SYNOPSIS REPORT

On

Probability Distribution Calculator

Submitted by

Srajan Jaiswal (Enroll No. R134218170)

Amandeep Singh (Enroll. No. R134218203)

Aman Saraogi (Enroll. No. R134218204)

Raghavendra Singh (Enroll. No. R134218128)

Under the guidance of

Dr. Manoj Kumar
AP-SS
Department of Systemics

UPES
UNIVERSITY WITH A PURPOSE

SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF PETROLEUM & ENERGY STUDIES
Bidholi Campus, Energy Acres, Dehradun – 248007.

July Dec-2020

School of Computer Science

University of Petroleum & Energy Studies, Dehradun

Project Proposal Approval Form (2020)

Minor I

PROJECT TITLE: Probability Distribution Calculator

ABSTRACT

Probability Distributions always seems to be very tiring and needs lots of calculations.
Therefore, we want to develop an application using C programming for calculating these distributions within seconds for the user.
The semester-long Project is divided into phases i.e. (Idea, Requirements, knowledge on concepts of Probability Distributions), Coding Implementation, Testing and Maintenance, Deployment etc.
This Project will be supervised by our mentor Manoj Kumar sir (AP-SS Department of Systematics).

KEYWORDS:

- Mean/Mode
- Variance
- Standard Deviation
- Binomial and Poisson
- MGF

INTRODUCTION:

All projects start with an idea for a product, service, new capability, or other desired outcome. This project is based on the idea for servicing the users by getting rid from the tiring and complex calculations. This project is based on C programing language.

It is a terminal-based application in which user input his probability distributions and get the desired output (Mean, Mode, Standard Deviation, Variance, Poisson Ratio, Moment Generating Functions, Sampling Data).

PROBLEM STATEMENT:

- Taking input from user (data could be an integer, decimal and fractional value).
- User should be asked to give input in certain constraints as set by by the coding team.
- User should get the output in the form of probability distributions (with required time complexity).
- Constraints and time limit should be defined by the development team.

LITERATURE REVIEW:

Title	Link	Remarks
C programming	https://www.cprogramming.com/	Concepts of C programming is well defined.
Probability Distributions	https://en.wikipedia.org/wiki/Probability_distribution	Concepts of Probability Distributions are well explained.
File Handling in C programming	https://www.geeksforgeeks.org/basics-file-handling-c/	File handling Concepts are well structured in easy manner.

OBJECTIVES:

The program will do the following tasks:

- Taking data from a file in the form of two strings and converting them to floating-point values.
- Data could be an integer, decimal or fractional value.
- Finding the Median and Mode of N integers.
- Input real numbers and Find the Mean, Variance, and Standard Deviation.
- Moment Generating Function for Discrete data.
- Binomial & Poisson distribution.
- Significance tests for sampling distributions.
- Program should pass all the valid test cases provided by the user with required time limit.

METHODOLOGY:

For this project, the language used is C.
The entire implementation of this project can be summarized into the following steps:

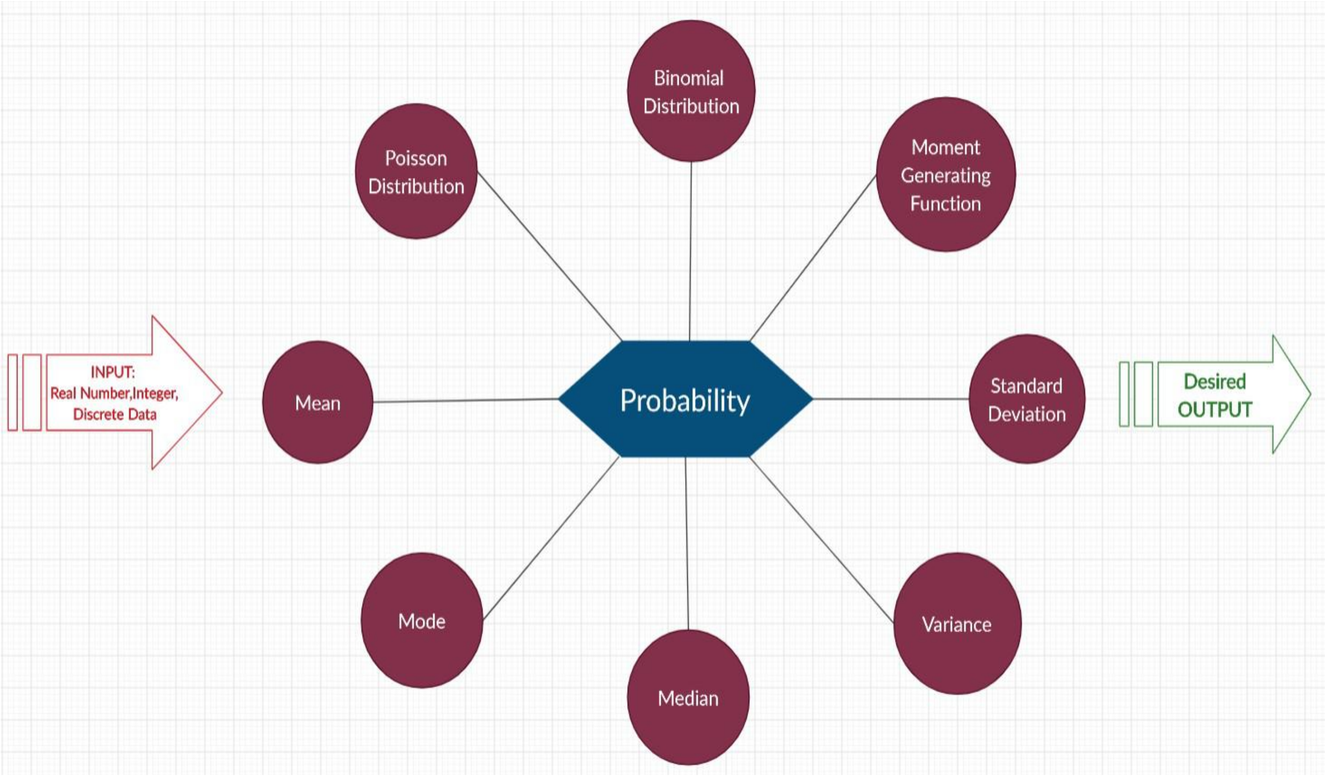
1. **Import the libraries and header file:** Libraries will be imported in coding implementation for the required use of logic.

2. **Taking the data from user**: Taking data from a file in the form of two strings and converting them to floating-point values. Data could be an integer, decimal or fractional value. User will input test case data (with given constraints) in the required format to get the required output.

3. **Uses of switch Cases**: Majorly program will be structured in different cases with switch method in C. Different Cases will be handled by different team members for calculating different Probability Distributions.

4. **Methodologies Required:** Good knowledge of methodologies for Mean, Mode, Standard Deviation, Binomial & Poisson distribution, Moment Generating Functions, Significance tests for sampling distributions will be required for the implementation of application.

PICTORIAL REPRESENTATION:



The above picture represents a basic idea behind the project and summarizes its major features describing an overall understanding of the projects, the inputs to take and the functions it will be providing after its completion.

It includes all key components except the Sampling part that is done on the basis of the result we get after the calculation of the required values.

SYSTEM REQUIREMENTS:

Hardware:

- RAM: 4-8GB
- Disk Space: 4GB

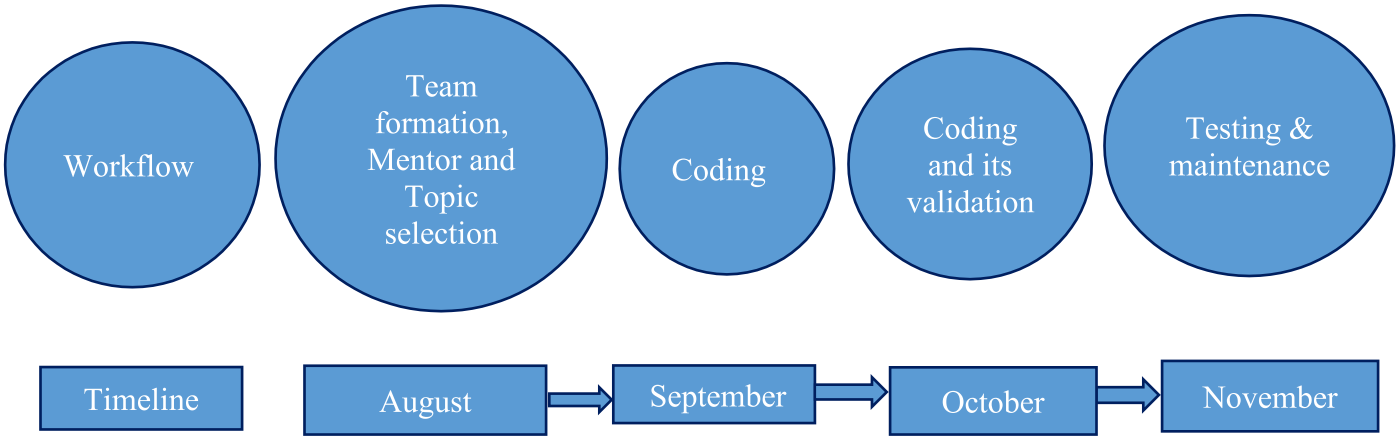
Software:

- **Compiler:** MinGW
- **Editor**: Notepad, Sublime Text, VScode, CodeBlocks.

Operating System:

- Windows.
- Linux.
- Mac OS.

SCHEDULE:



Pert Chart

REFERENCES:

[1]https://www.geeksforgeeks.org/switch-statement-cc/
[2]https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm
[3]https://www.javatpoint.com/c-switch
[4]https://www.w3schools.com/cpp/cpp_switch.asp
[5]https://www.programiz.com/c-programming
[6]https://beginnersbook.com/2014/01/c-tutorial-for-beginners-with-examples/
[7]https://www.learn-c.org/
[8] https://app.creately.com/diagram/JNCKLrO98AV/edit

Code:

```
/*
This program does the following tasks combined :-
1) Taking data from a file in the form of two strings and converting them to floating point values.
   Data could be an integer,decimal or fractional value.
2) Finding median and mode of N integers.
3) Input real numbers and find the mean, variance and standard deviation.
4) Moment generating function for discrete data.
5) Binomial Distribution
*/


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

/*(Amandeep)*/
double RealMean(double sum, int n);
double RealVariance(double sumForVariance, int n);
double RealStdDeviation(double variance);

/*(Amandeep)*/
int factorial(int num);
int combination(int n, int r);
double Biprobability(double event, int power);

/*(Amandeep)*/
double PsnMean(int n, double p);
double PsnVariance(double mean);
double PsnSkewness(double mean);
double PsnKurtosis(double mean);
double PsnMoment3(double mean);
double PsnMoment4(double mean);
int factorial(int num);
```

```
double PsnProbability(double mean, int x);

/*MGF(Srajan Jaiswal)*/
double dataMean(double SumXF, double SumF);
double dataVariance(double Moment2);
double dataSkewness(double Moment3, double Moment2);
double dataKurtosis(double Moment4, double Moment2);
void checkSkewness(double skewness);
void checkKurtosis(double kurtosis);

/*Data Extraction*/
enum numberType
{
    integer,
    decimal,
    fraction
} type;

double checkAndReturn(char *string);
static int funcCallCounter = 0;

/* MAIN STARTS HERE */
int main()
{
    system("cls");
    system("COLOR FC");
    int mainchoice;
    printf("\t\tWelcome to Probability Distributions \n\n");
    printf("1. Find median and mode of N integers \n");
    printf("2. Find mean, variance and standard deviation of N real numbers \n");
    printf("3. Binomial Distribution \n");
    printf("4. Poisson Distribution \n");
    printf("5. Moment Generating Function for random variables\n");
    printf("6. Significance tests for sampling distribution\n\n");
    printf("Choice: ");
    fflush(stdin);
    scanf("%d", &mainchoice);
    switch (mainchoice)
    {
    case 1: /* Aman Saraogi (Median and Mode)*/
    {
        system("cls");
        system("COLOR FC");
        int i, j;
        int n, max = 0, k = 0, c = 1;
        double *N, *ascN, temp, median, mode, Mode[20];
        printf("Enter the number of trails N: "); /* N Data Points */
        fflush(stdin);
        scanf("%d", &n);
        N = (double *)malloc(n * sizeof(double)); /* Space for N data points */
        printf("Enter the data points:\n");
        fflush(stdin);
        for (i = 0; i < n; i++)
        {
            scanf("%lf", &N[i]);
        }
        ascN = (double *)malloc(n * sizeof(double));
        for (i = 0; i < n; i++) /*Assigning the Ascending order array of data poins same as the array of
entered data points first */
        {
            ascN[i] = N[i];
        }
        for (i = 0; i < n; i++) /* Arranging data points in Ascending order for median */
        {
            for (j = i + 1; j < n; j++)
            {
                if (ascN[i] > ascN[j])
                {
                    temp = ascN[i];
                    ascN[i] = ascN[j];
                    ascN[j] = temp;
                }
            }
        }
        /* Median */
        if (n % 2 == 0) /*Using formula for median if even number of terms are there*/
            median = (double)(ascN[n / 2] + ascN[(n - 1) / 2]) / 2;
        else /* formula for odd number of terms */
            median = (double)ascN[(n - 1) / 2];
        printf("Median = %.0lf", median);
        /* Mode */
        for (i = 0; i < n - 1; i++)
        {
            mode = 0;
            for (j = i + 1; j < n; j++)
            {
                if (ascN[i] == ascN[j])
                {
                    mode++;
                }
            }
            if ((mode > max) && (mode != 0))
            {
                k = 0;
                max = mode;
                Mode[k] = ascN[i];
                k++;
            }
            else if (mode == max)
            {
                Mode[k] = ascN[i];
                k++;
            }
        }
        for (i = 0; i < n; i++)
        {
            if (ascN[i] == Mode[i])
                c++;
        }
        if (c == n)
            printf("\nThere is no mode");
        else
        {
            printf("\nMode = ");
            for (i = 0; i < k; i++)
                printf("%.0lf", Mode[i]);
        }
    }
    break;


    case 2: /*Mean,Standard Deviation, Variance,Poission,Binomial(Amandeep Singh) */
    {
        system("cls");
        system("COLOR FC");
        int i, n;
        double *N, mean, variance, stdDeviation, sumForMean = 0, sumForVariance = 0;
        printf("Enter the number of trails N: "); //N data points
        scanf("%d", &n);
      N = (double *)malloc(n * sizeof(double));
        printf("Enter the data points:\n");
        for (i = 0; i < n; i++)
        {
            scanf("%lf", &N[i]);
            sumForMean += N[i]; //Sum of entered elements for calculating mean later
        }
        mean = RealMean(sumForMean, n); // we find mean here

        for (i = 0; i < n; i++) /* Separate loop for computing variance  and standard deviation  */
        {
            sumForVariance += pow((N[i] - mean), 2); // calculating variance
        }
        variance = RealVariance(sumForVariance, n);                    //variance
        stdDeviation = RealStdDeviation(variance);                // standard deviation
        printf("\nAverage of data points = %.2lf\n", mean);    // printing mean
        printf("Variance = %.2lf\n", variance);                       // printing variance
        printf("Standard Deviation = %.2lf\n", stdDeviation); // printing standard deviation
        free(N);
    }
    break;
```

```
    case 3: // Binomial Distribution(Amandeep Singh)
     {
        system("cls");
        system("COLOR FC");
        int n, ch, x;
        char ch1 = 'n';
        double *P, p, q, mean, mode, variance, skewness, kurtosis, Finfo;
        printf("Enter the number of trials N: ");
        scanf("%d", &n);
        P = (double *)malloc(n * sizeof(double)); /* Space for probabilities of random variables */
        printf("Probability of success p: ");
        scanf("%lf", &p);
        q = 1 - p;
        printf("Probability of failure q: %.2lf \n\n", q);
        printf("Menu: \n");
        printf("1) Probablility Mass Function for a variable \n2) Other Statistical Parameters \n   a)
Mean \n   b) Mode \n   c) Variance \n   d) Skewness \n   e) Kurtosis \n\n");
        do
        {
            if (ch1 == 'y' || ch1 == 'Y')
            {
                system("cls");
                system("COLOR FC");
                printf("Number of trials N: %d \n", n);
                printf("Probability of success p: %.2lf \n", p);
                printf("Probability of failure q: %.2lf \n\n", q);
                printf("Menu: \n");
                printf("1) Probablility Mass Function for a variable \n2) Other Statistical
Parameters \n   a) Mean \n   b) Mode \n   c) Variance \n   d) Skewness \n   e) Kurtosis \n\n");
            }
            printf("Choice: ");
            fflush(stdin);
            scanf("%d", &ch);
            switch (ch)
            {
            case 1:
                printf("Enter variable x: ");
                scanf("%d", &x);
                P[x] = combination(n, x) * Biprobability(p, x) * Biprobability(q, n - x); //
Binomial Distribution
                printf("P[%d] = %.2lf\n", x, P[x]);
                break;
            case 2:
                mean = n * p;   // mean
                mode = (n + 1) * p; //mode
                variance = n * p * q; //variance
                skewness = (1 - (2 * p)) / sqrt(n * p * q);  //skewness
                kurtosis = (1 - (6 * p * q)) / (n * p * q); // kurtosis
                Finfo = n / (p * q);
                printf("\na) Mean = %.2lf", mean);
                printf("\nb) Mode = %.2lf", mode);
                printf("\nc) Variance = %.2lf", variance);
                printf("\nd) Skewness = %.2lf", skewness);
                printf("\ne) Kurtosis = %.2lf", kurtosis);

                break;
            default:
                printf("\nWrong choice!");
                break;
            }
            printf("\nWant to enter again? (y/n) :");
            fflush(stdin);
            scanf("%c", &ch1);
        } while (ch1 == 'y' || ch1 == 'Y');
        free(P);
    }
    break;

    case 4: // Poisson Distribution(Amandeep Singh)
    {
        system("cls");
        system("COLOR FC");
        int n, x;
        float p;
        double *P, mean, variance, moment3, moment4, skewness, kurtosis;
        printf("Enter the number of trials N: ");
        scanf("%d", &n);
        P = (double *)malloc(n * sizeof(double)); /* Space for probabilities of random variables */
        printf("Enter the probability of success:");
        scanf("%f", &p);

        mean = PsnMean(n, p); ///mean
        variance = PsnVariance(mean); //variance
        skewness = PsnSkewness(mean); //skewness
        kurtosis = PsnKurtosis(mean); //kurtosis
        printf("Enter the random variable:");
        scanf("%d", &x);
        P[x] = PsnProbability(mean, x); // poisson distribution
        printf("\nP[%d] = %f", x, P[x]);
        printf("\nMean = %.4lf \nVarience = %.4lf  \nSkewness = %.4lf \nKurtosis = %.4lf \n", mean,
variance, skewness, kurtosis);
        free(P);
    }
    /*End of Amandeep*/
    break;
    case 5: /* Moment Generating Function(Srajan Jaiswal)*/
    {
        system("cls");
        system("COLOR FC");
        int datachoice;
        printf("\tMoment Generating Function for random variables\n\n");
        printf("How do you want to provide data?\n");
        printf("1. Enter data manually \n");
        printf("2. Fetch data from data.txt file \n\n");
        printf("Choice: ");
        fflush(stdin);
        scanf("%d", &datachoice);
        switch (datachoice)
        {
        case 1:
        {
            system("cls");
            system("COLOR FC");
            int elements, i, j, r, choice; /* elements are the columns whereas rows = 2 ( for X and F )
*/
            double *data[2], mean, variance, skewness, kurtosis, SumF = 0, SumXF = 0, centralM[4]
= {0}, rawM[4] = {0}, point;
            printf("Enter number of elements:");
            scanf("%d", &elements);
            printf("\nEnter the random variables and their probabilites: \n");
            for (i = 0; i < 2; i++) /* Getting variables and their probabilities here */
            {
                data[i] = (double *)malloc(elements * sizeof(double));
                for (j = 0; j < elements; j++)
                {
                    if (i == 0)
                    {
                        printf("X[%d] = ", j + 1);
                        scanf("%lf", &data[i][j]);
                    }
                    if (i == 1)
                    {
                        printf("F[%d] = ", j + 1);
                        scanf("%lf", &data[i][j]);
                        SumF += data[i][j];
                    }
                }
                printf("\n");
            }
            /* Loop for calculating sum used in mean formula */
            for (j = 0; j < elements; j++)
            {
                SumXF += data[0][j] * data[1][j]; /* equivalent to X[i]*F[i] */
            }
            mean = dataMean(SumXF, SumF); // mean calculated by the mgf
          printf("Menu : \n\n1) First 4 Central Moments with other parameters \n2) First 4 Raw
Moments about a point \n\nChoice: ");
          scanf("%d", &choice);
          switch (choice)
            {
            case 1: /* Finding central moment */
                for (r = 0; r < 4; r++)
                {
                    for (j = 0; j < elements; j++)
                    {
                    centralM[r] += (pow((data[0][j] - mean), r + 1) * data[1][j]) / SumF;
                    }
                }
                break;
            case 2:                /* Finding raw moment */
                printf("About point:");
                scanf("%lf", &point);
                for (r = 0; r < 4; r++)
                {
                    for (j = 0; j < elements; j++)
                    {
                        rawM[r] += (pow((data[0][j] - point), r + 1) * data[1][j]) / SumF;
                    }
                }
                break;
            default:
                printf("Wrong Choice!");
                break;
            }
            if (choice == 1)
            {
                printf("\nThe central moments are: \n");
                for (r = 0; r < 4; r++)
                {
                    printf("C[%d] = %.4lf \n", r + 1, centralM[r]);
                }

                printf("\nMean = %lf \n", mean);

                variance = dataVariance(centralM[1]);
                printf("Variance = %.4lf \n", variance);

                skewness = dataSkewness(centralM[2], centralM[1]);
                printf("Skewness = %.4lf ", skewness);
                checkSkewness(skewness); // this function is  used for checking the curve

                kurtosis = dataKurtosis(centralM[3], centralM[1]);
                printf("Kurtosis = %.4lf ", kurtosis);
                checkKurtosis(kurtosis);
            }
```

```
            else if (choice == 2)
            {
                printf("\nThe raw moments are: \n");
                for (r = 0; r < 4; r++)
                {
                    printf("R[%d] = %.4lf \n", r + 1, rawM[r]);
                }
                printf("\nMean = %lf \n", mean);
            }
        }
        break;
        case 2: /* Data Extraction(Srajan Jaiswal) */
        {
          system("cls");
          system("COLOR FC");
          FILE *file = fopen("data.txt", "r");
          int Xindex = -1, Findex = -1, i, j, length;
          double firstNo[20], secondNo[20];
          static char stringOne[100] = {0}, stringTwo[100] = {0}, c;
          int r, choice;
         double *data[2], mean, variance, skewness, kurtosis, SumF = 0, SumXF = 0, centralM[4] =
{0}, rawM[4] = {0}, point;
            while (!feof(file)) /*A loop for moving the file pointer and omitting the first line of the
data file since it has stray text */
            {
                c = fgetc(file);
                if (c == '\n')
                    break;
            }
            while (!feof(file))
            {

                if (funcCallCounter % 2 == 0) /* If checkAndReturn function is called even
number of times then 1 line is read and index is increased */
                {
                    Xindex++;
                    Findex++;
                }
                fscanf(file, "%s\t%s", stringOne, stringTwo);
                firstNo[Xindex] = checkAndReturn(stringOne);
                secondNo[Findex] = checkAndReturn(stringTwo);
            }
            fclose(file);
            length = Xindex; /* Xindex = Findex after loop */
            for (i = 0; i < 2; i++)
            {
                data[i] = (double *)malloc((length + 1) * sizeof(double)); /* length + 1
allocation since length is an index */
            }
            for (j = 0; j < length; j++)
            {
                data[0][j] = firstNo[j];
                data[1][j] = secondNo[j];
                SumF += data[1][j];
            }
            /* Loop for calculating sum used in mean formula */
            for (j = 0; j < length; j++)
            {
                SumXF += data[0][j] * data[1][j]; /* equivalent to X[i]*F[i] */
            }
            mean = dataMean(SumXF, SumF);
            printf("Menu : \n\n1) First 4 Central Moments with other parameters \n2) First 4 Raw
Moments about a point \n\nChoice: ");
            scanf("%d", &choice);
            switch (choice)
            {
            case 1: /* Finding central moment */
                for (r = 0; r < 4; r++)
                {
                    for (j = 0; j < length; j++)
                    {
                        centralM[r] += (pow((data[0][j] - mean), r + 1) * data[1][j]) /
SumF;
                    }
                }
                break;
            case 2: /* Finding raw moment */
                printf("About point:");
                scanf("%lf", &point);
                for (r = 0; r < 4; r++)
                {
                    for (j = 0; j < length; j++)
                    {
                        rawM[r] += (pow((data[0][j] - point), r + 1) * data[1][j]) / SumF;
                    }
                }
                break;
            default:
                printf("Wrong Choice!");
                break;
            }
            if (choice == 1)
            {
                printf("\nThe central moments are: \n");
                for (r = 0; r < 4; r++)
                {
                    printf("C[%d] = %.4lf \n", r + 1, centralM[r]);
                }

                printf("\nMean = %lf \n", mean);

                variance = dataVariance(centralM[1]);
                printf("Variance = %.4lf \n", variance);

                skewness = dataSkewness(centralM[2], centralM[1]);
                printf("Skewness = %.4lf ", skewness);
                checkSkewness(skewness);

                kurtosis = dataKurtosis(centralM[3], centralM[1]);
                printf("Kurtosis = %.4lf ", kurtosis);
                checkKurtosis(kurtosis);
            }
            else if (choice == 2)
            {

                printf("\nThe raw moments are: \n");
                for (r = 0; r < 4; r++)
                {
                    printf("R[%d] = %.4lf \n", r + 1, rawM[r]);
                }
                printf("\nMean = %lf \n", mean);
            }
        }
        break;
        }
        /* End of Data Extraction(Srajan Jaiswal)*/
    }
    break;


    case 6: /* Sampling (Raghavendra Singh) */
    {
        int n, ch, x, N[100], sum = 0, B[100], sum1 = 0, v, i;
        double n2;
        double A[10][5] = {{1.00, 6.34, 12.71, 31.82, 63.66},
                           {0.816, 2.92, 4.3, 6.96, 9.92},
                           {0.765, 2.35, 3.18, 4.54, 5.84},
                           {0.741, 2.13, 2.78, 3.75, 4.60},
                           {0.727, 2.02, 2.57, 3.36, 4.03},
                           {0.718, 1.94, 2.45, 3.14, 3.71},
                           {0.711, 1.90, 2.36, 3.00, 3.50},
                           {0.706, 1.86, 2.31, 2.90, 3.36},
                           {0.703, 1.83, 2.26, 2.82, 3.25},
                           {0.700, 1.81, 2.23, 2.76, 3.17}};
        char ch1 = 'n';
        double *P, p, q, mean, mode, variance, skewness, kurtosis, Finfo, z, e, e1, d, stdDeviation,
mean1, t, tv, tv1;
        system("cls");
        system("COLOR FC");
        printf("Enter the number of trials N: ");
        scanf("%d", &n);
        P = (double *)malloc(n * sizeof(double)); /* Space for probabilities of random variables */
        printf("Probability of success p: ");
        scanf("%lf", &p);
        q = 1 - p;
        printf("Probability of failure q: %.5lf \n\n", q);
        printf("Menu: \n");
        printf("1) Test of Significance for large Samples\n2) Test of Significance for Means of two
large Samples\n3) Test of Significance of a Sample Mean\n");
        do
        {
            if (ch1 == 'y' || ch1 == 'Y')
            {
                system("cls");
                system("COLOR FC");
                printf("Number of trials N: %d \n", n);
                printf("Probability of success p: %.2lf \n", p);
                printf("Probability of failure q: %.2lf \n\n", q);
                printf("Menu: \n");
                printf("1) Test of Significance for large Samples(z)\n");  // FISHER z TEST
                printf("2) Test of Significance for Means of two large Samples\n"); // CHI
SQUARE METHOD
                printf("3) Test of Significance of a Sample Mean\n"); // student-t test
            }
            printf("\nChoice: ");
            fflush(stdin);
            scanf("%d", &ch);
            switch (ch)
            {
            case 1:
                printf("Enter variable x: ");
                scanf("%d", &x);
                z = (x - (n * p)) / sqrt(n * p * q); //standard normal variate
                printf("\nStandard normal variate = %.5lf", z);
                if (z < 1.96)
                    printf("\n\nThe difference between the observes and expected number of
successes is\n not significant.");
                if (z > 1.96 && z < 2.58)
                    printf("\n\nThe difference is significant at 5%% level of significance.");
                if (z > 2.58)
                    printf("\n\nThe difference is significant at 1%% level of significance.");
                break;
            case 2:
                printf("Enter the number of trials for second sample N2: ");
                scanf("%lf", &n2);
                e = (sqrt((n + n2) * p * q)) * (sqrt((1 / n) + (1 / n2))); // expected value
                z = (n * p) - (n2 * p) / e; // FORMULA FOR CALCULATING Z
                printf("\nStandard normal variate = %.5lf\n", z);
                if (z > 1.96 && z < 3)
                    printf("\n\nThe difference is significant at 5%% level of significance.");
                if (z > 3)
                    printf("\n\nIt is Highly probable that either the samples have not been
drawn from the\n same population or the sampling is not simple.");
```

```
                d = (n * p) - (n2 * p);
                e1 = sqrt(((n * p * q) / n) + ((n2 * p * q) / n2)); // CALCULATING new
expected value
                if (d > (10 * e1))
                    printf("\n\nIt is Highly significant.");
                break;
            case 3:
                printf("Enter the sample elements: \n");
                for (i = 0; i < n; i++)
                {
                    scanf("%d", &N[i]);
                    sum = sum + N[i];
                }
                mean = sum / n;
                printf("\nMean: %.5lf\n", mean);
                for (i = 0; i < n; i++)
                {
                    B[i] = N[i] - mean;
                    sum1 += i * i;
                }
                stdDeviation = sum1 / (n - 1);
                printf("\nEnter the assumed mean of the normal population: ");
                scanf("%.5lf", &mean1);
                t = ((mean - mean1) * sqrt(n)) / (sqrt(stdDeviation));
                printf("\nThe statistic t is: %.5lf", t);
                v = n - 1;
                tv = A[v][2];
                tv1 = A[v][4];
                if (t > tv)
                    printf("\n\nThe difference between mean and mean1 is said to be
significant at 5% level of significance.");
                if (t > tv1)
                    printf("\n\nThe difference is significant at 1% level of significance.");
                if (t < tv)
                    printf("\n\nThe data is said to be consistent with the hypothesis that
mean1\n is the mean of the population.");
                break;
            default:
                printf("\nWrong choice!");
                ;
                break;
            }
            printf("\nWant to enter again? (y/n) :");
            fflush(stdin);
            scanf("%c", &ch1);
        } while (ch1 == 'y' || ch1 == 'Y');
        free(P);
    }
    break;
        //(End of Raghavendra Singh)
    }
}
```

```
/* END OF MAIN */

/* FUNCTIONS USED IN THE ABOVE METRICS CALCULATED */

/* Amandeep  funcs */

double RealMean(double sumForMean, int n)
{
        double mean;
        mean = sumForMean / n;
        return mean;
}

double RealVariance(double sumForVariance, int n)
{
        double variance;
        variance = sumForVariance / n;
        return variance;
}

double RealStdDeviation(double variance)
{
        double stdDeviation;
        stdDeviation = sqrt(variance);
        return stdDeviation;
}
/* End of Amandeep funcs */

/*  funcs */
int factorial(int num)
{
        int i = 1, fact = 1;
        while (i <= num)
        {
                fact = fact * i;
                i++;
        }
        return fact;
}
int combination(int n, int r)
{
        int comb;
        comb = (factorial(n)) / (factorial(r) * factorial(n - r));
        return comb;
}
double Biprobability(double event, int power)
{
        double prob;
        prob = pow(event, power);
        return prob;
}                               /* End of  funcs */
```

```
/*case3 funcs */
double PsnProbability(double mean, int x)
{
        double calP;
        calP = (exp(-mean) * pow(mean, x)) / factorial(x);
        return calP;
}

double PsnMean(int n, double p)
{
        double mean;
        mean = n * p;
        return mean;
}


double PsnVariance(double mean)
{
        return mean;
}

double PsnSkewness(double mean)
{
        double skewness;
        skewness = (1 / mean);
        return skewness;
}

double PsnKurtosis(double mean)
{
        double kurtosis;
        kurtosis = 3 + (1 / mean);
        return kurtosis;
}
/* End of case3 funcs    Amandeep*/






/* Srajan MGF Functions */
double dataMean(double SumXF, double SumF)
{
        double mean;
        mean = SumXF / SumF;
        return mean;
}

double dataVariance(double Moment2)
{
        double variance;
        variance = Moment2;
        return variance;
}

double dataSkewness(double Moment3, double Moment2)
{
        double skewness;
        skewness = pow(Moment3, 2) / pow(Moment2, 3);
        return skewness;
}

double dataKurtosis(double Moment4, double Moment2)
{
        double kurtosis;
        kurtosis = Moment4 / pow(Moment2, 2);
        return kurtosis;
}

void checkSkewness(double skewness)
{
        if (skewness > 0)
        {
                printf("The curve is positively skewed\n");
        }
        else if (skewness < 0)
        {
                printf("The curve is negatively skewed\n");
        }
        else if (skewness == 0)
        {
                printf("The curve is symmetric\n");
        }
}

void checkKurtosis(double kurtosis)
{
        if (kurtosis > 3)
        {
                printf("The curve is leptokurtic\n");
        }
        else if (kurtosis < 3)
        {
                printf("The curve is platykurtic\n");
        }
        else if (kurtosis == 3)
        {
                printf("The curve is normal\n");
        }
}
/* End of MGF Functions */
/* Data Extraction Functions */
double checkAndReturn(char *string) /* A function which checks the type of value in file and converts it into
decimals */
{
    int i = 0, num, denom;
    double number;
    char stringNum[20], stringDenom[20];
    while (string[i] != '\0')
    {
        if (string[i] == '.')
        {
            type = decimal;
            break;
        }
        if (string[i] == '/')
        {
            type = fraction;
            break;
        }
        type = integer;
        i++;
    }
    if (type == integer)
    {
        number = atoi(string);
    }
    else if (type == decimal)
    {
        number = atof(string);
    }
    else if (type == fraction)
    {
        int k = 0;
        while (string[k] != '\0')
        {
            if (string[k] == '/')
            {
                int start1 = 0, currPos = k, j = k + 1, start2 = 0;
                while (string[start1] != string[currPos])
                {
                    stringNum[start1] = string[start1];
                    stringNum[start1 + 1] = '\0';
                    start1++;
                }
                while (string[j] != '\0')
                {
                    stringDenom[start2] = string[j];
                    stringDenom[start2 + 1] = '\0';
                    start2++;
                    j++;

    }
```

```
                }
            k++;
        }
        num = atoi(stringNum);
        denom = atoi(stringDenom);
        number = (double)num / denom;
    }
    funcCallCounter++;
    return number;
}
/* End of Data Extraction Functions */
```

Synopsis Draft verified by

Dr. Manoj Kumar
Project Guide
(Name & Sign)

HOD
(Dept. of Systemics)