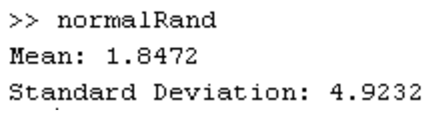
Homework 4 Solutions

1. The script below generates the random numbers and verifies the mean and standard deviation:

```
% normalRand
% homework 4 problem 1, making a vector of 500 random numbers that are
% normally distributed with mean 2 and standard deviation 5

r=randn(500,1)*5+2;

% verify that the mean and std are close to 2 and 5
disp(['Mean: ' num2str(mean(r))]);
disp(['Standard Deviation: ' num2str(std(r))]);
```

When run, the script displays the following:

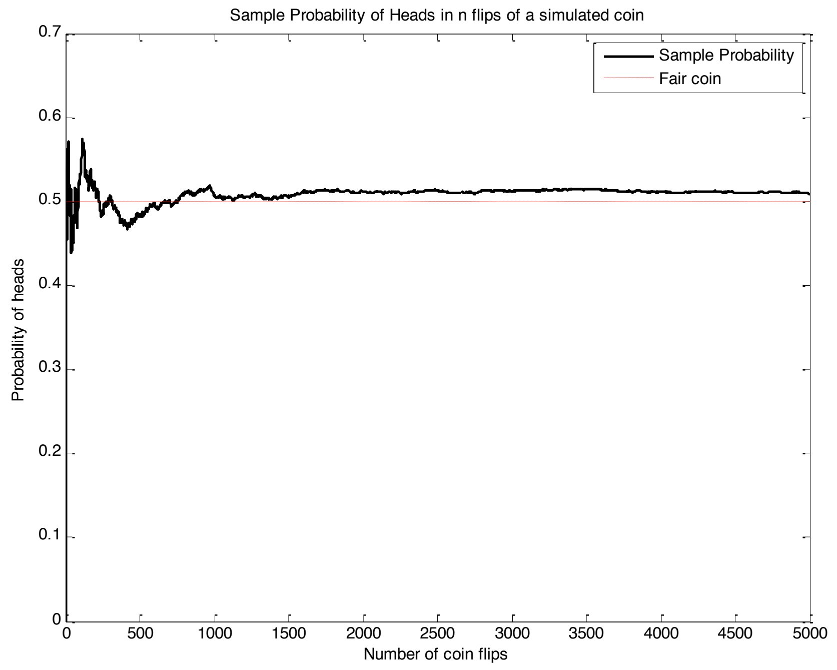


```
>> normalRand
Mean: 1.8472
Standard Deviation: 4.9232
```

2. The `coinTest.m` script is pasted below

```
% coinTest

% generate 1000 random numbers
r=rand(5000,1);
% round them so 1 means heads
r=round(r);

% count up the number of heads and divide by the count
count=1:length(r);
headProb=cumsum(r)./count';

% plot the probability of heads
figure
plot(count,headProb,'k','linewidth',1.5);
hold on
plot([1 length(r)],[.5 .5],'r--');

title('Sample Probability of Heads in n flips of a simulated coin');
xlabel('Number of coin flips');
ylabel('Probability of heads');
legend('Sample Probability','Fair coin');
```

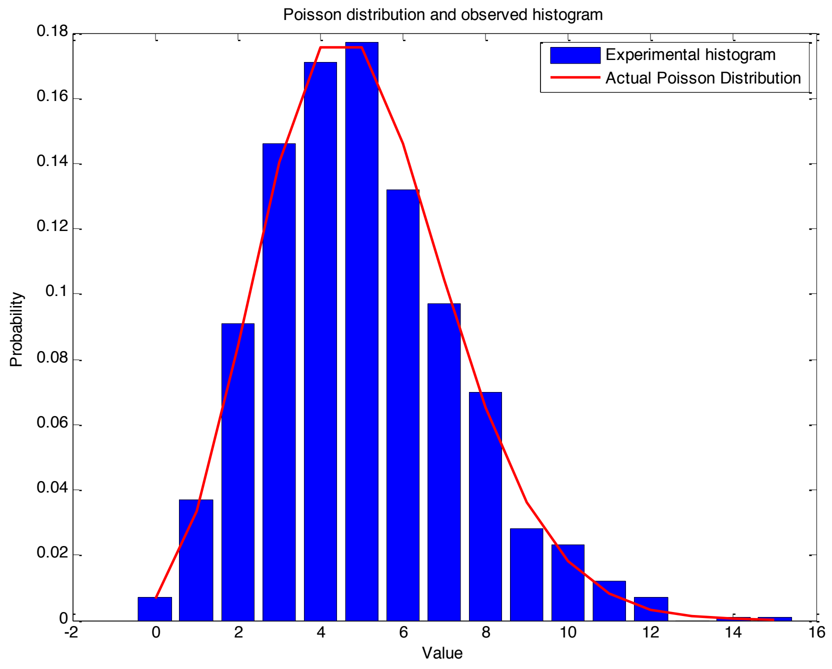
The figure generated by this script is:



3. The `plotPoisson.m` script is pasted below.

```
% plotPoisson
% makes a histogram of poisson random variables and plots the actual
% poisson distribution on top

% get the poisson random variables, with lambda=5
r=poissrnd(5,1000,1);

% make the histogram and normalize
[n,x]=hist(r,0:max(r));
n=n/sum(n);

% plot the histogram and actual pdf
figure
bar(x,n,'b');
hold on
plot(x,poisspdf(x,5),'r','linewidth',1.5);
xlabel('Value');
ylabel('Probability');
title('Poisson distribution and observed histogram');
legend('Experimental histogram','Actual Poisson Distribution');
```

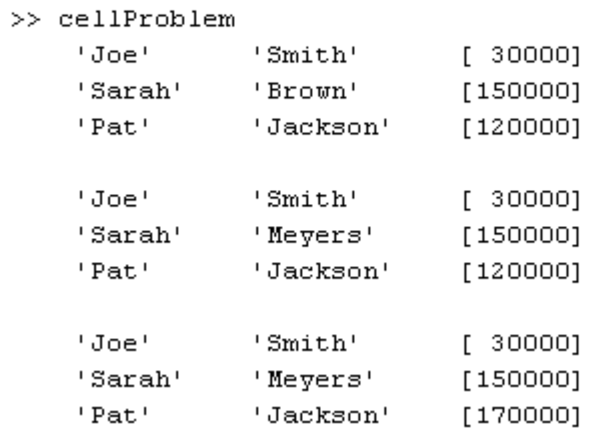
The script generates the figure below:



4. The script below initializes the cell and changes its values appropriately, displaying it after each change.

```
% cellProblem
% initializes a cell, puts some values in it, and changes some of them

% part a
% initialize the cell
c=cell(3,3);

% add elements
c{1,1}='Joe';
c{2,1}='Sarah';
c{3,1}='Pat';

c{1,2}='Smith';
c{2,2}='Brown';
c{3,2}='Jackson';

c{1,3}=30000;
c{2,3}=150000;
c{3,3}=120000;

% display the cell
disp(c)

% part b
% change sarah's name to Meyers
c{2,2}='Meyers';
disp(c)

% part c
% add 50000 to pat's salary
c{3,3}=c{3,3}+50000;
disp(c)
```

The above script generates this output:

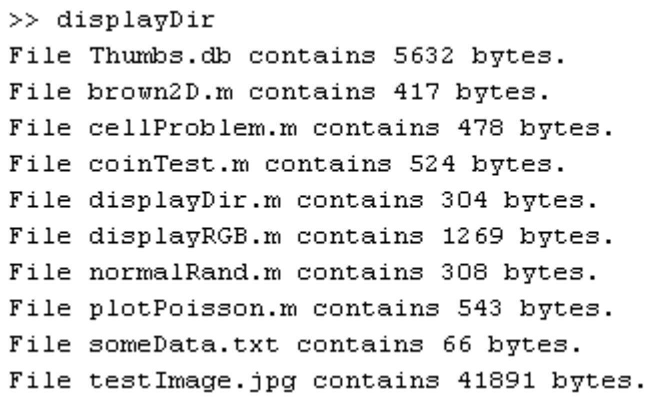


```
>> cellProblem
    'Joe'      'Smith'      [ 30000]
    'Sarah'    'Brown'      [150000]
    'Pat'      'Jackson'    [120000]

    'Joe'      'Smith'      [ 30000]
    'Sarah'    'Meyers'     [150000]
    'Pat'      'Jackson'    [120000]

    'Joe'      'Smith'      [ 30000]
    'Sarah'    'Meyers'     [150000]
    'Pat'      'Jackson'    [170000]
```

5. a) The size of the struct will vary with the number of files and folders in the directory. The fields are: name, date, bytes, isdir, datenum.
b) The loop is in the `displayDir.m` function below
c) The `displayDir.m` function is pasted below:

```
% displayDir
%
% displays a list of the files in the current directory and the size of
% each file in bytes

function displayDir

% get current directory info
a=dir;

% loop through it and display each file name and size
for n=1:length(a)
    % if it's not a directory, display the information
    if ~a(n).isdir
        disp(['File ' a(n).name ' contains ' num2str(a(n).bytes) ' bytes.']);
    end
end
```

When `dispDir.m` is run, it displays the following output:

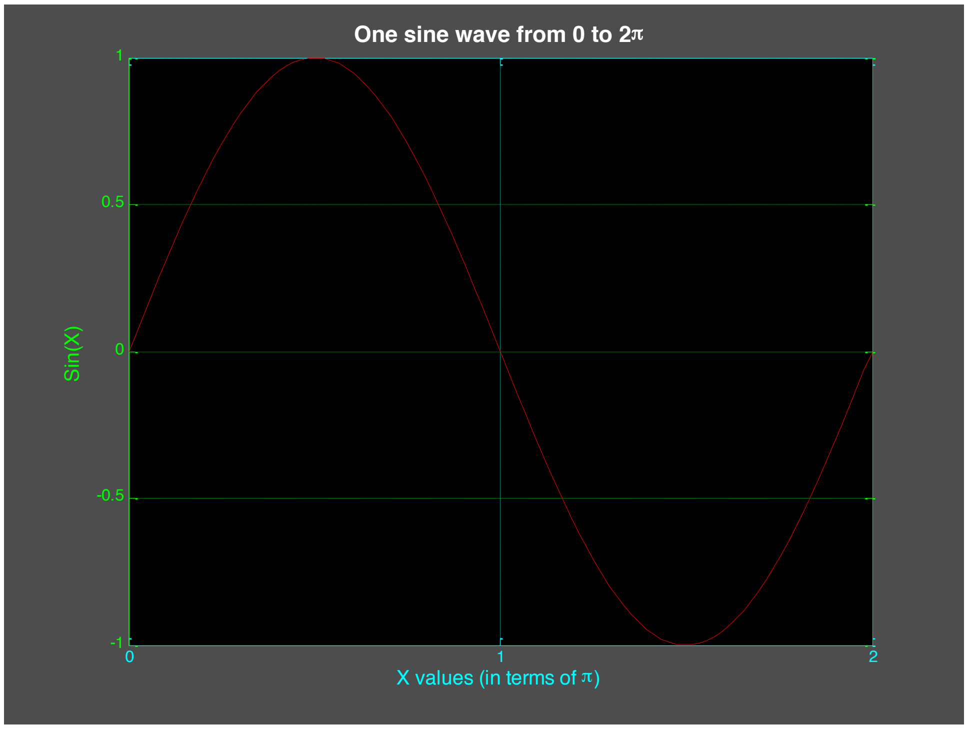


```
>> displayDir
File Thumbs.db contains 5632 bytes.
File brown2D.m contains 417 bytes.
File cellProblem.m contains 478 bytes.
File coinTest.m contains 524 bytes.
File displayDir.m contains 304 bytes.
File displayRGB.m contains 1269 bytes.
File normalRand.m contains 308 bytes.
File plotPoisson.m contains 543 bytes.
File someData.txt contains 66 bytes.
File testImage.jpg contains 41891 bytes.
```

6. The `handlesPractice.m` code is pasted below:

```
% handlesPractice
% plot the function
x=linspace(0,2*pi,100);
y=sin(x);
figure;
plot(x,y,'r');
xlim([0 2*pi]);
% set the ticks
set(gca,'xtick',[0 pi 2*pi],'xticklabel',{'0','1','2'});
set(gca,'ytick',[-1:.5:1]);
% turn on grid and set axis and background colors
grid on;
set(gca,'ycolor','g')
set(gca,'xcolor','c');
set(gca,'color','k');
set(gcf,'color',[.3 .3 .3]);
% add the labels
title('One sine wave from 0 to
2\pi','fontsize',14,'fontweight','bold','color','w')
xlabel('X values (in terms of \pi)','fontsize',12,'color','c');
ylabel('Sin(X)','fontsize',12,'color','g');
```

The figure it generates is pasted below

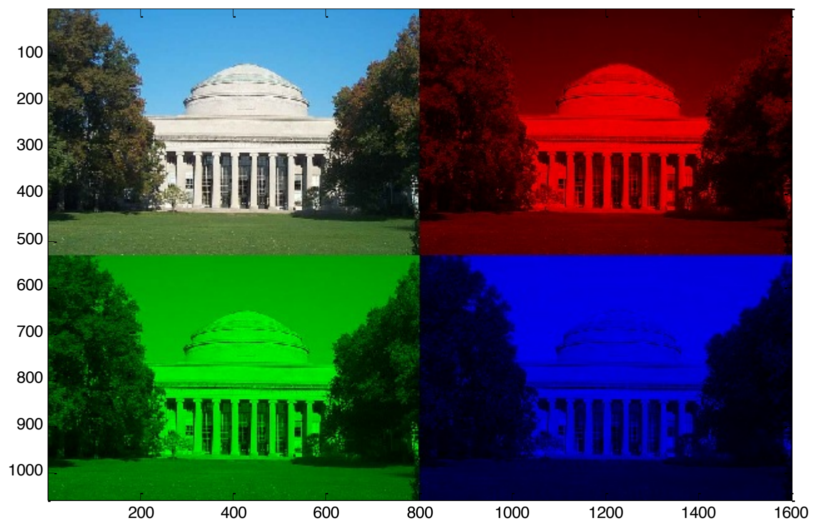


7. The `displayRGB.m` function is pasted below:

```
% im=displayRGB(filename)
%
% reads in the jpg file in filename (filename should include the
% extension), displays the original image as well as each color layer in a
% figure separately, and returns this composite image in im. the original
% image is resampled so the largest final dimension is 800 pixels and the
% aspect ratio is preserved.

function im=displayRGB(filename)

% load the image
im0=imread(filename);

% interpolate it to have 800 pixels in the larger dimension
% get the original x and y
M=size(im0,1);
N=size(im0,2);
x0=1:N;
y0=1:M;
[X0,Y0]=meshgrid(x0,y0);

% make appropriate meshgrids
if M>=N % taller than it is wide
    x1=linspace(1,N,round(800*N/M));
    y1=linspace(1,M,800);
else % wider than it is tall
    x1=linspace(1,N,800);
    y1=linspace(1,M,round(800*M/N));
end
% make the new meshgrid
[X1,Y1]=meshgrid(x1,y1);

% interpolate each layer and store it
im1=zeros(length(y1),length(x1),3);
for n=1:3
    im1(:,:,n)=interp2(X0,Y0,im0(:,:,n),X1,Y1,'cubic');
end

% make the big blank image to be twice as wide and tall as im1
im=zeros(2*size(im1,1),2*size(im1,2),3);

% make the top left quadrant the original image
im(1:size(im1,1),1:size(im1,2),:)=im1;
% make the top right quadrant the red image
im(1:size(im1,1),size(im1,2)+1:end,1)=im1(:,:,1);
% make the bottom left quadrant the green image
im(size(im1,1)+1:end,1:size(im1,2),2)=im1(:,:,2);
% make the bottom right quadrant the blue image
im(size(im1,1)+1:end,size(im1,2)+1:end,3)=im1(:,:,3);

im=uint8(im);
```

```
% display the image in a figure
figure
image(im)
axis equal
axis tight
```

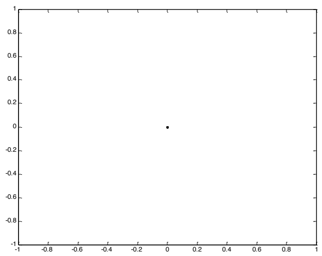
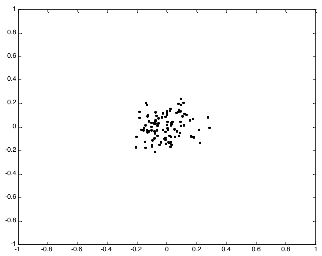
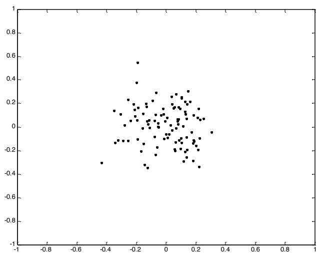
When run with a picture of the MIT dome, it generates this figure:



8. The `brown2D.m` function is pasted below:

```
% brown2D(N)
%
% simulates the Brownian motion of N points. all N points begin in the same
% location and diffuse away from it as time passes. The animation runs for
% about 10 seconds

function brown2D(N)

% make a figure
figure;
axis square
axis([-1 1 -1 1]);

% make the x and y positions of the points
x=zeros(N,1);
y=zeros(N,1);
h=plot(x,y,'.k');

% run a loop to display the points
for n=1:1000
    % update the positions
    x=x+randn(size(x))*.005;
    y=y+randn(size(y))*.005;

    % plot them
    set(h,'xdata',x,'ydata',y);
    axis([-1 1 -1 1])
    pause(0.01);
end
```

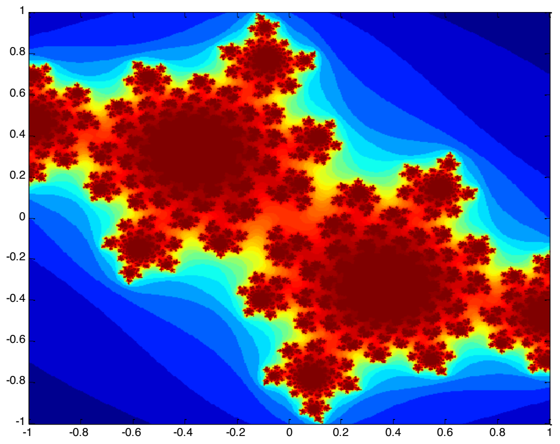
The figures below are from the start, middle, and end of the simulation:

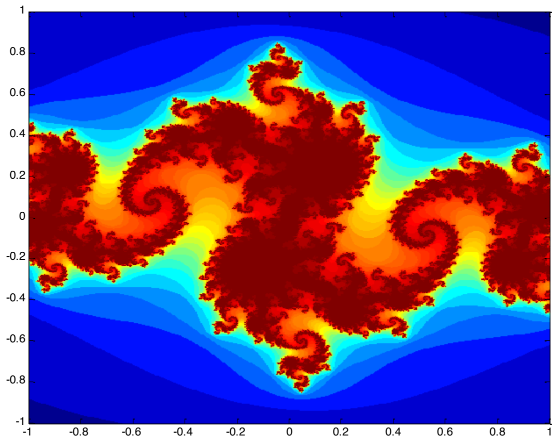




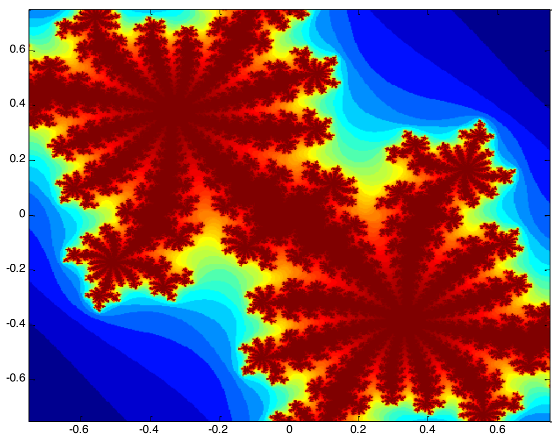
9. The `juliaAnimation` code is pasted below.

```
% juliaAnimation(zMax,c,N)
%
% displays an animation of a Julia Set
%
% inputs:
% zMax - the complex numbers for which escape velocities are computed will
%       be 500 values between -zMax and zMax for both real and imaginary
%       part
% c - the complex number c, which is a parameter in the Julia set
% N - the maximum number of iterations to use when computing escape
%       velocities; also equals the number of frames in the animation
%
% EXAMPLES
%           juliaAnimation(1,-.4+i*.6,75);
%           juliaAnimation(1,-.8+i*.156,100);
%           juliaAnimation(.75,-.297491+i*.641051,150);

function juliaAnimation(zMax,c,N)

% make the complex grid, with real and imaginary part going from -zMax to
% zMax
res=500; % resolution, number of points on a side of the grid
temp=linspace(-zMax,zMax,res);
[R,I]=meshgrid(temp,temp);
Z=R+1i*I;

% initialize the map to have 'escape velocities' all equal to N
M=N*ones(size(Z));

% loop N times, each time generating the new Z and visualizing it
figure
for n=1:N
    % calculate Zn for the current n
    Z=Z.^2+c;
    % find where the modulus is >2
    inds=find(abs(Z)>2);
    % set these inds in M to contain the loop iteration
    M(inds)=n;
    % set these inds in Z to be nan
    Z(inds)=nan;
    imagesc(temp,temp,atan(0.1*M));
    axis xy
    drawnow
end
```

Running the examples listed in the help `juliaAnimation` help file generates these fractals (when run in Matlab, they show up as animations; the images below just show the final fractal)


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.6
-0.4
-0.2
0
0.2
0.4
0.6