Chapter 7
Cell and Structure Arrays

Outline

7.1 Concept: Collecting Dissimilar Objects

7.2 Cell Arrays

7.3 Structures

7.4 Structure Arrays

7.5 Engineering Example—Assembling a Physical Structure

7.1 Concept: Collecting Dissimilar Objects

This chapter presents two different ways to collect items of dissimilar types (heterogeneous collections) rather than the collections seem so far that are homogeneous (all items in the collection of the same data type)

- Cell arrays are indexed collections of cells – containers that can encapsulate objects of any size and any type – double, logical, char, even cells and structures.
- Structures encapsulate the same data types, but access them by name rather than number
- Structure arrays are indexed collections of structures

7.2 Cell Arrays

A Cell is a MATLAB data type that can act as a container for any other data types.

A cell array is an array of cells.

To put a MATLAB object into a cell, enclose it in braces {...}. `{[4 5 6]} -> [4 5 6]` a cell containing a vector.

To concatenate multiple objects into a cell array:

```
ca = {true, [4 5 6], 'aceg'} -> [1] [4 5 6] 'aceg'
```

7.2 Cell Arrays

Notice that the MATLAB echo is different from that produced from a normal object: vectors have [...] and strings have '…'

Based on these examples, we observe the following

- A cell array can contain any legal MATLAB object
- Just as with number arrays, cell arrays can be created “on the fly“ by assigning values to an indexed variable

Accessing Cell Data

- Cell arrays can be indexed like vectors to manipulate the cells. Continuing the previous example,

```
ca(end:-1:1) -> 'aceg' [4 5 6] [1]
```

- To extract cell contents, index with {...}

```
ca{2}-> 4 5 6
```

Accessing Cell Data

- To store cell contents, also index with {...}

```
ca{4} = 'xx' -> 'aceg' [4 5 6] [1] 'xx‘
```

- You can, but rarely want to, put cells in cell arrays:

```
ca{4} = {'xx'} -> 'aceg' [4 5 6] [1]
{'xx'}
```

Accessing Cell Data

- Notice the following observations
  - When we extract the contents of multiple cells, this results in multiple assignments being made.
  - The multiple assignments CANNOT be made to a single variable.
  - Cell arrays can be “sliced” with normal vector indexing assignments.
  - The deal(…) function is provided to capture multiple results from multiple cells.

```
[b c]=ca{1:2} or
[b c]=deal(ca{1:2})
```

Using Cell Arrays

- Containing lists of possible values for switch/case statements, as we saw in Chapter 4
- Substituting for parameter lists in function calls
- Suppose we are provided with a cell array and have been asked for a function that find the length of all the numeric vectors it contains

Using Cell Arrays

```
function ans=listing7_2(ca)
% count the numbers in a cell array

    ans=0;
    for in=1:length(ca)
        item=ca{in};
        if isnumeric(item)
            ans=ans+prod(size(item));
        end
    end
end
```

7.3 Structures

A structure is a container for cell-like objects called fields that are accessed by field name. For example:

```
fred.first  = 'Fred';
fred.last  = 'Jones';
fred.age  = 42
```

creates a structure describing a person named Fred.

We can copy the structure fred to a sally structure with similar attributes:

```
sally = fred;
sally.first  = ‘Sally';
```

7.4 Structure Arrays

To create an address book consisting of information for various people, we can use indexing, concatenation or the function **`struct`**.

- **`book(1) = fred;`**
- **`book = [book sally]`**
- **`two_names = struct('first',{'A','B'}, …`**
  `'last', 'Jones', …`
  `'age', 35)`
- **`book = [book two_names]`**

Notice that struct(…) takes pairs of entries: a field name and value. If the value is a cell array, a structure array of that length is created. Otherwise, the value is applied to all entries in the structure array.

Operating on Structure Arrays

- The following functions operate on structure arrays:
  - **`fieldnames(book)`** returns the field names in a cell array
  - **`getfield(book(1), field_name)`** is the same as **`book(1).field_name`**
  - **`isfield(book, field_name)`** returns **`true`** if the field is a field of the structure
  - **`sa = rmfield(book, field_name)`** returns a new structure array with the specified field removed
  - **`book(1) = setfield(book(1), field_name, value)`** is the same as **`book(1).field_name = value`**.

Extracting Values from a Structure Array

The **`getfield(…)`** function accesses fields in one structure. To extract all the values of a field from a structure array, you need to provide a container.

- If the field has numeric values, use:

```
ages = [book.age] -> [42 42 35 35];
```

- If the field has other than numeric values, use:

```
ages = {book.first} ->
'Fred','Sally','A','B'
```

Homework on Chapter 7 is posted on the website:

http://www.ee.nmt.edu/~erives/289_F12/EE289.html

Homework is due within a week