6.094 Introduction to MATLAB®
January (IAP) 2009

6.094
Introduction to Programming in MATLAB®

Lecture 4: Advanced Methods

Sourav Dey
Danilo Šćepanović
Ankit Patel
Patrick Ho

IAP 2009

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images and Animation
(4) Debugging
(5) Symbolic Math
(6) Other Toolboxes

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.

Random Numbers

- Many probabilistic processes rely on random numbers

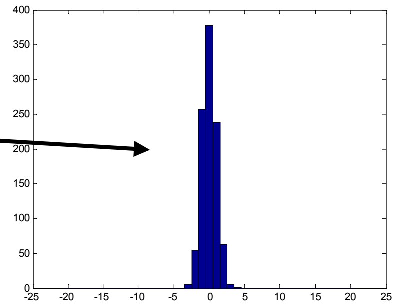
- MATLAB contains the common distributions built in
  » **rand**
    - draws from the uniform distribution from 0 to 1

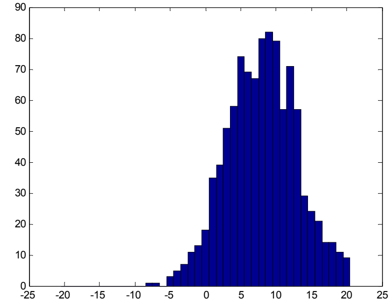
  » **randn**
    - draws from the standard normal distribution (Gaussian)

  » **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)**

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) Symbolic Math

(6) Other Toolboxes

Advanced Data Structures

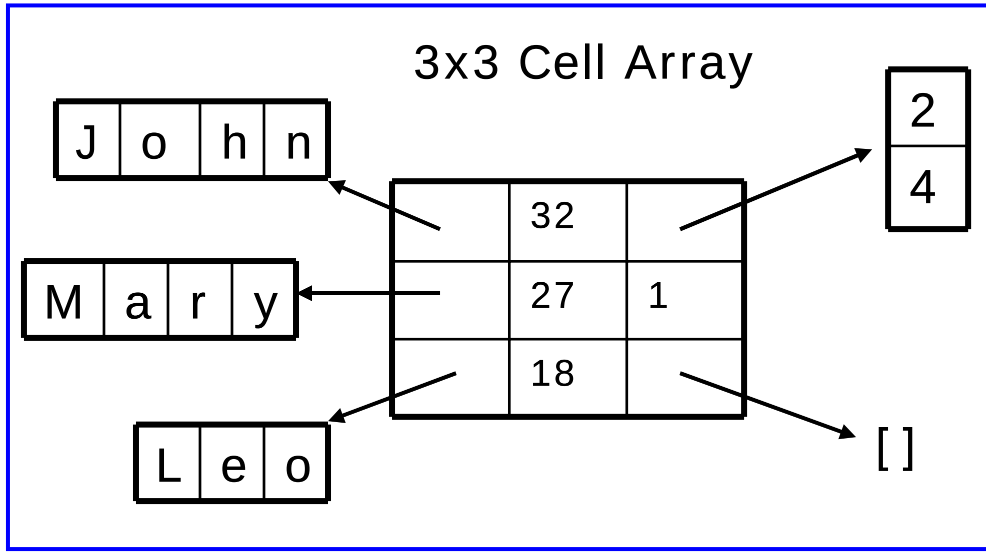
- We have used 2D matrices
  - Can have n-dimensions
  - Every element must be the same type (ex. integers, doubles, characters…)
  - Matrices are space-efficient and convenient for calculation
- Sometimes, more complex data structures are more appropriate
  - Cell array: it's like an array, but elements 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,[]});`

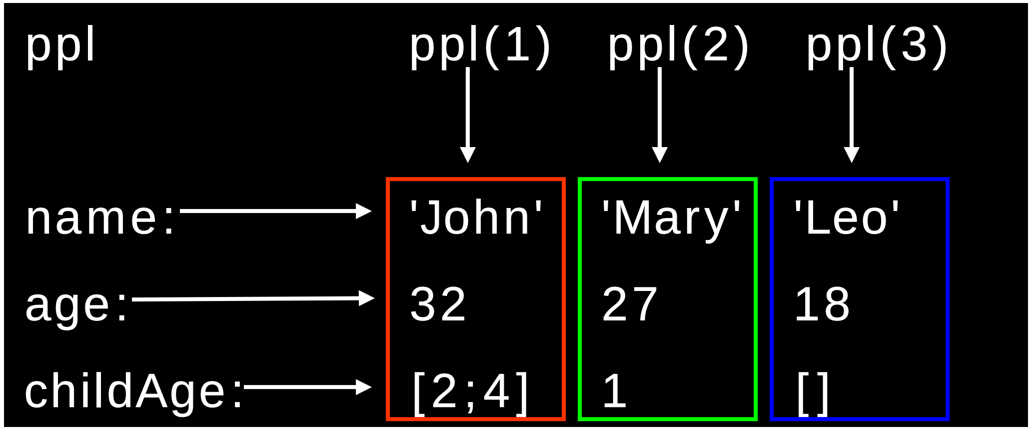
  - size(s2)=1x3
  - every cell must have the same size

  » `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'



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

Exercise: Cells

- Write a script called sentGen
- Make a 3x2 cell, and put people's 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 people's 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) Symbolic Math
(6) Other Toolboxes

Importing/Exporting 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(mat,jet(256),'test.jpg','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
```

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);**
- See book appendix or docs for more information

Handles

- Every graphics object has a handle
  - » **`h=plot(1:10,rand(1,10));`**
    - ➢ gets the handle for the plotted line
  - » **`h2=gca;`**
    - ➢ gets the handle for the current axis
  - » **`h3=gcf;`**
    - ➢ gets the handle for the current figure
- To see the current property values, use get
  - » **`get(h);`**
  - » **`yVals=get(h,'YData');`**
- To change the properties, use set
  - » **`set(h2,'FontName','Arial','XScale','log');`**
  - » **`set(h,'LineWidth',1.5,'Marker','*');`**
- Everything you see in a figure is completely customizable through handles

Outline

(1) Probability and Statistics

(2) Data Structures

(3) Images and Animation

(4) Debugging

(5) Symbolic Math

(6) Other Toolboxes

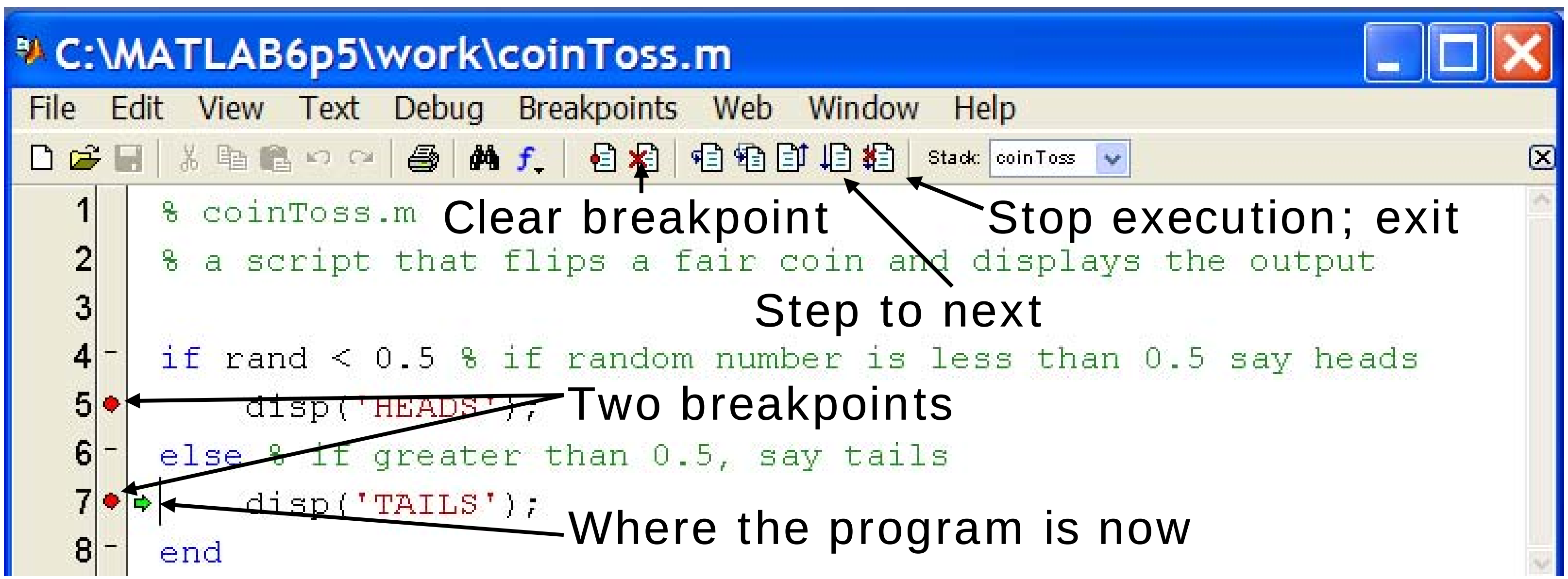
display

- 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

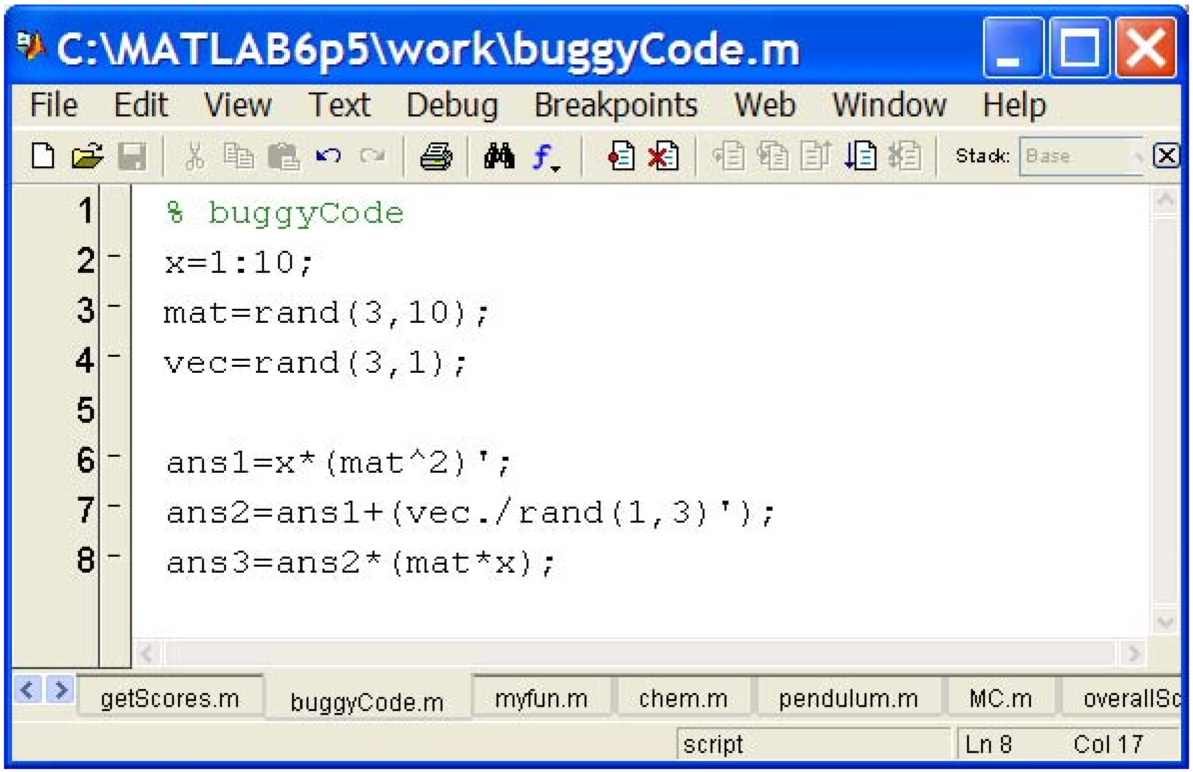
- To use the debugger, set breakpoints
  - Click on – next to line numbers in MATLAB files
  - Each red dot that appears is a breakpoint
  - Run the program
  - The program pauses when it reaches a breakpoint
  - Use the command window to probe variables
  - Use the debugging buttons to control debugger



Courtesy of The MathWorks, Inc. Used with permission.

Exercise: Debugging

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



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	

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images and Animation
(4) Debugging
(5) Symbolic Math
(6) Other Toolboxes

What are Toolboxes?

- Toolboxes contain functions specific to a particular field
  - ➢ for example: signal processing, statistics, optimization

- It's generally more efficient to use MATLAB's toolboxes rather than redefining the functions yourself
  - ➢ saves coding/debugging time
  - ➢ some functions are compiled, so they run faster
  - ➢ HOWEVER there may be mistakes in MATLAB's functions and there may also be surprises

- MATLAB on Athena contains all the toolboxes

- Here are a few particularly useful ones for EECS...

Symbolic Toolbox

- Don’t do nasty calculations by hand!
- Symbolics vs. Numerics

	Advantages	Disadvantages
Symbolic	•Analytical solutions •Lets you intuit things about solution form	•Sometimes can't be solved •Can be overly complicated
Numeric	•Always get a solution •Can make solutions accurate •Easy to code	•Hard to extract a deeper understanding •Num. methods sometimes fail •Can take a while to compute

Symbolic Variables

- Symbolic variables are a type, like double or char

- To make symbolic variables, use sym
  - » **`a=sym('1/3');`**
  - » **`b=sym('4/5');`**
    - ➢ fractions remain as fractions
  - » **`c=sym('c','positive');`**
    - ➢ can add tags to narrow down scope
    - ➢ see **`help sym`** for a list of tags

- Or use syms
  - » **`syms x y real`**
    - ➢ shorthand for x=sym('x','real'); y=sym('y','real');

Symbolic Expressions

- Multiply, add, divide expressions
  - » **d=a*b**
    - ➢ does 1/3*4/5=4/15;

```
ans =
4/15
```

  - » **expand((a-c)^2);**
    - ➢ multiplies out

```
ans =
1/9-2/3*c+c^2
```

  - » **factor(ans)**
    - ➢ factors the expression

```
ans =
1/9*(3*c-1)^2
```

Cleaning up Symbolic Statements

» **pretty(ans)**

- makes it look nicer

```
                2
1/9 - 2/3 c + c
```

» **collect(3*x+4*y-1/3*x^2-x+3/2*y)**

- collects terms

```
ans =
2*x+11/2*y-1/3*x^2
```

» **simplify(cos(x)^2+sin(x)^2)**

- simplifies expressions

```
ans =
1
```

» **subs('c^2',c,5)**

- Replaces variables with numbers or expressions

```
ans=
25
```

» **subs('c^2',c,x/7)**

```
ans=
1/49*x^2
```

More Symbolic Operations

- We can do symbolics with matrices too
  - » **mat=sym('[a b;c d]');**
  - » **mat2=mat*[1 3;4 -2];**
    - ➢ compute the product
  - » **d=det(mat)**
    - ➢ compute the determinant
  - » **i=inv(mat)**
    - ➢ find the inverse

```
mat2 =
[    a+4*b, 3*a-2*b]
[    c+4*d, 3*c-2*d]
```

```
d =
a*d-b*c
```

```
i =
[  d/(a*d-b*c), -b/(a*d-b*c)]
[ -c/(a*d-b*c),  a/(a*d-b*c)]
```

- You can access symbolic matrix elements as before
  - » **i(1,2)**

```
ans =
-b/(a*d-b*c)
```

Exercise: Symbolics

- The equation of a circle of radius r centered at (a,b) is given by: $(x-a)^2 + (y-b)^2 = r^2$.
- Expand this equation into the form $Ax^2 + Bx+Cxy + Dy + Ey^2 = F$ and find the expression for the coefficients in terms of a,b, and r.

Exercise: Symbolics

- The equation of a circle of radius r centered at (a,b) is given by: (x-a)^2 + (y-b)^2 = r^2.
- Expand this equation into the form Ax^2 + Bx+Cxy + Dy + Ey^2 = F and find the expression for the coefficients in terms of a,b, and r.

```
» syms a b r x y
» pretty(expand((x-a).^2 + (y-b).^2))
```

Outline

(1) Probability and Statistics
(2) Data Structures
(3) Images and Animation
(4) Debugging
(5) Symbolic Math
(6) Other Toolboxes

Signal Processing Toolbox

- MATLAB is often used for signal processing (fft)
- What you can do:
  - filter design
  - statistical signal processing
  - Laplace transforms

- Related Toolboxes
  - Communications
  - Wavelets
  - RF
  - Image Processing

Control System Toolbox

- The control systems toolbox contains functions helpful for analyzing systems with feedback

- Simulation of LTI system function
- Discrete time or continuous time
- You will be exposed to it in 6.003
- Can easily study step response, etc. modal analysis.

- Related toolboxes:
  - System Identification
  - Robust Control – modern control theory
  - Model Predictive Control

Statistics Toolbox

- For hardcore statistics and data-analysis
  - Principal component analysis
  - Independent component analysis
  - Tests of significance (chi squared, t-tests...)

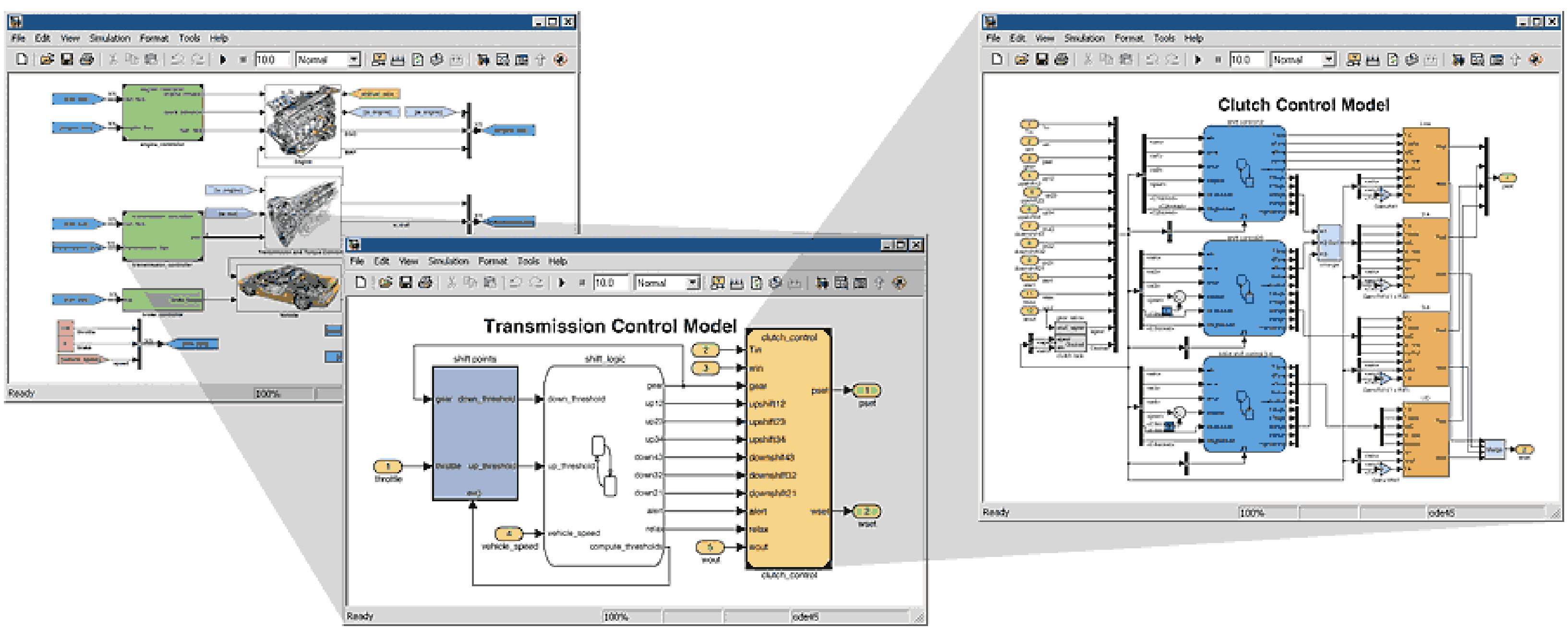
- Related Toolboxes
  - Spline – for fitting
  - Bioinformatics
  - Neural Networks

Optimization Toolbox

- For more hardcore optimization problems – that occur in OR, business, engineering
  - linear programming
  - interior point methods
  - quadratic methods

SIMULINK

- Interactive graphical environment
- Block diagram based MATLAB add-on environment
- Design, simulate, implement, and test control, signal processing, communications, and other time-varying systems



Courtesy of The MathWorks, Inc. Used with permission.

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/

MATLAB Final Exam

- Brownian Motion stop-animation – integrating loops, randomization, visualization

- Make a function brown2d(numPts), where numPts is the number of points that will be doing Brownian motion
- Plot the position in (x,y) space of each point (start initially at 0,0). Set the x and y limits so they're consistent.
- After each timestep, move each x and y coordinate by randn*.1
- Pause by 0.001 between frames
- Turn on the DoubleBuffer property to remove flicker
  » `set(gcf,'DoubleBuffer','on');`

- Ask us for help if needed!

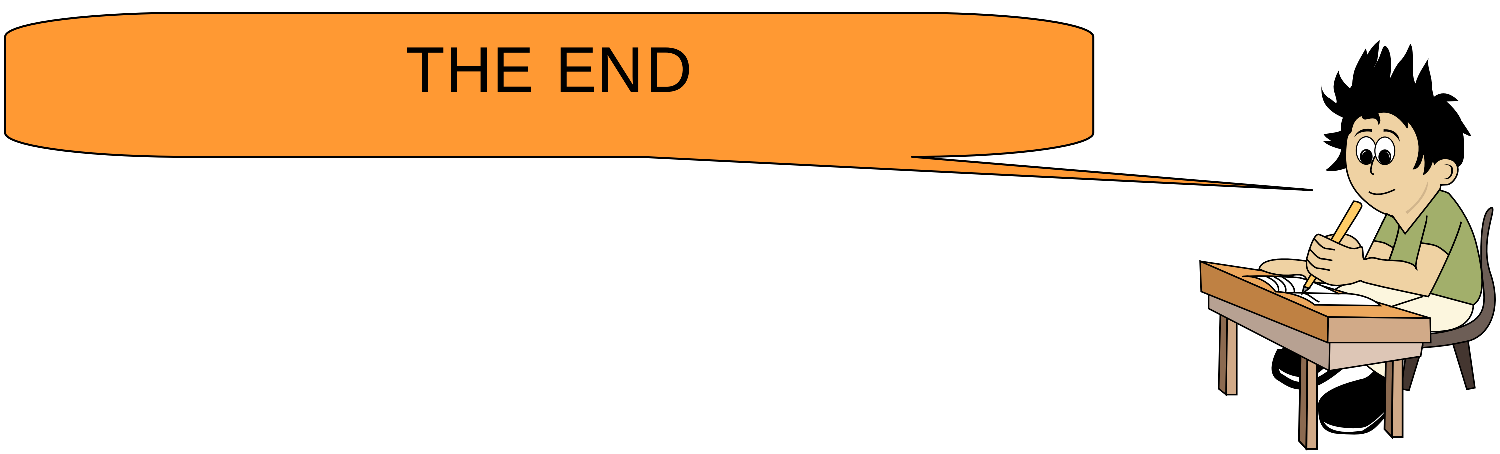
End of Lecture 4

(1) Data Structures

(2) Symbolics

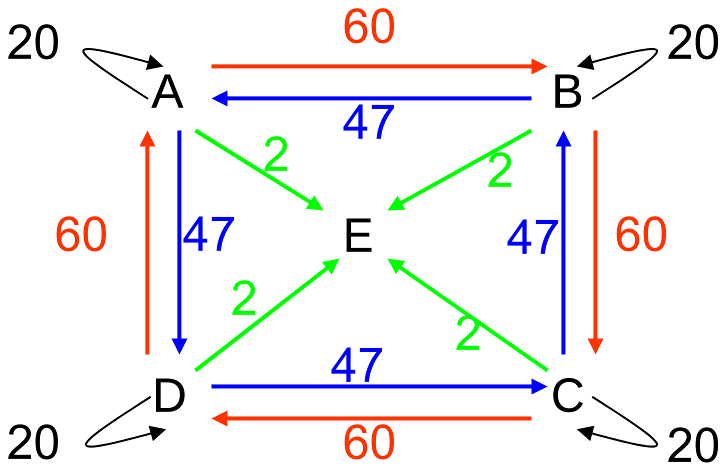
(3) Probability

(4) Toolboxes



Monte-Carlo Simulation

- A simple way to model complex stochastic systems
- Use random numbers to control state changes



- This system represents a complex reaction
- The numbers by the arrows show the propensity of the system to go from one state to another
- If you start with 1 molecule of A, how does the system behave with time?

Example: Monte-Carlo

- This MATLAB file will track the behavior of the molecule

```
function s=MC(timeSteps)
state=0; % start off in A
% B=1, C=2, D=3, E=4 to make things easy
s=zeros(timeSteps,1); % to store current state
borders=[20 60 47 2]/sum([20 60 47 2]);
outcome=cumsum(borders);
direction=[0 1 -1];
for n=1:timeSteps
    s(n)=state;
    if state~=4 %if not in E
        C=min(find(rand<outcome));
        if C==4 % enter E
            state=4;
        else % go right or left
            state=mod(state+direction(C),4);
        end
    end
end
```

Courtesy of The MathWorks, Inc. Used with permission.

Example: Monte-Carlo

- We can run the code 1000 times to simulate 1000 molecules

```
» s=zeros(200,5);
» for n=1:1000
»   st=MC(200);
»   for state=0:4
»       s(:,state+1)= s(:,state+1)+(st==state);
»   end
» end
```