Lecture 8: Strings, Cell Arrays, and Integer Data Types

CS227-Scientific Computing

November 9, 2011
Not everything is a matrix of double-precision floating-point numbers

By default, MATLAB treats everything as a matrix of double-precision floating-point numbers.

But sometimes we want to deal with other kinds of data: character strings, integers represented exactly, ‘heterogeneous’ lists whose items are different kinds of data, etc.

While MATLAB is not really set up for this, its designers grudgingly acknowledged that real scientists have to work with non-numeric and non-floating point data, and (eventually) designed these tools into the language.
A text-analysis problem

We’ll first use MATLAB to analyze the contents of a file that holds all the words (in the order in which they were encountered) from a long novel.

We want to know the frequencies of the different words that occur in this book.

To do this, we’ll have to take a look at how MATLAB handles character strings, cell arrays, sorting, and printing.
Strings

We can create character string data by assigning a string contained within single quotation marks to a variable. This is treated as an array of characters, and can be manipulated accordingly.

Note the use of the MATLAB command `whos` in the example in the next slide to tell us the type of the variable $x$. 
String creation and manipulation: Example

```
>> x = 'strings'
x = strings
>> whos x
Name      Size    Bytes   Class   Attributes
  x        1x7     14      char
>> x(3:6)
ans = ring
>> x(2) = 'p'
x = springs
>> y = ['hope ', x, ' eternal']
y = hope springs eternal
```
Built-in string manipulation functions

```matlab
>> z=upper(x)
  z = SPRINGS
>> z==x
  ans = 0 0 0 0 0 0 0 0 0 0 0 0
>> %== is not what you want for comparing strings
>> strcmp(z,x)
  ans = 0
>> strcmp(z(1:3),'SPR')
  ans = 1
```
A cell in MATLAB is an abstract container, into which you can put anything.

When you try to access the individual elements of a cell array, there is a formal distinction between the cells themselves (which you get with expressions like \texttt{x(2)}, \texttt{x(1:3)}) and the contents of the cells, which you get by using curly braces \{\} in place of parentheses.

MATLAB is not completely consistent about this: in some instances, operations that you would have thought would be reserved for cell contents can be applied to the cells themselves.
Cell Arrays: Example

```matlab
>> mycellarray = {4, 'cat', [1; 2]};
>> whos mycellarray
   Name      Size  Bytes  Class  Attributes
  mycellarray    1x3    366  cell
>> u = mycellarray(2)
u = 'cat'
>> whos u
   Name      Size  Bytes  Class  Attributes
     u        1x1    118  cell
```
>> v=mycellarray{2}
v = cat
>> whos v
    Name      Size     Bytes  Class      Attributes
    v          1x3          6  char
>> mycellarray(1)==4
??? Undefined function or method 'eq' for input arguments of type 'cell'.
>> mycellarray{1}==4
ans = 1
Cell Arrays Example–Continued

```
>> strcmp(mycellarray{2},’cat’)
ans = 1
>> strcmp(mycellarray(2),’cat’)
ans = 1
>> % !! Go figure.
>> mycellarray(1)=2
??? Conversion to cell from double is not possible.
   >> mycellarray{1}=2
mycellarray =
    [2]   ’cat’    [1x2 double]
>> mycellarray(2)=’bat’
??? Conversion to cell from char is not possible.
   >> mycellarray{2}=’bat’
mycellarray =
    [2]   ’bat’    [1x2 double]
```
The text is contained in the file bhnew. Extracting the words from the original novel, removing punctuation, and setting up the lines were all done using tools outside of MATLAB. Once that is done, the MATLAB’s Import Wizard works fine and is able to store the data in an appropriate structure, a cell array in which each cell contains a string. (You cannot use a 2-dimensional array of characters, because all the strings have different lengths.)

>> >> whos bhnew

Name             Size         Bytes  Class   Attributes
bhnew             362025x1    43535962 cell
Counting distinct words

To count the number of distinct words, we first convert everything to upper-case.

We then sort the array (MATLAB’s sort function will work on a cell array of strings)

Then we traverse the sorted array to count the number of cells in which the string differs from the next string.

The algorithm is illustrated on the next slide, the MATLAB code for carrying this out on the following slide.
Counting distinct words–continued

Figure:
Counting distinct words–continued

```matlab
>> bhnew=upper(bhnew);
>> bhnew(1:5)
ans = 'BLEAK'
    'HOUSE'
    'BY'
    'CHARLES'
    'DICKENS'
>> sorted_words=sort(bhnew);
>> sorted_words(1:5)
ans = 'A'
    'A'
    'A'
    'A'
    'A'
```
The code below finds 15060 distinct words among the 362025 words in the novel.

```matlab
>> count = 1;
>> for j=1:length(sorted_words)-1
    if(~strcmp(sorted_words(j),sorted_words(j+1)))
        count=count+1;
    end
end
>> count
count =15060
```
Tabulating Frequencies

We now create a table that contains each word together with the number of times it occurs in the list.
The idea is that we scan the list of words once again. Each time we encounter a word that is identical to the last word in the list, we simply increment the frequency field in the current entry of the table.
If we encounter a word that is different from the last word in the list, we enter it into the next entry of the table and set the frequency field to 1.
Tabulating frequencies–continued

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AARDVARK</td>
<td>1</td>
</tr>
<tr>
<td>ABANDON</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AARDVARK</td>
<td>1</td>
</tr>
<tr>
<td>ABANDON</td>
<td>1</td>
</tr>
</tbody>
</table>

sorted_words

freq_table

Figure:
The algorithm is implemented with this script:

```matlab
freq_table=cell(15060,2);
freq_table{1,1}=sorted_words{1};
freq_table{1,2}=1;
pos = 1;
for j=2:length(sorted_words)
    if(strcmp(sorted_words{j},sorted_words{j-1}))
        freq_table{pos,2}=freq_table{pos,2}+1;
    else
        pos=pos+1;
        freq_table{pos,1}=sorted_words{j};
        freq_table{pos,2}=1;
    end
end
```
Let's just check and make sure it looks right.

```matlab
>> freq_table(1:5,:)
ans =
    'A'          [7718]
       'ABACK'    [  1]
      'ABANDON'   [  7]
      'ABANDONED' [ 13]
   'ABANDONING' [  1]
```
We now want to sort our table in descending order of frequency. Unfortunately, MATLAB’s built-in `sort` function cannot be applied to the kind of table we have made. One possibility is to implement a sorting algorithm ourselves. Another is to extract the table of frequencies and sort it in descending order, but at the same time to call `sort` with a second output argument that records the original positions of the items. The algorithm is illustrated in the next slide.
**Figure:** Extract the column of frequencies, sort, and then use second output argument of sort to create new table.
We have to explicitly convert the second column of the cell array to a matrix, then reconstitute the frequency table in sorted order.

```matlab
>> frequencies=cell2mat(freq_table(:,2));
>> [sorted_frequencies,perm]=sort(frequencies,'descend');
>> freq_table{perm(1),1}
>> for j=1:15060
      freq_table2{j,2}=sorted_frequencies(j);
      freq_table2{j,1}=freq_table{perm(j),1};
   end
```
Exporting the result

This prints the table sorted by frequency in a file 'celldata.dat'

```matlab
>> filename='celldata.dat';
>> fid=fopen(filename,'w');
>> for row=1:15060
    fprintf(fid,'%s	%d
',freq_table2{row,:});
  end
>> fclose(fid);
```
These have names like int8, uint8, int16, uint16, int32, uint32.

For example, int8 means ‘signed 8-bit integer’, which represents integers with 8 bits in the range $-2^7 = -128$ to $2^7 - 1 = 127$. uint16 means 'unsigned 16-bit integer’, which represents integers in the range 0 to $2^{16} - 1 = 65535$.

Arithmetic with such integers is exact–there is no roundoff. The rules for conversion between types are a little peculiar.
Integer Data Types–some Examples

>> a=uint8(47.5)
a =48
>> b=ceil(47.5);
>> format hex
>> a
a =30
>> b
b =4048000000000000
>> format short
>> c=a+16.7
c =65
>> c/3
ans =22
The original image file downloaded from the Web is in compressed jpeg format. But MATLAB imports it as a 2-dimensional array of 8-bit unsigned integers. Each array entry represents a single pixel in the image, with a value between 0 (black) and 255 (white). MATLAB recognized this as a grayscale image, but in some instances it will even treat black-and-white photographs as color images and import them as 3-dimensional arrays with the layers representing the red, green and blue components of the pixel.

```
>> whos bogart2
    Name      Size       Bytes  Class       Attributes
bogart2    270x220  59400     uint8
```
Processing the Image Data

Let’s do a little image distortion. If we divide each pixel value by 256 and multiply the result by 255, we do not get back the original pixel value, or anything very close to it.

Instead, by the somewhat peculiar way that MATLAB does integer arithmetic, each pixel value will be either 0 or 255, resulting in a monochrome image.

```
>> bogart3=255*(bogart2/256);
>> imshow(bogart3)
```
Monochrome Distortion