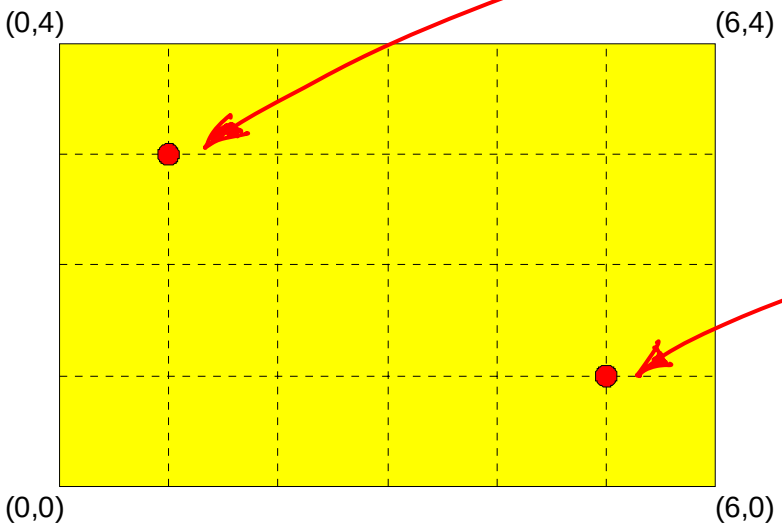
Think of z as an elevation that depends on the coordinates x and y of the location.



Visualize temperature distribution—contour plot



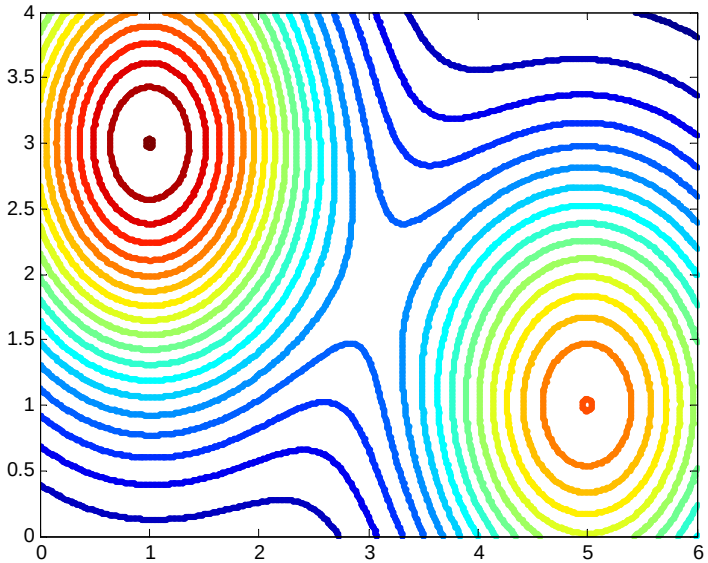
Visualize temperature distribution—contour plot



Heat Sources

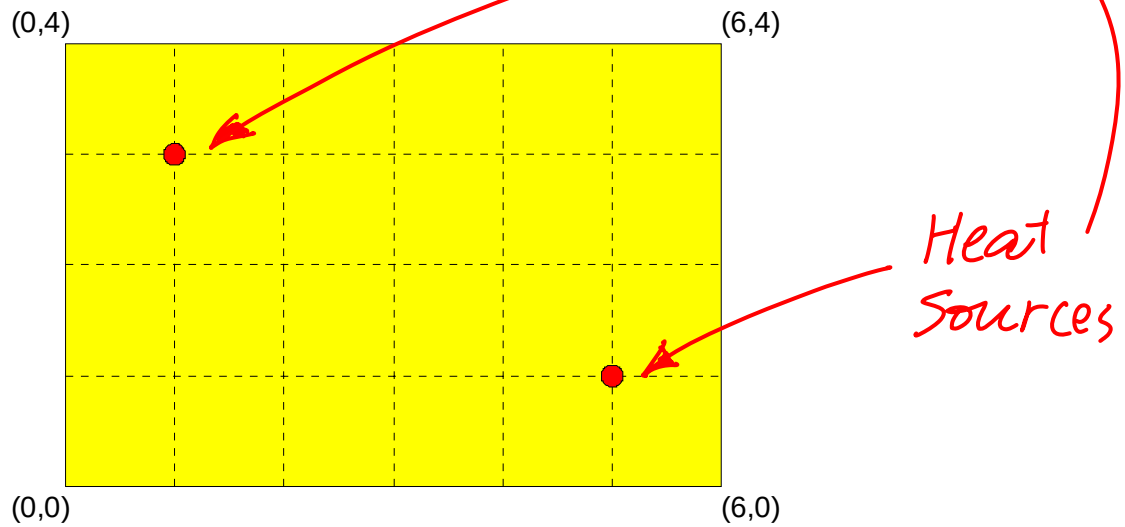
$$T(x,y) = 100e^{-.4((x-1)^2+.7(y-3)^2)} + 80e^{-.2(2(x-5)^2+1.5(y-1)^2)}$$

Making a contour plot



1. Set up x-y grid
2. Evaluate function at all grid points
3. Draw the contours

Visualize temperature distribution—contour plot



$$T(x,y) = 100e^{-.4((x-1)^2+.7(y-3)^2)} + 80e^{-.2(2(x-5)^2+1.5(y-1)^2)}$$

```
function t = T_plate(x,y)
% t is temperature at (x,y)

t = 100*exp(-.4*(  (x-1)^2 + 0.7*(y-3)^2))+...
     80*exp(-.2*(2*(x-5)^2 + 1.5*(y-1)^2));
```

Setting up the grid

```
x = linspace(0,6,13);
y = linspace(0,4,9);

% fVals is matrix of function
% values at all the grid points
fVals = zeros(___,___);
```

A 13,9

B 9,13

C 13,13

D 9,9

Making a Contour Plot

```
x = linspace(0,6,13);
y = linspace(0,4,9);
fVals = zeros(___,___);
for j=1:___
  for i=1:___
     fVals(i,j) = T_plate(_____,_____);
  end
end

contour(x,y,fVals,20)
```

Use 20 contours

Making a Contour Plot

```
x = linspace(0,6,13);
y = linspace(0,4,9);
fVals = zeros(9,13);
for j=1:13
  for i=1:9
     fVals(i,j) = T_plate(x(j),y(i));
  end
end
contour(x,y,fVals,20)
```

Use 20 contours

Setting up a matrix of function evaluations

General setup to get a matrix of function evaluations

```
function fVals = fOnGrid(x,y,f)
% fVals is a matrix where
% fVals(i,j) = f(x(j),y(i))

nX = length(x);  mY = length(y);
fVals = zeros(mY,nX);
for j = 1:nX
  for i = 1:mY
     fVals(i,j) = f(x(j),y(i));
  end
end
```

General setup to get a matrix of function evaluations

```
function fVals = fOnGrid(x,y,f)
% fVals is a matrix where
% fVals(i,j) = f(x(j),y(i))

nX = length(x);  mY = length(y);
fVals = zeros(mY,nX);
for j = 1:nX
  for i = 1:mY
     fVals(i,j) = f(x(j),y(i));
  end
end
```

This parameter is a **function**, not a value

Calling **fOnGrid** and passing a function handle

```
x = linspace(0,6,301);
y = linspace(0,4,201);
TVals = fOnGrid(x,y,@T_plate);
```

T_plate is the function name.
@ is required when the argument is a function, not a variable.

Calling `fOnGrid` and passing a function handle

```
x = linspace(0,6,301);
y = linspace(0,4,201);
TVals = fOnGrid(x,y,@T_plate);
```

```
function fVals = fOnGrid(x,y,f)
% fVals(i,j) = f(x(j),y(i))
nX = length(x);  mY = length(y);
fVals = zeros(mY,nX);
for j = 1:nX
  for i = 1:mY
     fVals(i,j) = f(x(j),y(i));
  end
end
```