Received: 5 July 2023 | Revised: 9 August 2023 | Accepted: 11 August 2023 | Published online: 14 August 2023

RESEARCH ARTICLE

Archives of Advanced Engineering Science
2024, Vol. 2(1) 37–52
DOI: 10.47852/bonviewAAES32021322

BON VIEW PUBLISHING

Statistical Study of Bisection Method for Cubic Equations with Random Coefficients

Shikha Kumari[1] and Soubhik Chakraborty[1,*]

[1]*Department of Mathematics, Birla Institute of Technology, Mesra, India*

Abstract: The bisection method is an iterative approach used in numerical analysis to find solutions to nonlinear equations. The main purpose of this paper is to study how the parameters of a probability distribution characterizing the coefficients of a cubic polynomial can influence the convergence of the bisection method. The study covers discrete and continuous distributions, including discrete uniform, continuous uniform, and normal distributions. It was found that for both types of uniform distribution inputs, a second-degree polynomial equation can predict the average iteration for a given parameter r, where r indicates the distribution interval $[-r, r]$. Interestingly, the coefficients of the second-degree polynomial are nearly identical for discrete and continuous uniform distributions. For normal distribution input, the average iteration does not depend upon the standard deviation when the mean is fixed and the standard deviation is varying. But when the standard deviation is fixed and the mean is varying, the second-degree polynomial is still the best fit. This means the average iteration depends upon the mean of the normal distribution. Overall, our paper concludes that: I. For uniform distribution input, the average iteration does not depend on whether the distribution is discrete or continuous but rather depends on the range of the distribution which is its parameter. II. For non-uniform distribution input, the average iteration depends on the mean of the distribution (location parameter) but not on the standard deviation (scale parameter). Finally, a curtain is raised in the future direction of research in which we propose to combine the bisection method with the regula falsi and Newton-Raphson methods to increase the rate of convergence.

Keywords: bisection method, uniform distribution, normal distribution, cubic equation, statistical analysis

1. Introduction

1.1. Introduction to bisection method

It is sometimes difficult to locate the solution of equations $f(x) = 0$ in scientific and technical inquiries. Either $f(x)$ is a quadratic or cubic or biquadratic equation; there is algebraic approach for locating the roots in terms of the coefficients. However, if $f(x)$ is a higher-degree polynomial or an equation with transcendental functions, algebraic approaches are not available. For instance, when M_0, g, u, u_0, and u_f are supplied, the equation

$$\frac{M_0}{M_0 - u_f\, t} = e^{(u+gt)/u_0} \tag{1}$$

is a nonlinear equation for t. This kind of equation is used in rocket research. It is difficult to locate the roots of such nonlinear equations which is frequently encountered in engineering. As a result, several numerical methods have been developed with the goal of offering effective ways to discover numerical solutions to such issues, among which the bisection method was one of the first numerical methods to be developed. The next section will include the explanation of the bisection method.

*Corresponding author: Soubhik Chakraborty, Department of Mathematics, Birla Institute of Technology, Mesra, India. Email: soubhikc@yahoo.co.in

1.2. Bisection method

To "bisect" something implies to slice it down the middle. The bisection method reduces the search area by half at each stage while looking for a solution. The bisection method is thought to possess a linear rate of convergence and provides acceptable accuracy.

1.2.1. Steps to follow in bisection method

Step 1: Chose a and b such that $f(a) > 0$ and $f(b) < 0$.
Step 2: Compute a midpoint, $c = (a + b)/2$ as the average. It is known as interval halving.
Step 3: Evaluate the function f for a specific value of c.
Step 4: Only when $f(c) = 0$ the root of the function can be located.
Step 5: If $f(c) \neq 0$, the sign must be determined:

i. If the sign of $f(c)$ is similar to the sign of $f(a)$, a is swapped with c while b is kept at its current value.
ii. If the sign of $f(c)$ is similar to the sign of $f(b)$, b is swapped with c while a is kept at its current value.

Return to step 2 and recalculate c in order to obtain the appropriate value using the new values of a or b.

1.3. Literature review

The fundamental approach to finding a root is the bisection method. Every loop implies a halving of the interval. Since *f(a)* and *f(b)* should have different signs and "*f*" is a continuous function in the interval [*a, b*], then method will surely converge to a root of "*f*." The role of bisection method by Solanki et al. (2014) helped us to learn more about the area of the bisection method. In addition to focusing on the bisection method's importance in computer science research, this work also offered a novel method that combines bisection with other methods, such as the Newton-Raphson method which keeps the root bracketed while enabling us to take advantage of the Newton-Raphson method's speed. To further address the issue of nonlinear unconstrained minimization, Morozova (2008) suggested an adaptation of a new multidimensional bisection method for minimizing function over the simplex. This method does not need the function to be differentiable and is guaranteed to converge to the minimizer for the class of strictly unimodal functions.

Another interesting research study on the bisection method for triangles was conducted by Adler (1983). According to this study, the longest edge of a triangle was picked and bisected to give birth to two daughter triangles and continued the bisection procedure indefinitely. He established precise estimates for the longest j^{th} generation edge and demonstrated that the infinite family of triangles so formed falls into a finite number of similarity classes. A geometrical approach to the bisection method by Gutierrez et al. (2004) helped us to know that how the behavior of the bisection method depends on the classification of triangles which is to be bisected. Since, the bisection method is the consecutive bisection of a triangle by the median of the longest side, therefore partition of the triangles into classes reflects this behavior by taking into account some fundamental geometrical properties. Its main finding is an asymptotic upper constraint on the total number of triangle similarity classes that may be established on an iterative bisection-created mesh, that was previously unknown. It shows that there are finite number of directions on the plane that may be given by the sides of the resultant triangles. In addition to these research articles, we made use of other books and works that are cited in the reference section (Chapra & Canale, 1985; Frost, 2023a; Frost, 2023b; Sastry, 2003; Kreyszig, 2006) (http://amsi.org.au/ESA_Senior_Years/SeniorTopic3/3j/3j_2content_1.html).

For further literature on bisection method, the reader is referred to Sikorski (1982), Sikorski (1985), Graf et al. (1989), Novak (1989), Oliveira and Takahashi (2020), Mourrain et al. (2002), Vrahatis (2020), Kearfott (1979), Corliss (1977), and Dichotomy Method—Encyclopedia of Mathematics (2015) (https://corporatefinanceinstitute.com/resources/data-science/uniform-distribution/).

1.4. Motivation and problem statement

Sometimes we can solve problems in a good, easy, or accurate way. For instance, equations like quadratic and linear equations may be solved precisely. However, certain equations might be considerably trickier to solve than others precisely. In rare cases, it may even be difficult to pen down an accurate expression for a solution.

Exercises in mathematics textbooks for schools are frequently purposefully made to provide exact solutions. However, there is no reason to anticipate a particularly good outcome when solving many mathematical equations derived from real-life situations. Most of the time, what we can expect is an approximate solution with the required level of precision. We can use approximate numerical methods to obtain a solution when equations are challenging to solve. Sometimes getting an approximate solution is more effective.

This paper primarily focuses on the bisection method, one of the widely used numerical methods for locating roots. It determines how the parameter of a probability distribution which characterizes the coefficient of a cubic polynomial influence the convergence of the bisection method. Therefore, this study would help us in taking a decision whether to recommend or not to recommend bisection method for a solution if we have prior knowledge that the coefficient of the equations to be solved coming from a particular probability distribution. Further if we have knowledge of parameters of the underlined distributions, we may be able to predict the average iterations as the function of the distribution parameter. Research in this direction might be important in tackling many issues emerging in diverse fields of higher mathematics. Such a statistical analysis of the bisection method speeds up the process of solving a problem.

1.5. Paper alignment

The alignment of this paper is as follows:

Section 1: This section presents the background and basis for the paper. It will provide an overview of bisection method, its methodology, and research problem.
Section 2: This section examines the performance of bisection method for a cubic equation with discrete uniform coefficients through a statistical approach.
Section 3: This section examines the performance of bisection method for a cubic equation with continuous uniform coefficients through a statistical approach.
Section 4: This section examines the performance of bisection method for a cubic equation, with coefficients normally distributed, through a statistical approach.
Section 5: This section provides a summary of the results and makes recommendations for further study.

2. Study on Cubic Equation for Discrete Uniform Coefficients

2.1. Overview

The term "uniform distribution" in statistics refers to a kind of probability distribution wherein each potential outcome has an equal chance of occurring, i.e., the probability is constant while every variable has an equal number of chances of being the outcome.

For illustration, each person who passes by has an equal chance of receiving the 100-rupee note, if you were to start randomly

handing out the note while standing on a street corner. The probability as a percentage equals to 1/entire number of possibilities (the number of onlookers). On the other hand, the chances of short persons or women receiving the 100-rupee note are higher than those of other onlookers if you favor them. But this is not what is meant by uniform probability.

Based on the types of probable outcomes, uniform distribution can be divided into two categories: Discrete and Continuous. This section will cover the discrete uniform distribution in detail.

2.2. Discrete uniform distribution

The discrete uniform distribution is a statistical distribution in probability theory and statistics where the probability of outcomes is equal and has finite values, for instance the possible results of throwing a 6-sided die. 1, 2, 3, 4, 5, and 6 are the possible values. Each of the six numbers has a similar chance of appearing in this scenario. Consequently, each time the 6-sided die is thrown, each side gets a chance of 1/6.

There is a finite number of values. When rolling a fair die, it is impossible to obtain a value of 1.3, 4.2, or 5.7. The distribution, however, is no longer uniform if a second die is added, and they are both thrown, as the likelihood of the sums is not the same. The likelihood of tossing a coin is another straightforward illustration. There can only be two outcomes in such a situation. Consequently, 2 is the finite value.

To be more specific, take x to be a discrete random variable with η values in interval $[a,b]$. Let X has a discrete uniform distribution if its probability mass function (pmf) can be expressed as follows:

$$f(x) = \frac{1}{k}, \quad x = 1,\ 2,\ 3, \ldots, k \tag{2}$$

2.2.1. Expected value and variance

Two statistics which are often obtained are the expected value and the variance.

The discrete uniform random variable's expected value is given by:

$$\mathcal{E}(X) = \sum_{x=1}^{k} x.\text{Þ}(X = x) \tag{3}$$

which for discrete uniform variate X is

$$\mathcal{E}(X) = \frac{k+1}{2} \tag{4}$$

The expression for variance is given by:

$$\vartheta(X) = \mathcal{E}(X^2) - [\mathcal{E}(X)]^2 \tag{5}$$

where $\mathcal{E}(X^2)$ is given by:

$$\mathcal{E}(X^2) = \sum_{x=1}^{k} x^2.\ \text{Þ}(X = x) \tag{6}$$

which, in our case, gives

$$\sigma^2 = \frac{k^2 - 1}{12} \tag{7}$$

where σ is standard deviation.

2.3. Methodology

This section will provide the method that has been used to fulfill the objective of this section. Here, coefficients of cubic equation have been generated using the function *dis_unirand()* which includes inbuilt function *rand* and *rng.*

2.3.1. dis_unirand()

u=dis_unirand(r) generates random numbers from the discrete uniform distribution specified by r which implies the range upto which random number is generated, i.e.,

$[-r, r]$. The function *dis_unirand()* includes following functions:

rand:

$X = rand()$ generates a random scalar in the range (0,1) that is chosen at random from the uniform distribution.

Example:

$r = rand()$

$r = 0.8140$

$X = rand(\eta)$ generates a uniformly distributed η -by- η matrix of random integers.

Example:

$r = rand(3)$

$r = 3 \times 3$

0.0945 0.1526 0.8103

0.1220 0.9174 0.5069

0.9008 0.2705 0.6624

$X = rand(\eta_1, \eta_2 \ldots, \eta_n)$ generates an array of random numbers of size η_1-by-η_2-by- . . . .-by- η_n, where $\eta_1, \eta_2 \ldots, \eta_n$ indicate the size of each dimension.

Rng:

In random number generator, *rng* is used to change the seed in MATLAB to prevent using the same random numbers repeatedly. By producing a seed based on the current time, *rng* provides a simple method for doing that. "*Shuffle"* reseeds the generator with a different seed each time you use it. *rng* can be called without any inputs to reveal the exact seed that was used.

Example:

rng shuffle

$r = rand()$

$r = 0.1929$

INPUT ARGUMENTS

In MATLAB, the *"rand()"* function is used to obtain random numbers from a uniform distribution. The function can take one or two input arguments. When called with a single argument, *"η,"* *"rand(η)"* obtain a *"η"*-by-*"η"* matrix of random numbers uniformly distributed between 0 and 1.

Alternatively, when called with two arguments, *"η_1"* and *"η_2,"* *"rand(η_1, η_2)"* generates a *"η_1"*-by-*"η_2"* matrix of random numbers with same distribution. In both cases, the resulting matrix of random numbers is returned as the output of the *"rand()"* function.

MATLAB CODE—REFER Appendix A

2.3.2. Steps to follow in Matlab code

1) Enter the range of the random number as r and the value of tolerance is assumed to be 0.001.
2) Using the function *dis_unirand()*, the cubic equation's coefficients are generated at random from a discrete uniform distribution.

   Example: a_0=*dis_unirand(10)*
   $a_0 = -7$
   Similarly, other coefficients a_1, a_2, a_3 are generated.
3) The following conditions are checked with the while loop:
   i. If $a_3 = 0$, then a_3 is replaced by another value generated using *dis_unirand()* until the condition is false.
   ii. If $a_3 < 0$, then $a_0 = -a_0$, $a_1 = -a_1$, $a_2 = -a_2$, $a_3 = -a_3$ until the condition is false.
4) Now, the guess value is calculated as a and b such that $f(a).f(b) < 0$.
5) The values of $c = \frac{a+b}{2}$ and $f(c)$ are calculated.
6) An if-else conditional statement is now used to check a condition. If $f(c) < 0$, then $a = c$; otherwise, $b = c$.
7) Repeat step 5.
8) Increment the value of k by 1, which counts the number of iterations required to reach the root.
9) Steps 6, 7, and 8 have been repeated until the absolute value of $f(c)$ exceeds the tolerance value using a while loop.
10) An array called *arr* is used to record the final value of k—the number of iterations.
11) Simultaneously with the root, the respective values of a, b, c, and $f(c)$ are displayed.
12) Using a for loop, steps 2–11 are repeated 100 times.
13) The array of iterations (*arr*) has been displayed.
14) The *arr*'s mean and standard deviation are computed and displayed at the end.

2.4. Results

Following the above instructions in the MATLAB code stated in Appendix A, the mean and standard deviation of iteration (k) for a certain range [$-r$, r] of parameters over 100 trials were obtained, as shown in Table 1.

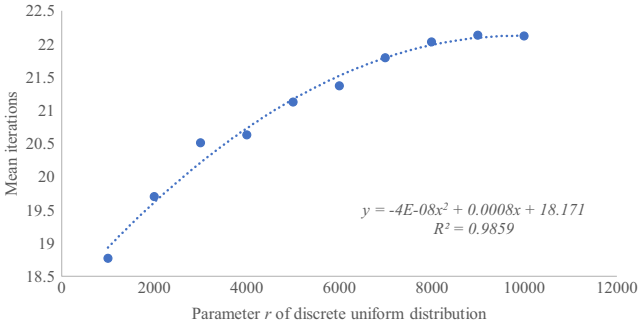
Table 1
Data of mean and standard deviation of iteration (k) for discrete uniform distribution

S. No.	Range of parameter (r)	Mean iteration (mean k)	Standard deviation of iteration
1.	1000	18.77	1.9584
2.	2000	19.7	1.9566
3.	3000	20.51	1.7085
4.	4000	20.63	1.8183
5.	5000	21.12	1.5194
6.	6000	21.37	1.8071
7.	7000	21.79	1.9914
8.	8000	22.03	1.9304
9.	9000	22.13	2.2322
10.	10,000	22.12	1.6894

2.5. Statistical analysis and discussion

Figure 1 gives the graph showing how mean iterations in bisection method varies w.r.t the parameters r of discrete uniform $U(-r, r)$ distribution characterizing the coefficients of cubic equations.

Figure 1
Graph showing mean iterations for bisection method versus parameter r of discrete uniform $U(-r, r)$ coefficients of a cubic equation



From Figure 1, it is clear that resultant second-degree polynomial equation can be used to predict the average iteration for a given parameter.

3. Study on Cubic Equation for Continuous Uniform Coefficients

3.1. Overview

Instead of being discrete, some uniform distributions are continuous. The most commonly used distribution among the two is a continuous uniform distribution. Every outcome in this distribution has an equal chance of appearing though the number of outcomes is infinite. In the previous section, discrete uniform distribution has been discussed broadly. Now, this section contains a detailed analysis of continuous uniform distribution.

3.2. Continuous uniform distribution

The random variable in a continuous uniform distribution, X, can have values between γ and δ (lower and upper bounds). Both γ and δ are referred to as the continuous uniform distribution parameters in the discipline of statistics. We cannot have a result that is either bigger than δ or smaller than γ. Every variable has an identical chance of appearing in a continuous uniform distribution, also known as a rectangle distribution, where the density function is constant or flat. It has infinite number of outcomes in a given range.

For example: The time it takes a student to finish a mathematics test ranges evenly between 30 and 60 min, despite the fact that there are an unlimited number of points between 30 and 60.

To be more specific, take x as a continuous random variable within the interval [γ, δ], X ~ U(γ, δ), if its probability distribution function (pdf) can be expressed as follows:

$$f_X(x) = \begin{cases} \frac{1}{\delta-\gamma}, & \gamma < x < \delta \\ 0, & x < \gamma \text{ or } x > \delta \end{cases} \quad (8)$$

3.2.1 Expected value and variance

The continuous uniform random variable's expected value is given by:

$$\mathcal{E}(X) = \int_{-\infty}^{\infty} x\, f_X(x)\, dx \quad (9)$$

Hence, the expected value is

$$\mathcal{E}(X) = \frac{\gamma + \delta}{2} \quad (10)$$

The expression for continuous uniform distribution's variance is given by:

$$\vartheta(X) = \mathcal{E}(X^2) - [\mathcal{E}(X)]^2 \quad (11)$$

where $\mathcal{E}(X^2)$ is given by:

$$\mathcal{E}(X^2) =^2 f_X(x)\, dx \quad (12)$$

Hence, the variance is

$$\sigma^2 = \frac{(\delta - \gamma)^2}{12} \quad (13)$$

where σ is standard deviation.

3.3. Methodology

This section will provide the method that has been used to fulfill the objective of this section. Here, coefficients of the cubic equation have been generated using the function *con_unirand()* which includes inbuilt function *rand* and *rng*.

3.3.1. con_unirand()

u=con_unirand(r) generates random numbers from the continuous uniform distribution specified by *r* which implies the range upto which random number is generated, i.e.,

[*-r, r*]. The function *con_unirand()* includes following functions:

rand:

X = rand() generates a random scalar in the range (0,1) that is chosen at random from the uniform distribution.

Example:

r = rand()

r = 0.8140

X = rand(η) generates a uniformly distributed *η* -by- *η* matrix of random integers.

Example:

r = rand(3)

r = 3×3

0.0945 0.1526 0.8103

0.1220 0.9174 0.5069

0.9008 0.2705 0.6624

X = rand(η1, η2 . . . , ηn) generates an array of random numbers of size η_1-by-η_2-by- . . . .-by- η_n, where $\eta_1, \eta_2 \ldots, \eta_n$ indicate the size of each dimension.

Rng:

In random number generator, *rng* is used to change the seed in MATLAB to prevent using the same random numbers repeatedly. By producing a seed based on the current time, *rng* provides a simple method for doing that. *"Shuffle"* reseeds the generator with a different seed each time you use it. *rng* can be called without any inputs to reveal the exact seed that was used.

Example:

rng shuffle

r = rand()

r = 0.1929

INPUT ARGUMENTS

In MATLAB, the *"rand()"* function is used to obtain random numbers from a uniform distribution. The function can take one or two input arguments. When called with a single argument, *"η,"* *"rand(η)"* obtain a *"η"*-by-*"η"* matrix of random numbers uniformly distributed between 0 and 1.

Alternatively, when called with two arguments, *"η_1"* and *"η_2,"* *"rand(η_1, η_2)"* generate a *"η_1"*-by-*"η_2"* matrix of random numbers with same distribution. In both cases, the resulting matrix of random numbers is returned as the output of the *"rand()"* function.

MATLAB CODE—REFER Appendix B

3.3.2. Steps to follow in Matlab code

1) Enter the range of the random number as *r* and the value of tolerance is assumed to be 0.001.
2) Using the function *con_unirand()*, the cubic equation's coefficients are generated at random from a continuous uniform distribution.

   Example: a_0=*con_unirand(10)*

   a_0 = 5.3115

   Similarly, other coefficients a_1, a_2, a_3 are generated.
3) The following conditions are checked with the while loop:
   i. If a_3=*0*, then a_3 is replaced by another value generated using *con_unirand()* until the condition is false.
   ii. If a_3<*0*, then a_0=- a_0, a_1=- a_1, a_2=- a_2, a_3=- a_3 until the condition is false.
4) Now, the guess value is calculated as *a* and *b* such that *f(a). f(b) <0.*
5) The values of $c = \frac{a+b}{2}$ and *f(c)* are calculated.
6) An if-else conditional statement is now used to check a condition. If *f(c)<0*, then *a = c*; otherwise, *b = c*.
7) Repeat step 5.
8) Increment the value of *k* by 1, which counts the number of iterations required to reach the root.
9) Steps 6, 7, and 8 have been repeated until the absolute value of *f(c)* exceeds the tolerance value using a while loop.
10) An array called *arr* is used to record the final value of *k*—the number of iterations.
11) Simultaneously with the root, the respective values of *a, b, c,* and *f(c)* are displayed.
12) Using a for loop, steps 2–11 are repeated 100 times.
13) The array of iterations (*arr*) has been displayed.
14) The *arr*'s mean and standard deviation are computed and displayed at the end.

3.4. Results

Following the above instructions in the MATLAB code stated in APPENDIX B, the mean and standard deviation of iteration (*k*) for

Table 2
Data of mean and standard deviation of iteration (k) for continuous uniform distribution

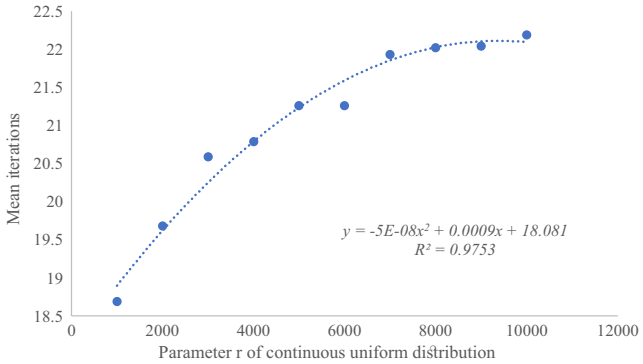
S.No.	Range of parameter (r)	Mean iteration (mean k)	Standard deviation of iteration
1.	1000	18.69	2.1728
2.	2000	19.68	1.9010
3.	3000	20.59	1.8592
4.	4000	20.79	1.6162
5.	5000	21.26	1.7034
6.	6000	21.26	2.2680
7.	7000	21.93	1.5971
8.	8000	22.02	1.7865
9.	9000	22.04	2.1692
10.	10,000	22.19	2.0582

a certain range $[-r, r]$ of parameters over 100 trials were obtained, as shown in Table 2.

3.5. Statistical analysis and discussion

Figure 2 gives the graph showing how mean iterations in bisection method vary w.r.t the parameters r of continuous uniform $U(-r, r)$ distribution characterizing the coefficients of cubic equations.

Figure 2
Graph showing mean iterations for bisection method versus parameter r of continuous uniform $U(-r, r)$ coefficients of a cubic equation



From Figure 2, it is clear that resultant second-degree polynomial equation can be used to predict the average iteration for a given parameter.

3.6. Comparison between discrete and continuous uniform distribution results

In discrete as well as continuous uniform distribution, second-degree polynomial equation can be used to predict the average iterations for a given parameter, i.e., second-degree polynomial is the best fit. On comparison, even the coefficients of both the second-degree polynomial are almost same which implies that the average iteration does not depend on whether the distribution is discrete or continuous but rather depend on the range of the distribution which is a parameter. Hence, in uniform distribution second-degree polynomial is the best fit.

4. Study on Cubic Equation for Coefficients Normally Distributed

4.1. Overview

The normal distribution, sometimes known as the Gaussian distribution, is the most important probability distribution in statistics for independent, random variables. Its well-known bell-shaped curve is readily noticed in statistics reports.

Most of the observations tend to cluster around the central peak of a normal distribution, which is a type of continuous probability distribution that is symmetrically distributed around its mean. The probability of obtaining values that are further away from the mean decreases at an equal rate in both directions, and extreme values in the tails of the distribution are also infrequent. It is worth noting that while the normal distribution is symmetrical, not all symmetrical distributions are normal.

The normal distribution is a probability distribution that describes how a variable's values are distributed, similar to other probability distributions. Due to its ability to accurately represent the distribution of values for many natural phenomena, it is considered the most important probability distribution in statistics. Normal distributions are commonly used to describe characteristics that are the result of multiple independent processes. For example, the normal distribution is commonly observed in traits such as height, blood pressure, measurement error, and IQ scores.

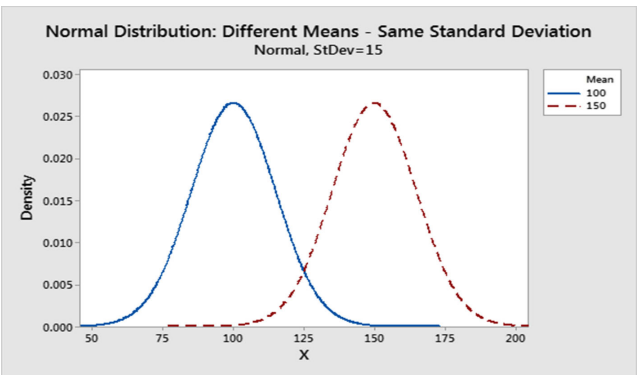
4.2. Parameters of normal distribution

The normal distribution's parameters ultimately determine its structure and probabilities, just like with any other probability distribution. The mean and standard deviation are the two variables that jointly form the normal distribution. There is not just one variant of the Gaussian distribution. Instead, the form alters according to the values of the parameter.

Mean:

The mean of the normal distribution represents its center of tendency and identifies the location of the peak of the bell curve. The majority of the data is clustered around the mean. When the

Figure 3
Graph shows the shift of the normal distribution curve as the mean changes



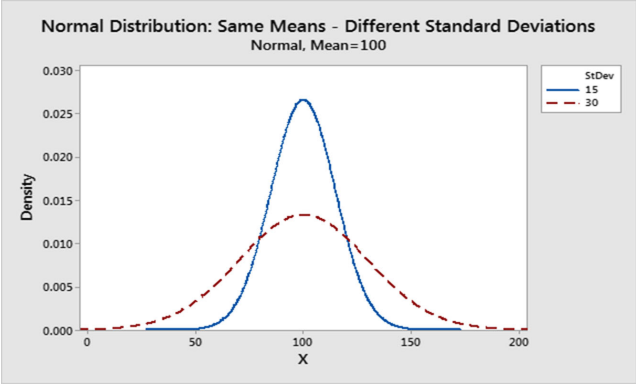
mean is changed on a graph, the entire curve shifts to the left or right on the X-axis, as illustrated in Figure 3 (Frost, 2023a).

Standard deviation:

The standard deviation is a measure of variability that determines the spread of the normal distribution. It quantifies how much the data typically deviate from the mean and represent the typical distance between observations and the mean. It is used to display the normal separation between the average and the observations, with a larger standard deviation indicating that the data are more spread out, while a smaller standard deviation indicates that the data are more tightly clustered around the mean.

As shown in Figure 4 (Frost, 2023b), altering the standard deviation causes the width of the distribution along the X-axis to either tighten or spread out. Wider distributions are produced by higher standard deviations.

Figure 4
Graph shows the structure of the normal distribution curve as the standard deviation changes



When distributions are narrow, there is a higher probability that values will not deviate significantly from the mean. Conversely, as the dispersion of the bell curve widens, there is a greater likelihood that observations will deviate further away from the mean. In other words, the risk of significant deviations from the mean increases as the normal distribution becomes more spread out.

4.3. Normal distribution probability density function

Normal distribution is obtained as a limiting case of binomial distribution. Let X be a binomial variate with parameters n and p. Let

$$Z = \frac{(X - np)}{\sqrt{npq}} \tag{14}$$

where $np=\theta$ and $\sqrt{npq}=\sigma$, which means

$$Z = \frac{(X-\theta)}{\sigma} \tag{15}$$

Naturally, Z is a standardized form of binomial variate. When the two limiting conditions:

i. $n\to\infty$
ii. p is neither small nor large

are applied on Z, then it can be shown that Z becomes a standard normal variate with probability density function (pdf) given by:

$$\varphi(z) = \frac{1}{\sqrt{2}} e^{\frac{-z^2}{2}},\ -\infty < z < \infty \tag{16}$$

$$\varphi(z)dz = \frac{1}{\sqrt{2\pi}} e^{\frac{-z^2}{2}} dz\ ,\ where\ \ dz = \frac{dx}{\sigma} \tag{17}$$

$\phi(z)$ is probability density function of Z; $\phi(z)dz$ is called probability differential of Z.

X becomes a normal variate with probability density function given by:

$$f = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\theta)^2}{2\sigma^2}}\ , where\ -\infty < x < \infty,\ -\infty < \theta < \infty,\ \sigma > 0 \tag{18}$$

$$f(x)dx = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\theta)^2}{2\sigma^2}} dx \tag{19}$$

$f(x)$ is probability density function of X; $f(x)dx$ is called probability differential of X.

X is a normal variate with mean θ and variance σ^2. In symbols, $X\sim N(\theta, \sigma^2)$.

Z is a standard normal variate. In symbols, $Z\sim N(0,1)$.

Normal distribution is so called because there was an attempt to project this distribution as the benchmark for all continuous probability distributions. The attempt failed as many continuous distributions turned out to be very different from normal. However, the nomenclature "normal" stayed.

4.4. Methodology

This section will provide the method that has been used to fulfill the objective of this section. Here, coefficients of the cubic equation have been generated using the function *normal()* which includes inbuilt function *rand* and *rng*.

4.4.1. normal()

u = *normal(m, sigma)* generates random numbers from the normal distribution specified by m and *sigma* which implies the mean and standard deviation, respectively. The function *normal()* includes following functions:

rand:

X = *rand()* generates a random scalar in the range (0,1) that is chosen at random from the uniform distribution.

Example:

r= *rand()*

$r = 0.8140$

X = *rand(η)* generates a uniformly distributed η -by- η matrix of random integers.

Example:

r = *rand(3)*

$r = 3\times3$

0.0945 0.1526 0.8103

0.1220 0.9174 0.5069

0.9008 0.2705 0.6624

X = *rand*($\eta_1, \eta_2 \ldots, \eta_n$) generates an array of random numbers of size η_1-by-η_2-by- . . . .-by- η_n, where $\eta_1, \eta_2 \ldots, \eta_n$ indicate the size of each dimension.

Rng:

In random number generator, *rng* is used to change the seed in MATLAB to prevent using the same random numbers repeatedly. By producing a seed based on the current time, *rng* provides a simple

method for doing that. "*Shuffle*" reseeds the generator with a different seed each time you use it. *rng* can be called without any inputs to reveal the exact seed that was used.

Example:
rng shuffle
r = rand()
r = 0.1929

INPUT ARGUMENTS

In MATLAB, the *"rand()"* function is used to obtain random numbers from a uniform distribution. The function can take one or two input arguments. When called with a single argument, *"η," "rand(η)"* obtain a *"η"*-by-*"η"* matrix of random numbers uniformly distributed between 0 and 1.

Alternatively, when called with two arguments, *"$η_1$"* and *"$η_2$," "rand($η_1$, $η_2$)"* generates a *"$η_1$"*-by-*"$η_2$"* matrix of random numbers with same distribution. In both cases, the resulting matrix of random numbers is returned as the output of the *"rand()"* function.

MATLAB CODE—REFER Appendix C

4.4.2. Steps to follow in Matlab code

1) Enter the value of mean as *m* and the value of standard deviation as *sigma,* and the value of tolerance is assumed to be 0.001.
2) Using the function *normal()*, the cubic equation's coefficients are generated at random from a normal distribution.

   Example: a_0 =*normal(100, 10)*
   a_0=*101.7212*
   Similarly, other coefficients a_1, a_2, a_3 are generated.
3) The following condition is checked with the while loop: If a_3=0, then a_3 and a_2 are replaced by another value generated using *normal()* until the condition is false.
4) Now, the guess value is calculated as *a* and *b* such that *f(a). f(b) <0.*
5) The values of $c = \frac{a+b}{2}$ and *f(c)* are calculated.
6) An if-else conditional statement is now used to check a condition. If *f(c)<0*, then *a=c*; otherwise, *b=c*.
7) Repeat step 5.
8) Increment the value of *k* by 1, which counts the number of iterations required to reach the root.
9) Steps 6, 7, and 8 have been repeated until the absolute value of *f(c)* exceeds the tolerance value using a while loop.
10) An array called *arr* is used to record the final value of *k*—the number of iterations.
11) Simultaneously with the root, the respective values of *a, b, c,* and *f(c)* are displayed.
12) Using a for loop, steps 2–11 are repeated 100 times.
13) The array of iterations (*arr*) have been displayed.
14) The *arr*'s mean and standard deviation are computed and displayed at the end.

4.5. Results

Following the above instructions in the MATLAB code stated in Appendix C, the mean and standard deviation of iteration (*k*), once for a fixed parameter—mean(*m*) and varied parameter—standard deviation(*sigma*) of normal distribution, as shown in Tables 3 and 4 and another for a fixed parameter—standard deviation(*sigma*) and varied parameter—mean(*m*) of normal distribution, as shown in Tables 5 and 6 over 100 trials were obtained.

Table 3
Data of mean and standard deviation of iteration (*k*) for normal distribution where *m* = 100 (fixed)

S.No.	Standard deviation (*sigma*) of normal distribution	Mean iteration (mean *k*) (over 100 trials)	Standard deviation of iteration (over 100 trials)
1.	2	15.6	1.4071
2.	4	15.78	1.5412
3.	6	15.72	1.3263
4.	8	15.65	1.6229
5.	10	15.62	1.6924
6.	12	15.69	1.4404
7.	14	15.87	1.2922
8.	16	15.78	1.4184
9.	18	15.65	1.9867
10.	20	15.55	1.6291

Table 4
Data of mean and standard deviation of iteration (*k*) for normal distribution where *m* = 30,000 (fixed)

S.No.	Standard deviation (*sigma*) of normal distribution	Mean iteration (mean *k*) (over 100 trials)	Standard deviation of iteration (over 100 trials)
1.	1000	23.72	1.7643
2.	2000	23.74	1.5349
3.	3000	23.81	1.4886
4.	4000	23.67	1.6457
5.	5000	23.9	1.2185
6.	6000	23.91	1.4005
7.	7000	23.97	1.4596
8.	8000	23.89	1.7402
9.	9000	23.89	1.7460
10.	10,000	24.13	1.3607

Table 5
Data of mean and standard deviation of iteration (*k*) for normal distribution where *sigma* = 2 (fixed)

S.No.	Mean(*m*) of normal distribution	Mean iteration (mean *k*) (over 100 trials)	Standard deviation of iteration (over 100 trials)
1.	100	15.67	1.4637
2.	200	16.48	1.3218
3.	300	17.31	1.5155
4.	400	17.78	1.5014
5.	500	17.93	1.4017
6.	600	18.24	1.3935
7.	700	18.5	1.4737
8.	800	18.63	1.3901
9.	900	18.85	1.1924
10.	1000	19.06	1.4759

Table 6
Data of mean and standard deviation of iteration (k) for normal distribution where *sigma* = 1000 (fixed)

S.No.	Mean(m) of normal distribution	Mean iteration (mean k) (over 100 trials)	Standard deviation of iteration (over 100 trials)
1.	10,000	22.46	1.2425
2.	20,000	23.24	1.5513
3.	30,000	24.06	1.4482
4.	40,000	24.34	1.5389
5.	50,000	24.63	1.5996
6.	60,000	24.85	1.4240
7.	70,000	25.07	1.4788
8.	80,000	25.12	1.8764
9.	90,000	25.44	1.2579
10.	100,000	25.61	1.3991

4.6. Statistical analysis and discussion

For Table 3, the equation obtained is $y = 15.691$, i.e., almost constant value is coming for a given set of inputs, where mean is fixed at 100 and standard deviation is varying for small intervals.

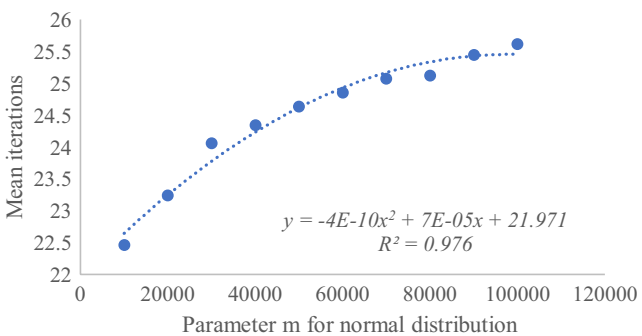
For Table 4, the equation obtained is $y = 23.863$, i.e., almost constant value is coming for a given set of inputs, where mean is fixed at 30,000 and standard deviation is varying for large intervals.

For Table 5, the equation obtained is $y = -4\text{E-}06x^2 + 0.0077x + 15.104$ and $R^2 = 0.9815$, i.e., second-degree polynomial equation can be used to predict the average iterations for a given parameter, where standard deviation is fixed at 2 and mean is varying for small intervals.

For Table 6, the equation obtained is $y = -4\text{E-}10x^2 + 7\text{E-}05x + 21.971$ and $R^2 = 0.976$, i.e., second-degree polynomial equation can be used to predict the average iterations for a given parameter, where standard deviation is fixed at 1000 and mean is varying for large intervals as shown in Figure 5.

For further literature on numerical methods and analysis, the reader may consult (Epperson, 2007).

Figure 5
Graph showing mean iterations for bisection method versus parameter *m* of a cubic equation for coefficients normally distributed



5. Conclusion and Future Scope

5.1. Conclusion

Having knowledge of statistics allows us to make informed decisions about the most appropriate data collection techniques, apply suitable statistical analyses, and effectively communicate the resulting findings. Making decisions based on data, predicting future outcomes, and making scientific discoveries all rely on statistical knowledge and skills. In other words, statistics is a crucial tool that allows us to draw meaningful conclusions from data and make evidence-based decisions, therefore applying statistical analysis on bisection method to analyze its performance in a cubic equation when coefficients are coming from particular distribution.

The convergence of the bisection method for solving a cubic equation will depend on the coefficients of the equation. Now, if the coefficients are coming from some probability distribution, then it is logical that the parameters of that probability distribution will influence the convergence. Hence, our study is to investigate in what way the parameter influences the convergence and considered both uniform and non-uniform distribution which includes discrete uniform distribution, continuous uniform distribution, and normal distribution. In order to generate random number from this distribution, various functions have been created like *dis_unirand()*, *con_unirand()*, *normal()* with the help of inbuilt functions *rand* and *rng* in MATLAB.

After analysis, in all the three distributions second-degree polynomial equation can be used to predict the average iteration for a given parameter. On the other hand, coefficients of the polynomial are almost same for discrete and continuous uniform distribution, whereas in the normal distribution the coefficients are different, which conclude the following:

i. In case of uniform distribution input, the average iteration does not depend on whether the distribution is discrete or continuous but rather depend on the range of the distribution which is a parameter.
ii. In case of non-uniform distribution input, the average iteration depends on the mean of the distribution but not on the standard deviation. Thus, it depends on the location parameter but not on the scale parameter.

Our statistical study of bisection method for cubic equations is intended to inspire other scholars to conduct related research.

5.2. Future scope of this work

Other distributions can also be used to study bisection method like Bernoulli distribution, exponential distribution, etc. Since the bisection method is guaranteed to converge, though it is slow, therefore a new improvised method can be generated to improve the rate of convergence of bisection method by combining it with the regula falsi method or the Newton-Raphson method. A similar statistical study of the new improvised method can be done along with comparison with the traditional method but such new methods would add new limitations to the data set (for instance, in case of the Newton-Raphson method, the first derivative of the polynomial should not be equal to 0) and therefore demands a confirmable new data set for the resultant output and comparison. Hence, it provides new scope to this work in future with more improvised algorithm and appropriate data set and makes it more reliable to find the root of any equation in real life for wider applications.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

References

Adler, A. (1983). On the bisection method for triangles. *Mathematics of Computation*, *40*(162), 571–574.

Chapra, S. C., & Canale, R. P. (1985). *Numerical methods for engineers: With personal computer applications.* USA: McGraw Hill.

Corliss, G. (1977). Which root does the bisection algorithm find? *SIAM Review*, *19*(2), 325–327. https://doi.org/10.1137/1019044.

Dichotomy Method – Encyclopedia of Mathematics (2015). Retrieved from: www.encyclopediaofmath.org.

Epperson, J. F. (2007). *An introduction to numerical methods and analysis*. USA: John Wiley & Sons.

Frost, J. (2023a). *Normal distribution: Different means-same standard deviations*. Retrieved from: https://statisticsbyjim.com/basics/normal-distribution/.

Frost, J. (2023b). *Normal distribution: Same means-different standard deviations*. Retrieved from: https://statisticsbyjim.com/basics/normal-distribution.

Graf, S., Novak, E., & Papageorgiou, A. (1989). Bisection is not optimal on the average. *Numerische Mathematik*, *55*(4), 481–491. https://doi.org/10.1007/BF01396051.

Gutierrez, C., Gutierrez, F., & Rivara, M. C. (2004). A geometric approach to the bisection method. In *LATIN 2004: Theoretical Informatics*, 172–180.

Kearfott, B. (1979). An efficient degree-computation method for a generalized method of bisection. *Numerische Mathematik*, *32*(2), 109–127. https://doi.org/10.1007/BF01404868.

Kreyszig, E. (2006). *Advanced engineering mathematics*. USA: John Wiley & Sons.

Morozova, E. (2008). A multidimensional bisection method for unconstrained minimization problem. In *Proceedings of the 14th Symposium on Computing: The Australasian Theory, 77*, 57–62.

Mourrain, B., Vrahatis, M. N., & Yakoubsohn, J. C. (2002). On the complexity of isolating real roots and computing with certainty the topological degree. *Journal of Complexity*, *18*(2), 612–640. https://doi.org/10.1006/jcom.2001.0636.

Novak, E. (1989). Average-case results for zero finding. *Journal of Complexity*, *5*(4), 489–501. https://doi.org/10.1016/0885-064X(89)90022-8.

Oliveira, I. F. D., & Takahashi, R. H. C. (2020). An enhancement of the bisection method average performance preserving minmax optimality. *ACM Transactions on Mathematical Software*, *47*(1), 1–24. https://doi.org/10.1145/3423597.

Sastry, S. S. (2003). *Introductory methods of numerical analysis*, India: PHI Learning Ptd.

Sikorski, K. (1982). Bisection is optimal. *Numerische Mathematik*, *40*(1), 111–117. https://doi.org/10.1007/BF01459080.

Sikorski, K (1985). Optimal solution of nonlinear equations. *Journal of Complexity*, *1*(2), 197–209. https://doi.org/10.1016/0885-064X(85)90011-1.

Solanki, C., Thapliyal, P., & Tomar, K. (2014). Role of bisection method. *International Journal of Computer Applications Technology and Research*, *3*(8), 533–535.

Vrahatis, M. N. (2020). Generalizations of the intermediate value theorem for approximating fixed points and zeros of continuous functions. In *NUMTA 2019: Numerical Computations: Theory and Algorithms*, 223–238. https://doi.org/10.1007/978-3-030-40616-5_17.

How to Cite: Kumari, S. & Chakraborty, S. (2024). Statistical Study of Bisection Method for Cubic Equations with Random Coefficients. *Archives of Advanced Engineering Science,* 2(1), 37–52, https://doi.org/10.47852/bonviewAAES32021322

Appendix A Matlab Code for Discrete Uniform Distribution

```
% Clearing Screen
clc

       % Input
        r= input('enter the range of random numbers as r for (-r,r):');
        e = 0.001;
        l=1;
        arr= zeros(1,100);



        % Initializing coefficients of polynomial

       for j= 1:1:100

       a0=dis_unirand(r); a1=dis_unirand(r);
       a2=dis_unirand(r); a3=dis_unirand(r);
        while a3==0
            a3=dis_unirand(r);
        end
        while a3<0
            a0=-a0; a1=-a1; a2=-a2; a3=-a3;
        end
        y= @(x)(a3*x^3+a2*x^2+a1*x+a0);

       % calculating guess value
       i=-1000;
       fa = y(i);
       fb = y(i+1);
       while fa*fb > 0
           i=i+1;
           fa = y(i);
           fb = y(i+1);
       end
           a=i;
           b=i+1;
```

```
% implementing bisection method
    k=1;
    c = (a+b)/2;
    fc = y(c);
    fprintf('\n\na\t\tb\t\tc\t\tf(c)\n');
    while abs(fc)>e
        fprintf('%f\t%f\t%f\t%f\n',a,b,c,fc);
        if fc< 0
            a =c;
        else
            b =c;
        end
        c = (a+b)/2;
        fc = y(c);
        k=k+1;
    end
    arr(1,l)=k;
    l=l+1;
    fprintf('%f\t%f\t%f\t%f\n',a,b,c,fc);
    fprintf('\nRoot is: %f\n', c);
end
fprintf('\niteration is:\n');
fprintf(' %d\t', arr);

 % finding mean and standard deviation of arr
 fprintf('\nMean of iterations:\n');
 disp(mean(arr));
 fprintf('\nStandard deviation of iterations:\n');
 disp(std(arr));

% generating R.N

function u= dis_unirand(r)
rng shuffle
u1=rand();
u=(-r)+ fix(u1*(2*r+1));
end
```

Appendix B Matlab Code for Continuous Uniform Distribution

```
% Clearing Screen
clc

       % Input
        r= input('enter the range of random numbers as r for (-r,r):');
        e = 0.001;
        l=1;
        arr= zeros(1,100);



        % Initializing coefficients of polynomial

       for j= 1:1:100

        a0=con_unirand(r); a1=con_unirand(r);
        a2=con_unirand(r); a3=con_unirand(r);
        while a3==0
            a3=con_unirand(r);
        end
        while a3<0
            a0=-a0; a1=-a1; a2=-a2; a3=-a3;
        end
        y= @(x)(a3*x^3+a2*x^2+a1*x+a0);

        % calculating guess value
        i=-1000;
        fa = y(i);
        fb = y(i+1);
        while fa*fb > 0
            i=i+1;
            fa = y(i);
            fb = y(i+1);
        end
            a=i;
            b=i+1;
```

```
        % implementing bisection method
            k=1;
            c = (a+b)/2;
            fc = y(c);
            fprintf('\n\na\t\tb\t\tc\t\tf(c)\n');
            while abs(fc)>e
                fprintf('%f\t%f\t%f\t%f\n',a,b,c,fc);
                if fc< 0
                    a =c;
                else
                    b =c;
                end
                c = (a+b)/2;
                fc = y(c);
                k=k+1;
            end
            arr(1,l)=k;
            l=l+1;
            fprintf('%f\t%f\t%f\t%f\n',a,b,c,fc);
            fprintf('\nRoot is: %f\n', c);
        end
        fprintf('\niteration is:\n');
        fprintf(' %d\t', arr);


        % finding mean and standard deviation of arr
        fprintf('\nMean of iterations:\n');
        disp(mean(arr));
        fprintf('\nStandard deviation of iterations:\n');
        disp(std(arr));

% generating R.N

function u= con_unirand(r)
        rng shuffle
        u=(-1)*r+(r-(-1)*r)*rand();
        end
```

Appendix C Matlab Code for Normal Distribution

```
% Clearing Screen
clc

       % Input
        m= input('enter the value of mean:');
        sigma= input('enter the value of Standard Deviation:');
        e = 0.001;
        l=1;
        arr= zeros(1,100);



        % Initializing coefficients of polynomial

       for j= 1:1:100

       arr2=normal(m, sigma);
       arr3=normal(m, sigma);
       a0=arr2(1,1); a1=arr2(1,2); a2=arr3(1,1); a3=arr3(1,2);

       while a3==0
            arr3=normal(m, sigma);
            a2=arr3(1,1); a3=arr3(1,2);
        end

        y= @(x)(a3*x^3+a2*x^2+a1*x+a0);

        % calculating guess value
        i=-1000;
        fa = y(i);
        fb = y(i+1);
        while fa*fb > 0
            i=i+1;
            fa = y(i);
            fb = y(i+1);
        end
            a=i;
            b=i+1;
```

```
        % implementing bisection method
            k=1;
            c = (a+b)/2;
            fc = y(c);
            fprintf('\n\na\t\tb\t\tc\t\tf(c)\n');
            while abs(fc)>e
                fprintf('%f\t%f\t%f\t%f\n',a,b,c,fc);
                if fc< 0
                    a =c;
                else
                    b =c;
                end
                c = (a+b)/2;
                fc = y(c);
                k=k+1;
            end
            arr(1,l)=k;
            l=l+1;
            fprintf('%f\t%f\t%f\t%f\n',a,b,c,fc);
            fprintf('\nRoot is: %f\n', c);
        end
        fprintf('\niteration is:\n');
        fprintf(' %d\t', arr);

        % finding mean and standard deviation of arr
        fprintf('\nMean of iterations:\n');
        disp(mean(arr));
        fprintf('\nStandard deviation of iterations:\n');
        disp(std(arr));

% generating R.N

function arr1= normal(m,sigma)
rng shuffle
arr1= zeros(1,2);
u1= rand();
u2= rand();
z1= sqrt(-2*log(u1))*cos(2*pi*u2);
z2= sqrt(-2*log(u1))*sin(2*pi*u2);
x1= m+z1*sigma;
x2= m+z2*sigma;
arr1(1,1)=x1;
arr1(1,2)=x2;
end
```