Today’s Lecture:

- Cell arrays
- File input/output

Announcements:

- Lab 3 exercise is very important for A2 and Test2. Make sure that you learn that material
- Test 1 (second run, optional) Wednesday 2:30pm in class
- Assignment 2 due next Thursday, 11/17, at 11:59pm
- Assignment 1b graded; resubmit by Friday, 11/11
Array vs. Cell Array

- **Simple array**
  - Each component stores **one scalar**. E.g., one char, one double, or one logical value
  - All components have the same type

- **Cell array**
  - Each cell can store something “bigger” than one scalar, e.g., a vector, a matrix, a string (vector of chars)
  - The cells may store items of different types
Lecture 18

1-d and 2-d examples ...

Vectors and matrices store values of the same type in all components.

Cell array: individual components may contain different types of data.

5 x 1 matrix

4 x 5 matrix

5 x 1 matrix

4 x 5 matrix

3 x 2 cell array
Cell Arrays of Strings

\[ C = \{ 'Alabama', 'New York', 'Utah' \} \]

\[ C \]

\[
\begin{array}{c}
'Alabama' \\
'New York' \\
'Utah'
\end{array}
\]

\[ C = \{ 'Alabama'; 'New York'; 'Utah' \} \]

\[ C \]

\[
\begin{array}{c}
'Alabama' \\
'New York' \\
'Utah'
\end{array}
\]

Contrast with 2-d array of characters

\[ M = [ 'Alabama'; ... 'New York'; ... 'Utah' ] \]

\[
\begin{array}{c}
'A' \\
'I' \\
'a' \\
'b' \\
'a' \\
'm' \\
'a' \\
'

'N' \\
'e' \\
'w' \\
'Y' \\
'o' \\
'r' \\
'k'
\end{array}
\]
Use braces \{ \} for creating and addressing cell arrays

Matrix

- Create

\[
\begin{bmatrix}
5 & 4 \\
1 & 2 \\
0 & 8
\end{bmatrix}
\]

- Addressing

\[
m(2,1) = \pi
\]

Cell Array

- Create

\[
\begin{bmatrix}
\text{ones}(2,2), 4 \\
'a b c', \text{ones}(3,1) \\
9, 'a c e l l'
\end{bmatrix}
\]

- Addressing

\[
C_{2,1} = 'A B C' \\
C_{3,2} = \pi \\
disp(C_{3,2})
\]
Creating cell arrays…

```matlab
C = {'Oct', 30, ones(3,2)};
```

is the same as

```matlab
C = cell(1,3); % not necessary
C{1}= 'Oct';
C{2}= 30;
C{3}= ones(3,2);
```

You can assign the empty cell array:  

```
D = {}
```
Example: Build a cell array of Roman numerals for 1 to 3999

\[
\begin{align*}
C\{1\} & = \text{‘I’} \\
C\{2\} & = \text{‘II’} \\
C\{3\} & = \text{‘III’} \\
\vdots & \\
C\{2007\} & = \text{‘MMVII’} \\
\vdots & \\
C\{3999\} & = \text{‘MMMXXMXXCIX’}
\end{align*}
\]
Example

1904 = 1*1000 + 9*100 + 0*10 + 4*1

= M CM IV

= MCMIV
Concatenate entries from these cell arrays!
function r = Ones2R(x)
% x is an integer that satisfies
% 0 <= x <= 9
% r is the Roman numeral with value x.

Ones = {'I', 'II', 'III', 'IV', ...
       'V', 'VI','VII', 'VIII', 'IX'};

if x==0
    r = '';    
else
    r = Ones{x};
end
function r = Ones2R(x)
% x is an integer that satisfies
% 0 <= x <= 9
% r is the Roman numeral with value x.

Ones = {'I', 'II', 'III', 'IV', ...
        'V', 'VI','VII', 'VIII', 'IX'};

if x==0
    r = ''; 
else
    r = Ones{x};
end
Similarly, we can implement these functions:

```matlab
function r = Tens2R(x)
% x is an integer that satisfies
%    0 <= x <= 9
% r is the Roman numeral with value 10*x.
```

```matlab
function r = Hund2R(x)
% x is an integer that satisfies
%    0 <= x <= 9
% r is the Roman numeral with value 100*x
```

```matlab
function r = Thou2R(x)
% x is an integer that satisfies
%    0 <= x <= 3
% r is the Roman numeral with value 1000*x
```
Now we can build the Roman numeral cell array for 1,…,3999

for a = 0:3
    for b = 0:9
        for c = 0:9
            for d = 0:9
                n = a*1000 + b*100 + c*10 + d;
                if n>0
                    C{n} = [Thou2R(a) Hund2R(b)…
                            Tens2R(c) Ones2R(d)];
                end
            end
        end
    end
end
end
Now we can build the Roman numeral cell array for 1,\ldots,3999

\begin{verbatim}
for a = 0:3   % possible values in thous place
  for b = 0:9 % values in hundreds place
    for c = 0:9 % values in tens place
      for d = 0:9 % values in ones place
        n = a*1000 + b*100 + c*10 + d;
        if n>0
          C{n} = [Thou2R(a)  Hund2R(b)...
                   Tens2R(c)  Ones2R(d)];
        end
      end
    end
  end
end
\end{verbatim}

The \(n\textsuperscript{th}\) component of cell array \(C\)

Four strings concatenated together
Example: subset of clicker IDs

IDs
['d091314'; ... 'h134d83'; ... 'h4567s2'; ... 'fr83209']

Find subset that begins with ‘h’

L = {}; k = 0;
for r=1:size(IDs,1)
  if IDs(r,1) == 'h'
    k = k+1;
    L{k} = IDs(r,:);
  end
end

Directly assign into a particular cell—good!

L = {}; k = 0;
for r=1:size(IDs,1)
  if IDs(r,1) == 'h'
    k = k+1;
    L = [L, IDs(r,:)];
  end
end

Concatenate cells or cell arrays—prone to problems!
A 3-step process to read data from a file or write data to a file

1. (Create and) open a file
2. Read data from or write data to the file
3. Close the file
Read the data in a file line-by-line and store the results in a cell array.

```
GATTTCGAG
GAGCCACTGGTC
ATAGATCCT
```

How are lines separated?
How do we know when there are no more lines?
In a file there are hidden “markers”

- Carriage return marks the end of a line
- `eof` marks the end of a file
Read data from a file

1. Open a file
2. Read it line-by-line until eof
3. Close the file
1. Open the file

```plaintext
fid = fopen('geneData.txt', 'r');
```

- An open file has a file ID, here stored in variable `fid`.
- Built-in function to open a file.
- Name of the file opened. `txt` and `dat` are common file name extensions for plain text files.
- `'r'` indicates that the file has been opened for reading.
2. Read each line and store it in cell array

```matlab
fid = fopen('geneData.txt', 'r');

k = 0;
while ~feof(fid)
    k = k + 1;
    Z{k} = fgetl(fid);
end
```

*False until end-of-file is reached*

*Get the next line*
3. Close the file

```matlab
fid = fopen('geneData.txt', 'r');

k = 0;
while ~feof(fid)
    k = k + 1;
    Z{k} = fgetl(fid);
end

fclose(fid);
```
function CA = file2cellArray(fname)
% fname is a string that names a .txt file
% in the current directory.
% CA is a cell array with CA{k} being the
% k-th line in the file.

fid= fopen([fname '.txt'], 'r');
k= 0;
while ~feof(fid)
    k= k+1;
    CA{k}= fgetl(fid);
end
fclose(fid);