Computer Programming: Mathematical Operations with Arrays

Asst. Prof. Dr. Yalçın İşler
Izmir Katip Celebi University

Outline

- Addition and Subtraction
- Multiplication
- Division
- Element-by-Element Operations
- Arrays in Functions
- Analyzing Arrays
- Random Numbers
- Examples

Addition and Subtraction

- The operations + (addition) and – (subtraction) can be used to add (subtract) arrays of identical size (the same number of rows and columns), and to add (subtract) a scalar to an array. When two arrays are involved, the sum, or the difference of two arrays is obtained by adding, or subtracting, their corresponding elements.

```
>> VectA=[8 5 4]; VectB=[10 2 7];
```

Define two vectors.

```
>> VectC=VectA+VectB
```

Define a vector VectC that is equal to VectA + VectB.

```
VectC =
    18       7      11
>> A=[5 -3 8; 9 2 10]
A =
     5      -3       8
     9       2      10
>> B=[10 7 4; -11 15 1]
```

Define two 2×3 matrices A and B.

```
B =
    10       7       4
   -11      15       1
>> A-B
```

Subtracting matrix B from matrix A.

```
ans =
     -5    -10       4
     20    -13       9
>> C=A+B
```

Define a matrix C that is equal to A + B.

```
C =
    15       4      12
    -2      17      11
>> C-8
```

The number 8 is subtracted from the matrix C.

```
ans =
     7      -4       4
   -10       9       3
```

Define a vector named VectA.

```
>> VectA=[1 5 8 -10 2]
VectA =
     1     5     8   -10     2

>> VectA+4
```

Add the scalar 4 to VectA.

```
ans =
     5     9    12    -6
```

4 is added to each element of VectA.

Define a 2 × 3 matrix A.

```
>> A=[6 21 -15; 0 -4 8]
A =
     6    21   -15
     0    -4     8

>> A-5
```

Subtract the scalar 5 from A.

```
ans =
     1    16   -20
    -5    -9     3
```

5 is subtracted from each element of A.

Multiplication

- The multiplication operation * is executed by MATLAB according to the rules of linear algebra. This means that if *A* and *B* are two matrices, the operation *A***B* can be carried out only if the number of columns in matrix *A* is equal to the number of rows in matrix *B*. The result is a matrix that has the same number of rows as *A* and the same number of columns as *B*.

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

A =
     1     4     2
     5     7     3
     9     1     6
     4     2     8
```

Define a 4 × 3 matrix A.

```
>> B=[6 1; 2 5; 7 3]

B =
     6     1
     2     5
     7     3
```

Define a 3 × 2 matrix B.

```
>> C=A*B

C =
    28    27
    65    49
    98    32
    84    38
```

Multiply matrix A by matrix B and assign the result to variable C.

```
>> D=B*A
??? Error using ==> *
Inner matrix dimensions must agree.
```

Trying to multiply B by A, B*A, gives an error since the number of columns in B is 2, and the number of rows in A is 4.

```
>> F=[1 3; 5 7]

F =
     1     3
     5     7
```

Define two 2 × 2 matrices F and G.

```
>> G=[4 2; 1 6]
G =
     4     2
     1     6
>> F*G
ans =
     7    20
    27    52
>> G*F
ans =
    14    26
    31    45
>> AV=[2 5 1]
AV =
     2     5     1
>> BV=[3; 1; 4]
BV =
     3
     1
     4
>> AV*BV
ans =
    15
>> BV*AV
ans =
     6    15     3
     2     5     1
     8    20     4
>>
```

Multiply F*G

Multiply G*F

Note that the answer for G*F is not the same as the answer for F*G.

Define a three-element row vector AV.

Define a three-element column vector BV.

Multiply AV by BV. The answer is a scalar. (Dot product of two vectors.)

Multiply BV by AV. The answer is a 3×3 matrix.

```
>> A=[2 5 7 0; 10 1 3 4; 6 2 11 5]

A =
     2     5     7     0
    10     1     3     4
     6     2    11     5
```

Define a 3 × 4 matrix A.

```
>> b=3

b =
     3
```

Assign the number 3 to the variable b.

```
>> b*A
```

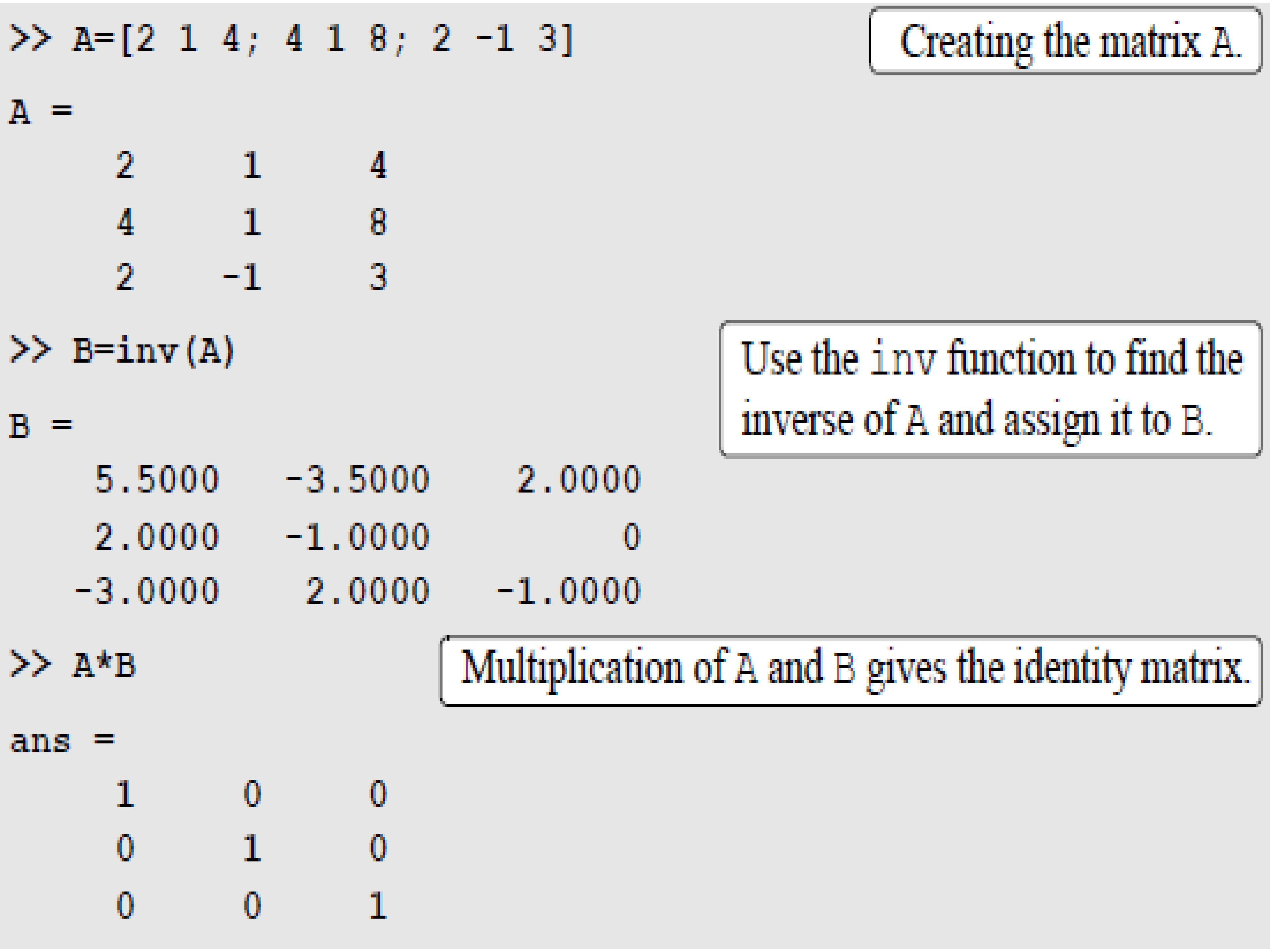
Multiply the matrix A by b. This can be done by either typing b*A or A*b.

```
ans =
     6    15    21     0
    30     3     9    12
    18     6    33    15

>> C=A*5

C =
    10    25    35     0
    50     5    15    20
    30    10    55    25
```

Multiply the matrix A by 5 and assign the result to a new variable C. (Typing C = 5*A gives the same result.)

Division

- The identity matrix (I) is a square matrix in which the diagonal elements are 1’s, and the rest of the elements are 0’s.
- It can be created in MATLAB with the eye command.
- The matrix *B* is the inverse of the matrix *A* if when the two matrices are multiplied the product is the identity matrix. Both matrices must be square and the multiplication order can be AB or BA.
- AB = BA = I
- Not every matrix has an inverse. A matrix has an inverse only if it is square and its determinant is not equal to zero.



```
>> A=[2 1 4; 4 1 8; 2 -1 3]

A =

     2     1     4
     4     1     8
     2    -1     3
```

Creating the matrix A.

```
>> B=inv(A)

B =

    5.5000   -3.5000    2.0000
    2.0000   -1.0000         0
   -3.0000    2.0000   -1.0000
```

Use the `inv` function to find the inverse of A and assign it to B.

```
>> A*B

ans =

     1     0     0
     0     1     0
     0     0     1
```

Multiplication of A and B gives the identity matrix.

Left and Right Divisions

- The left division is used to solve the matrix equation $AX = B$ to find X where X and B are column vectors.

$$AX = B$$
$$A^{-1}AX = A^{-1}B$$
$$IX = A^{-1}B$$
$$X = A^{-1}B$$

$$\boldsymbol{X = A \backslash B}$$

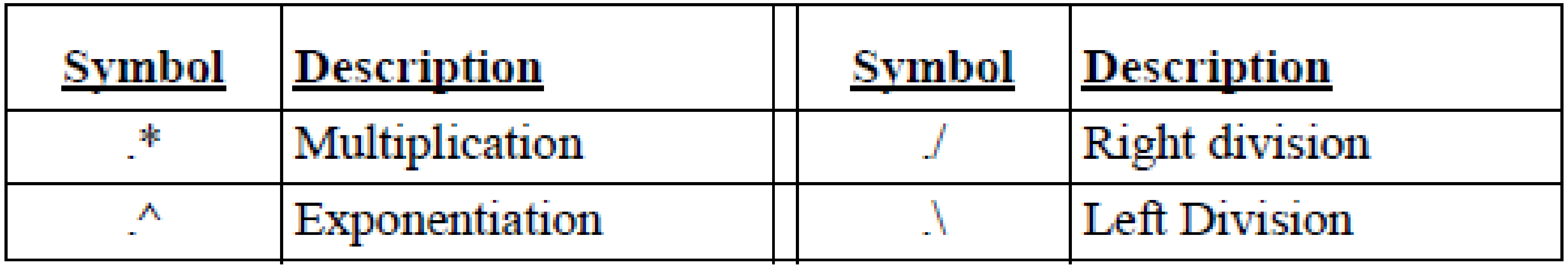
- The right division is used to solve the matrix equation $XC = D$ to find X where X and D are column vectors.

$$XC = D$$
$$XCC^{-1} = DC^{-1}$$
$$XI = DC^{-1}$$
$$X = DC^{-1}$$

$$\boldsymbol{X = D/C}$$

Element-by-Element Operations

- when the regular symbols for multiplication and division are used with arrays (* and /), the mathematical operations follow the rules of linear algebra.
- Element-by-element multiplication, division, and exponentiation of two vectors or matrices is entered in MATLAB by typing a period in front of the arithmetic operator.



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

```
>> A=[2 6 3; 5 8 4]
A =
     2     6     3
     5     8     4
```

Define a 2×3 array A.

```
>> B=[1 4 10; 3 2 7]
B =
     1     4    10
     3     2     7
```

Define a 2×3 array B.

```
>> A.*B
ans =
     2    24    30
    15    16    28
```

Element-by-element multiplication of array A by B.

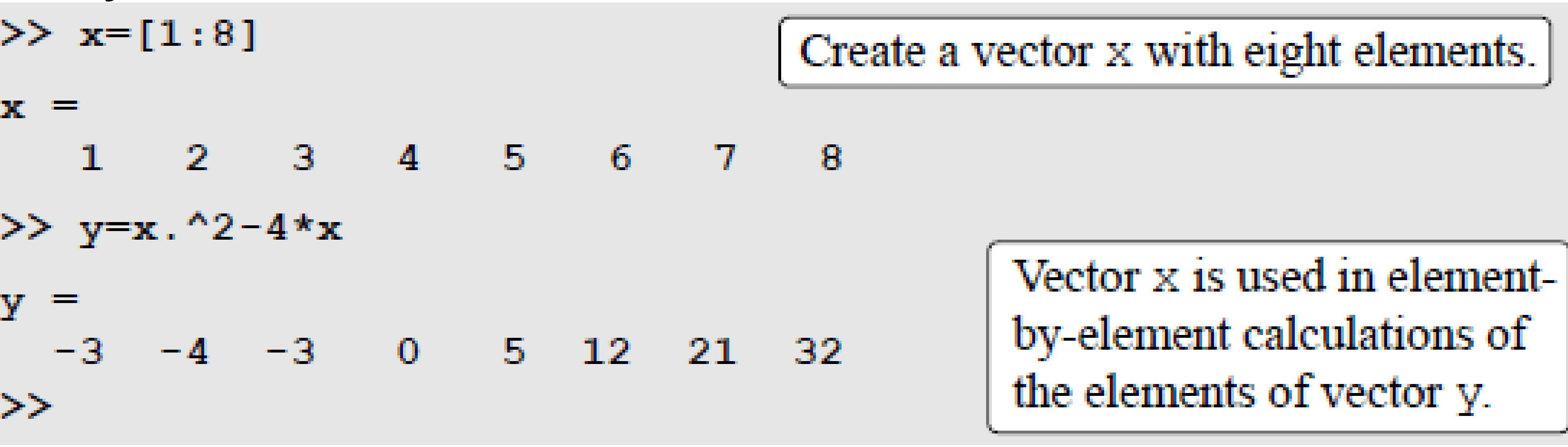
```
>> C=A./B
C =
    2.0000    1.5000    0.3000
    1.6667    4.0000    0.5714
```

Element-by-element division of array A by B. The result is assigned to variable C.

```
>> B.^3

ans =
     1    64  1000
    27     8   343
```

Element-by-element exponentiation of array B. The result is an array in which each term is the corresponding term in B raised to the power of 3.

- $y = x^2 - 4x$

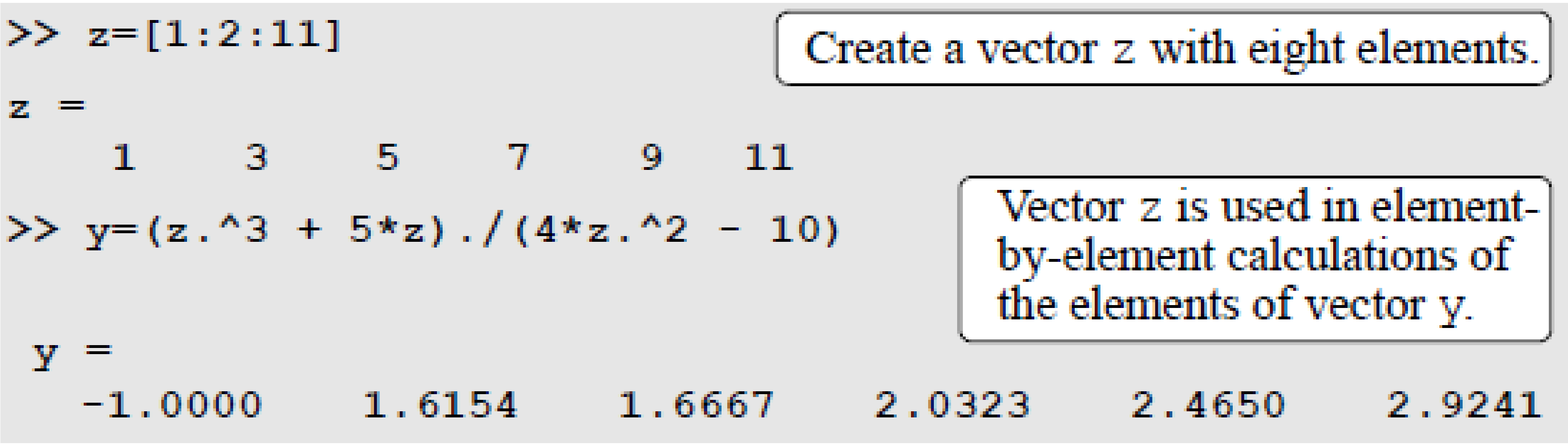


```
>> x=[1:8]

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.

- $y = \frac{z^3+5z}{4z^2-10}$



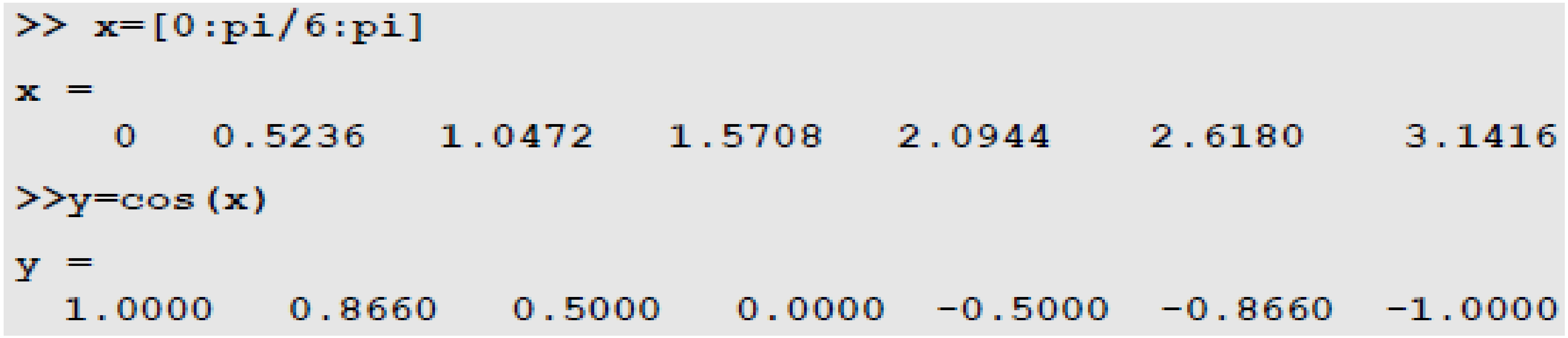
```
>> z=[1:2:11]

z =
      1      3      5      7      9     11

>> y=(z.^3 + 5*z)./(4*z.^2 - 10)

 y =
    -1.0000    1.6154    1.6667    2.0323    2.4650    2.9241
```

Create a vector z with eight elements.

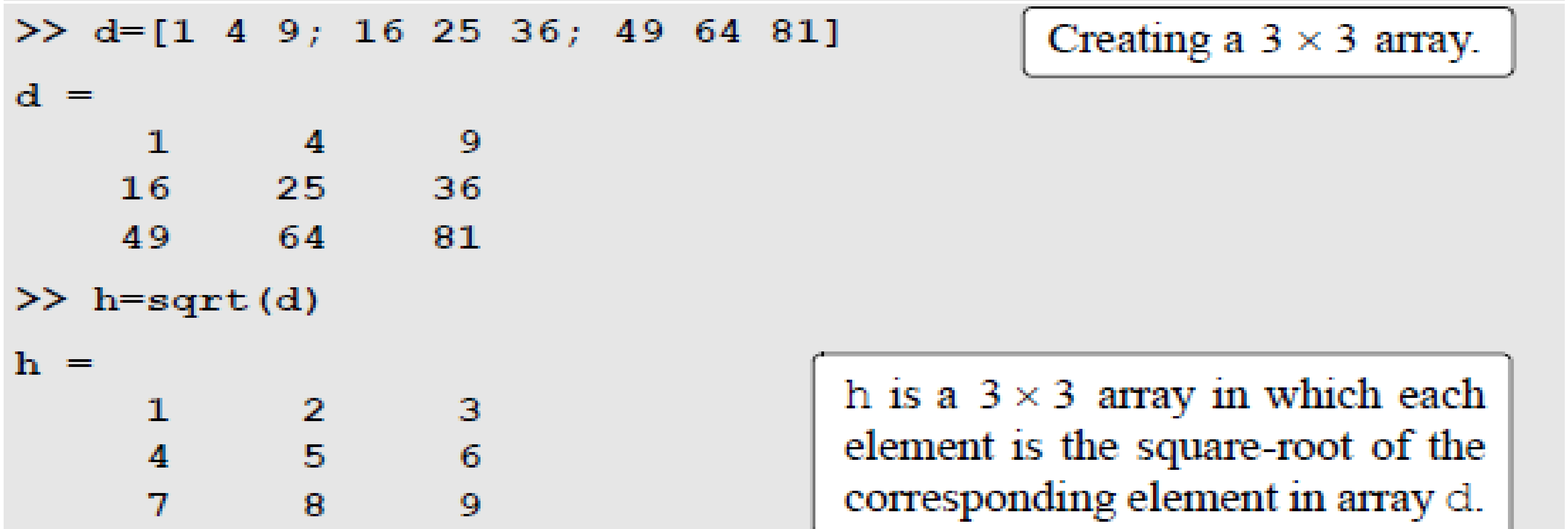
Vector z is used in element-by-element calculations of the elements of vector y.

Arrays in Functions

The built-in functions in MATLAB are written such that when the argument (input) is an array, the operation that is defined by the function is executed on each element of the array. The feature of MATLAB, in which arrays can be used as arguments in functions, is called vectorization.



```
>> 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
```



```
>> d=[1 4 9; 16 25 36; 49 64 81]

d =
      1      4      9
     16     25     36
     49     64     81

>> h=sqrt(d)

h =
      1      2      3
      4      5      6
      7      8      9
```

Creating a 3 × 3 array.

h is a 3 × 3 array in which each element is the square-root of the corresponding element in array d.

Analyzing Arrays

Function	Description	Example

`mean(A)`	If `A` is a vector, returns the mean value of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> mean(A)`
		`ans =`
		`     5`

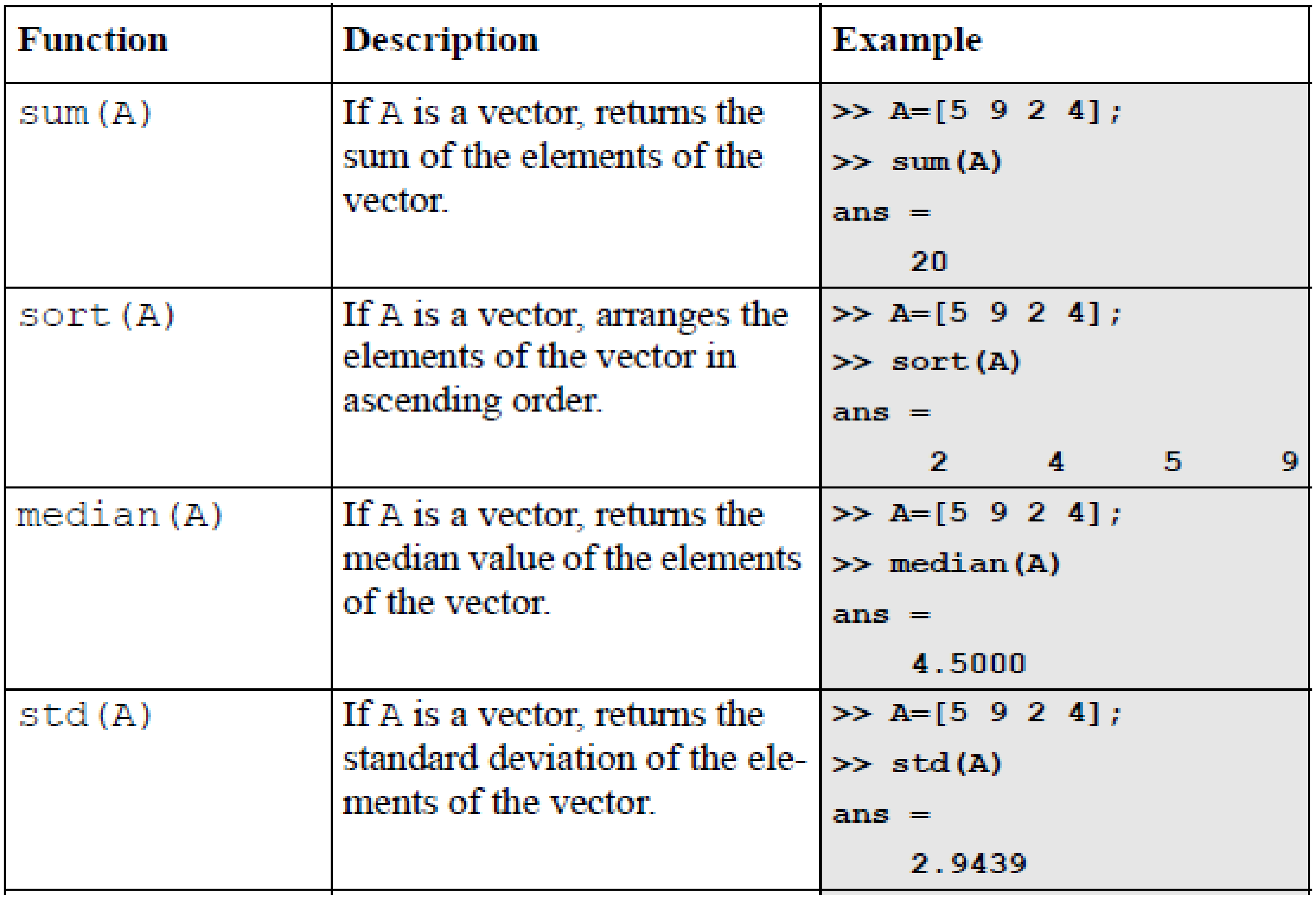
`C=max(A)`	If `A` is a vector, `C` is the largest element in `A`. If `A` is a matrix, `C` is a row vector containing the largest element of each column of `A`.	`>> A=[5 9 2 4 11 6 11 1];`
		`>> C=max(A)`
		`C =`
		`    11`

`[d,n]=max(A)`	If `A` is a vector, `d` is the largest element in `A`, `n` is the position of the element (the first if several have the max value).	`>> [d,n]=max(A)`
		`d =`
		`    11`
		`n =`
		`     5`

`min(A)`	The same as `max(A)`, but for the smallest element.	`>> A=[5 9 2 4];`
		`>> min(A)`
		`ans =`
		`     2`

`[d,n]=min(A)`	The same as `[d,n]= max(A)`, but for the smallest element.	

Analyzing Arrays (cont'd)



Function	Description	Example

`sum(A)`	If `A` is a vector, returns the sum of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> sum(A)`
		`ans =`
		`    20`

`sort(A)`	If `A` is a vector, arranges the elements of the vector in ascending order.	`>> A=[5 9 2 4];`
		`>> sort(A)`
		`ans =`
		`     2     4     5     9`

`median(A)`	If `A` is a vector, returns the median value of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> median(A)`
		`ans =`
		`    4.5000`

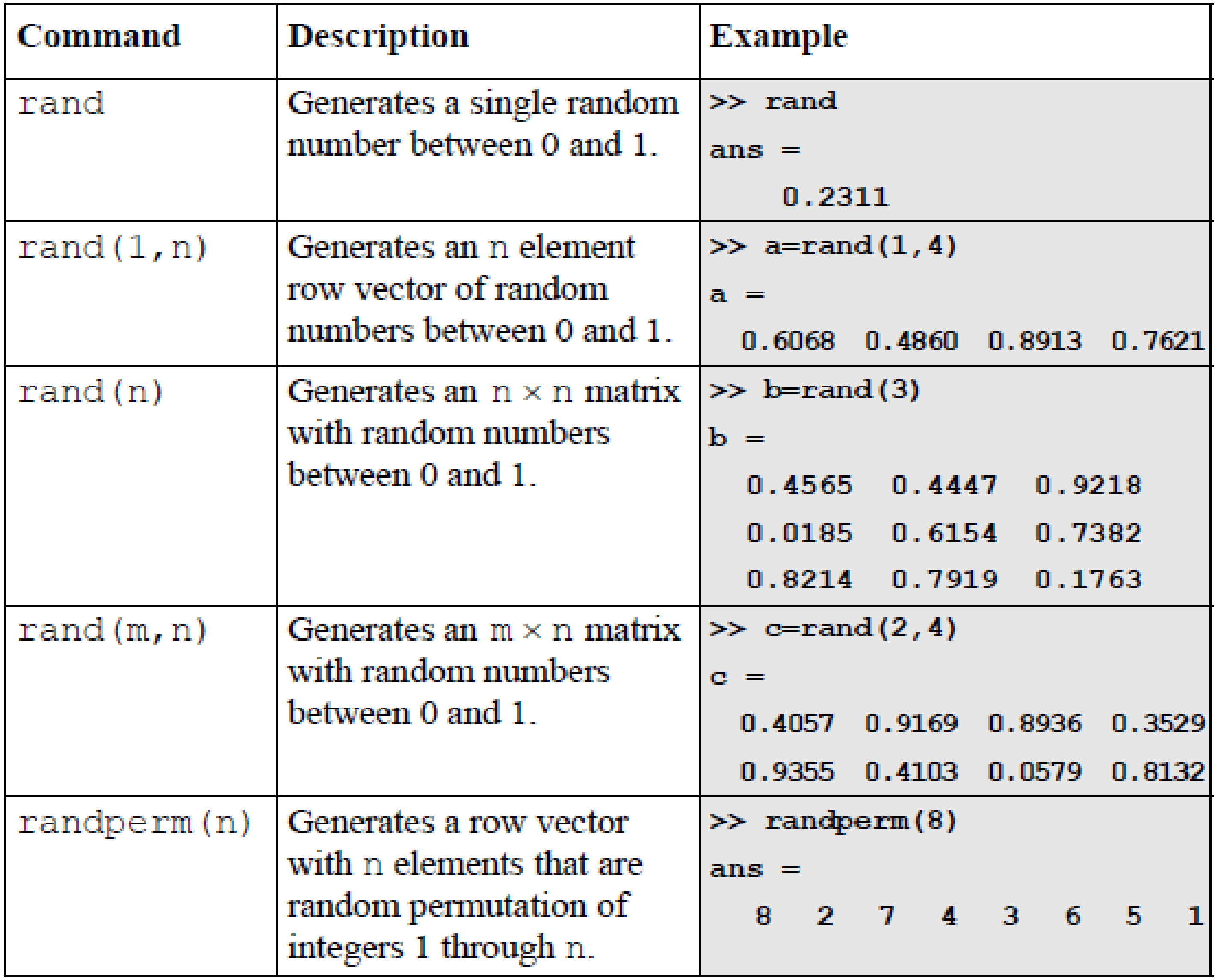
`std(A)`	If `A` is a vector, returns the standard deviation of the elements of the vector.	`>> A=[5 9 2 4];`
		`>> std(A)`
		`ans =`
		`    2.9439`

Analyzing Arrays (cont’d)

Function	Description	Example
`det(A)`	Returns the determinant of a square matrix `A`.	`>> A=[2 4; 3 5];` `>> det(A)` `ans =` `-2`
`dot(a,b)`	Calculates the scalar (dot) product of two vectors `a` and `b`. The vectors can each be row or column vectors.	`>> a=[1 2 3];` `>> b=[3 4 5];` `>> dot(a,b)` `ans =` `26`
`cross(a,b)`	Calculates the cross product of two vectors `a` and `b`, (`axb`). The two vectors must have 3 elements.	`>> a=[1 3 2];` `>> b=[2 4 1];` `>> cross(a,b)` `ans =` `-5 3 -2`
`inv(A)`	Returns the inverse of a square matrix `A`.	`>> A=[2 -2 1; 3 2 -1; 2 -3 2];` `>> inv(A)` `ans =` `0.2000 0.2000 0` `-1.6000 0.4000 1.0000` `-2.6000 0.4000 2.0000`

Random Numbers

Simulations of many physical processes and engineering applications frequently requires using a number (or a set of numbers) that has a random value. MATLAB has two commands rand and randn that can be used to assign random numbers to variables.



Command	Description	Example

`rand`	Generates a single random number between 0 and 1.	`>> rand`
		`ans =`
		`    0.2311`

`rand(1,n)`	Generates an n element row vector of random numbers between 0 and 1.	`>> a=rand(1,4)`
		`a =`
		`  0.6068  0.4860  0.8913  0.7621`

`rand(n)`	Generates an n × n matrix with random numbers between 0 and 1.	`>> b=rand(3)`
		`b =`
		`  0.4565  0.4447  0.9218`
		`  0.0185  0.6154  0.7382`
		`  0.8214  0.7919  0.1763`

`rand(m,n)`	Generates an m × n matrix with random numbers between 0 and 1.	`>> c=rand(2,4)`
		`c =`
		`  0.4057  0.9169  0.8936  0.3529`
		`  0.9355  0.4103  0.0579  0.8132`

`randperm(n)`	Generates a row vector with n elements that are random permutation of integers 1 through n.	`>> randperm(8)`
		`ans =`
		`  8  2  7  4  3  6  5  1`

Laboratory Session

Do sample applications in Chapter 3 of the textbook.

Homework #5

Not later than the next week:

Solve problems 2, 4, 10, 12, 17, 18, and 19 from the Chapter 3 of the textbook using Matlab.