matlab$^{\text{tricks.com}}$

28 October, 2013

Tutorial on matrix indexing in MATLAB

- *How can I extract rows, columns, elements from a MATLAB matrix?*
- *I want to index a more complicated area, how can I do it?*
- *How can I collect elements from a matrix fufiling given conditions?*
- *Can you show me some examples of MATLAB matrix indexing?*

Indexing into a matrix is the way to access its elements, rows or columns: in MATLAB there are several different approaches of this operation. At the end of this post a demonstration with several examples is available.

If you are not familiar with the colon (:) operator, please have a look at MATLAB documentation.

About matrices

A matrix is generally an array of numbers in rectangular form:

$$A = \begin{bmatrix} 1 & 6 & 7 & 8 & 2 \\ 1 & 5 & 6 & 7 & 3 \\ 1 & 8 & 9 & 2 & 3 \\ 0 & 5 & 2 & 9 & 9 \end{bmatrix}$$

A is a matrix having four rows and five columns. Its second row is marked blue, its third column is red. The element at position *(2,3)* is green: this can be marked $\mathbf{A}_{2,3}$ too. Note, that the first index identifies the row, while the second is for the column.

If a matrix has a single row, it is called a *row vector*, if it has a single column, it is a *column vector*. If the matrix has one row and one column, it is a *scalar*.

Below B is a row vector, C is a column vector and D is a scalar.

$$B = \begin{bmatrix} 1 & 6 & 7 & 8 & 2 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$
$$D = \begin{bmatrix} 1 \end{bmatrix}$$

Indexing methods

In MATLAB each matrix has a name. For example if we have created a matrix **A**:

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

Later we can invoke it by its name:

```
>> A
```

```
A =
    1    2    3    4
    5    6    7    8
```

If we want to index the elements, we mostly use one of the following forms, where **A** shall be of course the identifier of matrix we want to index into:

```
% usual matrix indexing of 2d matrix
A(rowId, columnId)

% usual matrix indexing of 3d matrix
A(rowId, columnId, planeId)

% linear matrix indexing
A(linearId)
```

The most important thing is to note: if one parameter is given, it is always *linear indexing*, otherwise it is *usual indexing*.

Usual indexing

The first parameter contains the row, the second parameter contains the column index of the element we want to select. Multiple indices can be given, too:

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

>> A(2, 3)
ans =
    7

>> A([1 2], [3 4])
ans =
    3    4
    7    8
```

The first example selects $\mathbf{A}_{2,3}$, the second example selects all elements between rectangle $\mathbf{A}_{1,3}$ - $\mathbf{A}_{2,4}$.

To visualize the selection process, look at the following code and figure, where row #2 and #3, and column #1, #3 and #4 are selected:

```
>> A([2:3], [1 3:4])
ans =
    1    6    7
    1    9    2
```

$$A = \begin{bmatrix} \color{red}{1} & 6 & \color{red}{7} & \color{red}{8} & 2 \\ \color{green}{1} & \color{blue}{5} & \color{green}{6} & \color{green}{7} & \color{blue}{3} \\ \color{green}{1} & \color{blue}{8} & \color{green}{9} & \color{green}{2} & \color{blue}{3} \\ \color{red}{0} & 5 & \color{red}{2} & \color{red}{9} & 9 \end{bmatrix}$$

The selected rows are blue, the selected columns are red. *The intersection*, which *is the result*, is green. If we give a single row and column index, the result will be a scalar value.

To select a whole row or column, use the colon (:) operator: using a single colon means, that all elements shall be selected in the appropriate dimension. For example to select all elements in the first row, use:

```
>> A(1, :)
ans =
   1    6    7    8    2
```

Usual indexing into 3d matrices

A colored image is a 3d matrix. It has the two good old dimensions, and a *plane* dimension containing the red, green and blue values respectively.

$$\begin{array}{cccccc} R & R & \cdots & R & & \\ R & R & \cdots & R & G & \\ \vdots & \vdots & \ddots & \vdots & G & B \\ R & R & \cdots & R & \vdots & B \\ & G & G & \cdots & G & \vdots \\ & & B & B & \cdots & B \end{array}$$

To extract the different planes, use the following code:

```
% read image from an external file
im = imread('image.png');

% extract the color planes: each row, each column and given plane
% R, G and B will contain the red, green and blue pixels respectively
R = im(:, :, 1);
G = im(:, :, 2);
B = im(:, :, 3);
```

Linear indexing

Although usual indexing is good for selecting rectangular areas, rows and columns, sometimes selecting other kind of formation is needed, which cannot be solved by this way. For example have a look at the figure below, and suppose, you want to select the elements marked by red:

$$A = \begin{bmatrix} 1 & 6 & 7 & 8 & 2 \\ 1 & 5 & 6 & 7 & 3 \\ 1 & 8 & 9 & 2 & 3 \\ 0 & 5 & 2 & 9 & 9 \end{bmatrix}$$

It is impossible with usual indexing. But in MATLAB each element of an array can be indexed by a so-called linear index, too. This increases in the following way – the linear index is in superscript:

$$A = \begin{bmatrix} 1^1 & 6^5 & 7^9 & 8^{13} & 2^{17} \\ 1^2 & 5^6 & 6^{10} & 7^{14} & 3^{18} \\ 1^3 & 8^7 & 9^{11} & 2^{15} & 3^{19} \\ 0^4 & 5^8 & 2^{12} & 9^{16} & 9^{20} \end{bmatrix}$$

In case of linear indexing, the only input parameter is a vector containing the linear indices of the needed elements. The output follows the form of the input:

```
>> A = [1 6 7 8 2; 1 5 6 7 3; 1 8 9 2 3; 0 5 2 9 9]
A =
   1    6    7    8    2
```

```
     1     5     6     7     3
     1     8     9     2     3
     0     5     2     9     9

>> A([1 5 6 10 11 15 16 20])
ans =
     1     6     5     6     9     2     9     9

>> A([1 5; 6 10; 11 15; 16 20])
ans =
     1     6
     5     6
     9     2
     9     9
```

Calculating linear indices

It may seem hard to calculate the appropriate linear indices, but MATLAB provides sub2ind function to do this. This way we only need the *x* and *y* coordinates of the desired points and the *size* of the indexed matrix: the output of this function is a group of linear indices.

Have a look at our previous example again:

$$A = \begin{bmatrix} {\color{red}1} & {\color{red}6} & 7 & 8 & 2 \\ 1 & {\color{red}5} & {\color{red}6} & 7 & 3 \\ 1 & 8 & {\color{red}9} & {\color{red}2} & 3 \\ 0 & 5 & 2 & {\color{red}9} & {\color{red}9} \end{bmatrix}$$

Sample code:

```
% the input matrix
A = [1 6 7 8 2; 1 5 6 7 3; 1 8 9 2 3; 0 5 2 9 9]

% the row and column indices of the desired points
r = [1 1 2 2 3 3 4 4];
c = [1 2 2 3 3 4 4 5];

% the calculated linear indices
indices = sub2ind(size(A), r, c)

% indexing into A
A(indices)
```

The output is:

```
A =
     1     6     7     8     2
     1     5     6     7     3
     1     8     9     2     3
     0     5     2     9     9

indices =
     1     5     6    10    11    15    16    20

ans =
     1     6     5     6     9     2     9     9
```

The output is the same as in the previous example.

The end keyword

We can use a special keyword for indexing: *end*, which will be equal to the size of the indexed dimension. Any mathematical operations can be done on this keyword.Take our good old input **A** matrix:

```
A =
    1    6    7    8    2
    1    5    6    7    3
    1    8    9    2    3
    0    5    2    9    9
```

See this example for usual indexing:

```
A(1 : end, 1 : end)
```

Here the first *end* is 4 and the second *end* is 5, since matrix **A** has four rows and five columns.

At linear indexing *end* means the number of elements, since in this case we see matrix **A** as a one-dimensional vector. So *end* now equals to 20:

```
A(1 : end)
```

This is an example of mathematical operations on *end*, the intersection of the first two rows and the first three columns will be selected:

```
A(1 : end / 2, 1 : end - 2)
```

Logical indexing

This is the third way of indexing in MATLAB. But there is a big difference in the logic behind them.

> In case of usual or linear indexing we tell, which indices we do need. In case of logical indexing we use logical values to tell for each element that we need or not need it. So not indices, but logical values are used for indexing.

Some examples will make it clear:

```
% the input matrix
A = [1 6 7 8 2; 1 5 6 7 3; 1 8 9 2 3; 0 5 2 9 9]

% usual indexing
A([2 3], [1 3 5])

% logical indexing
A(logical([0 1 1 0]), logical([1 0 1 0 1]))
```

The output is:

```
A =
    1    6    7    8    2
    1    5    6    7    3
    1    8    9    2    3
    0    5    2    9    9
```

```
ans =
    1    6    3
    1    9    3

ans =
    1    6    3
    1    9    3
```

The output is the same in both cases. At usual indexing we told, that we need rows #2 and #3. At logical indexing we told by a logical array for each rows, that they are *not-needed - needed - needed - not needed*, which equals to the following logical array:

```
[0 1 1 0]
```

The same is true for the columns.

There is another often used way for logical indexing. In this case the parameter is a logical matrix, which has equal dimensions as the indexed one, an it tells for each element, whether it is needed, or not:

```
% the input matrix
A = [1 6 7 8 2; 1 5 6 7 3; 1 8 9 2 3; 0 5 2 9 9]

% matrix for logical indexing
indices = logical([1 0 1 0 0;
                   1 0 0 1 0;
                   0 0 0 0 0;
                   1 0 1 1 0])

% logical indexing of each element
A(indices)
```

The output is:

```
A =
    1    6    7    8    2
    1    5    6    7    3
    1    8    9    2    3
    0    5    2    9    9

indices =
    1    0    1    0    0
    1    0    0    1    0
    0    0    0    0    0
    1    0    1    1    0

ans =
    1
    1
    0
    7
    2
    7
    9
```

Only those elements are selected, where *true* stands in the indexing matrix. Note, that the elements follow each other by their linear index.

This approach is mostly used for cases, like this:

```
% the input matrix
A = [1 6 7 8 2; 1 5 6 7 3; 1 8 9 2 3; 0 5 2 9 9]

% get a logical array of elements greater than 7
indices = A > 7

% index the matrix, get the appropriate elements
A(indices)
```

The output is:

```
A =
    1    6    7    8    2
    1    5    6    7    3
    1    8    9    2    3
    0    5    2    9    9

indices =
    0    0    0    1    0
    0    0    0    0    0
    0    1    1    0    0
    0    0    0    1    1

ans =
    8
    9
    8
    9
    9
```

All elements greater than seven were selected from the **A** matrix by a simple comparison and logical indexing.

Demonstration

Please click below onto the label to load the interactive demonstration on MATLAB matrix indexing. The selected cells are visualized. A format of a cell is: the linear index is in the superscript while the 2d coordinates are in subscript. The value of a cell equals to its linear index for simplicity.

Click to load application

Select the different examples by clicking on their names.

Tweet 0 +1 0

Tags: tutorial matrix indexing application

New comment

comments powered by Disqus

Atom · Powered by Nibbleblog – © Zoltán Fegyver – 2013