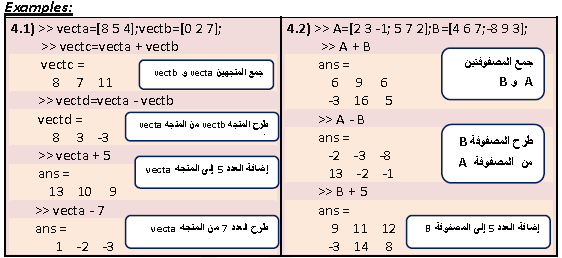
****

Once variables are created in MATLAB they can be used in a wide variety of mathematical operations.

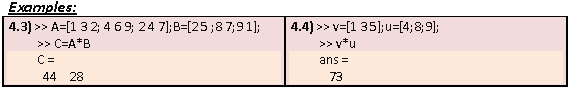
**4.1 Addition and Subtraction**

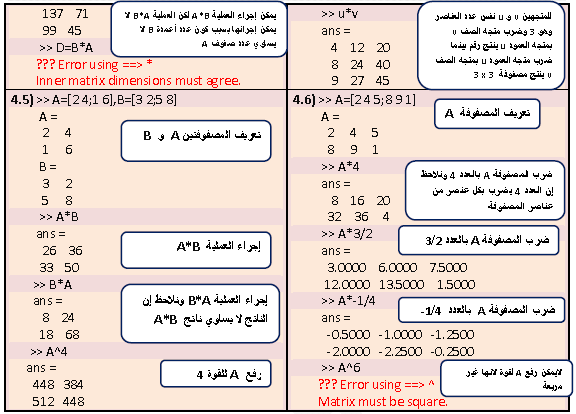
The operations + (addition) and – (subtraction) can be used with arrays of identical size (the same number of rows and columns). The sum, or the difference of two arrays is obtained by adding, or subtracting, their corresponding elements. When a scalar (number) is added to, or subtracted from, an array,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



**4.2 Array 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. The product of the multiplication of two square matrices (they must be of the same size) is also a square matrix of the same size. However, the multiplication of matrices is not commutative. This means that if A and B are matrices, then A \* B ≠ B \* A . Also, the power operation can only be executed with a square matrix (since A \* A can be carried out only if the number of columns in the first matrix is equal to the number of rows in the second matrix). Two vectors can multiply each other only if both have the same number of elements, and one is a row vector and the other is a column vector. The multiplication of a row vector times a column vector gives a 1 x 1 matrix, which is a scalar. This is the dot product of two vectors. The multiplication of a column vector times a row vector, both with n elements gives an n x n matrix.When an array is multiplied by a number, each element in the array is multiplied by the number.

****

****

Linear Algebra rules of array multiplication provide a convenient way for writing a system of linear equations. For example, the following system of three equations with three unknowns:



can be written in a matrix form by:



and in matrix notation by:



**4.3 Array Division**

The division operation is also associated with the rules of linear algebra. The division operation can be explained with the help of the identity matrix and the inverse operation. MATLAB has two types of array division, which are the right division and the left division.

**Definition 1:**

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. As was shown before, an identity matrix can be created in MATLAB with the **eye** command. When the identity matrix multiplies another matrix (or vector), that matrix (or vector) is unchanged. If a matrix A is square, it can be multiplied by the identity matrix I, from the left or from the right: AI = IA = A.

**Example 4.7:**

>> I=eye(3)

**تعريف مصفوفة واحدية من الدرجة الثالثة**

I =

1 0 0

0 1 0

0 0 1

>> v=[4 7 2]

**تعريف متجه v**

v =

4 7 2

>> v\*I

**V\*I=Vبسبب تأثير المصفوفة الواحدية**

ans =

4 7 2

**لا يمكن إجراء العمليةv\*I بسبب عدم تناسق الأبعاد**

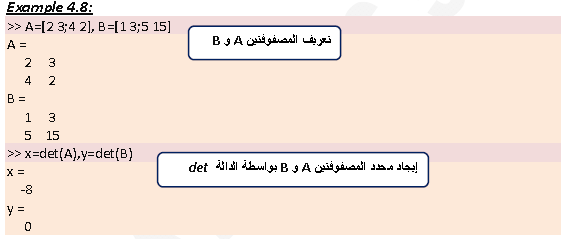
>> I\*v

??? Error using ==> \*

Inner matrix dimensions must agree.

**Definition 2:**

Determinant is a function that associates with each square matrix A a number, called the determinant of the matrix. The determinant is typically denoted by **det(A)** or **|A**|. The determinant is calculated according to specific rules. The determinant of a square matrix A can be calculated with the **det(A)** command. If A is a square matrix and **|A| = 0** , then A is called **singular** matrix.



**Definition 3:**

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 non‐singular and the multiplication order can be BA or AB. That is AB = BA = I. Obviously B is the inverse of A, and A is the inverse of B.

The inverse of a matrix A is typically written as A-1. In MATLAB the inverse of a matrix can be obtained either by raising A to the power of ‐1, A-1, or with the **inv(A)** function.

**Example 4.9:**

>> A=[2 3;4 2]; B=[1 3;5 15];

**ايجاد معكوس المصفوفة A بواسطة الدالة inv**

>> C=inv(A)

C =

‐0.2500 0.3750

0.5000 ‐0.2500

>> A\*C

ans =

1 0

**لكون المصفوفتين احدهما معكوس الاخرC\*A=I**

0 1

>> C\*A

ans =

1 0

0 1

>> D=A^‐1

**طريقة أخرى لإيجاد معكوس المصفوفة A بواسطة رفع A للقوة -1**

D =

‐0.2500 0.3750

0.5000 ‐0.2500

>> inv(B)

Warning: Matrix is singular to working precision.

(Type "warning off MATLAB:singularMatrix" to suppress this warning.)

ans = 

Inf Inf

Inf Inf

**4.3.1 Left Division \**

The left division is used to solve the matrix equation AX=B, in this equation X and B are

column vectors. This equation can be solved by multiplying on the left both sides by the

inverse of A:

A-1A **.** X = A-1 **.** B

The left‐hand side of this equation is X since:

A-1 A **.** X = I X = X

So, the solution of AX = B is: X= A-1 **.** B

In MATLAB the last equation can be written by using the left division character: X = A\ B. It should be pointed out here that although the last two operations appear to give the same result, the method by which the MATLAB calculates X is different. In the first, MATLAB calculates A-1 and then use it to multiply B. In the second, (left division) the solution X is obtained numerically with a method that is based on the Gauss elimination method. The left division method is recommended for solving a set of linear equations because the calculation of the inverse may be less accurate than the Gaussian elimination method when large matrices are involved.

**4.3.2 Right Division /**

The right division is used to solve the matrix equation XC = D. In this equation X and D

are row vectors. This equation can be solved by multiplying on the right both sides by the

inverse of C:

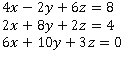
X C**.** C-1 =D **.** C-1

which gives: X = D **.** C-1

In MATLAB the last equation can be written by using the right division character:X = D /C.

**Example 4.10:**

Use matrix operations to solve the following linear equations system.



Solution:

The above system of equations can be written in the matrix form AX=B

 . The solution of this form is:

>> A=[4 ‐2 6;2 8 2;6 10 3];

>> B=[8;4;0];

**حل النظام باستخدام القسمة من اليسار**

>> X=A\B

X =

‐1.8049

0.2927

2.6341

طريقة أخرى لحل النظام باستخدام معكوس المصفوفة

>> X=inv(A)\*B

X =

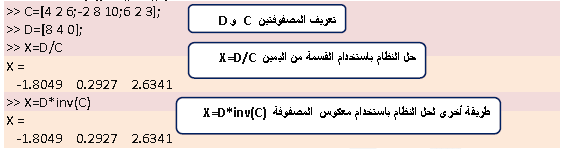
‐1.8049

0.2927

2.6341

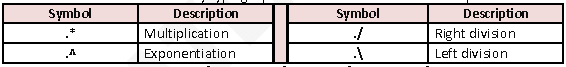
Also, the above system of equations can be written in the matrix form XC=D

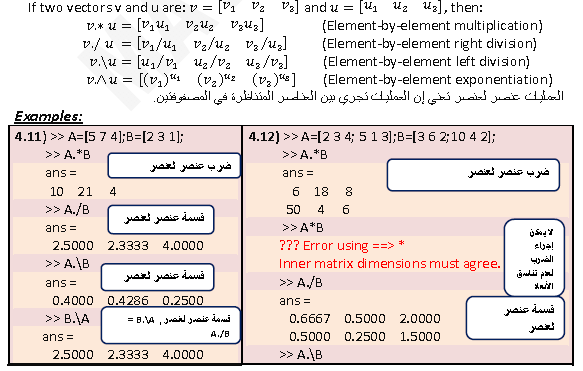


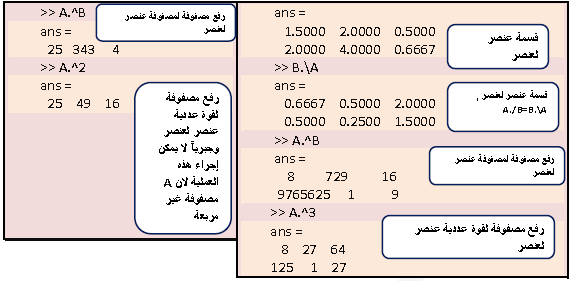


**4.4 Element‐By‐Element Operations**

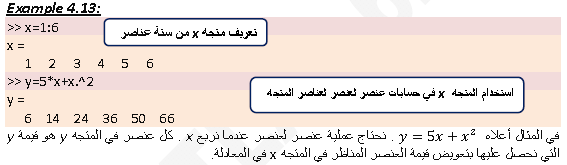
There are many situations that require element‐by‐element operations. These operations are carried out on each of the elements of the array (or arrays). Addition and subtraction are already, by definition, element‐by‐element operations since when two arrays are added (or subtracted) the operation is executed with the elements that are in the same position in the array. Element‐by‐element operations can only be done with arrays of the same size. Element‐by‐element multiplication, division, and multiplication of two vectors or matrices is entered in MATLAB by typing a period in front of the arithmetic operator.



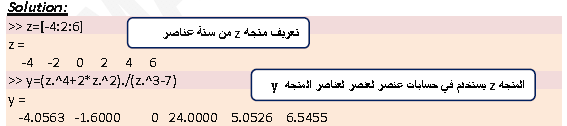




Element‐by‐element calculations are very useful for calculating the value of a function at many values of its argument. This is done by first defining a vector that contains values of the independent variables, and then using this vector in element‐by‐element computations to create a vector in which each element is the corresponding value of the function.



**Example 4.14:**For the function y = . Calculate the value of y for the following values of z:‐4,‐2,0,2,4,6.



**4.5 Using Arrays in MATLAB Built‐in Math Functions(Vectorization)**

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 result (output) from such an operation is an array in which each element is calculated by entering the corresponding element of the argument (input) array into the function. The feature of MATLAB, in which arrays can be used as argument in functions, is called **vectorization.**

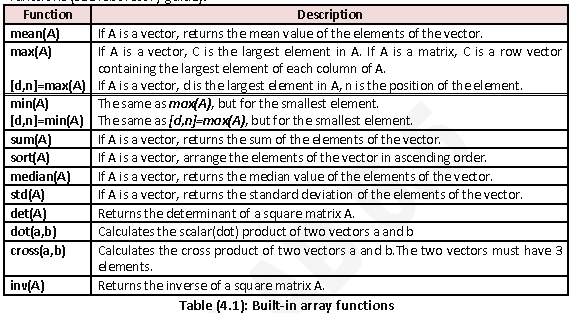
يمكن أن تكون المصفوفات متغيرات لدوال Matlab المعرفة وحينها تطبق الدالة على كل عنصر في المصفوفة.

**Examples:**

|  |  |
| --- | --- |
| 4.15) >> x=[0:pi/4:pi]  x =  0 0.7854 1.5708 2.3562 3.1416  >> y=sin(x)  y =  0 0.7071 1.0000 0.7071 0.0000  **تطبيق الدالة sin على كل عنصر من عناصر المتجه x** | 4.16) >> A=[2 4 9;3 25 49]  A =  **تطبيق الدالة sqrt على كل عنصر من عناصر المصفوفة A**  2 4 9  3 25 49  >> z=sqrt(A)  z =  1.4142 2.0000 3.0000  1.7321 5.0000 7.0000 |

**4.6 Built‐In Functions for Analyzing Arrays**

MATLAB has many built‐in functions for analyzing arrays. Table (4.1) lists some of these functions (see laboratory guide).



**4.7 Generation of Random Numbers**

Simulation of many physical processes and engineering applications frequently requires using a number (or 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.

**4.7.1 The rand command**

The **rand** command generates uniformly distributed numbers with values between 0 and

1. The command can be used to assign these numbers to a scalar, a vector, or a matrix as

shown in table (4.2) below.

الايعاز rand يولد أعداد عشوائية موزعة توزيعاً منتظماً بين 0 و 1.

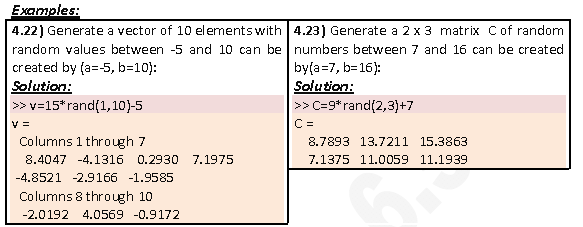
|  |  |  |
| --- | --- | --- |
| **Examples** | **Description** | **Command** |
| 4.17) >> rand  ans =  0.9501 | Generates a single random number.  توليد عدد عشوائي وحيد. | rand |
| 4.18) >> rand(1,3)  ans =  0.2311 0.6068 0.4860 | Generates an n elements row vector of random numbers between 0 and 1.  توليدn من الأعداد العشوائية التي قيمها بين 0 و 1 كمتجه صف. | rand(1, n) |
| 4.19) >> rand(2)  ans =  0.8913 0.4565  0.7621 0.0185 | Generates an n x n matrix of random numbers between 0 and 1.  توليد مصفوفة n×n من الأعداد العشوائية التي قيمها بين 0 و 1. | rand(n) |
| 4.20) >> rand(2,3)  ans =  0.8214 0.6154 0.9218  0.4447 0.7919 0.7382 | Generates an m x n matrix of random numbers between 0 and 1.  توليد مصفوفة m×n من الأعداد العشوائية التي قيمها بين 0 و 1. | rand(m, n) |
| 4.21) >> randperm(5)  ans =  1 2 5 4 3 | Generates a row vector with n elements that are random permutation of integers 1 through n.  توليد متجه صف منn من الأعداد العشوائية والتي هي تباديل الاعداد من 1 الى n. | randprem(n) |

**Table (4.2): The rand command**

Random numbers that are distributed in a range (a,b) can be obtained by multiplying

rand by (b ‐ a) and adding the product to a :





Random numbers that are all integers can be generated by using one of the rounding

functions (see pp.19).

**Example 4.24:**

Generate a 2 x 4 matrix of random integers with values that range from 1 to 100.

**Solution:**

This matrix can be generated as follows:

>> A=fix(99\*rand(2,4)+1)

**نستطيع استخدام أي من دوال التدوير (التقريب) round, ceil, floorبدل الدالة fix .**

A =

38 50 43 19

83 71 31 20

**4.7.2 The randn Command**

The randn command generates normally distributed numbers with mean 0 and standard

deviation of 1. The command can be used to generate a single number, a vector, or a

matrix in the same way as the **rand** command.

الايعاز randn يستخدم لتوليد أعداد عشوائية ذات توزيع طبيعي بمتوسط 0 وانحراف معياري 1.

