CS 4: Lecture 27
Monday, May 1, 2006

MATLAB
======
Matlab is a computer language for supporting interactive numerical mathematics. The "Mat" in Matlab stands for matrix, and Matlab has strong support for matrix computations. It also has good support for plotting graphs, and it's for that purpose that we will use it with the Lunar Lander project.

Here are some of the differences between Matlab and Java.
- Matlab has much more support for high-level mathematical operations, like matrix multiplication. You could write (or find) libraries to do these operations in Java, but it's a lot more work.
- Matlab is interpreted (like Dr. Java), not compiled like Java. This makes it easy to experiment interactively.
- Matlab runs more slowly than Java, except for doing built-in matrix operations like finding eigenvalues (for which Matlab is usually faster).
- Matlab is expensive, whereas you can download Java for free.

Using Matlab

Because Matlab is interpreted, you can type expressions on the command line and get a result immediately, just like in Dr. Java.

```
>> 1 + 2              <- You type "1 + 2" after the prompt ">>"
ans =                 <- What Matlab replies
   3
```

You don't have to declare variables explicitly, nor choose their type. When you assign a value to a variable, the variable is created automatically if it didn't already exist.

```
>> a = 3.4
a =
   3.4
>> a = 6.6;           <- Semicolon suppresses Matlab's printed reply
```

You can create 1D or 2D arrays as follows.

```
>> arr1 = [ 1  2  3  4 ]
arr1 =
   1  2  3  4
>> arr2 = [ 1  2;  3  4;  5  6 ]
arr2 =
   1  2
   3  4
   5  6
```

These arrays are matrices too; that's how Matlab sees them. We won't be using matrix operations--just plotting--but if you know matrix index notation, it will be helpful when you want to look up an entry in a matrix.

```
>> arr2(1, 2)         <- Indexes row 1, column 2 of arr2.  Note (), not [].
ans =
   2
>> arr2(3, 1)
ans =
   5
```

Notice that the first row/column has index 1, not index 0 as in Java.

When you use a colon ":" as an index, it means "all values". So arr2(:, 1) picks out all the values in column 1, and arr2(2, :) picks out row 2.

```
>> arr2(:, 1)
ans =
   1
   3
   5
>> arr2(2, :)
ans =
   3  4
```

Matlab treats almost everything as a matrix. Even a scalar value like "3" is a 1x1 matrix.

If you want Matlab to forget all the variables you've created and start over with a blank slate, type "clear".

Plotting

For the Lunar Lander project, we're interested in Matlab mainly to plot functions of one variable. Suppose we want to plot the sine function.

Matlab's "plot" command takes two 1xn (or nx1) matrices. One represents a set of values from the domain, and one represents values from the range of the function. Suppose we want to evaluate sin t for values of t spaced 0.05 apart. Let's create a domain vector.

```
>> t = 0:0.05:2 * pi
t =
   0  0.0500  0.1000  0.1500  0.2000  0.2500  0.3000  ...  6.2000  6.2500
```

If we evaluate the sine of t, we get the sine of every value in the vector t.

```
>> sin(t)
ans =
   0  0.0500  0.0998  0.1494  0.1987  0.2474  0.2955  ... -0.0831 -0.0332
```

The plot command plots the curve defined by these points.

```
>> plot(t, sin(t))
```

(You'll have to try this yourself to see the plot this creates; I won't attempt to render it here.)

You can also add labels to the plot

```
>> xlabel('Time')     <- Matlab uses single quotes for text, not double
>> ylabel('sin t')
```

Getting Data from Java to Matlab

You'll need to take data generated by your Lunar Lander project and move it to Matlab for plotting. Here are two ways to do that.

(1) Cut and paste using Notepad (or another editor, if you prefer another).

- Have your Java program print out data in a two-column format.

```
0   3.8
1   4.65
2   5.33
3   6.1
...
```

- Select the text with the cursor. Right-click on it, and "copy" it.
- Run Notepad (or another text editor) and paste the text there.
- Save the text in a file.
- Read the file into Matlab and plot the data.

```
>> data = load('lunardata.txt');      <- Reads an nx2 matrix
>> plot(data(:, 1), data(:, 2));      <- Picks out columns 1 and 2
>> xlabel('Time');
>> ylabel('sin t');
```

By choosing "Save" from the "File" menu, you can save the figure in a variety of formats, including PDF, JPG, or EPS (encapsulated PostScript).

(2) Write a file directly from Java.

```
PrintWriter outFile;
try {
  FileWriter fw = new FileWriter("lunardata.txt");   // Name of file.
  outFile = new PrintWriter(fw);
} catch (IOException e) {
  System.out.println("IO Error " + e);
  System.exit(0);                                    // Stop program.
}

// Now print anything you want to the file, using outFile.println().
for (int i = 0; i < domain.length; i++) {
  outFile.println(domain[i] + " " + range[i]);
}

// When you're done, close the file.
outFile.close();
```

I won't say anything about what the FileWriter and PrintWriter classes are, but you can look them up in the online Java API (linked from the CS 4 Web page) if you want to learn more about doing file I/O. However, note that PrintWriters have println() and print() methods, just like System.out. The only difference is that they write to a file on disk, instead of the screen.

What are "try" and "catch"? Java has a mechanism called _exceptions_ designed to handle exceptional cases, like when you try to open a file but fail because the disk is full or defective. Most methods that can _throw_ exceptions--including the FileWriter constructor--need to be called inside a "try" clause. If the FileWriter constructor throws an IOException, the "catch" clause executes special-purpose code that reports the error and stops the program. You don't need to understand any of this, but you do need to copy the try/catch code if you want Java to write to a file.

Once the file is closed, you can import it into Matlab as described in part (1) above.