Creating and Concatenating Matrices

Overview

The most basic MATLAB® data structure is the *matrix*: a two-dimensional, rectangularly shaped data structure capable of storing multiple elements of data in an easily accessible format. These data elements can be numbers, characters, logical states of `true` or `false`, or even other MATLAB structure types. MATLAB uses these two-dimensional matrices to store single numbers and linear series of numbers as well. In these cases, the dimensions are 1-by-1 and 1-by-n respectively, where `n` is the length of the numeric series. MATLAB also supports data structures that have more than two dimensions. These data structures are referred to as *arrays* in the MATLAB documentation.

MATLAB is a matrix-based computing environment. All of the data that you enter into MATLAB is stored in the form of a matrix or a multidimensional array. Even a single numeric value like `100` is stored as a matrix (in this case, a matrix having dimensions 1-by-1):

```
A = 100;

whos A
  Name      Size                    Bytes  Class

  A         1x1                         8  double array
```

Regardless of the class being used, whether it is numeric, character, or logical `true` or `false` data, MATLAB stores this data in matrix (or array) form. For example, `'Hello World'` is a 1-by-11 matrix of individual character elements in MATLAB. You can also build matrices composed of more complex classes, such as MATLAB structures and cell arrays.

To create a matrix of basic data elements such as numbers or characters, see

- Constructing a Simple Matrix
- Specialized Matrix Functions

To build a matrix composed of other matrices, see

- Concatenating Matrices
- Matrix Concatenation Functions

Constructing a Simple Matrix

The simplest way to create a matrix in MATLAB is to use the matrix constructor operator, `[]`. Create a row in the matrix by entering elements (shown as `E` below) within the brackets. Separate each element with a comma or space:

```
row = [E1, E2, ..., Em]          row = [E1 E2 ... Em]
```

For example, to create a one row matrix of five elements, type

```
A = [12 62 93 -8 22];
```

To start a new row, terminate the current row with a semicolon:

```
A = [row1; row2; ...; rown]
```

This example constructs a 3 row, 5 column (or 3-by-5) matrix of numbers. Note that all rows must have the same number of elements:

```
A = [12 62 93 -8 22; 16 2 87 43 91; -4 17 -72 95 6]
A =
    12    62    93    -8    22
    16     2    87    43    91
    -4    17   -72    95     6
```

The square brackets operator constructs two-dimensional matrices only, (including 0-by-0, 1-by-1, and 1-by-n matrices). To construct arrays of more than two dimensions, see Creating Multidimensional Arrays.

For instructions on how to read or overwrite any matrix element, see Matrix Indexing.

Entering Signed Numbers

When entering signed numbers into a matrix, make sure that the sign immediately precedes the numeric value. Note that while the following two expressions are equivalent,

```
7 -2 +5                                        7 - 2 + 5
ans =                                           ans =
    10                                              10
```

the next two are *not*:

```
[7 -2 +5]                                     [7 - 2 + 5]
ans =                                           ans =
     7    -2     5                                  10
```

Specialized Matrix Functions

MATLAB has a number of functions that create different kinds of matrices. Some create specialized matrices like the Hankel or Vandermonde matrix. The functions shown in the table below create matrices for more general use.

Function	Description
`ones`	Create a matrix or array of all ones.
`zeros`	Create a matrix or array of all zeros.
`eye`	Create a matrix with ones on the diagonal and zeros elsewhere.
`accumarray`	Distribute elements of an input matrix to specified locations in an output matrix, also allowing for accumulation.
`diag`	Create a diagonal matrix from a vector.
`magic`	Create a square matrix with rows, columns, and diagonals that add up to the same number.
`rand`	Create a matrix or array of uniformly distributed random numbers.
`randn`	Create a matrix or array of normally distributed random numbers and arrays.
`randperm`	Create a vector (1-by-n matrix) containing a random permutation of the specified integers.

Most of these functions return matrices of type `double` (double-precision floating point). However, you can easily build basic arrays of any numeric type using the `ones`, `zeros`, and `eye` functions.

To do this, specify the MATLAB class name as the last argument:

```
A = zeros(4, 6, 'uint32')
A =

  4×6 uint32 matrix

   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
   0   0   0   0   0   0
```

Examples

Here are some examples of how you can use these functions.

Creating a Magic Square Matrix. A magic square is a matrix in which the sum of the elements in each column, or each row, or each main diagonal is the same. To create a 5-by-5 magic square matrix, use the `magic` function as shown.

```
A = magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Note that the elements of each row, each column, and each main diagonal add up to the same value: 65.

Creating a Diagonal Matrix. Use `diag` to create a diagonal matrix from a vector. You can place the vector along the main diagonal of the matrix, or on a diagonal that is above or below the main one, as shown here. The `-1` input places the vector one row below the main diagonal:

```
A = [12 62 93 -8 22];

B = diag(A, -1)
B =
     0     0     0     0     0     0
    12     0     0     0     0     0
     0    62     0     0     0     0
     0     0    93     0     0     0
     0     0     0    -8     0     0
     0     0     0     0    22     0
```

Concatenating Matrices

Matrix concatenation is the process of joining one or more matrices to make a new matrix. The brackets `[]` operator discussed earlier in this section serves not only as a matrix constructor, but also as the MATLAB concatenation operator. The expression `C = [A B]` horizontally concatenates matrices `A` and `B`. The expression `C = [A; B]` vertically concatenates them.

This example constructs a new matrix `C` by concatenating matrices `A` and `B` in a vertical direction:

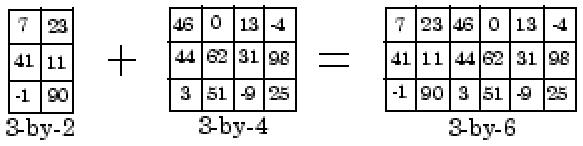
```
A = ones(2, 5) * 6;        % 2-by-5 matrix of 6's
B = rand(3, 5);            % 3-by-5 matrix of random values

C = [A; B]                 % Vertically concatenate A and B
C =
    6.0000    6.0000    6.0000    6.0000    6.0000
    6.0000    6.0000    6.0000    6.0000    6.0000
    0.9501    0.4860    0.4565    0.4447    0.9218
    0.2311    0.8913    0.0185    0.6154    0.7382
    0.6068    0.7621    0.8214    0.7919    0.1763
```

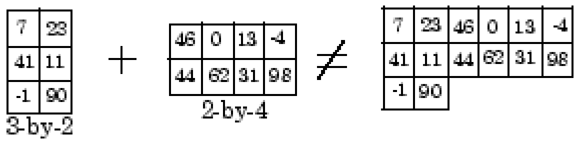
Keeping Matrices Rectangular

You can construct matrices, or even multidimensional arrays, using concatenation as long as the resulting matrix does not have an irregular shape (as in the second illustration shown below). If you are building a matrix horizontally, then each component matrix must have the same number of rows. When building vertically, each component must have the same number of columns.

This diagram shows two matrices of the same height (i.e., same number of rows) being combined horizontally to form a new matrix.



The next diagram illustrates an attempt to horizontally combine two matrices of unequal height. MATLAB does not allow this.



Matrix Concatenation Functions

The following functions combine existing matrices to form a new matrix.

Function	Description
`cat`	Concatenate matrices along the specified dimension
`horzcat`	Horizontally concatenate matrices
`vertcat`	Vertically concatenate matrices
`repmat`	Create a new matrix by replicating and tiling existing matrices
`blkdiag`	Create a block diagonal matrix from existing matrices

Examples

Here are some examples of how you can use these functions.

Concatenating Matrices and Arrays. An alternative to using the `[]` operator for concatenation are the three functions `cat`, `horzcat`, and `vertcat`. With these functions, you can construct matrices (or multidimensional arrays) along a specified dimension. Either of the following commands accomplish the same task as the command `C = [A; B]` used in the section on Concatenating Matrices:

```
C = cat(1, A, B);       % Concatenate along the first dimension
C = vertcat(A, B);      % Concatenate vertically
```

Replicating a Matrix. Use the `repmat` function to create a matrix composed of copies of an existing matrix. When you enter

```
repmat(M, v, h)
```

MATLAB replicates input matrix `M` `v` times vertically and `h` times horizontally. For example, to replicate existing matrix `A` into a new matrix `B`, use

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


B = repmat(A, 2, 4)
B =
   8   1   6   8   1   6   8   1   6   8   1   6
   3   5   7   3   5   7   3   5   7   3   5   7
   4   9   2   4   9   2   4   9   2   4   9   2
   8   1   6   8   1   6   8   1   6   8   1   6
   3   5   7   3   5   7   3   5   7   3   5   7
   4   9   2   4   9   2   4   9   2   4   9   2
```

Creating a Block Diagonal Matrix. The `blkdiag` function combines matrices in a diagonal direction, creating what is called a block diagonal matrix. All other elements of the newly created matrix are set to zero:

```
A = magic(3);
B = [-5 -6 -9; -4 -4 -2];
C = eye(2) * 8;

D = blkdiag(A, B, C)
D =
   8   1   6   0   0   0   0   0
   3   5   7   0   0   0   0   0
   4   9   2   0   0   0   0   0
   0   0   0  -5  -6  -9   0   0
   0   0   0  -4  -4  -2   0   0
   0   0   0   0   0   0   8   0
   0   0   0   0   0   0   0   8
```

Generating a Numeric Sequence

Because numeric sequences can often be useful in constructing and indexing into matrices and arrays, MATLAB provides a special operator to assist in creating them.

This section covers

- The Colon Operator
- Using the Colon Operator with a Step Value

The Colon Operator

The colon operator (`first:last`) generates a 1-by-n matrix (or *vector*) of sequential numbers from the `first` value to the `last`. The default sequence is made up of incremental values, each 1 greater than the previous one:

```
A = 10:15
A =
    10    11    12    13    14    15
```

The numeric sequence does not have to be made up of positive integers. It can include negative numbers and fractional numbers as well:

```
A = -2.5:2.5
A =
   -2.5000   -1.5000   -0.5000    0.5000    1.5000    2.5000
```

By default, MATLAB always increments by exactly 1 when creating the sequence, even if the ending value is not an integral distance from the start:

```
A = 1:6.3
A =
     1     2     3     4     5     6
```

Also, the default series generated by the colon operator always contains increments rather than decrements. The operation shown in this example attempts to increment from 9 to 1 and thus MATLAB returns an empty matrix:

```
A = 9:1
A =
   Empty matrix: 1-by-0
```

The next section explains how to generate a nondefault numeric series.

Using the Colon Operator with a Step Value

To generate a series that does not use the default of incrementing by 1, specify an additional value with the colon operator (`first:step:last`). In between the starting and ending value is a `step` value that tells MATLAB how much to increment (or decrement, if `step` is negative) between each number it generates.

To generate a series of numbers from 10 to 50, incrementing by 5, use

```
A = 10:5:50
A =
    10    15    20    25    30    35    40    45    50
```

You can increment by noninteger values. This example increments by `0.2`:

```
A = 3:0.2:3.8
A =
    3.0000    3.2000    3.4000    3.6000    3.8000
```

To create a sequence with a decrementing interval, specify a negative step value:

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