محاضرة مادة الحاسبات للمرحلة الثانية

م.م رشا ياسين عبد

Arrays and Matrix Operations

Arrays and Matrix Operations

**Section**

1. The Primary MATLAB Data Structure

2. Entering Arrays and Matrices

3 .Accessing and Manipulation Array Elements

4 .Element-by-Element Array Operations

5 .Binary Matrix Operations

6 .Unary Matrix Operations

7. Multidimensional Array

**Objectives**

* After reading this chapter, you should be able to
* Create arrays and matrices.
* Access elements in arrays and matrices.
* Add, modify, and delete elements from arrays.
* Perform element-by-element arithmetic operations on arrays.
* Perform vector and matrix multiplication.
* Perform matrix exponentiation.
* Compute the transpose, determinant, and inverse of a matrix.

2.ENTERING ARRAYS AND MATRICES

You can create arrays and matrices several ways in MATLAB. You have already been shown how to enter arrays in the Command window by typing text commands. There are several other ways to enter arrays in MATLAB. You can enter arrays by loading a script file that creates the arrays. You can view and edit arrays and matrices by using a graphical user interface called the Array Editor. Finally, you can quickly enter several types of special matrices by using some of MATLAB’s built-in matrix generators.

2.1 Command Line Entry

Let us review how to enter arrays in the Command window. The syntax includes brackets for the whole array and delimiters for rows and columns. Elements in the same row are separated by commas or spaces. A new row is created by using a semicolon or a new line. The whole array is bracketed by square braces. The array

is entered as

>> [1, 3, 5; 2, 4, 6; 7, 7, 7]

ans =

1 3 5

2 4 6

7 7 7

You can also create the same array by using spaces to separate the elements in the same row instead of commas:

>> [1 3 5; 2 4 6; 7 7 7];

You can also create the same array by moving to a new line every time you designate a new row:

>> [1 3 5

2 4 6

7 7 7]

Note that the continuation symbol is not required.

Moreover, you can enter array elements in series more concisely by using the *colon operator*.

2.2 The Array Editor

The Array Editor is a graphical interface that displays the contents of workspace objects and allows you to edit them. If the Workspace window is not visible, select from the Menu bar. Enter the following command in the Command window:

>> A = 5 : 0.5 : 7

A =

5.0000 5.5000 6.0000 6.5000 7.0000

2.3 Built-In Matrix Generators

We use several types of arrays so frequently that MATLAB has provided special functions to support the generation of these arrays. We call a matrix in which all of the elements are zero a *zero matrix*. You can create a zero array or matrix by using the *zeros* function. The syntax of the *zeros* function is

zeros(*dim1*, *dim2*, *dim3*, ...)

If you specify a single, scalar parameter *dim1*, MATLAB returns a square zero matrix of order *dim1*. For example, the following command creates a matrix of zeros:

>>A = zeros(2)

A =

 0 0

 0 0

If you specify multiple parameters, a multidimensional zero array of order

is returned. The following command creates a matrix of zeros:

>> A = zeros(2, 4)

A =

0 0 0 0

0 0 0 0

We call a matrix in which all of the elements are the number one a *ones matrix*. You can create a ones matrix by using the *ones* function. The syntax of the *ones* function is identical to the syntax of the *zeros* function. The following command creates a matrix of ones:

>> A = ones(3, 2)

A =

 1 1

 1 1

 1 1

Similarly, you can generate an array of pseudorandom numbers by using one of MATLAB’s several random array generator functions. One of these, the *rand* function, generates an array of random numbers whose elements are uniformly distributed in the range (0, 1). A uniform distribution is one in which there is an equal probability of occurrence

for any value within the given range (0, 1)—for example,

>> A = rand(2,5)

A =

0.9501 0.6068 0.8913 0.4565 0.8214

0.2311 0.4860 0.7621 0.0185 0.4447

Another commonly used matrix form is the identity matrix. An *identity matrix* is a matrix in which every element of the main diagonal is one and every other element is zero. You can generate an identity matrix by using the *eye* function with the syntax, as shown here:

eye(*n*).Here is an example:

>> eye(3)

ans =

1 0 0

0 1 0

0 0 1

You can use two arguments to specify both dimensions. An identity matrix with ones on the diagonal and zeros elsewhere can be generated by using the syntax

eye(*m,n*).For example, we might have

>> eye(4,3)

ans =

1 0 0

0 1 0

0 0 1

0 0 0

3 .ACCESSING AND MANIPULATING ARRAY ELEMENTS

3.1 Accessing Elements of an Array

You have already accessed array elements by using subscripts. Let us review what you have learned and cover a few more tricks for accessing array elements. We will use the following two-dimensional array *A* for the next few examples:

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

A =

1 3 5

2 4 6

3 5 7

An element in a two-dimensional array can be accessed by using two subscripts, the first for the row and the second for the column; for example,

>> A(2,3)

ans =

6

You can also access elements in a two-dimensional array by using a single subscript. In this case, imagine the columns lay end to end as follows:

A = [1

2

3

3

4

5

5

6

7]

This makes more sense if we look at how MATLAB stores arrays internally. Data are stored internally in a linear sequence of memory locations. MATLAB stretches out an array into a single sequence for storage.We think of array *A* as a two-dimensional array. If *A* were stored one row at a time in memory, it might look like this:

1 3 5 2 4 6 3 5 7

This is called *row-major order*. However, if *A* were stored one column at a time in memory, it would look like the following:

1 2 3 3 4 5 5 6 7

This is called *column-major order*. MATLAB stores arrays in column-major order. MATLAB functions are written to take advantage of the underlying storage mechanism to speed up array operations.

The following examples demonstrate how to access an array element by using a single subscript:

>> A(1)

ans =

1

>> A(4)

ans =

3

>> A(8)

ans =

6

We have already used the colon operator to generate arrays. You can also use the colon operator to access multiple array elements simultaneously. You do this by using the colon operator to define a subscript range. For example, the use of 1:2:9 as a subscript returns the first, third, fifth, seventh, and ninth elements of *A*:

>> A(1:2:9)

ans =

1 3 4 5 7

When used alone, the colon denotes all rows or columns. The following command returns

all columns of row two from array *A*:

>> A(2,:)

ans =

2 4 6

The following command returns the second and third rows of the first and secondcolumns from array *A*:

>> A(2:3, 1:2)

ans =

2 4

3 5

3.2 Expanding the Size of an Array

You can dynamically expand an array simply by adding more elements—for example,

>> A = [3 5 7]

A =

3 5 7

>> A = [A 9]

A =

3 5 7 9

When appending arrays to multidimensional arrays, the newly appended parts must conform

to the dimensions of the original array. For example, if adding a new row to a twodimensional

array, the row must have the same number of columns as the original array:

>> A = [3 5 7];

>> B = [1 3 5];

>> C = [A; B]

C =

3 5 7

1 3 5

If you try to append to an array and the appended part does not conform dimensionally,

an error will result. Here’s an example:

>> A = [3 5 7];

>> B = [2 4];

PROGRAMMING TIP 1!

MATLAB supports preallocation of arrays by allowing the creation of an array that is filled with all zeros or ones. If you are using very large arrays in your programs, preallocation is more efficient than slowly growing

an array. If you know the size of your array ahead of time (e.g., 20,000), create a zero-filled array by using zero(20000); which is much faster than extending the size of the array one element at a time.

>> C = [A; B]

??? Error using ==> vertcat

All rows in the bracketed expression must have the same number of columns.

3.3 Deleting Array Elements

You can delete array elements by replacing them with the empty array, which we designate as []. In the following example, the second element of vector *A* is removed:

>> A = [3 5 7];

>> A(2) = []

A =

3 7

You cannot remove a single element from a multidimensional array, since the array would no longer be conformant. This results in an error, as shown in this example:

>> A = [1 3 5; 2 4 6]

A =

1 3 5

2 4 6

>> A(2,3) = []

??? Indexed empty matrix assignment is not allowed.

You can use the colon operator in deletion operations. The colon operator allows deletion of whole rows or columns. In the next example, the second row of the array *A* is removed:

>> A = [1 3 5; 2 4 6]

A =

1 3 5

2 4 6

>> A(2,:) = []

A =

1 3 5

The following example removes the first, third, and fifth columns from array *A*:

>> A = [1 2 3 4 5 6; 7 8 9 10 11 12]

A =

1 2 3 4 5 6

7 8 9 10 11 12

>> A(:, 1:2:5) = []

A =

2 4 6

8 10 12

4 ELEMENT-BY-ELEMENT ARRAY OPERATIONS

4.1 Array Addition

MATLAB performs addition or subtraction of two arrays of the same order by adding or subtracting each pair of respective elements. The result is an array of the same order. For example, given that

is calculated as follows: Here is another example, this time we are using two-dimensional arrays. Given that and the sum of *A* and *B* is

If two arrays are not of the same order, we say they are not *conformable* for addition or subtraction. For example, the following matrices *C* and *D* are not conformable for addition or subtraction because *C*’s dimensions are 1 \* 3 and *D*’s dimensions are 2 \* 3:

*A* + *B* = C

-4 9 19

 2 2 10

 9 15 10

S

*B* = C

-5 6 14

0 -2 4

2 8 3

S

*A* = C

1 3 5

2 4 6

7 7 7

S

[*A*112 + *B*112 *A*122 + *B*122 *A*132 + *B*132] = [11 15 19].

*A* + *B*

*A* = [1 3 5] and *B* = [10 12 14]

Let array

>> A = [ 1 0 1 0

0 2 0 2

3 1 3 1 ]

Write commands that will perform each of the following operations on array *A*:

1. Return the second column of *A*.

2. Return the first and third rows of *A*.

3. Delete the first and second columns of *A*.

4. Append the column vector [7; 8; 9] to *A*.

Re-create array *A* again before each problem. Check your answers by using MATLAB.

PRACTICE 1!

Arrays and Matrix Opera ti o n s 263

Element-by-Element Array Operations

>> C = [1,3,5]

C =

1 3 5

>> D = [2,4,6;3,5,7]

D =

2 4 6

3 5 7

>> C + D

??? Error using ==> +

Matrix dimensions must agree.

As you see, attempting to add them will result in an error.

The addition of arrays is commutative—that is,

The addition and subtraction of arrays is associative—that is,

4.2 Array Multiplication

MATLAB performs array multiplication by multiplying each pair of respective elements

in two arrays of the same order. The symbol for array multiplication is a period followed

by an asterisk The following example demonstrates array multiplication:

>> A = [1, 3, 5; 2, 4, 6]

A =

1 3 5

2 4 6

>> B = [2, 3, 4; -1, -2, -3]

B =

2 3 4

-1 -2 -3

>> A .\* B

ans =

2 9 20

-2 -8 -18

To be conformable for array multiplication, the two arrays must be of the same order,

unless one array is a scalar, in which case, each element of the other array is multiplied

by the scalar—for example,

>> A = [5]

A =

5

>> B = [2, 4, 6]

B =

2 4 6

>> A .\* B

ans =

10 20 30

1.\*2.

*A* + 1*B* + *C*2 = 1*A* + *B*2 + *C*.

*A* + *B* = *B* + *A*.

264 Arrays and Matrix Operations

Arrays and Matrix Operations

In this case (where at least one operand is a scalar), the period before the multiplication

symbol is not required. The alone will produce the same result. (See Section 5.2, titled

“Matrix Multiplication” for details.) Here’s an example:

>> A\*B

ans =

10 20 30

4.3 Array Right Division

MATLAB performs array right division of arrays *A* and *B* by dividing each element in

array *A* by the respective element in array *B*. The symbol for array right division is a period

followed by a forward slash (./). To be conformable for array right division, the two

arrays must be of the same order, unless one array is a scalar. The following example

demonstrates array right division:

>> A = [2, 4, 6]

A =

2 4 6

>> B = [2, 2, 2]

B =

2 2 2

>> A./B

ans =

1 2 3

4.4 Array Left Division

MATLAB performs array left division of arrays *A* and *B* by dividing each element in

array *B* by the respective element in array *A*. The symbol for array left division is a period

followed by a back slash (.\). To be conformable for array left division, the two arrays

must be of the same order, unless one array is a scalar. The following example demonstrates

array left division, using the arrays *A* and *B* from the previous example:

>> A.\B

ans =

1.0000 0.5000 0.3333

4.5 Array Exponentiation

MATLAB performs array exponentiation of arrays *A* and *B* by raising each element in

array *A* to the power of its respective element in array *B*. The symbol for array exponentiation

is a period followed by the caret symbol To be conformable for array exponentiation,

the two arrays must be of the same order, unless one array is a scalar. The

following example demonstrates array exponentiation:

>> A = [2, 3, 4]

A =

2 3 4

>> B = [3, 2, 0.5]

B =

3.0000 2.0000 0.5000

1.؟2.

“\*”

Arrays and Matrix Opera ti o n s 265

Binary Matrix Operations

>> A.^B

ans =

8 9 2

5 BINARY MATRIX OPERATIONS

A binary operation is a mathematical computation performed by using two matrices as

inputs. Binary matrix operations are not as straightforward to compute as element-byelement

operations. Binary matrix operations have many applications, such as the solution

of systems of linear equations.

5.1 Vector Multiplication

We will first describe vector multiplication mathematically and then show you how to

perform the operation by using MATLAB. Two vectors and are multiplied by computing

their dot product. The *dot product*, sometimes called the *inner product*, is calculated

by adding the products of each pair of respective elements in vectors and To

be conformable for vector multiplication must be a row vector and must be a column

vector. In addition, the vectors must contain the same number of elements, unless

one is a scalar. If row vector

and column vector

the dot product

a # b = *a*1*b*1 + *a*2*b*2 + ء + *anbn*.

b = D

*b*1

*b*2

ء

*bn*

T

a = [*a*1 *a*2 ء *an*],

a b

a b.

a b

Given

A = [2 0 2; 1 0 1]

and

B = [4 4 4; 9 9 9]

calculate the following by hand:

1. A + B

2. A \* 3

3. A .\* 3

4. A .^ 3

5. (A + B) ./ B

6. (A + B) ./ A

Use MATLAB to check your answers.

PRACTICE 2!

266 Arrays and Matrix Operations

Arrays and Matrix Operations

The MATLAB symbol for vector multiplication is the asterisk —for example,

A = [1, 5, -6]

A =

1 5 -6

B = [-2; -4; 0]

B =

-2

-4

0

C = A \* B

C = -22

The result was calculated as follows:

A\*B = (1\*-2) + (5\*-4) + (-6\*0) = -22

Note how this differs from array multiplication, which would fail, since *A* and *B* are not

conformable for array multiplication.

If you attempt to use nonconformable vectors, MATLAB returns an error. Here’s

an example:

>> A = [1, 2, 3]

A =

1 2 3

>> B = [2, 3, 4]

B =

2 3 4

>> A \* B

??? Error using ==> \*

Inner matrix dimensions must agree.

5.2 Matrix Multiplication

MATLAB performs the multiplication of matrix *A* by a scalar by multiplying each element

of *A* by the scalar. Any array or matrix can be multiplied by a scalar. The following

is an example:

A = [1, 3; -2, 0]

A =

1 3

-2 0

B = A \* 5

B =

5 15

-10 0

MATLAB performs multiplication of nonscalar *A* and *B* by computing the dot products

of each row in *A* with each column in *B*. Each result becomes a row in the resulting matrix.

We will try to make this clearer by walking through an example:

>> A = [1 3 5; 2 4 6]

A =

1 3 5

2 4 6

1\*2

Arrays and Matrix Opera ti o n s 267

Binary Matrix Operations

>> B = [-2 4; 3 8; 12 -2]

B =

-2 4

3 8

12 -2

Note that the number of rows in equals the number of columns in

To be conformable for matrix multiplication, the number of rows in *A* must

equal the number of columns in *B*. The result will be an matrix. In the example,

the result will be a matrix.

The first step is to compute the dot product of row one of *A* and column one of *B*:

(1 \* -2) + (3 \* 3) + (5 \* 12)= 67

Place the result in cell (1, 1) of the result matrix. Next, compute the dot product of row

one of *A* and column two of *B*:

(1 \* 4) + (3 \* 8) + (5 \* -2)= 18

Place the result in cell (1, 2) of the result matrix. Next, compute the dot product of row

two of *A* and column one of *B*:

(2 \* -2) + (4 \* 3) + (6 \* 12)= 80

Place the result in cell (2, 1) of the result matrix. Finally, compute the dot product of row

two of *A* and column two of *B*:

(2 \* 4) + (4 \* 8) + (6 \* -2)= 28

Place the result in cell (2, 2) of the result matrix. The resulting product is

>> A\*B

ans =

67 18

80 28

For most cases of *A* and *B*, matrix multiplication is not commutative; that is,

5.3 Matrix Division

The operations for left and right matrix division are not straightforward. We will not walk

through the underlying algorithm for their computation in this text. However, we will

show you an application of the left matrix division operator.

A common and useful application of matrices is the representation of systems of

linear equations. The linear system

can be represented compactly as the matrix product

C

3 2 1

1 2 3

-5 -10 -5

S C

*x*1

*x*2

*x*3

S = C

5

13

0

S

*AX* = *B*:

-5*x*1 - 10*x*2 - 5*x*3 = 0

*x*1 + 2*x*2 + 3*x*3 = 13

3*x*1 + 2*x*2 + *x*3 = 5

*AB* Z *BA*.

2 \* 2

*mA* \* *nB*

*B*1*nB* = 22.

*A*1*mA* = 22

268 Arrays and Matrix Operations

Arrays and Matrix Operations

MATLAB uses a complex algorithm to compute the solution to a linear system of the

form The operation is denoted by the matrix left division operator (the backslash)

\ *B*.

The solution to the preceding linear system can be determined as follows:

>> A = [3 2 1; 1 2 3; -5 -10 -5];

>> B = [5; 13; 0];

>> X = A\B

X =

2.5000

-4.5000

6.5000

Verify that MATLAB produced a correct answer by substituting the results into the original

three equations. You will learn more about solutions to linear systems when you take

a course in linear algebra.

*X* = *A*

*AX* = *B*.

6 UNARY MATRIX OPERATIONS

Unary matrix operations are mathematical computations that are performed by using a

single matrix as an input.

6.1 Transpose

We call the matrix that is created by exchanging the rows and columns of matrix *A* the

*transpose* of *A*. For example, given

*A* = C

1 2 3

4 5 6

7 8 9

S

Given vectors

A = [ 2 –3 4 0]

B = [ 4; -12; 4; -12]

C = [ 2 12 0 0]

compute the following operations by hand and then check your answers by using

MATLAB:

1. A \* B

2. A \* C

3. B \* C

4. C \* B

Given the matrices

A = [ 12 4; 3 –5]

B = [ 2 12; 0 0]

compute the following operations by hand and then check your answers by using

MATLAB:

5. A \* B

6. B \* A

PRACTICE 3!

Arrays and Matrix Opera ti o n s 269

Unary Matrix Operations

the transpose of *A*, denoted in mathematics as is

The MATLAB prime operator returns the transpose of its argument—for example,

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

A =

1 2 3

4 5 6

7 8 9

>> A'

ans =

1 4 7

2 5 8

3 6 9

6.2 Determinant

The *determinant* of a matrix is a transformation of a square matrix that results in a scalar.

We denote the determinant of a matrix *A* mathematically as or det *A*. In this text, we

will use the second notation, since it resembles the MATLAB function for computing a

determinant.

If a matrix has a single entry, then the determinant of the matrix is the value of the

entry. For example, if the determinant of We write this as

If a square matrix *A* has order 2, then the determinant of *A* is calculated as follows:

MATLAB has a function that computes the determinant named *det*. The syntax for the

*det* function is

where A must be a square matrix—for example,

A =

2 3

6 4

>> det(A)

ans =

-10

First, we will show you how to calculate mathematically the determinant of a matrix

with order Then we will show you how to use MATLAB to perform the same

computation.

The strategy for calculating the determinant of a matrix with order involves

subdividing the matrix into smaller sections called *minors* and *cofactors*. If row *i* and column

*j* of a square matrix *A* are deleted, the determinant of the resulting matrix is called

*n* 7 2

*n* 7 2.

det 1A2

detc

*a*11 *a*12

*a*21 *a*22

d = *a*11 # *a*22 - *a*21 # *a*12

det *A* = 3.

*A* = [3], *A* = 3.

ƒ*A*ƒ

1؟2

*AT* = C

1 4 7

2 5 8

3 6 9

S

*AT*,

270 Arrays and Matrix Operations

Arrays and Matrix Operations

the minor of We denote the minor as For example, given

then the minor of (deleting row 1 and column 2) is

The cofactor of is denoted as and is calculated as follows:

In our example, the cofactor of is

The general form for the calculation of a determinant is

where *i* is any row in square matrix *A* of order *n*. The answer is the same no matter which

row is chosen. A similar formula works by choosing any column in *A*. Let us follow the

example and expand *A* around row 2:

We find that by using MATLAB to compute the determinant of *A* results in

A =

1 2 3

4 5 6

7 8 9

>> det(A)

ans =

0

As you can see, the determinant of a high order matrix is tedious to calculate by hand.

Moreover, because the calculation of a higher order determinant is computationally intensive

and involves a series of recursive steps, the rounding error can be significant.

6.3 Inverse

The *inverse* of a square matrix *A*, if it exists, is defined to be a square matrix such that

where *I* is the identity matrix of the same order as *A*. The matrix inverse operation is denoted

mathematically by using a negative one exponent, *A*-1.

*AA*-1 = *I*,

= 0.

= 1-4 # -62 + 15 # -122 + 1-6 # -62

+ 6 # 1-122+3 # detc

1 2

7 8

d

det *A* = 4 # 1-122+1 # detc

2 3

8 9

d + 5 # 1-122+2 # detc

1 3

7 9

d

det *A* = *ai*1 *Ai*1 + *ai*2 *Ai*2 + ء + *ain Ain*

*A*12 = 1-121+2 114 # 92 - 16 # 722 = 6.

*a*12

*Aij* = 1-12*i*+*j Mij*.

*aij Aij*

*M*12 = detc

4 6

7 9

d .

*a*12

*A* = C

1 2 3

4 5 6

7 8 9

S

*aij*. *Mij*.

Arrays and Matrix Opera ti o n s 271

Unary Matrix Operations

There is a method for determining if and when the inverse of a matrix exists. It depends

on understanding the concept of matrix *singularity*. A square matrix is singular if

and only if its determinant is equal to zero. Otherwise, a matrix is nonsingular. Furthermore,

a square matrix has an inverse if and only if it is nonsingular. So, a square matrix *A*

has an inverse if and only if

However, on a computer, zero is not always zero. Computer representations of real

numbers are usually approximations. Thus, calculations can result in highly accurate, but

approximate results. If the determinant of a matrix is close to zero, MATLAB will give a

warning that the inverse of *A* may not be correct.

The syntax for MATLAB’s inverse function *inv* is

inv(*square-matrix*)

Recall from the previous example that the determinant of the matrix

A =

1 2 3

4 5 6

7 8 9

is singular (i.e., ) and should not have an inverse. MATLAB returns a warning

noting this:

>> inv(A)

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 1.541976e-

018.

ans =

1.0e+016 \*

-0.4504 0.9007 -0.4504

0.9007 -1.8014 0.9007

-0.4504 0.9007 -0.4504

6.4 Matrix Exponentiation

MATLAB computes the positive integer power of a square matrix *A* by multiplying *A*

times itself the requisite number of times. The multiplication operation that is performed

is matrix multiplication, not element-by-element multiplication—for example,

>> A = [1, 2; 3, 4]

A =

1 2

3 4

>> A^2

ans =

7 10

15 22

>> A^3

ans =

37 54

81 118

The negative integer power of a square matrix *A* is computed by performing matrix multiplication

of the inverse of *A* the requisite number of times. For example, to compute

the second negative root of *A*, we type

*det*1*A*2 = 0

*det*1*A*2 Z 0.

272 Arrays and Matrix Operations

Arrays and Matrix Operations

>> A^-2

ans =

5.5000 -2.5000

-3.7500 1.7500

This only works if the matrix is nonsingular. MATLAB issues a warning if the computed

determinant of A is equal or very close to zero. Here’s an example:

>> A = [1,1; 0,0]

A =

1 1

0 0

>> det(A)

ans =

0

>> A^-2

Warning: Matrix is singular to working precision.

ans =

Inf Inf

Inf Inf

Given the square matrices

A = [ 2 0; 1 -5]

B = [ 3 –2 0; 4 1 5; 0 –3 4]

compute the following operations by hand and then check your answers by using

MATLAB:

1. A'

2. det(A)

3. B'

4. det(B)

Compute the following with MATLAB:

5. A^2

6. inv(A)

7. inv(B)

8. A^-2

PRACTICE 4!

7 MULTIDIMENSIONAL ARRAYS

We have previously used examples of two-dimensional arrays. Many of MATLAB’s array

operations can be extended to more than two dimensions.

The following command creates a three-dimensional array of order

Since MATLAB cannot display the whole array at once, it displays the array a page at a

time. There are two pages in the following example, as the third dimension takes two levels:

>> A = ones(2,3,2)

A(:,:,1) =

1 1 1

1 1 1

2 \* 3 \* 2.

Arrays and Matrix Opera ti o n s 273

Useful Array Functions

A(:,:,2) =

1 1 1

1 1 1

8 USEFUL ARRAY FUNCTIONS

MATLAB contains scores of useful functions for manipulating and extracting information

from arrays. This section presents a few of the most commonly used array

functions.

***ndims***

The *ndims* function returns the number of dimensions of its argument—for example,

>> A = ones(2,3,2);

>> ndims(A)

ans =

3

***size***

The *size* function returns the length of each dimension, or the order of the array. The result

is a vector that contains the size of dimension 1, dimension 2, dimension 3, etc.

Here’s an example,

>> A = zeros(2,3,2,4);

>> size(A)

ans =

2 3 2 4

You can also use the size function to return the size of each dimension to a separate

variable—for example,

>> [m, n, s, t] = size(A)

m =

2

n =

3

s =

2

t =

4

***diag***

The *diag* function returns the elements of the main diagonal. For a matrix, *diag* returns

the elements with equal row and column indices (i.e., elements (1,1), (2,2),

(3,3), etc.):

>> A = [1 3 5; 2 4 6; 0 2 4]

A =

1 3 5

2 4 6

0 2 4

274 Arrays and Matrix Operations

Arrays and Matrix Operations

>> diag(A)

ans =

1

4

4

The main diagonal is also called the *zero diagonal*. A second argument may be passed to

*diag* that specifies the *n*th diagonal above or below zero. If the second argument is positive,

the *n*th diagonal above the zero diagonal is returned, as in this example:

>> diag(A,1)

ans =

3

6

If the second argument is negative, the *n*th diagonal below the zero diagonal is returned.

Here’s an example:

>> diag(A,-1)

ans =

2

2

***length***

The *length* function returns the length of the largest dimension of an array. For a onedimensional

array (vector), this equals the number of elements in the vector. The

length of *A* in the following example is three, which is the size of the largest dimension:

>> A = [1 3; 2 4; 0 2];

>> length(A)

ans =

3

***reshape***

The *reshape* function reshapes an array. It has the syntax

reshape(*A*, *m*, *n*, *p*, ...)

where *A* is the array to be reshaped, and *m*, *n*, *p*, are the new dimensions. The number

of elements in the old array must equal the number of elements in the new array.

Consider the array

>> A = ones(2,6,2);

Since the number of elements in we should be able to reshape *A*

into any order in which the product of the dimensions equals 24—for example,

>> reshape(A,2,12)

ans =

1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1

An attempt to reshape an array into a nonconforming array results in an error. Here’s an

example:

*A* = 2 \* 6 \* 2 = 24,

ء

Arrays and Matrix Opera ti o n s 275

Useful Array Functions

>> reshape(A,3,5)

??? Error using ==> reshape

To RESHAPE the number of elements must not change.

We shall consider another example. The following transformation makes sense, since

you know that MATLAB stores arrays in column-major order:

>> A = [ 1 2 3; 4 5 6; 7 8 9; 10 11 12]

A =

1 2 3

4 5 6

7 8 9

10 11 12

>> reshape(A, 2, 6)

ans =

1 7 2 8 3 9

4 10 5 11 6 12

***sort***

The *sort* function sorts arrays. When used on a vector, the sort is in ascending order:

>> A = [4 2 3 9 1 2];

>> sort(A)

ans =

1 2 2 3 4 9

When used on a two-dimensional array, MATLAB performs the sort on each column:

>> A = [5 0 4; 2 2 1]

A =

5 0 4

2 2 1

>> sort(A)

ans =

2 0 1

5 2 4

For more than two dimensions, MATLAB performs the sort on the first dimension with

the size greater than one. We call a dimension of size one a *singleton dimension*. Another

way of stating this rule is that the sort is performed on the first nonsingleton dimension.

You can specify the dimension on which to sort as a second argument. For example,

if we want to sort the two-dimensional array *A* across rows instead of down columns,

we could use the following command:

>> A = [5 0 4; 2 2 1]

A =

5 0 4

2 2 1

>> sort(A,2)

ans =

0 4 5

1 2 2

You can perform descending sorts by using the colon operator.

276 Arrays and Matrix Operations

Arrays and Matrix Operations

***max, min, mean, median***

The *max*, *min*, *mean*, and *median* functions each work in a similar fashion to the *sort*

function. Given a vector argument, the functions return the maximum, minimum, mean,

or median value, respectively. If given a two-dimensional array, each function returns a

vector that contains the result of the operation on each column.

Because these functions each work in a similar fashion, we will demonstrate their

use with the *min* function. First, we will use a vector as an example:

>> A = [ 3 2 -6 1 10];

>> min(A)

ans =

-6

Next, we will show an example that uses a two-dimensional array:

>> A = [ 2 1 3; 4 2 2; 5 0 -2]

A =

2 1 3

4 2 2

5 0 -2

>> min(A)

ans =

2 0 -2

Note that *min* returns the minimum for each column.

Each of the five columns in matrix A represents the four exam grades for a student in

a MATLAB programming class:

A = [ 89 97 55 72 95

100 92 63 85 91

82 96 71 91 82

90 98 48 83 70 ]

1. Give a command that sorts each student’s grades and returns a matrix with the

sorted grades.

2. Give a command that computes the mean of each student’s grades and returns a

vector with the results.

3. Give a command that computes the median of each student’s grades and returns

a vector with the results.

4. Give a single command that returns the overall mean grade for all five students

in the course.

Now, change your view of matrix *A*. Assume that each of the four rows in matrix *A*

represents the five exam grades of a student. *Note*: Each row represents a student.

5. Give a command that sorts each student’s grades and returns a matrix with the

sorted grades.

6. Give a command that computes the mean of each student’s grades and returns a

vector with the results.

7. Give a command that computes the median of each student’s grades and returns

a vector with the results.

8. Give a single command that returns the overall mean grade for all five students

in the course.

PRACTICE 5!

Arrays and Matrix Opera ti o n s 277

Key Terms

APPLICATION! COMMUNICATION ROUTES

The calculation of the number of communication paths

is important in a variety of fields, for example, the control

of network router traffic. Scientists use the same

theory to model behavior in fields such as human communication,

political influence, and the flow of money

through organizations.

A common example, used to demonstrate principles

of communication routes, is the number of roads

connecting cities. In this diagram we depict four cities

along with the roads connecting them:

Table 2 shows the number of direct routes between

each pair of cities. A direct route does not go through

any intermediate city. For example, there are two direct

routes between City 1 and City 4. The table expresses

this information redundantly. You can see the routes between

City 1 and City 4 by looking at either (row 1, column

4) or (row 4, column 1). We have presented the

data in such a manner, so that it can be stored in a

square matrix.

The square matrix *A* summarizes the connectivity

between the cities. For example, indicates

that there are two direct routes from City 1 to

City 4:

*A*11,42 = 2

A =

0 1 1 2

1 0 0 0

1 0 0 1

2 0 1 0

Note that *A* is symmetric. This means that the cells

above the main diagonal are a mirror image of the cells

below the diagonal when reflected along the diagonal.

Symmetry is also defined as for any

*m* and *n*.

It is known that the matrix *A*^2 represents the

number of ways to travel between any two cities by

passing through only one intermediate city.

>> B = A^2

B =

6 0 2 1

0 1 1 2

2 1 2 2

1 2 2 5

Matrix B summarizes the number routes between pairs

of cities if the route contains one intermediate city: Note

the six ways to travel from City 1 back to City 1 by passing

through exactly one other city: *a*–*a*, *c*–*c*, *c*–*d*, *d*–*c*, *d*–*d*,

and *e*–*e*. The two ways to travel from City 2 to City 4 by

passing through exactly one other city are *e*–*c* and *e*–*d*.

We count traveling in one direction differently than

traveling the same route in the opposite direction. Thus,

there are two routes from City 1 to City 4, *c*–*d* and *d*–*c*.

*A*1*n*, *m*2 = *A*1*m*, *n*2

**TABLE 2.** Number of routes among four cities.

**City 1 City 2 City 3 City 4**

CITY 1 0 1 1 2

CITY 2 1 0 0 0

CITY 3 1 0 0 1

CITY 4 2 0 1 0

1 2

3 4

*a c*

*b*

*e*

*d*

KEY TERMS cofactor determinant main diagonal

colon operator dot product matrix

column-major order functional form minor

column vector identity matrix ones matrix

command form inner product order

compact format inverse row-major order

conformable loose format row vector

278 Arrays and Matrix Operations

Arrays and Matrix Operations

short format square matrix zero matrix

singleton dimension transpose

singularity zero diagonal

NEW MATLAB FUNCTIONS, COMMANDS, AND RESERVED WORDS

*clock*—returns current date and time

*det*—returns the determinant of a square matrix

*diag*—returns the diagonal of a matrix

*etime*—returns time elapsed between 2 times

*eye*—returns identity matrix

*format*—formats numeric output

*get*—returns the named properties of an object

*inv*—returns the inverse of a square matrix

*length*—returns the number of elements in a vector

*max*—returns the maximum element(s) along the first non-singleton dimension

*mean*—returns the mean element(s) along the first non-singleton dimension

*median*—returns the median element(s) along the first non-singleton dimension

*min*—returns the minimum element(s) along the first non-singleton dimension

*ndims*—returns the number of dimensions of an array

*ones*—returns an array of ones

*rand*—returns uniformly distributes pseudo-random numbers in [0,1]

*reshape*—reshapes an array

*size*—returns the order (size) of an array

*sort*—sorts an array in ascending order

*zeros*—creates an array of zeros

SOLUTIONS TO PRACTICE PROBLEMS

1. 1. A(:,2)

2. A(1:2:3,:)

3. A(:,1:2) = []

4. A = [A [7; 8; 9]]

2. 1. A+B = [6 4 6; 10 9 10]

2. A\*3 = [6 0 6; 3 0 3]

3. A.\*3 = [6 0 6; 3 0 3]

4. A.^3 = [8 0 8; 1 0 1]

5. (A + B)./B = [1.500 1.000 1.500;

1.1111 1.0000 1.1111]

6. (A + B)./A = Warning: Divide by zero.

[ 3 Inf 3

10 Inf 10]

3. 1. A\*B = 60

2. A\*C = ??? Error using ==> \*

Inner matrix dimensions must agree.

3. B\*C = [ 8 48 0 0

-24 -144 0 0

8 48 0 0

-24 -144 0 0]

4. C\*B = -136

Arrays and Matrix Opera ti o n s 279

Problems

5. A\*B = [ 24 144; 6 36 ]

6. B\*A = [ 60 -52; 0 0 ]

4. 1. A' = [ 2 1; 0 –5 ]

2. det(A) = -10

3. B' = [ 3 4 0

-2 1 -3

0 5 4 ]

4. det(B) = 89

5. A^2 = [ 4 0; -3 25 ]

6. inv(A) = [ 0.5000 0; 0.1000 -0.2000 ]

7. inv(B) = [ 0.2135 0.0899 -0.1124

-0.1798 0.1348 -0.1685

-0.1348 0.1011 0.1236]

8. A^-2 = [ 0.2500 0; 0.0300 0.0400 ]

5. 1. sort(A)

2. mean(A)

3. median(A)

4. mean(mean(A))

5. sort(A,2)

6. mean(A,2)

7. median(A,2)

8. mean(mean(A,2))

**Problems**

**Section 1.**

What is the order and main diagonal of the following matrices?

1. [3, 4; 5, 6; 7, 8]

2. [2 3 4 5; 6 7 8 9]

3. [2 1 0; 2 1; 4 0 0; 3 2 1]

Verify your answers by using appropriate MATLAB functions.

**Section 2.**

4. Create a vector *A* that contains the following fractions:

>> A

A =

1/2 2/3 3/4 4/5 5/6

What command changes your format so the vector displays rational fractions instead

of decimals?

5. What command creates a matrix that contains all zeros?

**Section 3.**

6. The loads in kilograms on the center points of five beams are

400.3

521.1

4 \* 5

-3

280 Arrays and Matrix Operations

Arrays and Matrix Operations

212.1

349.5

322.2

Create a row vector named “Loads” that contains the five values. What is a

single command that replaces the second and fourth values of “Loads” with

zero? What is a single command that deletes the third and fifth elements of

“Loads”?

7. Re-create the original row vector “Loads” from the previous problem. The

lengths in meters of the five beams are, respectively,

14.3

6.2

22.6

2.4

10.2

Create a row vector named “Lengths” that contains the five beam lengths in

meters. In a single command, create a matrix named “Beams” by concatenating

“Loads” and “Lengths”. “Beams” should have two rows with the load values on

the first row and the respective lengths on the second row. Your answer should

look like the following:

>> Beams =

400.3000 521.1000 212.1000 349.5000 322.2000

14.3000 6.2000 22.6000 2.4000 10.2000

**Section 4.**

8. Assume that the loads for the five beams in Problem 6 are distributed evenly

across the length of each beam. Using array arithmetic, and the original vectors

“Loads” and “Lengths”, create a vector that represents the average load in kg/m

for each beam.

9. The command *rand*(1,*n*) produces a row vector of *n* uniformly distributed,

pseudorandom numbers between 0.0 and 1.0. Use array arithmetic and the

*rand* function to create 100 uniformly distributed pseudorandom numbers between

8.0 and 10.0.

**Section 5.**

10. Express the following linear system in matrix form as matrices *A* and *B*:

11. Use the MATLAB left matrix division operator to find the solution of the linear

system in the previous problem.

**Section 6.**

12. The transpose of the transpose of a matrix equals the original matrix. This can

be stated as Using MATLAB, demonstrate that the theorem is true

for the following matrix:

A = [1 2 4 6; 4 3 2 1].

1*AT*2*T* = *A*.

-5*x*1 + 10*x*2 = 0

3*x*1 + 2*x*2 = 4

Arrays and Matrix Opera ti o n s 281

Problems

13. Experiment with the transpose operator on a few example matrices. What conclusion

do you reach about the main diagonal of a matrix and the main diagonal

of its transpose?

14. Given a matrix representation of a system of linear equations if the

determinant of *A* equals zero, the system does not have a unique solution.

Two possibilities are that the system has no solutions and that the system has

an infinite number of solutions. Determine if the following system has a

unique solution:

15. The inverse of an array *A* multiplied by itself should equal the identity matrix of

the same order as *A*. Show how you would test this assumption. Use matrix

multiplication and the *inv*, *eye*, and *size* functions.

16. The following matrix represents the numbers of direct paths between four network

routers:

R1 R2 R3 R4

R1 0 2 1 3

R2 2 0 0 2

R3 1 0 0 2

R4 3 2 2 0

How many paths are there from router two to router four if each path passes

through exactly one other router?

**Section 7.**

17. Create a array of random numbers. Replace the cell contents in the

third page of the array with zeros.

18. Create a three-dimensional array of order Fill page one with 1’s,

page two with 2’s, and page 3 with 3’s. Can you solve the problem in a single

command?

**Section 8.**

19. Create the following array:

Reshape *A* into a two-column array. What is the bottom number in each column?

**Challenge Problem**

20. Reread Programming Tip 1. Test the assertion in the tip by writing a program

that creates a row vector of ones in a single command. Write another

program that creates a row vector and then builds a vector

of ones a single cell at a time using a loop.

Time both programs and compare the efficiency of the two methods. *Hints*:

The function *clock* returns a six-element vector containing the current date and

time. The meaning of each element in the vector is [year, month, day, hour,

minute, seconds].

1 \* 1 1 \* 20000

1 \* 20000

*A* = [1:10; 11:20; 21:30].

6 \* 2 \* 3.

2 \* 4 \* 3

*x*1 +

3

2 *x*2 + 2*x*3 = 0

-*x*1 + 3*x*2 - *x*3 = 12

2*x*1 + 3*x*2 + 4*x*3 = 10

*AX* = *B*,

282 Arrays and Matrix Operations

Arrays and Matrix Operations

The elapsed time function, *etime*(*t2*, *t1*), returns the elapsed time in seconds

between time *t2* and time *t1*. The following code segment computes the time

taken to execute the code between *t1* and *t2*:

t1 = clock

...

...

t2 = clock

ElapsedTime = etime(t2,t1)

Arrays and Matrix Opera ti o n s 283