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 continues 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 mu 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 $X_1, X_2, ..., X_n$. To make a histogram, we break up the range of values covered by the data into k disjoint adjacent intervals $[b_0, b_1), [b_1, b_2), ...[b_{k-1}, b_k]$. All the intervals should be the same width, say, $\Delta b = b_j – b_{j-1}$. For $j = 1, 2, ..., k$, let h_j

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

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

which we plot as a function of x.

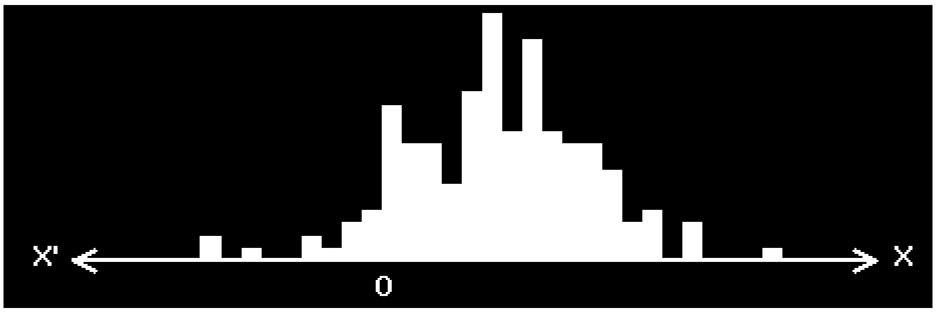

Figure 1: Histogram of errors with $\Delta b = 0.2$

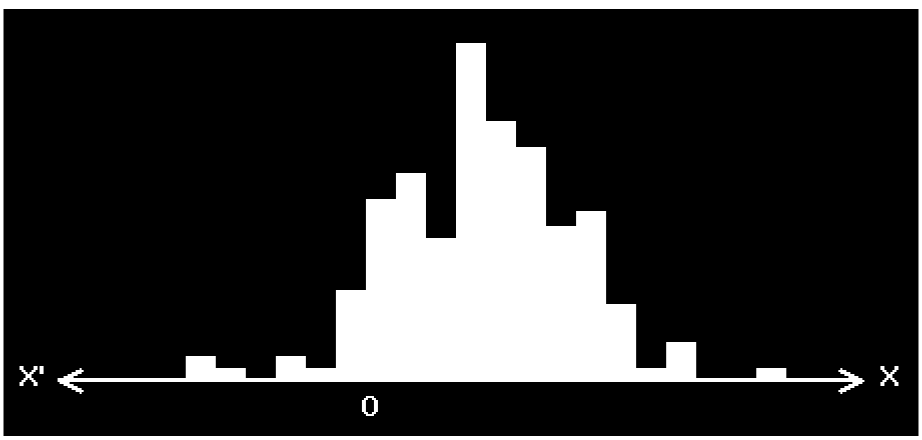

Figure 2: Histogram of errors with $\Delta b = 0.3$

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

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

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

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

The maximum likelihood estimators of the mean and the variance are

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

$$\bar{\sigma} = (\frac{1}{n}\Sigma_{i=1}^{n}(x_i-\bar{\mu})^2)^{\frac{1}{2}} \tag{6}$$

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

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

Here, $\chi^2 = 165.8701$

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

#include<graphics.h>
#include<conio.h>
```

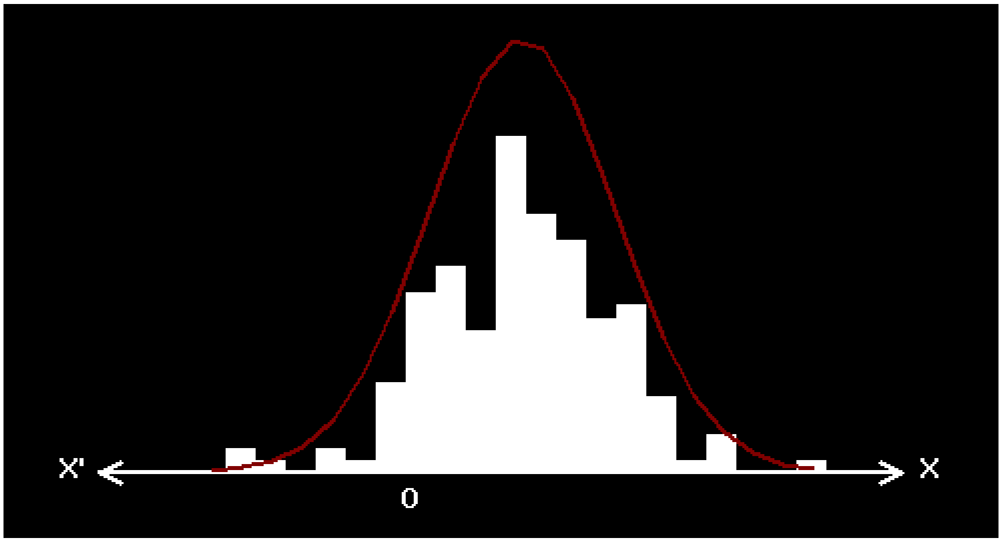

Figure 3: Histogram of density function and frequency with d = 0.3

```
#define PI 3.1416

void histogram(float *errorSample, int sampleNum, float d){
    int maxInterval = 100;
    int i, j;
    float intervalArr[maxInterval+1], densityArr[maxInterval];
    int freqArr[maxInterval], intervalNum = 0;
    // making array of end points of interval; initializing fregArr[i]
    intervalArr[0] = errorSample[0] - d;
    while(intervalArr[intervalNum]<errorSample[sampleNum-1]){
        intervalArr[intervalNum+1] = intervalArr[intervalNum]+d;
        freqArr[intervalNum] = 0;
        intervalNum++;
    }
    // calculation of frequency
    for(i = 0; i<sampleNum; i++){
        for(j = 0; j<intervalNum; j++){
            if(errorSample[i] < intervalArr[j+1]){
                freqArr[j] += 1;
                break;
            }
        }
    }
    printf("Interval number: %d\n", intervalNum);
    // plot histogram
    int gd = DETECT, gm; // VGA, gm = 0;
    //initgraph(&gd, &gm, "C:\\TC\\BGI");
    char pathtodriver[] = "";//"C:\\TC\\BGI";
    initgraph(&gd, &gm, pathtodriver);

```

```
    int errorcode = graphresult();
    if (errorcode != grOk)
    {
        printf("Graphics error: %s", grapherrormsg(errorcode));
        return;
    }

    int left, top, right, bottom;
    int x0 = 250, y0 = 400, xscale = 50, yscale = 1000, xleft = 150, xright=250;
    setlinestyle(SOLID_LINE,0,2);

    line(x0-xleft, y0, x0+xright, y0);// x-axis
    line(x0-xleft, y0, x0-xleft+10, y0-5); // for arrows
    line(x0-xleft, y0, x0-xleft+10, y0+5);
    line(x0+xright-10, y0-5, x0+xright, y0);
    line(x0+xright-10, y0+5, x0+xright, y0);
    outtextxy(x0-xleft-20, y0-10, "X'");
    outtextxy(x0+xright+10, y0-10, "X");
    outtextxy(x0, y0+5, "O");

    for(i = 0; i<intervalNum; i++){
        left = (x0 + xscale*intervalArr[i]);
        top = (y0 - yscale*freqArr[i]/sampleNum);
        right = (x0 + xscale*intervalArr[i+1]);
        bottom = y0;
        bar(left, top, right, bottom);
    }
    getch();
    closegraph();
}

void densityFrequencyComp(float *errorSample, int sampleNum, float mu, float sigma, float d){
    int maxInterval = 100;
    int i, j;
    float intervalArr[maxInterval+1], densityArr[maxInterval];
    int freqArr[maxInterval], intervalNum = 0;
    // making array of end points of interval; initializing fregArr[i]
    intervalArr[0] = errorSample[0] - d;
    while(intervalArr[intervalNum]<errorSample[sampleNum-1]){
        intervalArr[intervalNum+1] = intervalArr[intervalNum]+d;
        freqArr[intervalNum] = 0;
        intervalNum++;
    }
    // calculation of frequency
    for(i = 0; i<sampleNum; i++){
        for(j = 0; j<intervalNum; j++){
            if(errorSample[i] < intervalArr[j+1]){
                freqArr[j] += 1;
                break;
            }
        }
    }
    // calculation of probability from density function
    // f(x) = (1/(2*pi*sigma^2)^1/2)*exp(-(x-mu)^2 / 2*sigma^2)
```

```
    float temp, x;
    for(i = 0; i<intervalNum; i++){
        x = intervalArr[i] + d/2;
        temp = exp(-pow(x - mu, 2) / (2*sigma*sigma));
        densityArr[i] = temp / (sigma*sqrt(2*PI));
    }
    //printf("Interval number: %d\n", inum);
    // plot histogram
    int gd = DETECT, gm; // VGA, gm = 0; //initgraph(&gd, &gm, "C
    :\\TC\\BGI");
    char pathtodriver[] = "";//"C:\\TC\\BGI";
    initgraph(&gd, &gm, pathtodriver);

    int errorcode = graphresult();
    if (errorcode != grOk)
    {
        printf("Graphics error: %s", grapherrormsg(errorcode));
        return;
    }

    int left, top, right, bottom;
    int x0 = 250, y0 = 400, xscale = 50, yscale = 1000, xleft =
    150, xright=250;
    setlinestyle(SOLID_LINE,0,2);

    line(x0-xleft, y0, x0+xright, y0);// x-axis
    line(x0-xleft, y0, x0-xleft+10, y0-5); // for arrows
    line(x0-xleft, y0, x0-xleft+10, y0+5);
    line(x0+xright-10, y0-5, x0+xright, y0);
    line(x0+xright-10, y0+5, x0+xright, y0);
    outtextxy(x0-xleft-20, y0-10, "X'");
    outtextxy(x0+xright+10, y0-10, "X");
    outtextxy(x0, y0+5, "O");
    // plot bar
    for(i = 0; i<intervalNum; i++){
        left = (x0 + xscale*intervalArr[i]);
        top = (y0 - yscale*freqArr[i]/sampleNum);
        right = (x0 + xscale*intervalArr[i+1]);
        bottom = y0;
        bar(left, top, right, bottom);
    }
    // plot probability line
    setcolor(RED);
    setlinestyle(SOLID_LINE,0,2);
    yscale = 500;
    for(i = 0; i<intervalNum-1; i++){
        left = x0 + xscale*(intervalArr[i]+d/2);
        top = y0 - yscale*densityArr[i];
        right = x0 + xscale*(intervalArr[i+1]+d/2);
        bottom = y0 - yscale*densityArr[i+1];
        line(left, top, right, bottom);
    }

    getch();
    closegraph();
}
```

```c
void chiSquareTest(float *errorSample, int sampleNum, float d){
    int maxInterval = 100;
    int i, j;
    float intervalArr[maxInterval+1], chiArr[maxInterval];
    int freqArr[maxInterval], intervalNum = 0;
    // making array of end points of interval; initializing fregArr[i]
    intervalArr[0] = errorSample[0] - d;
    while(intervalArr[intervalNum]<errorSample[sampleNum-1]){
        intervalArr[intervalNum+1] = intervalArr[intervalNum]+d;
        freqArr[intervalNum] = 0;
        intervalNum++;
    }
    // npj value
    float pj = 1.0/intervalNum;
    float npj = sampleNum*pj;
    // calculation of frequency
    for(i = 0; i<sampleNum; i++){
        for(j = 0; j<intervalNum; j++){
            if(errorSample[i] < intervalArr[j+1]){
                freqArr[j] += 1;
                break;
            }
        }
    }
    // calculation of chi-square test value
    // (Nj - npj)^2 / npj
    float totalVal = 0;
    for(i = 0; i<intervalNum; i++){
        chiArr[i] = (freqArr[i] - npj)*(freqArr[i] - npj) / npj;
        totalVal += chiArr[i];
        printf("%d   %d    %.3f     %.3f\n", i, freqArr[i], npj, chiArr[i]);
    }
    printf("%.4f\n", totalVal);
}

int main(){
    // open file to read data on errors in the diameter of ball bearings
    FILE *file = fopen("errors-diameter.txt", "r");
    if(file == NULL){
        printf("File can not be opened");
        return 0;
    }

    int bufferLength = 100;
    int sampleNum = 154;
    char buffer[bufferLength];
    float errorSample[sampleNum];
    char *token;

    int idx = 0;
    while(fgets(buffer, bufferLength, file)){
        //printf("%s", buffer);
        token = strtok(buffer, " ");
        while(token != NULL){
```

```
            float err = strtof(token, NULL);
            errorSample[idx] = err;
            idx++;
            //printf("%.2f  ", err);
            token = strtok(NULL, " ");
        }
        //printf("\n");
    }
    //hypothesize families of distributions form
    // sorting error
    int i, j;
    float temp;
    for(i = 0; i<sampleNum-1; i++){
        temp = errorSample[i];
        for(j = i+1; j<sampleNum; j++){
            if(errorSample[j] < temp){
                errorSample[i] = errorSample[j];
                errorSample[j] = temp;
                temp = errorSample[i];
            }
        }
    }
    // summary statistics
    float mean, median, variance, cv, skewness, sum = 0.0, t;
    printf("Maximum: %.2f\n", errorSample[sampleNum-1]);
    printf("Minimum: %.2f\n", errorSample[0]);
    printf("Mean: ");
    for(i = 0; i<sampleNum; i++)
        sum += errorSample[i];
    mean = sum/sampleNum;
    printf("%.3f   %.3f   %.3f\n", mean, mean-errorSample[0], errorSample[sampleNum-1]-mean);
    median = (errorSample[(sampleNum-1)/2] + errorSample[sampleNum/2])/2;
    printf("Median: %.3f\n", median);
    // mean and median are almost equal
    sum = 0.0;
    for(i = 0; i<sampleNum; i++)
        sum += (errorSample[i]-mean)*(errorSample[i]-mean);
    variance = sum / sampleNum;
    cv = sqrt(variance)/mean;
    printf("Variance: %.3f\n", variance);
    printf("Co-efficient of Variance: %.3f\n", cv);

    sum = 0.0;
    for(i = 0; i<sampleNum; i++)
        sum += pow(errorSample[i]-mean, 3);
    t = (sampleNum-1)*(sampleNum-2);
    skewness = (sampleNum*sum)/(t*pow(variance, 3/2));
    printf("Skewness: %.3f\n", skewness);

    // histograms
    float d = 0.1;
    //histogram(errorSample, sampleNum, d);
    d = 0.2;
    //histogram(errorSample, sampleNum, d);
    d = 0.3;
```

```
    //histogram(errorSample, sampleNum, d);
    d = 0.4;
    //histogram(errorSample, sampleNum, d);

    // estimation of parameters, mu and sigma for normal distribution
    float mu, sigma;
    mu = mean;
    sigma = sqrt(variance); //(sampleNum-1)*variance/sampleNum
    printf("\nEstimated parameters:\n");
    printf("Mu: %.3f\n", mu);
    printf("Sigma: %.3f\n", sigma);

    // fitness test
    d = 0.2;
    //densityFrequencyComp(errorSample, sampleNum, mu, sigma, d);

    //chi-square test
    chiSquareTest(errorSample, sampleNum, d);

    return 0;
}
```

Listing 1: Simulation code