

CSE - 411

Student ID: 1505112

November 12, 2020

1 Input

Independent observations on deviation from the desired diameter of ball bearings produced by a new high speed machine. These observations are continuous data.

2 Hypothesizing a Distribution Form

Different techniques like Summary statistics and Histograms are used to hypothesize a distribution form.

2.1 Summary Statistics

Maximum	4.00
Minimum	-1.72
Mean	1.225
Median	1.220
Variance	0.844
Co-efficient of Variance	0.750
Skewness	-0.208

For a symmetric continuous distribution (e.g. normal), the mean μ is equal to the median.

Here, the estimates mean and median are almost equal, this is some indication that the underlying distribution may be symmetric.

2.2 Histograms

For a continuous data set, a histogram is essentially a graphical estimate of the plot of the density function corresponding to the distribution of our data X_1, X_2, \dots, X_n . To make a histogram, we break up the range of values covered by the data into k disjoint adjacent intervals $[b_0, b_1), [b_1, b_2), \dots, [b_{k-1}, b_k]$. All the intervals should be the same width, say, $\Delta b = b_j - b_{j-1}$. For $j = 1, 2, \dots, k$, let h_j

be the proportion of the X_i 's that are in the j th interval $[b_{j-1}, b_j)$. We define the function,

$$h(x) = \begin{cases} 0 & \text{if } x < b_0 \\ h_j & \text{if } b_{j-1} \leq x < b_j \quad j = 1, 2, \dots, k \\ 0 & \text{if } x > b_k \end{cases} \quad (1)$$

which we plot as a function of x .

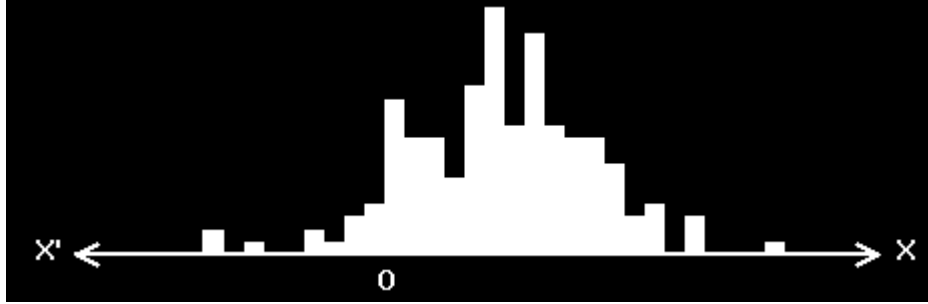


Figure 1: Histogram of errors with $\Delta b = 0.2$

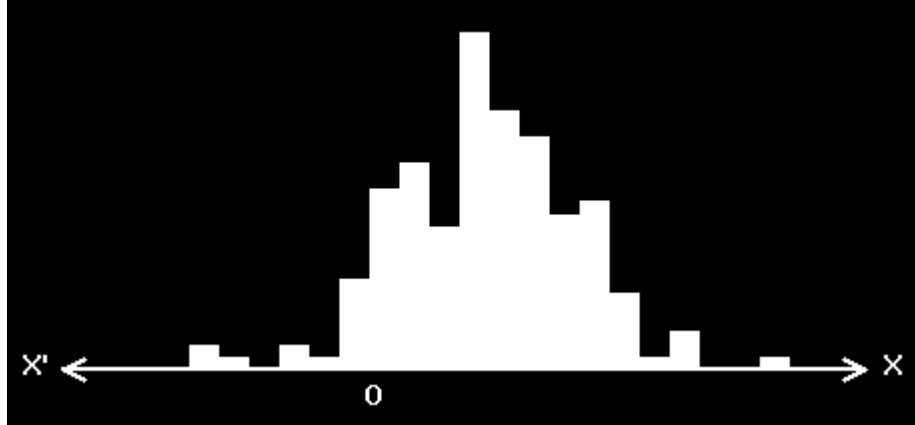


Figure 2: Histogram of errors with $\Delta b = 0.3$

As mean and median are almost equal and the shapes of histogram are likely normal distribution, all of these indicates that the distribution is normal distribution.

3 Estimation of Parameters

Maximum Likelihood Estimator (MLE) is used to estimate parameters.

The likelihood function is

$$L(\mu, \sigma) = \prod_{i=1}^n f_X(x_i, \mu, \sigma) \quad (2)$$

$$= \prod_{i=1}^n (2\pi\sigma)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \frac{(x_i - \mu)^2}{\sigma^2}\right) \quad (3)$$

$$= (2\pi\sigma)^{-\frac{n}{2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2\right) \quad (4)$$

The maximum likelihood estimators of the mean and the variance are

$$\bar{\mu} = \bar{X}(n) = \frac{1}{n} \sum_{i=1}^n x_j \quad (5)$$

$$\bar{\sigma} = \left(\frac{1}{n} \sum_{i=1}^n (x_i - \bar{\mu})^2\right)^{\frac{1}{2}} \quad (6)$$

Using these estimators, we can calculate estimated values of μ and σ .
Estimated parameters:

Parameter	Value
μ	1.225
σ	0.919

4 Goodness of fit

4.1 Density-Histogram Plots and Frequency Comparisons

Here, bars indicate frequency distribution and curve indicates density distribution.

4.2 Chi-Square Tests

$$\chi^2 = \sum_{i=1}^k \frac{(N_i - np_i)^2}{np_i} \quad (7)$$

Here, $\chi^2 = 165.8701$

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<math.h>
4 #include<string.h>
5
6 #include<graphics.h>
7 #include<conio.h>
```

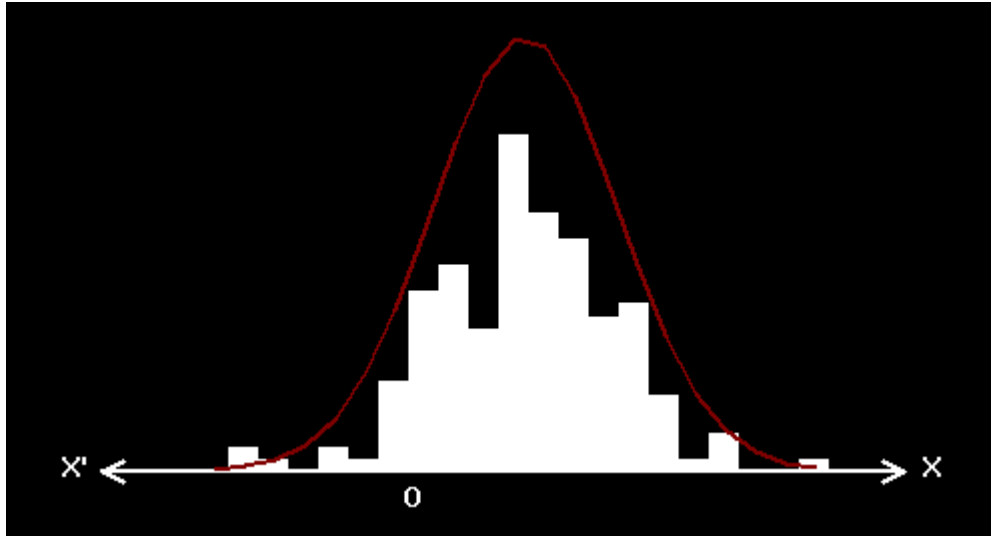


Figure 3: Histogram of density function and frequency with $d = 0.3$

```

8
9 #define PI 3.1416
10
11 void histogram(float *errorSample, int sampleNum, float d){
12     int maxInterval = 100;
13     int i, j;
14     float intervalArr[maxInterval+1], densityArr[maxInterval];
15     int freqArr[maxInterval], intervalNum = 0;
16     // making array of end points of interval; initializing freqArr
17     intervalArr[0] = errorSample[0] - d;
18     while(intervalArr[intervalNum] < errorSample[sampleNum-1]){
19         intervalArr[intervalNum+1] = intervalArr[intervalNum] + d;
20         freqArr[intervalNum] = 0;
21         intervalNum++;
22     }
23     // calculation of frequency
24     for(i = 0; i < sampleNum; i++){
25         for(j = 0; j < intervalNum; j++){
26             if(errorSample[i] < intervalArr[j+1]){
27                 freqArr[j] += 1;
28                 break;
29             }
30         }
31     }
32     printf("Interval number: %d\n", intervalNum);
33     // plot histogram
34     int gd = DETECT, gm; // VGA, gm = 0;
35     //initgraph(&gd, &gm, "C:\\TC\\BGI");
36     char pathtodriver[] = ""; // "C:\\TC\\BGI";
37     initgraph(&gd, &gm, pathtodriver);
38

```

```

39     int errorcode = graphresult();
40     if (errorcode != grOk)
41     {
42         printf("Graphics error: %s", grapherrormsg(errorcode));
43         return;
44     }
45
46     int left, top, right, bottom;
47     int x0 = 250, y0 = 400, xscale = 50, yscale = 1000, xleft =
150, xright=250;
48     setlinestyle(SOLID_LINE,0,2);
49
50     line(x0-xleft, y0, x0+xright, y0); // x-axis
51     line(x0-xleft, y0, x0-xleft+10, y0-5); // for arrows
52     line(x0-xleft, y0, x0-xleft+10, y0+5);
53     line(x0+xright-10, y0-5, x0+xright, y0);
54     line(x0+xright-10, y0+5, x0+xright, y0);
55     outtextxy(x0-xleft-20, y0-10, "X");
56     outtextxy(x0+xright+10, y0-10, "X");
57     outtextxy(x0, y0+5, "0");
58
59     for(i = 0; i<intervalNum; i++){
60         left = (x0 + xscale*intervalArr[i]);
61         top = (y0 - yscale*freqArr[i]/sampleNum);
62         right = (x0 + xscale*intervalArr[i+1]);
63         bottom = y0;
64         bar(left, top, right, bottom);
65     }
66     getch();
67     closegraph();
68 }
69
70 void densityFrequencyComp(float *errorSample, int sampleNum, float
mu, float sigma, float d){
71     int maxInterval = 100;
72     int i, j;
73     float intervalArr[maxInterval+1], densityArr[maxInterval];
74     int freqArr[maxInterval], intervalNum = 0;
75     // making array of end points of interval; initializing freqArr
[i]
76     intervalArr[0] = errorSample[0] - d;
77     while(intervalArr[intervalNum]<errorSample[sampleNum-1]){
78         intervalArr[intervalNum+1] = intervalArr[intervalNum]+d;
79         freqArr[intervalNum] = 0;
80         intervalNum++;
81     }
82     // calculation of frequency
83     for(i = 0; i<sampleNum; i++){
84         for(j = 0; j<intervalNum; j++){
85             if(errorSample[i] < intervalArr[j+1]){
86                 freqArr[j] += 1;
87                 break;
88             }
89         }
90     }
91     // calculation of probability from density function
92     // f(x) = (1/(2*pi*sigma^2)^1/2)*exp(-(x-mu)^2 / 2*sigma^2)

```

```

93     float temp, x;
94     for(i = 0; i<intervalNum; i++){
95         x = intervalArr[i] + d/2;
96         temp = exp(-pow(x - mu, 2) / (2*sigma*sigma));
97         densityArr[i] = temp / (sigma*sqrt(2*PI));
98     }
99     //printf("Interval number: %d\n", inum);
100    // plot histogram
101    int gd = DETECT, gm; // VGA, gm = 0; //initgraph(&gd, &gm, "C:\\TC\\BGI");
102    char pathtodriver[] = ""; //"C:\\TC\\BGI";
103    initgraph(&gd, &gm, pathtodriver);
104
105    int errorcode = graphresult();
106    if (errorcode != grOk)
107    {
108        printf("Graphics error: %s", grapherrormsg(errorcode));
109        return;
110    }
111
112    int left, top, right, bottom;
113    int x0 = 250, y0 = 400, xscale = 50, yscale = 1000, xleft =
114    150, xright=250;
115    setlinestyle(SOLID_LINE,0,2);
116
117    line(x0-xleft, y0, x0+xright, y0); // x-axis
118    line(x0-xleft, y0, x0-xleft+10, y0-5); // for arrows
119    line(x0-xleft, y0, x0-xleft+10, y0+5);
120    line(x0+xright-10, y0-5, x0+xright, y0);
121    line(x0+xright-10, y0+5, x0+xright, y0);
122    outtextxy(x0-xleft-20, y0-10, "X");
123    outtextxy(x0+xright+10, y0-10, "X");
124    outtextxy(x0, y0+5, "0");
125    // plot bar
126    for(i = 0; i<intervalNum; i++){
127        left = (x0 + xscale*intervalArr[i]);
128        top = (y0 - yscale*freqArr[i]/sampleNum);
129        right = (x0 + xscale*intervalArr[i+1]);
130        bottom = y0;
131        bar(left, top, right, bottom);
132    }
133    // plot probability line
134    setcolor(RED);
135    setlinestyle(SOLID_LINE,0,2);
136    yscale = 500;
137    for(i = 0; i<intervalNum-1; i++){
138        left = x0 + xscale*(intervalArr[i]+d/2);
139        top = y0 - yscale*densityArr[i];
140        right = x0 + xscale*(intervalArr[i+1]+d/2);
141        bottom = y0 - yscale*densityArr[i+1];
142        line(left, top, right, bottom);
143    }
144
145    getch();
146    closegraph();
147 }

```

```

148 void chiSquareTest(float *errorSample, int sampleNum, float d){
149     int maxInterval = 100;
150     int i, j;
151     float intervalArr[maxInterval+1], chiArr[maxInterval];
152     int freqArr[maxInterval], intervalNum = 0;
153     // making array of end points of interval; initializing freqArr
154     intervalArr[0] = errorSample[0] - d;
155     while(intervalArr[intervalNum]<errorSample[sampleNum-1]){
156         intervalArr[intervalNum+1] = intervalArr[intervalNum]+d;
157         freqArr[intervalNum] = 0;
158         intervalNum++;
159     }
160     // npj value
161     float pj = 1.0/intervalNum;
162     float npj = sampleNum*pj;
163     // calculation of frequency
164     for(i = 0; i<sampleNum; i++){
165         for(j = 0; j<intervalNum; j++){
166             if(errorSample[i] < intervalArr[j+1]){
167                 freqArr[j] += 1;
168                 break;
169             }
170         }
171     }
172     // calculation of chi-square test value
173     // (Nj - npj)^2 / npj
174     float totalVal = 0;
175     for(i = 0; i<intervalNum; i++){
176         chiArr[i] = (freqArr[i] - npj)*(freqArr[i] - npj) / npj;
177         totalVal += chiArr[i];
178         printf("%d    %d    %.3f    %.3f\n", i, freqArr[i], npj,
179             chiArr[i]);
180     }
181     printf("%.4f\n", totalVal);
182 }
183 int main(){
184     // open file to read data on errors in the diameter of ball
185     // bearings
186     FILE *file = fopen("errors-diameter.txt", "r");
187     if(file == NULL){
188         printf("File can not be opened");
189         return 0;
190     }
191     int bufferLength = 100;
192     int sampleNum = 154;
193     char buffer[bufferLength];
194     float errorSample[sampleNum];
195     char *token;
196
197     int idx = 0;
198     while(fgets(buffer, bufferLength, file)){
199         //printf("%s", buffer);
200         token = strtok(buffer, " ");
201         while(token != NULL){

```

```

202         float err = strttof(token, NULL);
203         errorSample[idx] = err;
204         idx++;
205         //printf("%.2f ", err);
206         token = strtok(NULL, " ");
207     }
208     //printf("\n");
209 }
210 //hypothesize families of distributions form
211 // sorting error
212 int i, j;
213 float temp;
214 for(i = 0; i<sampleNum-1; i++){
215     temp = errorSample[i];
216     for(j = i+1; j<sampleNum; j++){
217         if(errorSample[j] < temp){
218             errorSample[i] = errorSample[j];
219             errorSample[j] = temp;
220             temp = errorSample[i];
221         }
222     }
223 }
224 // summary statistics
225 float mean, median, variance, cv, skewness, sum = 0.0, t;
226 printf("Maximum: %.2f\n", errorSample[sampleNum-1]);
227 printf("Minimum: %.2f\n", errorSample[0]);
228 printf("Mean: ");
229 for(i = 0; i<sampleNum; i++)
230     sum += errorSample[i];
231 mean = sum/sampleNum;
232 printf("%.3f   %.3f   %.3f\n", mean, mean-errorSample[0],
233     errorSample[sampleNum-1]-mean);
234 median = (errorSample[(sampleNum-1)/2] + errorSample[sampleNum
235     /2])/2;
236 printf("Median: %.3f\n", median);
237 // mean and median are almost equal
238 sum = 0.0;
239 for(i = 0; i<sampleNum; i++)
240     sum += (errorSample[i]-mean)*(errorSample[i]-mean);
241 variance = sum / sampleNum;
242 cv = sqrt(variance)/mean;
243 printf("Variance: %.3f\n", variance);
244 printf("Co-efficient of Variance: %.3f\n", cv);
245
246 sum = 0.0;
247 for(i = 0; i<sampleNum; i++)
248     sum += pow(errorSample[i]-mean, 3);
249 t = (sampleNum-1)*(sampleNum-2);
250 skewness = (sampleNum*sum)/(t*pow(variance, 3/2));
251 printf("Skewness: %.3f\n", skewness);
252
253 // histograms
254 float d = 0.1;
255 //histogram(errorSample, sampleNum, d);
256 d = 0.2;
257 //histogram(errorSample, sampleNum, d);
258 d = 0.3;

```



```

257 //histogram(errorSample, sampleNum, d);
258 d = 0.4;
259 //histogram(errorSample, sampleNum, d);
260
261 // estimation of parameters, mu and sigma for normal
distribution
262 float mu, sigma;
263 mu = mean;
264 sigma = sqrt(variance); //(sampleNum-1)*variance/sampleNum
265 printf("\nEstimated parameters:\n");
266 printf("Mu: %.3f\n", mu);
267 printf("Sigma: %.3f\n", sigma);
268
269 // fitness test
270 d = 0.2;
271 //densityFrequencyComp(errorSample, sampleNum, mu, sigma, d);
272
273 //chi-square test
274 chiSquareTest(errorSample, sampleNum, d);
275
276 return 0;
277 }

```

Listing 1: Simulation code