MATLAB for biologists
Lecture 6

Kevin Smith
Light Microscopy Centre ETH Zurich
kevin.smith@lmc.biol.ethz.ch

April 4, 2012

1 Cell Arrays

So far we have only worked with numeric arrays in MATLAB . Cell arrays are similar to numeric arrays, but are more general. In addition to numeric values, cell arrays may also contain

- strings
- structures
- numeric arrays
- cell arrays

There are several ways to initialize a cell array

```
>> C = cell(1,5)

C =

    []    []    []    []    []
```

or

```
>> C{1,5} = [];

C =

    []    []    []    []    []
```

Cell arrays are useful for storing lists of things that have varying dimensions, such as a list of file names or a list of images of different size.

```
>> a = {1, rand(3), 'hello', imread('peppers.png')}
a =

    [1]    [3x3 double]    'hello'    [384x512x3 uint8]
```

Cell arrays can also contain other cell arrays.

```
>> b = {8, [2 54], a}

b =

    [8]    [1x2 double]    {1x4 cell}
```

Accessing data in cell arrays is slightly more tricky than numeric arrays.

```
>> a(2)

ans =

    [3x3 double]

>> class(a(2))

ans =

cell
```

This is not the result we would expect based on our experience with numeric arrays. To access the contents of a cell array, we need to use curly braces **{}**. Use parentheses **()** for indexing into a cell array to collect a subset of cells together in another cell array.

To access the 2^{nd} element of `a`, use curly braces

```
>> a{2}

ans =

    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157
```

We can use parentheses to extract the first two elements from `a` to form a new cell `d`

```
>> d = a(1:2)

d =

    [1]    [3x3 double]
```

We can extract the first two elements of `a` into separate numeric arrays `a1` and `a2`

```
>> [a1 a2] = a{1:2}

a1 =

     1
```

```
a2 =

    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157
```

As an exercise, write a function that generates a Fibonacci sequence. The input of the function should be the length of the sequence, N. Note that the sequence should always have $N + 1$ elements, we don't count the first element 0. The function should have two outputs

1. a vector containing the values of the Fibonacci sequence until N

   ```
   >> [fib fibCell] = fibDemo(6)

   fib =

        0     1     1     2     3     5     8
   ```

2. a cell array where entry n contains the sequence up to that point

```
>> fibCell{2}

ans =

     0     1     1


>> fibCell{end}

ans =

     0     1     1     2     3     5     8
```

0	0	0	0	0	0
1	1	1	1	1	1
	1	1	1	1	1
		2	2	2	2
			3	3	3
				5	5
					8

Hint: initialize the sequence to be `[0 1]`, loop from 3 to $N + 1$.

2 Structures and arrays of structures

Structures are a useful way of grouping arrays in MATLAB that belong together. For example, you might want to collect data about a person in a structure.

```
>> myStruct.name = 'Fred';
>> myStruct.height = 1.80;
>> myStruct.age = 33

myStruct =

      name: 'Fred'
    height: 1.8000
       age: 33
```

You could initialize the exact same structure using

```
>> clear;
>> myStruct = struct('name', 'Fred', ...
               'height', 1.80, 'age', 33)

myStruct =

      name: 'Fred'
    height: 1.8000
       age: 33
```

The structure array we created contains pairs of fields and values. The values can be a numeric array, string, cell, or scalar. The field names must begin with a character and are case-sensitive. In the example below, the field names appear on the left of the : and the values appear on the right. Let's add some new fields to the structure.

```
>> myStruct.favoriteFoods = {'pizza', 'chocolate'}
>> myStruct.image = imread('images/fred.jpg')

myStruct =

        firstName: 'Fred'
           height: 1.8000
              age: 33
    favoriteFoods: {'pizza'  'chocolate'}
            image: [277x220 uint8]
```

We can grow the array to include other people and measurements. By simply setting the value to one of the fields in the 2^{nd} element, the entire 2^{nd} element is initialized. However, the unspecified fields remain empty.

```
>> myStruct(2).name = 'Ginger'

myStruct =

1x2 struct array with fields:
    name
    height
    age
    favoriteFoods
    image

>> myStruct(2)

ans =

             name: 'Ginger'
           height: []
              age: []
    favoriteFoods: []
            image: []
```

If we want to fill in the missing values, we can specify each of them individually.

```
>> myStruct(2).height = 1.65;
>> myStruct(2).age = 21;
>> myStruct(2).favoriteFoods = {'Spaghetti', 'Kiwi'}
>> myStruct(2).image = imread('images/ginger.jpg');
>> myStruct(2)

ans =

             name: 'Ginger'
           height: 1.6500
              age: 21
    favoriteFoods: {'Spaghetti'  'Kiwi'}
            image: [280x220 uint8]
```

Question: What happens if we add a new field/value to `myStruct(2)`?

We've seen previously that some MATLAB functions such as `regionprops` return structure arrays as output. Another useful command that outputs structure arrays is `dir`.

```
>> d = dir

d =

7x1 struct array with fields:
    name
    date
    bytes
    isdir
    datenum

>> d(1)

ans =

       name: '.'
       date: '04-Apr-2012 00:20:12'
      bytes: 0
      isdir: 1
    datenum: 7.3496e+05

>> d(3)

ans =

       name: 'cellDemo.m'
       date: '03-Apr-2012 21:27:06'
      bytes: 1120
      isdir: 0
    datenum: 7.3496e+05
```

Useful functions related to structures: `setfield`, `getfield`, `fieldnames`, `orderfields`, `rmfield`.

3 Cell array and structure array example

Let's combine our knowledge of cell arrays and structure arrays to write a function that looks at the contents of a directory, finds all the image files, and displays the images sorted by date.

```
function showDirectoryImages(pathname)

% get directory structures filtered for different image
% types
djpg = dir([pathname '*.jpg']);
dbmp = dir([pathname '*.bmp']);
dpng = dir([pathname '*.png']);
dtif = dir([pathname '*.tif']);

% concatenate the directory structures into a single
% structure array
d = [djpg; dbmp; dpng; dtif];

% sort the array by the date
datenums = [d(:).datenum];
[datenumSorted, inds] = sort(datenums);
d = d(inds);

% initialize a cell which will store the images
images = cell(1,numel(d));

% open a figure to display the images
figure;

% loop through the images in d, load them, display them,
% and print their information
for i = 1:numel(d)
    images{i} = imread([pathname d(i).name]);
    fprintf('%d. %s %s\n', i, d(i).date, d(i).name);
    imshow(images{i});
    pause;
end
```

We can run this function from the prompt by passing it the path to the folder as an argument.

```
>> showDirectoryImages([pwd '/images/']);
```

4 Profiling your code

The MATLAB profiler helps you debug and optimize code by tracking their execution time. For each MATLAB function, MATLAB subfunction, or MEX-function in the file, profile records information about execution time, number of calls, parent functions, child functions, code line hit count, and code line execution time.

```
>> profile on;
>> profileDemo('images/corporatefatcat.jpg');
>> profview;
>> profile off;
```

5 User interface

MATLAB demo extending the segmentation example to include some useful user interfaces.

```
>> cellDemo
```

Functions we will use in our demo: `uigetfile`, `uiputfile`, `questdlg`, `uicontrol`, `impoly`.

Other useful functions related to user interfaces: `errordlg`, `inputdlg`, `uigetdir`, `uiopen`, `uisave`.