M=[1,2,3;4,5,6] Creates array

M(1,2) = 2 - Row 1 column 2 holds 2

V=[5,4,3,4,7,6] - Vector is one row or one column array

V(1)=5 - The number in the parentheses specifies what element of the array

vc=V' - Apostrophe makes the vector into the opposite organization (from row vector to column vector, or vice versa)

vc =

3

4

5

6

arrayA(:, 2) - Display all values in the 2nd column, all rows - colon ':' function means all values

and it's located in either the row or column value slot

arrayA(3, :) - Display all values in the 3rd row, all columns

arrayA(3, 1) = 0 - Sets the specified element (Row 3 column 1) to 0--can set it equal to almost anything else

arrayB=arrayA(2:3, 1:2) - Only Rows 2 to 3 and Columns 1 to 2 of array A are selected for array B

------------------------------------------------------------------

u=3\*V - Multiplies each vector component by 3 ---Division, Multiplication, Subtraction, Addition of Vectors (w=u+v)

w=u+v - Only add vectors of same size

--Period here is an element by element function

w=u.\*v - Multiplies vector element by element

w=u./v - Divides vector element by element

3./v - Divides 3 by each element of the vector

2.^v - Raises 2 to the power of each element of the vector

u=v1.\*v2 - v1 and v2 must be the same size ---Division

--Not element by element

v1\*v2 - Regular matrix multiplication

--Outer/Inner product

v=[1; 2; 3];

w=[4; 5; 6];

y=v'\*w - Inner product (a scalar value, which is 32 here) Row vector \* column vector

y=v\*w' - Outer product (matrix)

y=[u, v] - Combines list of u and v. Starting with u. (Vice versa if it was y=[v,u])

---------------------------------------------------------------------------------------------------------------------------

w=zeros(1,5) - Creates a 1 row 5 column vector array of just zeros

=ones(

x=linspace(0, 2, 12) - Creates an array of 12 columns that have equally spaced vaues between 0 and 2

u=[4, 5, 6]

length(u)=3 - Gives number of values in array; also used for number of characters in a string

min(u) = 4 - Gives minimum number in element

max(u) = 6 - Max

sum(u) = 15 - Sum

norm(u) - Square root of the sum of the squares of elements in u (Like calculating magnitude of vector)

cross(a,b) - Cross product of vectors a and b

dot(a,b) - Dot product of vectors a and b

rand(nr, nc) - Array with 'nr' rows and 'nc' columns populated with random values between 0 and 1

randi([a,b],nr,nc) - Array with 'nr' rows and 'nc' columns populated with random INTEGERS between [a,b] inclusive

randi(a,nr,nc) - Array with such rows with random integers between 1 and 'a' inclusive

rng('shuffle') - Resets random number generator

Use '' for apostrophe in a string ( Just ' will end the string)

Use == for equal to (if ToBikeorNot == 1) ( = means to put something into a variable)

input(PROMPT, 's') - \*\*No brackets

- program will now convert answer to a string (letters and numbers)

EDIT Window: Click on the dash next to number (dash represents the existence of a function)

Then press Run

-Now, you can check to see if the program works step by step

-The program will start checking from where you created the flag (red circle)

To Fix Indenting: Select the area of interest in the edit window and press CRTL + I

- or Editor-->Indent-->Smart

disp([ ]) -Certain commands need the brackets inside parentheses for MULTIPLE strings and commands

title([ ])

fortune{1}(1:3) -the ':' sign means from what number to the other. Can be used outside of for loops.

- Output: Pur

sort() - sorts vector into ascending or descending order

-for default input: sort(vector) - function sorts the vector into ASCENDING order

MATLAB interprets 0 as false and a 1 (or any other nonzero number) as true

MATLAB interprets expressions "true" and false" as representing logical values

T=95

isTooHot = T>90 - Stores value 1, because it's true (stores value 0 if false)

isTooHot = true - Stores value 1

isTooHot = false - Stores value 0

(a>0) && (a<=100) - a is greater than zero AND a is less than or equal to 100

year ~= 1960 - year is NOT EQUAL to 1960

~ - means NOT

(Tc>Tmin) || (alpha>alphaMax) - Tc is greater than Tmin OR alpha is greater than alphaMax

----Comparing Strings-----

if strcmp(str1, str2) - Checks to see if two strings match each other:

-If matching, then it stores value 1

-If not matching, then it stores value 0

if strcmp(str1, str2)==false - Can do this to have the if statement work when strcmp is false

-Can also do ' if strcmp(str1, str2) == 0 '

- '==1' for substitution of 'true'

strcmpi('Hello', 'heLLo') - Ignores the case of letters

-Output is true here

----Switch Statements---------

switch <switchExpression> -switchExpression can be a variable with a value assigned to it

-Ex: switch iDoor (where iDoor=randi(4))

case <case1\_exp> case 1

<statement block1> disp('It''s a new car!');

case <case2\_exp> case {2, 3} - the integer 2 OR 3 will BOTH match up with this case

<statement block2> disp('It''s a picture of someone else''s car!');

case <case3\_exp> ...

<statement block3>

...

otherwise otherwise

<default statement block> disp('Something has gone wrong.')

end end

-If switchExpression==caseN\_exp then <statement blockN> is executed.

Otherwise <default statement block> is executed

-Works with strings or numbers

Plot used for vectors of equal-length

x=[1,4,6];

y=[0,1,3];

plot(x,y) - Plots the vectors together as x-y coordinate system

plot(x,y,'color/marker/type of line)

plot(x,y,'o-') - Marks the points on the graphical representation (d - diamond marker, etc..)

........ 'ro-') - Graphed line becomes red (g - green, b - blue, etc..)

........ 'r--') - Makes the line dashed

axis([0, 7, 0, 4)]) - X-axis now goes from 0 to 7 and Y-axis goes from 0 to 4 (RESETS when vectors are regraphed)

- 'inf' means infinity

grid on - Turns on grid (RESETS when vectors are regraphed)

grid off

help plot - A bunch of information about colors, markers, and types of line available

doc plot - Even more information that opens up in a separate document

----Two ways to layer graphs

plot(x1, y1, ' o', x2, y2, '-') - Two graphs at once

plot(x1, y1, 'o') - Freezes the graph and then (in the command after 'hold on')

hold on adds in another axis

plot(x2, y2, '-')

hold off

------------------------------------------------------------------------------------------------------------------------------

--PlotQuadratic outline saved

xlabel('x (m)'); - labels X-axis 'x (m)'

ylabel('f') - labels Y-axis 'f'

title('Quadratic function f=x^2'); - Titles the graph with the string mentioned

title(['<text>', '<text>', num2str...]) - Need the BRACKETS to include multiple strings and commands

-Like disp([ ])

----

Click on Figure. Go to Edit. Copy it.

----

Double click on arrays in Workspace to open a table that contains the array

----

Right-click on Workspace and go to Choose Columns and select size and whatnot

---FOR loops----

Repeatedly execute a block of commands

---------------

for i=8:-3:1 - counter goes from 8 and towards 1 by subtracting 3 each time

Ex: "i" will equal 8, 5, and 2

--The loop variable: icount---

iMax=5;

for icount=1:iMax

disp(['Count is: ', num2str(icount)]);

end

-icount is created by the for statement

-initially set to 1

-Executes the statement block underneath

-Then incremented by 1 until the value of 'iMax' is reached

icount=1:2:iMax - icount will increment by two each time through the loop (will still stop before rather than surpass iMax)

----archery.m-----

deltat = 0.02;

Nmax=50; -decreasing 'Nmax' shortens the arrow's path

for kb=1:Nmax

clc

disp([blanks(kb), '>>--->']) -'blanks(kb)' shifts the arrow 'kb' blanks

pause(deltat)

end

clc

disp([blanks(Nmax), '>>---<'])

pause(deltat)

clc

disp([blanks(Nmax), '>>-<-'])

pause(deltat)

......

for kb=1:Nmax

clc

disp([blanks(Nmax-kb), '<---<<'])

pause(deltat)

end

-----Vocab and Good Programming Practice---

"loop unrolling" means to write out the loop function in its equivalent number of independent statements

"loop rolling" means to condense a series of independent statements into a loop statement

Never change the value of the LOOP INDEX VARIABLE in the loop - Don't change value of icount within the loop

Use loop index variables names that begin with i, j, k, m, n to indicate integers

Choose loop index variable names that are related to the quantity being changed in the loop

-Ex: a loop over time

for it=1:Nt

-ix, irow, ie (energies), it - MAKE CODE CLEAR

Always use indenting (Editor | Indent| Smart --> or Ctrl + I

---Accumulation----

accumulator=0 - initialize a variable called the accumulator

-usually set to zero

accumulator = accumulator + newValue

- Each time through the loop compute a new quantity and add it to the current value of the accumulator,

storing the result back in the accumulator

-----Series--------

N=100;

S=0; -accumulator

for k=1:N

S=S+k; -adding values onto the accumulator (adds 1 with 2, then 1+2 with 3, then 1+2+3 to 4, etc..., eventually reaches to the addition of all the integers)

end

clc

disp(['Sum of the first ', num2str(N), ...

'integers is ', num2str(S)]);

Output: Sum of the first 100 integers is 5050.

-----Geometric Sum-------

N=5;

a=1/3;

S=0;

for k=0:N

S = S + a^k;

end

-Geometric series sum of (1/3)^k to five terms

--Nested For----

for icol=1:Ncols

for irow=1:Nrows

disp(['(', num2str(irow), ',', num2str(icol), ')', ...

' element is ', num2str(A(irow,icol))]);

end

end

(1, 1) element is 60

(2, 1) element is 81

(3, 1) element is 84

....

(1, 2) element is.....

(2, 2) element is.....

(3, 2) elment is......

.....

-Goes through icol with its initialized value

and then through the entire loop of the next for statement

before rolling into the next value of the initial for statement

Control Flow Commands are commands that control the flow information, the "if" statement

being one of them.

---------------------------------------------------------------------------------------

if fuelLevel < lowFuelLeevel

disp('Warning--low fuel!')

end

//

if <logical expression>

<statement block>

end

-If <logical expression> evalutes to true, <statement block> is executed.

Otherwise it is not, and execution proceeds to the next statement.

-"end" tells the program to stop reading the next lines as statement blocks

(concludes if statement)

-----------------------------------------------------------

if <logical expression>

<statement block1>

else

<statement block2>

end

-If <logical expression> evalutes to true, <statement block1> is executed

Otherwise <statement block2> is executed.

----------------------------------------------------------------

if <logical expression>

<statement block1>

elseif <logical expression2>

<statement block2>

elseif <logical expression3>

<statement block3>

...

else

<statement block>

end

--If <logical expression> evalutes to true, <statement block1> is executed.

Otherwise, execution proceeds to the evaluation of next 'elseif' logical expression.

-Exeuction moves continuously to the next "elseif" statement if previous logical expressions

are evaluted to be false.

-When a logical expression is evaluated to be true, the corresponding

statement block is then evaluated and the statement ends.

-If none are true, then execution moves to "else" statement.

--------------------------------------------------------------------

if <logical expression>

<statement block1>

elseif <logical expression2>

<statement block2>

elseif <logical expression3>

<statement block3>

if <logical expression3b> - If 'elseif' function here is true, then execution

<statement block3b> proceeds to evaluate the logical expression of the included "if" statement

end - Or <logical expression3> && <logical expression3b>

... can be used with the original "elseif" statement

else

<statement block>

end

--Animation Basics----

Draw a sine curve smoothly from left to right

parameters T, tmin, tmax, and Nt=300 is given.

t = linspace(tmin, tmax, Nt);

y=sin(2\*pi\*t/T);

for it=1:Nt

plot(t(it), y(it), 'ro', ...

t(1:it), y(1:it), 'b') - this second graph creates blue trail

axis([tmin, tmax, -1.1, 1.1]); -Need to take control of scaling.

Or else Matlab rescales every time

grid on

xlabel... -Because it replots every loop, you need to

-relabel/grid and such every time too

ylabel...

drawnow

end

----------------------------------------------------------------------

Animate drawing of a circle

Nth=100;

R=1;

theta=linspace(0, 4\*pi, Nth); - 4\*pi means it goes around circle twice

x=R\*cos(theta);

y=R\*sin(theta);

for ith=1:Nth

plot(x(ith), y(ith), 'ro', ...

x(1:ith), y(1:ith), 'b');

axis(1.2\*[-R, R, -R, R]); -Taking control of axis (and making it a little bigger for comfort)

axis square -Frames the xy axis into a square

grid on

drawnow

end

---WHILE statements----

while <logical expression>

<statement block>

end

---<logical expression> is evaluated continuously and as long as it's true,

<statement block> will be executed

-------Basic example----

sum=0;

maxsum=10; -Initialization

k=1; -Explicitly initialize all values involved (set a certain value for them)

while sum<maxsum -Condition

sum=sum+k; -Work

k=k+1; -Increment

end

----sayAmen----

correctResp='Amen!';

resp=input('Give me an Amen! ', 's');

while ~strcmp(resp, correctResp) -Remember, '~' means NOT

disp('I say it again!');

resp=input('Give me an Amen!', 's';

end

disp('Thank you!');

-----------sayAmenN-------

correctResp='Amen!';

nTriesMax=4;

nTries=1;

resp=input('Give me an Amen! ', 's');

done=strcmp(resp, correctResp);

while ~done

if nTries < nTriesMax

disp('I say it again!')

resp=input( ' Give me an Amen! ', 's');

done=strcmp(resp, correctResp); -Make sure you include this in loop so that it knows when to be done

if done

disp('Thank you!');

end

nTries=nTries +1;

else

disp('Never mind. God bless you!')

done=true;

end

end

---Common Animation Struture------

%% set parameters

%% calculate motion (and other information)

%% animate motion

for it=1:Nt

plot curves

...(optional: line, text, patch, image...)

set axes limits

label axes

drawnow

end

------------------------------

----Remember to add on a program slowly

-Get basic structure down and then add to it

%% ManhattanRandomWalk

% random walk on a square grid

N = 1e3;

x=zeros(1, N);

y=zeros(1, N);

doAnimate=false;

%% calculate a walk starting at the origin

for istep=2:N-1

idirection=randi(4);

switch idirection

case 1 % north

x(istep+1)=x(istep);

y(istep+1)=y(istep)+1; --the -1 and +1's are the steps in the left/right/up/down directions to

correlate with west/east/north/south

case 2 % south

x(istep+1)=x(istep)

y(istep+1)=y(istep)-1;

case 3 % west

x(istep+1)=x(istep)-1;

y(istep+1)=y(istep);

case 4 % east

x(istep+1)=x(istep)+1;

y(istep+1)=y(istep);

end

end

%% animate plot of walk

if doAnimate

for istep=2:N

plot(x(1:istep), y(1:istep), ... % draws path

x(istep), y(istep), 'ro', ... % draws path point -To have line and a marker, only do the color/shape thing at the end

0, 0, 'r^') -This creates a marker at the center

xyMax=max(abs([x(1:istep),y(1:istep)]));

axis([-xyMax, xyMax, -xyMax, xyMax]); -The axis are set to the maximum values of x an y.

This way one set of axis are enough.

And do not need to be constantly changing

axis square

drawnow

end

else

plot(x,y,... -Skips the animation of the circle, and goes straigh to final results

x(end), y(end), 'ro', ...

0, 0, r^, ...

'MarkerFaceColor', 'r') -Colors in the Markers with a certain color (in this case - red)

xyMax = max(abs([x, y, 10]));

axis([-xyMax, xyMax, -xyMax, xyMax]);

line([-xyMax, xyMax], [0, 0], 'Color', 'k');

line([0, 0], [-xyMax, xyMax, 'Color', 'k');

axis square

grid on

------- Line Command---------

Line command does not create axis or redraw figure

Use it to add lines to existing figure

line(X, Y);

line(X,Y, 'Color', 'r');

line(X,Y,'Color', 'r', 'LineWidth', 2);

line(X, Y, Z);

Type: doc primitive line properties

to see list of properties

-----------------Handle--------------------

The line properties can also be changed by getting the "handle" of the linen obejct

and then setting the properties values using the "dot" notation

hL=line([0, 0.5], [0, 0.5]);

hL.Color = 'r';

hL.LineWidth=1.2;

--------Inserting Images---------

1. monkey = imread('monkey.bmp'); - Read in image data from file

2. image('CData', monkey, ...

'XData', [x1, x2], 'YData', [y1, y2]); -Create image on current figure and set its position by setting

value of the XData and YData properties

-XData and YData are 2 x 1 row vectors that have, as their coordinates, lower left and upper right

-CData represents the color properties of the image

------Complex Numbers-------

i = j = sqrt(-1)

z=a+ib

Euler's formula:

e^iz = cos(z) +isin(z)

-----------------------------------------------

z=complex(4,5);

=

z=4+5i;

z4=4\*exp(j\*pi/4); -

z5=3\*exp(1i\*pi/4); -safe way to always get an i (or j)

---------------------------------------------------

z = x+i\*y x: real part

z = r\*e^i(theta) y: imaginary part

z = x - i\*y = r\*e^(-i\*theta) r: modulus

theta: angle

real(z) real part of z

imag(z) imaginary part of z

abs(z) modulus of z

angle(z) phase angle of z in radians

conj(z) complex conjugate of z

------------------------------------------------------------------------

%complexReflection

lamba = 1;

xmin=0;

xmax=3\*lambda;

k=2\*pi/lambda;

Nx=200;

R=1;

phi=pi/4;

r=R\*exp(1i\*phi);

x=linspace(xmin, xmax, Nx);

f = exp(1i\*k\*x) + r\*exp(=1i\*k\*x);

plot(x, abs(f), 'k-', ...

x, real(f), 'b', ...

x, imag(f), 'r', ...

x, angle(f)/pi, 'k--')

legend('|f|', 'Re(f)', 'Im(f)', 'angle(f)/\pi');

grid on

xlabel('x')

title(['f(x)=e^(ikx) + r e^(-ikx) where', ...

'r=Re^(i/phi), R= ', num2tstr(r), ...

' \phi= ', num2str(phi/pi), ' \pi']);

---------Multidimensional Arrays----------

M(k1, k2, k3) - k1: row

- k2: column

- k3: page (which set of 2D array)

- Can have more....

----------Cell Arrays----------------------------

Arrays aggregate elements that are of the same class. Individual elements are referenced by integer indices.

M(k, n) -all numbers within this array

isOn(iswitch) -all logic operators within this array

Cell arrays, by contrast, can aggregate elements of different classes. Individual elements are also referenced

by integer indices.

------------

C{1,1} = 57.23; -Curly brackets indicate Cell Arrays

C{1,2} = 'Moses';

C{2, 1} = rand(2);

C{2, 2} = 4+7i;

Output:

C =

[57.23] 'Moses'

[2x2 double] [4 + 7i]

C{2, 1}(1, 2) -Gets the row 1 column 2 value of the matrix within the row 2 column 1 of the cell array

-------------

Cell arrays can hold strings of different lengths in one cell array

--------------

% list Lawyers

partners{'Warpe', 'Wistful', 'Kibutschek'};

np=length(partners);

partners{np+1}='McMingus';

clc

np=length(partners);

disp('Law partners: ')

for ip = 1:np

disp(partners{ip});

end

--------------Structures-----------------------

A MATLAB structure, class struct, can aggregate elements of different classes.

Individual elements are referenced, not by indices, but by "field names".

------------------------

Input:

student.firstname = 'Galusha';

student.lastname='Pennypacker';

student

Output:

firstname: 'Galusha'

lastname: 'Pennypacker'

-----------------------

Input:

student.scores=[100, 99.5, 99];

student

Output:

scores: [100, 99.5, 99];

-------------------------

Input:

disp['Most recent test score: ', num2str(student.scores(end)) ]);

Output:

Most recent test score: 100

------------Fieldnames-----------

Fieldnames have no particular order.

Fieldnames can have no spaces.

Data contained in each field can be of any class (including another struct)

Structs can be aggregated into arrays.

----------------------

classlist(3).firstname

-Robin

for is = 1:length(classlist)

disp(classlist(is).firstname); -classlist is a vector of firstname, lastname, gpa, and scores

end

---goes down the list of names

--------MATLAB data classes---------

double

char(string)

logical

complex

handles

handle objects

function handles

objects

-------Aggregating Containers-------

array: homogenous class

1D vector, 2D matrix

multidimensional

access; indices

cell aray: heterogenous classes

access: indices

struct: heterogenous class

access: field name

maps

----Functions with multiple returns---------

Syntax: [var1, var2, … ] = funcName(x1, x2, x3, …);

After var1, other returned variables are optional, but they must occur in the prescribed order.

Each input or output variable can be any MATLAB data structure

-Ex: double, string, vector, matrix, other

Specify in header comments what each input and output represent

>> help myfun prints header comments

If user calls function with just one variable on left, then only one function is returned.

-The more variables asked on left, the more returned,

AS LONG AS function is defined to have that many inputs (x1, x2, x3, etc…)

Function [thetad, r] = function cart2pold(x, y)

r = sqrt(x.^2 + y.^2);

thetad = atan2d(y,x);

-----Functions------

First executable statement should be function declaration

Ex: function y = mypoly(x)

[then comments and the rest of the code]

a = 1;

b = 2;

c = 3;

y = a\*x.^2 + b\*x + c; - the dots allow this function to work for vectors and matrices as well

mypoly(0)

= 3

x=linsapce(-5, 5, 200);

z = mypoly(x);

plot(x, z); - Can plot functions

-------Defining a function—

First line must start with the word “function” and the declaration

File name must be the same as the function name, including capitalization (mypoly.m)

Function must calculate at least one value to be returned

Function can be used like other MATLAB functions;

Ex: z = mypoly(1.2);

y = mypoly(x)

y = return variable, mypoly = name, x = input

-----------Function input and output--------

a = 101;

z = 2;

f = mypoly(z);

-Calculates mypoly(2) to return 11.

-HOWEVER, the ‘a’ assignment does not affect the ‘a’ variable within the function (this being a local variable)

-Only interacts with outside world by receiving an input and giving an output

-------Intro to Debugging-------

Function Call Stack:

-Change your view of inside or outside the function

---------------

help mypoly

-Displays all the comments written on the function

(all % lines before the first %%)

Functions cannot alter the value of a variable in the calling statement (the z in mypoly(z))

------Local Variables------

Variables created inside the function are local

-i.e. Not accessible outside the function

Local variables can have same name as variables outside the function without conflict

-Each function has its own private workspace

Local variables are temporary.

-They are created when the function runs and disappear afterwards.

-------Function Prime Directive------

Each function should do one thing and do it well.

function [g, dgdx] = gauss(x, x0, a)

prefactor = 1/(a\*sqrt(2\*pi));

g = prefactor\*exp( -(x-x0).^2 / (2\*a^2));

dx = (1e-4)\*a;

xp = x+dx/2;

xm = x-dx/2;

gp = prefactor\*exp( -(xp-x0).^2 / (2\*a^2));

gm = prefactor\*exp( -(xm-x0).^2 / (2\*a^2));

dgdx = (gp, -gmm)/dx;

----------------

%% plotgauss.m

x0 = 0;a

a = 0.5;

xmin =-5;

xmax = +5;

Nx = 300;

iPlotChoice = 3;

x = linspace(xmin, xmax, Nx);

g = gauss(x, x0, a); -invoking function

switch iPlotChoice

case 1

plot(x, g)

ylabel(‘g(x)’)

case 2

plot(x, dgdx)

ylabel(‘dg/dx’)

case 3

plot(x, g, x, dgdx);

legend(‘g(x)’, ‘dg/dx’)

end

xlabel(‘x’)

title(‘Gaussian’)

grid on

-------

Now we can call either:

g = gauss(0, 0, 0.5)

or

[g, dgdx] = gauss(1, 0, 0.5)

-----Debugging MATLAB functions------

Cannot run function without argument by pressing green save-and-run button

1. Execute from command line with specific arguments:

gauss(0, 0, 0.25)

To see if it correlates with the right value

2. Run simple function in outside program

-Plotting, calculating some number, etc…

3. Modify run button

-Click on bottom arrow on Green Run-and-Save button

-Enter the inputs of function

-Ex: gauss(0, 0, 0.5)

-To remove this function, right click the run-and-save option and press delete

-----------------

Can have several functions in one file

1st function must have same name as file (primary function)

-Other functions are called subfunctions

Other functions must still have function declaration (but don’t have to have an end associated with them)

Subfunctions can only be executed by another function in the file (whether it be primary function or some other sub function)

---------

Keep functions organized in same folder

DO NOT USE “ADD TO PATH”

list = Y;

for i=Y+1:N+Y-1

list = [num2str(list), ', ', num2str(i)];

end

for i=1:5

if (class(i).seats(1, 1) - class(i).seats(1, 2)) > 4

disp(class(i).title)

end

end

case 4 % east

x(istep+1)=x(istep)+1;

y(istep+1)=y(istep);

end

end

%% animate plot of walk

for istep=2:N

plot(x(1:istep), y(1:istep), ...

x(istep), y(istep), 'ro')

axis equal

drawnow

end

ispin=1;

while stash(ispin) > bet

r=rand;

if r < Pwin

% win

stash(ispin+1)=stash(ispin)+bet;

else

% lose

stash(ispin+1)=stash(ispin)-bet;

end

ispin=ispin+1;

% display the results so far

plot([1:ispin], stash,...

ispin, stash(ispin), 'ro')

axis([0, 1.05\*ispin, 0, inf])

xlabel('spin')

ylabel('stash($) ')

line([0, ispin], [stash(1), stash(1)], 'Color', 'r');

drawnow

end

C = length(colors);

M = length(messages);

BeMine = 0;

SpecialHearts = 0;

for i=1:Candies

icolor = randi(C);

imessage = randi(M);

hearts(i).message = messages(imessage);

hearts(i).colors = colors(icolor);

if strcmp(hearts(i).message, 'Be Mine')

BeMine = BeMine + 1;

elseif strcmp(hearts(i).colors, 'red') && strcmp(hearts(i).message, 'You Rock')

SpecialHearts = SpecialHearts + 1;

end

end

%% animate plot of the motion

for it=1:Nstride:Nt

plot3(x(1:it), y(1:it), z(1:it), 'b',... %draws line as it progresses

x(it), y(it), z(it), 'ro'); %updates marker

axis([-r0, r0, -r0, r0, min(z), max(z)]);

axis square

grid on

xlabel('x (cm)')

ylabel('y (cm)')

title(['Exponential decay with \tau = ', num2str(tau)']);

drawnow

end

----------------------------------------------------

t = linspace(0, Tf, Nt);

for it = 1:Nt

x(it) = r\*cos(a\*(2\*pi)/T\*t(it));

y(it) = r\*sin(b\*(2\*pi)/T\*t(it) + phi);

end

-----------------------------

for istep=2:N

idirection=2\*pi\*rand(1, 1);

x(istep) = x(istep-1)+step\*cos(idirection); % x position

y(istep) = y(istep-1)+step\*sin(idirection); % y position

end

if doAnimate

for istep=2:N

plot( x(1:istep), y(1:istep), ...

x(istep), y(istep), 'ro')

axisMax=max(abs([x(1:istep), y(1:istep)]))

axis([-axisMax, axisMax, -axisMax, axisMax]);

axis square

grid on

drawnow

end

else

plot( x, y, ...

x(end), y(end), 'ro')

axisMax=max(abs([x(1:istep), y(1:istep)]));

axis([-axisMax, axisMax, -axisMax, axisMax]);

axis square

grid on

end