

Module 02: Vectors and Matrices

Chul Min Yeum

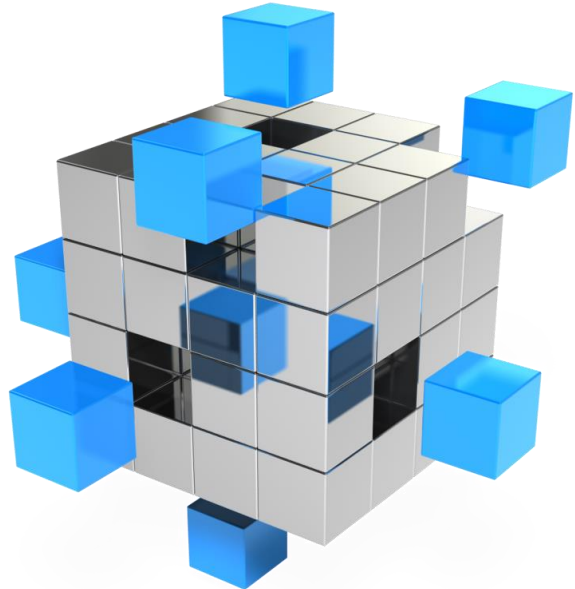
Assistant Professor

Civil and Environmental Engineering

University of Waterloo, Canada



UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING



Module 2: Intended Learning Outcomes

- Create an array in MATLAB, which includes a scalar, vector, and matrix.
- Refer elements in an array using index or subscript.
- Modify and concatenate an array.
- Perform array operations (element-wise operation and matrix operation)
- Define and evaluate a character and character vector
- Create a 3D matrix and refer its element.
- Understand a linear indexing method

Matrix, Row Vector, and Column Vector

A matrix is an ordered rectangular array of numbers. A general matrix with m rows and n columns has the following structure:

$$\begin{array}{c} \text{column, } j \\ \begin{array}{cccccc} & 1 & 2 & 3 & \cdots & n \\ \text{row, } i & \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & \cdots & a_{mn} \end{bmatrix} & m \times n \end{array} \end{array}$$

where a_{ij} is each number in the array indexed by its row position i and column position j .

Matrix, Row Vector, and Column Vector in MATLAB

- A **matrix** is used to store a set of values of the same type; every value is stored in an **element**.
- A matrix looks like a table; it has both rows and columns
- A matrix with m rows and n columns is called **$m \times n$** ; these are called its **dimensions**;
- A **vector** is a special case of a matrix in which one of the dimensions is 1
- The term **array** is frequently used in MATLAB to refer generically to a matrix or a vector
 - a row vector with n elements is $1 \times n$, e.g. 1×4 :
 - a column vector with m elements is $m \times 1$, e.g. 3×1 :
- A **scalar** is an even more special case; it is 1×1 , or in other words, just a single value

Creating Row Vectors

- Direct method: put the values you want in square brackets, separated by either **commas** or **spaces**
- **Colon operator**: iterates through values in the form `first:step:last`
 - If no step is specified, the default is 1 so for example `2:4` creates the vector `[2 3 4]`
 - Can go in reverse e.g. `4:-1:1` creates `[4 3 2 1]`
 - Will not go beyond last e.g., `1:2:6` creates `[1 3 5]`

```
rvec1 = [1 2 3 4]; % sepr. by space  
rvec2 = [1,2,3,4]; % sepr. by comma  
rvec3 = [1    2    3 4];  
rvec4 = 1:4; % use a colon operator  
rvec5 = 1:1:4; % step by 1  
rvec6 = [1:4]; % okay with put. brackets
```

Name	Value
rvec1	[1 2 3 4]
rvec2	[1 2 3 4]
rvec3	[1 2 3 4]
rvec4	[1 2 3 4]
rvec5	[1 2 3 4]
rvec6	[1 2 3 4]


Creating Row Vectors (Continue)

```
rvec1 = [1 2 3 4 5 6];  
rvec2 = [1:6];  
  
rvec3 = [1:2:6]; % step by 2  
  
rvec4 = [1:3:6]; % step by 3  
  
rvec5 = [6:-1:1]; % step by -1
```

Name	Value
rvec1	[1 2 3 4 5 6]
rvec2	[1 2 3 4 5 6]
rvec3	[1 3 5]
rvec4	[1 4]
rvec5	[6 5 4 3 2 1]

```
my_rvec1 = [1:5:6];  
  
my_rvec2 = [5:-2:1];
```

Name	Value
rvec1	[1 6]
rvec2	[5 3 1]

 **Remember !** `variable = first:step:last`

Remind: Transpose

Remind

The **transpose** of a matrix is the operation of flipping the rows to columns and vice versa across the diagonal of the matrix. For example,

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}_{3 \times 2} \quad \text{has transpose} \quad A^T = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2 \times 3}$$

where the power of T indicates the transpose of a matrix. Note that in the example, the dimensions of the matrix changes from 3×2 to 2×3 (with the same total number of elements) and diagonal elements remain the same.

Creating Column Vectors

- A **column** vector is an $m \times 1$ vector
- **Direct method:** can create by separating values in square brackets with ***semicolons*** (e.g., `[4; 7; 2]`)
- You cannot directly create a column vector using methods such as the colon operator, but you can create a row vector and then ***transpose*** it to get a column vector using the transpose operator (e.g., `[4 7 2]'`)

```
rvec1 = [1;2;3;4];  
rvec2 = [1 2 3 4]';  
rvec3 = [1:4];  
rvec4 = rvec3';
```

Name	Value
rvec1	<code>[1; 2; 3; 4]</code>
rvec2	<code>[1; 2; 3; 4]</code>
rvec3	<code>[1 2 3 4]</code>
rvec4	<code>[1; 2; 3; 4]</code>

Referring to Elements

- The elements in a vector are numbered sequentially; each element number is called the **index**, or **subscript** and are shown above the elements here:

```
myvec = [5 33 11 -4 2];
```

Index
Element

1	2	3	4	5
5	33	11	-4	2

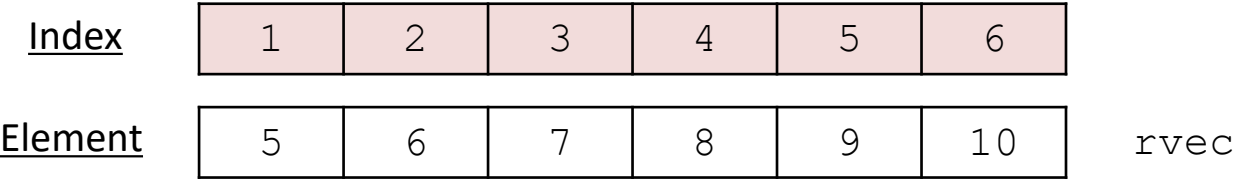
- Refer to an element using its **index** or **subscript** in parentheses, e.g. `vec(4)` is the 4th element of a vector `vec` (assuming it has at least 4 elements)
- Can also refer to a subset of a vector by using an **index vector** which is a vector of indices e.g. `vec([2 5])` refers to the 2nd and 5th elements of `vec`.

⚠: The index in MATLAB starts from 1 !!

Example: Referring to Elements

```
rvec = [5:10];  
cvec = rvec';  
  
val1 = rvec(2);  
val2 = rvec(5);  
  
vec1 = rvec([1 4]);  
vec2 = rvec([1 2 6]);  
  
vec3 = cvec([1 4]);  
vec4 = cvec([1 2 6]);
```

Name	Value
rvec	[5 6 7 8 9 10]
cvec	[5; 6; 7; 8; 9; 10]
val1	6
val2	9
vec1	[5 8]
vec2	[5 6 10]
vec3	[5; 8]
vec4	[5; 6; 10]




Modifying Vectors

- Elements in a vector can be changed.

variable(index) = expression

```
rvec = 1:4;  
rvec1 = rvec;  
rvec1(4) = 10;
```

Name	Value
rvec	[1 2 3 4]
rvec1	[1 2 3 10]

 Value(s) computed from *expression* are assigned to *variable* at *index* location(s).

```
rvec = 1:5;  
rvec1 = rvec;  
rvec2 = rvec;  
rvec3 = rvec;  
  
rvec1([1 3]) = [0 0];  
rvec2(1:3) = 4:6;  
rvec3([1 3]) = rvec4([3 1]);
```

Name	Value
rvec	[1 2 3 4 5]
rvec1	[0 2 0 4 5]
rvec2	[4 5 6 4 5]
rvec3	[3 2 1 4 5]

Modifying Vectors (Continue)

- A vector can be extended by referring to elements that do not yet exist
- A vector cannot be read by an index that does not yet exist

variable(index) = expression

```
rvec1 = 1:4;  
rvec1(5) = 5;  
  
rvec2 = 1:4;  
rvec2([5 6]) = [3 2];
```

Name	Value
rvec1	[1 2 3 4 5]
rvec2	[1 2 3 4 3 2]

```
rvec1 = 1:4;  
val1 = rvec1(5)
```

Error: Index exceeds the number of array elements (4)

```
rvec1 = 1:4;  
rvec1(1:2) = [1 2 3];
```

Error: Unable to perform assignment because the left and right sides have a different number of elements.

Concatenation

- Vectors can be created by joining together existing vectors, or adding elements to existing vectors
- This is called **concatenation**

```
rvec = [1 2];  
rvec1 = [rvec 8 9];  
rvec2 = [rvec rvec1];  
  
cvec = [4;5];  
cvecp = [6;7];  
cvec1 = [cvec; cvecp];
```

Name	Value
rvec	[1 2]
rvec1	[1 2 8 9]
rvec2	[1 2 1 2 8 9]
cvec	[4 5]'
cvecp	[6 7]'
cvec1	[4 5 6 7]'

```
rvec = [1 2];  
rvec1 = [rvec; 8];
```

*Error: using vertcat
Dimensions of arrays being concatenated
are not consistent.*

Creating a Matrix

- Separate values within rows with ***blanks*** or ***commas***, and separate the rows with ***semicolons***
- Can use any method to get values in each row (any method to create a row vector, including colon operator)
- **There must ALWAYS be the same number of values in every row!!**

```
m1 = [1 2 3;4 5 6];  
  
r1 = [1 2 3];  
r2 = [4 5 6];  
  
m2 = [r1;r2];  
m3 = [r1;4 5 6];  
% it works not recommend
```

m1, m2, m3

1	2	3
4	5	6

Name	Value
m1	[1 2 3;4 5 6]
r1	[1 2 3]
r2	[4 5 6]
m2	[1 2 3;4 5 6]
m3	[1 2 3;4 5 6]

Functions that Create Matrices

There are many built-in functions to create matrices

- **zeros (n)** creates an $n \times n$ matrix of all zeros
- **zeros (n,m)** creates an $n \times m$ matrix of all zeros
- **ones (n)** creates an $n \times n$ matrix of all ones
- **ones (n,m)** creates an $n \times m$ matrix of all ones
- **eye (n)** creates an $n \times n$ identity matrix.

⚠: **zeros (n)** or **ones (n)** is a matrix, not a column or row vector.

```
m1 = zeros(3);  
m2 = zeros(3,2);
```

```
m3 = ones(2,1);  
m4 = eye(3);
```

m1

0	0	0
0	0	0
0	0	0

m2

0	0
0	0
0	0

Name	Value
m1	[0 0 0; 0 0 0; 0 0 0]
m2	[0 0; 0 0; 0 0]
m3	[1; 1]
m4	[1 0 0; 0 1 0; 0 0 1]

m3

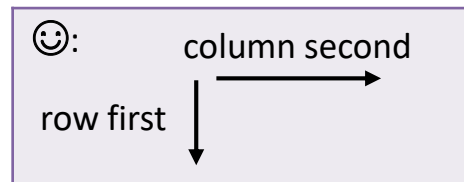
1
1

m4

1	0	0
0	1	0
0	0	1

Matrix Elements

- To refer to an element in a matrix, you use the matrix variable name followed by ***the index of the row, and then the index of the column***, in parentheses
- ALWAYS refer to the row first, column second
- This is called **subscripted indexing**.



Variable (row indexes, column indexes)

```
m1 = [1 2 3; 4 5 6];
```

1	2	3
4	5	6

Element

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)

Indexing

Matrix Elements (Continue)

- You can index and refer an entire row or column using a colon (:).
- Can also refer to any subset of a matrix
 - To refer to the entire m^{th} row: `m1 (m, :)`
- To refer to the entire n^{th} column: `m1 (:, n)`
- To refer to the last row or column use **end**, (e.g. `m1 (end, m)` is the m^{th} value in the last row)
- Can modify an element or subset of a matrix in an assignment statement

Variable (row indexes, column indexes)

Used for **referring values** and **reading values**

Example: Matrix Elements

```
m1 = [1 2 3;4 5 6;7 8 9];
```

```
c1 = m1(:,1);
```

```
r1 = m1(2,:);
```

```
c2 = m1(:,end);
```

```
c3 = m1(1:3, end);
```

```
c4 = m1(1:end, end);
```

```
r2 = m1(end,:);
```

```
r3 = m1(3,:);
```

```
m2 = m1(2:3, 2:3);
```

```
m3 = m1(2:end, 2:end);
```

```
m4 = m1([1 3], [1 3]);
```

m1

1	2	3
4	5	6
7	8	9

c1

1
4
7

r1

4	5	6
---	---	---

c2

3
6
9

c3

3
6
9

c4

3
6
9

r2

7	8	9
---	---	---

r3

7	8	9
---	---	---

m2

5	6
8	9

m3

5	6
8	9

m4

1	3
7	9

Modifying Matrices

- An individual element in a matrix can be modified by assigning a new value to it.
- Entire rows and columns can also be modified.
- Any subset of a matrix can be modified, as long as what is being assigned has the same dimensions as the subset being modified.
- Exception to this: a scalar can be assigned to any size subset; the same scalar is assigned to every element in the subset.

Variable (row indexes, column indexes) = Expression

Used for **referring** elements and **reading** values

Example: Modifying Matrices

```
>> m1 = [1 2 3; 4 5 6; 7 8 9];
```

```
>> m2 = m1
```

```
m2 =
```

1	2	3
4	5	6
7	8	9

```
>> m2(1,1) = 10
```

```
m2 =
```

10	2	3
4	5	6
7	8	9

```
>> m2(1,end) = 30
```

```
m2 =
```

10	2	30
4	5	6
7	8	9

... continue ...

```
>> m2(:,1) = [10; 10; 10];
```

```
m2 =
```

10	2	30
10	5	6
10	8	9

```
>> m2(end,:) = zeros(1, 3);
```

```
m2 =
```

10	2	30
10	5	6
0	0	0

```
>> m2(3,:) = m2(1,:)
```

```
m2 =
```

10	2	30
10	5	6
10	2	30

Example: Modifying Matrices (Continue)

... continue ...

```
>> m2(:,1) = [10; 10; 10];  
m2 =
```

10	2	30
10	5	6
10	8	9

```
>> m2(end,:) = zeros(1, 3);  
m2 =
```

10	2	30
10	5	6
0	0	0

```
>> m2(2,:) = 1;  
m2 =
```

10	2	30
1	1	1
0	0	0


... continue ...

```
>> m2(:, end) = 3;  
m2 =
```

10	2	3
1	1	3
0	0	3

```
>> m2([1 3], [1 3]) = 10;  
m2 =
```

10	2	10
1	1	3
10	0	10

 Exception to this: a scalar can be assigned to any size subset; the same scalar is assigned to every element in the subset.

Example: Modifying Matrices



Q: Swap the 2nd and 3rd columns in mat1

```
mat1 = [1 2 3; 4 5 6; 7 8 9];  
  
% write your code here  
vec = mat1(:, 2);  
  
mat1(:,2) = mat1(:,3);  
mat1(:,3) = vec;
```

```
mat1 = [1 2 3; 4 5 6; 7 8 9];  
  
mat1(:, [3 2]) = mat1(:, [2 3]);
```

1	2	3
4	5	6
7	8	9

mat1



1	3	2
4	6	5
7	9	8

mat1

Matrix Dimension

- There are several functions to determine the dimensions of a vector or matrix:
 - `size` returns the # of rows and columns for a vector or matrix
 - Important: capture both of these values in an assignment statement
`[r, c] = size(mat)`
 - `numel` returns the total # of elements in a vector or matrix

⚠: Very important to generalize your script: do not assume that you know the dimensions of a vector or matrix – use `size` or `numel` to find out!

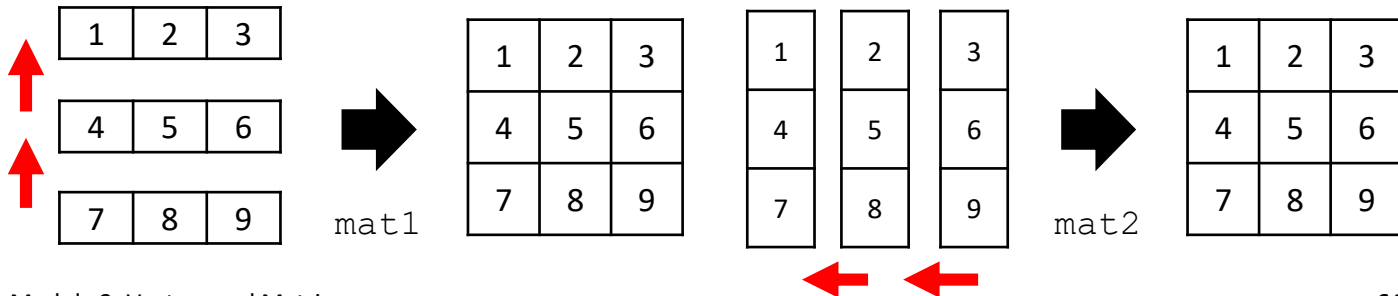
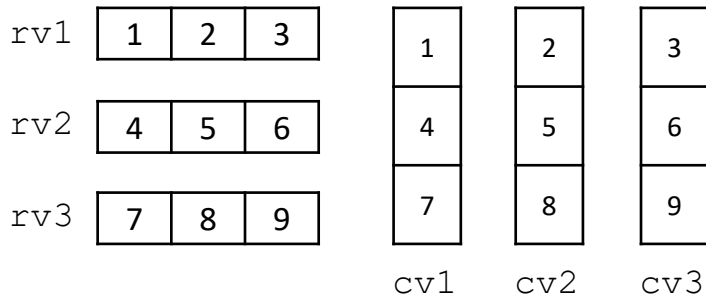
```
m1 = [1 2 3;4 5 6];  
  
[r1, c1] = size(m1);  
  
n_m1 = numel(m1);
```

Name	Value
m1	[1 2 3;4 5 6]
r1	2
c1	3
n_m1	6

Matrix Concatenation

```
rv1 = [1 2 3];  
rv2 = [4 5 6];  
rv3 = [7 8 9];  
  
mat1 = [rv1; rv2; rv3];  
  
cv1 = [1; 4; 7];  
cv2 = [2; 5; 8];  
cv3 = [3; 6; 9];  
  
mat2 = [cv1 cv2 cv3]
```

Matrices can be created by joining, vectors or adding elements.



Empty Vectors and Matrices

- An **empty vector** is a vector with no elements; an empty vector can be created using square brackets with nothing inside []
- To **delete** an element from a vector, assign an empty vector to that element
- Delete an entire row or column from a matrix by assigning []
 - Note: cannot delete an individual element from a matrix

```
>> v1 = 2:6
```

```
v1 =
```

```
      2      3      4      5      6
```

```
>> v1(end) = []
```

```
v1 =
```

```
      2      3      4      5
```

```
>> v1(3) = []
```

```
v1 =
```

```
      2      3      5
```

```
>> m1 = [1 2 3;4 5 6;7 8 9]
```

```
m1 =
```

```
      1      2      3
```

```
      4      5      6
```

```
      7      8      9
```

```
>> m1(2,:) = []
```

```
m1 =
```

```
      1      2      3
```

```
      7      8      9
```

Scalar Operations

- Numerical operations can be performed on every element in a vector or matrix
- For example, ***Scalar multiplication***: multiply every element by a scalar

```
>> v1 = [4  0  11] * 3  
v1 =  
  
    12     0    33
```

- Another example: ***Scalar addition***; add a scalar to every element

```
>> v2 = zeros(1,3) + 5  
v2 =  
  
     5     5     5
```

Array Operations

Array operations on two matrices A and B .

- These are applied term-by-term, or element-by-element
- In MATLAB:
 - matrix addition: $A + B$
 - matrix subtraction: $A - B$ or $B - A$
- For operations that are based on multiplication (multiplication, division, and exponentiation), a dot must be placed in front of the operator
 - array (element-wise) multiplication: $A .* B$
 - array (element-wise) division: $A ./ B$, $A .\ B$
 - array (element-wise) exponentiation $A .^ 2$
- **Matrix multiplication: NOT an array operation:** $A .* B$ is not equal to $A * B$.

Example: Array Operations

```
m1 = [1 2 3; 4 5 6; 7 8 9];  
m2 = [1 1 1; 2 2 2; 1 1 1];  
sv = 3;
```

```
mat1 = m1 + m2;  
mat2 = m1 .* m2;  
mat3 = m1 - m2;  
mat4 = m1 ./ m2;
```

```
mat5 = m1 + sv;  
mat6 = m1 * sv;
```

mat5 =

4	5	6
7	8	9
10	11	12

mat6 =

3	6	9
12	15	18
21	24	27

m1

1	2	3
4	5	6
7	8	9

m2

1	1	1
2	2	2
1	1	1

sv

3

mat1 =

1	2	3
4	5	6
7	8	9

+

1	1	1
2	2	2
1	1	1

=

2	3	4
6	7	8
8	9	10

mat2 =

1	2	3
4	5	6
7	8	9

.*

1	1	1
2	2	2
1	1	1

=

1	2	3
8	10	12
7	8	9

mat3 =

1	2	3
4	5	6
7	8	9

-

1	1	1
2	2	2
1	1	1

=

0	1	2
2	3	4
6	7	8

mat4 =

1	2	3
4	5	6
7	8	9

./

1	1	1
2	2	2
1	1	1

=

1	1	1
2	2.5	3
7	8	9

Matrix Multiplication: Dimensions

- In MATLAB, the multiplication **operator** `*` performs matrix multiplication
- In order to be able to multiply a matrix A by a matrix B , the number of columns of A must be the same as the number of rows of B
- If the matrix A has dimensions $m \times n$, that means that matrix B must have dimensions $n \times \text{something}$;
 - In mathematical notation, $[A]_{m \times n} [B]_{n \times p}$
 - We say that the **inner dimensions** must be the same
- The resulting matrix C has the same number of rows as A and the same number of columns as B
 - in other words, the **outer dimensions** $m \times p$
 - In mathematical notation, $[A]_{m \times n} [B]_{n \times p} = [C]_{m \times p}$.

⚠: Matrix multiplication is NOT an array operation. It does NOT mean multiplying term by term (element by element)

Matrix Times a Vector

A linear system of equations with coefficient matrix A , variable vector \vec{x} and constant term vector \vec{b} , can be expressed as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$\begin{matrix} \text{matrix} & \text{vector} & \text{vector} \\ (m \times n) & (n \times 1) & = (m \times 1) \end{matrix}$$

$$b_i = \sum_j^n x_j a_{ij}$$

Compatibility – the number of columns in the matrix **must** equal the number of rows in the vector.

Example: Matrix Times a Vector

```
m1 = [1 1 1; 2 2 2; 3 3 3];
```

```
v1 = [2 2 2]';
```

```
c1 = m1*v1;
```

```
c21 = m1(1,:) * v1;
```

```
c22 = m1(2,:) * v1;
```

```
c23 = m1(3,:) * v1;
```

```
c2 = [c21; c22; c23];
```

1	1	1
2	2	2
3	3	3

 *

2
2
2

 =

6
12
18

$$m1 * v1 = c1$$

1	1	1
---	---	---

 *

2
2
2

 =

6

$$m1(1,:) * v1 = c21$$

2	2	2
---	---	---

 *

2
2
2

 =

12

$$m1(2,:) * v1 = c22$$

3	3	3
---	---	---

 *

2
2
2

 =

18

$$m1(3,:) * v1 = c23$$

Matrix multiplication is an extension of matrix and vector multiplication.

Consider the product of an $m \times n$ matrix A , and an $n \times p$ matrix B :

$$AB = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ b_{31} & b_{32} & \cdots & b_{3p} \\ \vdots & \vdots & & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix}$$

$$\begin{matrix} \text{matrix} & \text{matrix} & \text{matrix} \\ (m \times n) & (n \times p) & = (m \times p) \end{matrix}$$

Compatibility – the number of columns in the first matrix **must** equal the number of rows in the second matrix.

Example: Matrix Times a Matrix

```
m1 = [1 2;3 4];
```

```
m2 = [1 2;2 1];
```

```
m3 = m1*m2;
```

```
m411 = m1(1,:) * m2(:,1);
```

```
m421 = m1(2,:) * m2(:,1);
```

```
m412 = m1(1,:) * m2(:,2);
```

```
m422 = m1(2,:) * m2(:,2);
```

```
m4 = [m411 m412;m421 m422];
```

$$\begin{bmatrix} 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \boxed{11}$$

$$m1(2,:) * m2(:,1) = m421$$

⚠: * and .* are different!

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 11 & 10 \end{bmatrix}$$

$$m1 * m2 = m3$$

$$\begin{bmatrix} 1 & 2 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \boxed{5}$$

$$m1(1,:) * m2(:,1) = m411$$

$$\begin{bmatrix} 1 & 2 \end{bmatrix} * \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \boxed{4}$$

$$m1(1,:) * m2(:,2) = m412$$

$$\begin{bmatrix} 3 & 4 \end{bmatrix} * \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \boxed{10}$$

$$m1(2,:) * m2(:,2) = m422$$

Example: Matrix Times a Matrix or Vector

Q. Write a script to compute A, B, C, D using m1 and v1 .

$$m1 = \begin{bmatrix} 2 & 4 \\ 3 & 2 \\ 5 & 3 \end{bmatrix} \quad v1 = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 3 & 2 \\ 5 & 3 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 & 3 & 5 \\ 4 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \quad C = \begin{bmatrix} 2 & 3 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 3 \end{bmatrix} \begin{bmatrix} 2 & 3 & 5 \\ 4 & 2 & 3 \end{bmatrix}$$

```
m1 = [2 4; 3 2; 5 3];
```


```
v1 = [1; 3; 1];
```

```
A = v1'*m1
```

```
B = m1'*v1
```

```
C = m1(:,1)'*v1
```

```
D = v1(1:2)'*m1'
```

: A' is a transpose of A.

Character and Character Vector

- A ***character*** is a single character in single quotes
- A character vector is sequences of characters in single quotes, e.g. 'hello and how are you?'
- **A character vector is a vector in which every element is a single character.**

```
ch_vec = 'MATLAB';  
  
n_ch = numel(ch_vec);  
  
ch3 = ch_vec(3);  
ch4 = ch_vec(1:3);  
  
ch_vec1 = [ch_vec ' is fun.']
```

Name	Value
ch_vec	'MATLAB'
n_ch	6
ch3	'T'
ch4	'MAT'
ch_vec1	'MATLAB is fun.'

- The numeric type cast function can also convert a character to its equivalent numeric value.
- All characters are mapped to equivalent numeric values.

```
ch1 = 'a';
```

```
ch2 = 'b';
```

```
ch3 = 'd';
```

```
ch_vec1 = 'abc';
```

```
double(ch1)
```

```
char(97)
```

```
char(98)
```

```
double('d')
```

```
char([97 99])
```

```
'ab' + 1
```

```
'cd' - 1
```

```
char('cd'-1)
```

```
ch_vec1 + 1
```

```
char(ch_vec1 + 1)
```

```
97
```

```
'a'
```

```
'b'
```

```
100
```

```
'ac'
```

```
[98 99]
```

```
[98 99]
```

```
'bc'
```

```
[98 99 100]
```

```
'bcd'
```

3D Matrices

- A three dimensional matrix has dimensions $m \times n \times p$
- For example, we can create a $3 \times 4 \times 2$ matrix of random integers; there are 2 layers, each of which is a 3×4 matrix

```
mat3D = zeros(3, 4, 2);  
  
mat1 = [1 2 3 4; 5 6 7 8; 9 10 11 12];  
mat2 = [13 14 15 16; 17 18 19 20; 21 22 23 24];  
  
mat3D(:,:,1) = mat1;  
mat3D(:,:,2) = mat2;
```

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

mat1

13	14	15	16
17	18	19	20
21	22	23	24

mat2


	13	14	15	16
1	2	3	4	20
5	6	7	8	24
9	10	11	12	

mat3D

3D Matrices (Continue)

```
val1 = mat3D(1, 3, 1);  
  
val2 = mat3D(1, 1, 2);  
  
mats = mat3D(1:2, 1:2, 1);
```

Name	Value
mat3D	3x4x2 double
val1	3
val2	13
mats	[1 2; 5 6]

: Large matrices are not printed out in *Workspace*.

Challenging

mat3D

				13	14	15	16
				17	18	19	20
				21	22	23	24
1	2	3	4				
5	6	7	8				
9	10	11	12				

Element

				(1,1,2)	(1,2,2)	(1,3,2)	(1,4,2)
				(2,1,2)	(2,2,2)	(2,3,2)	(2,4,2)
				2)	(3,3,2)	(3,4,2)	
(1,1,1)	(1,2,1)	(1,3,1)	(1,4,1)				
(2,1,1)	(2,2,1)	(2,3,1)	(2,4,1)				
(3,1,1)	(3,2,1)	(3,3,1)	(3,4,1)				

Subindex

Linear Indexing

Challenging

- Linear indexing: only using one index into a matrix
- MATLAB will unwind it column-by-column going from top to bottom.

```
mat1 = [1 5 3 7; 3 2 2 4; 4 2 8 9];  
val1 = mat1(2, 1)  
val2 = mat1(2)  
val3 = mat1(3, 4)  
val4 = mat1(12)  
  
vec1 = mat1(3:5)
```

Name	Value
val1	3
val2	3
val3	9
val4	9
vec1	[4 5 2]

1	5	3	7
3	2	2	4
4	2	8	9

Element

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)

Subscripted indexing

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

Linear indexing

Function: **reshape(x, sz)**

Challenging

`B = reshape(A, sz)` reshapes `A` using the size vector, `sz`, to define `size(B)`. `sz` must contain at least 2 elements, product of elements in `sz` must be the same as `numel(A)`.

```
vec = 3:14; % 12 elements
```

```
mat0 = reshape(vec, [2 6]);
```

```
mat1 = reshape(vec, [3 4]);
```

```
mat2 = reshape(mat1, [6 2]);
```

mat1

3	6	9	12
4	7	10	13
5	8	11	14

vec

3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	----	----	----	----	----

mat0

3	5	7	9	11	13
4	6	8	10	12	14

mat2

3	9
4	10
5	11
6	12
7	13
8	14

Types of Errors

Syntax error

```
a1 = [1 2 3];  
3 = a1 + 2;
```

Error: Incorrect use of '=' operator. To assign a value to a variable, use '='. To compare values for equality, use '=='.

```
a@2 = [1 2 3];
```

Error: Invalid expression. Check for missing multiplication operator, missing or unbalanced delimiters, or other syntax error.

Runtime error

```
a2 =[11 2 3];  
b2 = a2(4) + 1;
```

Index exceeds the number of array elements (3).

```
A = [1 2;3 4];  
val1 = A(3,4);
```

Index in position 1 exceeds array bounds (must not exceed 2).

Logical error

```
A = [1 2;3 4];  
B = [3 4;5 6];
```

```
m1 = A.*B;  
m2 = A*B;
```

Example: Types of Errors



Q. Which of the following scripts occur run-time or syntax error?

(1)	<pre>mat1 = ones(10,10); mat1(1:2, 5:10) = [];</pre>
(2)	<pre>mat1 = ones(10,10); mat1(1, 5) = [];</pre>
(3)	<pre>mat1 = ones(10,10); mat1(1, 1:2) = 3;</pre>
(4)	<pre>mat1 = ones(10,10); mat1 = [1 2; 3 4];</pre>

1. (1), (2)
2. (1), (2), (3)
3. (1), (2), (4)
4. (3), (4)
5. (1), (3), (4)

Example (Midterm in S19)



Q. Write a code to compute `val1`, `val2`, and `mat1` using `A`, `B`, `r`, and `c`

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} \quad \mathbf{r} = [r_1 \quad r_2 \quad r_3 \quad r_4]$$

$$\mathbf{val1} = c_1 r_1 + c_2 r_2 + c_3 r_3$$

$$\mathbf{val2} = [c_1 r_1 \quad c_2 r_2 \quad c_3 r_3 \quad c_4 r_4]$$

$$\mathbf{mat1} = \begin{bmatrix} b_{11}r_1 & b_{12}r_2 & b_{13}r_3 & b_{14}r_4 \\ b_{21}r_1 & b_{22}r_2 & b_{23}r_3 & b_{24}r_4 \\ b_{31}r_1 & b_{32}r_2 & b_{33}r_3 & b_{34}r_4 \end{bmatrix}$$

```
val1 = r(1:3)*c(1:3);  
val2 = r.*c';  
  
mat1 = B.*[r; r; r; r];
```