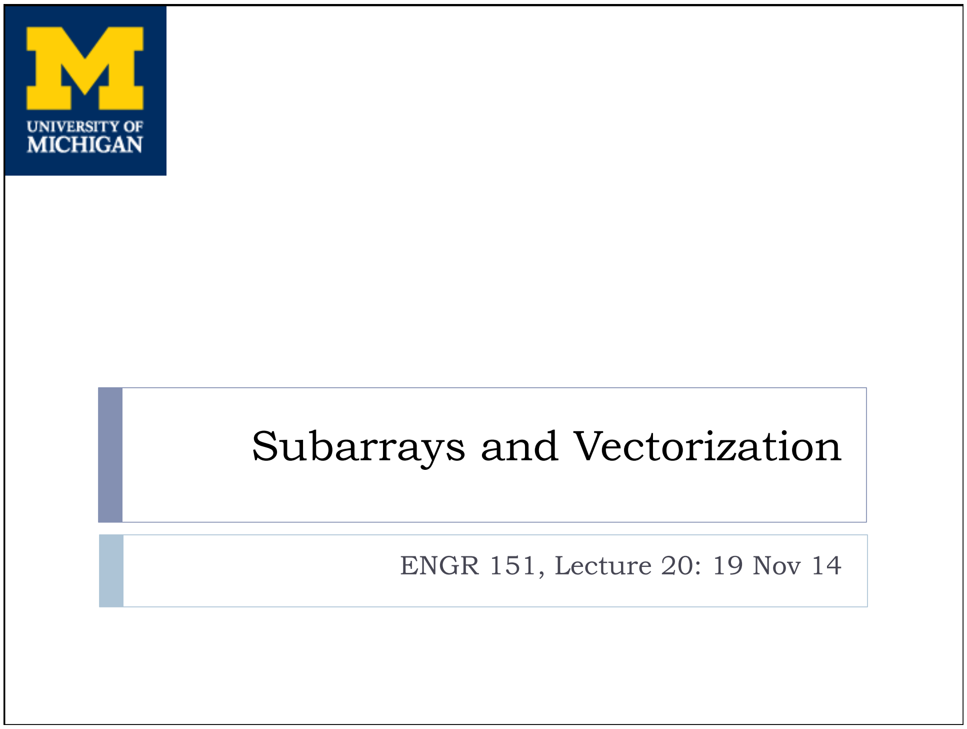
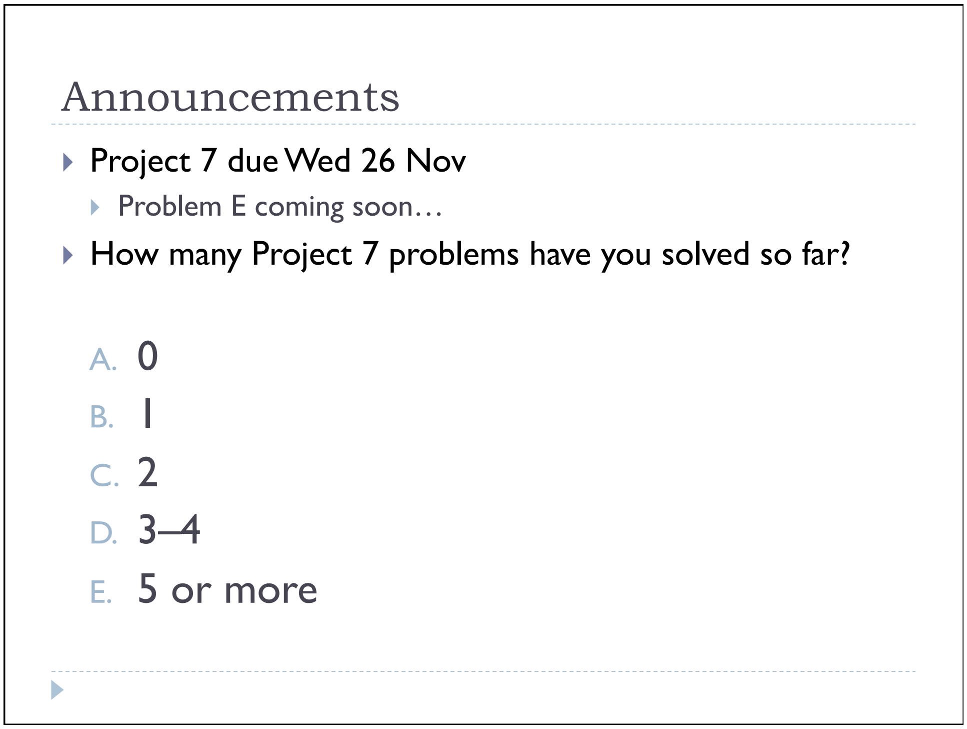


Subarrays and Vectorization

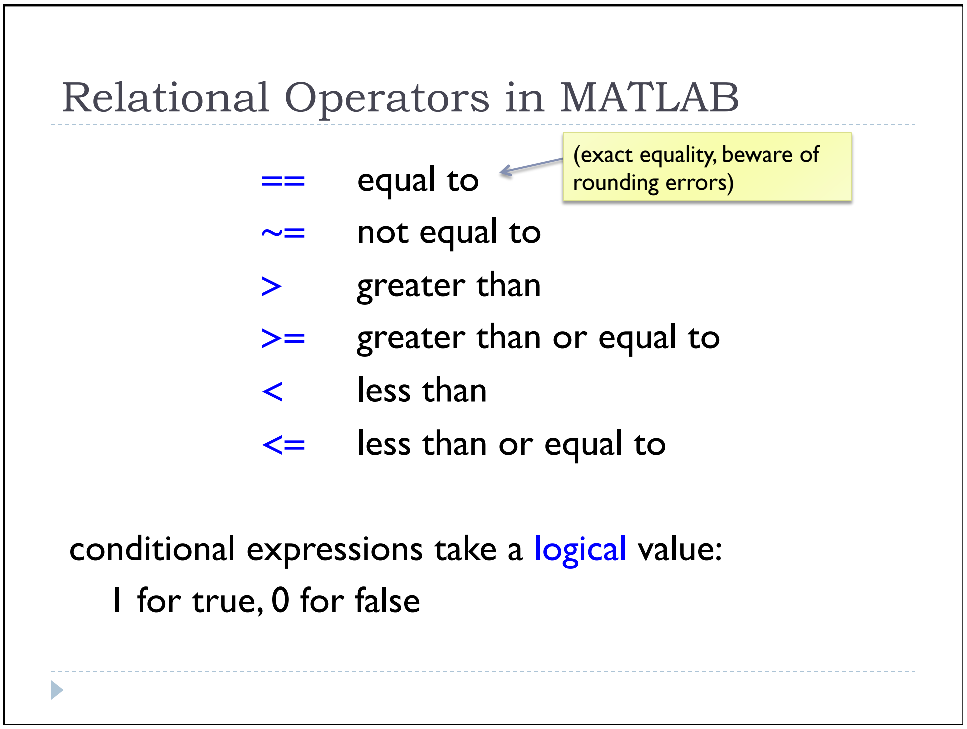
ENGR 151, Lecture 20: 19 Nov 14



Announcements

- Project 7 due Wed 26 Nov
  - Problem E coming soon…
- How many Project 7 problems have you solved so far?

A. 0
B. 1
C. 2
D. 3–4
E. 5 or more



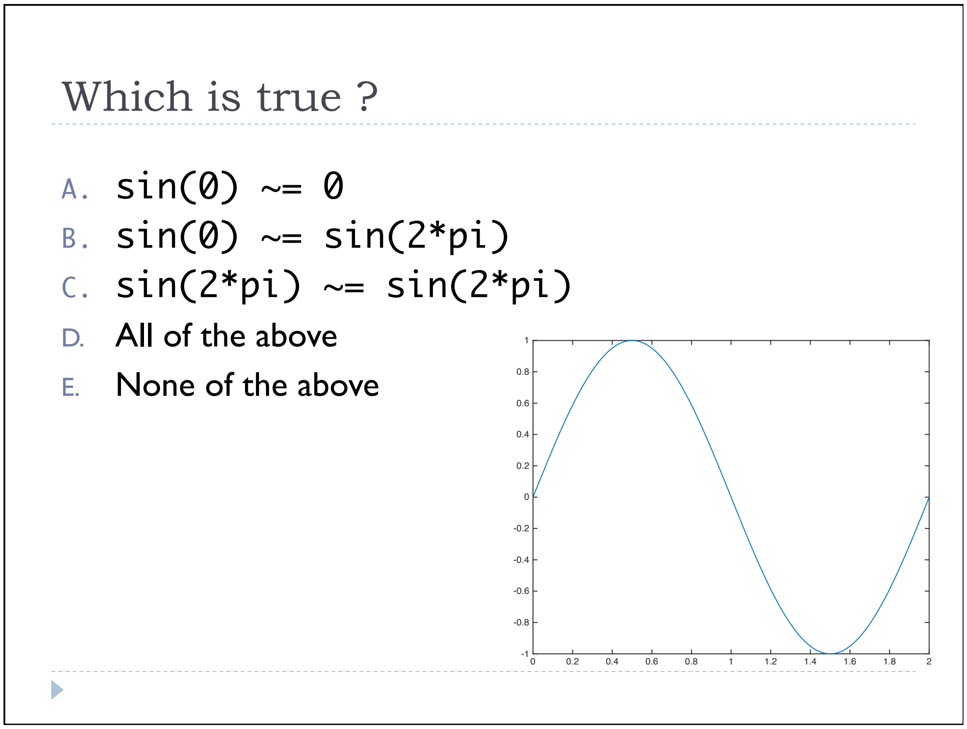
Relational Operators in MATLAB

==	equal to
~=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

(exact equality, beware of rounding errors)

conditional expressions take a logical value:

1 for true, 0 for false



Which is true ?

A. `sin(0) ~= 0`
B. `sin(0) ~= sin(2*pi)`
C. `sin(2*pi) ~= sin(2*pi)`
D. All of the above
E. None of the above

1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

Relational Operators on Arrays

- Apply elt-by-elt between arrays

$$\begin{pmatrix} 1 & 5 \\ 3 & -1 \end{pmatrix} > \begin{pmatrix} 3 & 1 \\ -1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- Also apply elt-by-elt between a scalar and an array

$$5 > \begin{pmatrix} 8 & 2 \\ 10 & -1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

Logical Expressions

operator	function	meaning
&	and	*and*
|	or	*or*
~	not	*not*
	xor	*exclusive or*

p	q	~p not(p)	p & q and(p,q)	p | q or(p,q)	xor(p,q)
false	false	1	0	0	0
false	true	1	0	1	1
true	false	0	0	1	1
true	true	0	1	1	0

Short-Circuit Evaluation

- && and || are short-circuit versions of & and |
  - do not evaluate right operand if left operand is sufficient
- Examples:

```
true | [1; 2 3]       →  error!
true || [1; 2 3]      →  true
false || [1; 2 3]     →  error!
```

xor

- Suppose L1 and L2 are two logical vectors, of the same length.
- What is xor(L1,xor(L1,L2)) ?

A. vector of false

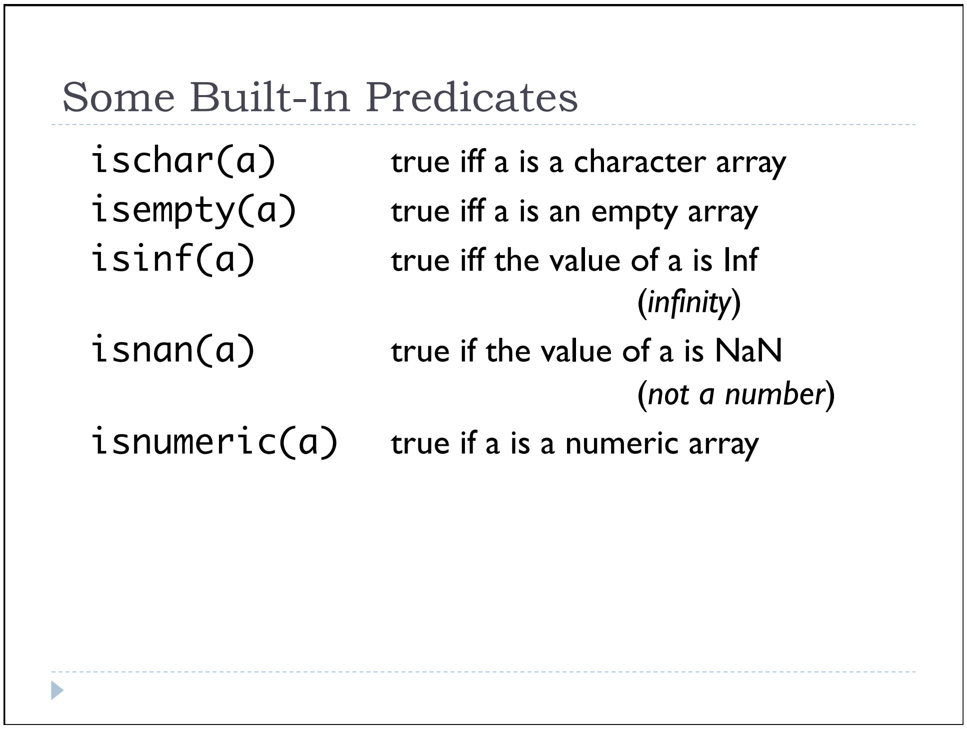
B. L1 & L2

C. L2

D. xor(L2,L1)

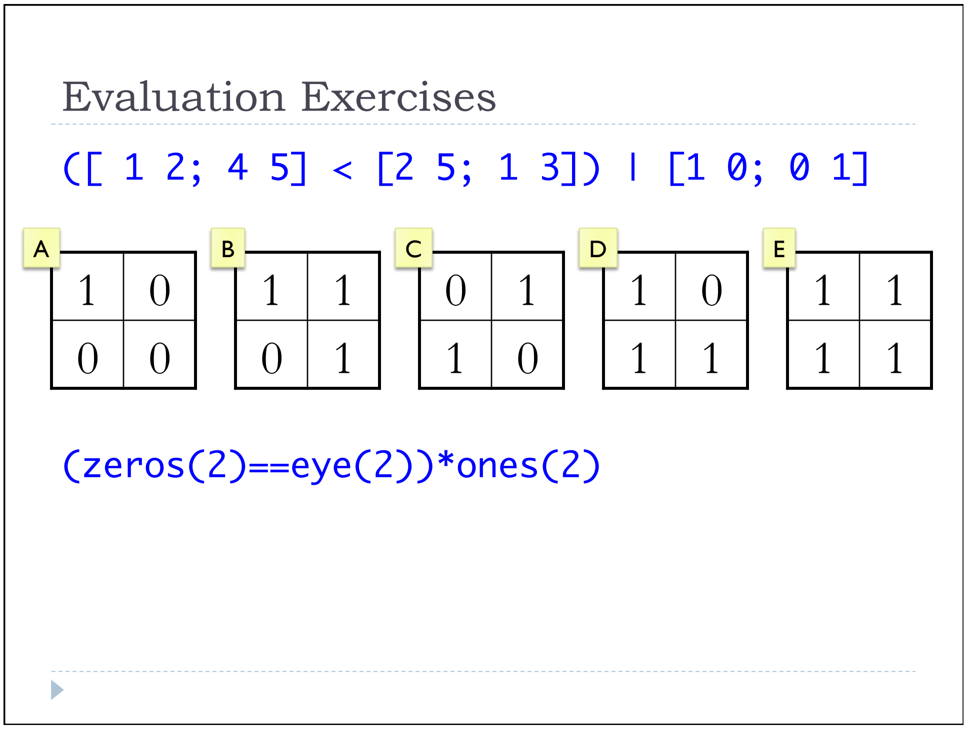
E. None of the above

p	q	xor(p,q)
false	false	0
false	true	1
true	false	1
true	true	0



Some Built-In Predicates

`ischar(a)`	true iff a is a character array
`isempty(a)`	true iff a is an empty array
`isinf(a)`	true iff the value of a is Inf (*infinity*)
`isnan(a)`	true if the value of a is NaN (*not a number*)
`isnumeric(a)`	true if a is a numeric array

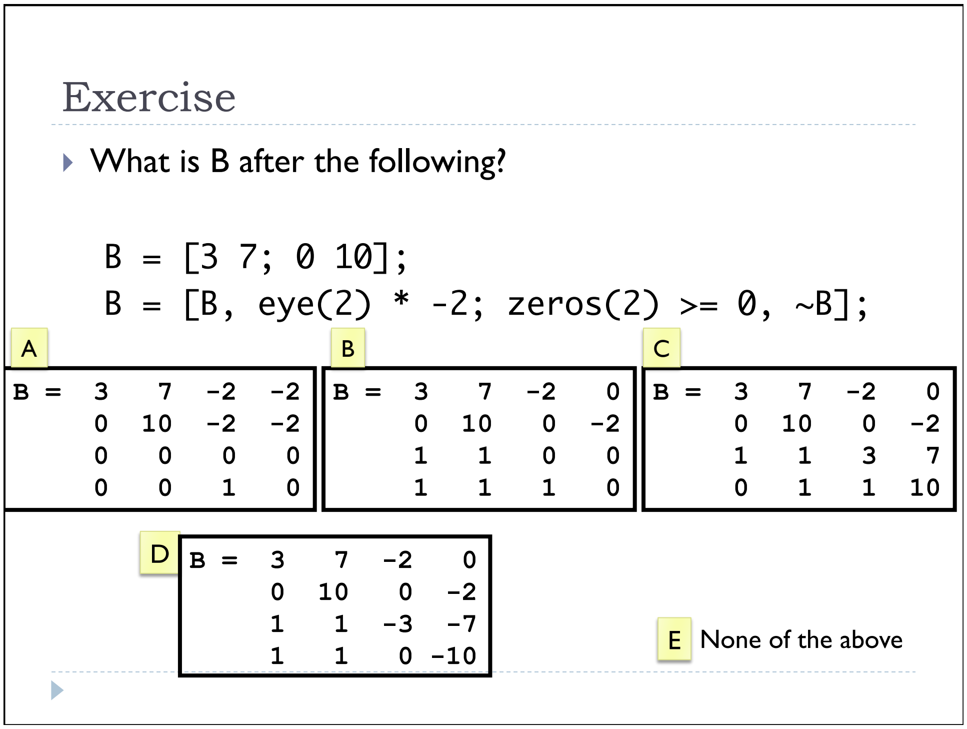


Evaluation Exercises

```
([ 1 2; 4 5] < [2 5; 1 3]) | [1 0; 0 1]
```

A $\begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 0 \\ \hline\end{array}$
B $\begin{array}{|c|c|}\hline 1 & 1 \\ \hline 0 & 1 \\ \hline\end{array}$
C $\begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & 0 \\ \hline\end{array}$
D $\begin{array}{|c|c|}\hline 1 & 0 \\ \hline 1 & 1 \\ \hline\end{array}$
E $\begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline\end{array}$

```
(zeros(2)==eye(2))*ones(2)
```



Exercise

- What is B after the following?

```
B = [3 7; 0 10];
B = [B, eye(2) * -2; zeros(2) >= 0, ~B];
```

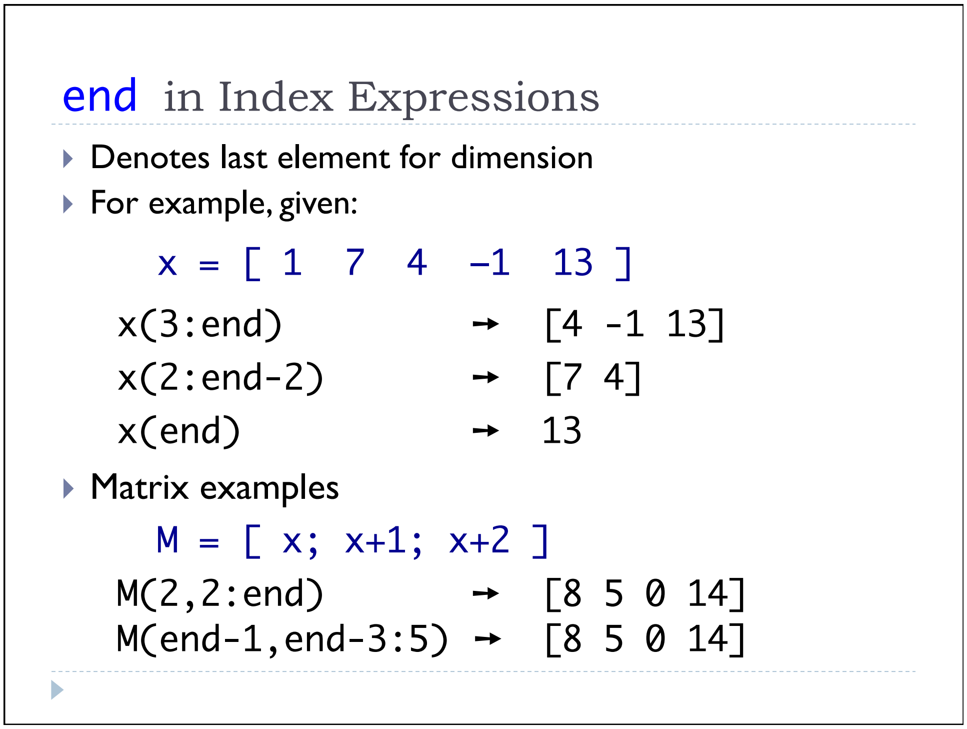
A
```
B =   3    7   -2   -2
      0   10   -2   -2
      0    0    0    0
      0    0    1    0
```

B
```
B =   3    7   -2    0
      0   10    0   -2
      1    1    0    0
      1    1    1    0
```

C
```
B =   3    7   -2    0
      0   10    0   -2
      1    1    3    7
      0    1    1   10
```

D
```
B =   3    7   -2    0
      0   10    0   -2
      1    1   -3   -7
      1    1    0  -10
```

E None of the above



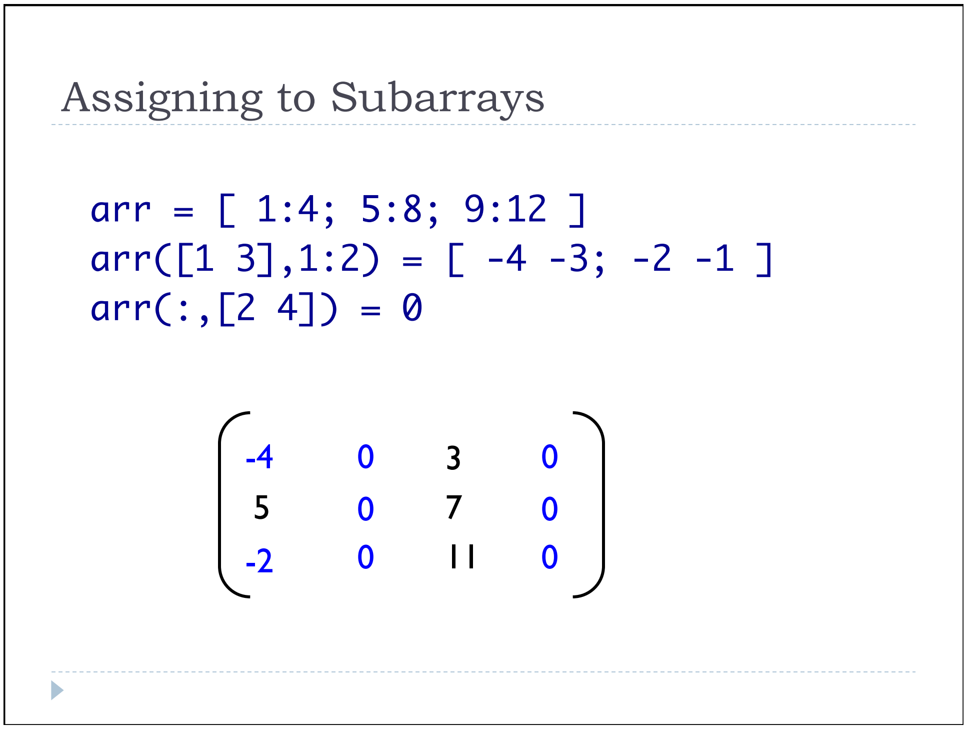
end in Index Expressions

- Denotes last element for dimension
- For example, given:

```
   x = [ 1  7  4  -1  13 ]
x(3:end)         →  [4 -1 13]
x(2:end-2)       →  [7 4]
x(end)           →  13
```

- Matrix examples

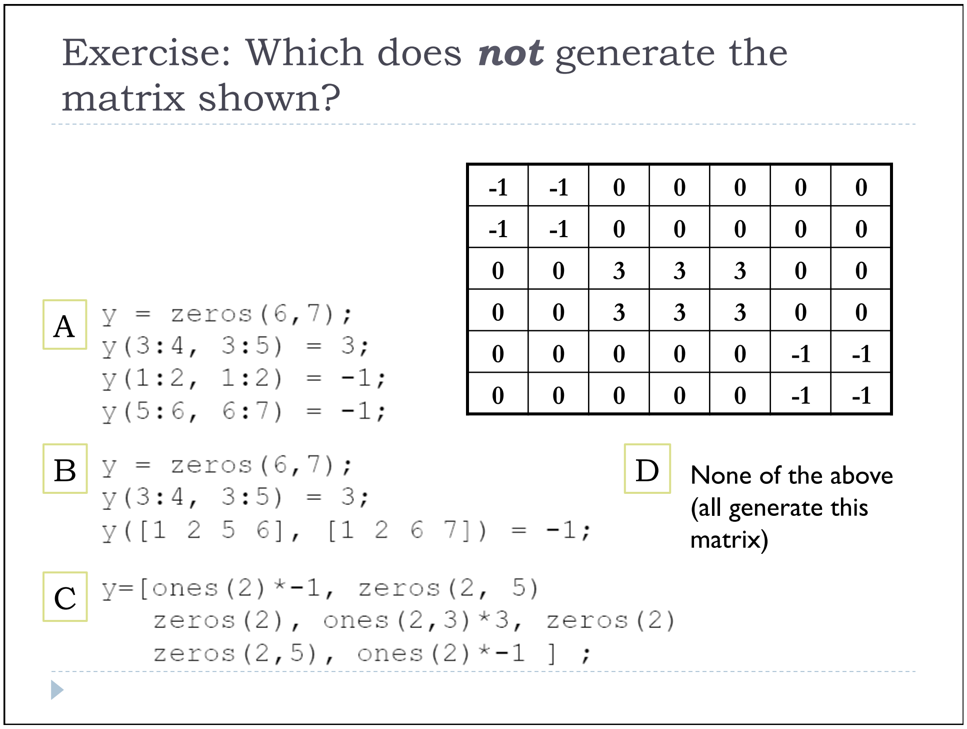
```
   M = [ x; x+1; x+2 ]
M(2,2:end)       →  [8 5 0 14]
M(end-1,end-3:5) →  [8 5 0 14]
```



Assigning to Subarrays

```
arr = [ 1:4; 5:8; 9:12 ]
arr([1 3],1:2) = [ -4 -3; -2 -1 ]
arr(:,[2 4]) = 0
```

$$\begin{pmatrix} -4 & 0 & 3 & 0 \\ 5 & 0 & 7 & 0 \\ -2 & 0 & 11 & 0 \end{pmatrix}$$



Exercise: Which does ***not*** generate the matrix shown?

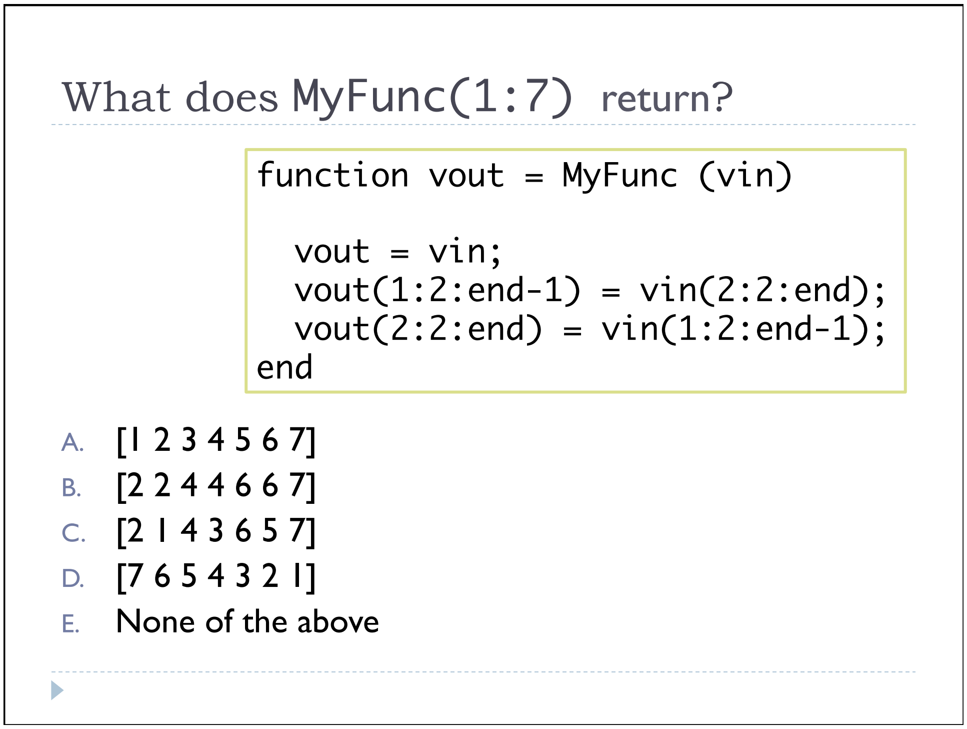
-1	-1	0	0	0	0	0
-1	-1	0	0	0	0	0
0	0	3	3	3	0	0
0	0	3	3	3	0	0
0	0	0	0	0	-1	-1
0	0	0	0	0	-1	-1

A
```
y = zeros(6,7);
y(3:4, 3:5) = 3;
y(1:2, 1:2) = -1;
y(5:6, 6:7) = -1;
```

B
```
y = zeros(6,7);
y(3:4, 3:5) = 3;
y([1 2 5 6], [1 2 6 7]) = -1;
```

C
```
y=[ones(2)*-1, zeros(2, 5)
    zeros(2), ones(2,3)*3, zeros(2)
    zeros(2,5), ones(2)*-1 ] ;
```

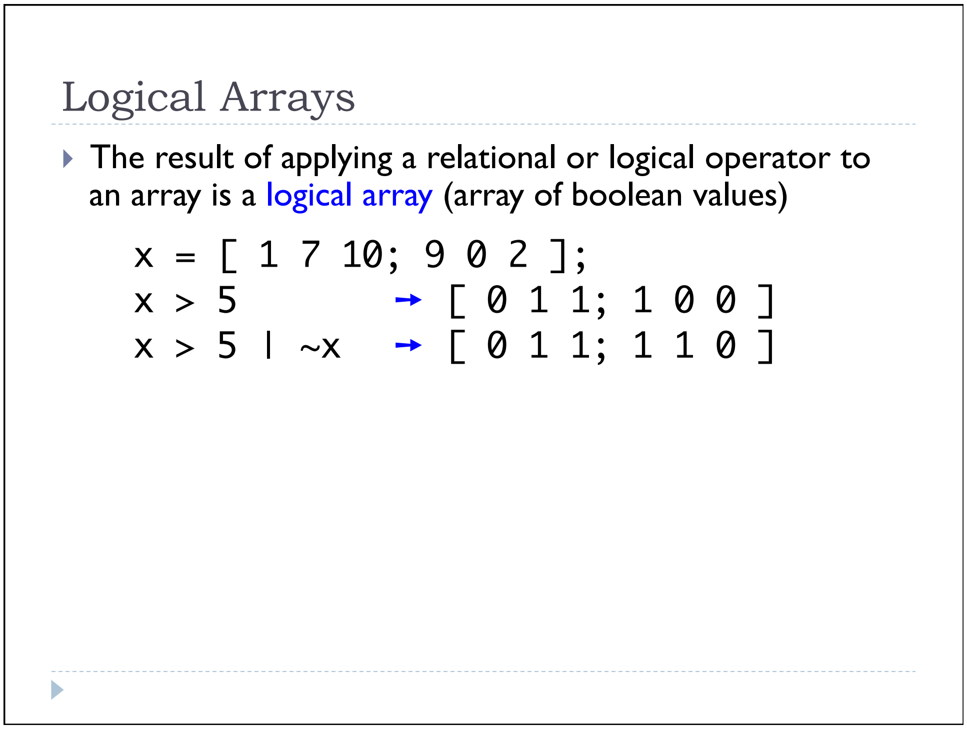
D None of the above (all generate this matrix)



What does MyFunc(1:7) return?

```
function vout = MyFunc (vin)

  vout = vin;
  vout(1:2:end-1) = vin(2:2:end);
  vout(2:2:end) = vin(1:2:end-1);
end
```

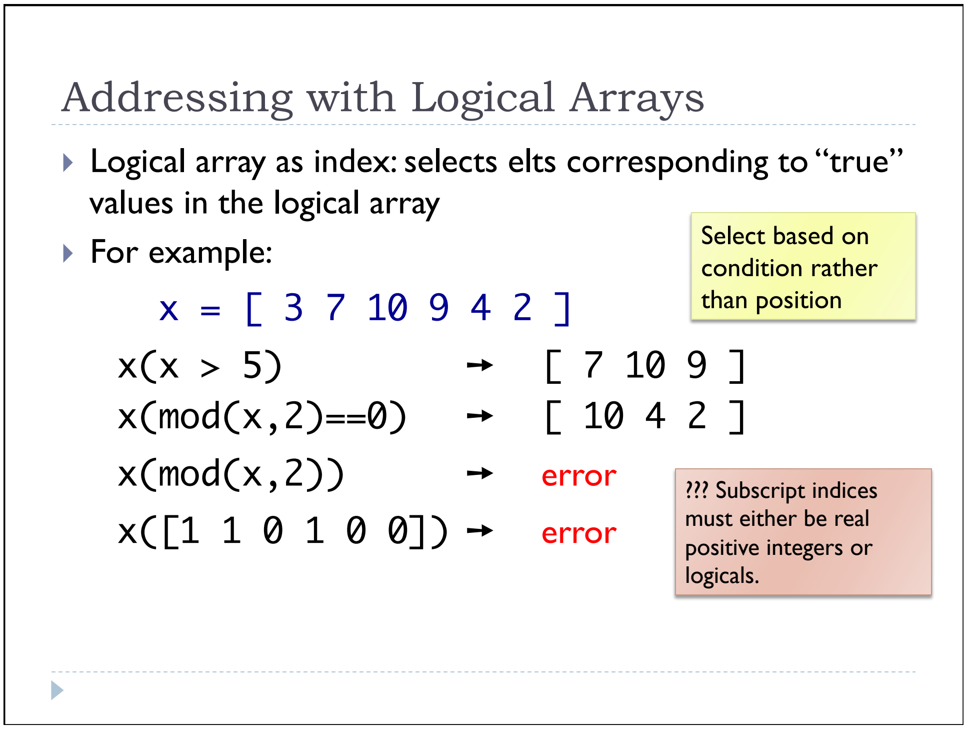
A. [1 2 3 4 5 6 7]
B. [2 2 4 4 6 6 7]
C. [2 1 4 3 6 5 7]
D. [7 6 5 4 3 2 1]
E. None of the above



Logical Arrays

- The result of applying a relational or logical operator to an array is a logical array (array of boolean values)

```
x = [ 1 7 10; 9 0 2 ];
x > 5          ➝ [ 0 1 1; 1 0 0 ]
x > 5 | ~x     ➝ [ 0 1 1; 1 1 0 ]
```



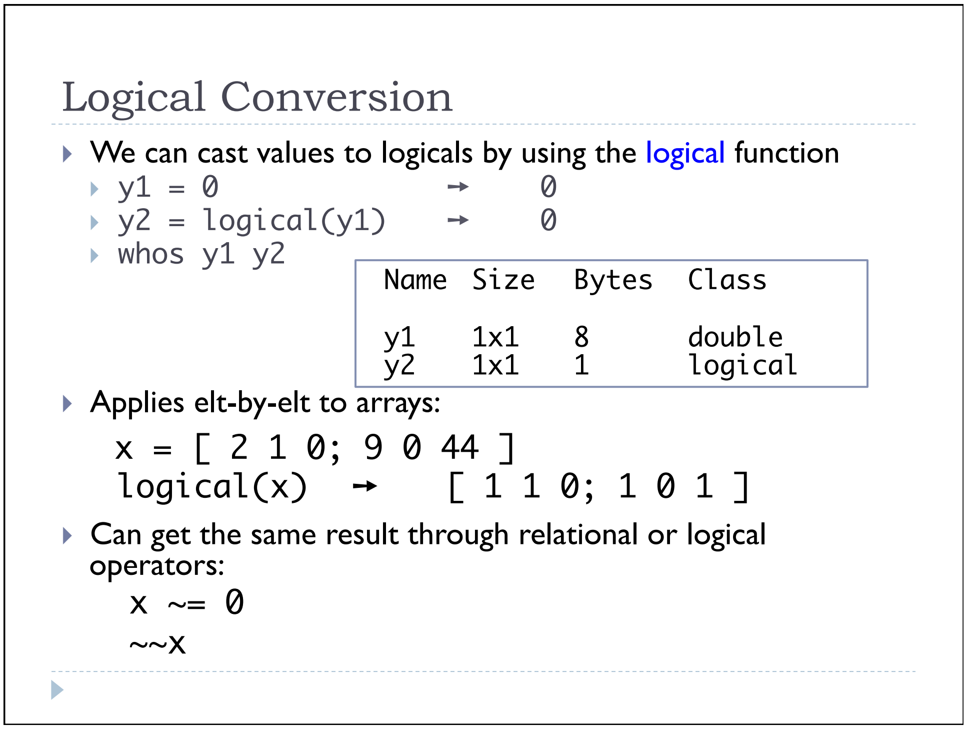
Addressing with Logical Arrays

- Logical array as index: selects elts corresponding to “true” values in the logical array
- For example:

```
  x = [ 3 7 10 9 4 2 ]
x(x > 5)           →   [ 7 10 9 ]
x(mod(x,2)==0)     →   [ 10 4 2 ]
x(mod(x,2))        →   error
x([1 1 0 1 0 0])   →   error
```

Select based on condition rather than position

??? Subscript indices must either be real positive integers or logicals.



Logical Conversion

- We can cast values to logicals by using the logical function
  - `y1 = 0            →    0`
  - `y2 = logical(y1)  →    0`
  - `whos y1 y2`

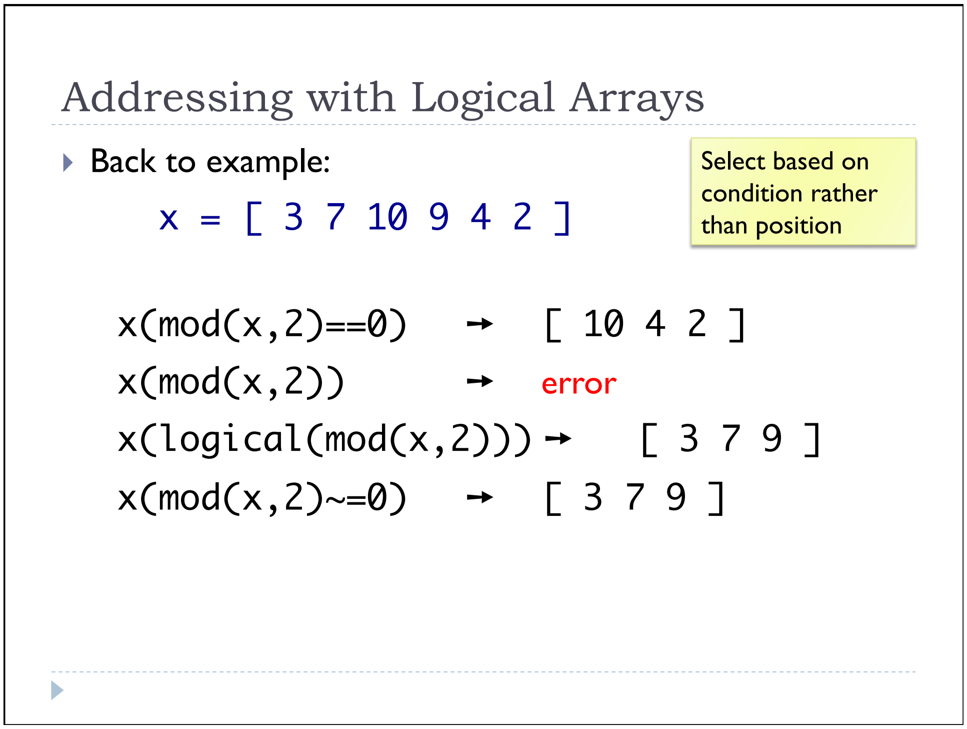
```
Name  Size   Bytes   Class

y1    1x1    8       double
y2    1x1    1       logical
```

- Applies elt-by-elt to arrays:

```
x = [ 2 1 0; 9 0 44 ]
logical(x)  →    [ 1 1 0; 1 0 1 ]
```

- Can get the same result through relational or logical operators:

```
x ~= 0
~~x
```



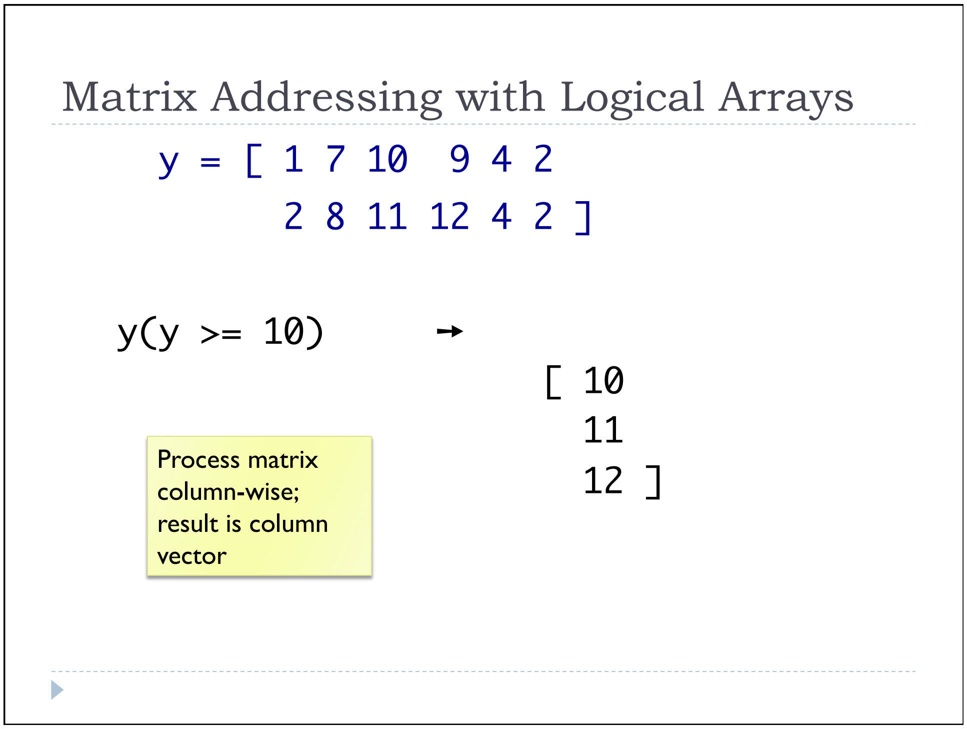
Addressing with Logical Arrays

- Back to example:

```
x = [ 3 7 10 9 4 2 ]
```

Select based on condition rather than position

```
x(mod(x,2)==0)     →   [ 10 4 2 ]
x(mod(x,2))        →   error
x(logical(mod(x,2)))→    [ 3 7 9 ]
x(mod(x,2)~=0)     →   [ 3 7 9 ]
```

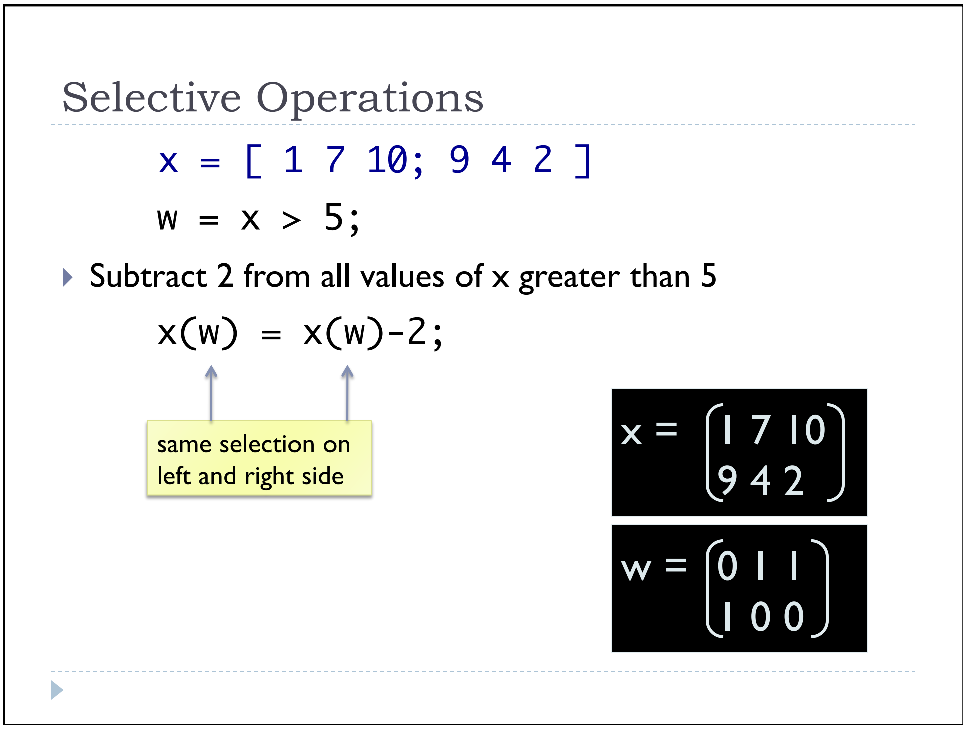


Matrix Addressing with Logical Arrays

```
y = [ 1 7 10  9 4 2
      2 8 11 12 4 2 ]
```

```
y(y >= 10)     →
                    [ 10
                      11
                      12 ]
```

Process matrix column-wise; result is column vector



Selective Operations

```
x = [ 1 7 10; 9 4 2 ]
w = x > 5;
```

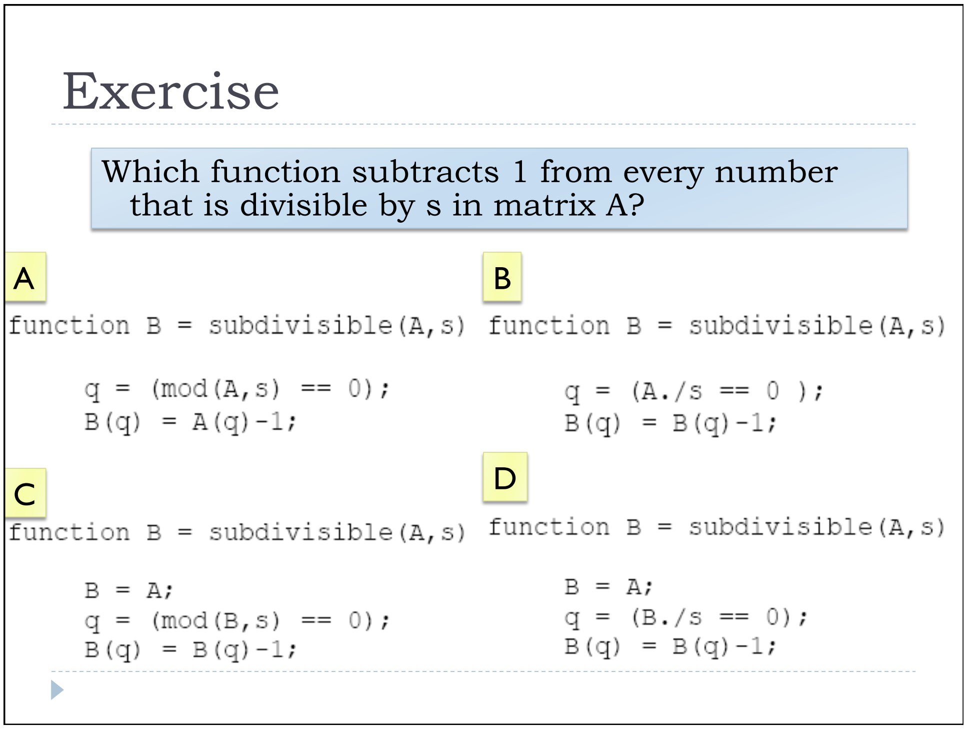
- Subtract 2 from all values of x greater than 5

```
x(w) = x(w)-2;
```

same selection on left and right side

$$x = \begin{pmatrix} 1 & 7 & 10 \\ 9 & 4 & 2 \end{pmatrix}$$

$$w = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$



Exercise

Which function subtracts 1 from every number that is divisible by s in matrix A?

A

```
function B = subdivisible(A,s)

    q = (mod(A,s) == 0);
    B(q) = A(q)-1;
```

B

```
function B = subdivisible(A,s)

    q = (A./s == 0 );
    B(q) = B(q)-1;
```

C

```
function B = subdivisible(A,s)

    B = A;
    q = (mod(B,s) == 0);
    B(q) = B(q)-1;
```

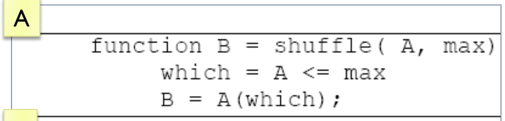
D

```
function B = subdivisible(A,s)

    B = A;
    q = (B./s == 0);
    B(q) = B(q)-1;
```

Exercise: Which MATLAB Function is Equivalent?

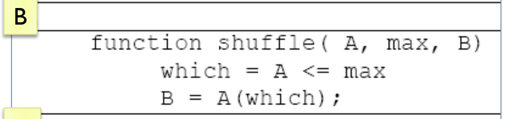
```
void shuffle (vector <int> A, int max,
              vector <int> & B)
{
  B.clear( );
  for (int i=0;
       i< A.size( ); i++)
    if (A[ i ] <= max)
      B.push_back(A[ i ]);
  return;
}
```

A



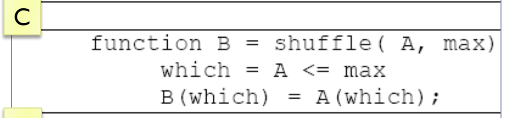
```
function B = shuffle( A, max)
     which = A <= max
     B = A(which);
```

B



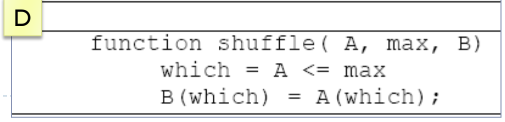
```
function shuffle( A, max, B)
     which = A <= max
     B = A(which);
```

C



```
function B = shuffle( A, max)
     which = A <= max
     B(which) = A(which);
```

D



```
function shuffle( A, max, B)
     which = A <= max
     B(which) = A(which);
```

More Selection using Logical Indexing

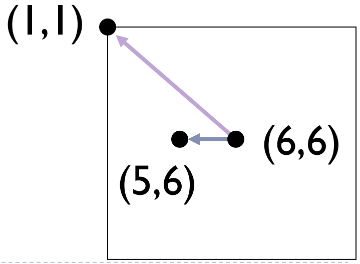
- Plot a sine wave, but clip negative values at zero

```
x = 0 : pi/50 : 4*pi;
f = sin(x);
f(f < 0) = 0;
plot(x, f);
```

Vectorized Programming

- Write a short program with no loops that:
  - creates an 11 x 11 matrix,
  - where cell (r,c) contains the distance of point (r,c) from the center location (6,6),
  - unless the distance is less than 2, in which case it should be replaced by 2.

e.g., M(1,1) = sqrt(25+25);
M(5,6) = 1 → 2;



(programming)

```
col = ones(11,1) * (-5:5);
row = col';
M = sqrt(row.^2 + col.^2);
```

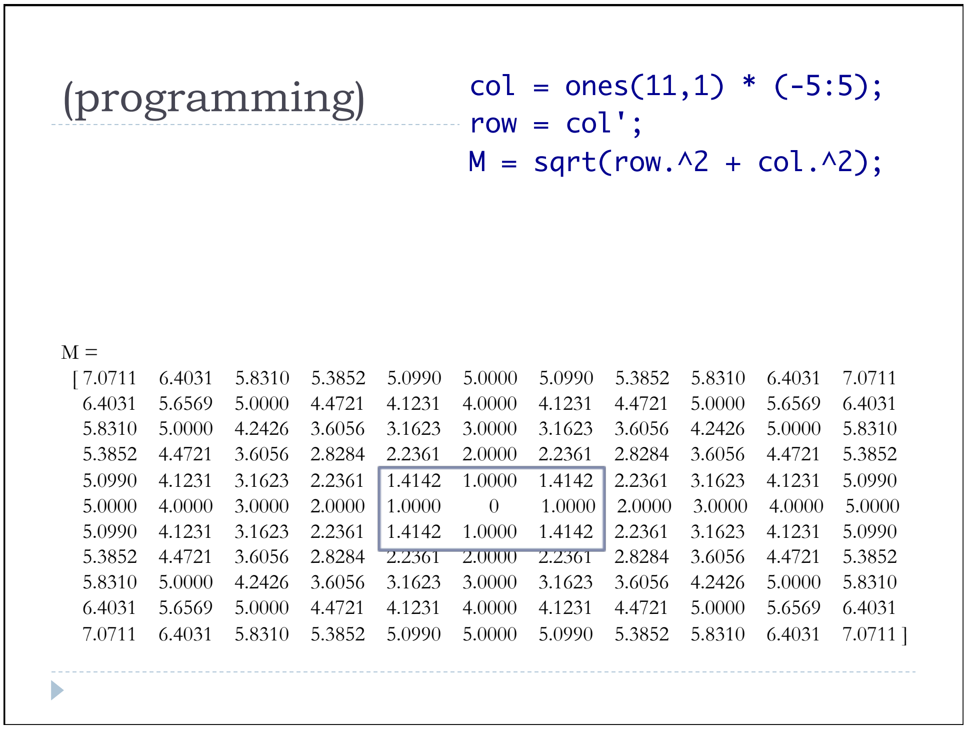
-5	-4	-3	-2	-1	0	1	2	3	4	5

col

-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5
-5	-4	-3	-2	-1	0	1	2	3	4	5

row

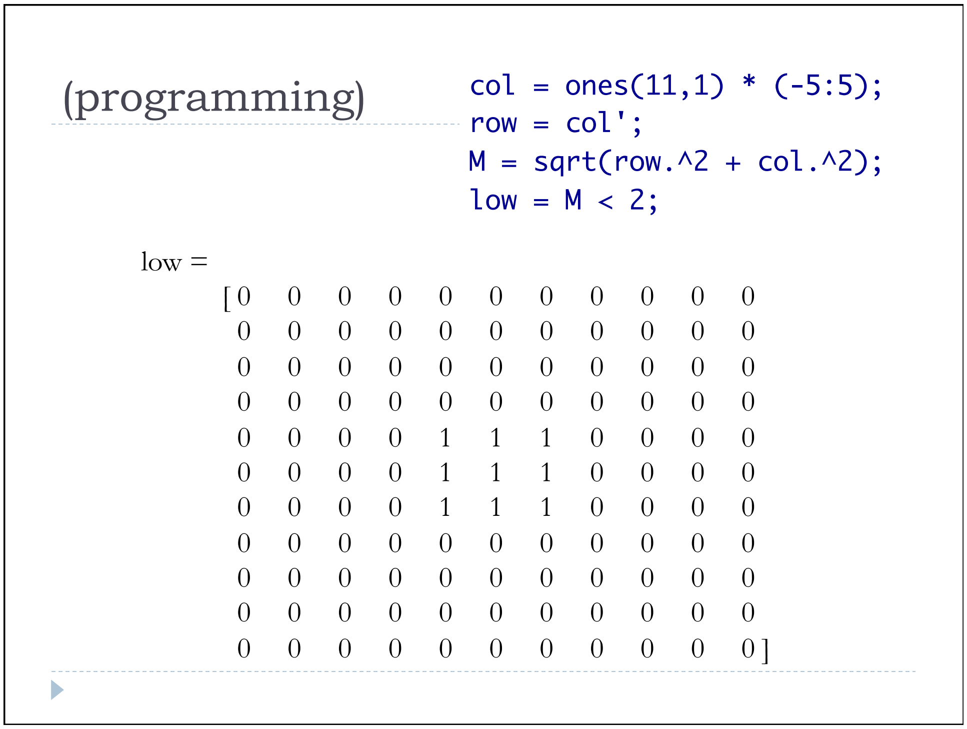
-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5
-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4
-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
-2	-2	-2	-2	-2	-2	-2	-2	-2	-2	-2
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5



(programming)

```
col = ones(11,1) * (-5:5);
row = col';
M = sqrt(row.^2 + col.^2);
```

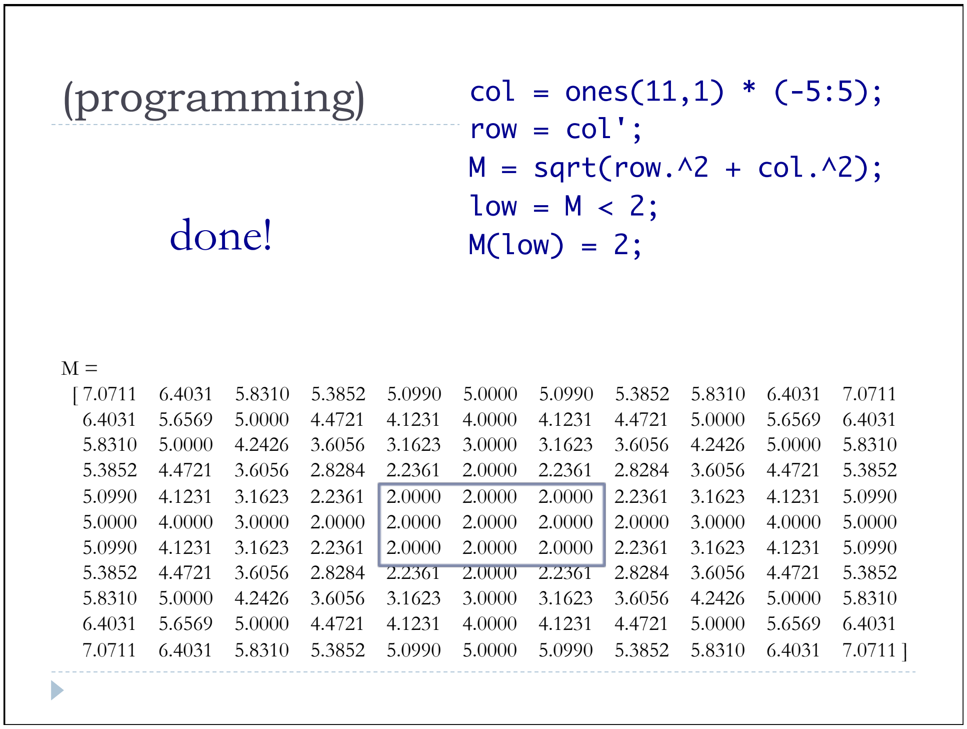
```
M =
 [ 7.0711   6.4031   5.8310   5.3852   5.0990   5.0000   5.0990   5.3852   5.8310   6.4031   7.0711
   6.4031   5.6569   5.0000   4.4721   4.1231   4.0000   4.1231   4.4721   5.0000   5.6569   6.4031
   5.8310   5.0000   4.2426   3.6056   3.1623   3.0000   3.1623   3.6056   4.2426   5.0000   5.8310
   5.3852   4.4721   3.6056   2.8284   2.2361   2.0000   2.2361   2.8284   3.6056   4.4721   5.3852
   5.0990   4.1231   3.1623   2.2361   1.4142   1.0000   1.4142   2.2361   3.1623   4.1231   5.0990
   5.0000   4.0000   3.0000   2.0000   1.0000        0   1.0000   2.0000   3.0000   4.0000   5.0000
   5.0990   4.1231   3.1623   2.2361   1.4142   1.0000   1.4142   2.2361   3.1623   4.1231   5.0990
   5.3852   4.4721   3.6056   2.8284   2.2361   2.0000   2.2361   2.8284   3.6056   4.4721   5.3852
   5.8310   5.0000   4.2426   3.6056   3.1623   3.0000   3.1623   3.6056   4.2426   5.0000   5.8310
   6.4031   5.6569   5.0000   4.4721   4.1231   4.0000   4.1231   4.4721   5.0000   5.6569   6.4031
   7.0711   6.4031   5.8310   5.3852   5.0990   5.0000   5.0990   5.3852   5.8310   6.4031   7.0711 ]
```



(programming)

```
col = ones(11,1) * (-5:5);
row = col';
M = sqrt(row.^2 + col.^2);
low = M < 2;
```

```
low =
    [ 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   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0
      0   0   0   0   1   1   1   0   0   0   0
      0   0   0   0   1   1   1   0   0   0   0
      0   0   0   0   1   1   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   0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0   0   0   0 ]
```

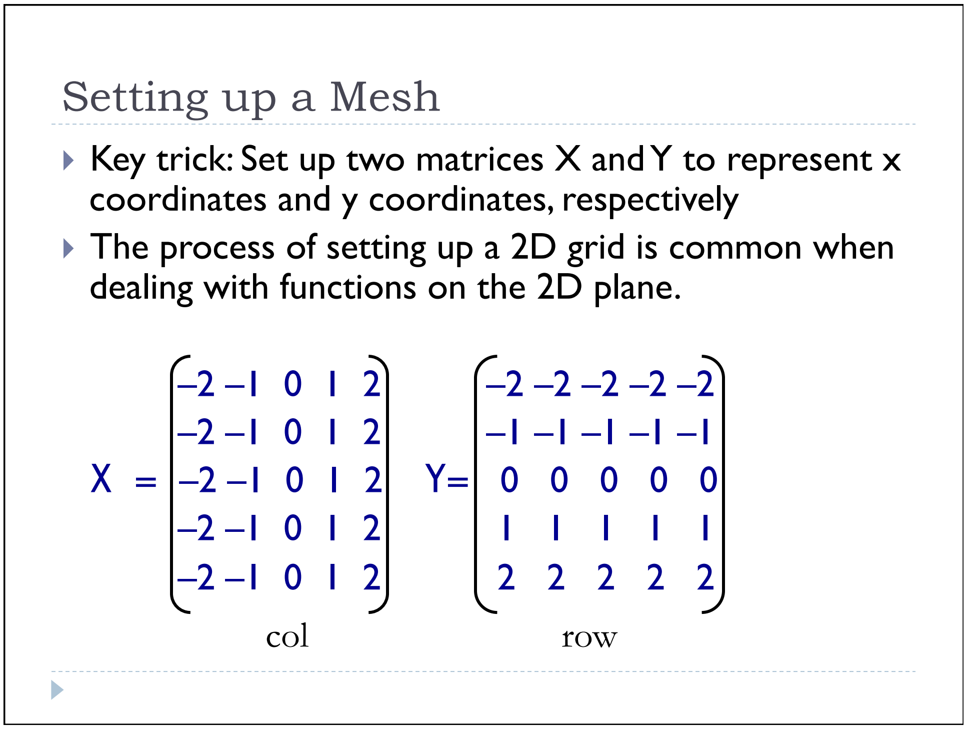


(programming)

done!

```
col = ones(11,1) * (-5:5);
row = col';
M = sqrt(row.^2 + col.^2);
low = M < 2;
M(low) = 2;
```

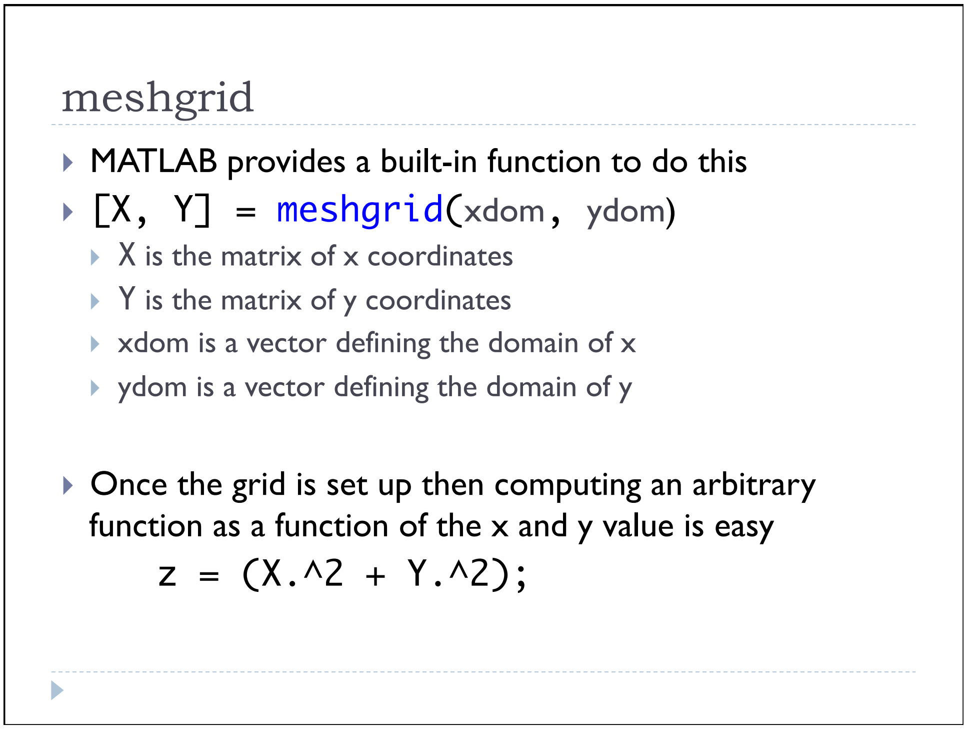
```
M =
 [ 7.0711  6.4031  5.8310  5.3852  5.0990  5.0000  5.0990  5.3852  5.8310  6.4031  7.0711
   6.4031  5.6569  5.0000  4.4721  4.1231  4.0000  4.1231  4.4721  5.0000  5.6569  6.4031
   5.8310  5.0000  4.2426  3.6056  3.1623  3.0000  3.1623  3.6056  4.2426  5.0000  5.8310
   5.3852  4.4721  3.6056  2.8284  2.2361  2.0000  2.2361  2.8284  3.6056  4.4721  5.3852
   5.0990  4.1231  3.1623  2.2361  2.0000  2.0000  2.0000  2.2361  3.1623  4.1231  5.0990
   5.0000  4.0000  3.0000  2.0000  2.0000  2.0000  2.0000  2.0000  3.0000  4.0000  5.0000
   5.0990  4.1231  3.1623  2.2361  2.0000  2.0000  2.0000  2.2361  3.1623  4.1231  5.0990
   5.3852  4.4721  3.6056  2.8284  2.2361  2.0000  2.2361  2.8284  3.6056  4.4721  5.3852
   5.8310  5.0000  4.2426  3.6056  3.1623  3.0000  3.1623  3.6056  4.2426  5.0000  5.8310
   6.4031  5.6569  5.0000  4.4721  4.1231  4.0000  4.1231  4.4721  5.0000  5.6569  6.4031
   7.0711  6.4031  5.8310  5.3852  5.0990  5.0000  5.0990  5.3852  5.8310  6.4031  7.0711 ]
```



Setting up a Mesh

- Key trick: Set up two matrices X and Y to represent x coordinates and y coordinates, respectively
- The process of setting up a 2D grid is common when dealing with functions on the 2D plane.

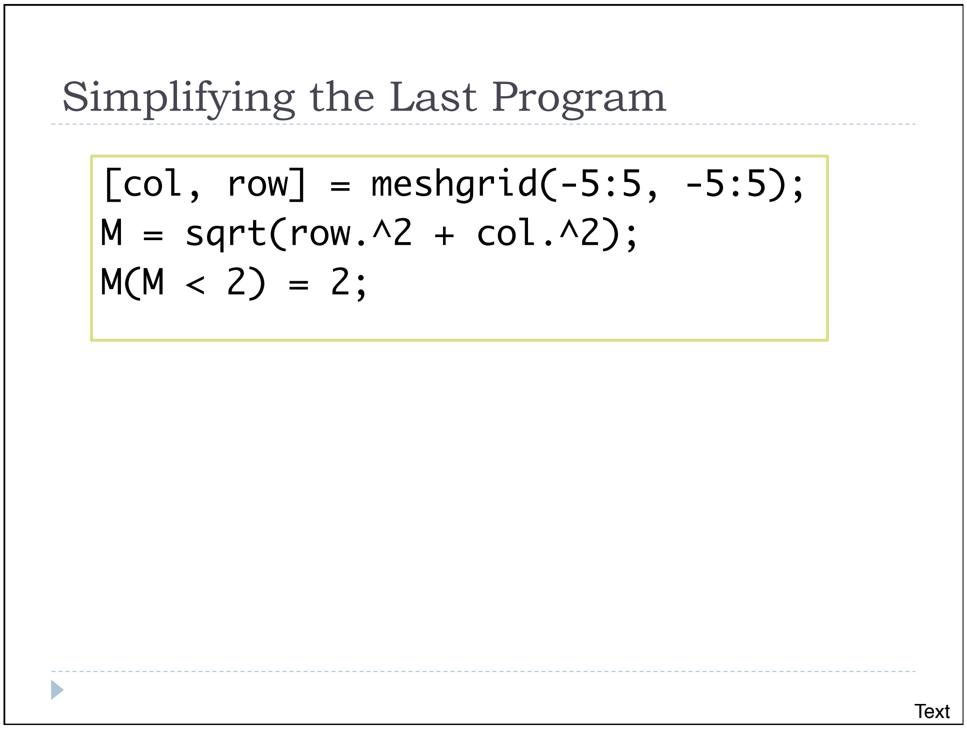
$$X = \underbrace{\begin{pmatrix} -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \end{pmatrix}}_{\text{col}} \quad Y = \underbrace{\begin{pmatrix} -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}}_{\text{row}}$$



meshgrid

- MATLAB provides a built-in function to do this
- `[X, Y] = meshgrid(xdom, ydom)`
  - X is the matrix of x coordinates
  - Y is the matrix of y coordinates
  - xdom is a vector defining the domain of x
  - ydom is a vector defining the domain of y
- Once the grid is set up then computing an arbitrary function as a function of the x and y value is easy

```
    z = (X.^2 + Y.^2);
```



Simplifying the Last Program

```
[col, row] = meshgrid(-5:5, -5:5);
M = sqrt(row.^2 + col.^2);
M(M < 2) = 2;
```

Text