Today’s Objectives

- Creating and Using Matlab Arrays
- Example using simple array of wildfire statistics
- Download files from github and look at wildfires.csv in the data subdirectory

A *string* is an array of characters

Strings have many uses in MATLAB

- Display text output
- Specify formatting for plots
- Input arguments for some functions
- Text input from user or data files

- Create a string by typing characters within single quotes (')
  - Many programming languages use the quotation mark (") for strings. Not MATLAB
- Can have letters, digits, symbols, spaces
  - To type single quote in string, use two consecutive single quotes, e.g., make the string of English "Greg's car" by typing `'Greg''s car'`
  - Examples: `'ad ef'`, `'3%fr2'`, `'edcba:21!'`, `'MATLAB'`

In a string variable

- Numbers are stored as an array
- A one-line string is a row vector
    - Number of elements in vector is number of characters in string

```
>> name = 'Howard the Duck';
>> size( name )
ans =
   1 15
```

MATLAB stores strings with multiple lines as an array. This means each line must have the same number of columns (characters)

```
>> names = [ 'Greg'; 'John' ]

names =

    Greg

    John

>> size( names )

ans =

    2  4
```

An *array* is MATLAB's basic data structure

- Can have any number of dimensions. Most common are
  - *vector* - one dimension (a single row or column)
  - *matrix* - two or more dimensions
- Arrays can contain numbers or letters
- *Number of columns must be the same in each row of matrix*
- *Overall, Matlab is very similar to Numpy in Python.*

Type this in the command window

```
aa = [1 2 3];

bb = [1 , 2,  3];

cc = [1 ; 2 ; 3];
```

In python, do square brackets mean a tuple or a list?

- All variables are arrays
  - *Scalar* - array with only one element
  - *Vector* - array with only one row or column
  - *Matrix* - array with multiple rows and columns
- Assigning to variable specifies its dimension
  - Don't have to define variable size before assigning to it, as you do in many programming languages
- Reassigning to variable changes its dimension to that of assignment

Address of element is its position in the vector

- "address" often called *index*
- Addresses in Matlab always start at 1 (not 0 as in Python)
  - Address 1 of row vector is leftmost element
  - Address 1 of column vector is topmost element
- To access element of a vector represented by a variable, follow variables name by address inside parentheses, e.g., `v(2)=20` sets second element of vector `v` to `20`

2.5.2 Matrix

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
```

Column 1

```
MAT =  3      11       6       5
       4       7      10       2
      13       9       0       8
```

Element in row 3 and column 1 →

Row 3

```
>> MAT(3,1)

ans = 13

>> MAT(3,1)=20
```

Assign new value to element in row 3 and column 1

```
MAT = 3      11       6       5
      4       7      10       2
     20       9       0       8
```

Only this element changed

```
>> MAT(2,4)-MAT(1,2)

ans = -9
```

The colon : lets you address a range of elements

- Vector (row or column)
  - `va(:)` - all elements
  - `va(m:n)` - elements `m` through `n`
- Matrix
  - `A(:,n)` - all rows of column `n`
  - `A(m,:)` - all columns of row `m`
  - `A(:,m:n)` - all rows of columns `m` through `n`
  - `A(m:n,:)` - all columns of rows `m` through `n`
  - `A(m:n,p:q)` - columns `p` through `q` of rows `m` through `n`
  - `Note Matlab is row then column order`
  - `Which option in Python is that F order or C order?`

- From github, download the course repository
- find the matlab directory
- national wildfires.csv file
  - Look at it with a text editor
  - 3 columns: year, no. of fires, acres burned
- We will do a bunch using the matlab code: wildfires.m
  - open that in matlab or click on the code to start up matlab

MATLAB has many built-in functions for working with arrays. Some common ones are:

- `length(v)` - number of elements in a vector
- `size(A)` - number of rows and columns in a matrix or vector
- `reshape(A,m,n)` - changes number of rows and columns of a matrix or vector while keeping total number of elements the same. For example, changes 4x4 matrix to 2x8 matrix

To create a vector with specified number of terms between first and last

```
 v = linspace( xi, xf, n )
```

- `xi` is first number
- `xf` is last number
- `n` is number of terms (= 100 if omitted)

```
yrc = 1:no_times;
yrb = no_times:-1:1;
```

`Is numpy.linspace different?`

2.2.1 The `zeros`, `ones` and, `eye` Commands

`zeros(m,n)` - makes matrix of `m` rows and `n` columns, all with zeros

`ones(m,n)` - makes matrix of `m` rows and `n` columns, all with ones

`eye(n)` - makes square matrix of `n` rows and columns. Main diagonal (upper left to lower right) has ones, all other elements are zero

- `diag(v)` - makes a square matrix of zeroes with vector in main diagonal
- `diag(A)` - creates vector equal to main diagonal of matrix

When adding/subtracting two arrays *A* and *B*, MATLAB adds/subtracts the corresponding elements, i.e.,

- It adds/subtracts the element in the first row and first column of *A* to the element in the first row and column of *B*
- It adds/subtracts the element in the first row and second column of *A* to the element in the first row and second column of *B*, etc.

This is *elementwise addition/subtraction*

- Use + to add two arrays or to add a scalar to an array
- Use – to subtract one array from another or to subtract a scalar from an array
  - When using two arrays, they must both have the same dimensions (number of rows and number of columns)
  - Vectors must have the same dimensions (rows and columns), not just the same number of elements

MATLAB has lots of functions for operating on arrays. For a vector `v`

- `mean(v)` – mean (average)
- `max(v)` – maximum value, optionally with index of maximum
- `min(v)` – minimum value, optionally with index of minimum
- `sum(v)` – sum
- `median(v)` – median
- `std(v)` – standard deviation
- `sort(v)` – elements sorted into ascending order

There are two ways of multiplying matrices – matrix multiplication and elementwise multiplication

MATRIX MULTIPLICATION

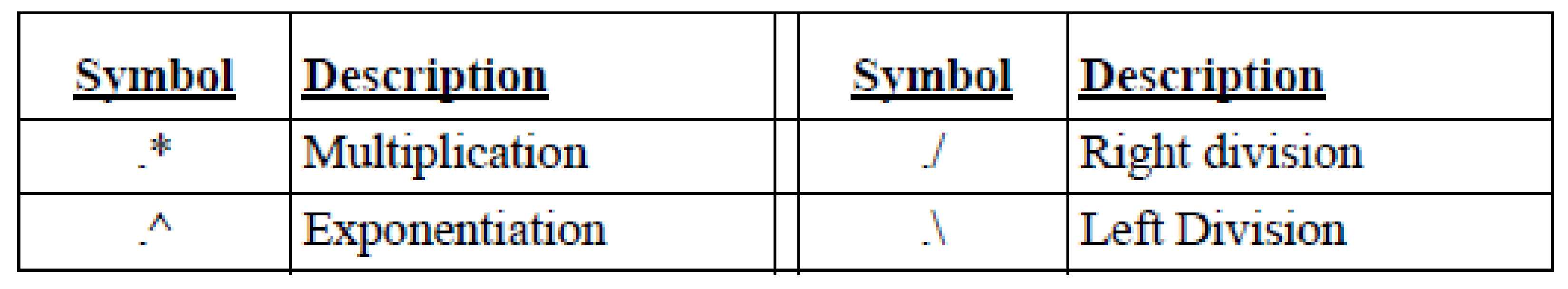
- Type used in linear algebra
- MATLAB denotes this with asterisk (*)
- Number of columns in left matrix must be same as number of rows in right matrix

Another way of saying *elementwise* operations is *element-by-element* operations

- Addition and subtraction of arrays is always elementwise
- Multiplication, division, exponentiation of arrays can be elementwise
- Both arrays must be same dimension
- ***Most codes you develop will use elementwise multiplication and division***

3.4 Element-by-Element Operations

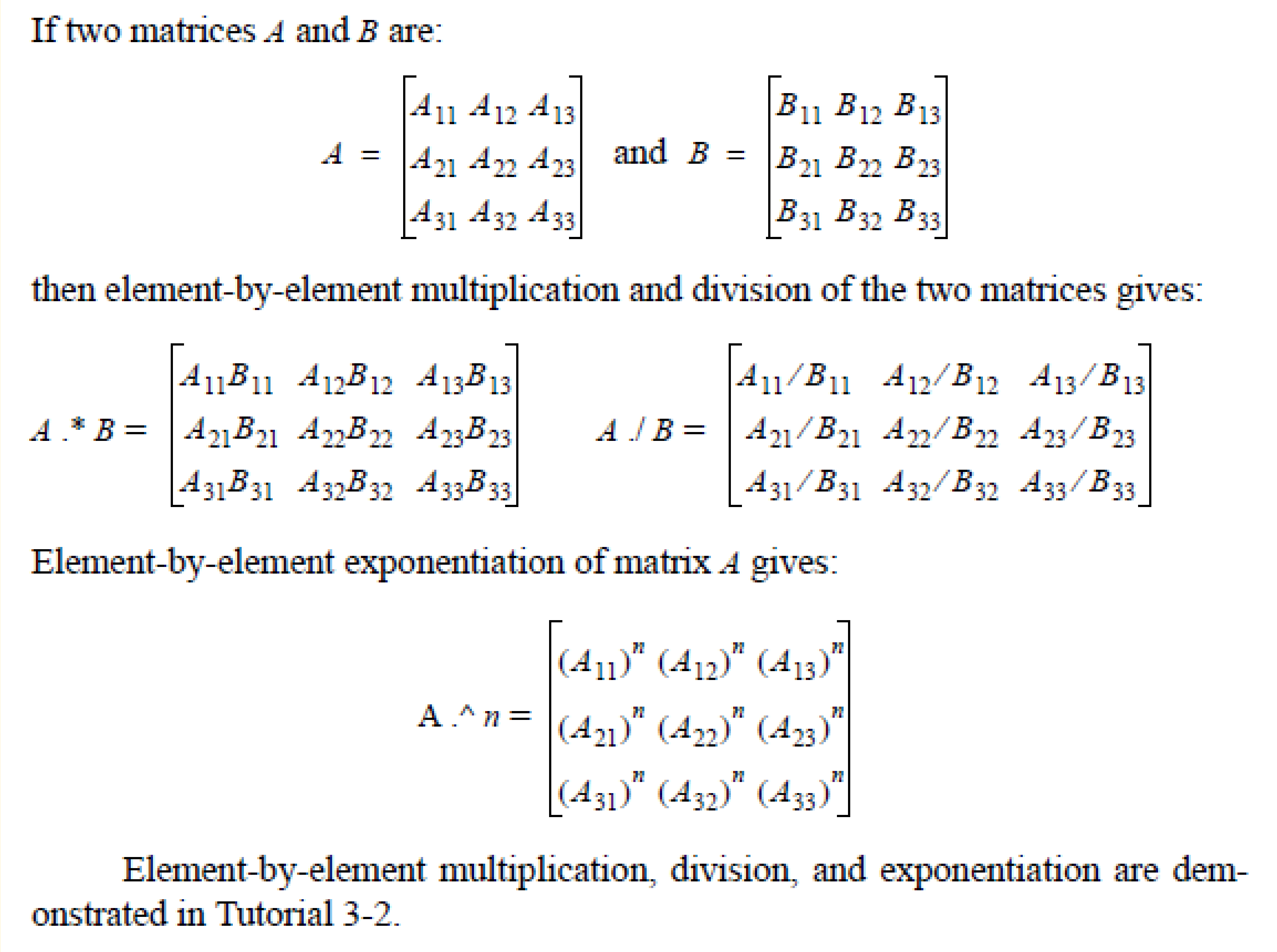
Do elementwise multiplication, division, exponentiation by putting a period in front of the arithmetic operator



Symbol	Description	Symbol	Description
.*	Multiplication	./	Right division
.^	Exponentiation	.\	Left Division

Do you need to use the . in Python or just * in numpy?*

3.4 Element-by-Element Operations



If two matrices A and B are:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \text{ and } B = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{bmatrix}$$

then element-by-element multiplication and division of the two matrices gives:

$$A\ .^{*}\ B = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & A_{13}B_{13} \\ A_{21}B_{21} & A_{22}B_{22} & A_{23}B_{23} \\ A_{31}B_{31} & A_{32}B_{32} & A_{33}B_{33} \end{bmatrix} \qquad A\ ./\ B = \begin{bmatrix} A_{11}/B_{11} & A_{12}/B_{12} & A_{13}/B_{13} \\ A_{21}/B_{21} & A_{22}/B_{22} & A_{23}/B_{23} \\ A_{31}/B_{31} & A_{32}/B_{32} & A_{33}/B_{33} \end{bmatrix}$$

Element-by-element exponentiation of matrix A gives:

$$\text{A}\ .\hat{}\ n = \begin{bmatrix} (A_{11})^n & (A_{12})^n & (A_{13})^n \\ (A_{21})^n & (A_{22})^n & (A_{23})^n \\ (A_{31})^n & (A_{32})^n & (A_{33})^n \end{bmatrix}$$

Element-by-element multiplication, division, and exponentiation are demonstrated in Tutorial 3-2.

If matrices not same dimension in elementwise multiplication, MATLAB gives error

```
>> A = [ 1 2; 3 4];
>> B = [1 0]';
>> A .* B % Meant matrix multiplication!
??? Error using ==> times
Matrix dimensions must agree.
>> A * B % this works
ans =
    1
    3
```

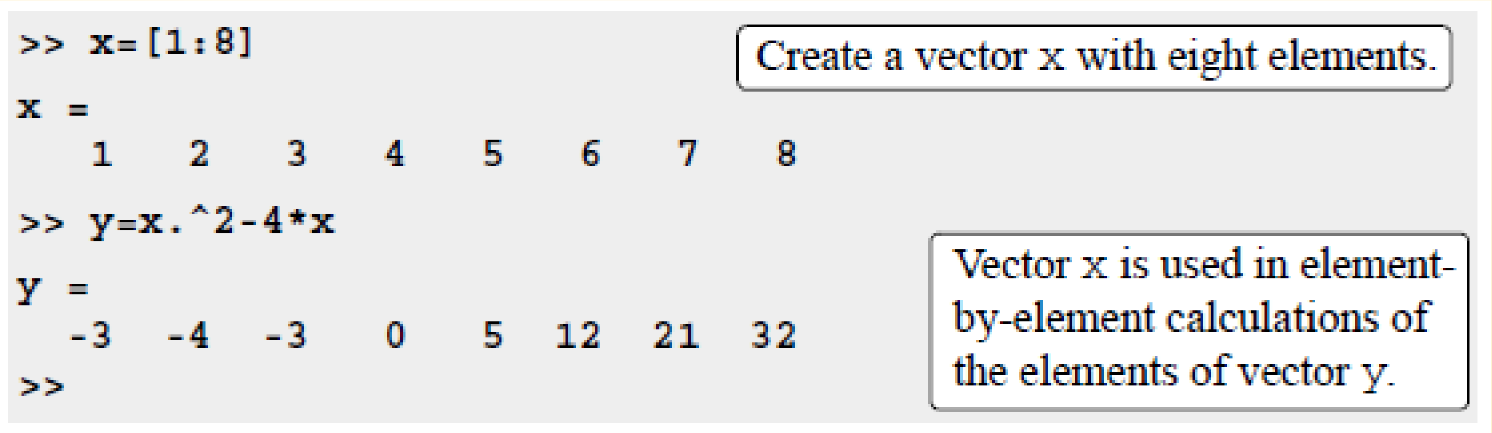
Be careful – when multiplying square matrices

- Both types of multiplication always work
- If you specify the wrong operator, MATLAB will do the wrong computation and there will be no error!
  - Difficult to find this kind of mistake

EXAMPLE

```
>> A = [1 2; 3 4];
>> B = [0 1/2; 1 -1/2];
>> A .* B
>> ans
   0  1
   3 -2
>> A * B
ans =
   2.0000  -0.5000
   4.0000  -0.5000
```

Elementwise computations useful for calculating value of a function at many values of its argument



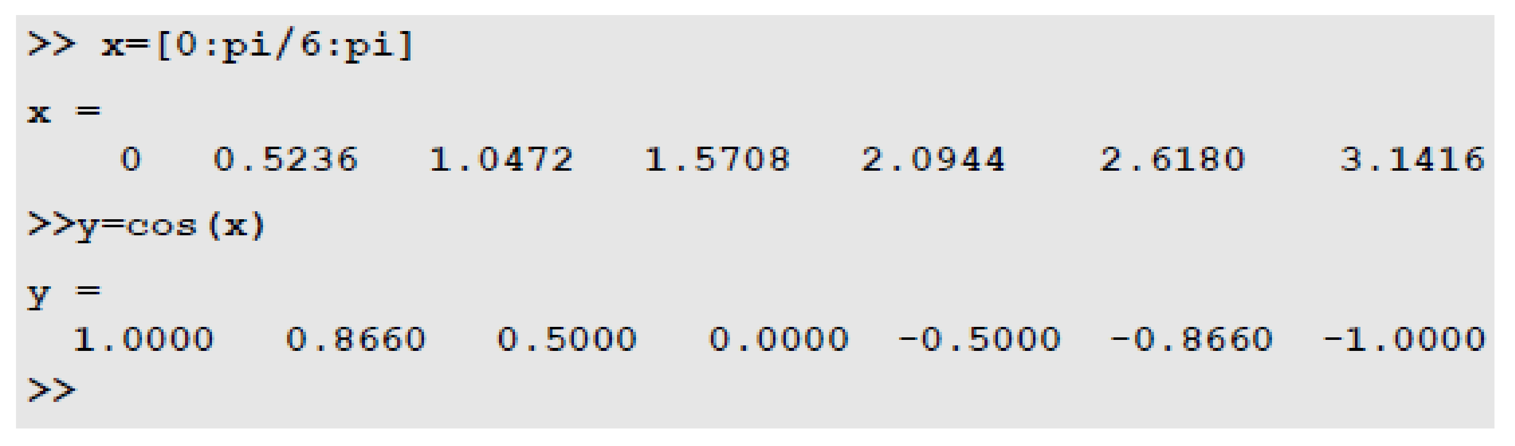
```
>> x=[1:8]

x =
     1     2     3     4     5     6     7     8

>> y=x.^2-4*x

y =
    -3    -4    -3     0     5    12    21    32
>>
```

Create a vector x with eight elements.

Vector x is used in element-by-element calculations of the elements of vector y.

Built-in MATLAB functions can accept arrays as inputs

- When input is array, output is array of same size with each element being result of function applied to corresponding input element
  - Example: if x is a 7-element row vector, cos(x) is [cos(x_1) cos(x_2) cos(x_3) cos(x_4) cos(x_5) cos(x_6) cos(x_7)]



```
>> x=[0:pi/6:pi]

x =
     0    0.5236    1.0472    1.5708    2.0944    2.6180    3.1416

>>y=cos(x)

y =
   1.0000    0.8660    0.5000    0.0000   -0.5000   -0.8660   -1.0000
>>
```