**TABLE OF CONTENTS:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CHAPTER**  **NO:** | **TITLE:** | | | | **PAGE NO:** |
|  | **LIST OF FIGURES** | | | | I |
|  | **ABSTRACT** | | | | II |
| 1 | **INTRODUCTION** | | | | 1 |
|  | 1.1 | | | Introduction | 2 |
|  | 1.2 | | | Software Cost Estimation | 2 |
|  | 1.3 | | | Problem Definition | 3 |
| 2 | **LITERATURE SURVEY** | | | | 4 |
|  | 2.1 | Related work of Software Cost Estimation | | | 5 |
|  | 2.2 | Introduction to Improved Environmental Adaption Method with Real Parameters | | | 6 |
|  | 2.3 | Description of fitness functions and evaluation metrics. | | | 6 |
|  | 2.4 | Software cost models | | | 8 |
| 3 | **METHODOLOGY** | | | | 10 |
|  | 3.1 | Blueprint | | | 11 |
|  | 3.2 | Description of the Algorithm | | | 12 |
| 4 | **IMPLEMENTATION** | | | | 21 |
|  | 4.1 | Software Used | | | 22 |
|  | 4.2 | Modules Imported in Python routines | | | 32 |
| 5 | **Program Results** | | | | 45 |
|  | 5.1 | | Program Results | | 46 |
| 6 | **Conclusion** | | | | 50 |
| 7 | **References** | | | | 52 |

**LIST OF FIGURES:**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO:** | **LIST:** | **PAGE NO:** |
| 1 | The model | 11 |
| 2 | Output-1 | 46 |
| 3 | Output-2 | 47 |
| 4 | Output-Graph | 48 |
| 5 | Comparison chart | 49 |

**ABSTRACT**

Environmental adaptation method (EAM) is one of the evolutionary algorithms for solving single objective optimization problems. After the first proposal of EAM, other variants have been suggested to speed up the convergence and to maintain the population diversity. Among them, IEAM-RP works with real numbers and was able to achieve the desired goal during the optimization process. In this paper, IEAM-RP is used to predict the effort required to develop the software product. The experiments are carried out on NASA software project dataset to check the effectiveness of IEAM-RP. The experimental results demonstrated that the overall performance of IEAM-RP is quite satisfactory in predicting the effort required to develop a software.

COCOMO II is an objective cost model for planning and executing software projects. It is an important ingredient for managing software projects or software lines of business. A cost model provides a framework for communicating business decisions among the stakeholders of a software effort. COCOMO II supports contract negotiations, process improvement analysis, tool purchases, architecture changes, component make/buy tradeoffs and several other return-on-investment decisions with a credible basis of estimate. COCOMO II incorporates several field-tested improvements to both broaden its applicability and improve its estimating accuracy for modern software development approaches. COCOMO II includes two underlying information models. The first is a framework for describing a software project, including models for process, culture, stakeholders, methods, tools and the size/complexity of the software product. The second is an experience base that can be used to estimate the likely includes significant updates to COCOMO to improve its applicability to modern processes, methods, tools and technologies. It also includes a much larger, more pertinent database of modern precedents and improves the adaptability of the model so it can be optimized across a broad spectrum of domains and project circumstances.

CHAPTER 1

INTRODUCTION

\

**1.1 Introduction :**

In the last three decades, many software cost estimation models have been suggested. This area is so significant that it has gained continuous research attention. Evaluating the project estimated cost, duration, and maintenance cost during the software development is a valuable goal that needs to be achieved to reduce overall development cost. The primary element that affects the estimated efforts is development line of code (DLOC). Boehm et al. , suggested one of the famous model to estimate the software effort in an efficient way. This model is called as COnstructive COst MOdel (COCOMO). In this model, a total of 63 software projects were used to evaluate the performance. In the literature, various techniques have been given for estimating the software development effort. These techniques involve soft computing, swarm-based algorithms, evolutionary algorithms, and many more. Among them, evolutionary algorithms were found successful in estimating the effort in an efficient way due to their population-based search technique. Shepperd et al. suggested an analogy-based system to estimate the software effort. Kumar et al. introduced a fuzzy logic and neural network-based model for software cost estimation. Kaczmarek et al. introduced a model for size and effort estimation for applications written in Java. Jeffery et al. suggested a software cost estimation model using public domain metrics. In recent years, some multiobjective techniques have also been suggested that optimize minimum two conflicting objectives (RMSE, MMRE etc.) simultaneously of cost estimation model. In addition to the application of cost estimation, some other applications of multiobjective techniques are surveyed in .

Recently a new evolutionary algorithm IEAM-RP has been suggested for solving single objective optimization problems. The performance of the suggested algorithm was checked using 24 benchmark functions of COCO framework. The performance of IEAM-RP was found quite satisfactory in terms of convergence rate and population diversity. IEAM-RP is also used to compute modularity value in . In this IEAM-RP is used to tune the parameters (a, b, c, d) of Sheta model for which MRE and MMRE are optimized as compared to existing models.

.

**1.2 Software Cost Estimation**

Software cost estimation is a prediction of the cost of the resources that will be required to complete all of the work of the software project. Software has a bad reputation about cost estimation. Large software projects have tended to have a very high frequency of schedule overruns, cost overruns, quality problems, and outright cancellations. Instead of it bad reputation, it is important to note that some large software projects are finished on time, stay within their budgets, and operate successfully when deployed. Currently a new generation of software processes and products is changing the way organizations develop software. The new approaches – evolutionary, risk driven and collaborative software processes; fourth generation languages and application generators; commercial off the shelf (COTS) and reuse driven software approaches; fast track software development approaches; software process maturity initiatives – lead to significant benefit in terms of improved software quality and reduced software cost, risk and cycle time. COCOMO II model tailored to these new forms of software development, including rationales for the model decisions. The major new modeling capabilities of COCOMO II are a tailorable family of software sizing models, involving Object Points, Function Points, and Source Lines of Code; nonlinear models for software reuse and reengineering; an exponent-driver approach for modeling relative software diseconomies of scale; and several additions, deletions, and updates to previous COCOMO effort-multiplier cost drivers. This model is serving as a framework for an extensive current data collection and analysis effort to further refine and calibrate the model’s estimation capabilities.

**1.3 Problem Definition :**

**Vision:**

* The project aims at developing a code for Software Cost Estimation using Cocomo Model

**Mission:**

* This tool is developed by using Python

CHAPTER 2

LITERATURE SURVEY

**2.1. Related work of software cost estimation**

Boehm et al. introduced a non-linear mode of assessment method in 1981, called as COCOMO (COnstructive COst MOdel). This model was divided into three different levels, namely, basic model, intermediate model, and detailed model. This model is widely used to evaluate the software effort. Other techniques have also been suggested in the past for estimating the effort of a software project. Soft computing methods were explored to build efficient effort estimation models structures. Kelly utilized the concept of neural networks, genetic algorithms, and genetic programming to introduce a methodology for software cost estimation. Later, Dolado et al. provided a detailed study on using neural networks, genetic programming, and linear regression in solving the project effort estimation. In many datasets were given for software cost estimation. Kumar et al. introduced a fuzzy logic and neural network-based software cost estimation model. Recently, machine learning and soft computing techniques were explored to handle effort and cost estimation problems. The author provided an approach for a different set of models modified from the COCOMO model using genetic algorithms. After the proposal of many authors explored the same idea with some modification. Authors have provided a comparison of results to the work presented in. Nature inspired algorithms were found more suitable for solving software cost estimation problems due to their population based search technique. Many nature-inspired algorithms were suggested for cost estimation of a software project in recent years. Mishra et al. introduced a memetic algorithm for software cost estimation. Authors suggested a genetic algorithm-based parameter tuning approach for COCOMO II model for estimating software development effort. Lin et al. introduced a particle swarm optimization-based search technique to estimate software effort. Aljahdali et al. proposed a differential evolution-based software cost estimation technique for tuning the COCOMO model parameters.

**2.2. Introduction to Improved Environmental Adaptation Method with Real Parameter (IEAM-RP):**

The IEAM-RP is one of the evolutionary algorithms that is used to solve single objective optimization problems. This algorithm is based on the principle of adaptive learning developed by J. M. Baldwin . This algorithm achieves the principle of adaptive learning using its operators, adaptation, and selection. Adaptation operator basically receives parent population and creates offspring. In order to create offspring, adaptation operator divides the population into two classes. One class contains only one (best) solution, whereas another class contains remaining solutions. The best solution updates its position vector using its personal fitness and average fitness of the population as follows:

Pb+1 = Pb × F(Pb)/Favg + β (1)

Here, β is a random value between 0 to 1. Pb and Pb+1 are the old and updated position vectors of the best solution. F(Pb) is the fitness value of Pb and Favg is the average fitness of the population. Here, the symbol / is used for division operator. The remaining solutions receive the direction from the best solution and try to achieve a better phenotypic structure in the problem search space as follows:

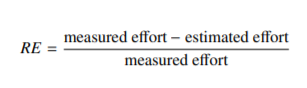
Pnb+1 = Pnb + β × (best\_position − worst\_position) (2) Here, Pnb, and Pnb+1 are the old and updated position vectors of the non-best solutions. In minimization problems, a solution with least fitness is called as the best solution, whereas a solution with maximum fitness is called as the worst solution. The difference of the best\_position to the worst\_position is the positional bandwidth of the best solution to the worst solution. The non-best solutions utilize this bandwidth to exploit in the problem search space. The left-hand side of equation 1, and 2 are the offspring corresponding to the best, and non-best solutions, respectively. In order to get the offspring corresponding to the population, we need to combine left-hand side of equations 1, and 2. The selection operator combines parent and offspring to get N best solutions. Here, N is the size of the initial population.

**2.3 Description of fitness functions and evaluation metrics**

MRE, MMRE, and PRED (L)] are used for cost estimation model. Here, MRE, and MMRE are taken as fitness functions that need to be minimized. The mathematical expressions of these measurement techniques are given below.

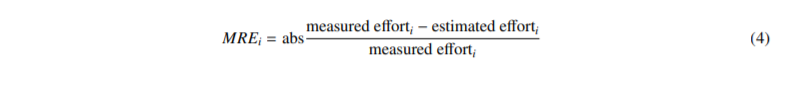
**2.3.1 Relative Error (RE)**

This error is one of the measurement technique that is used to measure the quality of a model. This value should be minimum for a better estimation model. The RE can be measured as follows:

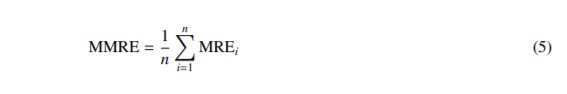
 

**2.3.2. Magnitude of Relative Error (MRE) :**

The absolute value of RE of the respective project is called as the magnitude of relative error, computed as follows:

**2.3.3 Mean Magnitude of Relative Error (MMRE):**

The absolute value of RE of the respective project is called as the magnitude of relative error, computed as follows:



**2.3.4 Prediction at Some Level (PRED):**

This measurement technique evaluates the performance of an estimation model. The PRED with prediction at level L can be computed as follows:

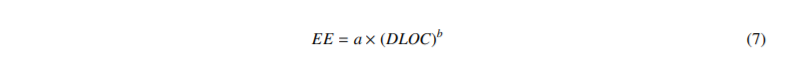


where L is the limit for k, whereas k represents the total number of observations less than or equal to L. Here, n is the total number of observations. In this paper, the value of L is generally taken as 0.25 for the measurement. The minimum value of MMRE and the maximum value of PRED is desirable for an estimation model.

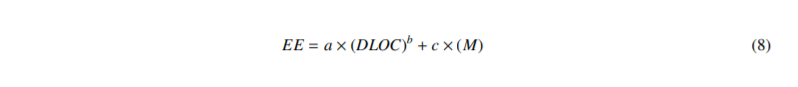
**2.4 Software Cost Estimation Models:**

Sheta proposed evolutionary models using genetic algorithms (GAs) for estimating the software effort . He applied GA to estimate the parameters of COCOMO effort estimation model. The performance of the proposed models was tested on 18 software project dataset taken from NASA . The magnitude of relative error (MRE) and mean magnitude of relative error (MMRE) were considered as the fitness functions to evaluate the performance of the cost estimation models. Sheta has given three models to calculate estimated efforts that are given below.

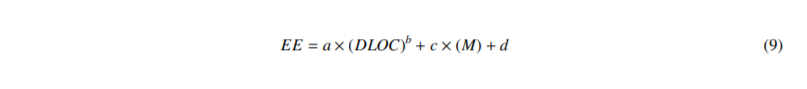
Model 1: This model uses two parameters, a, and b with a developed line of code (DLOC). The estimated effort (EE) for this model can be computed as follows:



Model 2: This model uses three parameters, a, b, and c with DLOC, and methodology (M) to compute estimated effort. Here, methodology is used to improve the prediction capability of the basic COCOMO model. The value of methodology for each software project is given in table 1. The estimated effort is computed as follows:



Model 3: This model uses four parameters (a, b, c, d) with DLOC and methodology to compute estimated effort as follows:



DLOC and M are given, the parameters are optimized in such a way that MRE and MMRE should be minimized. Boundary range of parameters of Sheta model is given below.

Search Domain for a {0, 10}

Search Domain for b {0.3, 2}

Search Domain for c {-0.5, 0.5}

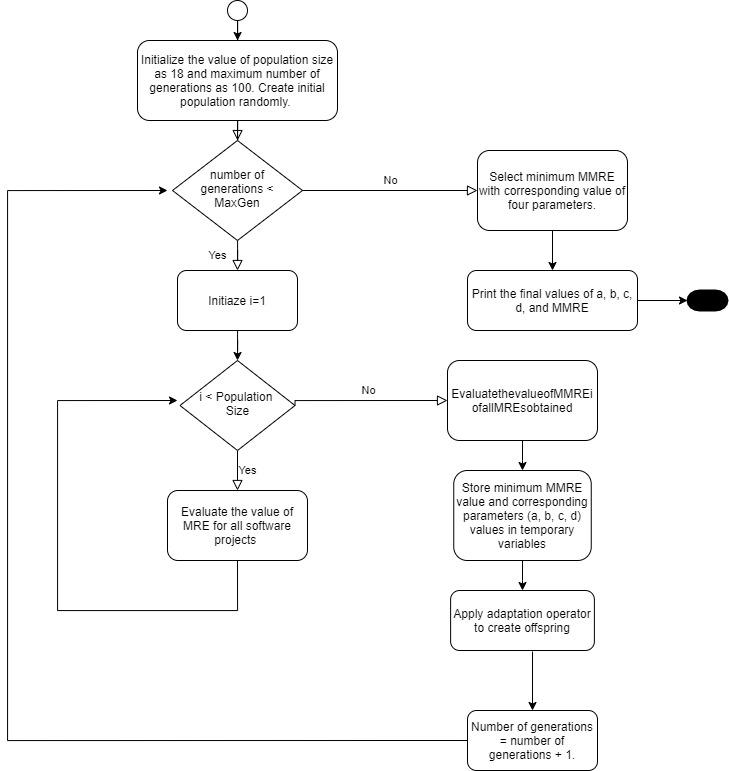
Search Domain for d {0, 20}.

The objective of the proposed technique is to tune the above parameters that provide most accuracy in cost estimation.

CHAPTER 3

METHODOLOGY

**3.1 BLUEPRINT**



**THE MODEL**

**3.2 DESCRIPTION OF THE ALGORITHM**

IEAM-RP is used for tuning the parameters of Sheta model for software effort estimation. The algorithm starts with a randomly initialized population of the specified size in the boundary range. Each individual of the population has four parameters (a, b, c, d). These parameters need to be estimated in such a way that the final MMRE value should be minimized. Random initialization of the individuals gives a matrix of size N×D. Here, N is the population size and D is the dimension size of the problem taken under consideration. In this paper, N = 18 and D = 4. After random initialization of the population, parametric values of the first row are used to compute EE and MRE for all software projects. Afterward, the average of all MREs i.e. MMRE is computed. The value of MMRE and the corresponding values of a, b, c, and d are stored in a temporary memory. This process will be continued for rest of the row vectors, and MMRE, four parametric values are computed. Finally, the minimum value of MMRE and the corresponding value of four parametric values are saved in temporary variables. After first generation, the four parametric values are updated using equations 1 and 2. These updated values make the offspring of size N for parent population. The parent population and offspring are combined and N best individuals are selected for the next generation. Now, these individuals are used to create offspring in the same way as discussed above. At the end of the program execution, the minimized value of MMRE and the corresponding value of parameters (a, b, c, d) are stored. Finally, the optimum value of MMRE and the corresponding value of four parameters will be obtained.

**3.2.1 ALGORITHM STEPS**

Step 1: Initialize the value of population size as 18 and maximum number of generations as 100. Create initial population randomly.

Step 2: While (number of generations < MaxGen)

Repeat the following steps 3 to 8 until the specified number of generations (MaxGen) has not reached.

Step 3: For each row vector of parameter values in the matrix do

for i=1, ... ,PS

Evaluate the value of MMREi of all MREs obtained using equation 5.Here

MMREi is the mean value of MREs of all software projects for i t individual of the population.

end for

Step 4: Store minimum MMRE value and corresponding parameters (a, b, c, d) values in temporary variables.

Step 5: Apply adaptation operator to create offspring.

Step 6: Initial/Intermediate population and offspring are combined.

Step 7: Apply selection operator on combined population to select N best solutions.

Step 8: Number of generations = number of generations + 1.

Step 9: End while.

Step 10: Select minimum MMRE with corresponding value of four parameters.

Step 11: Print the final values of a, b, c, d, and MMRE.

After termination of the above algorithm, the optimized value of MMRE with corresponding values of four parameters are obtained. On the basis of four parameters, the values of EE, and MRE can be calculated.

**3.2.2 Code Explanation :**

That's the basic logic of using Qt Designer and PyQt to design and develop a GUI application.

**1.generate()-**

This function is used to generate random values for the parameters

1.Production rate(a)

2.Scale Factor (b)

3.Cost Multiplier (c)

4.Effort drivers (d)

**def generate():**

a=round(random.uniform(-5,5),3)

b=round(random.uniform(-5,5),3)

c=round(random.uniform(-5,5),3)

d=round(random.uniform(-5,5),3)

return a,b,c,d

Search Domain for

a → {-5,5}

b→ {-5,5}

c→ {-5,5}

d→ {-5,5}

The function takes no arguments and returns the values of a,b,c,d which are generated in the function.

**2. get\_length(dataset) -**

This function returns number of rows of data in a data set or the length of the data set.It takes the dataset as argument and returns the length of data set.

def get\_length(sample\_dataset):

return len(sample\_dataset[0])

**3. generate\_dataset(l)**

This function creates population vectors by generating random values for a,b,c and d by utilizing the generate() user defined function and appending them for every iteration to get **l** population vectors.

The function takes one parameter l – length of the dataset I.e number of rows in the data set and returns n,which is a vector of populations.

def generate\_dataset(l):

n=[]

for i in range(l):

n.append(list(generate()))

return n

**4.** **formula(l,size,methodology)**

This method is used to calculate effort for each population using the formula

EE = a×(DLOC)b + c×(M) + d

It takes three parameters as inputs-

1. List of a,b,c,d values (l)

2. Size of the project(size)

3.Methodology used (methodology)

def formula(l,size,methodology):

#Effort Calculation for each population using formula:EE = a×(DLOC)b + c×(M) + d

a,b,c,d=l[0],l[1],l[2],l[3]

temp=pow(size,b)

temp=a\*temp+c\*methodology+d

return round(temp,3)

The function returns the effort for each population rounded to three digits.

**5. generate\_EE(data\_of\_abcd,dataset,length\_of\_dataset)**

This function is used to calculate the estimated effort for all the rows in the dataset over different values of a,b,c,d

It takes three parameters, the matrix containing abcd values ,the data set and the length of the data set used

def generate\_EE(data,dataset,l):

EE=[]

for j in range(l):

temp=[]

for i in range(l):

temp.append(formula(data[j],dataset[0][i],dataset[1][i]))

EE.append(temp)

return EE

It uses temp array to store all the estimated effort for one a,b,c,d value and EE array to store all the estimated effort for each value in data set over different range of a,b,c,d values as in the a,b,c,d matrix and returns Estimate Effort

**6. print\_data(n) :**

This function is just used to print the data .It takes array(n) as parameter and prints the array contents over its length.

def print\_data(n):

for i in range(len(n)):

print(n[i],end="\n")

**7. generate\_mre(measured effort,estimated effort) :**

This function is used to calculate the magnitude of relative error(MRE).MRE is the difference of measured effort and estimated effort divided by measured effort.

It takes two parameters measured effort array and estimated effort array.

It returns magnitude of relative error .It uses temp to store all mre over a single a,b,c,d value.

It uses mre to store all mre over different range of a,b,c,d values.

def generate\_mre(me,ee):

mre=[]

for j in range(len(ee[0])):

temp=[]

for i in range(len(ee)):

temp.append(round(abs((me[i]-ee[j][i])/me[i]),3))

mre.append(temp)

return mre

**8. generate\_mmre(mre\_list) :**

This function is used to calculate Mean of Magnitude of Relative Error(MMRE).

It takes an array containing mre’s of all rows in dataset over different ranges of a,b,c,d which is used to calculate MMRE

MMRE of a dataset for corresponding a,b,c,d is the sum of all the mre values in that data set to length of the dataset.

def generate\_mmre(mre):

#calculation of MMRE(Mean of Magnitude of relative error)

avg1=[]

for i in range(len(mre)):

av=round(sum(mre[i])/len(mre[i]),3)

avg1.append(av)

return avg1

It returns mmre for different a,b,c,d values over the data set.

**8. find\_avg(mmre\_list) :**

It is used to calculate the average of Mean of Magnitude of Relative Error (MMRE)

It takes list of mmre’s and returns its average.

def find\_avg(mmre):

#Fitness calculation(Average of MMRE)

for i in range(len(mmre)):

av=round(sum(mmre)/len(mmre),3)

return av

9.**find\_adaption(best\_data,bestfit\_value,average,beta)**

This function is a part of the adaption phase where we find the next best solution using the current solution.The method takes 4 parameters-

1. best\_data – the a,b,c,d row with the least MMRE(fitness) value

2. bestfit\_value-The lowest fitness(MMRE) value

3.average – average of all the MMRE values for each a,b,c,d value

The formula for adaption is -

Pb+1 = Pb ×F(Pb)/Favg + β

def find\_adaption(data,fit,avg,beta=0.5):

temp=fit/avg

for i in range(len(data)):

data[i]=round(data[i]\*temp,3)

data[i]=data[i]+beta

return data

**10. do\_aternation(new\_data,best\_data,worst\_data.beta)**

It is used to generate the solution using other a,b,c,d values besides the best solution

Pnb+1 = Pnb + β×(best\_position−worst\_position)

It takes a,b,c,d matrix,best a,b,c,d data and the worst a,b,c,d values and beta as inputs and returns

new a,b,c,d matrix

def do\_aternation(new\_data,best\_data,worst\_data,beta=0.5):

#Offspring Generation [Pnb+1 = Pnb + β×(best\_position−worst\_position) ]

temp=[]

for i in range(len(best\_data)):

temp.append(round(beta\*(best\_data[i]-worst\_data[i]),3))

for i in range(len(new\_data)):

for j in range(len(temp)):

new\_data[i][j]=round(new\_data[i][j]+temp[j],3)

return new\_data

**11. filtrate(abcd\_data,size of data):**

To filterate the a,b,c,d values and randomize them to be in their actual ranges,we use filterate method.It takes two parameters ,abcd matrix and size of the matrix and returns filtered data.

def filterate(new\_data,n):

data=new\_data

for i in range(0,len(new\_data)):

for j in range(0,n):

if new\_data[i][j]<-5 or new\_data[i][j]>5:

data[i][j]=round(random.uniform(-5,5),3)

return data

**2. plot(bestfit\_values):**

To generate a plot for each best fit value in each iteration,we use plot function.It takes best fit values as input and visualizes a graph of iterations vs best fit along x and y axes.

def plot(best\_fit\_values):

best\_fit\_values=pd.DataFrame(best\_fit\_values)

best\_fit\_values.columns=['best\_fit value']

df=best\_fit\_values

c=df.count()[0]

ax=df.plot(figsize=(10,8),markersize=22)

plt.scatter(x=list(range(0,c)),y=best\_fit\_values,s=80)

ax.set\_title('Best fit values for every iteration',size=12)

plt.plot(df[9:], 'or')

ax.legend('')

plt.grid()

ax.set\_xlabel('iterations',size=15)

ax.set\_ylabel('best fit',size=15)

ax.tick\_params(labelsize=14)

k=0

for i in df.values:

ax.annotate(str(i),(k,i+0.01),color='black')

k+=1

CHAPTER 4

IMPLEMENTATION

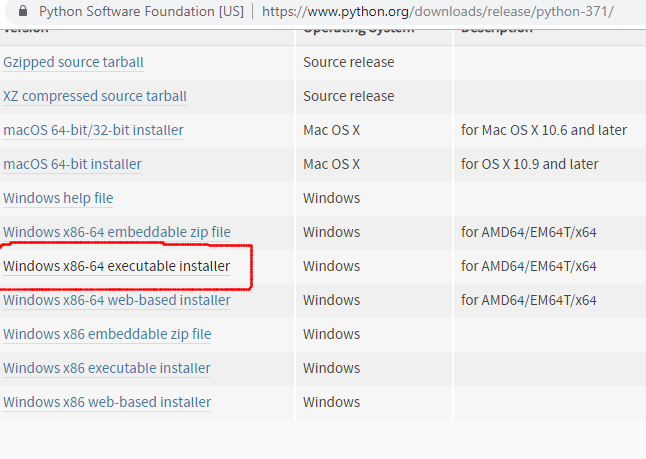
**4.1 Software Used :**

**4.1.1 Windows Installation :**

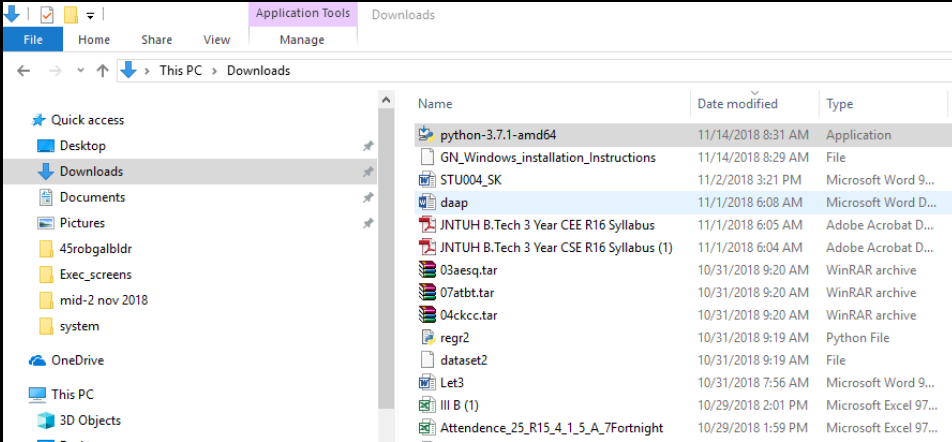
Installation Process in Windows:

1) If the system don't have Python, then install Python by down loading it from the following site.

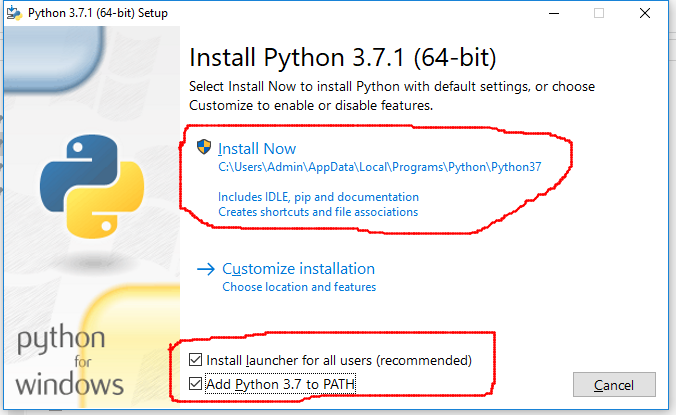
[https://www.python.org/downloads/release/python-374/](https://www.python.org/downloads/release/python-370/)

1.a) Go to the above site, come to the bottom of the page, and click on ‘ Windows x86-64 executable Installer ‘, as shown in the above figure.

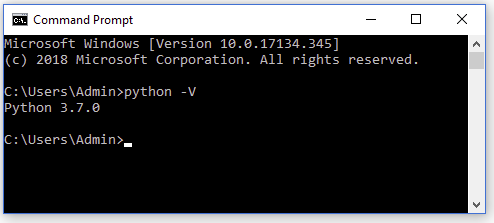
1.b) Python Application will be downloaded into the ‘Downloads’ folder as shown in the following figure.



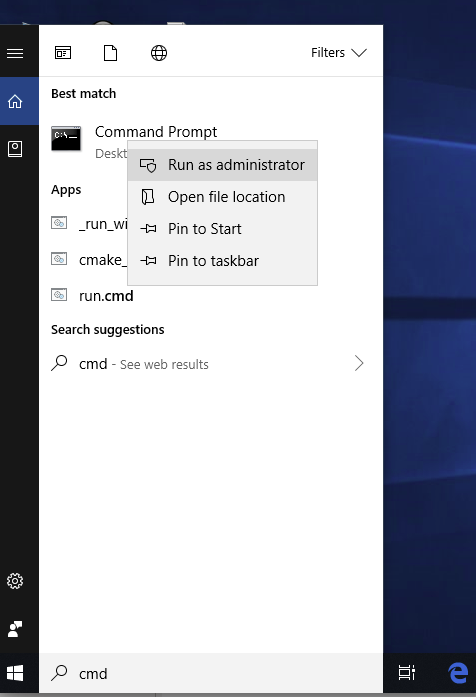
1.c) Click on python ‘ python-3.7.4-amd64’ to get the following screen. Select both the options at the bottom of the screen, and click on “Install Now”.



1.d) Follow the installation process, and, finish it. After completion, open command prompt, and give the command python -V, as shown below.

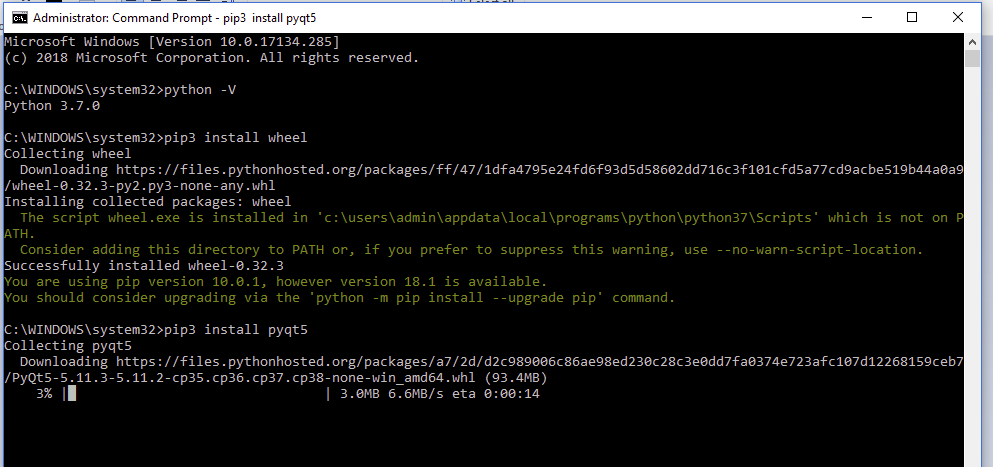


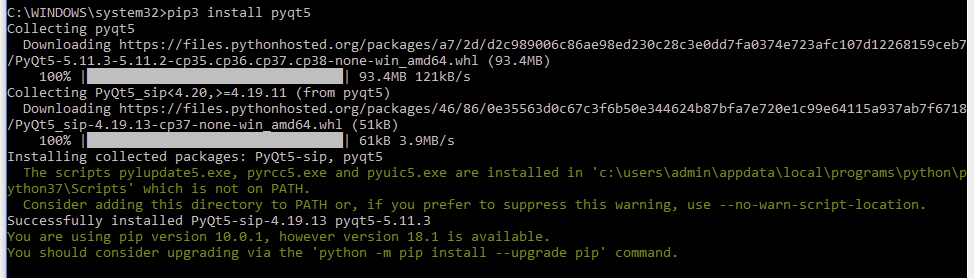
2) Close the above command prompt, and re-open it as an administrator, as shown in the following figure.



(Note: When you RIGHT-CLICK on command prompt, the submenu will be opened. From there, you can run it as administrator.)

Execute the command from command prompt: pip3 install pyqt5. You should get the pyqt5 successfully, installed as shown in the following two figures. Otherwise, discuss with us, when we meet next time.





**4.1.2 Languages :**

**Python :**

Python was conceived in the late 1980s, and its implementation began in December 1989 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing.

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations.

**Features And Philosophy**

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta programming and meta objects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

**Sample Code**

**Python Code**

import random

import math

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read\_excel('Book1 (1).xlsx',header=2)

sample\_dataset=[]

Size= df['Size'].tolist()

Meth=df['Methodlogy'].tolist()

me=df['Measurd Effort'].tolist()

sample\_dataset=[]

sample\_dataset.append(Size)

sample\_dataset.append(Meth)

sample\_dataset.append(me)

print(sample\_dataset)

#sample\_dataset=[[90,46.2,46.5,54.5,31,67.5],[30,20,19,20,35,29],[115,96,79,90,39.6,98.4]]

new\_population=[]

def generate():

#assigning random values for parameters(a,b,c,d)

#Search Domain for a {-5,5} Search Domain for b {-5,5} Search Domain for c {-5,5} Search Domain for d {-5,5}.

a=round(random.uniform(-5,5),3)

b=round(random.uniform(-5,5),3)

c=round(random.uniform(-5,5),3)

d=round(random.uniform(-5,5),3)

return a,b,c,d

def get\_length(sample\_dataset):

#returns the number of data sets

return len(sample\_dataset[0])

def generate\_dataset(l):

#generation of random values for Parameters (a,b,c,d) and creating Population Vectors

n=[]

for i in range(l):

n.append(list(generate()))

return n

def formula(l,size,methodology):

#Effort Calculation for each population using formula:EE = a×(DLOC)b + c×(M) + d

a,b,c,d=l[0],l[1],l[2],l[3]

temp=pow(size,b)

temp=a\*temp+c\*methodology+d

return round(temp,3)

def generate\_EE(data,dataset,l):

# Calculating Effort for all the projects in data set

EE=[]

for j in range(l):

temp=[]

for i in range(l):

temp.append(formula(data[j],dataset[0][i],dataset[1][i]))

EE.append(temp)

return EE

def print\_data(n):

for i in range(len(n)):

print(n[i],end="\n")

def generate\_mre(me,ee):

#calculation of MRE(Magnitude of relative error)

mre=[]

for j in range(len(ee[0])):

temp=[]

for i in range(len(ee)):

temp.append(round(abs((me[i]-ee[j][i])/me[i]),3))

mre.append(temp)

return mre

def generate\_mmre(mre):

#calculation of MMRE(Mean of Magnitude of relative error)

avg1=[]

for i in range(len(mre)):

av=round(sum(mre[i])/len(mre[i]),3)

avg1.append(av)

return avg1

def find\_avg(mmre):

#Fitness calculation(Average of MMRE)

for i in range(len(mmre)):

av=round(sum(mmre)/len(mmre),3)

return av

def find\_adaption(data,fit,avg,beta=0.5):

#Application of Adaption[ Pb+1 = Pb ×F(Pb)/Favg + β]

temp=fit/avg

for i in range(len(data)):

data[i]=round(data[i]\*temp,3)

data[i]=data[i]+beta

return data

def do\_aternation(new\_data,best\_data,worst\_data,beta=0.5):

#Offspring Generation [Pnb+1 = Pnb + β×(best\_position−worst\_position) ]

temp=[]

for i in range(len(best\_data)):

temp.append(round(beta\*(best\_data[i]-worst\_data[i]),3))

for i in range(len(new\_data)):

for j in range(len(temp)):

new\_data[i][j]=round(new\_data[i][j]+temp[j],3)

return new\_data

def filterate(new\_data,n):

data=new\_data

for i in range(0,len(new\_data)):

for j in range(0,n):

if new\_data[i][j]<-5 or new\_data[i][j]>5:

data[i][j]=round(random.uniform(-5,5),3)

return data

def plot(best\_fit\_values):

best\_fit\_values=pd.DataFrame(best\_fit\_values)

best\_fit\_values.columns=['best\_fit value']

df=best\_fit\_values

c=df.count()[0]

ax=df.plot(figsize=(10,8),markersize=22)

plt.scatter(x=list(range(0,c)),y=best\_fit\_values,s=80)

ax.set\_title('Best fit values for every iteration',size=12)

plt.plot(df[9:], 'or')

ax.legend('')

plt.grid()

ax.set\_xlabel('iterations',size=15)

ax.set\_ylabel('best fit',size=15)

ax.tick\_params(labelsize=14)

k=0

for i in df.values:

ax.annotate(str(i),(k,i+0.01),color='black')

k+=1

def main():

bestfits=[]

best\_param=[]

best\_MMRE=[]

new\_population=generate\_dataset(get\_length(sample\_dataset))

print("------------Initial Population----------",end="\n")

print\_data(new\_population)

i=0

best\_fit=9999

pre\_best=9999

while(i<100):

i=i+1

print("--------------EE calculated-----------",end="\n")

EE=generate\_EE(new\_population,sample\_dataset,get\_length(sample\_dataset))

print("--------------MRE calculated-------------",end="\n")

MRE=generate\_mre(sample\_dataset[2],EE)

print(MRE)

MMRE=generate\_mmre(MRE)

print("--------------MMRE calculated------------",end="\n")

pre\_best=best\_fit

best\_fit=min(MMRE)

best\_MMRE.append(best\_fit)

row\_best=MMRE.index(best\_fit)

print("best row =="+str(row\_best))

best\_data=new\_population[row\_best]

best\_param.append(best\_data)

average\_fitness=round(find\_avg(MMRE),3)

row\_data=find\_adaption(best\_data,best\_fit,average\_fitness,0.5)

new\_data=new\_population

worst\_fit=max(MMRE)

row\_worst=MMRE.index(worst\_fit)

worst\_data=new\_population[row\_worst]

del new\_data[row\_best]

new\_data=do\_aternation(new\_data,best\_data,worst\_data,0.5)

new\_data.append(best\_data)

print("NEWDATA\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

print(new\_data)

print("-----------values---------------")

new\_pop=row\_data+new\_data

print("data after iter:",i)

new\_population=filterate(new\_data,get\_length(new\_data))

print(new\_population)

bestfits.append(best\_fit)

print("Best MMRE=")

print(min(best\_MMRE))

print("\nBest data\n")

print(best\_param[best\_MMRE.index(min(best\_MMRE))])

print("\n")

plot(bestfits)

main()

**4.2 Modules Imported in Python routines:**

**4.2.1 Pandas Module:**

Pandas is an open source library in Python. It provides ready to use high-performance data structures and data analysis tools.

Pandas module runs on top of NumPy and it is popularly used for data science and data analytics.

NumPy is a low-level data structure that supports multi-dimensional arrays and a wide range of mathematical array operations. Pandas has a higher-level interface. It also provides streamlined alignment of tabular data and powerful time series functionality.

DataFrame is the key data structure in Pandas. It allows us to store and manipulate tabular data as a 2-D data structure.

Pandas provides a rich feature-set on the DataFrame. For example, data alignment, data statistics, slicing, grouping, merging, concatenating data, etc.

**Data Structures in Pandas module**

There are 3 data structures provided by the Pandas module, which are as follows:

**Series**: It is a 1-D size-immutable array like structure having homogeneous data.

**DataFrames**: It is a 2-D size-mutable tabular structure with heterogeneously typed columns.

**Panel**: It is a 3-D, size-mutable array.

DataFrame is the most important and widely used data structure and is a standard way to store data.

**Pandas Dataframe**

DataFrame has data aligned in rows and columns like the SQL table or a spreadsheet database.

We can either hard code data into a DataFrame or import a CSV file, tsv file, Excel file, SQL table, etc.We can use the below constructor for creating a DataFrame object.

**pandas.DataFrame(data, index, columns, dtype, copy)**

Below is a short description of the parameters:

**data** – create a DataFrame object from the input data. It can be list, dict, series, Numpy ndarrays or even, any other DataFrame.

**index** – has the row labels

**columns** – used to create column labels

**dtype** – used to specify the data type of each column, optional parameter

**copy** – used for copying data, if any

There are many ways to create a DataFrame. We can create DataFrame object from Dictionaries or list of dictionaries. We can also create it from a list of tuples, CSV, Excel file, etc.

**4.2.2 Some Pandas functions:**

**Importing Data**

Use these commands to import data from a variety of different sources and formats.

pd.read\_csv(filename) | From a CSV file

pd.read\_table(filename) | From a delimited text file (like TSV)

pd.read\_excel(filename) | From an Excel file

pd.read\_sql(query, connection\_object) | Read from a SQL table/database

pd.read\_json(json\_string) | Read from a JSON formatted string, URL or file.

pd.read\_html(url) | Parses an html URL, string or file and extracts tables to a list of

dataframes

pd.read\_clipboard() | Takes the contents of your clipboard and passes it to read\_table()

pd.DataFrame(dict) | From a dict, keys for columns names, values for data as lists

**Exporting Data**

Use these commands to export a DataFrame to CSV, .xlsx, SQL, or JSON.

df.to\_csv(filename) | Write to a CSV file

df.to\_excel(filename) | Write to an Excel file

df.to\_sql(table\_name, connection\_object) | Write to a SQL table

df.to\_json(filename) | Write to a file in JSON format

**Data Cleaning**

Use these commands to perform a variety of data cleaning tasks.

df.columns = ['a','b','c'] | Rename columns

pd.isnull() | Checks for null Values, Returns Boolean Arrray

pd.notnull() | Opposite of pd.isnull()

df.dropna() | Drop all rows that contain null values

df.dropna(axis=1) | Drop all columns that contain null values

df.dropna(axis=1,thresh=n) | Drop all rows have have less than n non null values

df.fillna(x) | Replace all null values with x

s.fillna(s.mean()) | Replace all null values with the mean (mean can be replaced with

almost any function from the statistics module)

s.astype(float) | Convert the datatype of the series to float

s.replace(1,'one') | Replace all values equal to 1 with 'one'

s.replace([1,3],['one','three']) | Replace all 1 with 'one' and 3 with 'three'

df.rename(columns=lambda x: x + 1) | Mass renaming of columns

df.rename(columns={'old\_name': 'new\_ name'}) | Selective renaming

df.set\_index('column\_one') | Change the index

df.rename(index=lambda x: x + 1) | Mass renaming of index

**4.2.2 Numpy:**

NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes.

NumPy is a module for Python. The name is an acronym for "Numeric Python" or "Numerical Python". It is an extension module for Python, mostly written in C. This makes sure that the precompiled mathematical and numerical functions and functionalities of Numpy guarantee great execution speed.

Furthermore, NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices. These data structures guarantee efficient calculations with matrices and arrays. The implementation is even aiming at huge matrices and arrays, better known under the heading of "big data". Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy needs Numpy, as it is based on the data structures of Numpy and furthermore its basic creation and manipulation functions. It extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier-transformation and many others.

Both NumPy and SciPy are not part of a basic Python installation. They have to be installed after the Python installation. NumPy has to be installed before installing SciPy.

NumPy is based on two earlier Python modules dealing with arrays. One of these is Numeric. Numeric is like NumPy a Python module for high-performance, numeric computing, but it is obsolete nowadays. Another predecessor of NumPy is Numarray, which is a complete rewrite of Numeric but is deprecated as well. NumPy is a merger of those two, i.e. it is built on the code of Numeric and the features of Numarray.

NumPy’s array class is called ndarray. It is also known by the alias array. Note that numpy.array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an ndarray object are:

**ndarray.ndim**

The number of axes (dimensions) of the array.

**ndarray.shape**

The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, the shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

**ndarray.size**

The total number of elements of the array. This is equal to the product of the elements of shape.

**ndarray.dtype**

An object describing the type of the elements in the array. One can create or specify dtype using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

**ndarray.itemsize**

The size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

**ndarray.data**

The buffer contains the actual elements of the array. Normally, we won’t need to use this attribute because we will access the elements in an array using indexing facilities.

**4.2.3 Random Module:**

The random module has various functions to accomplish all of the above tasks. We will see how to use these functions in the latter section of the article.

You need to import the random module in your program, and you are ready to use this module. Use the following statement to import the random module in your code.

**import random**

random() is the most basic function of the random module.

Almost all functions of the random module depend on the basic function random().

random() return the next random floating-point number in the range [0.0, 1.0).

To generate random integers, we can use the following functions.

**random.randint(a, b)**

Return a random integer Number such that a <= Number <= b.

A randint(a,b) works only for integers.

The randint(a,b) function accepts two parameters, and both are required.

The resultant random number is greater than or equal to a and less than or equal to b.

**random.randrange(start, stop [, step])**

Use this method to generate a random integer number within a given range. For Example, generate a random number between 10 to 50.

The step is a difference between each number in the sequence. The step is optional, and the default value of the step is 1.

**random.choice(seq)**

Use the random.choice method to pick a random element from the sequence. Here sequence can be list or string. This method returns a single item from the sequence.

**random.sample(population, k)**

Use this method when we want to pick more multiple random elements from a population.

Sample method returns a list of unique elements chosen from the population. Count of the total elements depends on the size of k.

The population can be the list, set or any sequence.

**random.choices()**

random.choices(population, weights=None, \*, cum\_weights=None, k=1)

When you want to choose more than one element from the sequence randomly, then use this method. Choices method introduced in python version 3.6 and it can repeat the elements.

**random.seed(a=None, version=2)**

The seed method is used to initialize the pseudorandom number generator in Python.

The random module uses the seed value as a base to generate a random number. If seed value is not present, it takes a system current time.

### random.uniform(start, end)

1.Use random.uniform() to **generate a floating point number within a given range**.

2.The end-point value may or may not be included in the range depending on floating-point rounding.

3.For example, Generate random float number between 10.5 to 25.5.

**4.2.4 Matplotlib Module:**

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

## [Installing an official release](https://matplotlib.org/users/installing.html#id9) :

Matplotlib and its dependencies are available as wheel packages for macOS, Windows and Linux distributions:

python -m pip install -U pip

python -m pip install -U matplotlib

**Importing matplotlib :**

from matplotlib import pyplot as plt

*or*

import matplotlib.pyplot as plt

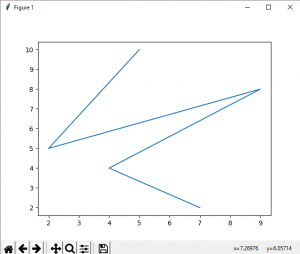
#### Basic plots in Matplotlib :

Matplotlib comes with a wide variety of plots. Plots helps to understand trends, patterns, and to make correlations. They’re typically instruments for reasoning about quantitative information. Some of the sample plots are covered here.

**Line plot :**

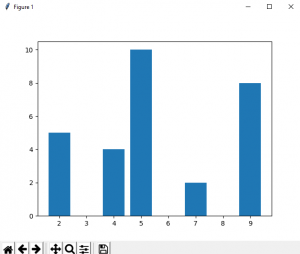
|  |
| --- |
| # importing matplotlib module  from matplotlib import pyplot as plt    # x-axis values  x = [5, 2, 9, 4, 7]    # Y-axis values  y = [10, 5, 8, 4, 2]    # Function to plot  plt.plot(x,y)    # function to show the plot  plt.show() |

**Output :**

  
   
**Bar plot :**

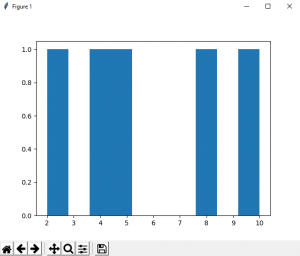
|  |
| --- |
| # importing matplotlib module  from matplotlib import pyplot as plt    # x-axis values  x = [5, 2, 9, 4, 7]    # Y-axis values  y = [10, 5, 8, 4, 2]    # Function to plot the bar  plt.bar(x,y)    # function to show the plot  plt.show() |

Output :

  
   
**Histogram :**

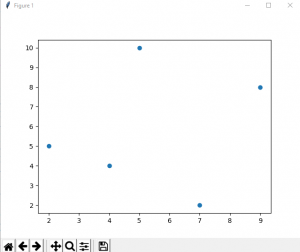
|  |
| --- |
| # importing matplotlib module  from matplotlib import pyplot as plt    # Y-axis values  y = [10, 5, 8, 4, 2]    # Function to plot histogram  plt.hist(y)    # Function to show the plot  plt.show() |

Output :



**Scatter Plot :**

|  |
| --- |
| # importing matplotlib module  from matplotlib import pyplot as plt    # x-axis values  x = [5, 2, 9, 4, 7]    # Y-axis values  y = [10, 5, 8, 4, 2]    # Function to plot scatter  plt.scatter(x, y)    # function to show the plot  plt.show() |

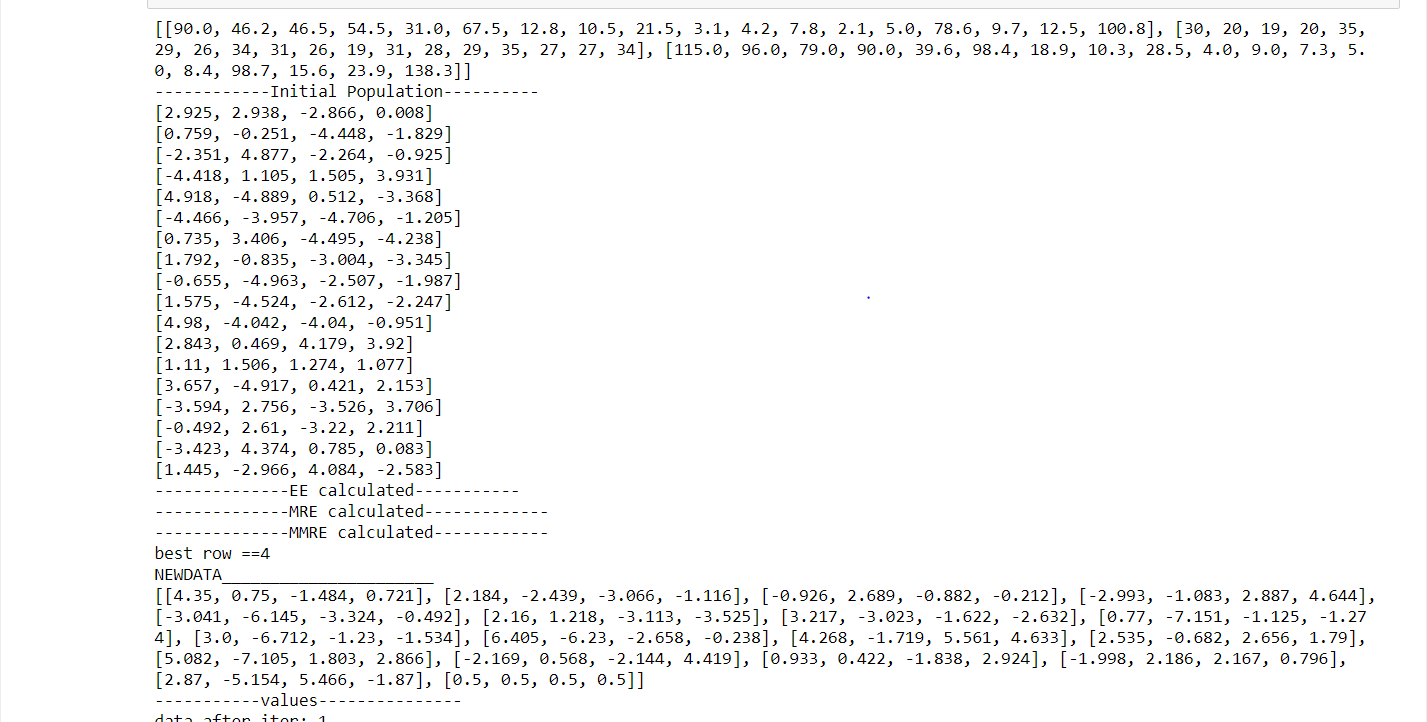
Output : 

CHAPTER 5

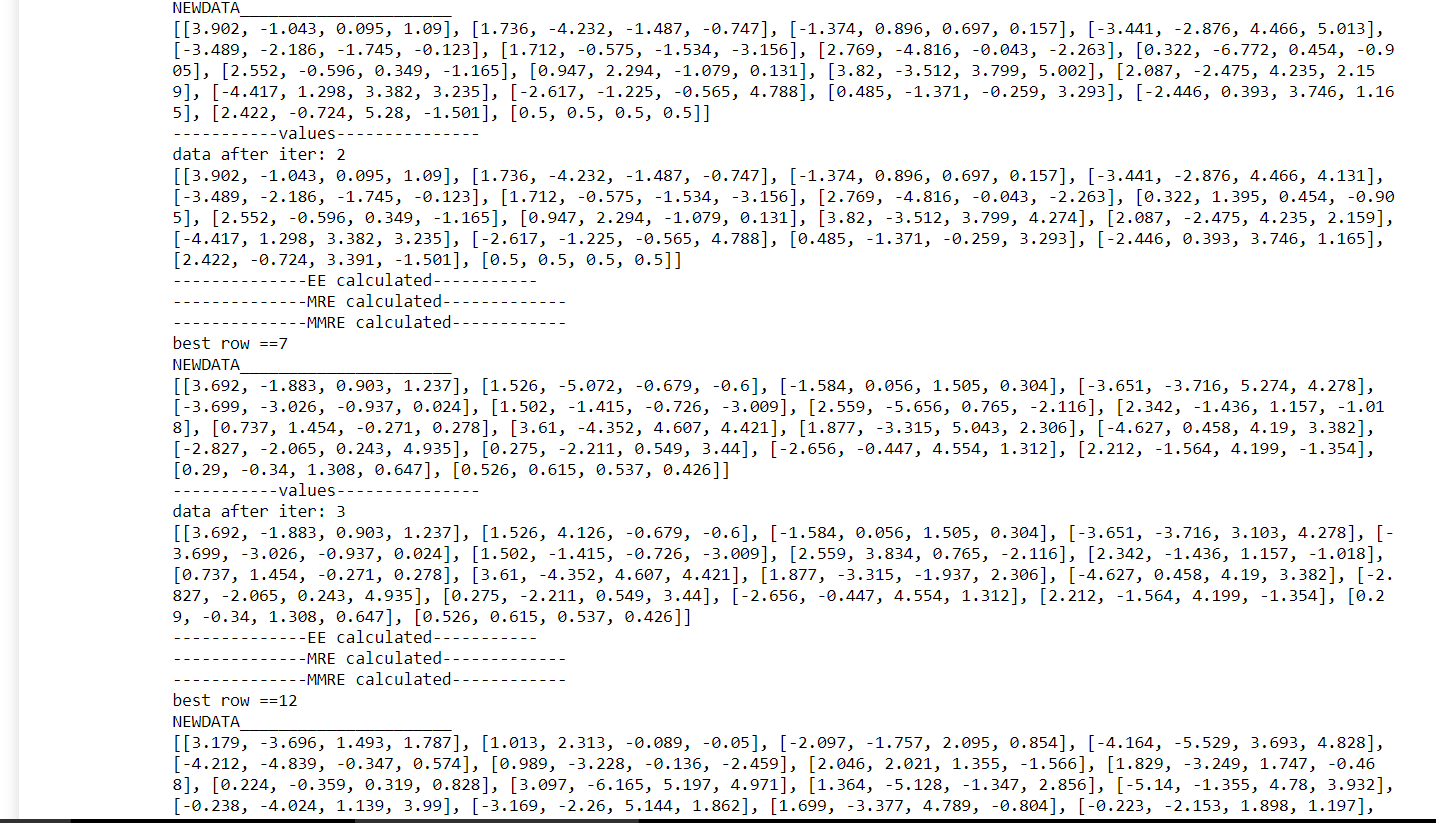
Program Results

**5.1 Program Results :**

The output is in the following format



Output-1



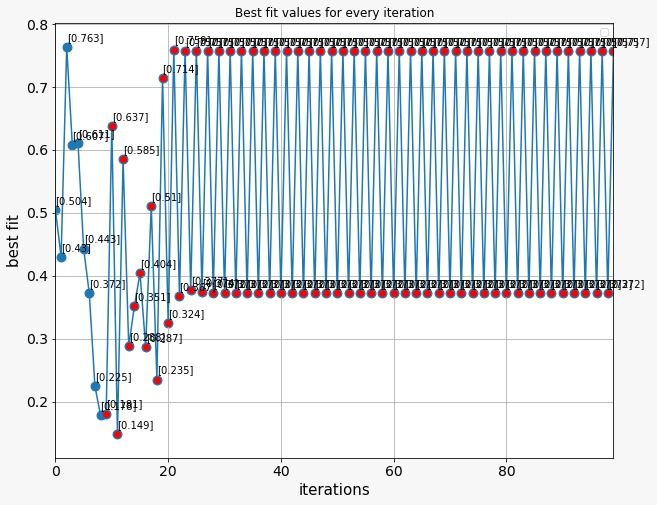
Output-2

The above data is displayed for 100 iterations in the same manner.

We are following a hit or trail method so the best values we obtained till now is

Best MMRE=0.149

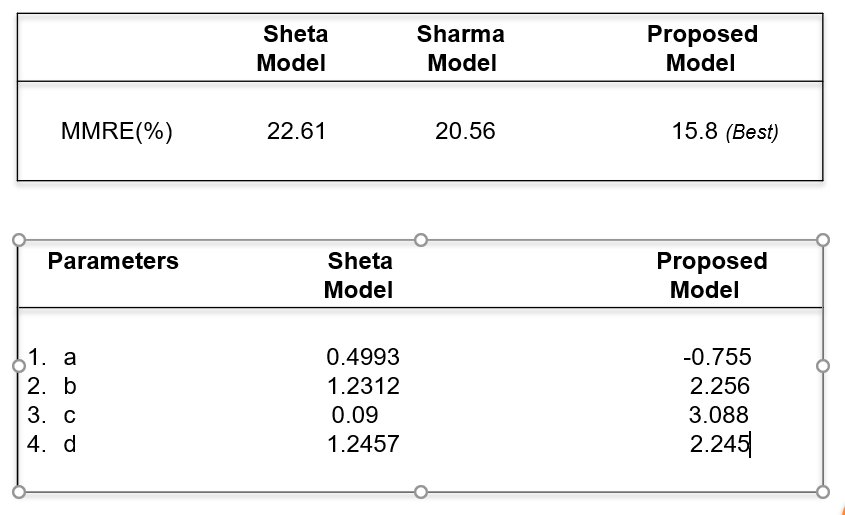
Best data=[-0.755, 2.256, 3.088, 2.245]



Output-Graph

**Comparision with existing models:**

The results produced by proposed model have shown its superiority compared to other existing techniques. This is because of the usage of IEAM-RP which has high convergence speed and diversity preservation capabilities of the solutions during the optimization process. This algorithm is able to minimize the diﬀerence between measured eﬀort and estimated eﬀort. The best result of MMRE is produced by the proposed model for all three models



Comparison chart

CHAPTER 6

CONCLUSION

**6 .CONCLUSION**

In this project, IEAM-RP is used for tuning the parameters of Sheta model of software cost estimation. The results produced by IEAM-RP have shown its superiority compared to other existing techniques. The reason behind the competitive performance of IEAM-RP is its convergence speed and diversity preservation capabilities of the solutions during the optimization process. This algorithm is able to minimize the diﬀerence between measured eﬀort and estimated eﬀort. The best result of MMRE is produced by IEAM-RP for all three models. The best prediction was given by EAMDGA, IEAM-RP, and EAMDGA for model 1, 2, and 3, respectively. Extensive experimental results and analysis demonstrated the eﬀectiveness of the proposed technique.

CHAPTER 7

REFERENCES

**7.References:**

[1]Software Cost Estimation Using Environmental Adaptation Method by Tribhuvan Singha,∗, Ranvijay Singha, Krishn Kumar Mishraa in 8th International Conference on Advances in Computing and Communication (ICACC-2018

[2]S.AljahdaliandA.F.Sheta. Software eﬀor testimation by tuning coocmo model parameters using diﬀerential evolution . In Computer Systems and Applications (AICCSA), 2010

IEEE/ACS International Conference on, pages 1–6. IEEE, 2010.

[3] J.W.BaileyandV.R.Basili. Ameta-model for software development resource expenditures. In Proceedings of the 5th international conference on Software engineering, pages 107–116. IEEE Press, 1981.

[4] J. M. Baldwin. A new factor in evolution (continued).

The American Naturalist, 30(355):536–553, 1896.

[5] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby. Cost models for future software life cycle processes: Cocomo 2.0. Annals of software engineering, 1(1):57–94, 1995.

[6] B. W. Boehm et al. Software engineering economics, volume 197. Prentice-hall Englewood Cliﬀs (NJ), 1981.

[7] J. J. Dolado and L. Fernandez. Genetic programming, neural networks and linear regression in software project estimation. In Procs. of International Conference on Software Process Improvement, Research, Education and Training, pages 157–171, 1998.

[8] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion mmre. IEEE Transactions on Software Engineering, 29(11):985–995, 2003.