While you are waiting

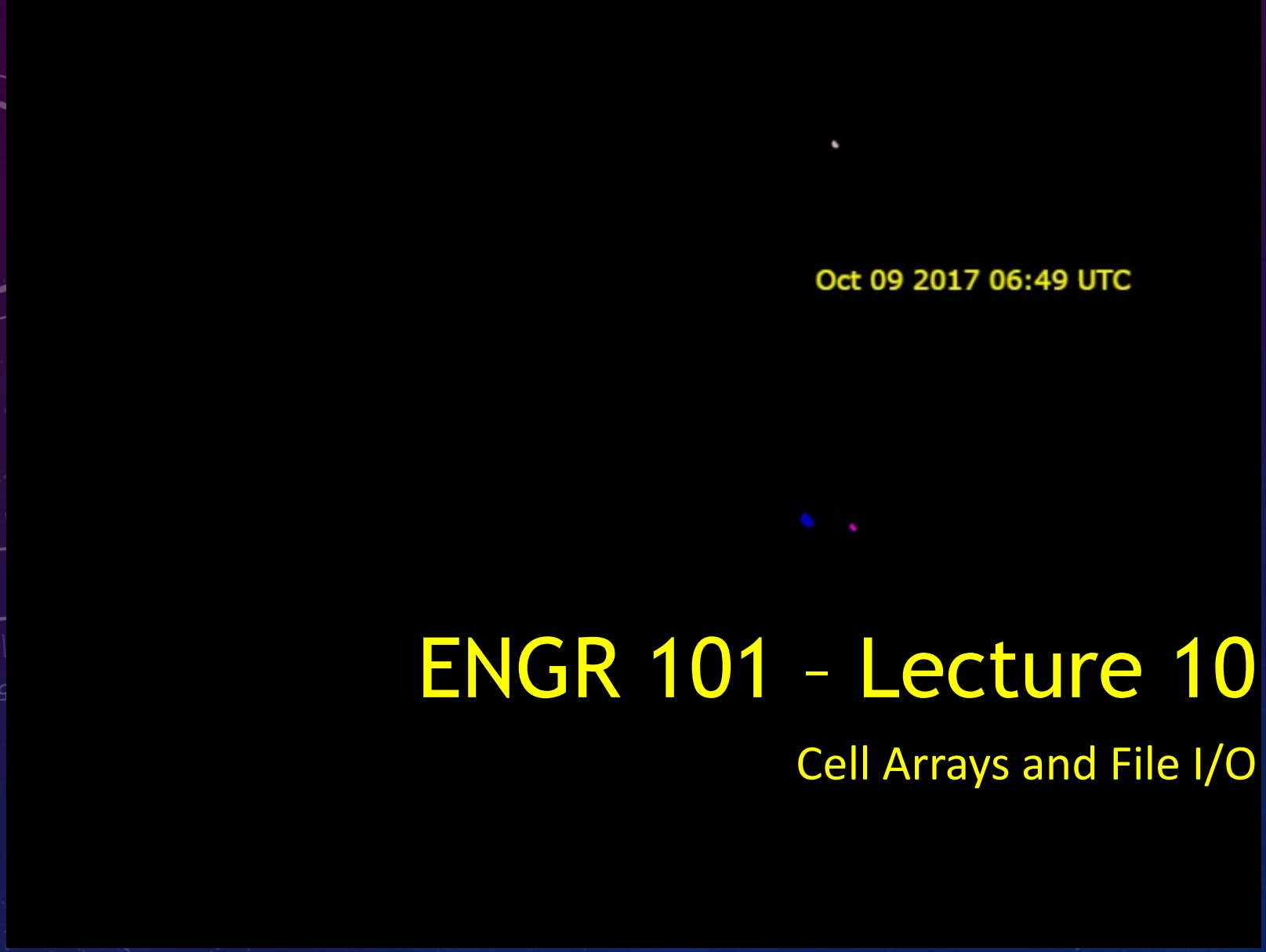
MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN ■ COLLEGE of ENGINEERING

Choose the correct spelling of this frequently misspelled word

resessitate

resusitate

resuscitate



ENGR 101 – Lecture 10

Cell Arrays and File I/O

Laura Alford, James Juett, Rick Niciejewski 10/9/17

Announcements

- No labs between Thursday, 12-OCT-2017 and Wednesday, 18-OCT-2017, inclusive – Fall Break

  - A couple of in-class exercises, plus a break today
  - Lecture and exercise data files in 00_Todays_Lecture on our Google Drive
  - Can stay for about an hour after lecture for office hours

Data Types

- MATLAB supports working with different **types** of data.
- Each type contains a different kind of data and supports a variety of different operations.
- Some examples of types we've seen so far:
  - double – A regular number.
  - uint8 – An integer between 0 and 255, inclusive.
  - char – A character (e.g. 'a')
  - logical – A true or false value, written as 0 or 1.

Scalars and Arrays

- A scalar is a single piece of data of a particular type.
- Arrays (vectors and matrices) contain several pieces of data, grouped together into a grid.

- Each array must be **homogenous** – it can only contain a single type of data, although it can contain as many pieces of data as we like.
- Arrays must also be "rectangular".
  (You can't have two rows in a matrix of different lengths.)

The whos function

- The whos function shows us the type of a variable, as well as its dimensions and storage requirement.

```
>> x = 1;
>> whos x
 Name        Size                Bytes  Class     Attributes
  x          1x1                     8  double

>> y = [1,2; 3,4];
>> whos y
 Name        Size                Bytes  Class     Attributes
  y          2x2                    32  double

>> word = ['h', 'e', 'l', 'l', 'o'];
>> whos word
 Name        Size                Bytes  Class     Attributes
  word       1x5                    10  char
```

Representing Strings in MATLAB

- A **string** is a sequence of characters (i.e. a "word").
- In MATLAB, a string is simply a vector of chars:

```
>> word1 = ['H','e','l','l','o']
word1 =
Hello
>> word2 = 'world'
word2 =
world
>> [word1,' ',word2]
ans =
Hello world
```

Instead of each character individually, we can also use a **string literal**.

Joining strings together is often called **concatenation**.

Storing a List of Strings

- Let's say we wanted to store several strings, for example, the names of countries in the election day example.
- One option is to use a matrix of characters.
  - Because the matrix must be rectangular, we have to pad with spaces.

'A'	'f'	'g'	'h'	'a'	'n'	'i'	's'	't'	'a'	'n'	' '	' '	' '
'A'	'l'	'b'	'a'	'n'	'i'	'a'	' '	' '	' '	' '	' '	' '	' '
'A'	'l'	'g'	'e'	'r'	'i'	'a'	' '	' '	' '	' '	' '	' '	' '
'A'	'm'	'e'	'r'	'i'	'c'	'a'	'n'	' '	'S'	'a'	'm'	'o'	'a'
'A'	'n'	'd'	'o'	'r'	'r'	'a'	' '	' '	' '	' '	' '	' '	' '

5x14 char

- One problem: If one string is a lot longer than the others, we have a lot of wasted space!

Storing a List of Strings

states

5x14 char

'A'	'f'	'g'	'h'	'a'	'n'	'i'	's'	't'	'a'	'n'	' '	' '	' '
'A'	'l'	'b'	'a'	'n'	'i'	'a'	' '	' '	' '	' '	' '	' '	' '
'A'	'l'	'g'	'e'	'r'	'i'	'a'	' '	' '	' '	' '	' '	' '	' '
'A'	'm'	'e'	'r'	'i'	'c'	'a'	'n'	' '	'S'	'a'	'm'	'o'	'a'
'A'	'n'	'd'	'o'	'r'	'r'	'a'	' '	' '	' '	' '	' '	' '	' '

- You need to remove the spaces before working with the string.

  For example:

```
>> [states(3,:), ' hello']
ans =
Algeria         hello

>> [deblank(states(3,:)), ' hello']
ans =
Algeria hello
```

The deblank function removes trailing spaces.

Cell Arrays

- In MATLAB, a cell array allows us to simulate a **heterogeneous** collection of elements.
- All elements in a cell array are of type "cell", but a cell may subsequently refer to any other type of data.
- Use the curly brackets { and } to create a cell array.

```
>> test = {1, 'hello', [1,2,3]}
test =
    [1]    'hello'    [1x3 double]

>> whos test
  Name       Size              Bytes  Class    Attributes
  test       1x3                 378  cell
```

Cell Arrays

- Why would you use a cell array ?
- One answer is creating a database, such as information on Driver License cards

Name	strings
Date of birth	integers, string
Address	integer, strings
Height	integers
Eye colour	string
Expiration Date	integers, string

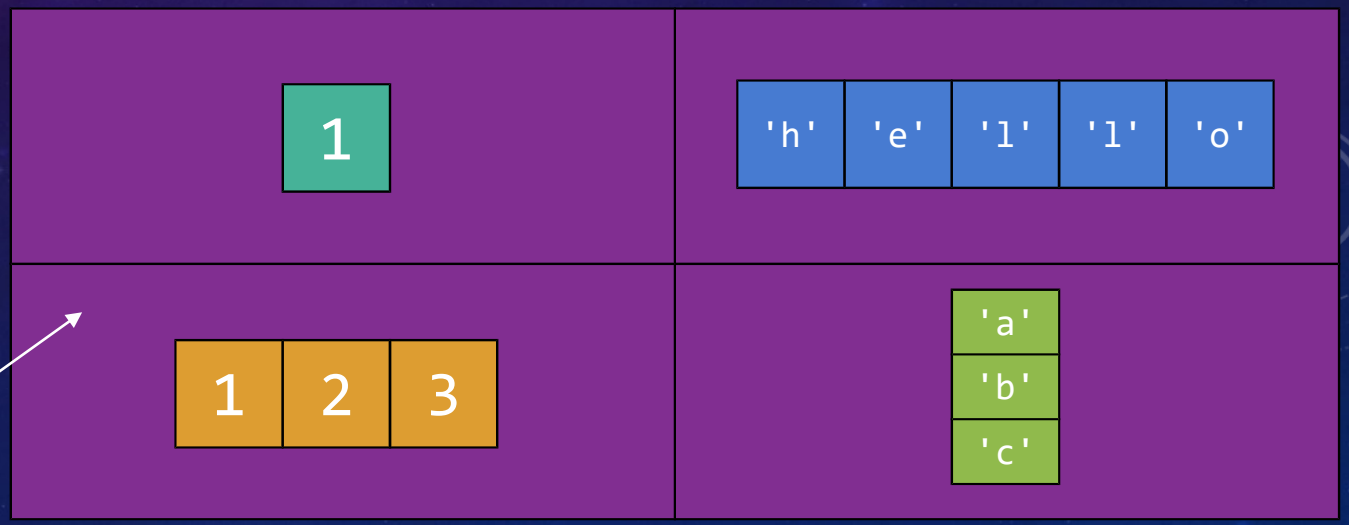
Cell Arrays

- The syntax for creating cell arrays is similar to normal arrays.

```
>> test = {1, 'hello'; [1,2,3], ['a';'b';'c']}
test =
    [          1]    'hello'
    [1x3 double]    [3x1 char]
```

For convenience, MATLAB will show the contents of cells containing strings (i.e. vectors of `char`).

MATLAB shows each cell in the cell array, as well as the type of data inside the cell.



The cell array is nice and "rectangular", but the contents of each cell can be any Data Type.

Indexing Into Cell Arrays

```
test = {1, 'hello'; [1,2,3], ['a';'b';'c']};
```

- There are two different ways to index into a cell array:
  - **Content Indexing** – uses { }
    Use this form if you want to select and work with the content of a cell. (This tends not to work well for groups of cells…)

```
>> test{2,1}
ans =
     1     2     3
```

A 1x3 double vector

  - **Cell Indexing** – uses ( )
    Use this form of indexing if you would like to select the cell itself. (Or a group of cells.)

```
>> test(2,1)
ans =
    [1x3 double]
```

A cell (which contains the vector)

Cell Arrays - Content Indexing

- Use the curly brackets { } for **content indexing**.
- You can use content indexing to read or write the element inside an individual cell.

```
>> test = {1, 'hello'; [1,2,3], ['a';'b';'c']}
test =
    [          1]    'hello'
    [1x3 double]    [3x1 char]

>> test{2,1} = 4
test =
    [1]    'hello'
    [4]    [3x1 char]

>> x = test{1,1}
x =
     1
```

Replace the vector in the selected cell with a 4 instead.

Read the contents of the cell at {1, 1} into a scalar.

Cell Arrays – Content Indexing

- Caution! Content indexing from a cell array to select multiple elements doesn't work like a regular array.
  - It will not yield a subarray !!!

```
>> test = {1, 'hello'; [1,2,3], ['a';'b';'c']}
test =
    [         1]    'hello'
    [1x3 double]    [3x1 char]

>> test{2,:} = 4
Expected one output from a curly brace or dot
indexing expression, but there were
2 results.

>> x = test{2,:}
x =
     1     2     3
```

Only the first element is displayed.

Cell Arrays - Cell Indexing

- Use parentheses ( ) for **cell indexing**.
- Use cell indexing to select or change the cells themselves.

```
>> test = {1, 'hello'; [1,2,3], ['a';'b';'c']}
test =
    [          1]    'hello'
    [1x3 double]    [3x1 char]

>> test(1,2) = {[1,2,3,4,5]}
test =
    [          1]    [1x5 double]
    [1x3 double]    [3x1 char  ]

>>  x = test(2,2)
x =
    [3x1 char]
```

Replace the entire cell with a new cell.

Extract the cell in row 2, column 2.

Cell Arrays – Cell Indexing

- Good news! Cell indexing to select multiple cells works pretty much like you would expect it to.
  - It simply yields a new cell array for those parts you select.

```
>> test = {1, 'hello'; [1,2,3], ['a';'b';'c']}
test =
    [          1]    'hello'
    [1x3 double]    [3x1 char]

>> test(2,:)
ans =
    [1x3 double]    [3x1 char]

>> x = test(1:end)
x =
    [1]    [1x3 double]    'hello'    [3x1 char]
```

IN-CLASS EXERCISE

Create three cell array variables that store people's names, verbs, and nouns. For example:

```
names = {'Harry', 'Xavier', 'Sue'};
verbs = {'loves', 'eats'};
nouns = {'baseballs', 'rocks', 'sushi'};
```

Write a test script that will initialize these cell arrays and then print sentences using one random element from each cell array (e.g., 'Xavier eats sushi').

Use `randi(2,1)` or `randi(3,1)` to choose the random element from the three cell arrays and then `display` and concatenation to construct the sentence.

SOLUTION

```
names = {'Harry','Xavier','Sue'};
verbs = {'loves','eats'};
nouns = {'baseballs','rocks','sushi'};
a = names{randi(3,1)};
b = verbs{randi(2,1)};
c = nouns{randi(3,1)};
v = [a ' ' b ' ' c];
disp(v)
```

A really sneaky use of Cell Arrays !!!

- Cell arrays allow for a heterogeneous collection of elements of different types.
  - It turns out this is not a common need.
  - Always prefer regular arrays if they will do the job.

- An example where cell arrays are useful is storing the names of states in 'election.mat'
  - State names are comprised of different length strings !!!

Storing Strings in Cell Arrays

- Because each cell in a cell array may contain a totally different type, we can store a list of strings like this:

```
>> states = cell(5,1);

>> states{1} = 'Afghanistan';
>> states{2} = 'Albania'
states =
    'Afghanistan'
    'Albania'
    []
    []
    []

>> [states{2}, ' hello']
ans =
Albania hello
```

Cell indexing gives us an empty 5x1 cell array.

Use content indexing to assign the 1x7 char vector 'Albania' into cell 2.

There are no extra spaces, because each char vector is exactly the right size for its string.

Converting Between Regular and Cell Arrays

- To create a cell array from a regular array of numbers:

```
C = num2cell(A)
```

- To create a cell array from a space-padded array of `char`:

```
C = cellstr(S)
```

- To create a regular array from a cell array of numbers:

```
A = cell2mat(C)
```

- To create a space-padded array of `char` from a cell array:

```
S = char(C)
```

Break Time

We'll start again in 5 minutes.

Assistance with Project #2

File I/O in MATLAB

- Today we'll cover the basics of a few functions for file input and output in MATLAB.
- Input
  - `csvread`
  - `xlsread`
- Output
  - `csvwrite`
  - `xlswrite`

The CSV Format

- CSV stands for "Comma Separated Values"
- These files contain one line for each row of data, with the entries of each row separated by commas.
- The file may contain headers and/or labels for the data.
- Files generally have the extension `.csv`

```
City,Population,Latitude,Longitude
Shanghai,24256800,31.20,121.50
Karachi,23500000,24.87,67.02
Beijing,21516000,39.90,116.40
Delhi,16349831,28.62,77.22
Lagos,16060303,6.45,3.40
...
```

cities.csv

csvread

```
City,Population,Latitude,Longitude
Shanghai,24256800,31.20,121.50
Karachi,23500000,24.87,67.02
Beijing,21516000,39.90,116.40
Delhi,16349831,28.62,77.22
Lagos,16060303,6.45,3.40
...
```

- The `csvread` function reads data from a CSV file into a matrix.
- You must always provide a parameter for the name of the file.
- You may optionally provide parameters to specify a subset of rows and columns you want to read.

```
cities = csvread('cities.csv', 1, 1);

whos cities
  Name          Size                Bytes  Class     Attributes

  cities       79x3                 1896  double
```

Start at row 2, column 2 to skip the headers and city names (which aren't numeric).

CAUTION
These numbers are 0-indexed, instead of 1-indexed!

TIP – default disp is format short

```
City,Population,Latitude,Longitude
Shanghai,24256800,31.20,121.50
Karachi,23500000,24.87,67.02
Beijing,21516000,39.90,116.40
Delhi,16349831,28.62,77.22
Lagos,16060303,6.45,3.40
...


>> csvread('cities.csv',1,1);
>> disp(ans(1,:))
   1.0e+07 *
    2.4257    0.0000    0.0000
>> format longg
>> disp(ans(1,:))
    24256800  31.2      121.5
```

cities.csv

csvwrite

- The `csvwrite` function writes a matrix to a CSV file.
- It only works with arrays containing numeric data.
- (It doesn't work with cell arrays.)

```
X = [1,2,3; 4,5,6; 7,8,9];

csvwrite('data.csv', X);
```

```
1,2,3
4,5,6
7,8,9
```

data.csv

xlsread

City	Population	Latitude	Longitude
Shanghai	24,256,800	31.20	121.50
Karachi	23,500,000	24.87	67.02
Beijing	21,516,000	39.90	116.40
Delhi	16,349,831	28.62	77.22
Lagos	16,060,303	6.45	3.40

- The `xlsread` function reads data from Microsoft Excel files, which generally have the `.xls` or `.xlsx` extension.
- Several optional parameters customize its behavior
  - See the documentation for full details.
- `xlsread` uses a compound return for numeric and text data.

```
[num, txt, raw] = xlsread('cities.xlsx','','','basic');
```

A regular array of only the numeric data.

A cell array containing only the text data.

A cell array containing all the data. (Cells allow us to combine numeric and text data.)

,'','','basic' options required if your computer does not have Excel

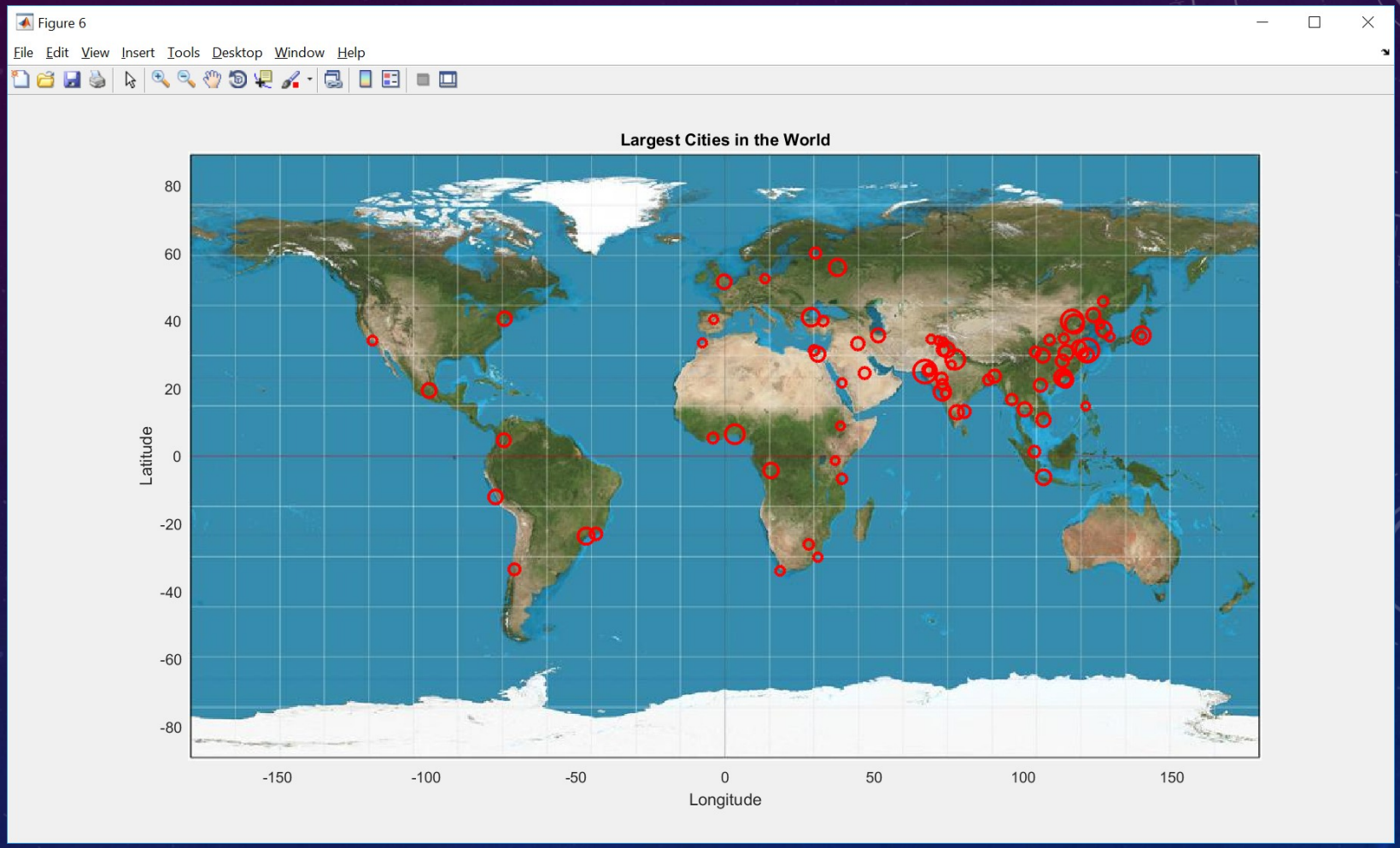
Exercise: Cities

- The data files `cities.csv` and `cities.xlsx` contain information on several dozen of the largest cities around the world.
  - Name
  - Population
  - Latitude
  - Longitude

Download these files from 00_Todays_Lecture
`cities.csv`
`cities.xlsx`
`world.jpg`

- We'll load this data into MATLAB and plot it.

Exercise: Cities (world.jpg is the *background*)



6 min

Your turn: Cities

- Use `csvread` to load the numeric data from `cities.csv`.
  - You'll need to specify an offset to skip the text data! [***use 1,1***]
  - For the instructions below, we assume you called the variable `data`.
- Create a figure and set the axes limits to [-180, 180] for x and [-90, 90] for y. (Use the `xlim` and `ylim` functions.)
- Make the following plots on the same figure:
  - `image([-180 180], [90 -90], imread('world.jpg'));`
  - `Values in image are [x1 x2] [y1 y2] corners [upperleft lowerright]`
  - `scatter(cities(:,3), cities(:,2), cities(:,1)./100000, 'red');`
- Add appropriate axis labels and a title.

Solution: Cities


```
cities = csvread('cities.csv', 1, 1);

figure();
xlim([-180,180]);
ylim([-90,90]);

hold on;
image([-180 180], [90 -90], imread('world.jpg'));
scatter(cities(:,3),cities(:,2),cities(:,1)/100000, 'red');

ylabel('Latitude');
xlabel('Longitude');
title('Largest Cities in the World');
```