Wichita State University
College of Engineering
Biomedical Engineering

Arrays cont..

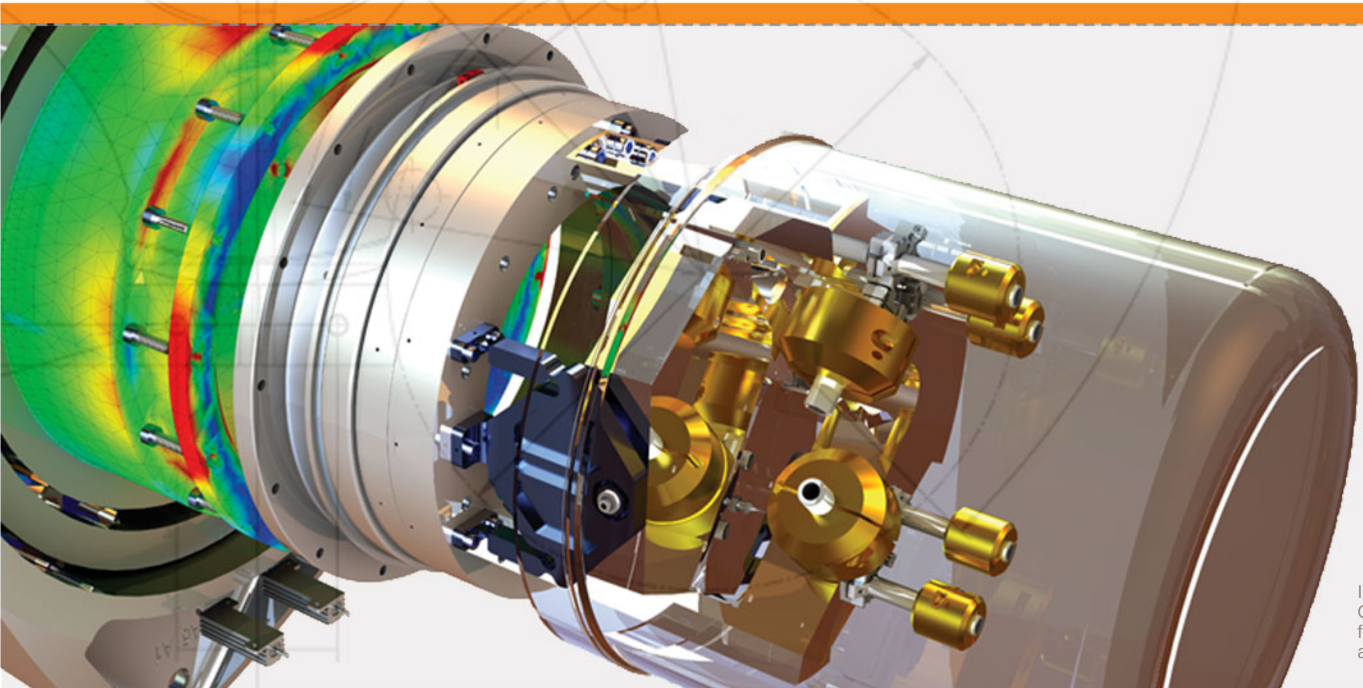


Image courtesy of National Optical Astronomy Observatory, operated by the Association of Universities for Research in Astronomy, under cooperative agreement with the National Science Foundation.

CREATING & MANIPULATING ARRAYS

Creating Arrays

```
>> a = [2.3  7.5  4.3  6]

a =

2.3000    7.5000    4.3000    6.0000


>> b = [2.3;  7.5;  4.3;  6]

b =    2.3000

       7.5000

       4.3000

       6.0000

>> c = 1:0.5:3

c = 1.0   1.5   2.0   2.5   3.0
```

```
>> d = [1.3  7.2  9.5  10; 2.6  5.1  4.7  8.1]


d =

    1.3000    7.2000    9.5000   10.0000
    2.6000    5.1000    4.7000    8.1000


>> e = [1.3  7.2  9.5; 2.6  5.1  4.7  8.1]

Error using vertcat
Dimensions of matrices being
concatenated are not consistent.


>> f = linespace (-1,3,5)

f =   -1   0   1   2   3
```

Note: Commas can be used between values in a row instead of spaces

Creating Arrays

FUNCTION	DESCRIPTION
A:INT:B	Creates a row vector of values starting at A, spaced by INT, terminating at B or just below B.
linspace(A, B, N)	Creates a row vector of N values equally spaced from A to B
eye(n,n)	Creates an n x n identity matrix
zeros(n,m)	Creates an n x m matrix of zeros
ones(n,m)	Creates an n x m matrix of ones

Caution: zeros(n) produces an n x n square matrix of zeros, not a vector of zeros
ones(n) produces an n x n square matrix of ones, not a vector of ones

Creating Arrays

FUNCTION	DESCRIPTION
randi([Imin,Imax], [n,m])	Creates an n x m matrix of integers uniformly distributed from Imin to Imax
randn(n,m)	Creates an n x m matrix of random numbers normally distributed with mean 0 and standard deviation of 1.
rand(n,m)	Creates an n x m matrix of random numbers uniformly distributed on the interval [0 1]
randperm(n,m)	Creates vector of m unique integers selected from the values 1, 2, … n

Note: The MATLAB command: `rng('shuffle')` seeds the random number generator based on the current time so that RAND, RANDI, and RANDN will produce a different sequence of numbers after each time you call rng.

Indexing Vectors

Suppose we create a vector:

```
>> x =  [–5    2    1    7    6   –3    2    4]
```

x is a vector with 8 values. To pull out a specific value or a set of values in x, we must index into the array, x.

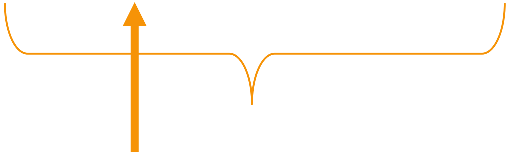
- In MATLAB®, array indexing starts with 1. This is different from many other programming languages including java, C, and C++ where array indexing starts with 0.
- x(1) is the 1st element in the vector x, x(2) is the 2nd element in the vector x, …

Indexing Vectors

- x(3:7) would be a vector with the 3rd through 7th entries of vector x. An error would be produced if x had less than 7 entries.
- To find out how many entries are in a vector x, use the command: >> N = length(x)

Example

```
>> x =  [-5    2    1    7    6   -3    2    4]
```



```
>> x(3)
```

MAX.

```
>> x(2:6)
```

START

```
>> x(1:2:8)

>> x( [1  6  8] )
```

```
>> x(3)      % 3rd entry
   ans =     1

>> x(2:6)    % entries 2 thru 6
   ans =  2    1    7    6   -3

>> x(1:2:8)  %entries 1, 3, 5 & 7
   ans =  -5    1    6    2

>> x( [1  6  8])   %entries 1, 6 & 8
   ans = -5    -3    4
```

Indexing Matrices

Assume A is a matrix defined in MATLAB that has at least 4 rows and at least 7 columns.

- A(3,5) is the entry in row 3, column 5 of matrix A.
- A(3,:) is a vector consisting of the entire 3rd row of matrix A. The **:** captures all columns.
- A(:,2) is a vector consisting of the entire 2nd column of matrix A. The **:** captures all the rows.
- A(2:4, 3:5) is a 3 x 3 matrix formed from rows 2,3, and 4 and columns 3,4, and 5 of matrix A.
- To determine how many rows and columns a matrix has, use the command: >> [rows,cols] = size(A)

Matrix Example

```
>> matrix = randi([0 10],[4,5])

matrix =

    7     7     5     2     7
    7     7    10     8     9
    8     1     3     2    10
    3     1     6     5     6

>> matrix(2,3)

>> matrix(:,4)

>> matrix(3,2:4)

>> matrix(2:3,3:5)
```

```
>> matrix(2,3)
ans =    10

>> matrix(:,4)
ans =
     2
     8
     2
     5

>> matrix(3,2:4)
ans =

     1     3     2

>> matrix(2:3,3:5)
ans =
    10     8     9
     3     2    10
```

Some Useful Functions for Indexing

FUNCTION	DESCRIPTION
size(A)	Gives the dimensions of array A. For matrix (2-d array), gives the number of rows and columns.
length(A)	Gives the largest dimension of array A. Most useful for vectors (one-dimensional arrays) since it will tell you how many elements are in the vector.

Looping thru a 1-d Array

What will the following code produce?

```
x = [-1  3  5  -7  53]
for k = 1:length(x)
     if x(k) < 0
          x(k) = 0;
     end
end
disp(‘x is now:’);
disp(x)
```

Looping thru a 1-d Array

```
x = [-1  3  5  -7  53]
for k = 1:length(x)
     if x(k) < 0
          x(k) = 0;
     end
end
disp(‘x is now:’);
disp(x)
```

```
k = 1      x(1) < 0?    Yes  Set x(1) = 0
x = [0  3  5  -7  53]

k = 2    x(2) < 0?      No
x = [0  3  5  -7  53]

k = 3    x(3) < 0?      No
x = [0  3  5  -7  53]

k = 4      x(4) < 0?    Yes  Set x(4) = 0
x = [0  3  5  0  53]

k = 5    x(5) < 0?      No
x is now:
    0    3    5    0   53
```

Looping thru 2-d Array

Need a nested loop for a matrix. What will this code produce?

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp('Matrix B =');disp(B)
```

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp('Matrix B =');disp(B)
```

r = 1 and c = 1

A(1,1) < 0? Yes

B(1,1) = 0

B = 0

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

r = 1 and c = 2
A(1,2) < 0? No
A(1,2) > 5? No
B(1,2) = A(1,2)

B = 0   3

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

r = 1 and c = 3

A(1,3) < 0? No

A(1,3) > 5? No

B(1,3) = A(1,3)

B = 0   3   5

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

r = 2 and c = 1
A(2,1) < 0? No
A(2,1) > 5? Yes
B(2,1) = 5

```
B =     0   3   5
        5   0   0
```

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

r = 2 and c = 2

A(2,2) < 0?   Yes

B(2,2) = 0

```
B =     0   3   5
        5   0   0
```

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp('Matrix B =');disp(B)
```

r = 2 and c = 3

A(2,3) < 0?   No

A(2,3) > 5?   No

B(2,3) = A(2,3)

```
B =     0   3   5
        5   0   3
```

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

r = 3 and c = 1
A(3,1) < 0?   No
A(3,1) > 5?   No
B(3,1) = A(3,1)

```
B =    0   3   5
       5   0   3
       4   0   0
```

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

r = 3 and c = 2

A(3,2) < 0?   Yes

B(3,2) = 0

```
B =     0   3   5
        5   0   0
        4   0   0
```

Looping thru 2-d Array

```
A = [-1 3 5; 6 -4 3; 4 -2 10]
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

r = 3 and c = 3
A(3,3) < 0?   No
A(3,3) > 5?   Yes
B(3,3) = 5

```
Matrix B =
       0   3   5
       5   0   3
       4   0   5
```

Pre-Allocating Memory Space

- In the previous program, the matrix B was created within the loops one entry at a time and therefore got larger with every cycle through the loop. If an array grows in size during the execution of a program, MATLAB must keep re-allocating space for the array which can be time-consuming.
- If you know ahead of time how big your array will be, you can avoid this problem by pre-allocating space for the array and filling the space with some numbers (often zeros).

Pre –Allocating Space

```
A = [-1 3 5; 6 -4 3; 4 -2 5]
B = zeros(size(A));    % Create a matrix B of 0s
[TotalRows TotalCols] = size(A);
for r = 1:TotalRows
      for c = 1:TotalCols
            if A(r,c) < 0
                  B(r,c) = 0;
            elseif A(r,c) > 5
                  B(r,c) = 5;
            else
                  B(r,c) = A(r,c);
            end
      end
end
disp(‘Matrix B =’);disp(B)
```

LOGIC AND RELATIONAL OPERATIONS WITH ARRAYS

Logic and Relational Operations

FUNCTION	DESCRIPTION
A == B	Entry by entry check for aij == bij. Produces a 1 if aij == bij and 0 otherwise.
A > B	Entry by entry check for aij > bij. Produces a 1 if aij > bij and 0 otherwise.
A < B	Entry by entry check for aij < bij. Produces a 1 if aij < bij and 0 otherwise.
A & B	Entry by entry logical **and** operation: aij & bij Produces a 0 if either entry is 0 (false). Otherwise, produces a 1.
A | B	Entry by entry logical **or** operation: aij | bij Produces 0 if both entries are 0 (false). Otherwise, produces a 1.

A and B must have the same dimensions unless one is a scalar
Remember, 0 = FALSE and Non-Zero = TRUE

Examples

```
>> A = randi ( [-10  10] , [4  3] )

A =
     7      3     10
     9     -8     10
    -8     -5     -7
     9      1     10

>> A > 5

ans =

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

Examples

```
>> A = [-1 5; 3 4]
A =
    -1     5
     3     4

>> B = [2 10; 3 1]
B =
     2    10
     3     1

>> A == B
ans =
     0     0
     1     0
```

Examples

```
>> A = [-1 5; 3 4]
A =
    -1     5
     3     4

>> B = [2 10; 3 1]
B =
     2    10
     3     1

>> A > B
ans =
     0     0
     0     1
```

Examples

```
>> A = [-1 5; 3 4]
A =
    -1      5
     3      4

>> B = [2 10; 3 1]
B =
     2     10
     3      1

>> A >= B
ans =
     0      0
     1      1
```

STRINGS
&
CELL ARRAYS

Indexing into a String

Each letter in a string is stored as a separate entry in a regular array:

```
>> name='JohnSmith'
name = JohnSmith

>> name(1)
ans = J

>> name(2)
ans =o

>> name(10)
??? Index exceeds matrix dimensions.
```

Cell Arrays

Regular Arrays do not work well at all for strings! Use Cell Arrays instead.

Cell arrays work extremely well for handling strings and for handling mixed data types.

For cell arrays, use curly braces { } rather than square brackets [ ] to enter the array. Other than that, indexing works exactly the same.

Cell Array Example

```
>> Months = { 'January'  'February'  'March'  'April'  'May'}

Months =

    'January'    'February'    'March'    'April'    'May'

>> Months(1)

ans = 'January'

>> Months(5)

ans = 'May'
```

Cell Array Example (con’t)

```
>> strcmp(Months,'February')    % String Comparison
ans =     0     1     0     0     0


>> strcmp(Months,'february')    % Case Sensitive
ans =     0     0     0     0     0


>> strcmpi(Months,'february')   % Case In-sensitive
ans =     0     1     0     0     0


>> strncmp(Months,'Feb',3)      % Compare 1st 3 letters
ans =     0     1     0     0     0
```

Useful Function: Find

find - goes through an array and finds the index of all entries satisfying some specified condition.

Example:

```
>> C = [4  3  2  1  8  9  1]

C =   4     3     2     1     8     9     1

>> LocationsOf3 = find(C==3)

LocationsOf3 =      2

>> LocationsOf1 = find(C==1)

LocationsOf1 =  4      7
```

```
>> GreaterThan3 = find(C > 3)

GreaterThan3  =   1      5      6

>> find(C > 2 & C < 9)

ans =  1      2      5
```

Using Max and Find

```
>> vector = randi([-5 10],[1,8])
vector =
   -5     2     1     7     7    -3     2     2

>> highest = max(vector)
highest =     7

>> location = find(vector == highest)
location =
     4     5
```

Note: Now we get both locations for maximum value

Structure Arrays

- These are Arrays composed of *structures*
- Allows you to store dissimilar arrays together
- Different to Cell Arrays as elements in structure are accessed using *named fields* as opposed to { }
- Basically like a database structure
- Example:

  Create a structure for student data with

  - Student Name
  - ID
  - Email address
  - Test scores

Structure Example

```
student.name = 'Gary Brooking';
student.ID = ‘ABC123’;
student.email = ‘gary@Wichita.edu’;
student.scores = [ 75 85 95]

>>student

  name: ‘Gary Brooking’
       ID: ‘ ABC123’
  email: ‘ gary@Wichita.ed’
scores: [75 85 95]
```

Structure Example

Or you can use the *struct* function:

```
student = struct (‘name’, 'Gary Brooking‘, ‘ID’, ‘ABC123’,
‘email’, ‘gary@Wichita.edu’, ‘scores’, [ 75 85 95] )
```

```
>>student

name: ‘Gary Brooking’
    ID: ‘ ABC123’
 email: ‘ gary@Wichita.ed’
scores: [75 85 95]
```

Structure Example

To add a second structure:

```
student(2).name = ‘Jane Doe';
student(2).ID = ‘XYZ789’;
student(2).email = ‘jane@Wichita.edu’;
student(2).scores = [ 78 92 94]
```

You can add to the database:

```
student(1).phone = ‘555-5555';
```

All the other structure will now have “phone” but empty

Comments

- Regular arrays are very convenient for doing many different types of numerical computations and for many programming applications.
- All entries in a regular array must be of the same type. You cannot have some entries that are doubles and some entries that are integers (type uint8 for examples).
- A cell array allows for mixed data types of varying lengths. It is useful for strings and for mixing strings and numbers.

Your Turn

Try the following commands in MATLAB

```
>> x = [-1  5  7  4  3 ]
>> x(3)
>> x(10)
>> x(3:5)
>> x([1 5])
>> N = length(x)

>> x(2:4) = [1 2 3];
```

Your Turn

Try the following commands in MATLAB

```
>> A = [-1  5  7  4; 3   6  10  13; 4  -17  12  15 ]
>> A(2,3)
>> A(1,5)
>> A(1,:)
>> A(:,3)
>> A(2:3,1:2)
>> [Rows Cols] = size(A)

>> A(2,3:4) = [0 0]
```

Your Turn

Try the following commands in MATLAB

```
>> A = [-1  5  7;  3   6  10 ]
>> B = [ 2  3  4; 12  6  10 ]
>> A > 0
>> A == 0
>> A == B
>> A > B
```