

EGR 106 – Week 3 – More on Arrays

- Brief review of last week
- Additional ideas:
 - Special arrays
 - Changing an array
 - Some array operators
 - Character arrays
- Element by element math operations
- Textbook – rest of chapter 2, 3.1, 3.4, 3.5

Review of Last Week

- The fundamental data unit in Matlab
 - Rectangular collection of data
 - All variables are considered to be arrays

$$\text{yield} = \begin{bmatrix} 4 & 5 & 3 & 9 \\ 10 & 4 & 66 & 20 \\ 18 & -3 & 2 & 0 \end{bmatrix}$$

- Data values organized into **rows** and **columns**

- Size of an array
- Construction:
 - Brute force using brackets
 - Concatenation of other arrays – L/R and U/D
 - The colon operator
- Addressing:
 - Individual elements
 - Subarrays

- Some new tricks:
 - **end** specifies the last row or column
 - a colon (:) specifies all of a row or column

$$\text{yield} = \begin{bmatrix} 4 & 5 & 3 & 9 \\ 10 & 4 & 66 & 20 \\ 18 & -3 & 2 & 0 \end{bmatrix}$$

yield(end,1) points to 18
yield(2,:) points to the second row [10 4 66 20]

- use a **single** index for row and column vectors

$$\text{bob} = [9 \ 7 \ 5 \ 7 \ 2]$$

bob(4) points to 7

Special Arrays

Square versions
↓

- Special predefined arrays:
 - all zeros **zeros(R,C)** **zeros(N)**
 - all ones **ones(R,C)** **ones(N)**
 - zeros with ones on the diagonal **eye(R,C)** **eye(N)**
 - random numbers (within [0 1]) **rand(R,C)** **rand(N)**

```
Command Window
>> eye(3,4)

ans =

     1     0     0     0
     0     1     0     0
     0     0     1     0

>> rand(2,5)
random on [ 0, 1 ]

ans =

    0.9501    0.6068    0.8913    0.4565    0.8214
    0.2311    0.4860    0.7621    0.0185    0.4447
```

- Linspace – like the colon operator, but **definitely** gets the last number on the list

`linspace (start, last, number of values)`

- examples:

`linspace(0,10,6)` → [0 2 4 6 8 10]

`linspace(0,1,4)` → [0 0.333 0.667 1]

- default for number is 100

`linspace(0,10)` → [0 0.101 0.202 ... 10]

Note increment for 100 points, 99 intervals

Changing an Array

Recall reading an array value:

```
test =
    0.4565    0.8214    0.6154
    0.0185    0.4447    0.7919

>> x = test(1,3)

x =
    0.6154
```

- To **change** a single value in an array

- Use addressing on the **left hand side of =**

```
test =
    0.4565    0.8214    0.6154
    0.0185    0.4447    0.7919

>> test(1,2) = 5

test =
    0.4565    5.0000    0.6154
    0.0185    0.4447    0.7919
```

- Can also be used to change a sub-array

```
test =
    0.9218    0.1763    0.9355
    0.7382    0.4057    0.9169

>> test([1 2],1) = zeros(2,1)

test =
    0    0.1763    0.9355
    0    0.4057    0.9169
```

Note – array on the right needs to be of the correct size !!!

- Watch for addressing errors:

```
test =
    0.4103    0.0579    0.8132
    0.8936    0.3529    0.0099

>> test(2,:) = [ 1 2 ]
Wrong number of elements for the assignment
??? In an assignment A(matrix,:) = B, the number must be the same.

>> test(:,2) = [ 1 2 ]
Wrong dimensions for assignment
??? In an assignment A(:,matrix) = B, the number of columns in B must be the same.
```

- Other useful methods that do work:

```
test =
    0.8462    0.6721    0.6813    0.5028
    0.5252    0.8381    0.3795    0.7095
    0.2026    0.0196    0.8318    0.4289

>> test(4,2) = 100
Assigning values with too large an index just grows the array

test =
    0.8462    0.6721    0.6813    0.5028
    0.5252    0.8381    0.3795    0.7095
    0.2026    0.0196    0.8318    0.4289
    0 100.0000    0 0
```

```
test =
    0.8462    0.6721    0.6813    0.5028
    0.5252    0.8381    0.3795    0.7095
    0.2026    0.0196    0.8318    0.4289

>> test([2 3],[2 3]) = 7

test =
    0.8462    0.6721    0.6813    0.5028
    0.5252    7.0000    7.0000    0.7095
    0.2026    7.0000    7.0000    0.4289
```

Scalars work for sub-array replacement – they just scale up to the right size

```
test =
    0.8462    0.6721    0.6813    0.5028
    0.5252    0.8381    0.3795    0.7095
    0.2026    0.0196    0.8318    0.4289

>> test(:,2) = []

test =
    0.8462    0.6813    0.5028
    0.5252    0.3795    0.7095
    0.2026    0.8318    0.4289
```

Replacing with a null matrix is the same as deleting – but it only works for entire rows or columns

Some Array Operators

- **Transpose** (single quote symbol ')
 - switches rows and columns

```
test =
    1     2     3     4
    5     6     7     8
    9    10    11    12

>> test'

ans =
    1     5     9
    2     6    10
    3     7    11
    4     8    12
```

- **Size** – the number of rows and columns
- **Length** – the larger of these two

```
test = [ 4 5 3
        10 4 66 ]
bob = [ 5 7 3 6 ]

>> size(test)
ans =
    2     3

>> size(bob)
ans =
    1     4

>> length(test)
ans =
    3

>> length(bob)
ans =
    4
```

Array answer

Scalar answer

- **Diag** – matrix ↔ vector operator for diagonal elements

```
>> diag([ 1 2 3 ])

ans =
    1     0     0
    0     2     0
    0     0     3

test =
    0.6449    0.3420    0.5341    0.8385
    0.8180    0.2897    0.7271    0.5681
    0.6602    0.3412    0.3093    0.3704

>> diag(test)

ans =
    0.6449
    0.2897
    0.3093
```

- **Reshape**
 - resize fixed set of elements

```
test =
    1     2     3     4
    5     6     7     8
    9    10    11    12

>> reshape(test,2,6)

ans =
    1     9     6     3    11     8
    5     2    10     7     4     12
```

Note – the set of elements is exactly the same

Note column-wise approach to re-ordering the values

Character Arrays

- Rows of the array are strings of alphanumeric characters, **one array entry per character**
- Enter using a single quotation mark (') at each end of the string

```
>> test = 'John'
test =
John
>> size(test)
ans =
     1     4
```

- For multi-row alphanumeric arrays, each row must have the same number of characters

```
name = [ 'Marty'; 'James'; 'Bob' ]
```

Note – need 2 spaces

- Use 2 quotation marks in a row to get 1 'Mary"s' → Mary's (a 1 by 6 array)
- Also, there are some built-in arrays
y = date → y = 05-Jan-2004 (a 1 by 11 array)

Element by Element Math Operations

- For arrays of identical sizes, addition is defined **term by term**:
 - the command F = A + B means

$$F(r,c) = A(r,c) + B(r,c)$$

- for all row and column pairs r,c **"element-by-element" addition**

- For example:

```
A =      B =      >> F = A + B
  1  2      7  8      F =
  3  4      9 10      8 10
  5  6     11 12     12 14
                          16 18
```

- Notes:

- Arrays must be of **identical** sizes
- One can be a scalar (it is "sized up")
- Subtraction is identical

- The other basic math operations work element by element using the **dot** notation (with A,B the **same** sizes):

- multiplication

$$F = A .* B \rightarrow F(r,c) = A(r,c) * B(r,c)$$

- division

$$F = A ./ B \rightarrow F(r,c) = A(r,c) / B(r,c)$$

- exponentiation:

$$F = A .^ B \rightarrow F(r,c) = A(r,c) ^ B(r,c)$$

note periods!

- For example:

```
a =      b =
  1  2  3      4  5  6
>> a.*b
ans =
  4 10 18      >> a.^b
ans =
  1 32 729
>> a./b
ans =
  0.2500  0.4000  0.5000
```

- One could be scalar: $a = [1\ 2\ 3]$ $b = 2$

```
>> a.*b
ans =
     2     4     6

>> a./b
ans =
    0.5000    1.0000    1.5000

>> a.^b
ans =
     1     4     9

>> b.*a
ans =
     2     4     6

>> b./a
ans =
    2.0000    1.0000    0.6667

>> b.^a
ans =
     2     4     8
```

- Built-in functions also work **element-by-element**:

```
>> b = [ 4 9 25; 1 2 10 ]
b =
     4     9    25
     1     2    10

>> sqrt(b)
ans =
    2.0000    3.0000    5.0000
    1.0000    1.4142    3.1623
```