Statistics 572
Homework 5

Alejandro Robles

October 30th, 2018

1 Kernel Density Estimate with Geyser Data

Generate the following (using $\hat{\sigma}$) :

1. Kernel Density Estimation with Normal Kernel
2. Kernel Density Estimation with Epanechnikov
3. Relative frequency histogram and superimpose the estimated densities on the same graph

MATLAB Code:

```
load geyser; data = geyser; n = 1000; % Data
[normal, epan, xx] = kernel(data, n, false); % Calling UDF

% Plot Relative Frequency Histogram,Normal, and Epanechnikov Kernels
[cnt,data]=hist(data,20);
bar(data,cnt/n,1); title('Normal Kernel'); hold on;
plot(xx,normal, 'red', xx,epan, 'green')
legend('Relative Frequency Histogram','Normal','Epanechnikov')
hold off
```

Discussion: As we can see the relative frequency closely matches both kernel density estimations from Normal and Epanechnikov. Also, we can see that both kernel estimates are very similar.

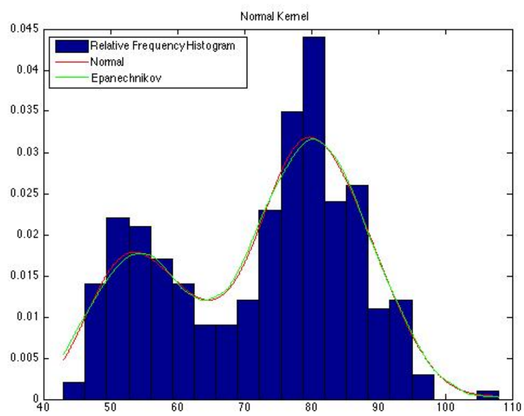


Figure 1: Kernek Density with Geyser Data

2 Kernel Density Estimate with Exponential Distribution

Random sample from $Exp(\lambda = 5)$ of size 100 and treat this as your raw data.

Generate the following :

1. Kernel Density Estimation with Normal Kernel
2. Kernel Density Estimation with Epanechnikov
3. Monte Carlo simulation to estimate MSE at $X_0 = 5$

MATLAB Code:

```
% Parameters
n = 100; lambda = 5; data = exprnd(lambda,n,1); size_sample = 1000;

% Using UDF to create kernels and plots
kernel(data,size_sample, true);

% ****** Monte Carlo simulation to estimate MSE at X0= 5 with Normal Kernel *******
x0= 5; M=200; fhat_normal = zeros(M,1);

for trials = 1:M;
data=exprnd(lambda,n,1);
h_normal = 1.06*1^(-1/5)*std(data);
    fnorm = exp(-(1/(2*h_normal^2))*(x0-data(1)).^2)/sqrt(2*pi)/h_normal;
    fhat_normal(trials,:) = fhat_normal(trials,:) + fnorm/(1);
end

MSE = var(fhat_normal)
fprintf('At x0 = %g, the MSE =  %g ',x0, MSE);
```

Result:

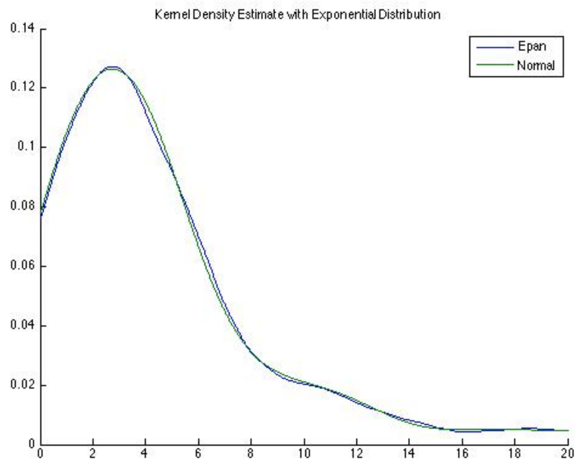


Figure 2: Kernek Density with $Exp(\lambda = 5)$

```
At x0 = 5, the MSE =  0.000407296
```

Discussion: As we can see both Kernels produce similar densities and the MSE is very low for Normal Kernel density.

3 Finite Mixture

1. Create artificial 3-term mixed data with size 1500.
   - 200 from $N(5, 3^2)$
   - 800 from $N(10, 1.5^2)$
   - 500 from $N(15, 2^2)$
2. Find FM estimates using csfinmix (use your own initials).
3. Generate random sample of size 1500 from the Finite Mixture model.
4. Draw the density histogram of data in 1 with the Finite Mixture estimate superimposed.
5. Draw the density histogram of the random sample in 3 and compare with the one in 4.
6. Repeat 3 and 4 until converge.

MATLAB Code:

```
% Parameters
n = [200 800 500]; mus = [5, 10 15];
sigmas = [3 1.5 2]; total_size = sum(n);

% Initialize data
data = zeros(total_size,1);

% Create artificial 3-term mixed data with size 1500.
data(1:n(1)) = normrnd(mus(1),sigmas(1),n(1),1);
data(n(1)+1:n(1)+n(2)) = normrnd(mus(2),sigmas(2),n(2),1);
data(n(1)+n(2)+1:total_size) = normrnd(mus(3),sigmas(3),n(3),1);

% Initialize guesses
muin = [2 7 12]; piesin = [0.333 0.333 0.333];
varin = [1 1 1]; max_it = 100; tol = 0.0001;

% Find FM estimates using csfinmix
[pies,mus,vars]= csfinmix(data,muin,varin,piesin,max_it,tol);

% Generating f-hat
banwith = 3;
step = 0.1;
xx = min(data) - banwith : step : max(data) + banwith ;
fhat = zeros(size(xx));
for i=1:3
    fhat = fhat+pies(i)*normpdf(xx,mus(i),sqrt(vars(i)));
end

% Generate random sample of size 1500 from the Finite Mixture model.
N=1500;
x = zeros(N,1);
r = rand(N,1);

% Find the number generated from component 1.
ind1 = length(find(r <= pies(1)));
ind2 = length(find(r <= pies(2)));
```

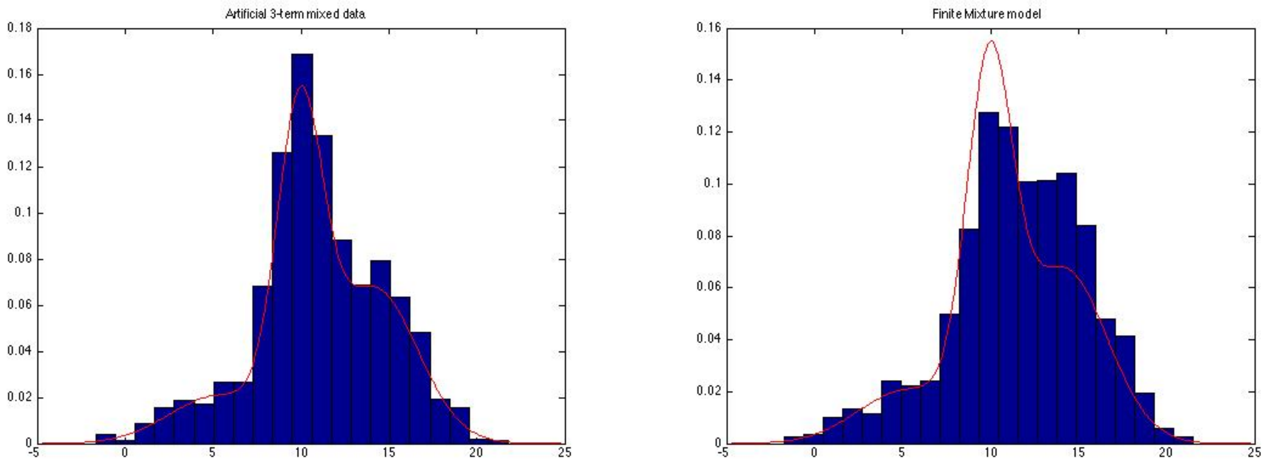
```
% Create mixture data
x(1:ind1) = normrnd(mus(1),sqrt(vars(1)),ind1,1);
x(ind1+1:ind2) = normrnd(mus(2),sqrt(vars(2)),ind2-ind1,1);
x(ind2+1:N) = normrnd(mus(3),sqrt(vars(3)),N-ind2,1);

% Plotting Original Data
numbins = 21;
figure(1)
[cnt,data]=hist(data,numbins);
bar(data,cnt/total_size,1)
title('Artificial 3-term mixed data')
hold on
plot(xx,fhat, 'red')
hold off

% Plotting From FM Estimates
figure(2)
[cnt,x]=hist(x,numbins);
bar(x,cnt/N,1)
title('Finite Mixture model')
hold on
plot(xx,fhat, 'red')
hold off

% Displaying final estimates
disp(mus);
disp(sqrt(vars));
```

Result:



(a) Artificial Data

(b) FM Model

Figure 3: Three Term Mixed Data

```
mus: 5.3618    9.9372   14.1860
sigmas: 2.8458    1.3083    2.4589
```

Discussion: As we can see, the Finite Mixture model estimates the true parameters very closely. From the figures we can see that the random samples follow the true densities. We can also identify 3 humps for each graph, which confirms what we expected.

4 Exercise 9.3

Generate 100 univariate normals and construct a histogram. Calculate the MSE at a point x_0 using Monte Carlo simulation. Do this for varying bin widths. What is the better bin width? Does the sample size make a difference? Does it matter whether x_0 is in the tails or closer to the mean? Repeat this experiment using the absolute error. Are your conclusions similar?

MATLAB Code:

```
% Parameters
n0 = 100; ns = [n0, n0*10];

% First Histogram
x0 = normrnd(0,1,n0,1);
nbins = 30;
[frequencies,binlocations] = hist(x0, nbins);
h = binlocations(2)- binlocations(1);
figure(1)
bar(binlocations,frequencies/(n0*h),1)

% Tails and Means
mean_data = mean(x0);
tail_data = mean_data + std(x0);
x0s = [mean_data, tail_data];

colors = ['r' 'b' 'g' 'y', 'k' 'm'];

% ******** MSE ************
counter = 1;
figure(2);
hold on;
for n = ns;
   for  x0 = x0s;
       mse_x0(n,x0, colors(counter), true) % Calling UDF
       lgd{counter} = sprintf('x0 = %g, n = %0.0f', x0, n);
       counter = counter + 1;
   end
end
legend(lgd)
ylabel('MSE')
xlabel('h')
hold off

% ******** Mean Absolute Error ************
counter = 1;
figure(3);
hold on;
for n = ns;
   for  x0 = x0s;
       mse_x0(n,x0, colors(counter), false) % Calling UDF
       lgd{counter} = sprintf('x0 = %g, n = %0.0f', x0, n);
       counter = counter + 1;
   end
end
legend(lgd)
```

```
ylabel('Mean Absolute Error')
xlabel('h')
hold off


% ****** Plotting *********
% sample size
n = 100;
% M monte carlo trials
M = 100;

% Data
x0 = normrnd(0,1,n,1);
mean_data = mean(x0);
tail_data = mean_data + 2*std(x0);

% Calling UDF
mc_erros(n, M,[mean_data, tail_data])
```

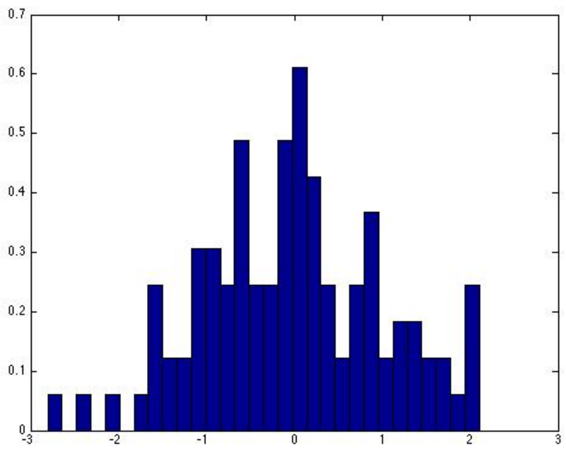
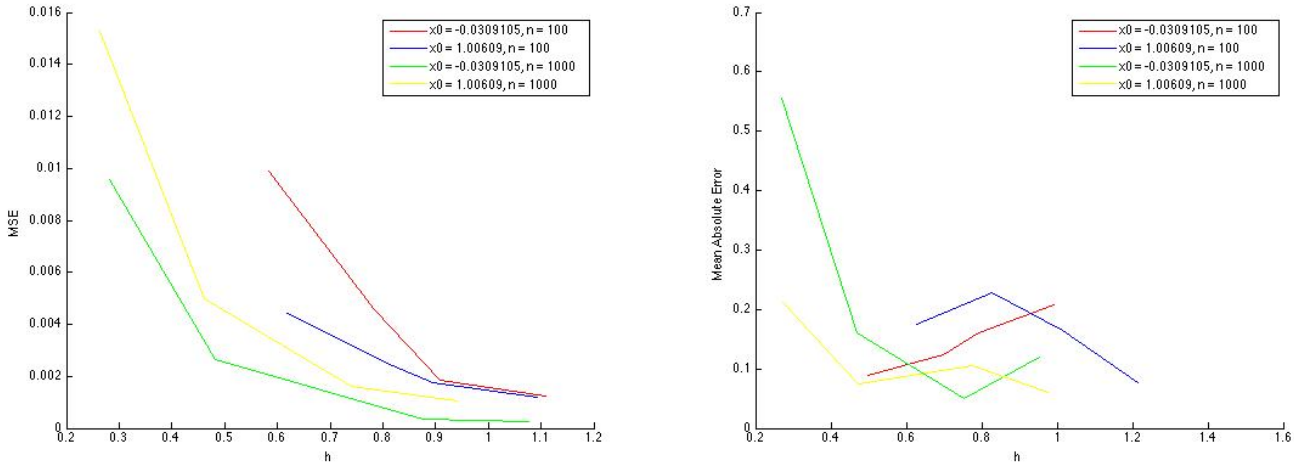


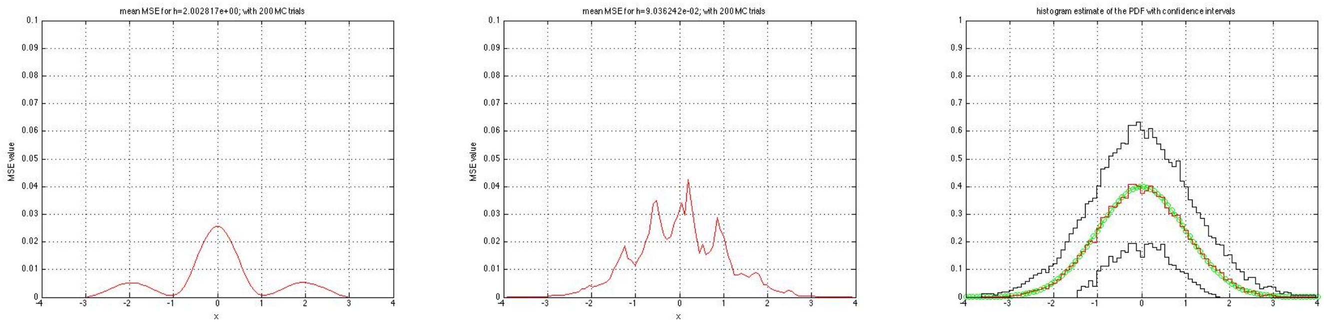
Figure 4: Histogram of random sample for Standard Gaussian



(a) MSE

(b) Average Absolute Error

Figure 5: Plots with respect to h



(a) h = 2

(b) h = 0.09

(c) Multiple Histograms

Figure 6: Plots with respect for various h

```
The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = -0.0309105 and n = 100:
    0.5842    0.7842    0.9090    1.1090
    0.0099    0.0046    0.0018    0.0012
    0.0884    0.1239    0.1617    0.2069

The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = 1.00609 and n = 100:
    0.6183    0.8183    0.8938    1.0938
    0.0044    0.0024    0.0017    0.0012
    0.1815    0.2349    0.1712    0.0850

The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = -0.0309105 and n = 1000:
    0.2816    0.4816    0.8758    1.0758
    0.0096    0.0027    0.0003    0.0002
    0.5507    0.1560    0.0504    0.1218

The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = 1.00609 and n = 1000:
    0.2626    0.4626    0.7414    0.9414
    0.0153    0.0050    0.0016    0.0010
    0.2138    0.0753    0.1062    0.0586
```

```
The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = -0.0309105 and n = 100:
    0.4991    0.6991    0.7916    0.9916
    0.0109    0.0050    0.0016    0.0011
    0.0895    0.1240    0.1608    0.2072

The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = 1.00609 and n = 100:
    0.6259    0.8259    1.0151    1.2151
    0.0047    0.0025    0.0017    0.0012
    0.1744    0.2276    0.1643    0.0774

The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = -0.0309105 and n = 1000:
    0.2700    0.4700    0.7525    0.9525
    0.0106    0.0030    0.0004    0.0003
    0.5559    0.1608    0.0511    0.1194

The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = 1.00609 and n = 1000:
    0.2739    0.4739    0.7762    0.9762
    0.0137    0.0045    0.0015    0.0009
    0.2126    0.0739    0.1051    0.0595
```

Discussion:

From Figure 5(a), we see that in general as bin width increases, the MSE goes down. Also, as n increases the mean tends to do better than the tails but the opposite is true as n decreases.

From Figure 5(b), we see that in general as bin width the mean absolute error tends to decrease with large sample size. Also, as n increases the tails tends to do better than the mean but the opposite is true as n decreases.

Overall, comparing Figure a with b, if we fix n, then tails and means seem to alternate trends.

In figure 6, we see that for small values of h we see that the mean pdf estimate is close to the true value. On the other hand, large h gives poor estimates of the true pdf but the variance of the estimates are much more precise.

5 Exercise 9.19

Using the method for generating random variables from a finite mixture that was discussed in this chapter, develop and implement an algorithm for generating random variables based on a kernel density estimate.

MATLAB Code:

```
% Load in data
load geyser
data = geyser;

% Parameters
n = length(data);
N = 1000;
sample_sigma = std(data);
min_x = mean(data) - 4 * sample_sigma;
max_x = mean(data) + 4 * sample_sigma;
h = 1.06*n^(-1/5)*sample_sigma;

% Density estimates at these x values.
x = linspace(min_x,max_x,N);
fhatnorm = zeros(1, N);
fnorm_all = zeros(1, n);

% Kernel function evaluated at x, centered at each data point
for i=1:n
    fnorm=exp(-(1/(2*h^2))*(x-data(i)).^2)/sqrt(2*pi)/h;
    fhatnorm = fhatnorm + fnorm/n;
end

% Step 3 + 4
mu = 0; sigma = 1;
r = unifrnd(mu,sigma,n,1);
u_frequencies=zeros(n,1);

for i=1:n
    left=(1/n)*(i-1);
    right=(1/n)*(i);
    u_frequencies(i)=length(find(r>=left & r<=right));
end

% Step 5
index=1;
sampled_data=zeros(n,1);
for i=1:n % Here we use pies = 1/n, mu = each data point , std = h
    sampled_data(index:index+u_frequencies(i)-1) = normrnd(data(i),h,u_frequencies(i),1);
    index = index + u_frequencies(i);
end

% Plotting data
nbins = 20;
[frequencies,bin_locations]=hist(sampled_data,20);
bar(bin_locations,frequencies/(n*h),1)
title('Finite Mixtures Viewed as a Kernel Estimate')
```

```
hold on
plot(x,fhatnorm, 'red')
hold off
```

Result:

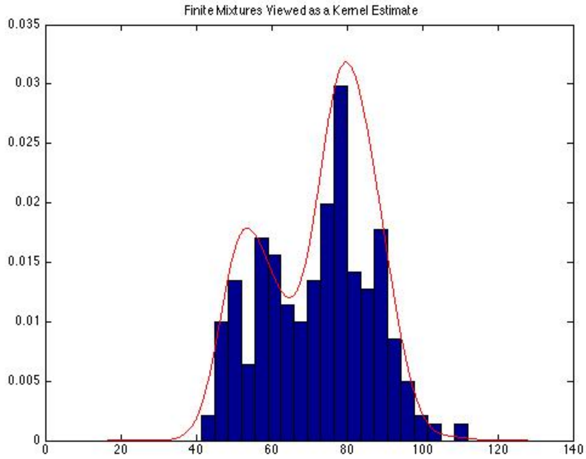


Figure 7: Finite Mixtures Viewed as a Kernel Estimate

Discussion: If the kernel density estimate is viewed as a finite mixture model the mixture components, means, and variances are given by $p_i = \frac{1}{n}, m_i = x_i, \sigma_i = h$. And so we see that kernel density estimate is a special case of finite mixture. From the figure, we see that this procedure generate random samples close to the finite mixture density.

User Defined Functions:

@kernel: Uses both Normal and Epanechnikov to create create kernel density estimates with option to plot

```
% ***** Kernel Function ************
function [normal, epan, x] = kernel(data, size_sample, plot_bool)
    % ******* Using Normal Kernel ********
    n = length(data);
    x = linspace(min(data),max(data),size_sample);
    fhat_normal = zeros(size(x));
    h_normal = 1.06*n^(-1/5)*std(data);

    % Using normal kernel function evaluated at centered at data
    for i=1:n
        fnorm = exp(-(1/(2*h_normal^2))*(x-data(i)).^2)/sqrt(2*pi)/h_normal;
        fhat_normal = fhat_normal+fnorm/(n);
    end

    % ******* Using Epanechnikov Kernel ********

    h_epan = h_normal*(30*sqrt(pi))^(1/5);
    fhatepan = zeros(size(x));
    fepan=zeros(1,n);


    for i=1:n
        domain=((x-data(i))/h_epan);
        for j=1:length(domain)

            if abs(domain(j))<=1
                fepan(j)=3*(1-((x(j)-data(i))/h_epan).^2)/(4*h_epan);
            else
                fepan(j)=0;
            end

        end
        fhatepan=fhatepan+fepan/(n);
    end

    if plot_bool
        hold on
        plot(x,fhatepan,x,fhat_normal);
        legend('Epan','Normal')
        title('Kernel Density Estimate with Exponential Distribution')
        hold off
    end

    normal = fhat_normal;
    epan = fhatepan;
```

@$find_h$: Finds optimal h using Freedman-Diaconis and Sturge's rule

```
function [freedman_diaconis, sturges] = find_h(data, plot_bool)
    % the freedman-diaconis rule is:
    n = length(data);
    hFD = 2*iqr(data)*n^(-1/3);
    h = hFD;

    % get the limits, bins, bin centers etc:
    x_lim_left = min(data)-1;
    x_lim_rght = max(data)+1;

    t0   = x_lim_left;
    tm   = x_lim_rght;
    rng  = tm-t0;
    nbin = ceil(rng/h);
    bins = t0:h:(nbin*h+t0);       % <- the bin edges ...
    bc   = bins(1:end-1)+0.5*h;    % <- the bin centers ...

    vk=histc(data,bins); vk(end)=[];
    fhat = vk/(n*h); % normalize:

    if plot_bool
        fh=figure;
        ah1=stairs( bins, [fhat,fhat(end)], '-r' ); grid on; %axis( [x_lim_left, x_lim_rght, 0, +Inf
        xlabel('spatial variable'); ylabel('probability distribution');

    end
    % now use Sturge's rule:
    nbin = 1 + log2(n);
    hS   = rng/nbin;           % <- compute the width depending on the number of bins
    h    = hS;
    bins = t0:h:(nbin*h+t0);       % <- the bin edges ...
    bc   = bins(1:end-1)+0.5*h;    % <- the bin centers ...

    vk=histc(data,bins); vk(end)=[];
    fhat = vk/(n*h); % normalize:
    if plot_bool
        figure(fh); hold on;
        ah2=stairs( bins, [fhat,fhat(end)], '-b' ); grid on; %axis( [x_lim_left
        legend( [ah1,ah2], {'freedman-diaconis rule','sturges rule'}, 'location', 'best' );
    end
    freedman_diaconis = hFD;
    sturges = hS;
```

@mse_{x0}: Gets band width, MSE, or mean absolute error with plotting

```
function [h, mse, mean_abs_error] = mse_x0(n,x0, color, mse_bool)
    fhat = zeros(4,n);
    freal = zeros(4,n);
    for i=1:n
      x = normrnd(0,1,1,n);
    % Get the histogram-default is 10 bins.
     [vk,bc] = hist(x);
    % Get the bin width.
       [h1, h2] = find_h(x, false);
       h = [h1; h1+0.2; h2; h2 + 0.2];
    % Find all of the bin centers less than xo.
        ind = find(bc < x0);
    % xo should be between these two bin centers.
        b1 = bc(ind(end));
        b2 = bc(ind(end)+1);
    % Put it in the closer bin.
        if (x0-b1) < (b2-x0) % then put it in the 1st bin
            fhat(:,i) = vk(ind(end))./(n*h);
        else

            fhat(:,i) = vk(ind(end)+1)./(n*h);
        end
        freal(:, i) = normpdf([x0-b1; b2-x0; b1; b2],0,1);
        MSE=var(transpose(fhat));
    end
        mean_abs_error = transpose(mean(abs(fhat - freal),2));
        fprintf('The H, MSE, and Mean Absolute Error of the bindwiths, respectively for x0 = %g and
        disp(transpose(h));
        disp(MSE);
        disp(mean_abs_error);
        h = transpose(h);
        mse = MSE;
        if mse_bool
            plot(h, mse, color)
        else
            plot(h, mean_abs_error, color)
        end
```

@mc_{erros}: MSE and mean absolute error with plotting

```
function [] = mc_erros(sample_size, m_trials, bandwidths)
    average_absolute_error = 0;
    for hi=1:length(bandwidths)
      %h = 0.1;          % <- an example bin width
      h = bandwidths(hi);
      % specify the bin widths:
      xLimitLeft = -4;
      xLimitRight = +4;
      t0   = xLimitLeft;
      tm   = xLimitRight;
      range  = tm-t0;
      nbin = ceil(range/h);
      bins = t0:h:(nbin*h+t0);       % <- the bin edges
      bc  = bins(1:end-1)+0.5*h;     % <- the bin centers
      % save each monte-carlo trial estimate of the probability distribution:
      all_fhats = zeros(m_trials,length(bins)-1);
      % save each monte-carlo trial estimate of the mean square error (we evaluate the MSE on a grid
      xMSE = linspace(xLimitLeft,xLimitRight,100);
      all_mse = zeros(m_trials,length(xMSE));
      for mci=1:m_trials
        x = randn(1,sample_size);
        x(find(x < xLimitLeft)) = xLimitLeft;
        x(find(x > xLimitRight)) = xLimitRight;
        vk=histc(x,bins);
        vk(end)=[];
        % normalize
        fhat = vk/(sample_size*h);
        all_fhats(mci,:) = fhat;
        % record the MSE of this approximate PDF
        fhat_interp = interp1(bc,fhat,xMSE);
        if( ~average_absolute_error )
          all_mse(mci,:) = (fhat_interp-normpdf(xMSE,0,1)).^2;
        else
          all_mse(mci,:) = abs(fhat_interp-normpdf(xMSE,0,1));
        end
      end

      % plot the last PDF estimate produced above
      if( 0 )
        fh = figure;
        plot( bc, normpdf(bc,0,1), '-go' );
        hold on;
        stairs( bins, [fhat,fhat(end)], '-r' );
        axis( [x_lim_left,x_lim_rght,0,0.45] );
        title( 'the last Monte-Carlo PDF estimate' );
      end

      % plot the mean PDF estimate and one standard deviation confidence intervals:
      if( 1 )
        mfh = mean(all_fhats);
        sfh = std(all_fhats);
        fh = figure;
        plot( xMSE, normpdf( xMSE, 0, 1 ), '-go' );
        hold on;
```

```
    stairs( bins, [mfh,mfh(end)], '-r' );
    tmp = max(mfh-sfh,0); stairs( bins, [tmp,tmp(end)], '-k' ); tmp=mfh+sfh; stairs( bins, [tmp,
    axis( [xLimitLeft,xLimitRight,0,1.0] );
    title( 'histogram estimate of the PDF with confidence intervals' );
  end

  % plot the expected MSE over all of these monte-carlo's:
  if( 1 )
    figure; plot( xMSE, mean(all_mse), '-r' ); grid on;
    xlabel( 'x' ); ylabel( 'MSE value' );
    title( sprintf('mean MSE for h=%e; with %d MC trials',h,m_trials) );
    axis( [xLimitLeft,xLimitRight,0,0.1] );
  end
end
```