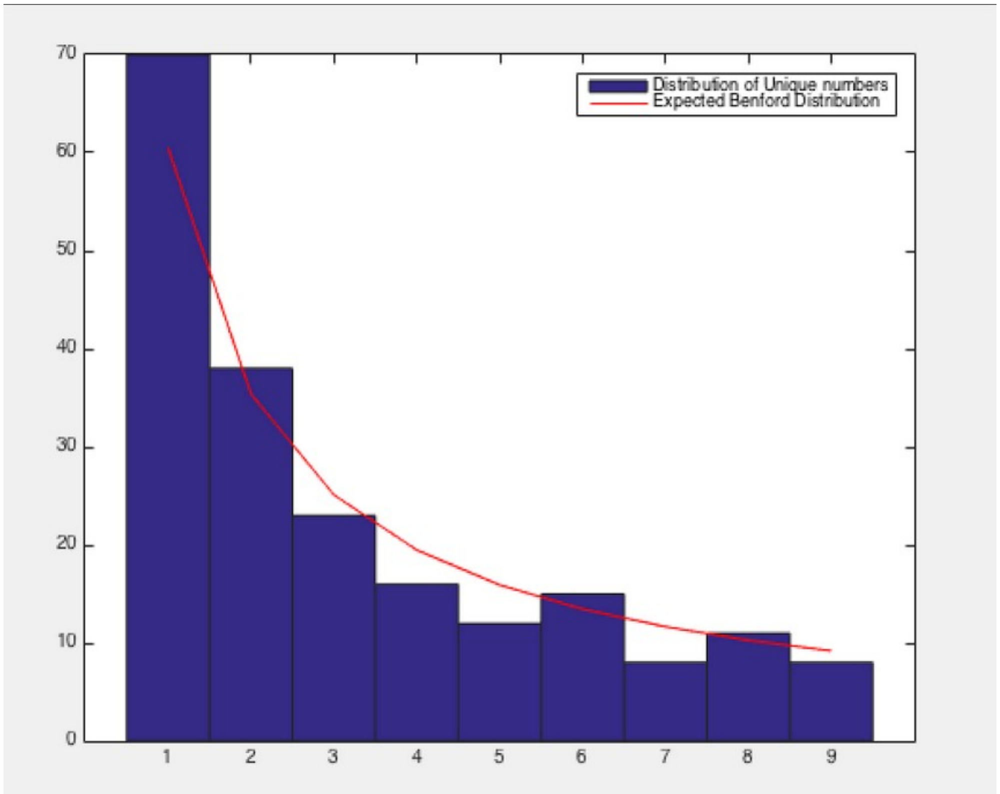
NAME : SRAVANI KAMISETTY
SID : 304414410
COURSE : MATHEMATICAL MODELLING
ASSIGNMENT #4

QUESTION 1:

```
function data = text_file_read(filename)
    idx=0;

    fid = fopen(filename)
    if fid == -1
         error(['Error Opening ',filename]);
    end
    while ~feof(fid)
       data_to_split = fgetl(fid);
       split = strsplit(data_to_split,',');
       [rows,cols] = size(split);
       while(cols ~= 0)
         idx=idx+1;
         data(idx) = split(1,cols);
         cols = cols -1;
       end
    end
    fclose(fid);
end

unique_data = unique(data);
[rows,cols] = size(unique_data);
count_arr(1:10) = 0

for i=1:cols
    clear x;
    x = unique_data(1,i){1};
    ans = strsplit(strtrim(x),' ');
    [rows1,cols1] = size(ans);
    j = 1;
    while(j<= cols1)
         clear ans;
         ans = strsplit(strtrim(x),' ');
         clear ans1;
         ans1 = str2num(ans(1,j){1});
         if(ans1 ~= 0)
            break;
         end
         j++;
    end
    if(j >cols1)
        ans1 = 10;
    end
    count_arr(ans1) += 1;
end
statPercent = count_arr(1:9)/ sum(count_arr (1:9));
bar(statPercent);
```


Distribution of Unique numbers
Expected Benford Distribution
70
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9

QUESTION 4:

a)

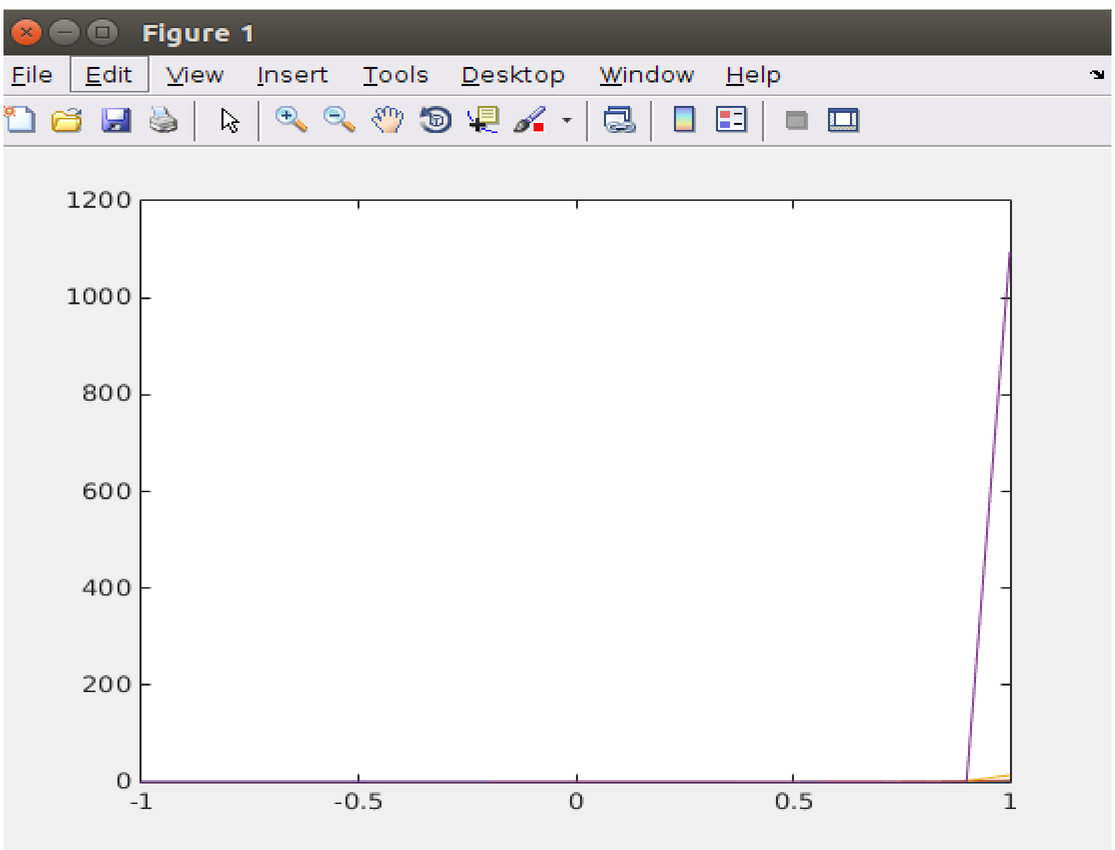


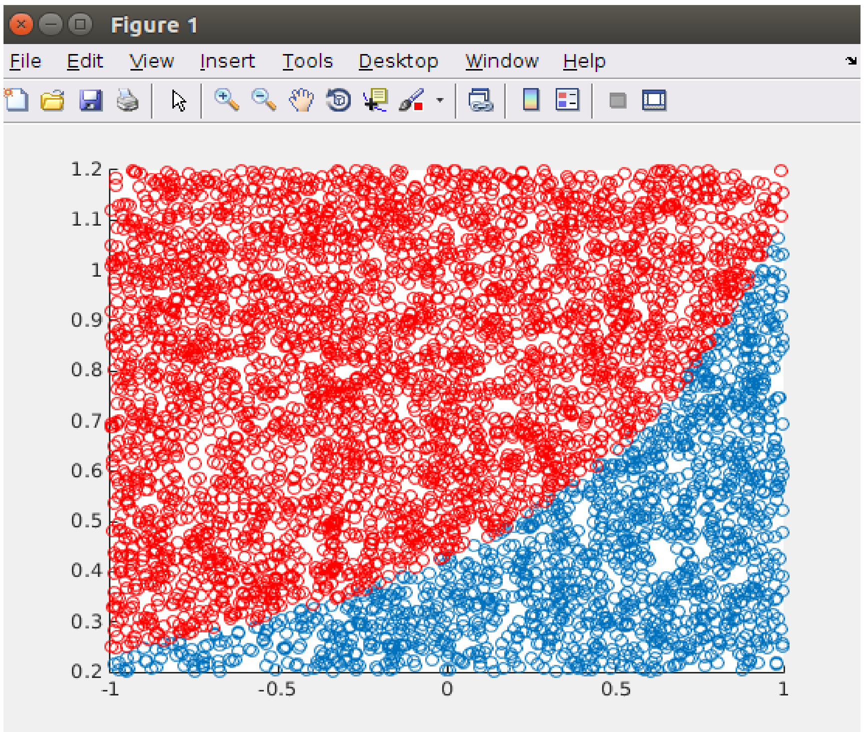
Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
1200
1000
800
600
400
200
0
-1
-0.5
0
0.5
1

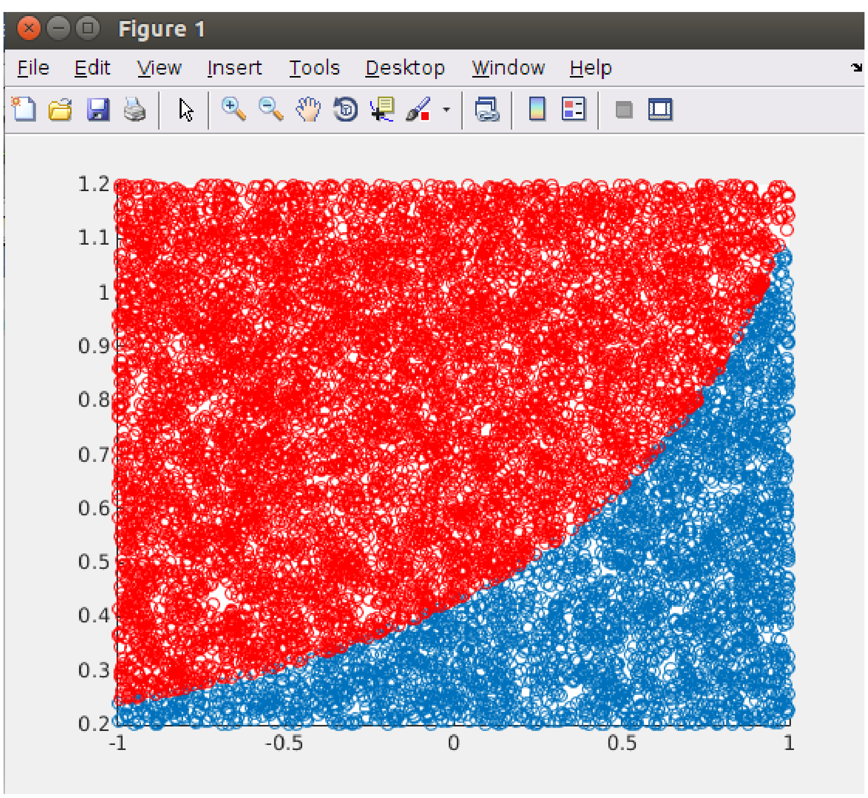
```
function [ x,y ] = henyey_greenstein( g )
    j=1;
    i=-1;
    g
    while(i<=1)
        i
        x(j) = i;
        clear numerator;
        clear denominator;
        numerator = 1-g*g;
        denominator =  (1+g*g-2*g*i)^1.5;
        product = numerator/denominator;
        value = 0.5*product;
        y(j) = value;
        j = j+1;
        i = i+0.1;
    end
end

hold all;
[x2,y2] = henyey_greenstein(0.97);
plot(x2,y2);
[x2,y2] = henyey_greenstein(0.75);
plot(x2,y2);
[x2,y2] = henyey_greenstein(0.50);
plot(x2,y2);
[x2,y2] = henyey_greenstein(0.25);
plot(x2,y2);
[x2,y2] = henyey_greenstein(0);
plot(x2,y2);
```

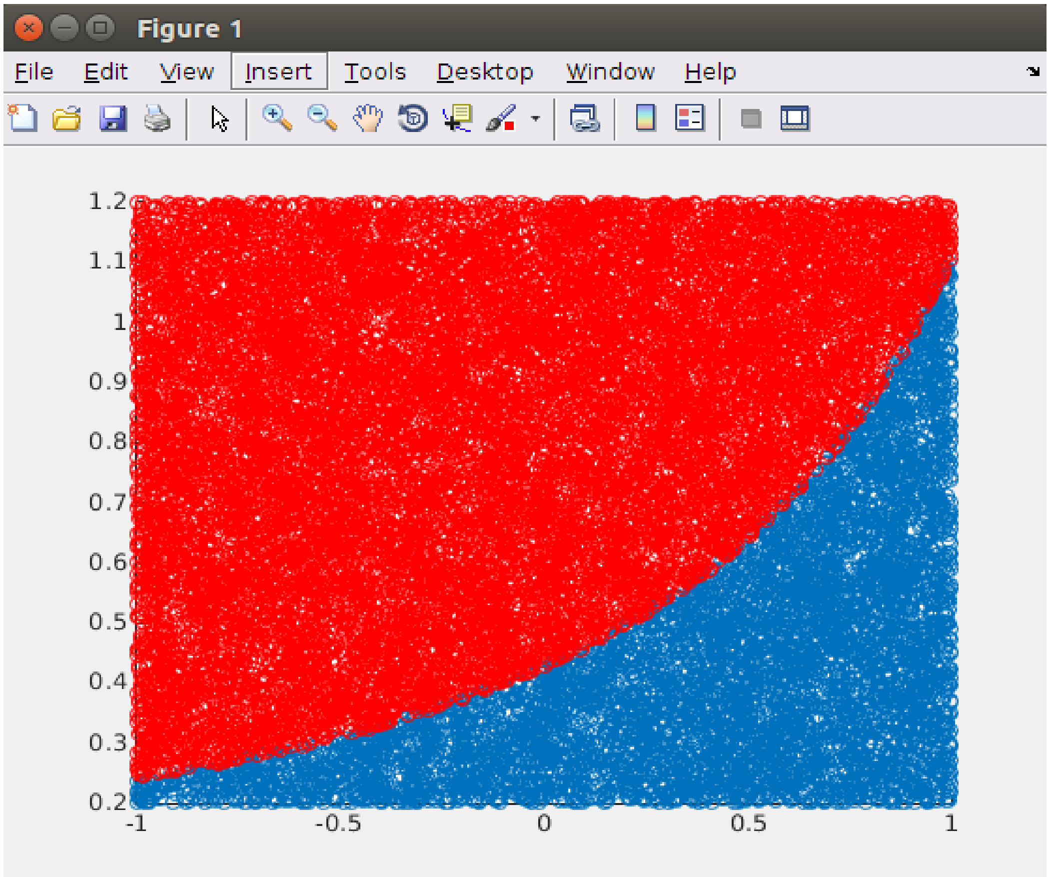
b)

```
function [ hits_array, misses_array ] = monte_carlo( total_points  )
    hits = 0;
    misses = 0;
    g = 0.25;
    for i=1:total_points
        x = 2*rand -1;
        y = rand + 0.2;
        numerator = 1-g*g;
        denominator =  (1+g*g-2*g*x)^1.5;
        product = numerator/denominator;
        value = 0.5*product;
        if (value - y >= 0)
            hits = hits+1;
            hits_array(hits,1) = x;
            hits_array(hits,2) = y;
        else
            misses = misses+1;
            misses_array(misses,1) = x;
            misses_array(misses,2) = y;
        end
    end
end

>> [hits,misses] = monte_carlo(5000);
>> scatter(hits(:,1),hits(:,2));
>> hold all
>> scatter(misses(:,1),misses(:,2),'r');
>> [hits,misses] = monte_carlo(10000);
```

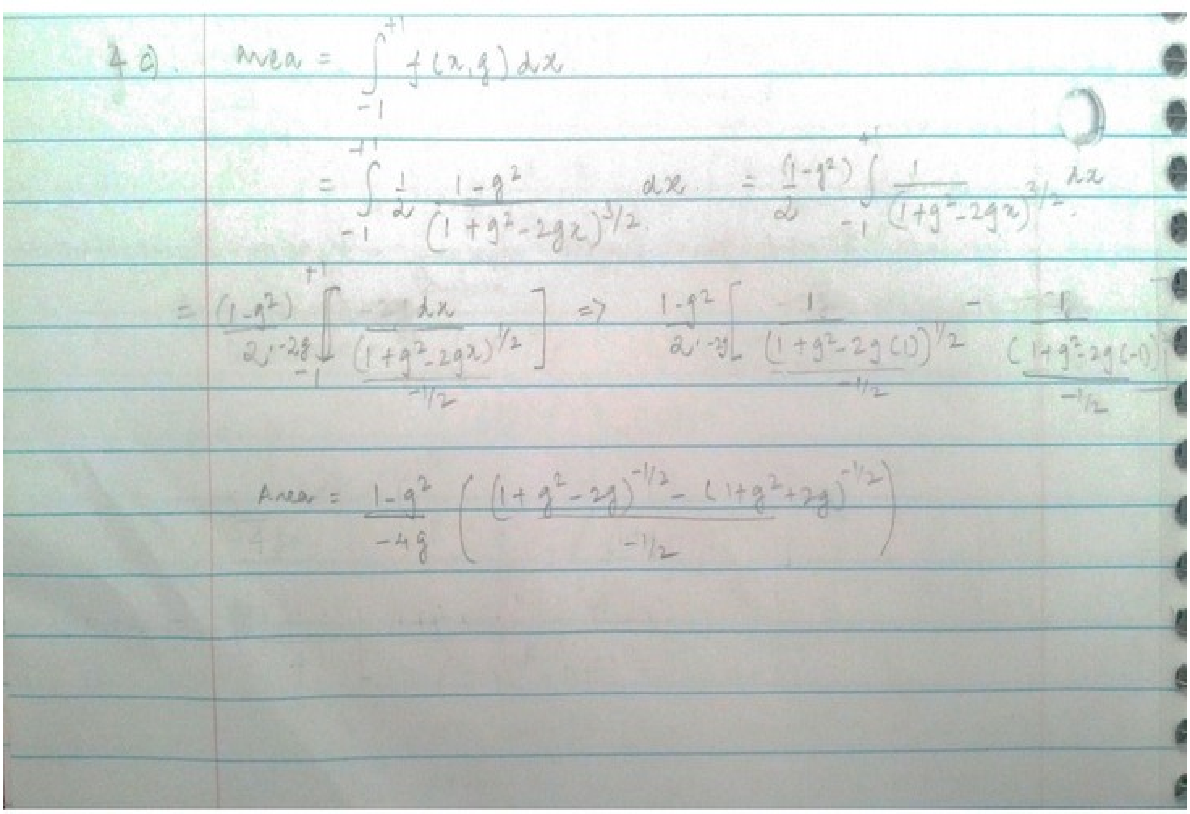
```
>> scatter(hits(:,1),hits(:,2));
>> hold all
>> scatter(misses(:,1),misses(:,2),'r');
>> [hits,misses] = monte_carlo(20000);
>> scatter(hits(:,1),hits(:,2));
>> hold all
>> scatter(misses(:,1),misses(:,2),'r');
```







4c)



4c) Area $= \int_{-1}^{+1} f(x,g)\,dx$

$$= \int_{-1}^{+1} \frac{1}{2}\,\frac{1-g^2}{(1+g^2-2gx)^{3/2}}\,dx = \frac{(1-g^2)}{2}\int_{-1}^{+1}\frac{1}{(1+g^2-2gx)^{3/2}}\,dx$$

$$= \frac{(1-g^2)}{2\cdot -2g}\left[\int_{-1}^{+1}\frac{-2g\,dx}{(1+g^2-2gx)^{3/2}}\right] \Rightarrow \frac{1-g^2}{2\cdot -2g}\left[\frac{1}{\frac{(1+g^2-2g(1))^{1/2}}{-1/2}} - \frac{1}{\frac{(1+g^2-2g(-1))^{1/2}}{-1/2}}\right]$$

$$\text{Area} = \frac{1-g^2}{-4g}\left(\frac{(1+g^2-2g)^{-1/2}-(1+g^2+2g)^{-1/2}}{-1/2}\right)$$

4d)

4d) $$ICDF = \int_{-1}^{y} f(x,g)\,dx = \int_{-1}^{y} \frac{1}{2}\,\frac{(1-g^2)}{(1+g^2-2gx)^{3/2}}$$

$$= \frac{(1-g)^2}{2}\int_{-1}^{y} \frac{1}{(1+g^2-2gx)^{3/2}} = \frac{1-g^2}{2}\left[\frac{-2g}{(1+g^2-2gx)^{1/2}}\right]_{-1}^{y}$$

$$= \frac{(1-g^2)(-2g)}{2}\left[\frac{1}{(1+g^2-2gy)^{1/2}} - \frac{1}{(1+g^2-2g(-1))^{1/2}}\right]$$

$$= (g^2-1)g\left[\frac{1}{(1+g^2-2gy)^{1/2}} - \frac{1}{(1+g^2+2g)^{1/2}}\right]$$

$$\frac{ICDF}{(g^2-1)g} + \frac{1}{(1+g^2+2g)^{1/2}} = \frac{1}{(1+g^2-2gy)^{1/2}}$$

$$\frac{ICDF}{(g^2-1)g} + \frac{1}{(1+g^2+2g)^{1/2}} = \frac{1}{(1+g^2-2gy)^{1/2}}$$

$$\left(\frac{ICDF}{(g^2-1)g} + \frac{1}{(1+g^2+2g)^{1/2}}\right)^2 = \frac{1}{1+g^2-2gy}$$

$$1+g^2 - \left(\frac{1}{\frac{ICDF}{(g^2-1)g} + \frac{1}{(1+g^2+2g)^{1/2}}}\right)^2 = 2gy$$

$$y = \frac{1}{2g}\left(1+g^2 - \left(\frac{1}{\frac{ICDF}{(g^2-1)g} + \frac{1}{(1+g^2+2g)^{1/2}}}\right)^2\right)$$

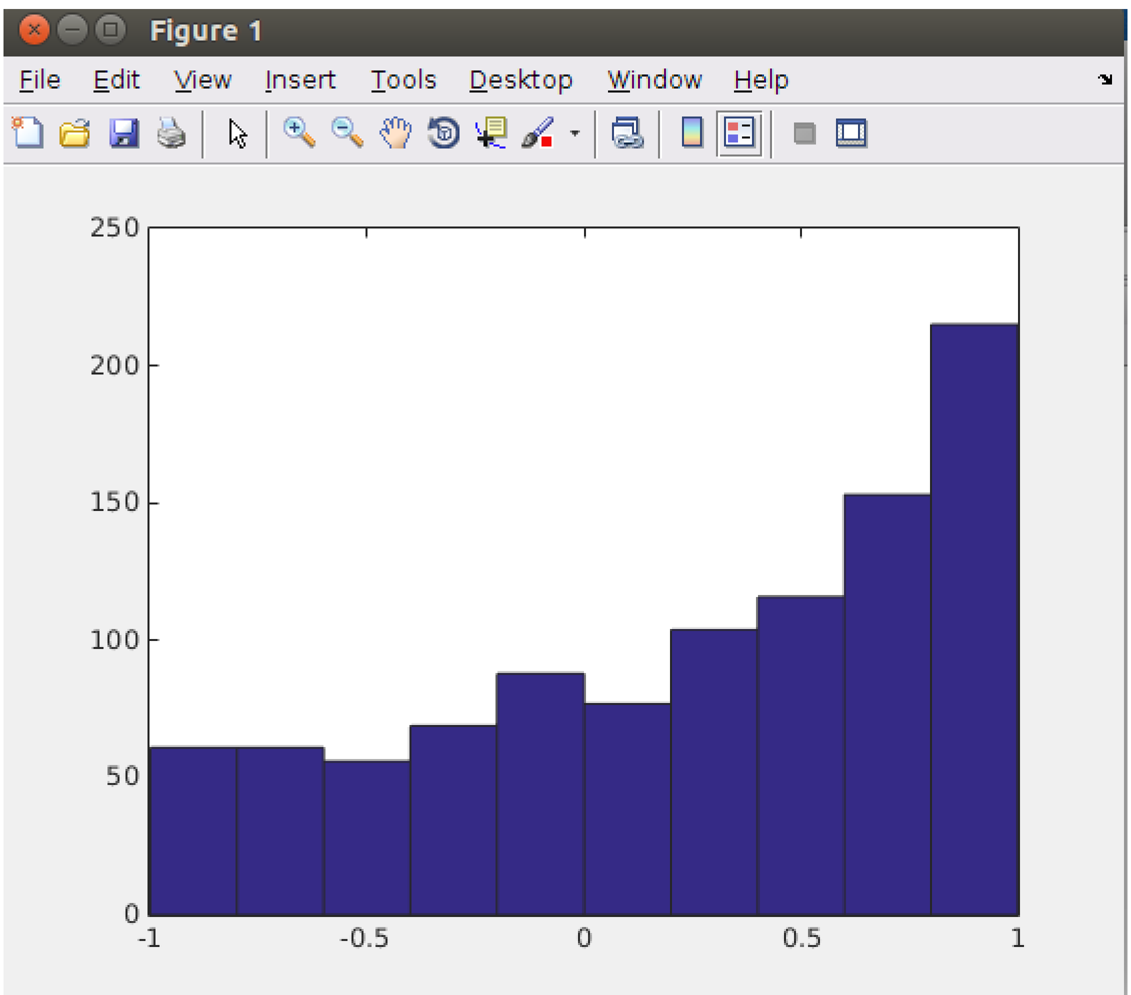
4e)

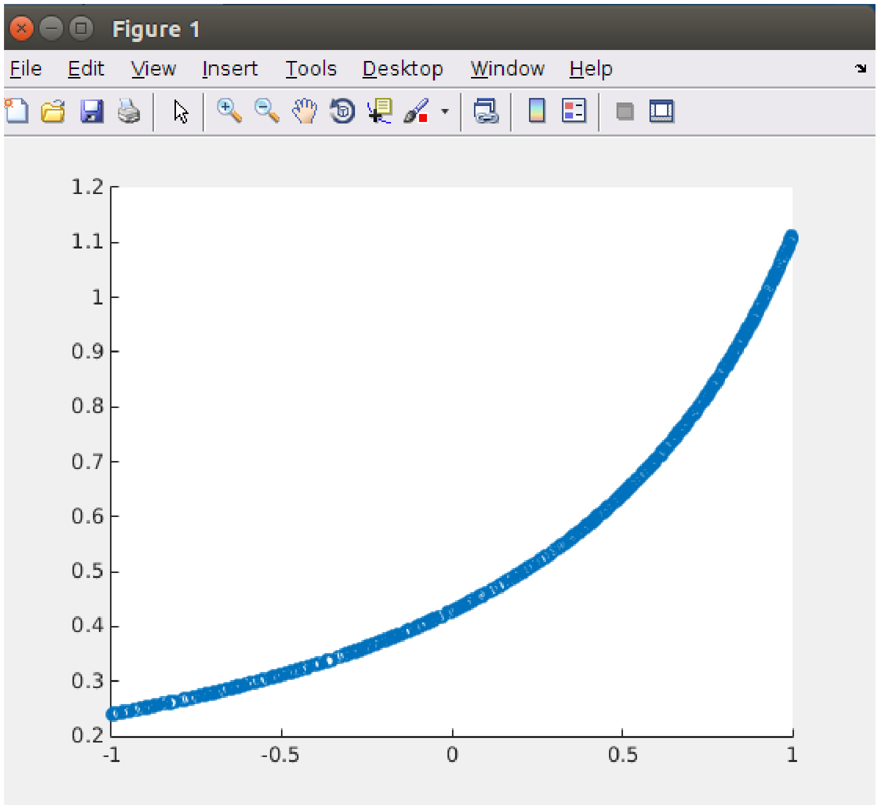
```
function [ y ] = henyey(g )
    cdf = rand(1) ;
    inner = ((cdf*2*g/(1-g^2)) + (1+g^2+2*g)^-0.5)^-2;
    y = (1+ g^2 - inner)/(2*g);
end
```

4f)

```
function [x,y]  = henyey_rand( g, samples )
    for i=1:samples
        x(i) = henyey(g);
        numerator = 1-g*g;
        denominator =  (1+g*g-2*g*x(i))^1.5;
        product = numerator/denominator;
        value = 0.5*product;
        y(i) = value;
    end
end
```

```
>> hist(x)
>> scatter(x,y)
```





QUESTION 3:

3a)

3a) Given that $\left\| \begin{pmatrix} y \\ 0 \end{pmatrix} - \begin{pmatrix} X \\ T \end{pmatrix} c \right\|^2$ should be min.

Eqn 1: $y - Xc = 0$

$y = Xc$

Normal eqn = $X^T y = X^T X c \rightarrow (1)$

Eqn 2: $0 - Tc = 0$

$0 = Tc$

Normal eqn = $T^T 0 = T^T T c$

$\Rightarrow 0 = T^T T c \rightarrow (2)$

Normal Eqn => $X^T y + 0 = X^T X c + T^T T c$

$X^T y = (X^T X + T^T T) c$

Solution for c => multiplying both ends by $(X^T X + T^T T)^{-1}$

=> $(X^T X + T^T T)^{-1} X^T y = (X^T X + T^T T)^{-1} (\cancel{X^T X} + T^T T) c$

or $c = (X^T X + T^T T)^{-1} X^T y$.

Pseudoinverse : for eqn $Ax = b$ $x = pinv(A) \cdot b$

Similarly PINV in this case = $(X^T X + T^T T)^{-1} X^T$.

3b)

3b). Singular values of $XX^T + TT^T$

Let $A = XX^T + TT^T = XX^T + \lambda I$.

If $X = USV^T$ then $X^T = VSU^T$

$XX^T = US^2U^T$

Singular value of $\lambda I = \lambda$.

So singular values of $XX^T + TT^T = S^2 + \lambda$.

Hat matrix $= X(XX^T + \lambda I)^{-1} X^T$

$= X X^{-1}(X^T)X^{-1T} + X \frac{1}{\lambda} I X^T$

$= I + \frac{1}{\lambda} I XX^T$ If $X = USV^T$ $XX^T = US^2U^T$

$= I + \frac{1}{\lambda} I US^2U^T$

Singular values $= I + \frac{1}{\lambda} I S^2 = I(1 + \frac{1}{\lambda} S^2)$

Special case of Hat Matrix when $\lambda = 0$.

$$\text{Hat matrix} = X(XX^T + 0I)^{-1}X^T = X(XX^T)^{-1}X^T$$

$$= XX^{-1}X^{T^{-1}}X^T = I \cdot I = I$$

Singular values of hat $= I$.

3d)

3d) $df(\lambda) = \text{trace}(H_\lambda)$

From 3c singular values of Hat Matrix $= I(1 + \frac{S^2}{\lambda})$

If $X = USV^T$ and $S = \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \\ & & & \sigma_n \end{bmatrix}$ then the

Singular values of Hat Matrix $= (\sigma_k)^2/_\lambda + 1$ for $1 \le k \le n$

Trace = Sum of singular values

$$= \sum_{k=1}^{N} 1 + (\sigma_k)^2/_\lambda = \sum_{k=1}^{N} 1 + \frac{1}{\lambda}\Sigma\sigma_k^2$$

$$= n + \frac{1}{\lambda}\Sigma\sigma_k^2$$

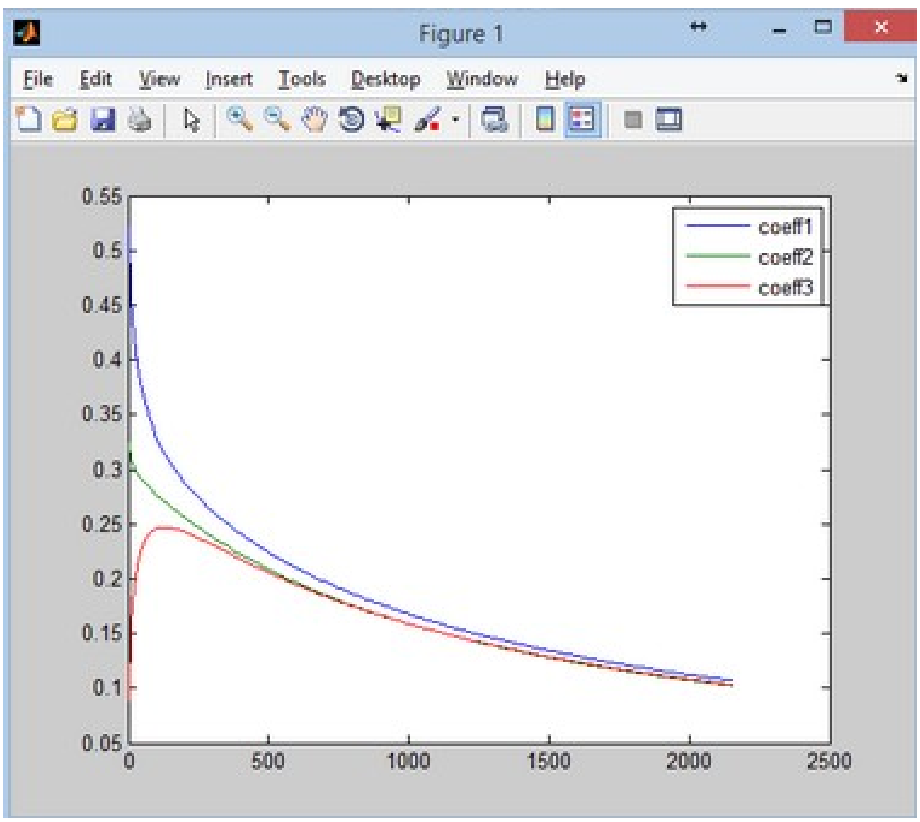
from 3c) In the case of $\lambda = 0$ the singular value matrix S of the hat matrix $= I$

So trace $df(\lambda = 0) =$ trace $(I) = n$.

3c)

```
function [lamda,c] = ridge(autos)
    one = ones(389);
    autos(:,1) = 1./autos(:,1);
```

```
    autos = zscore(autos);
    x = [autos(:,5), autos(:,4),autos(:,3)];
    limit = 2*norm(x'*x);
    increment = limit/200;
    i = 0;j=1;
    while(i<=limit)
        y = autos(:,1);
        pseudoinv = inv(x'*x+i*eye(3));
        pseudoinv = pseudoinv * x';
        c(:,j) = pseudoinv*y;
        lamda(j) = i;
        j = j+1;
        i = i+increment;
    end
end

>>[lamda,c] = ridge(autos);
>>plot(lamda,c(1,:);
>>hold all;
>>plot(lamda,c(2,:);
>>plot(lamda,c(3,:);
>>legend('coeff','coeff2',coeff3');
```



QUESTION 2:

2a)

- The timestep dt is defined in the variable deltaT in the template file. It is given a value of 0.01.
- The only force acting on the particles is the gravity. It is represented

by the vector : gravity = [0; -9.81];

2b)

```
for I = 1:particlesCount
    % Notice that both position and velocity are 2x1 vectors.
    % **********************************************************************
    posX = (maxX-minX)*rand + minX;
    posY = (maxY-minY)*rand + minY;
    particles(I).position = [posX;posY]; % TODO: Random position within the
boundaries minX to maxX, and minY to maxY.
    particles(I).velocity =(maxV-minV).*rand(2,1) + minV; % TODO: Random
velocity within the range minV to maxV for both x and y components.
    particles(I).color = rand(3,1);    % TODO: Random color triplet: [R, G,
B] between 0 and 1 each.
    % **********************************************************************
end;
```

2c)

```
oldV = particles(I).velocity;
particles(I).velocity = particles(I).velocity + deltaT .* gravity; % TODO:
integrate acceleration to get new velocity.

% Integrate velocity to find new position.
particles(I).position = particles(I).position + deltaT .* oldV;
```

2d)

```
fun = @(x)2*(particles(I).position(1)-x)+ 2*(particles(I).position(2)-
sin(x)) - cos(x);
xSol = fzero(fun,1);
particles(I).position = [xSol; sin( xSol )];
```

Entire code :

```
% Code to generate a particle system that simulates collisions with a sine
% curve, using the Matlab builtin fzero function.

% Boundaries in the plot.
minX = -10;
maxX = +10;
minY = 0;
maxY = +20;

% Initial velocity limits.
minV = -5;
maxV = +5;    % Apply for both x and y components.

% Particles array.
particlesCount = 5;
for I = 1:particlesCount
    % Notice that both position and velocity are 2x1 vectors.
    % **********************************************************************
    posX = (maxX-minX)*rand + minX;
    posY = (maxY-minY)*rand + minY;
    particles(I).position = [posX;posY]; % TODO: Random position within the
boundaries minX to maxX, and minY to maxY.
    particles(I).velocity =(maxV-minV).*rand(2,1) + minV; % TODO: Random
velocity within the range minV to maxV for both x and y components.
```

```
    particles(I).color = rand(3,1);    % TODO: Random color triplet: [R, G,
B] between 0 and 1 each.
    % **********************************************************************
end;

% The only force acting on particles is gravity. This is the constant g.
gravity = [0; -9.81];

% Coefficient of restitution to handle Newtonian collisions; modify this to
% get different behaviors.
restCoeff = 0.5;    %0 - inelastic, 1 - fully elastic.

% The ground for collisions (a sinusoidal function).
xSine = linspace( minX, maxX, 100 );
ySine = sin( xSine );

% The optimization parameters.
x0 = 0;         %Initial search point.

% Define time control variables.
deltaT = 0.01;
simulationTime = 0.0;
maxSimulationTime = 10.0;

% Set up window.
set( 0, 'Units', 'pixels' );                 %Set default units to pixels.
screenSize = get( 0, 'ScreenSize' );         %Get screen size and position.
figure( 'Renderer', 'OpenGL', ...
    'OuterPosition', [screenSize(3)/2-350 screenSize(4)/2-350 700 700]);

%create a viedo
%nFrames = maxSimulationTime/deltaT;
writerObj = VideoWriter('peaks.avi');
open(writerObj);

% Begin simulation.
while( simulationTime < maxSimulationTime )
    plot( xSine, ySine, 'Color', 'black', 'LineWidth', 2 );   % Draw
sinusoidal ground.
    hold on;

    % Iterate over each particle.
    for I = 1:particlesCount       %Solve for each particle: find its new
position.

        % ****************************************************************
        % Integrate acceleration to find new velocity.
        oldV = particles(I).velocity;
        particles(I).velocity = particles(I).velocity + deltaT .* gravity;
% TODO: integrate acceleration to get new velocity.

        % Integrate velocity to find new position.
        particles(I).position = particles(I).position + deltaT .* oldV; %
TODO: integrate velocity to get new 'candidate' position.
        %
*****************************************************************deltaT .*
particles(I).velocity; % TODO: integrate velocity to get new 'candidate'
position.
        % ****************************************************************

        % Check collision with ground.
```

```
        if( particles(I).position(2) < sin( particles(I).position(1) ) ) %
Is particle below ground?

            % Move particle above the ground, to the location on the sine
            % function where the distance between the current position and
            % the curve is minimal.

            % ***********************************************************
            % TODO: Use "fzero" to minimize your distance function.
            % xSol should be the x-coordinate that minimizes your distance
            % function.
            % ***********************************************************
            fun = @(x)2*(particles(I).position(1)-x)+
2*(particles(I).position(2)-sin(x)) - cos(x);
            xSol = fzero(fun,1);
            particles(I).position = [xSol; sin( xSol )];    % Move particle
on the ground.

            % Compute a new velocity based on the normal at the point where
            % the particle went through the ground (found in previous
step).
            commonFactor = 1/sqrt( 1 + cos( xSol )^2 );
            groundNormal = commonFactor * [ -cos( xSol ); 1 ];
            projOntoN = particles(I).velocity' * groundNormal;
            particles(I).velocity = particles(I).velocity - ...
                projOntoN*groundNormal - restCoeff*projOntoN*groundNormal;
        end;

        % Draw particle at its new position.
        plot( particles(I).position(1), particles(I).position(2), ...
            'Marker', 'o', ...
            'MarkerEdgeColor', particles(I).color*0.5, ...
            'MarkerSize', 10, ...
            'MarkerFaceColor', particles(I).color );

    end;
    xlabel( sprintf( 'Simulation time: %f', simulationTime ) );
    axis( [minX, maxX, minY-2, maxY] );                    % Give some room to
sinusoidal ground.
    hold off;

    % Instruct MatLab to refresh plot.
    drawnow;

    %frame(k) = getframe;
    %k = k + 1;
    frame = getframe;
    writeVideo(writerObj,frame);
```

```
    simulationTime = simulationTime + deltaT;           % Advance time.
end;

%close the vedio writer
%movie2avi(mov, 'myPeaks.avi', 'compression', 'None');
close(writerObj);
```