Chapter 2
Creating Arrays

The array is a fundamental form that MATLAB uses to store and manipulate data. An array is a list of numbers arranged in rows and/or columns. The simplest array (one-dimensional) is a row or a column of numbers. A more complex array (two-dimensional) is a collection of numbers arranged in rows and columns. One use of arrays is to store information and data, as in a table. In science and engineering, one-dimensional arrays frequently represent vectors, and two-dimensional arrays often represent matrices. This chapter shows how to create and address arrays, and Chapter 3 shows how to use arrays in mathematical operations. In addition to arrays made of numbers, arrays in MATLAB can also be a list of characters, which are called strings. Strings are discussed in Section 2.10.

2.1 CREATING A ONE-DIMENSIONAL ARRAY (VECTOR)

A one-dimensional array is a list of numbers arranged in a row or a column. One example is the representation of the position of a point in space in a three-dimensional Cartesian coordinate system. As shown in Figure 2-1, the position of point *A* is defined by a list of the three numbers 2, 4, and 5, which are the coordinates of the point.

The position of point *A* can be expressed in terms of a position vector:

$$\mathbf{r}_A = 2\mathbf{i} + 4\mathbf{j} + 5\mathbf{k}$$

where **i**, **j**, and **k** are unit vectors in the direction of the *x*, *y*, and *z* axes, respectively. The numbers 2, 4, and 5 can be used to define a row or a column vector.

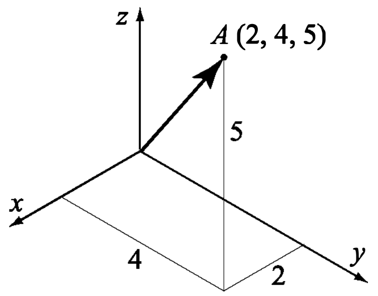

Figure 2-1: Position of a point.

Any list of numbers can be set up as a vector. For example, Table 2-1 contains population growth data that can be used to create two lists of numbers---one of the years and the other of the population values. Each list can be entered as elements in a vector with the numbers placed in a row or in a column.

Table 2-1: Population data

Year	1984	1986	1988	1990	1992	1994	1996
Population (millions)	127	130	136	145	158	178	211

In MATLAB, a vector is created by assigning the elements of the vector to a variable. This can be done in several ways depending on the source of the information that is used for the elements of the vector. When a vector contains specific numbers that are known (like the coordinates of point *A*), the value of each element is entered directly. Each element can also be a mathematical expression that can include predefined variables, numbers, and functions. Often, the elements of a row vector are a series of numbers with constant spacing. In such cases the vector can be created with MATLAB commands. A vector can also be created as the result of mathematical operations as explained in Chapter 3.

Creating a vector from a known list of numbers:

The vector is created by typing the elements (numbers) inside square brackets [ ].

```
variable_name = [ type vector elements ]
```

Row vector: To create a row vector type the elements with a space or a comma between the elements inside the square brackets.

Column vector: To create a column vector type the left square bracket [and then enter the elements with a semicolon between them, or press the **Enter** key after each element. Type the right square bracket] after the last element.

Tutorial 2-1 shows how the data from Table 2-1 and the coordinates of point *A* are used to create row and column vectors.

Tutorial 2-1: Creating vectors from given data.

```
>> yr=[1984 1986 1988 1990 1992 1994 1996]
```

The list of years is assigned to a row vector named `yr`.

```
yr =
    1984      1986      1988      1990      1992      1994      1996

>> pop=[127;  130;  136;  145;  158;  178;  211]
```

The population data is assigned to a column vector named `pop`.

```
pop =
   127
   130
   136
   145
   158
```

Tutorial 2-1: Creating vectors from given data. (Continued)

```
   178
   211
>> pntAH=[2,  4,  5]
pntAH =
     2     4     5
>> pntAV=[2
4
5]
pntAV =
     2
     4
     5
>>
```

The coordinates of point *A* are assigned to a row vector called `pntAH`.

The coordinates of point *A* are assigned to a column vector called `pntAV`. (The **Enter** key is pressed after each element is typed.)

Creating a vector with constant spacing by specifying the first term, the spacing, and the last term:

In a vector with constant spacing, the difference between the elements is the same. For example, in the vector $v = 2\ \ 4\ \ 6\ \ 8\ \ 10$, the spacing between the elements is 2. A vector in which the first term is ***m***, the spacing is ***q***, and the last term is ***n*** is created by typing:

`variable_name = [m:q:n]` or `variable_name = m:q:n`

(The brackets are optional.)

Some examples are:

```
>> x=[1:2:13]
x =
     1     3     5     7     9    11    13
>> y=[1.5:0.1:2.1]
y =
   1.5000   1.6000   1.7000   1.8000   1.9000   2.0000   2.1000

>> z=[-3:7]

z =
    -3    -2    -1     0     1     2     3     4     5     6
7
>> xa=[21:-3:6]
```

First element 1, spacing 2, last element 13.

First element 1.5, spacing 0.1, last element 2.1.

First element –3, last term 7. If spacing is omitted, the default is 1.

First element 21, spacing –3, last term 6.

```
xa =
      21     18     15     12      9      6
>>
```

- If the numbers m, q, and n are such that the value of n cannot be obtained by adding q's to m, then (for positive n) the last element in the vector will be the last number that does not exceed n.
- If only two numbers (the first and the last terms) are typed (the spacing is omitted), then the default for the spacing is 1.

Creating a vector with linear (equal) spacing by specifying the first and last terms, and the number of terms:

A vector with *n* elements that are linearly (equally) spaced in which the first element is *xi* and the last element is *xf* can be created by typing the linspace command (MATLAB determines the correct spacing):

```
variable_name = linspace(xi,xf,n)
```

First element   Last element   Number of elements

When the number of elements is omitted, the default is 100. Some examples are:

```
>> va=linspace(0,8,6)                6 elements, first element 0, last element 8.
va =
      0    1.6000    3.2000    4.8000    6.4000    8.0000
>> vb=linspace(30,10,11)           11 elements, first element 30, last element 10.
vb =
    30    28    26    24    22    20    18    16    14    12    10
>> u=linspace(49.5,0.5)              First element 49.5, last element 0.5.
                                     When the number of elements is
u =                                  omitted, the default is 100.
  Columns 1 through 10
   49.5000   49.0051   48.5101   48.0152   47.5202   47.0253
46.5303   46.0354   45.5404   45.0455
............                           100 elements are displayed.
Columns 91 through 100
    4.9545    4.4596    3.9646    3.4697    2.9747    2.4798
1.9848    1.4899    0.9949    0.5000
>>
```

2.2 Creating a Two-Dimensional Array (Matrix)

A two-dimensional array, also called a matrix, has numbers in rows and columns. Matrices can be used to store information like the arrangement in a table. Matrices play an important role in linear algebra and are used in science and engineering to describe many physical quantities.

In a square matrix the number of rows and the number of columns is equal. For example, the matrix

$$\begin{array}{ccc} 7 & 4 & 9 \\ 3 & 8 & 1 \\ 6 & 5 & 3 \end{array} \quad 3 \times 3 \text{ matrix}$$

is square, with three rows and three columns. In general, the number of rows and columns can be different. For example, the matrix:

$$\begin{array}{rrrrrr} 31 & 26 & 14 & 18 & 5 & 30 \\ 3 & 51 & 20 & 11 & 43 & 65 \\ 28 & 6 & 15 & 61 & 34 & 22 \\ 14 & 58 & 6 & 36 & 93 & 7 \end{array} \quad 4 \times 6 \text{ matrix}$$

has four rows and six columns. A $m \times n$ matrix has m rows and n columns, and m by n is called the size of the matrix.

A matrix is created by assigning the elements of the matrix to a variable. This is done by typing the elements, row by row, inside square brackets [ ]. First type the left bracket [then type the first row, separating the elements with spaces or commas. To type the next row type a semicolon or press **Enter**. Type the right bracket] at the end of the last row.

```
variable_name=[1st row elements; 2nd row elements; 3rd
               row elements; ... ; last row elements]
```

The elements that are entered can be numbers or mathematical expressions that may include numbers, predefined variables, and functions. *All the rows must have the same number of elements*. If an element is zero, it has to be entered as such. MATLAB displays an error message if an attempt is made to define an incomplete matrix. Examples of matrices defined in different ways are shown in Tutorial 2-2.

Tutorial 2-2: Creating matrices.

```
>> a=[5  35  43;  4  76  81;  21  32  40]
a =
     5    35    43
     4    76    81
    21    32    40
>> b = [7  2  76  33  8
1  98  6  25  6
5  54  68  9  0]
```

A semicolon is typed before a new line is entered.

The **Enter** key is pressed before a new line is entered.

Tutorial 2-2: Creating matrices. (Continued)

```
b =
     7     2    76    33     8
     1    98     6    25     6
     5    54    68     9     0
>> cd=6; e=3; h=4;
>> Mat=[e, cd*h, cos(pi/3); h^2, sqrt(h*h/cd), 14]
Mat =
    3.0000   24.0000    0.5000
   16.0000    1.6330   14.0000
>>
```

Three variables are defined.

Elements are defined by mathematical expressions.

Rows of a matrix can also be entered as vectors using the notation for creating vectors with constant spacing, or the `linspace` command. For example:

```
>> A=[1:2:11; 0:5:25; linspace(10,60,6); 67 2 43 68 4 13]
A =
     1     3     5     7     9    11
     0     5    10    15    20    25
    10    20    30    40    50    60
    67     2    43    68     4    13
>>
```

In this example the first two rows were entered as vectors using the notation of constant spacing, the third row was entered using the `linspace` command, and in the last row the elements were entered individually.

2.2.1 *The* `zeros`*,* `ones` *and,* `eye` *Commands*

The `zeros(m,n)`, `ones(m,n)`, and `eye(n)` commands can be used to create matrices that have elements with special values. The `zeros(m,n)` and the `ones(m,n)` commands create a matrix with *m* rows and *n* columns in which all elements are the numbers 0 and 1, respectively. The `eye(n)` command creates a square matrix with *n* rows and *n* columns in which the diagonal elements are equal to 1 and the rest of the elements are 0. This matrix is called the identity matrix. Examples are:

```
>> zr=zeros(3,4)
zr =
     0     0     0     0
     0     0     0     0
     0     0     0     0
>> ne=ones(4,3)
```

```
ne =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
>> idn=eye(5)
idn =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
>>
```

Matrices can also be created as a result of mathematical operations with vectors and matrices. This topic is covered in Chapter 3.

2.3 Notes About Variables in MATLAB

- All variables in MATLAB are arrays. A scalar is an array with one element, a vector is an array with one row or one column of elements, and a matrix is an array with elements in rows and columns.
- The variable (scalar, vector, or matrix) is defined by the input when the variable is assigned. There is no need to define the size of the array (single element for a scalar, a row or a column of elements for a vector, or a two-dimensional array of elements for a matrix) before the elements are assigned.
- Once a variable exists—as a scalar, vector, or matrix—it can be changed to any other size, or type, of variable. For example, a scalar can be changed to a vector or a matrix; a vector can be changed to a scalar, a vector of different length, or a matrix; and a matrix can be changed to have a different size, or be reduced to a vector or a scalar. These changes are made by adding or deleting elements. This subject is covered in Sections 2.7 and 2.8.

2.4 The Transpose Operator

The transpose operator, when applied to a vector, switches a row (column) vector to a column (row) vector. When applied to a matrix, it switches the rows (columns) to columns (rows). The transpose operator is applied by typing a single quote `'` following the variable to be transposed. Examples are:

```
>> aa=[3  8  1]
aa =
     3     8     1
>> bb=aa'
```

Define a row vector `aa`.

Define a column vector `bb` as the transpose of vector `aa`.

```
bb =
     3
     8
     1
>> C=[2 55 14 8; 21 5 32 11; 41 64 9 1]
C =
     2    55    14     8
    21     5    32    11
    41    64     9     1
>> D=C'
D =
     2    21    41
    55     5    64
    14    32     9
     8    11     1
>>
```

Define a matrix C with 3 rows and 4 columns.

Define a matrix D as the transpose of matrix C. (D has 4 rows and 3 columns.)

2.5 *ARRAY ADDRESSING*

Elements in an array (either vector or matrix) can be addressed individually or in subgroups. This is useful when there is a need to redefine only some of the elements, when specific elements are to be used in calculations, or when a subgroup of the elements is used to define a new variable.

2.5.1 Vector

The address of an element in a vector is its position in the row (or column). For a vector named `ve`, `ve(`*k*`)` refers to the element in position *k*. The first position is 1. For example, if the vector *ve* has nine elements:

ve = 35 46 78 23 5 14 81 3 55

then

ve(4) = 23, *ve*(7) = 81, and *ve*(1) = 35.

A single vector element, *v*(*k*), can be used just as a variable. For example, it is possible to change the value of only one element of a vector by assigning a new value to a specific address. This is done by typing: *v*(*k*) = *value*. A single element can also be used as a variable in a mathematical expression. Examples are:

```
>> VCT=[35 46 78 23 5 14 81 3 55]
VCT =
    35    46    78    23     5    14    81     3    55
>> VCT(4)
```

Define a vector.

Display the fourth element.

```
ans =
     23
>> VCT(6)=273
VCT =
     35     46     78     23      5    273     81      3     55


>> VCT(2)+VCT(8)
ans =
     49
>> VCT(5)^VCT(8)+sqrt(VCT(7))
ans =
   134
>>
```

Assign a new value to the sixth element.

The whole vector is displayed.

Use the vector elements in mathematical expressions.

2.5.2 *Matrix*

The address of an element in a matrix is its position, defined by the row number and the column number where it is located. For a matrix assigned to a variable *ma*, *ma*(*k*,*p*) refers to the element in row *k* and column *p*.

For example, if the matrix is: $ma = \begin{bmatrix} 3 & 11 & 6 & 5 \\ 4 & 7 & 10 & 2 \\ 13 & 9 & 0 & 8 \end{bmatrix}$

then *ma*(1,1) = 3 and *ma*(2,3) = 10.

As with vectors, it is possible to change the value of just one element of a matrix by assigning a new value to that element. Also, single elements can be used like variables in mathematical expressions and functions. Some examples are:

```
>> MAT=[3 11 6 5; 4 7 10 2; 13 9 0 8]
MAT =
      3     11      6      5
      4      7     10      2
     13      9      0      8
>> MAT(3,1)=20
MAT =
      3     11      6      5
      4      7     10      2
     20      9      0      8
>> MAT(2,4)-MAT(1,2)
ans =
     -9
```

Create a 3×4 matrix.

Assign a new value to the (3,1) element.

Use elements in a mathematical expression.

2.6 USING A COLON : IN ADDRESSING ARRAYS

A colon can be used to address a range of elements in a vector or a matrix.

For a vector:

va(:) Refers to all the elements of the vector *va* (either a row or a column vector).

va(*m*:*n*) Refers to elements *m* through *n* of the vector *va*.

Example:

```
>> v=[4 15 8 12 34 2 50 23 11]
v =
     4    15     8    12    34     2    50    23    11
>> u=v(3:7)
u =
     8    12    34     2    50
>>
```

A vector v is created.

A vector u is created from the elements 3 through 7 of vector v.

For a matrix:

A(:,*n*) Refers to the elements in all the rows of column *n* of the matrix *A*.

A(*n*,:) Refers to the elements in all the columns of row *n* of the matrix *A*.

A(:,*m*:*n*) Refers to the elements in all the rows between columns *m* and *n* of the matrix *A*.

A(*m*:*n*,:) Refers to the elements in all the columns between rows *m* and *n* of the matrix *A*.

A(*m*:*n*,*p*:*q*) Refers to the elements in rows *m* through *n* and columns *p* through *q* of the matrix *A*.

The use of the colon symbol in addressing elements of matrices is demonstrated in Tutorial 2-3.

Tutorial 2-3: Using a colon in addressing arrays.

```
>> A=[1 3 5 7 9 11; 2 4 6 8 10 12; 3 6 9 12 15 18; 4 8 12 16
20 24; 5 10 15 20 25 30]


A =
     1     3     5     7     9    11
     2     4     6     8    10    12
     3     6     9    12    15    18
     4     8    12    16    20    24
     5    10    15    20    25    30
>> B=A(:,3)
```

Define a matrix A with 5 rows and 6 columns.

Define a column vector B from the elements in all of the rows of column 3 in matrix A.

Tutorial 2-3: Using a colon in addressing arrays. (Continued)

```
B =
     5
     6
     9
    12
    15
>> C=A(2,:)
C =
     2     4     6     8    10    12
>> E=A(2:4,:)
E =
     2     4     6     8    10    12
     3     6     9    12    15    18
     4     8    12    16    20    24
>> F=A(1:3,2:4)
F =
     3     5     7
     4     6     8
     6     9    12
>>
```

Define a row vector C from the elements in all of the columns of row 2 in matrix A.

Define a matrix E from the elements in rows 2 through 4 and all the columns in matrix A.

Create a matrix F from the elements in rows 1 through 3 and columns 2 through 4 in matrix A.

In Tutorial 2-3 new vectors and matrices are created from existing ones by using a range of elements, or a range of rows and columns (using :). It is possible, however, to select only specific elements, or specific rows and columns of existing variables to create new variables. This is done by typing the selected elements or rows or columns inside brackets, as shown below:

```
>> v=4:3:34
v =
    4    7   10   13   16   19   22   25   28   31   34
>> u=v([3, 5, 7:10])

u =
   10   16   22   25   28   31
>> A=[10:-1:4; ones(1,7); 2:2:14; zeros(1,7)]
A =
   10    9    8    7    6    5    4
    1    1    1    1    1    1    1
    2    4    6    8   10   12   14
    0    0    0    0    0    0    0
>> B = A([1,3],[1,3,5:7])
```

Create a vector v with 11 elements.

Create a vector u from the 3rd, the 5th, and the 7th through 10th elements of v.

Create a 4×7 matrix A.

Create a matrix B from the 1st and 3rd rows, and 1st, 3rd, and the 5th through 7th columns of A.

```
B =
    10     8     6     5     4
     2     6    10    12    14
```

2.7 Adding Elements to Existing Variables

A variable that exists as a vector, or a matrix, can be changed by adding elements to it (remember that a scalar is a vector with one element). A vector (a matrix with a single row or column) can be changed to have more elements, or it can be changed to be a two-dimensional matrix. Rows and/or columns can also be added to an existing matrix to obtain a matrix of different size. The addition of elements can be done by simply assigning values to the additional elements, or by appending existing variables.

Adding elements to a vector:

Elements can be added to an existing vector by assigning values to the new elements. For example, if a vector has 4 elements, the vector can be made longer by assigning values to elements 5, 6, and so on. If a vector has n elements and a new value is assigned to an element with an address of $n+2$ or larger, MATLAB assigns zeros to the elements that are between the last original element and the new element. Examples:

```
>> DF=1:4
DF =
    1    2    3    4
>> DF(5:10)=10:5:35
DF =
    1    2    3    4   10   15   20   25   30   35
>> AD=[5  7  2]
AD =
    5    7    2
>> AD(8)=4
AD =
    5   7   2   0   0   0   0   4
>> AR(5)=24
AR =
    0    0    0    0   24
>>
```

Define vector DF with 4 elements.

Adding 6 elements starting with the 5th.

Define vector AD with 3 elements.

Assign a value to the 8th element.

MATLAB assigns zeros to the 4th through 7th elements.

Assign a value to the 5th element of a new vector.

MATLAB assigns zeros to the 1st through 4th elements.

Elements can also be added to a vector by appending existing vectors. Two examples are:

```
>> RE=[3  8 1  24];
```

Define vector RE with 4 elements.

```
>> GT=4:3:16;
>> KNH=[RE  GT]
KNH =
    3    8    1   24    4    7   10   13   16
>> KNV=[RE'; GT']
KNV =
     3
     8
     1
    24
     4
     7
    10
    13
    16
```

Define vector GT with 5 elements.

Define a new vector KNH by appending RE and GT.

Create a new column vector KNV by appending RE′ and GT′.

Adding elements to a matrix:

Rows and/or columns can be added to an existing matrix by assigning values to the new rows or columns. This can be done by assigning new values, or by appending existing variables. This must be done carefully since the size of the added rows or columns must fit the existing matrix. Examples are:

```
>> E=[1 2 3 4; 5 6 7 8]
E =
     1     2     3     4
     5     6     7     8
>> E(3,:)=[10:4:22]

E =
     1     2     3     4
     5     6     7     8
    10    14    18    22
>> K=eye(3)
K =
     1     0     0
     0     1     0
     0     0     1
>> G=[E  K]
G =
     1     2     3     4     1     0     0
     5     6     7     8     0     1     0
    10    14    18    22     0     0     1
```

Define a 2×4 matrix E.

Add the vector 10 14 18 22 as the third row of E.

Define a 3×3 matrix K.

Append matrix K to matrix E. The numbers of rows in E and K must be the same.

If a matrix has a size of $m \times n$ and a new value is assigned to an element with an address beyond the size of the matrix, MATLAB increases the size of the matrix to include the new element. Zeros are assigned to the other elements that are added. Examples:

```
>> AW=[3 6 9; 8 5 11]
AW =
     3     6     9
     8     5    11
>> AW(4,5)=17
AW =
     3     6     9     0     0
     8     5    11     0     0
     0     0     0     0     0
     0     0     0     0    17
>> BG(3,4)=15
BG =
     0     0     0     0
     0     0     0     0
     0     0     0    15
>>
```

Define a 2×3 matrix.

Assign a value to the (4,5) element.

MATLAB changes the matrix size to 4×5, and assigns zeros to the new elements.

Assign a value to the (3,4) element of a new matrix.

MATLAB creates a 3×4 matrix and assigns zeros to all the elements except BG(3,4).

2.8 DELETING ELEMENTS

An element, or a range of elements, of an existing variable can be deleted by reassigning nothing to these elements. This is done by using square brackets with nothing typed in between them. By deleting elements, a vector can be made shorter and a matrix can be made smaller. Examples are:

```
>> kt=[2 8 40 65 3 55 23 15 75 80]
kt =
     2     8    40    65     3    55    23    15    75    80
>> kt(6)=[]
kt =
     2    8    40    65    3    23    15    75    80
>> kt(3:6)=[]
kt =
     2     8    15    75    80
>> mtr=[5 78 4 24 9; 4 0 36 60 12; 56 13 5 89 3]
```

Define a vector with 10 elements.

Eliminate the 6th element.

The vector now has 9 elements.

Eliminate elements 3 through 6.

The vector now has 5 elements.

Define a 3×5 matrix.

```
mtr =
     5    78     4    24     9
     4     0    36    60    12
    56    13     5    89     3
>> mtr(:,2:4)=[]
mtr =
     5     9
     4    12
    56     3
>>
```

Eliminate all the rows of columns 2 through 4.

2.9 BUILT-IN FUNCTIONS FOR HANDLING ARRAYS

MATLAB has many built-in functions for managing and handling arrays. Some of these are listed below:

Table 2-2: Built-in functions for handling arrays

Function	Description	Example

`length(A)`	Returns the number of elements in the vector `A`.	`>> A=[5 9 2 4];`
		`>> length(A)`
		`ans =`
		`     4`

`size(A)`	Returns a row vector `[m,n]`, where `m` and `n` are the size $m \times n$ of the array `A`.	`>> A=[6 1 4 0 12; 5 19 6 8 2]`
		`A =`
		`   6   1   4   0  12`
		`   5  19   6   8   2`
		`>> size(A)`
		`ans =`
		`     2     5`

`reshape(A,m,n)`	Creates a `m` by `n` matrix from the elements of matrix `A`. The elements are taken column after column. Matrix `A` must have `m` times `n` elements.	`>> A=[5 1 6; 8 0 2]`
		`A =`
		`   5   1   6`
		`   8   0   2`
		`>> B = reshape(A,3,2)`
		`B =`
		`   5   0`
		`   8   6`
		`   1   2`

Table 2-2: Built-in functions for handling arrays (Continued)

Function	Description	Example

`diag(v)`	When `v` is a vector, creates a square matrix with the elements of `v` in the diagonal.	`>> v=[7 4 2];`
		`>> A=diag(v)`
		`A =`
		`    7    0    0`
		`    0    4    0`
		`    0    0    2`

`diag(A)`	When `A` is a matrix, creates a vector from the diagonal elements of `A`.	`>> A=[1 2 3; 4 5 6; 7 8 9]`
		`A =`
		`   1   2   3`
		`   4   5   6`
		`   7   8   9`
		`>> vec=diag(A)`
		`vec =`
		`     1`
		`     5`
		`     9`

Additional built-in functions for manipulation of arrays are described in the Help Window. In this window, select "MATLAB," then in the Contents "Functions," and then "By Category."

Sample Problem 2-1: Create a matrix

Using the ones and zeros commands, create a 4×5 matrix in which the first two rows are 0s and the next two rows are 1s.

Solution

```
>> A(1:2,:)=zeros(2,5)
A =
     0     0     0     0     0
     0     0     0     0     0
>> A(3:4,:)=ones(2,5)
A =
     0     0     0     0     0
     0     0     0     0     0
     1     1     1     1     1
     1     1     1     1     1
```

First, create a 2×5 matrix with 0s.

Add rows 3 and 4 with 1s.

A different solution to the problem is:

```
>> A=[zeros(2,5);ones(2,5)]
A =
         0         0         0         0         0
         0         0         0         0         0
         1         1         1         1         1
         1         1         1         1         1
```

Create a 4×5 matrix from two 2×5 matrices.

Sample Problem 2-2: Create a matrix

Create a 6×6 matrix in which the middle two rows and the middle two columns are 1s and the rest of the entries are 0s.

Solution

```
>> AR=zeros(6,6)
AR =
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
>> AR(3:4,:)=ones(2,6)
AR =
     0     0     0     0     0     0
     0     0     0     0     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     0     0     0     0     0     0
     0     0     0     0     0     0
>> AR(:,3:4)=ones(6,2)
AR =
     0     0     1     1     0     0
     0     0     1     1     0     0
     1     1     1     1     1     1
     1     1     1     1     1     1
     0     0     1     1     0     0
     0     0     1     1     0     0
```

First, create a 6×6 matrix with 0s.

Reassign the number 1 to the 3rd and 4th rows.

Reassign the number 1 to the 3rd and 4th columns.

Sample Problem 2-3: Matrix manipulation

Given are a 5×6 matrix A, a 3×6 matrix B, and a 9-element vector v.

$$A = \begin{bmatrix} 2 & 5 & 8 & 11 & 14 & 17 \\ 3 & 6 & 9 & 12 & 15 & 18 \\ 4 & 7 & 10 & 13 & 16 & 19 \\ 5 & 8 & 11 & 14 & 17 & 20 \\ 6 & 9 & 12 & 15 & 18 & 21 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 10 & 15 & 20 & 25 & 30 \\ 30 & 35 & 40 & 45 & 50 & 55 \\ 55 & 60 & 65 & 70 & 75 & 80 \end{bmatrix}$$

$$v = \begin{bmatrix} 99 & 98 & 97 & 96 & 95 & 94 & 93 & 92 & 91 \end{bmatrix}$$

Create the three arrays in the Command Window, and then, by writing one command, replace the last four columns of the first and third rows of A with the first four columns of the first two rows of B, the last four columns of the fourth row of A with the elements 5 through 8 of v, and the last four columns of the fifth row of A with columns 3 through 5 of the third row of B.

Solution

```
>> A=[2:3:17; 3:3:18; 4:3:19; 5:3:20; 6:3:21]
A =
     2     5     8    11    14    17
     3     6     9    12    15    18
     4     7    10    13    16    19
     5     8    11    14    17    20
     6     9    12    15    18    21
>> B=[5:5:30; 30:5:55; 55:5:80]
B =
     5    10    15    20    25    30
    30    35    40    45    50    55
    55    60    65    70    75    80
>> v=[99:-1:91]
v =
    99    98    97    96    95    94    93    92    91
>> A([1 3 4 5],3:6)=[B([1 2],1:4); v(5:8); B(3,2:5)]
```

4×4 matrix made of columns 3 through 6 of rows 1, 3, 4, and 5.

4×4 matrix. The first two rows are columns 1 through 4 of rows 1 and 2 of matrix `B`. The third row consists of elements 5 through 8 of vector `v`. The fourth row consists of columns 2 through 5 of row 3 of matrix `B`.

```
A =
     2     5     5    10    15    20
     3     6     9    12    15    18
     4     7    30    35    40    45
     5     8    95    94    93    92
     6     9    60    65    70    75
```

2.10 STRINGS AND STRINGS AS VARIABLES

- A string is an array of characters. It is created by typing the characters within single quotes.
- Strings can include letters, digits, other symbols, and spaces.
- Examples of strings: 'ad ef ', '3%fr2', '{edcba:21!', 'MATLAB'.
- A string that contains a single quote is created by typing two single quotes within the string.
- When a string is being typed in, the color of the text on the screen changes to maroon when the first single quote is typed. When the single quote at the end of the string is typed, the color of the string changes to purple.

Strings have several different uses in MATLAB. They are used in output commands to display text messages (Chapter 4), in formatting commands of plots (Chapter 5), and as input arguments of some functions (Chapter 7). More details are given in these chapters when strings are used for these purposes.

- When strings are being used in formatting plots (labels to axes, title, and text notes), characters within the string can be formatted to have a specified font, size, position (uppercase, lowercase), color, etc. See Chapter 5 for details.

Strings can also be assigned to variables by simply typing the string on the right side of the assignment operator, as shown in the examples below:

```
>> a='FRty 8'
a =
FRty 8
>> B='My name is John Smith'
B =
My name is John Smith
>>
```

When a variable is defined as a string, the characters of the string are stored in an array just as numbers are. Each character, including a space, is an element in the array. This means that a one-line string is a row vector in which the number of elements is equal to the number of characters. The elements of the vectors are addressed by position. For example, in the vector `B` that was defined above the 4th element is the letter n, the 12th element is J, and so on.

```
>> B(4)
ans =
n
>> B(12)
ans =
J
```

As with a vector that contains numbers, it is also possible to change specific elements by addressing them directly. For example, in the vector `B` above the name John can be changed to Bill by:

```
>> B(12:15)='Bill'
B =
My name is Bill Smith
>>
```

Using a colon to assign new characters to elements 12 through 15 in the vector `B`.

Strings can also be placed in a matrix. As with numbers, this is done by typing a semicolon ; (or pressing the **Enter** key) at the end of each row. Each row must be typed as a string, which means that it must be enclosed in single quotes. In addition, as with a numerical matrix, all rows must have the same number of elements. This requirement can cause problems when the intention is to create rows with specific wording. Rows can be made to have the same number of elements by adding spaces.

MATLAB has a built-in function named `char` that creates an array with rows having the same number of characters from an input of rows not all of the same length. MATLAB makes the length of all the rows equal to that of the longest row by adding spaces at the end of the short lines. In the `char` function, the rows are entered as strings separated by a comma according to the following format:

```
variable_name = char('string 1','string 2','string 3')
```

For example:

```
>> Info=char('Student Name:','John Smith','Grade:','A+')
Info =
Student Name:
John Smith
Grade:
A+
>>
```

A variable named `Info` is assigned four rows of strings, each with different length.

The function `char` creates an array with four rows with the same length as the longest row by adding empty spaces to the shorter lines.

A variable can be defined as either a number or a string made up of the same digits. For example, as shown below, x is defined to be the number 536, and y is defined to be a string made up of the digits 536.

```
>> x=536
x =
   536
>> y='536'
y =
536
>>
```

The two variables are not the same even though they appear identical on the screen. Note that the characters 536 in the line below the x= are indented, while the characters 536 in the line below the y= are not indented. The variable x can be used in mathematical expressions, whereas the variable y cannot.

2.11 Problems

1. Create a row vector that has the following elements: 8, $10/4$, 12×1.4, 51, $\tan 85°$, $\sqrt{26}$, and 0.15.

2. Create a row vector that has the following elements: $\sqrt{15} \times 10^3$, $\frac{25}{14-6^2}$, $\ln 35/0.4^3$, $\frac{\sin 65°}{\cos 80°}$, 129, and $\cos^2(\pi/20)$.

3. Create a column vector that has the following elements: 25.5, $\frac{(14 \tan 58°)}{(2.1^2+11)}$, $6!$, 2.7^4, 0.0375, and $\pi/5$.

4. Create a column vector that has the following elements: $\frac{32}{3.2^2}$, $\sin^2 35°$, 6.1, $\ln 29^2$, 0.00552, $\ln^2 29$, and 133.

5. Define the variables $x = 0.85$, $y = 12.5$, and then use them to create a column vector that has the following elements: y, y^x, $\ln(y/x)$, $x \times y$, and $x+y$.

6. Define the variables $a = 3.5$, $b = -6.4$, and then use them to create a row vector that has the following elements: a, a^2, a/b, $a \cdot b$, and $\sqrt{a}$.