MATLAB® News & Notes

MATLAB Programming Patterns

Simplify your code with comma-separated lists

by Nausheen Moulana and Peter Webb

Comma-separated lists of variables, such as, `x,y,z,` appear frequently in MATLAB code; they are most commonly found inside of `{}, ()` and `[]` . In most cases, specifying the list of variables explicitly is sufficient, but sometimes you need a more flexible technique. You can store variable values in a cell array or structure array, and then expand your data into a comma-separated list via `c{:}`, if c is a cell, or via `s.field`, if s is a structure array. This simplifies certain problems involving indexing and cell or structure array manipulation and often allows you to write shorter, more efficient code. The comma-separated list economizes the syntax for extracting multiple values from cell and structure arrays. For example, `c{:}` is equivalent to `c{1},c{2},c{3},…c{end}`, a list of values separated by commas. Similarly for a structure array, `s.field` is equivalent to `s(1).field,s(2).field,…s(end). field`. The four uses of comma-separated lists are:

- Within `[]` to perform horizontal matrix concatenation. For example, if c is a cell array containing scalars, where `c{i} = i`, you can create a matrix consisting of the individual elements of c using `[c{:}]`. This usage allows you to convert cell or structure arrays to numeric arrays in an efficient and convenient manner.

- As input or output parameters to function calls. Functions that take a variable number of input arguments or return a variable number of outputs can use comma-separated lists. Use this technique when you need to build argument lists to functions while your program is executing or when you need to store the return values from functions in a single variable, i.e., a cell or structure, for later processing. This usage is common with functions that use `varargin` and `varargout`.

- Within `()` to create an indexed expression. This usage is effective when dealing with n-dimensional arrays (see Pattern 1, below).

- Within `{}` to create cell arrays. For example, `b = {c{:}, magic(3)}` creates a cell array whose elements consists of the elements of cell c and a 3-by-3 magic square.

Writing code with cell or structure arrays allows you to take advantage of comma-separated list coding techniques, which are the foundation for some of the programming patterns in MATLAB.

Pattern 1: Comma-separated lists and indexing

Using comma-separated lists in indexing operations can simplify and speed up your code. For example, let's look at the `fftshift` function.

```
function y = fftshift(x)

numDims = ndims(x);
idx = cell(1, numDims);
for k = 1:numDims
      m = size(x, k);
      p = ceil(m/2);
      idx{k} = [p+1:m 1:p];
end
y = x(idx{:});
```

Given an N-dimensional matrix, `fftshift` swaps "half-spaces" along each dimension. This is fundamentally an indexing operation: given, for example, the vector `a = [5 6 7 8 9 0]`, we can swap the left and right halves of this vector with the command `b = a([4:6 1:3])`; note that since a is one-dimensional, swapping requires only one index vector. `fftshift` performs this kind of index-based swapping in N dimensions and thus must construct N index vectors. The swapping operation is simply `y = x(index1,index2,…, indexN)`.

```
if ndims(x) == 1
      y = x(index1);
else if ndims(x) == 2
      y = x(index1, index2);
end
```

If you are using explicit indexing, you'll need to write one if statement for each dimension you want your function to handle. A comma-separated list makes it very easy to generalize this swapping operation to an arbitrary number of dimensions.

`fftshift` stores the index vectors in a cell array. Building this cell array is relatively simple. For each of the N dimensions, determine the size of that dimension and find the integer index nearest the midpoint. Then, construct a vector that swaps the two halves of that dimension. Once all the vectors have been collected into this cell array, a single MATLAB command performs the swap:

```
y = x(idx{:});
```

This technique produces an algorithm that is dimension independent and compact.

Pattern 2: Manipulating data in structure arrays

You will often find that using comma-separated lists to manipulate structure arrays makes it easier to write efficient code. For example, if you want to search for and replace a certain value in your structure array, you can easily create a function to do this.

```
function index = findinstruct(a, value)
% findinstruct takes a
% structure with the field value as its first
% argument and a double search value as its
% second argument. This function assumes that the

% structure does not contain NaNs or empties.%
Generate the indices of the desired value index =
find([a.value] == value);
```

First, use `findinstruct` to get the indices that match the value being replaced. Note that

`findinstruct` converts its input structure array to a numeric array using `[]` and a structure field comma-separated list. Next, replace the existing value with the new value by issuing this command:

```
[a(index).value] = deal(newval);
```

You must use the function `deal` here, rather than a simple assignment statement, because only functions can assign to multiple left-hand side values; in this case, `deal` copies its input into each element of the output. As a general rule, whenever you need to assign to or from a comma-separated list, use `deal` in conjunction with the `[]` concatenation operator.

Comma-separated lists and objects

MATLAB classes can change the behavior of the cell and structure indexing operators (`{}` and `.`) by overloading the `subsref` and `subsasgn` functions. MATLAB calls `subsref` when an indexing operation appears on the right hand side of an assignment statement, and `subsasgn` when the indexing appears on the left hand side. For example:

- `subsref: a = obj{m:n};`
- `subsasgn: obj.distance = value;`

If `obj` is an array of objects, MATLAB applies the same rules it uses for cell and structure arrays; thus the `{} and .` operators produce comma-separated lists. When a comma-separated list appears on either side of an assignment statement, MATLAB checks that the number of variables on the left side of the assignment matches the number of values on the right, and requires that you use `deal` to perform the assignment (see Pattern 2). MATLAB performs this test before executing the overloaded `subsref` and `subsasgn` functions by calling the `numel` function to count the number of elements in `obj{m:n}` and `obj.distance`. The built-in version of numel returns n–m+1 for the `{m:n}` case and `prod(size(obj))` in the `obj.distance` case. If there is a mismatch between the number of values on either side of the assignment as a result of calling `numel`, an error occurs and the overloaded `subsref` and `subsasgn` functions are not executed.

Therefore, if you want to modify the behavior of the comma-separated list operators with respect to object arrays (for example, your class may use `{}` to perform string indexing), you need to indicate to MATLAB that these operators return 1, for the number of elements. To do this you need to overload the `numel` function and have it return 1 for both `{}` and . cases. With this overloaded `numel` in place, the above example assignments do not require the explicit use of `deal`, and MATLAB executes the overloaded `subsref` and `subsasgn` functions.

See the help for `numel` for details on how to use it.

Summary

Using comma-separated lists helps you write compact, efficient, and extensible code. Because most cell and structure array operations are built-in functions, the convenience and flexibility does not come at the cost of performance. When working with comma-separated lists, you need to remember to use `deal` appropriately. And if you're writing a MATLAB class, you should consider whether or not to overload `numel` for that class. With these points in mind, you will find comma-separated lists a powerful and effective technique, and a very useful tool to add to your programming toolkit.

To learn more about comma-separated lists, you can read previously published articles at:

- "Getting the Most Out of the deal Function"

- “Exploiting the Comma-Separated List”

All MATLAB documentation can be viewed online and printed in PDF format; just visit www.mathworks.com/support

related topics:

Using MathWorks Products For... | Training | MATLAB Based Books | Third-Party Products