Find Array Elements That Meet a Condition

Try This Example▾

This example shows how to filter the elements of an array by applying conditions to the array. For instance, you can examine the even elements in a matrix, find the location of all 0s in a multidimensional array, or replace `NaN` values in data. You can perform these tasks using a combination of the relational and logical operators. The relational operators (`>`, `<`, `>=`, `<=`, `==`, `~=`) impose conditions on the array, and you can apply multiple conditions by connecting them with the logical operators `and`, `or`, and `not`, respectively denoted by the symbols `&`, `|`, and `~`.

Apply a Single Condition

To apply a single condition, start by creating a 5-by-5 matrix that contains random integers between 1 and 15. Reset the random number generator to the default state for reproducibility.

```
rng default
A = randi(15,5)
```

```
A = 5×5

    13     2     3     3    10
    14     5    15     7     1
     2     9    15    14    13
    14    15     8    12    15
    10    15    13    15    11
```

Use the relational *less than* operator, `<`, to determine which elements of `A` are less than 9. Store the result in `B`.

```
B = A < 9
```

```
B = 5x5 logical array

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

The result is a logical matrix. Each value in `B` represents a logical `1` (`true`) or logical `0` (`false`) state to indicate whether the corresponding element of `A` fulfills the condition `A < 9`. For example, `A(1,1)` is `13`, so `B(1,1)` is logical `0` (`false`). However, `A(1,2)` is `2`, so `B(1,2)` is logical `1` (`true`).

Although `B` contains information about *which* elements in `A` are less than 9, it doesn't tell you what their *values* are. Rather than comparing the two matrices element by element, you can use `B` to index into `A`.

```
A(B)
```

```
ans = 8×1

     2
     2
     5
     3
     8
     3
     7
     1
```

The result is a column vector of the elements in `A` that are less than 9. Since `B` is a logical matrix, this operation is called **logical indexing**. In this case, the logical array being used as an index is the same size as the other array, but this is not a requirement. For more information, see Array Indexing.

Some problems require information about the *locations* of the array elements that meet a condition rather than their actual values. In this example, you can use the `find` function to locate all of the elements in `A` less than 9.

```
I = find(A < 9)
```

```
I = 8×1

     3
     6
     7
    11
    14
    16
    17
    22
```

The result is a column vector of linear indices. Each index describes the location of an element in `A` that is less than 9, so in practice `A(I)` returns the same result as `A(B)`. The difference is that `A(B)` uses logical indexing, whereas `A(I)` uses linear indexing.

Apply Multiple Conditions

You can use the logical `and`, `or`, and `not` operators to apply any number of conditions to an array; the number of conditions is not limited to one or two.

First, use the logical `and` operator, denoted `&`, to specify two conditions: the elements must be **less than 9** and **greater than 2**. Specify the conditions as a logical index to view the elements that satisfy both conditions.

```
A(A<9 & A>2)
```

```
ans = 5×1

     5
     3
     8
     3
     7
```

The result is a list of the elements in A that satisfy both conditions. Be sure to specify each condition with a separate statement connected by a logical operator. For example, you cannot specify the conditions above by A(2<A<9), since it evaluates to A(2<A | A<9).

Next, find the elements in A that are **less than 9** and **even numbered**.

```
A(A<9 & ~mod(A,2))
```

```
ans = 3×1

     2
     2
     8
```

The result is a list of all even elements in A that are less than 9. The use of the logical NOT operator, ~, converts the matrix mod(A,2) into a logical matrix, with a value of logical 1 (true) located where an element is evenly divisible by 2.

Finally, find the elements in A that are **less than 9** and **even numbered** and **not equal to 2**.

```
A(A<9 & ~mod(A,2) & A~=2)
```

```
ans = 8
```

The result, 8, is even, less than 9, and not equal to 2. It is the only element in A that satisfies all three conditions.

Use the find function to get the index of the element equal to 8 that satisfies the conditions.

```
find(A<9 & ~mod(A,2) & A~=2)
```

```
ans = 14
```

The result indicates that A(14) = 8.

Replace Values That Meet a Condition

Sometimes it is useful to simultaneously change the values of several existing array elements. Use logical indexing with a simple assignment statement to replace the values in an array that meet a condition.

Replace all values in A that are greater than 10 with the number 10.

```
A(A>10) = 10
```

```
A = 5×5

    10     2     3     3    10
    10     5    10     7     1
     2     9    10    10    10
    10    10     8    10    10
    10    10    10    10    10
```

Next, replace all values in A that are not equal to 10 with a NaN value.

```
A(A~=10) = NaN
```

```
A = 5×5

    10   NaN   NaN   NaN    10
    10   NaN    10   NaN   NaN
   NaN   NaN    10    10    10
    10    10   NaN    10    10
    10    10    10    10    10
```

Lastly, replace all of the NaN values in A with zeros and apply the logical NOT operator, ~A.

```
A(isnan(A)) = 0;
C = ~A
```

```
C = 5x5 logical array

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

The resulting matrix has values of logical 1 (true) in place of the NaN values, and logical 0 (false) in place of the 10s. The logical NOT operation, ~A, converts the numeric array into a logical array such that A&C returns a matrix of logical 0 (false) values and A|C returns a matrix of logical 1 (true) values.

See Also

Logical Operators: Short Circuit | and | find | isnan | nan | not | or | xor