

# Lecture 2

# Lecture 2: Vectors and Matrices

## ➤ Vectors and Matrices

- Vectors and matrices are used to store sets of values of the same type. A vector is a one-dimensional array. A matrix is a two-dimensional array.
- A vector can be either a row vector or a column vector

$$V = (v_1 \quad v_2 \quad \cdots \quad v_n)$$

$$W = (w_1 \quad w_2 \quad \cdots \quad w_n)' = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

- A matrix can be seen as a table of values
- The dimensions of a matrix are  $r \times c$ ,  $r$  is the number of rows and  $c$  is the number of columns

$$M = \begin{pmatrix} m_{11} & \cdots & m_{1c} \\ \vdots & \ddots & \vdots \\ m_{r1} & \cdots & m_{rc} \end{pmatrix}$$

The element is denoted by  $m_{ij}$ .

**Both vectors and matrices are treated as variables in MATLAB**

# Lecture 2: Vectors and Matrices

## ➤ Creating Row Vectors

- Most direct way: to put the values in square brackets, separated by either spaces or commas

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

`v =`

`1 2 3 4`

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

`v =`

`1 2 3 4`

What if we just enter: `>> [1, 2, 3, 4]`

# Lecture 2: Vectors and Matrices

## ➤ Creating Row Vectors (cont)

- Colon operator (if the values are regularly spaced.)

Form 1: **first : last** (The default step value is 1)

Example: `>> v = 1:5`

`v =`

`1 2 3 4 5`

Form 2: **first : step : last**

Example: `>> nv = 1:2:9`

`nv =`

`1 3 5 7 9`

What if `>> v = 0.1 : 2` ?

What if `>> v = 1 : 2 : 9.5` ?

The elements are calculated using the first and step values. The last value may not be the value of the last element

Can we use square brackets here?

Can we define a vector as `>> nv = 1 3 5` ?

The square brackets can be ignored when using colon operators?

# Lecture 2: Vectors and Matrices

## ➤ Creating Row Vectors (cont)

- Linspace function: `linspace(x,y,n)`

Round brackets here!!!

Example: `>> ls = linspace(3,15,5)`

`ls =`

`3 6 9 12 15`

- Concatenate vectors:

Example: `>> newvec = [nv ls]`

Square brackets must be used here!!!

`newvec =`

`1 3 5 7 9 3 6 9 12 15`

When should we use colon operators, and when should we use linspace function?

When do we use square brackets? When do we use round brackets?

When specifying the values of elements, square brackets are used.

# Lecture 2: Vectors and Matrices

## ➤ Referring to and Modifying Elements

- A particular element in a vector is accessed using the **vector name** and the **element number**: `vecname(elenum)`

Example1: `newvec = [ 1 3 5 7 9 3 6 9 12 15]`

`>> newvec(5)`

`ans =`

`9`

Parentheses should be used, not square brackets

Example2: `>> b = newvec(4:6)`

`b =`

`7 9 3`

What's the meaning of 4:6?

Example3: Can we use `>> c = newvec(4:2:8) ?`

If the indices are not regularly spaced, what should we do?

# Lecture 2: Vectors and Matrices

## ➤ Referring to and Modifying Elements (cont)

- Any vector can be used for the indices in another vector

Example: newvec = [ 1 3 5 7 9 3 6 9 12 15]

```
>> newvec([1 5 10])
```

```
ans =
```

```
1 9 15
```

vector [1 5 10] is called an index vector

•

Can we use `>> newvec(1 5 10)`? Why?

# Lecture 2: Vectors and Matrices

## ➤ Referring to and Modifying Elements

- The value stored in a vector element can be changed

Example: `>> b(2) = 11`

`b =`

`7 11 3`

- By using an index, a vector can be extended. Zeros are used to fill the gaps.

Example: `>> rv = [3 55 11]`

`>> rv(5) = 2`

`rv =`

`3 55 11 0 2`

# Lecture 2: Vectors and Matrices

## ➤ Creating Column Vectors

- One way is by explicitly putting the values in square brackets separated by semicolons

Example: `>> c = [1; 2; 3; 4]`

`c =`

`1`

`2`

`3`

`4`

- There is no direct way to use the colon operator to get a column vector. However, any row vectors created using the colon operators can be transposed to get a column vector

Example: `>> r = 1:3`

`>> c = r'`

`c =`

`1`

`2`

`3`

# Lecture 2: Vectors and Matrices

## ➤ Creating Matrix Variables

- Creating a matrix variable is just **generalization of creating row and column vectors**, that is, the values within a row are separated by either spaces or commas, the different rows are separated by semicolons

Example: `>> mat = [4 3 1; 2 5 6]`

`mat =`

4 3 1

2 5 6

- Iterators can also be used for the values on the rows using the **colon operator**

Example: `>> mat = [2:4; 3:5]`

`>> mat2 = [4:2:8; 3:5]`

`mat =`

2 3 4

3 4 5

`mat2 =`

4 6 8

3 4 5

- Different rows** in the matrix can also be specified by **pressing the Enter** key after each row instead of typing a semicolon

Example: `>> newmat = [2 6 88`

`33 5 2]`

`newmat =`

2 6 88

33 5 2

# Lecture 2: Vectors and Matrices

## ➤ Referring to and Modifying Matrix Elements

- To refer to matrix elements, the row and column indices are given in parentheses

Example:

```
>> mat = [2:4; 3:5]
```

```
mat =
```

```
2 3 4
```

```
3 4 5
```

```
>> mat(2,3)
```

```
ans =
```

```
5
```

The 1<sup>st</sup> number is the row index, and the 2<sup>nd</sup> number is the column index.

To refer to a subset of a matrix

```
>> mat(1:2,2:3)
```

```
ans =
```

```
3 4
```

```
4 5
```

A comma is used to separate row and column indices, NOT a semicolon as used to separate rows in a matrix.

Using a colon for the row or column index means all rows or columns, regardless of how many

```
>> mat(1,:)
```

```
ans =
```

```
2 3 4
```

# Lecture 2: Vectors and Matrices

## ➤ Referring to and Modifying Matrix Elements (Cont)

- Linear indexing: If a single index is used within a matrix, MATLAB unwinds the matrix column by column

Example: `>> mat = [2:4; 3:5]`

`mat =`

`2 3 4`

`3 4 5`

`>> mat(5)`

`ans =`

`4`

- An individual element in a matrix can be modified by assigning a value

Example: `>> mat(1,2) = 11`

`mat =`

`2 11 4`

`3 4 5`

`>> mat(2,:) = 5:7`

`mat =`

`2 11 4`

`5 6 7`

**A colon means all rows or all columns**

# Lecture 2: Vectors and Matrices

## ➤ Dimensions

- **length** function: returns the number of elements in a vector. For a matrix, the length function will return either the number of rows or the number of columns whichever is larger

Example: `>> vec = -2:1`  
`vec =`  
`-2 -1 0 1`  
`>> length(vec)`  
`ans =`  
`4`

- **size** function: returns the numbers of rows and columns in a matrix

Example: `>> size(vec)`  
`ans =`  
`1 4`

- **numel** function: returns the total number of elements in any array
- **end** expression: MATLAB has a built-in expression end that can be used to refer to the last element in a vector. For example, v(end) is equivalent to v(length(v))

Example: `>> mat = [1:3; 4:6]`  
`mat =`  
`1 2 3`  
`4 5 6`  
`mat(1, end)`  
`ans =`  
`3`

# Lecture 2: Vectors and Matrices

## ➤ Using Functions with Vectors and Matrices

- An entire vector or matrix can be passed as an argument to a function. MATLAB will evaluate the function on every element and return as a result a vector or matrix with the same dimensions as the original

Example: `>> vec=-3:4`

`>> abs(vec)`

## ➤ Empty Vectors

- An empty vector can be created using empty square brackets. Empty vectors can be used to delete elements from arrays.

Example: `>> vec = 1:8`

`vec =`  
`1 2 3 4 5 6 7 8`

`>> vec(2:4) = [ ]`

`vec =`  
`1 5 6 7 8`

# Lecture 2: Vectors and Matrices

## ➤ Arithmetic Operations for Vectors and Matrices

- MATLAB has two types of arithmetic operations: array operations and matrix operations.
- Matrix operations follow the rules of linear algebra (see reading assignment of this lecture).
- Array operations execute element by element operations and support multidimensional arrays.
- The period character “.” distinguishes the array operations from the matrix operations.

# Lecture 2: Vectors and Matrices

## ➤ Scalar Operators

Operation	Algebraic Form	MATLAB
addition	$a + b$	$a + b$
subtraction	$a - b$	$a - b$
multiplication	$a \times b$	$a * b$
division	$a / b$	$a / b$
exponentiation	$a^b$	$a ^b$

# Lecture 2: Vectors and Matrices

## ➤ Array and Matrix Operators

Operation	MATLAB Form	Comments
Array addition	$a + b$	Array addition and matrix addition are identical.
Array subtraction	$a - b$	Array subtraction and matrix subtraction are identical.
Array multiplication	$a .* b$	Element-by-element multiplication of $a$ and $b$ .
Matrix multiplication	$a * b$	Matrix multiplication of $a$ and $b$ . The number of columns in $a$ must equal the number of rows in $b$ .
Array right division	$a ./ b$	Element-by-element division of $a$ and $b$ : $a(i,j)/b(i,j)$ . Both arrays must be the same shape, or one of them must be a scalar.
Array left division	$a .\ b$	Element-by-element division of $a$ and $b$ , but with $b$ in the numerator: $b(i,j)/a(i,j)$ . Both arrays must be the same shape, or one of them must be a scalar.
Matrix right division	$a / b$	Matrix division defined by $a * \text{inv}(b)$ , where $\text{inv}(b)$ is the inverse matrix of $b$ .
Matrix left division	$a \backslash b$	Matrix division defined by $\text{inv}(a) * b$ , where $\text{inv}(a)$ is the inverse of matrix $a$ .
Array exponentiation	$a .^ b$	Element-by-element exponentiation of $a$ and $b$ : $a(i,j)^b(i,j)$ . Both arrays must be the same shape, or one of them must be a scalar.

# Lecture 2: Vectors and Matrices

## ➤ Summaries:

- Creating row and column vectors;
- Referring to and modifying elements in a vector;
- Creating matrices;
- Functions about dimensions of vectors and matrices, and changing dimensions;
- Using functions with vectors and matrices.

## ➤ Homework #2 (on Canvas)

- Due next Tuesday