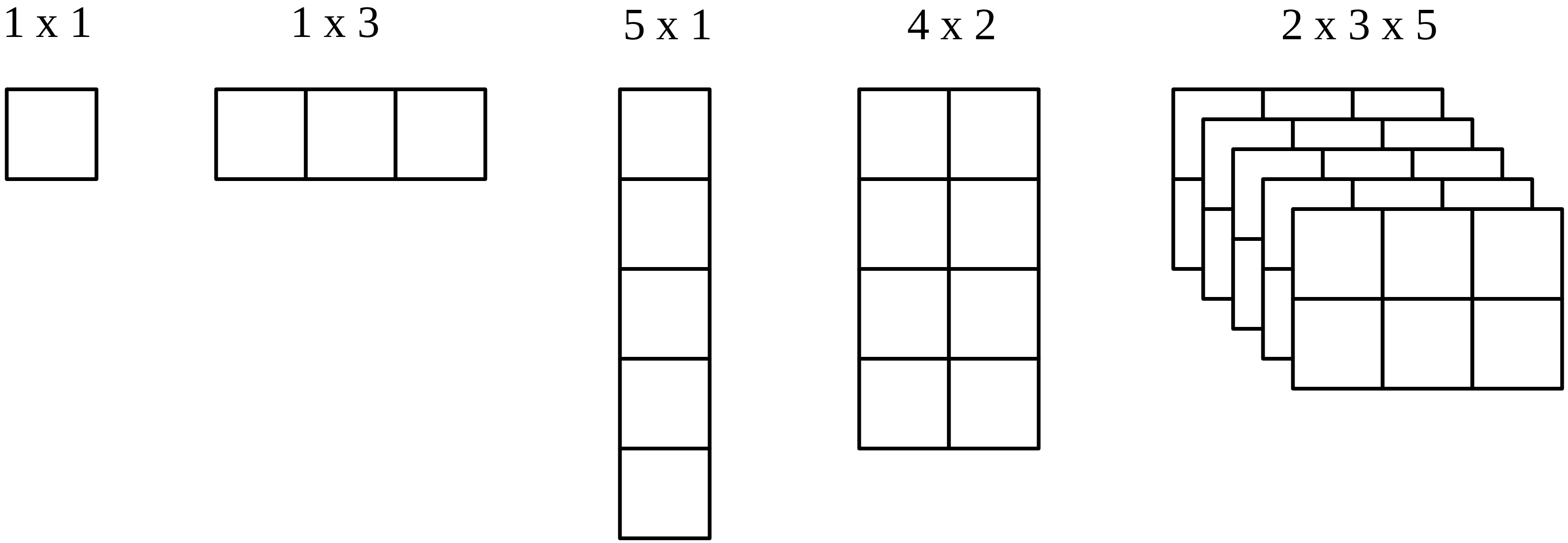
Vectors and Matrices I

Arrays

- an array is a multidimensional table
- the size of an array of dimension k is d_1 x d_2 x … x d_k
- in MATLAB
  - d_1 is the number rows and d_2 is the number of columns



Arrays

- all MATLAB variables are multidimensional arrays
- the size of array in MATLAB:

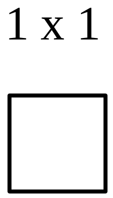
```
>> help size
```

- the notion of an empty array exists

```
>> size([])
```

Scalars

- a scalar in MATLAB is an array of size 1 x 1



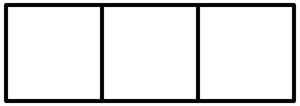
1 x 1

Vectors

- a vector is a 2-dimensional array where one of the size of one of the dimensions is 1

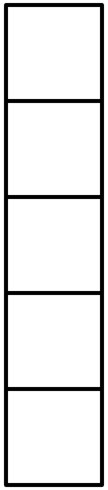
row vector

1 x 3



column vector

5 x 1



Creating row vectors

- a row vector can be created directly by entering the values of the vector inside a pair of square brackets with the values separated by spaces or commas

```
>> v = [1 2 3 4]
v =
     1     2     3     4

>> v = [1, 2, 3, 4]
v =
     1     2     3     4
```

Creating row vectors

- the colon operator can be used to create row vectors having values that are equally spaced

```
>> v = 1:4

v =

     1     2     3     4
```

Creating row vectors

- you can specify the spacing of values using the colon operator

```
>> v = 1:2:9

v =

     1     3     5     7     9

>> v = 1:2:8
>> v = 8:1
```

what does this result in?

and this?

Creating row vectors

- you can specify the spacing of values using the colon operator

```
>> start = 5;
>> step = 5;
>> stop = 25;
>> v = start:step:stop

v =

     5    10    15    20    25
```

Creating row vectors

- the step size can be negative if **`start > stop`**

```
>> start = 25;
>> step = -5;
>> stop = 5;
>> v = start:step:stop

v =

     25    20    15    10      5
```

Creating row vectors

- observe that the stop value is not guaranteed to be at the end of the vector

```
>> start = 25;
>> step = -5;
>> stop = 6;
>> v = start:step:stop

v =

     25     20     15     10
```

Creating row vectors

- the function **`linspace`** will generate a linearly spaced vector that includes the start and end values by calculating the step size for you

```
>> help linspace
 linspace Linearly spaced vector.
    linspace(X1, X2) generates a row vector of 100 linearly
    equally spaced points between X1 and X2.

    linspace(X1, X2, N) generates N points between X1 and X2.

    For N = 1, linspace returns X2.
```

Creating column vectors

- a column vector can be created directly by entering the values of the vector inside a pair of square brackets with the values separated by semi-colons

```
>> v = [1; 2; 3; 4]

v =
     1
     2
     3
     4
```

Creating column vectors

- a column vector can be created from a row vector by *transposing* the row vector

```
>> v = [start:step:stop]'

v =
     25
     20
     15
     10
```

the single quote after after a vector or matrix will compute the transpose* of the vector or matrix

*strictly speaking, the single quote is conjugate transpose operator

`.'` is the transpose operator

Creating column vectors

- a column vector can be created from a row vector by using the colon notation like so

```
>> v = 1:4;
>> w = v(:)

w =
     1
     2
     3
     4
```

notice that the colon has two different uses in this example

Number of elements in a vector

- the function **`length`** will return the number of elements in the vector

```
>> v = [1 2 3 4];
>> length(v)

ans =

     4
```

the function **`length`** does not compute the Euclidean length of a vector!

Magnitude of a vector

- the magnitude of a vector is what mathematicians call the *norm* of the vector
- there are many different norms
  - Euclidean norm (Euclidean length, L^2 norm, L^2 distance)
  - taxicab norm (Manhattan norm, Manhattan distance, L^1 norm)
  - and more…

Magnitude of a vector

- use the **`norm`** function to compute the vector norm
  - by default norm computes the Euclidean norm

```
>> v = [1 1];
>> norm(v)

ans =

    1.4142
```

Indexing elements of a vector

- the elements of the vector can be accessed by using an integer value called an *index*
- MATLAB uses a 1-based index
  - the first element of the vector has index 1
  - the second element has index 2, etc.

- use an index inside of **(   )** after the vector name to access an element of the vector

Indexing elements of a vector

```
>>  v = -5:3

v =

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

>> v(1)

ans =

     -5
```

get the value of the first element in v

Indexing elements of a vector

```
>>  v = -5:3

v =

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

>> v(2)

ans =

     -4
```

get the value of the second element in v

Indexing elements of a vector

```
>>  v = -5:3

v =

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

>> v(3) = 100
```

set the value of the third element in v to 100

```
v =

    -5    -4   100    -2    -1     0     1     2     3
```

Indexing elements of a vector

- the keyword **end** can be used to access the last element of the vector

```
>>  v = -5:3

v =

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

>> v(end)

ans =

     3
```

get the value of the last element in v

Indexing elements of a vector

- you can use arithmetic with **end**

```
>>  v = -5:3

v =

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

>> v(end - 1)

ans =

     2
```

get the value of the second last element in v

Indexing elements of a vector

- the index does not need to be a scalar
  - it can also be a vector of indices!

```
>>  v = -5:3

v =

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

>> v([1 3 5])
```

get a vector of the first, third and fifth elements of v

```
ans =

     -5    -3    -1
```

Indexing elements of a vector

- the index does not need to be a scalar
  - it can also be a vector of indices!

```
>>  v = -5:3

v =

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

>> v([1 3 5]) = [7 8 9]

v =

     7    -4     8    -2     9     0     1     2     3
```

set the first, third and fifth elements of v