6.094
Introduction to programming in MATLAB

Lecture 4: Advanced Methods

Danilo Šćepanović

IAP 2010

Homework 3 Recap

- How long did it take?

- Common issues:
- The ODE file should be separate from the command that solves it. ie. you should not be calling ode45 from within your ODE file
- The structure of the output of an ode solver is to have time running down the columns, so each column of y is a variable, and the last row of y are the last values

- HW 4 was updated today, so download it again if you already started. Show a `juliaAnimation`
- Today is the last required class: make sure the sign-in sheet is accurate regarding your credit/listener status

Outline

(1) Probability and Statistics

(2) Data Structures

(3) Images and Animation

(4) Debugging

(5) Online Resources

Statistics

- Whenever analyzing data, you have to compute statistics

```
» scores = 100*rand(1,100);
```

- Built-in functions
  - mean, median, mode

- To group data into a histogram

```
» hist(scores,5:10:95);
```

  - makes a histogram with bins centered at 5, 15, 25…95

```
» N=histc(scores,0:10:100);
```

  - returns the number of occurrences between the specified bin *edges* 0 to <10, 10 to <20…90 to <100. you can plot these manually:

```
» bar(0:10:100,N,'r')
```

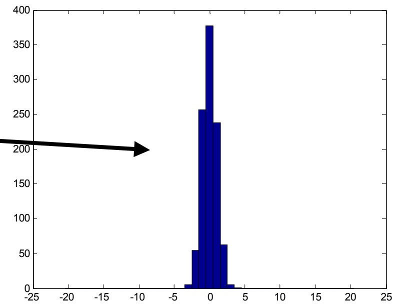
Random Numbers

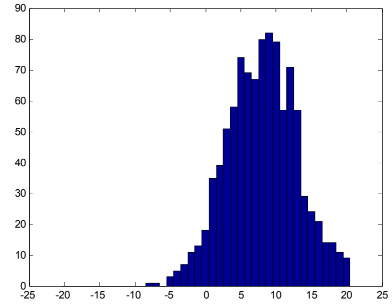
- Many probabilistic processes rely on random numbers
- MATLAB contains the common distributions built in
  - » **`rand`**
    - ➢ draws from the uniform distribution from 0 to 1
  - » **`randn`**
    - ➢ draws from the standard normal distribution (Gaussian)
  - » **`random`**
    - ➢ can give random numbers from many more distributions
    - ➢ see **doc random** for help
    - ➢ the docs also list other specific functions
- You can also seed the random number generators
  - »

```
rand('state',0); rand(1); rand(1);
rand('state',0); rand(1);
```

Changing Mean and Variance

- We can alter the given distributions
  - » `y=rand(1,100)*10+5;`
    - ➢ gives 100 uniformly distributed numbers between 5 and 15
  - » `y=floor(rand(1,100)*10+6);`
    - ➢ gives 100 uniformly distributed integers between 10 and 15. floor or ceil is better to use here than round
  - » `y=randn(1,1000)`
  - » `y2=y*5+8`
    - ➢ increases std to 5 and makes the mean 8





Exercise: Probability

- We will simulate Brownian motion in 1 dimension. Call the script 'brown'
- Make a 10,000 element vector of zeros
- Write a loop to keep track of the particle's position at each time
- Start at 0. To get the new position, pick a random number, and if it's <0.5, go left; if it's >0.5, go right. Store each new position in the k^{th} position in the vector
- Plot a 50 bin histogram of the positions.

Exercise: Probability

- We will simulate Brownian motion in 1 dimension. Call the script 'brown'
- Make a 10,000 element vector of zeros
- Write a loop to keep track of the particle's position at each time
- Start at 0. To get the new position, pick a random number, and if it's <0.5, go left; if it's >0.5, go right. Store each new position in the k^{th} position in the vector
- Plot a 50 bin histogram of the positions.

```
» x=zeros(10000,1);
» for n=2:10000
»     if rand<0.5
»         x(n)=x(n-1)-1;
»     else
»         x(n)=x(n-1)+1;
»     end
» end
» figure;
» hist(x,50);
```

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images and Animation
(4) Debugging
(5) Online Resources

Advanced Data Structures

- We have used 2D matrices
  - Can have n-dimensions
  - Every element must be the same type (ex. integers, doubles, characters…)
  - Matrices are space-efficient and convenient for calculation
  - Large matrices with many zeros can be made sparse:

```
» a=zeros(100); a(1,3)=10;a(21,5)=pi; b=sparse(a);
```

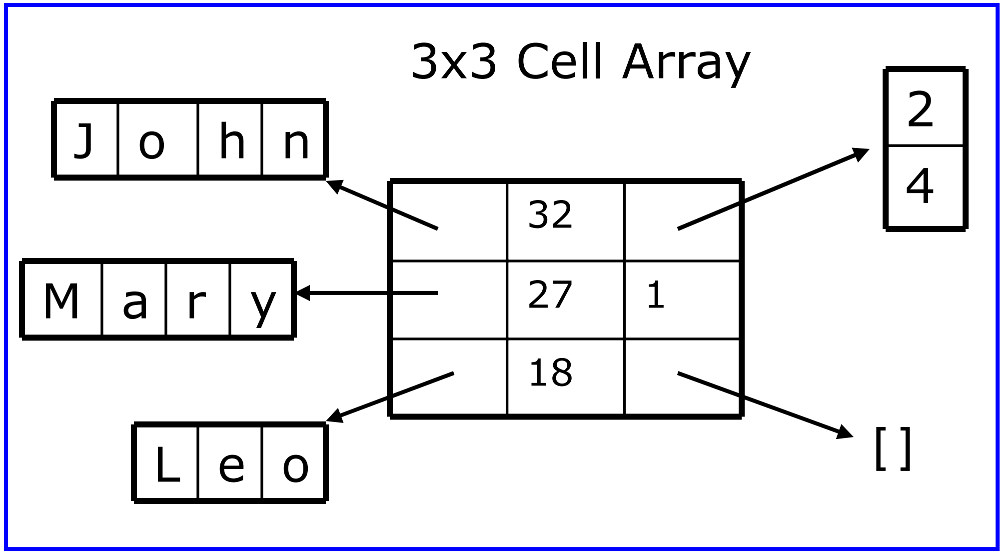
- Sometimes, more complex data structures are more appropriate
  - **Cell array**: it's like an array, but elements don't have to be the same type
  - **Structs**: can bundle variable names and values into one structure
    - Like object oriented programming in MATLAB

Cells: organization

- A cell is just like a matrix, but each field can contain anything (even other matrices):

3x3 Matrix

1.2	-3	5.5
-2.4	15	-10
7.8	-1.1	4



- One cell can contain people's names, ages, and the ages of their children
- To do the same with matrices, you would need 3 variables and padding

Cells: initialization

- To initialize a cell, specify the size
  - » `a=cell(3,10);`
    - ➢ a will be a cell with 3 rows and 10 columns
- or do it manually, with curly braces {}
  - » `c={'hello world',[1 5 6 2],rand(3,2)};`
    - ➢ c is a cell with 1 row and 3 columns
- Each element of a cell can be anything
- To access a cell element, use curly braces {}
  - » `a{1,1}=[1 3 4 -10];`
  - » `a{2,1}='hello world 2';`
  - » `a{1,2}=c{3};`

Structs

- Structs allow you to name and bundle relevant variables
  - Like C-structs, which are objects with fields
- To initialize an empty struct:

```
» s=struct([]);
```

  - size(s) will be 1x1
  - initialization is optional but is recommended when using large structs
- To add fields

```
» s.name = 'Jack Bauer';
» s.scores = [95 98 67];
» s.year = 'G3';
```

  - Fields can be anything: matrix, cell, even struct
  - Useful for keeping variables together
- For more information, see **doc struct**

Struct Arrays

- To initialize a struct array, give field, values pairs

```
» ppl=struct('name',{'John','Mary','Leo'},...
  'age',{32,27,18},'childAge',{[2;4],1,[]});
```

  - ➢ size(s2)=1x3
  - ➢ every cell must have the same size

```
» person=ppl(2);
```

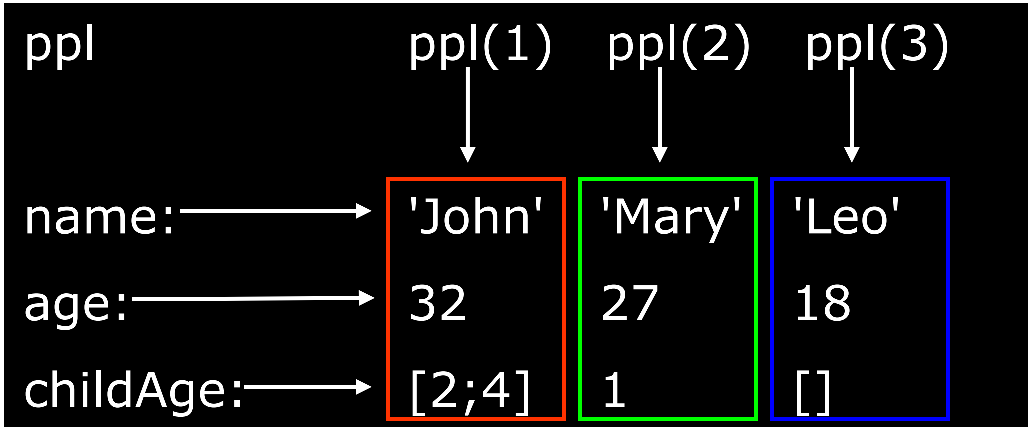
  - ➢ person is now a struct with fields name, age, children
  - ➢ the values of the fields are the second index into each cell

```
» person.name
```

  - ➢ returns 'Mary'

```
» ppl(1).age
```

  - ➢ returns 32



Structs: access

- To access 1x1 struct fields, give name of the field
  - » **`stu=s.name;`**
  - » **`scor=s.scores;`**
    - ➢ 1x1 structs are useful when passing many variables to a function. put them all in a struct, and pass the struct

- To access nx1 struct arrays, use indices
  - » **`person=ppl(2);`**
    - ➢ person is a struct with name, age, and child age
  - » **`personName=ppl(2).name;`**
    - ➢ personName is 'Mary'
  - » **`a=[ppl.age];`**
    - ➢ a is a 1x3 vector of the ages; this may not always work, the vectors must be able to be concatenated.

Exercise: Cells

- Write a script called `sentGen`
- Make a 3x2 cell, and put three **names** into the first column, and **adjectives** into the second column
- Pick two random integers (values 1 to 3)
- Display a sentence of the form '[name] is [adjective].'
- Run the script a few times

Exercise: Cells

- Write a script called `sentGen`
- Make a 3x2 cell, and put three **names** into the first column, and **adjectives** into the second column
- Pick two random integers (values 1 to 3)
- Display a sentence of the form '[name] is [adjective].'
- Run the script a few times

```
» c=cell(3,2);
» c{1,1}='John';c{2,1}='Mary-Sue';c{3,1}='Gomer';
» c{1,2}='smart';c{2,2}='blonde';c{3,2}='hot'
» r1=ceil(rand*3);r2=ceil(rand*3);
» disp([ c{r1,1}, ' is ', c{r2,2}, '.' ]);
```

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images and Animation
(4) Debugging
(5) Online Resources

Figure Handles

- Every graphics object has a handle
  - » `L=plot(1:10,rand(1,10));`
    - ➢ gets the handle for the plotted line
  - » `A=gca;`
    - ➢ gets the handle for the current axis
  - » `F=gcf;`
    - ➢ gets the handle for the current figure
- To see the current property values, use get
  - » `get(L);`
  - » `yVals=get(L,'YData');`
- To change the properties, use set
  - » `set(A,'FontName','Arial','XScale','log');`
  - » `set(L,'LineWidth',1.5,'Marker','*');`
- Everything you see in a figure is completely customizable through handles

Reading/Writing Images

- Images can be imported into matlab
  - » `im=imread('myPic.jpg');`

- MATLAB supports almost all image formats
  - ➢ jpeg, tiff, gif, bmp, png, hdf, pcx, xwd, ico, cur, ras, pbm, pgm, ppm
  - ➢ see **help imread** for a full list and details

- To write an image, give an rgb matrix or indices and colormap
  - » `imwrite(rand(300,300,3),'test1.jpg');`
  - » `imwrite(ceil(rand(200)*256),jet(256),...`
    `'test2.jpg');`
    - ➢ see **help imwrite** for more options

Animations

- MATLAB makes it easy to capture movie frames and play them back automatically

- The most common movie formats are:
  - avi
  - animated gif

- Avi
  - good when you have ‘natural’ frames with lots of colors and few clearly defined edges
- Animated gif
  - Good for making movies of plots or text where only a few colors exist (limited to 256) and there are well-defined lines

Making Animations

- Plot frame by frame, and **pause** in between

```
» close all
» for t=1:30
»     imagesc(rand(200));
»     colormap(gray);
»     pause(.5);
» end
```

- Can also use **drawnow** instead of **pause**
- When plotting lines or points, it's faster to change the `xdata` and `ydata` properties rather than plotting each time

```
» h=plot(1:10,1:10);
» set(h,'ydata',10:1);
```

Saving Animations as Movies

- A movie is a series of captured frames

```
» close all
» for n=1:30
»     imagesc(rand(200));
»     colormap(gray);
»     M(n)=getframe;
» end
```

- To play a movie in a figure window

```
» movie(M,2,30);
```

  - Loops the movie 2 times at 30 frames per second

- To save as an .avi file on your hard drive

```
» movie2avi(M,'testMovie.avi','FPS',30, ...
  'compression', 'cinepak');
```

- See **doc movie2avi** for more information

Making Animated GIFs

- You can use imwrite to save an animated GIF. Below is a trivial example

  » **`temp=ceil(rand(300,300,1,10)*256);`**

  » **`imwrite(temp,jet(256),'testGif.gif',...`**
  `'delaytime',0.1,'loopcount',100);`

- Alternatively, you can use **`getframe`**, **`frame2im`**, and **`rgb2ind`** to convert any plotted figure to an indexed image and then stack these indexed images into a 4-D matrix and pass them to **`imwrite`**. Read the doc on **`imwrite`** and these other functions to figure out how to do this.

Outline

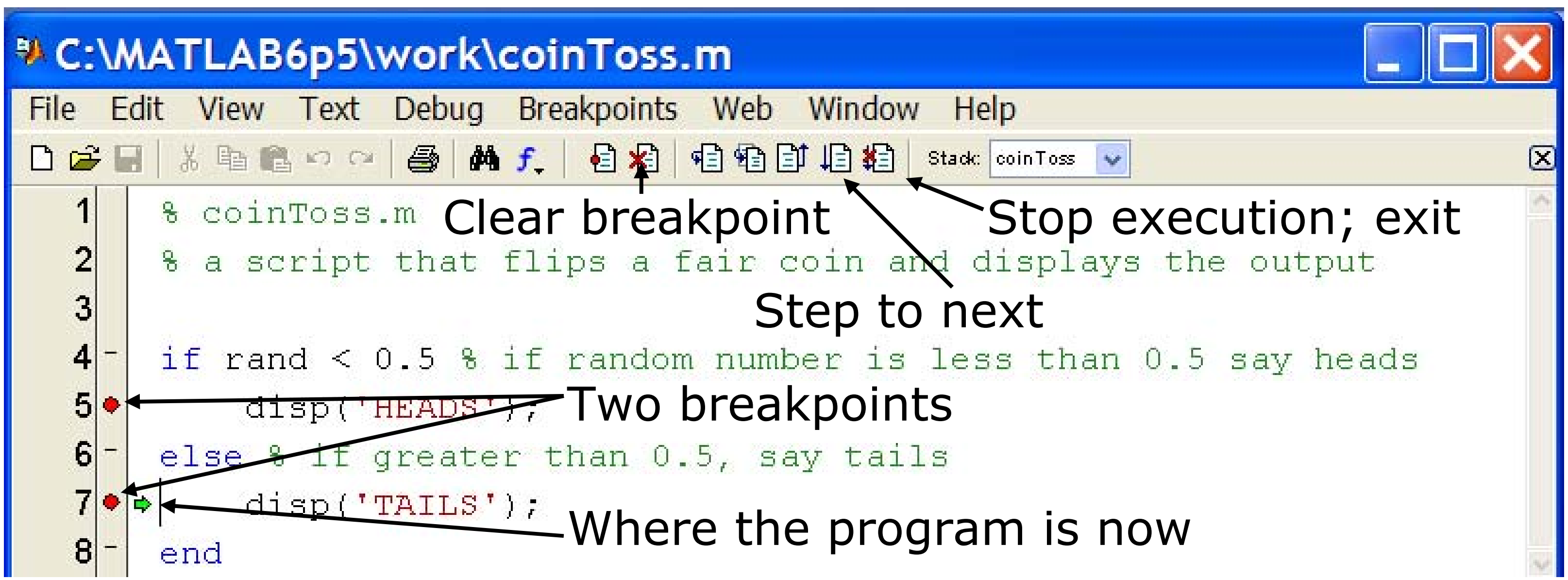
(1) Probability and Statistics
(2) Data Structures
(3) Images and Animation
(4) Debugging
(5) Online Resources

display

- When debugging functions, use **disp** to print messages
  - » `disp('starting loop')`
  - » `disp('loop is over')`
    - ➢ disp prints the given string to the command window
- It's also helpful to show variable values
  - » `disp(strcat(['loop iteration ',num2str(n)]));`
    - ➢ **strcat** concatenates the given strings
    - ➢ Sometimes it's easier to just remove some semicolons

Debugging

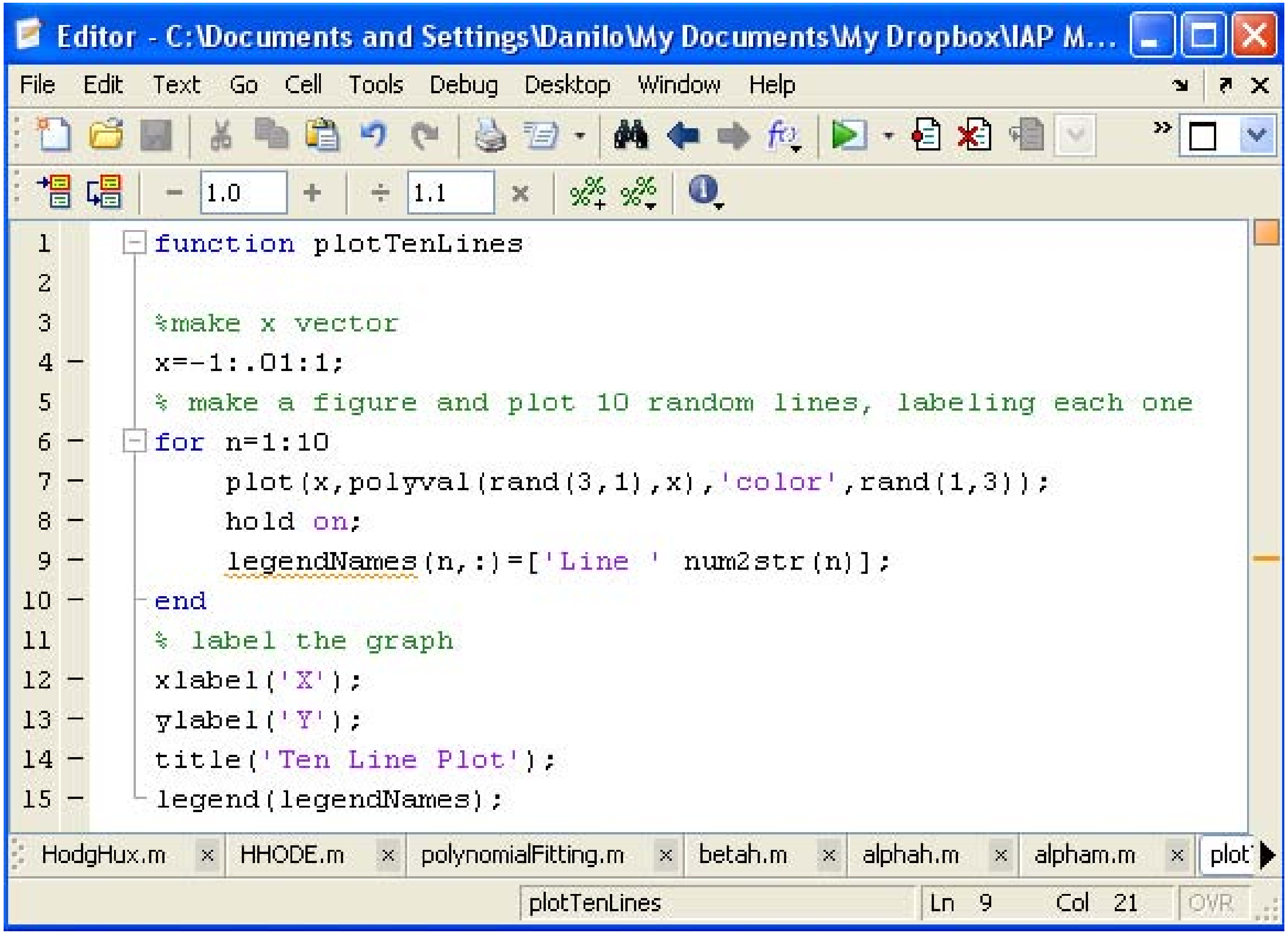
- To use the debugger, set breakpoints
    - Click on – next to line numbers in MATLAB files
    - Each red dot that appears is a breakpoint
    - Run the program
    - The program pauses when it reaches a breakpoint
    - Use the command window to probe variables
    - Use the debugging buttons to control debugger



Courtesy of The MathWorks, Inc. Used with permission.

Exercise: Debugging

- Use the debugger to fix the errors in the following code:




```
function plotTenLines

%make x vector
x=-1:.01:1;
% make a figure and plot 10 random lines, labeling each one
for n=1:10
    plot(x,polyval(rand(3,1),x),'color',rand(1,3));
    hold on;
    legendNames(n,:)=['Line ' num2str(n)];
end
% label the graph
xlabel('X');
ylabel('Y');
title('Ten Line Plot');
legend(legendNames);
```

Courtesy of The MathWorks, Inc. Used with permission.

Performance Measures

- It can be useful to know how long your code takes to run
  - ➢ To predict how long a loop will take
  - ➢ To pinpoint inefficient code

- You can time operations using **tic**/**toc**:

```
» tic
» CommandBlock1
» a=toc;
» CommandBlock2
» b=toc;
```

  - ➢ tic resets the timer
  - ➢ Each toc returns the current value in seconds
  - ➢ Can have multiple tocs per tic

Performance Measures

- For more complicated programs, use the profiler
  - » **profile on**
    - ➢ Turns on the profiler. Follow this with function calls
  - » **profile viewer**
    - ➢ Displays gui with stats on how long each subfunction took

Profile Summary

Generated 04-Jan-2006 09:53:26

Number of files called: 19

Filename	File Type	Calls	Total Time	Time Plot
newplot	M-function	1	0.802 s	
gcf	M-function	1	0.460 s	
newplot/ObserveAxesNextPlot	M-subfunction	1	0.291 s	
...matlab/graphics/private/clo	M-function	1	0.251 s	
allchild	M-function	1	0.100 s	
setdiff	M-function	1	0.050 s	

Courtesy of The MathWorks, Inc. Used with permission.

Outline

(1) Probability and Statistics

(2) Data Structures

(3) Images and Animation

(4) Debugging

(5) Online Resources

Central File Exchange

- The website – the MATLAB Central File Exchange!!
- Lots of people's code is there
- Tested and rated – use it to expand MATLAB's functionality

- http://www.mathworks.com/matlabcentral/

End of Lecture 4

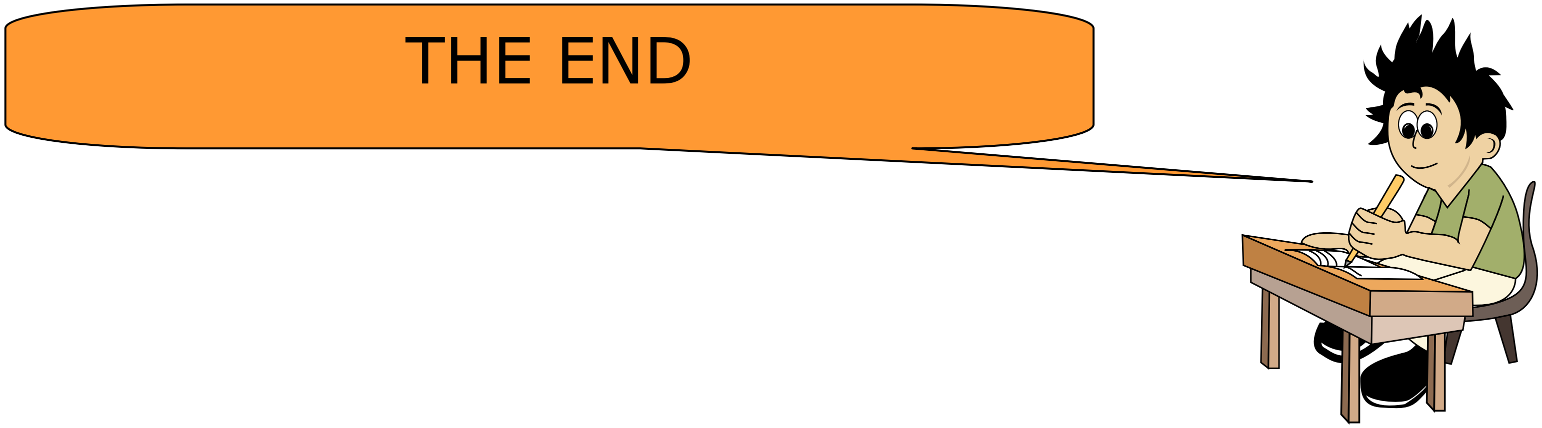
(1) Probability and Statistics

(2) Data Structures

(3) Images and Animation

(4) Debugging

(5) Online Resources



MIT OpenCourseWare
http://ocw.mit.edu

6.094 Introduction to MATLAB®
January (IAP) 2010

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.