Benchmarking Suite

Mentor : **Mr. Rajesh Kedia**

Submitted By :
Pranav Bhagat
Sachin Kumar Prajapati
Jay Kumar Modi

Project Report

GOALS

1. To Benchmark some Standard codes implemented in C++ and determine the variation in runtime and accuracy by varying appropriate parameters.
2. To study the performance of the code in ARM processor (Raspberry pi).

- **Prepration of SD card for Raspberrypi**

We used **RASPBIAN STRETCH** version of Raspbian Operating System on RaspberryPi. Apart from that for we installed the g++ compiler and OpenCV library (Version 3.3.0) which was required for some programs to run. Also, installation of OpenCV on Raspberrypi
requires some more space(approx 4-6 GB free space) so a memory card of approx 8GB is required to completely compile and install OpenCV library on your RaspberryPi.

1) Digit-Classification :-

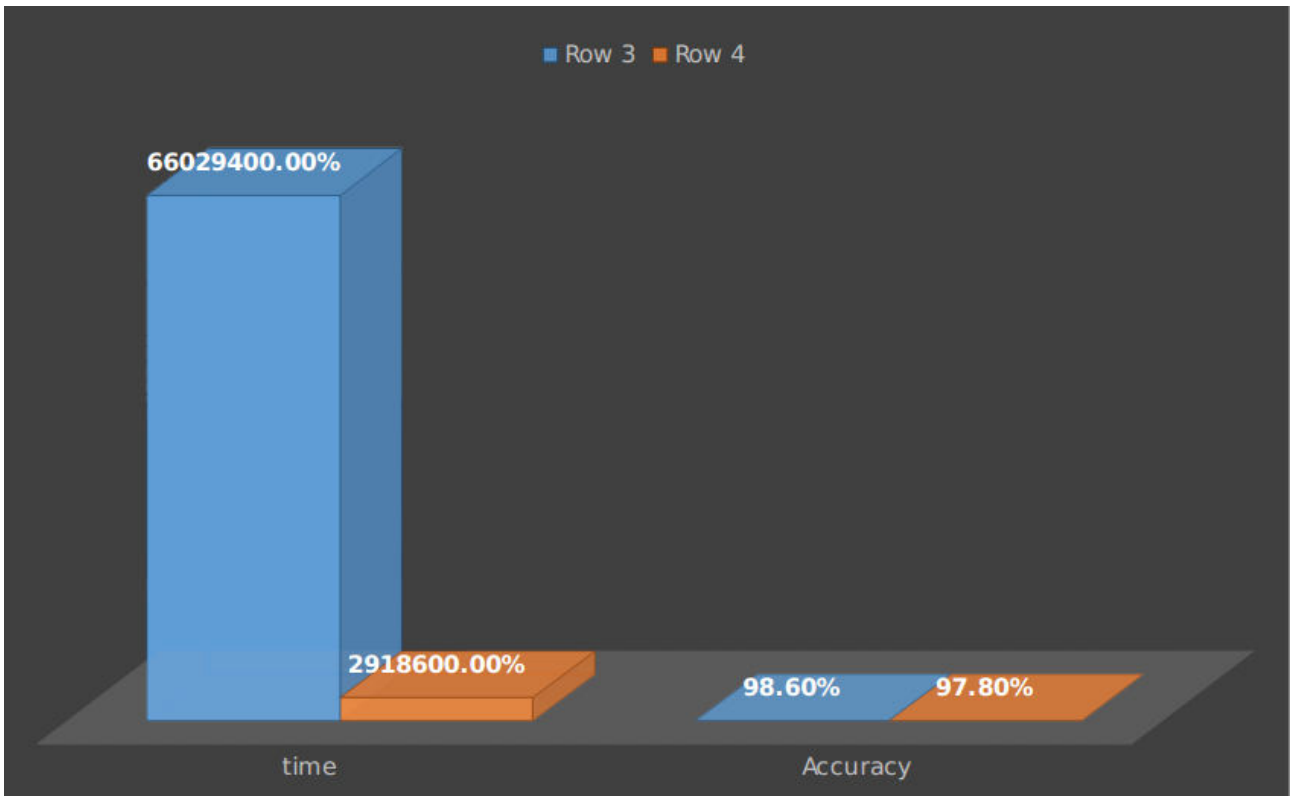
SPECIFICATIONS

This program is implemented in C++ and follows a simple and easy way of digit-classification using OpenCV library. The code and Datasets are taken from the link https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/ .
We changed the code a little bit to take input(about Kernel type and gamma in case of RBF) from the user and also calculate the time taken in testing the result.
Dataset for training and testing is present in the image digit.png in the same folder. It contains 5000 images in all — 500 images of each digit. Each image is 20×20 grayscale with a black background. 4500 of these digits will be used for training and the remaining 500 will be used for testing the performance of the algorithm.
Details are given in https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/ .

In this case, time and accuracy depends upon the factor **kernel_type(i.e RBF or LINEAR)** and **gamma(for non-linear kernel).** And we have studied variation of accuracy and time over these variable and obtained following reason.

RESULTS

a) Kernel type(RBF or LINEAR):
We have studied the variation of kernel type with Runtime and accuracy The following is the results we get:-
i) Variation of Runtime(micro sec) and Accuracy and **Kernel-type**(RBF or LINEAR):-

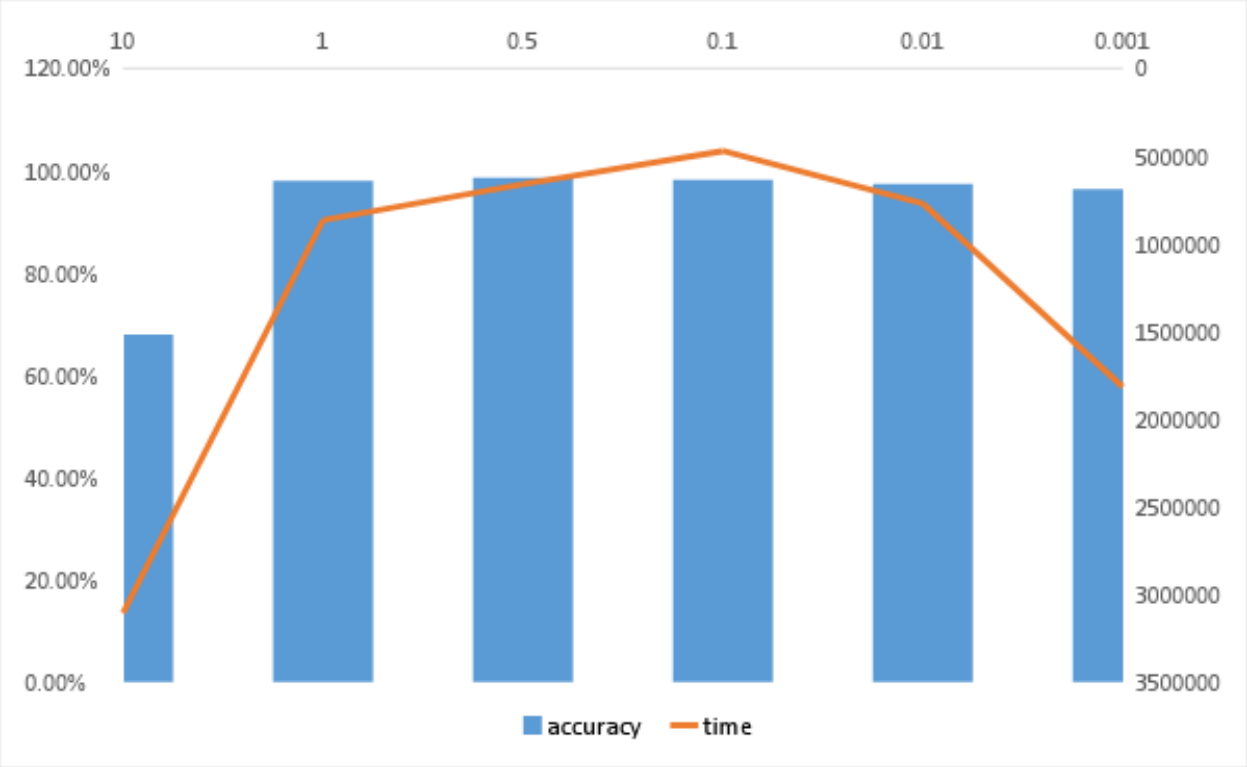


Row3- kernel type-> RBF
Row4- kernel type-> LINEAR
(time is in micro sec)

b) Gamma(for non-linear kernel type):

We have studied the variation of Runtime and accuracy for datasets wth **gamma** with kernel type set as RBF.
The following is the results we get:-

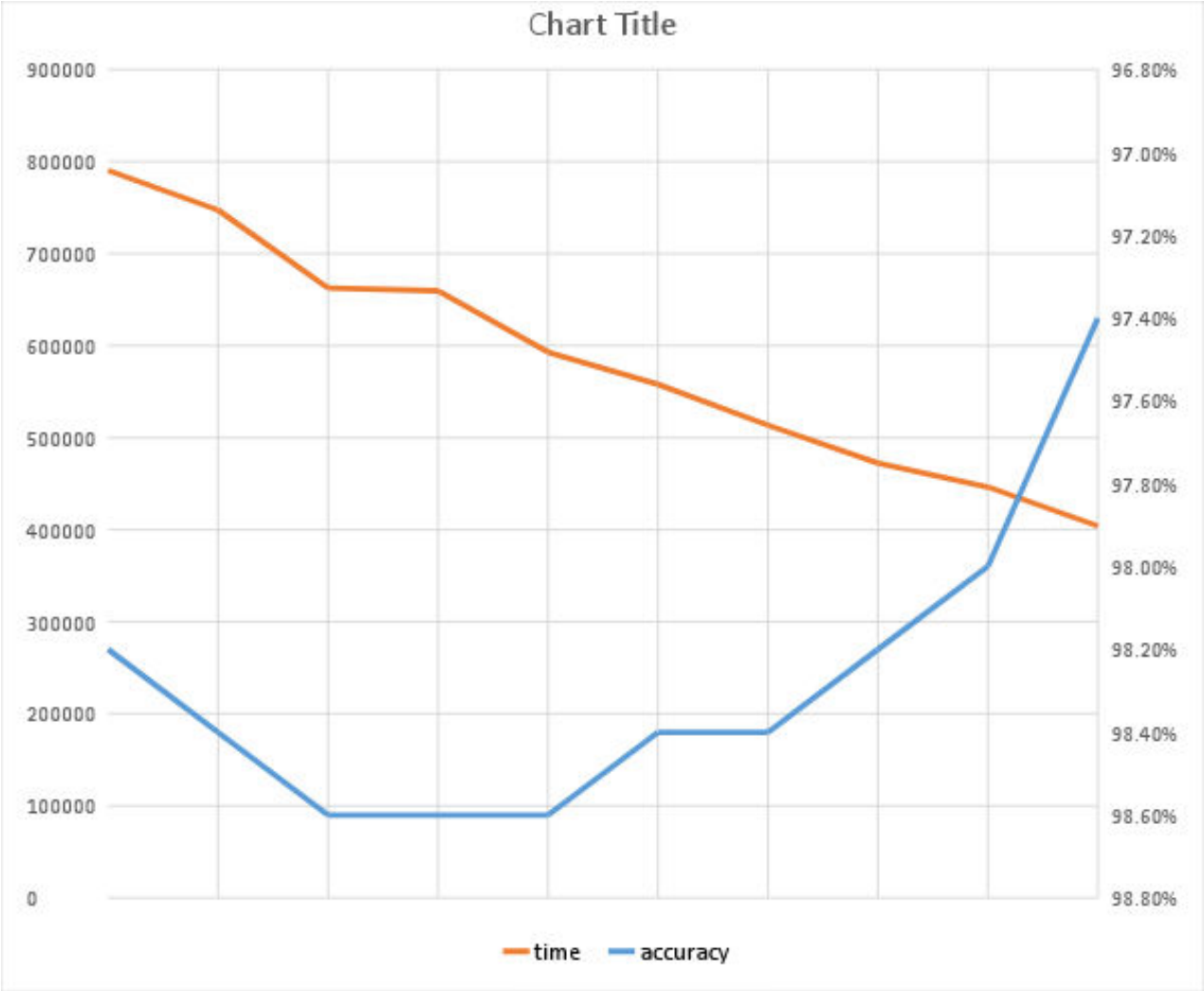
i) Variation of Runtime(ms) and Accuracy and **gamma** (X-axis):-



Data:

gamma	accuracy	time
0.001	96.40%	1815418
0.01	97.40%	769826
0.1	98.20%	471463
0.5	98.60%	660294
1	98%	866259
10	68%	3104844

And in the region near (0.1-1):



Data:

gamma	accuracy	time
0.8	98.20%	790428
0.7	98.40%	747277
0.6	98.60%	662660
0.5	98.60%	659554
0.4	98.60%	592891
0.3	98.40%	557990
0.2	98.40%	513586
0.1	98.20%	472269
0.05	98.00%	446471
0.03	97.40%	404126

CONCLUSION

Accuracy and time both are highly dependent on the Kernel type.
In case of RBF accuracy is high than Linear but time also increase with a significant amount.
Though the change in accuracy is not significant in this case but with diversity in dataset change in accuracy can be significant.

In case of RBF another factor is **gamma.** In a very small region, both accuracy and time both increases with gamma. So, depending upon the need we could choose a gamma(say 0.2) for which accuracy is maximum at only a small cost in time.

2)K-means Clustering :-

SPECIFICATIONS

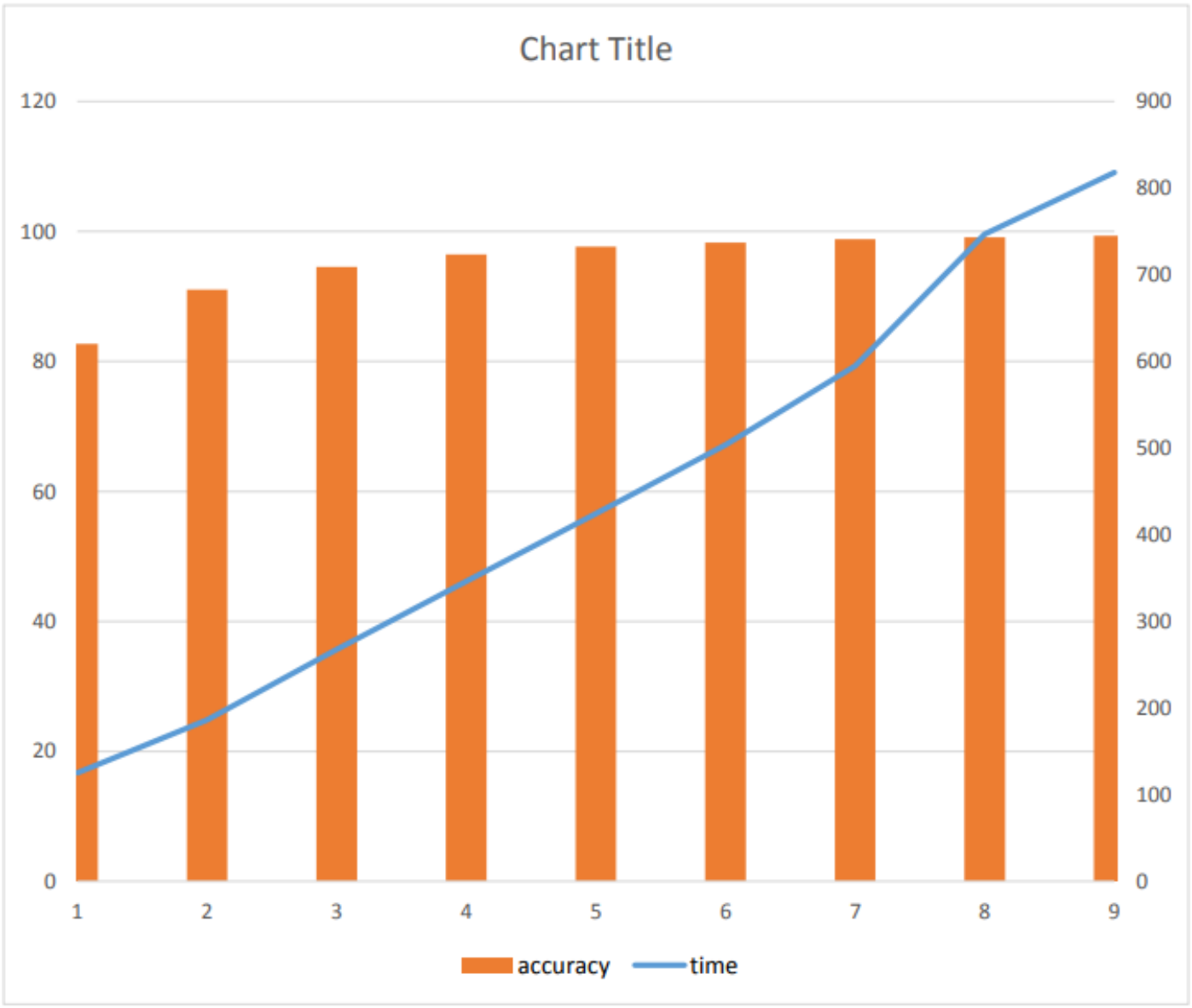
This program is implemented in C++ and follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). We have taken the code from
https://github.com/marcoscastro/kmeans
The actual code classifies the given data points through a number of clusters and print them at last. We made changes in the code to calculate accuracy and time of the code. And printed just the accuracy and time rather than the clusters.
For accuracy measurement, we compare each individual's distance to its own cluster mean and to that of the other cluster.
The euclidean distance was used for to calculate the distance of each data point for the centroid of cluster. The algorithm stops by maximum number of iterations or if no data point exchange cluster.

In this case, time and accuracy depends upon the factor **max_iterations**. But the code randomly generates initial set of clusters. So for measuring time and accuracy we did it for 1000 times and then took the average of both accuracy and time.

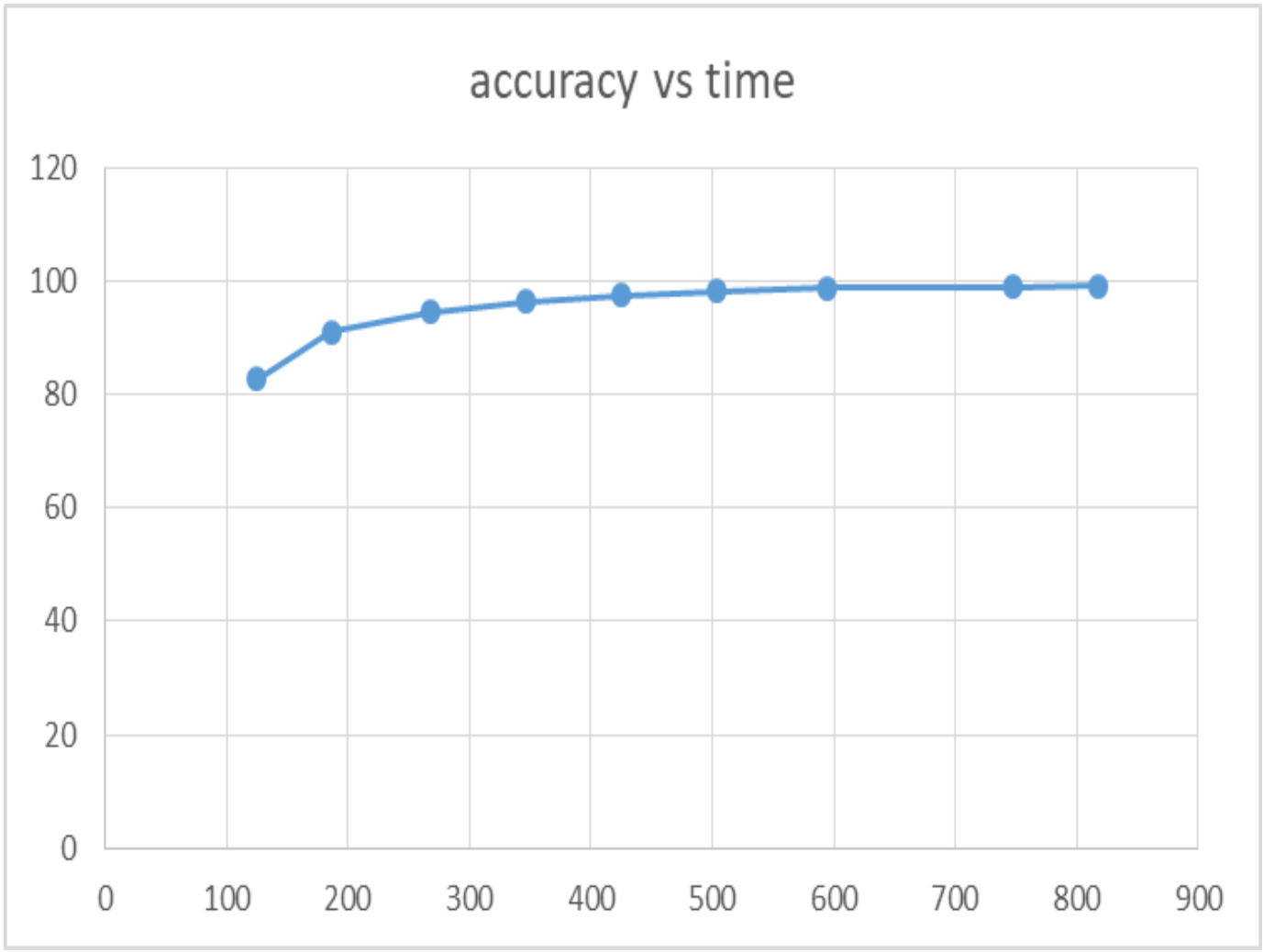
RESULTS

a) Dataset1:
We have studied the variation of **max_iterations** with Runtime and accuracy for Ionosphere datasets given in
https://archive.ics.uci.edu/ml/datasets/ionosphere .
The following is the results we get:-

i) Variation of Runtime(ms) and Accuracy and **max_iterations** (X-axis):-



ii) Variation of Accuracy (Y-axis) and Runtime(in ms)(X-axis):-

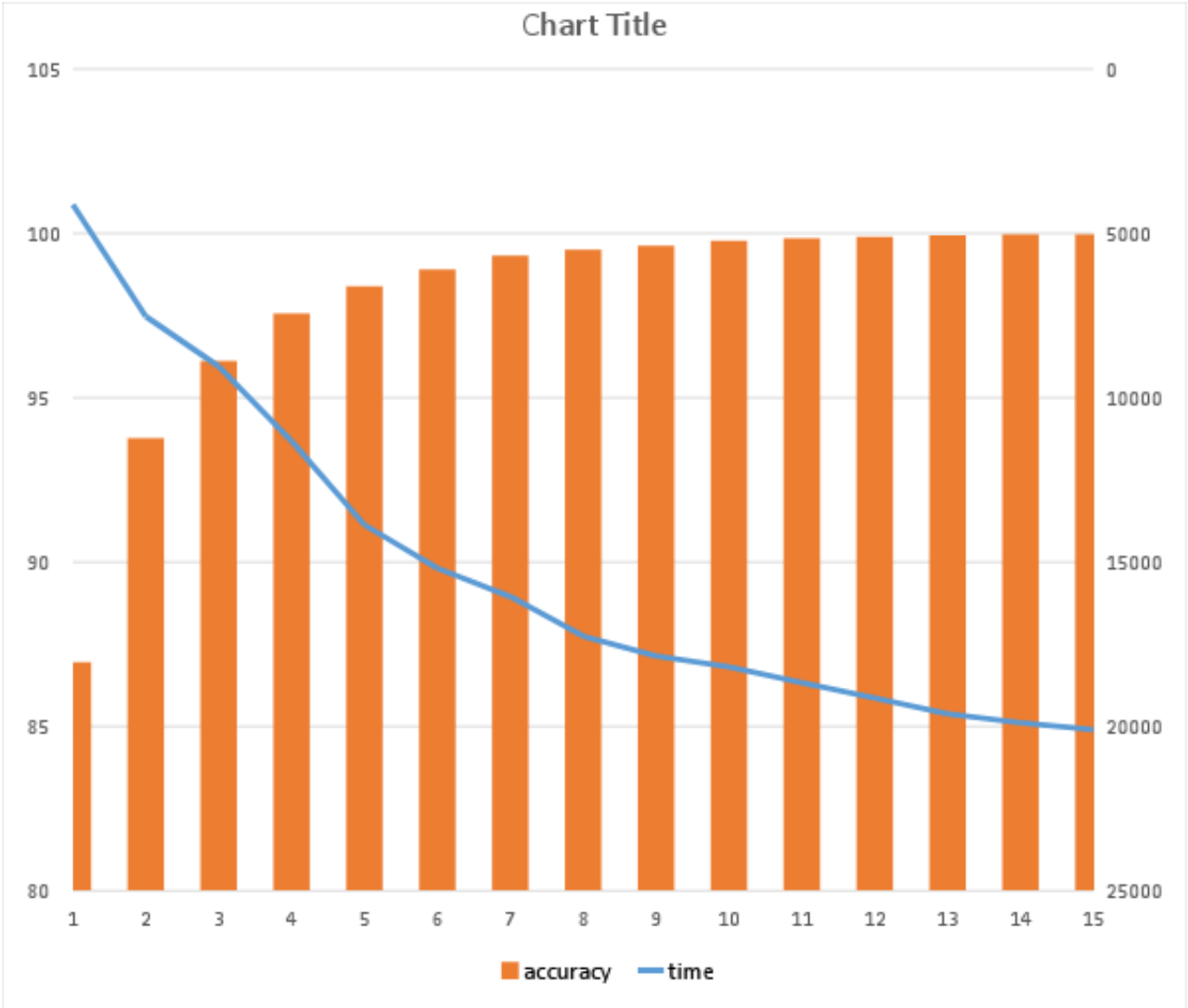


b) Dataset2

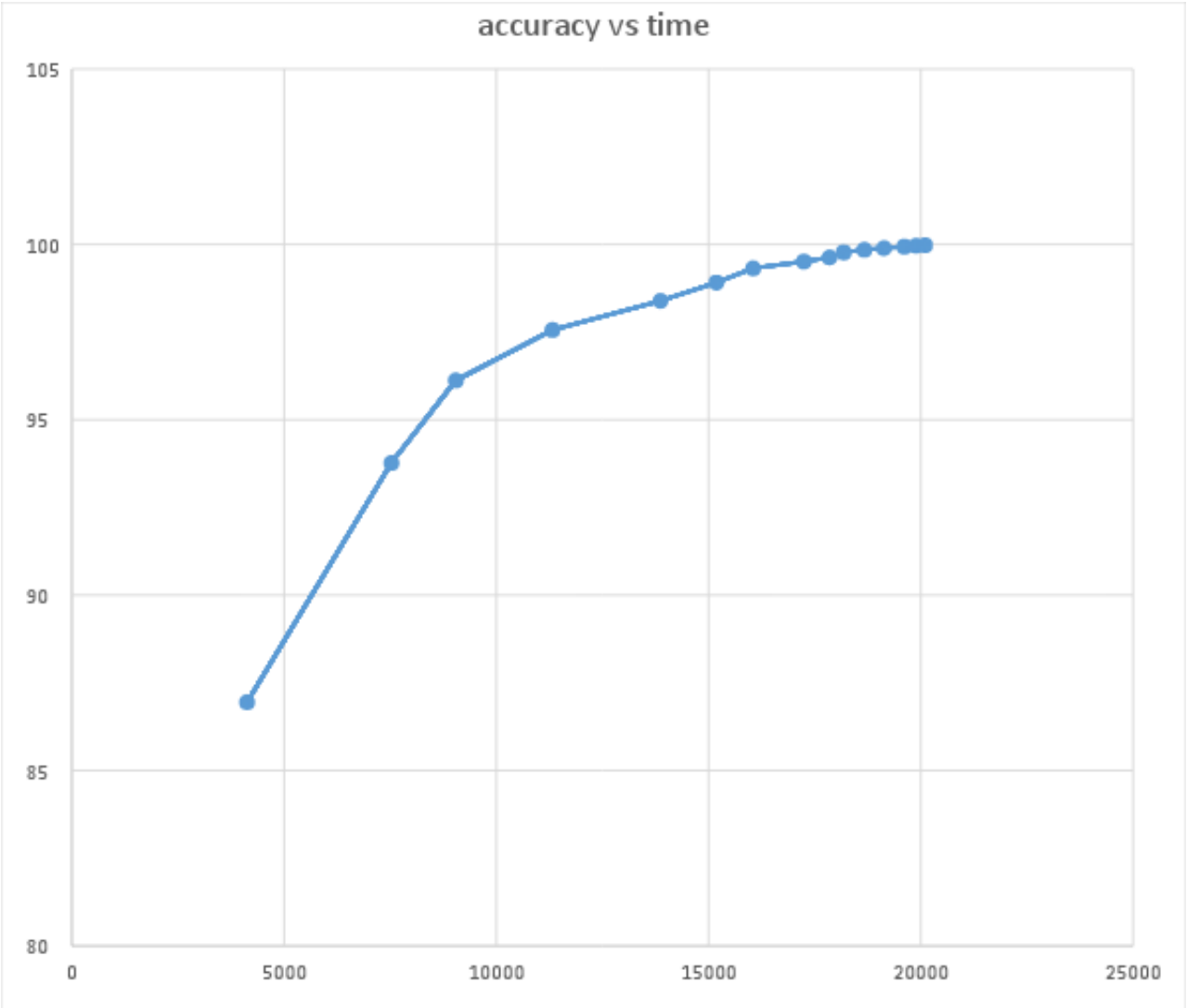
We have studied the variation of **max_iterations** with Runtime and accuracy for datasets given in https://github.com/marcoscastro/kmeans/tree/master/datasets .

The following is the results we get:-

i) Variation of Runtime(ms) and Accuracy and **max_iterations** (X-axis):-



ii) Variation of Accuracy (Y-axis) and Runtime(in ms)(X-axis):-



CONCLUSION

With increase in **max_iteration,** both accuracy and time increase. Initially increment in the accuracy and time is large and as the **max_iteration** increases the increment for both time and accuracy decreases. But in the case of accuracy increment decreases more rapidly than time. So, for a little change in accuracy we have pay a lot more in time.

So, depending upon the dataset there is a point after which accuracy changes very little at the cost of time(5 in 1st case 8 in 2nd case).

3)Pedestrian Detection:-

SPECIFICATIONS

Program is implemented in C++ and detects number of pedestrian for a given input. Input can be given in the format of Video, image or Directory(containing images). The code uses HOG Descriptor method of object detection and is implemented using OpenCV library.

The variation in performance (accuracy and run time) can be observed due to changes in HOG detect multi-scale, namely scaling factor, win-size and padding.

We study the variation of run-time and accuracy with scaling factor while keeping other parameters fixed.

Source for Code: By Steven Puttemans(Git URL: https://github.com/opencv/opencv/blob/master/samples/cpp/peopledetect.cpp)

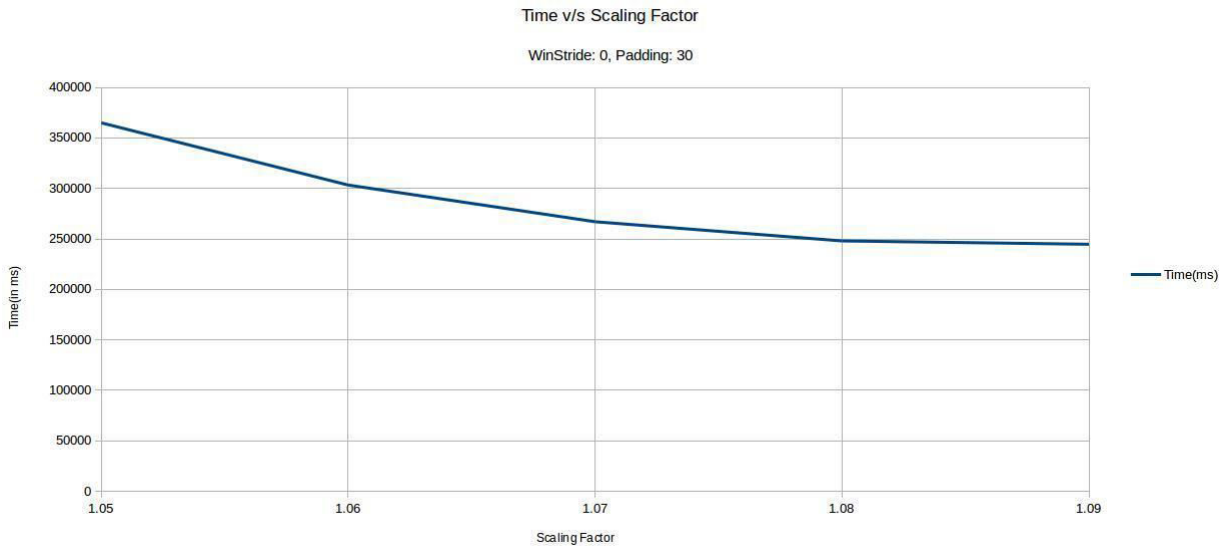
Source for Datasets: Dalmer Pedestrian Benchmark Datasets (http://www.gavrila.net/Datasets/Daimler_Pedestrian_Benchmark_D/daimler_pedestrian_benchmark_d.html)

RESULTS

a) Dataset1:

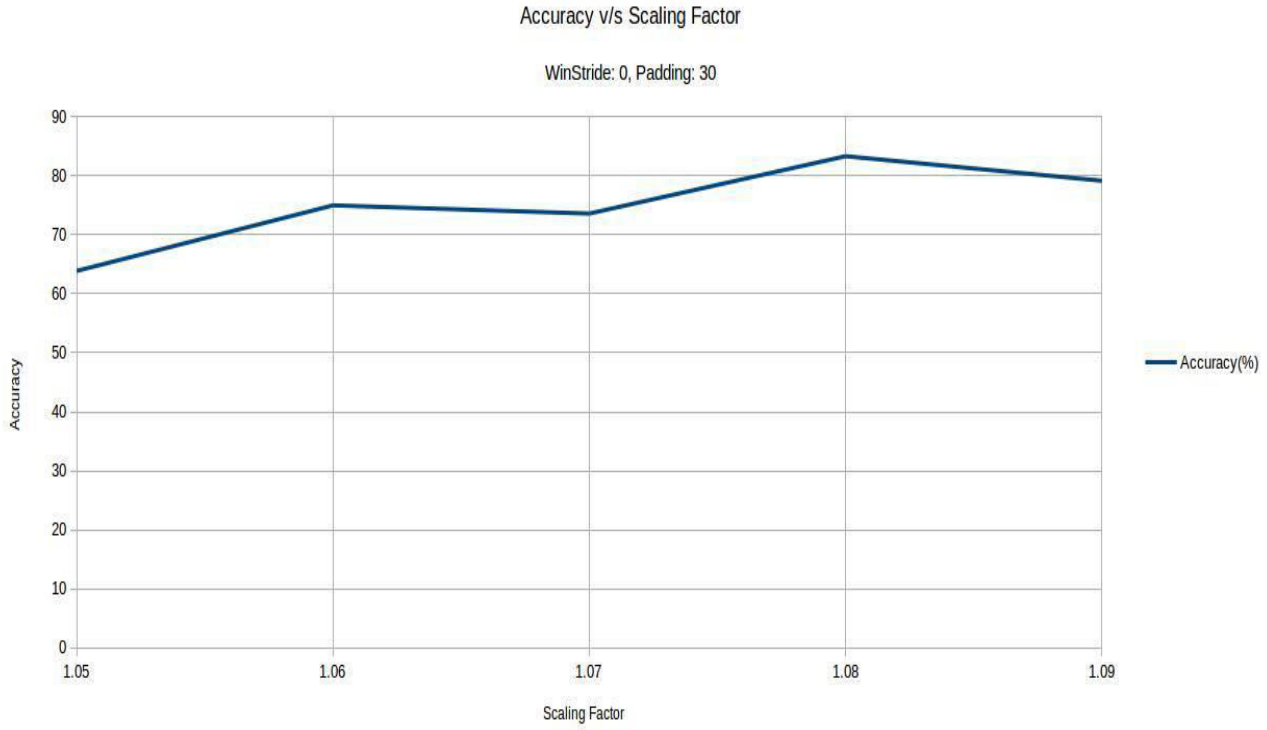
Following are the observed results when the code is ran against Dailmer video datasets.

Time v/s Scaling Factor



Accuracy v/s Time

Accuracy v/s Scaling Factor



4)Series Expansion:-

SPECIFICATIONS

Three programs used implements Sine, Cosine and Exponential series. Written in C++, these programs use the basic mathematical concepts and formulas of series expansion.

The variation studied are in Run time and the preciseness of the value generated under different conditions (for various parameters) with respect to the value generated under more rigorous parameter.

We study the variation of run-time and accuracy with number of terms of series used to produce the result while keeping other parameters fixed.

Source for Code:

Sine: https://www.codingconnect.net/cpp-program-sine-series/

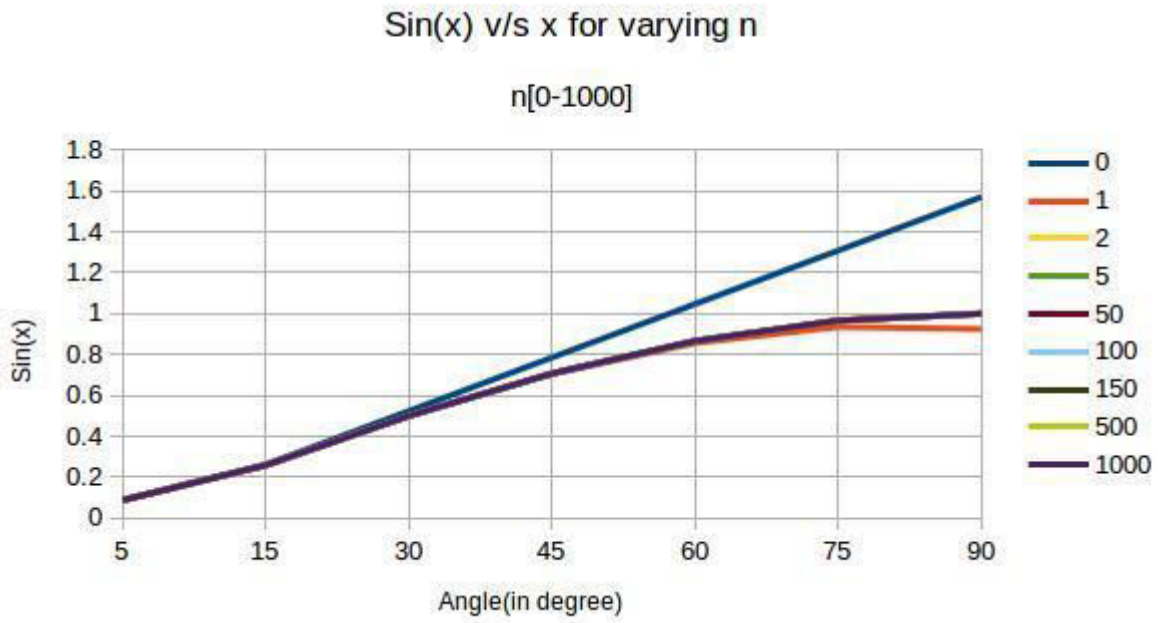
Cosine: https://www.codingconnect.net/cpp-program-cosine-series/

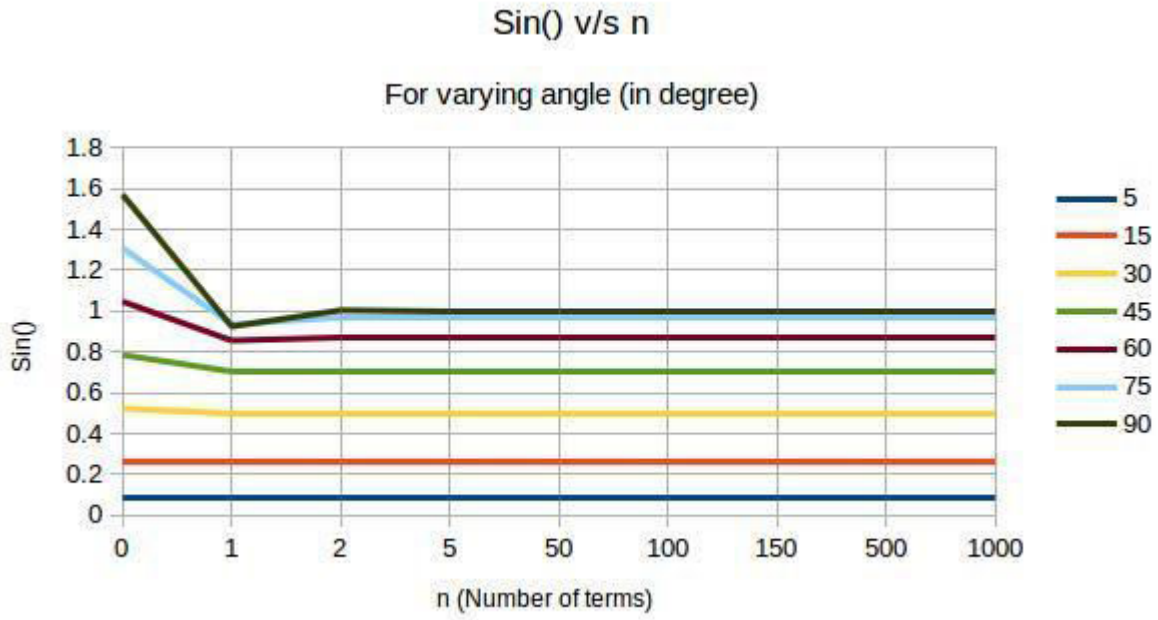
Exponential: https://www.codingconnect.net/cpp-program-exponential-series/

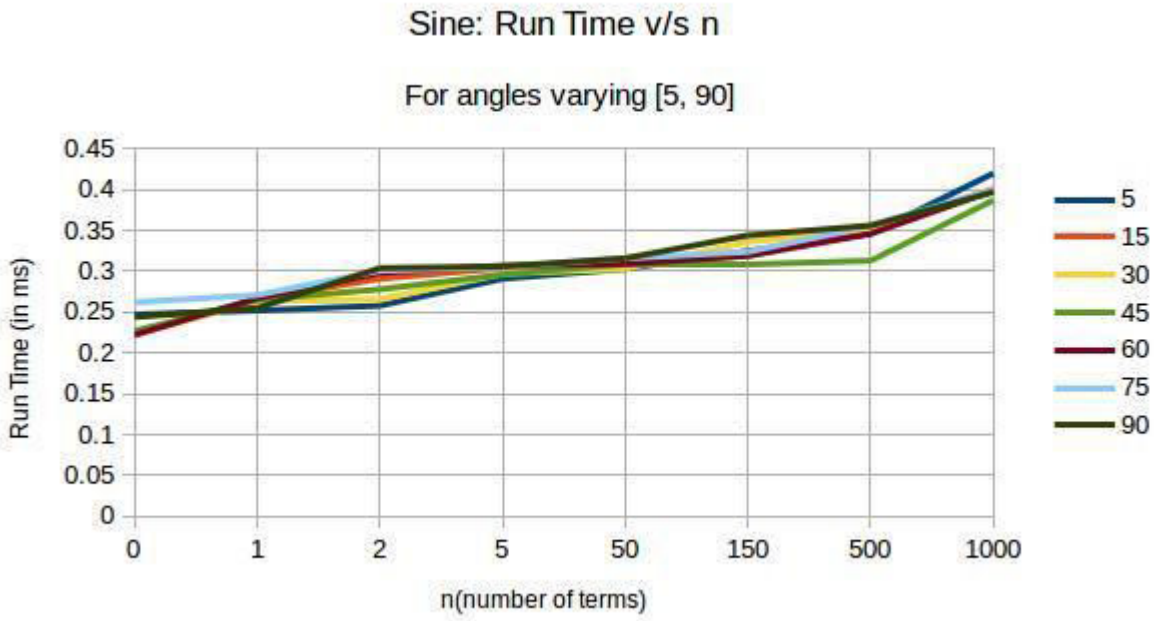
RESULTS

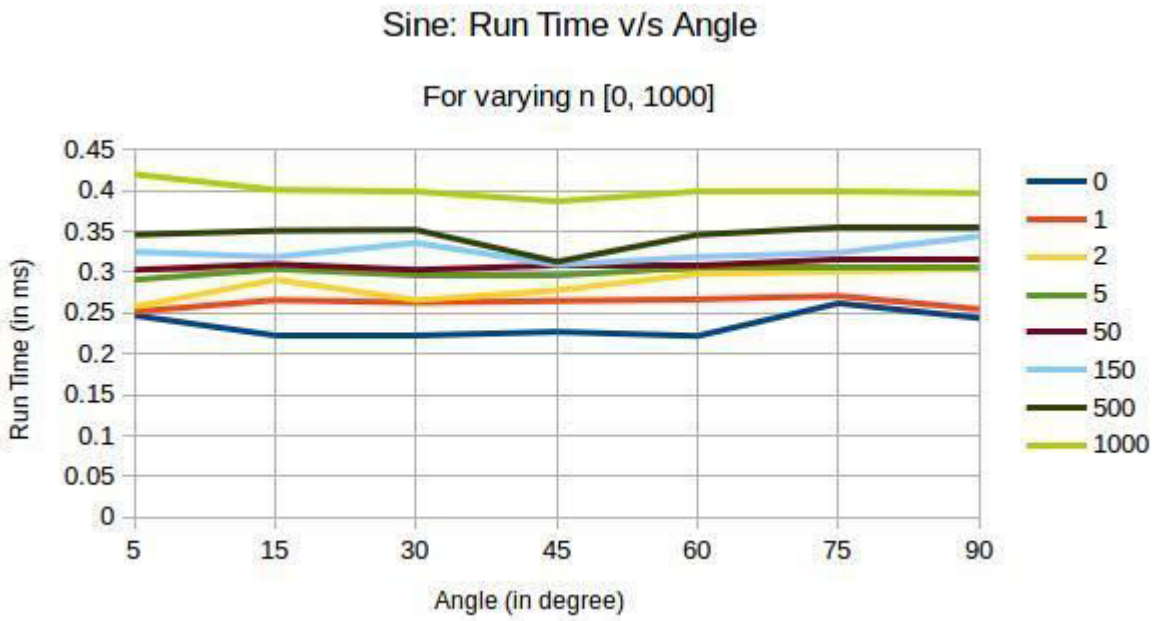
Following are the observed results when the codes are ran against some standard input values.

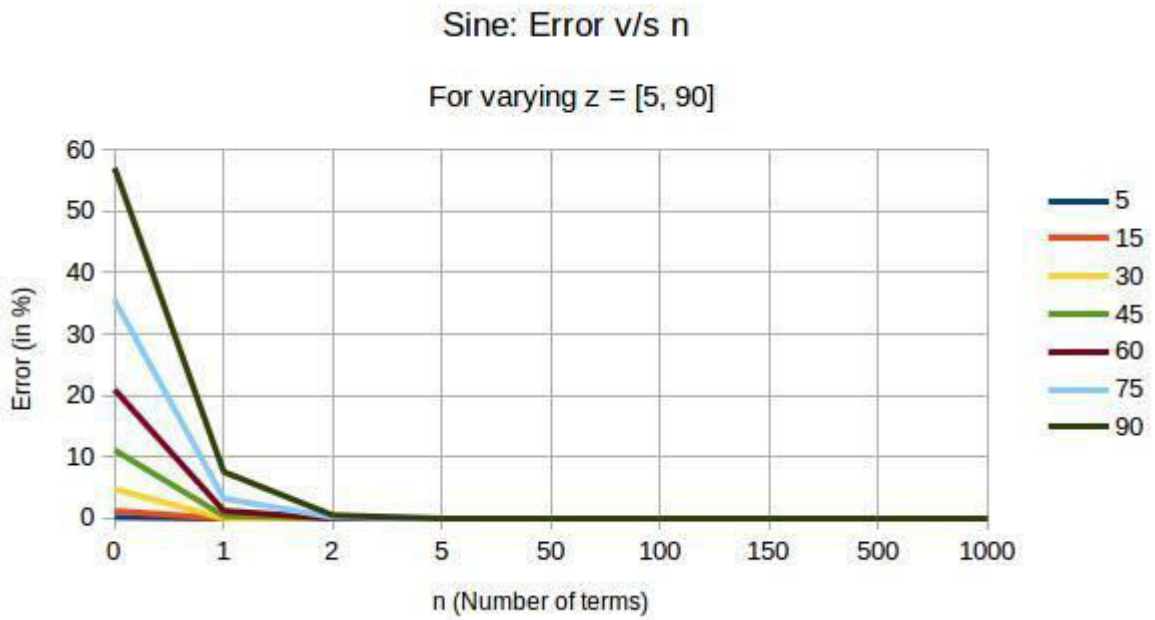
A. Sine Series



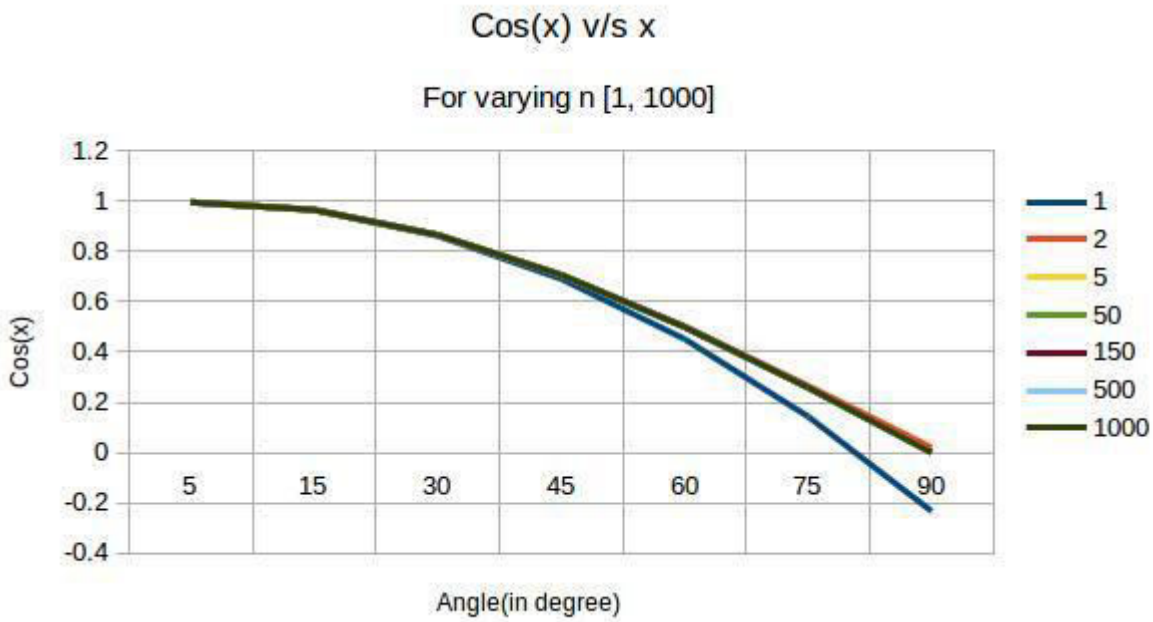


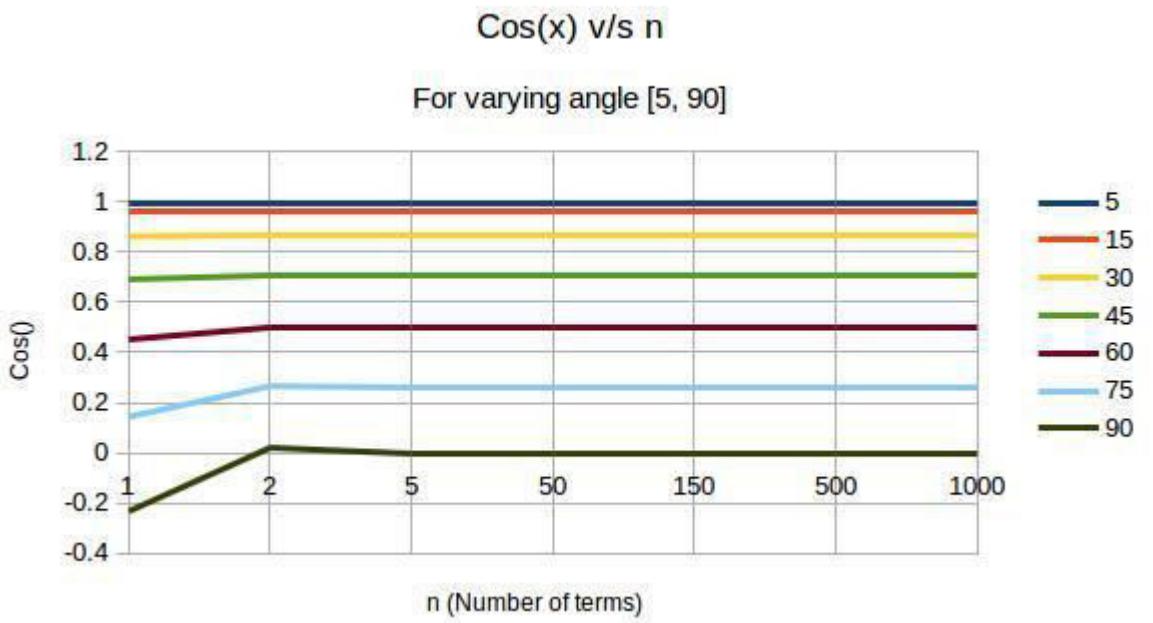

Sine: Run Time v/s n
For angles varying [5, 90]
Run Time (in ms)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
1
2
5
50
150
500
1000
n(number of terms)
5
15
30
45
60
75
90

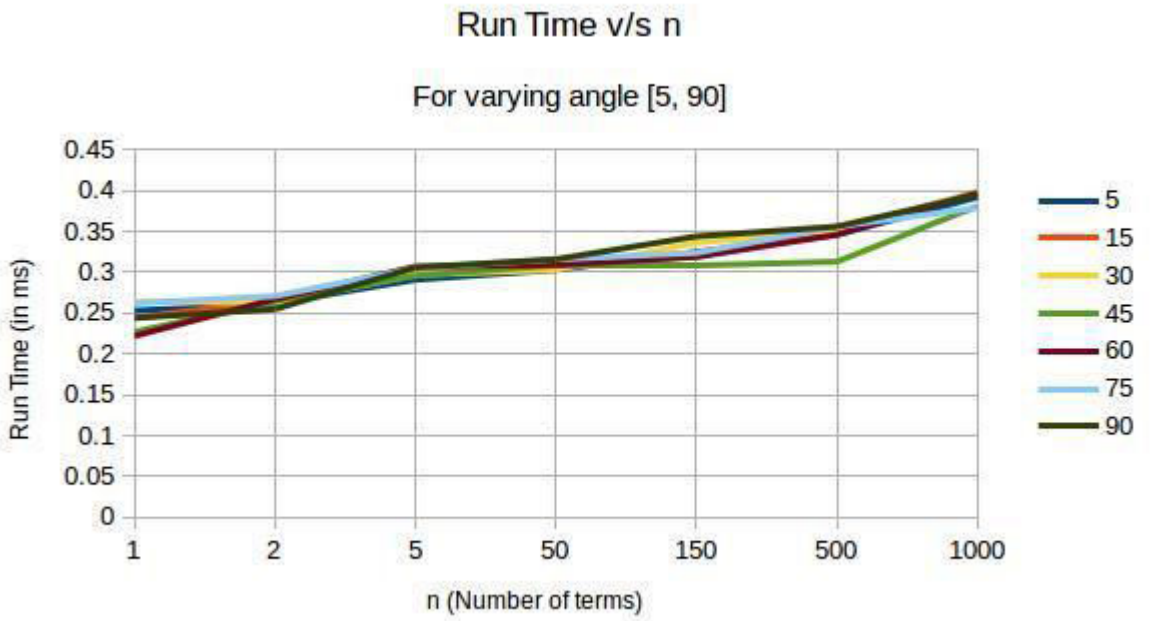

Sine: Run Time v/s Angle
For varying n [0, 1000]
Run Time (in ms)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
5
15
30
45
60
75
90
Angle (in degree)
0
1
2
5
50
150
500
1000

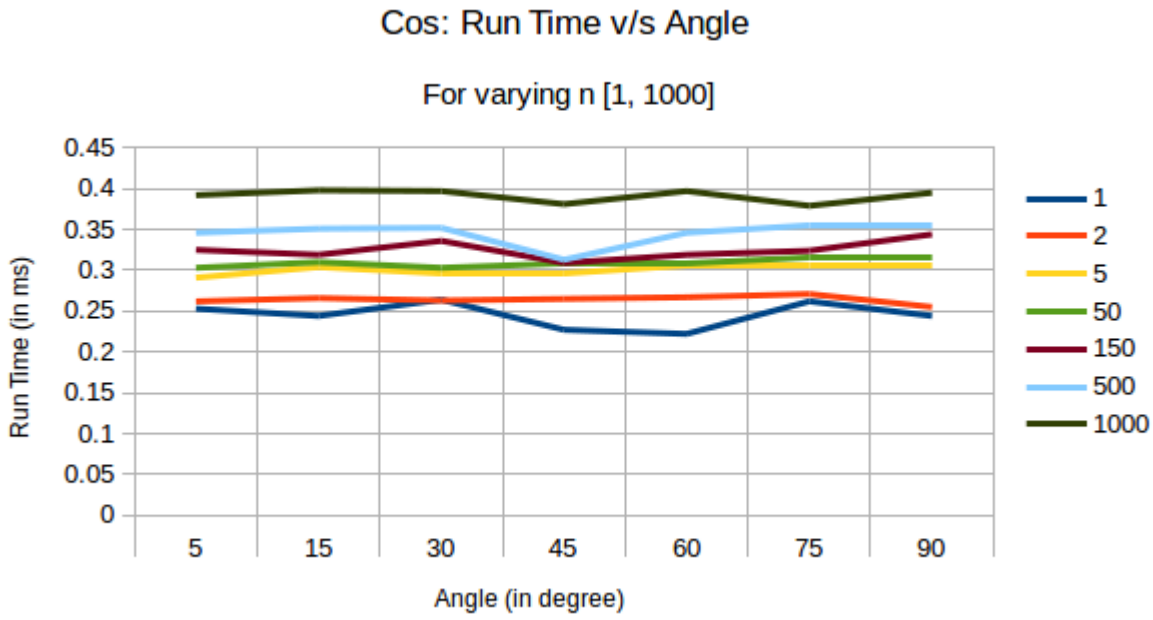

Sine: Error v/s n
For varying z = [5, 90]
Error (in %)
60
50
40
30
20
10
0
0
1
2
5
50
100
150
500
1000
n (Number of terms)
5
15
30
45
60
75
90

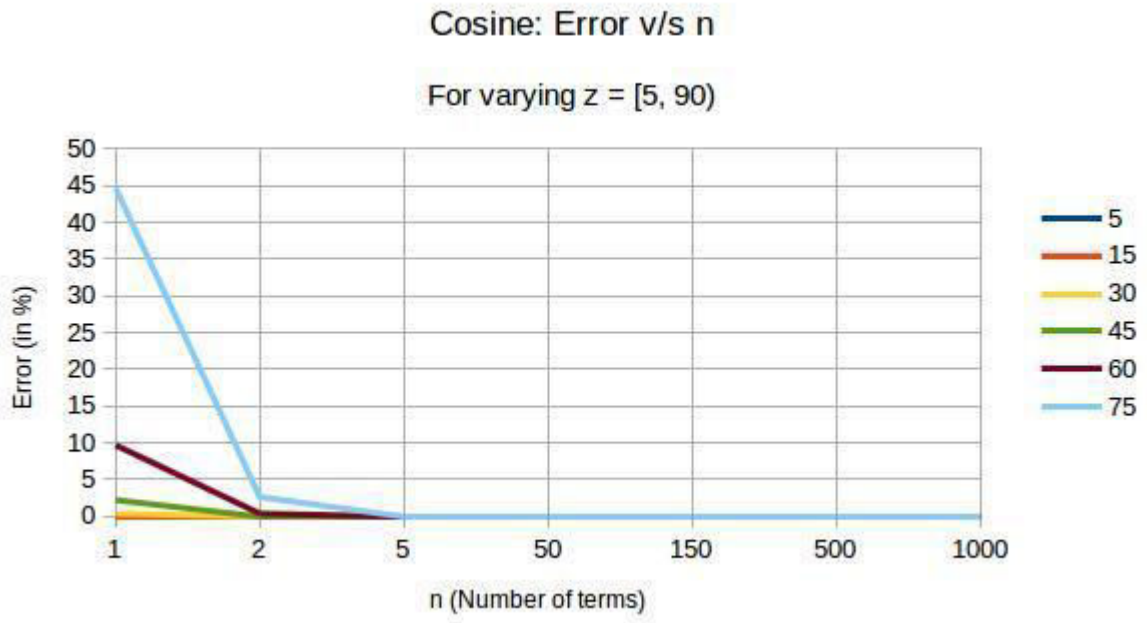
B. Cosine Series

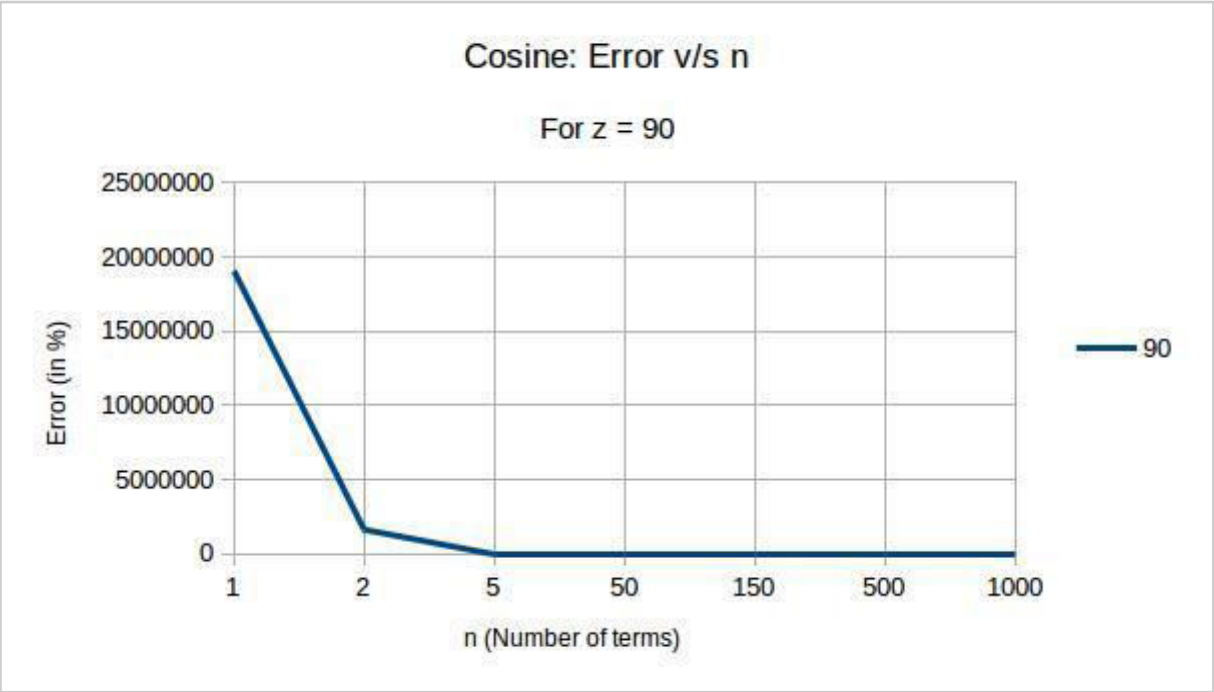




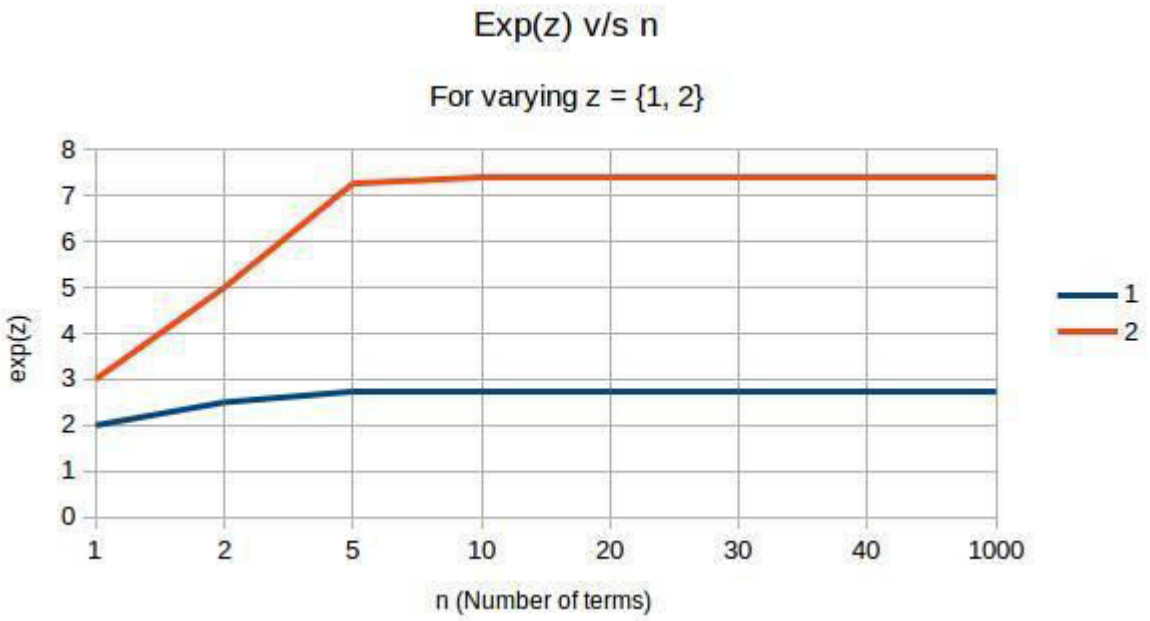

Run Time v/s n
For varying angle [5, 90]
Run Time (in ms)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
5
50
150
500
1000
n (Number of terms)
5
15
30
45
60
75
90

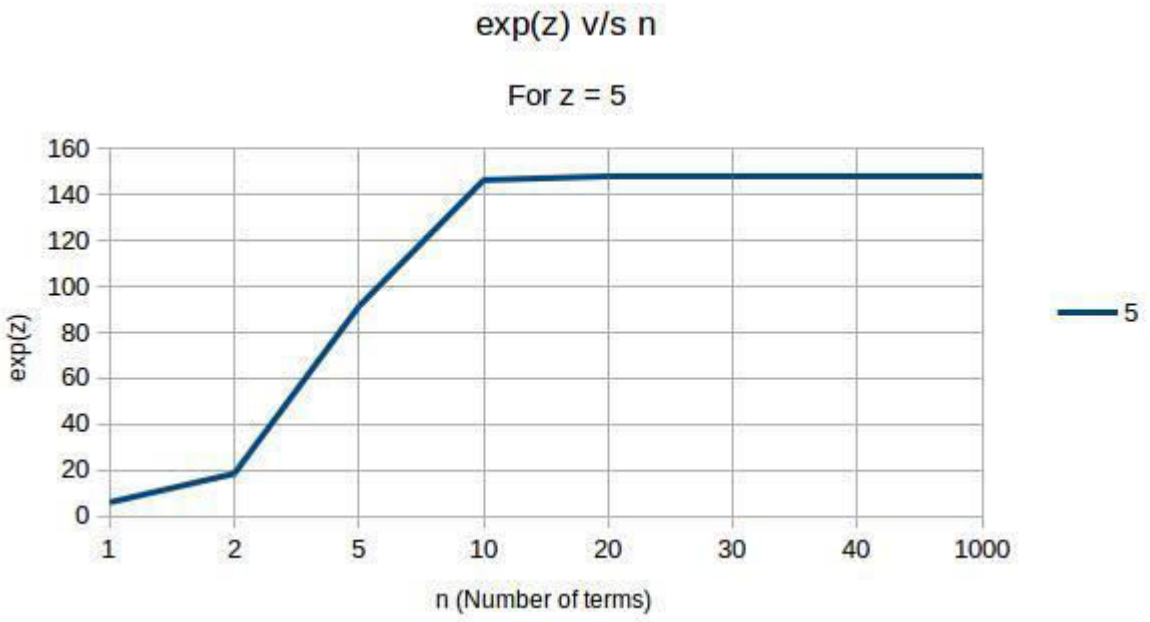

Cos: Run Time v/s Angle
For varying n [1, 1000]
Run Time (in ms)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
5
15
30
45
60
75
90
Angle (in degree)
1
2
5
50
150
500
1000

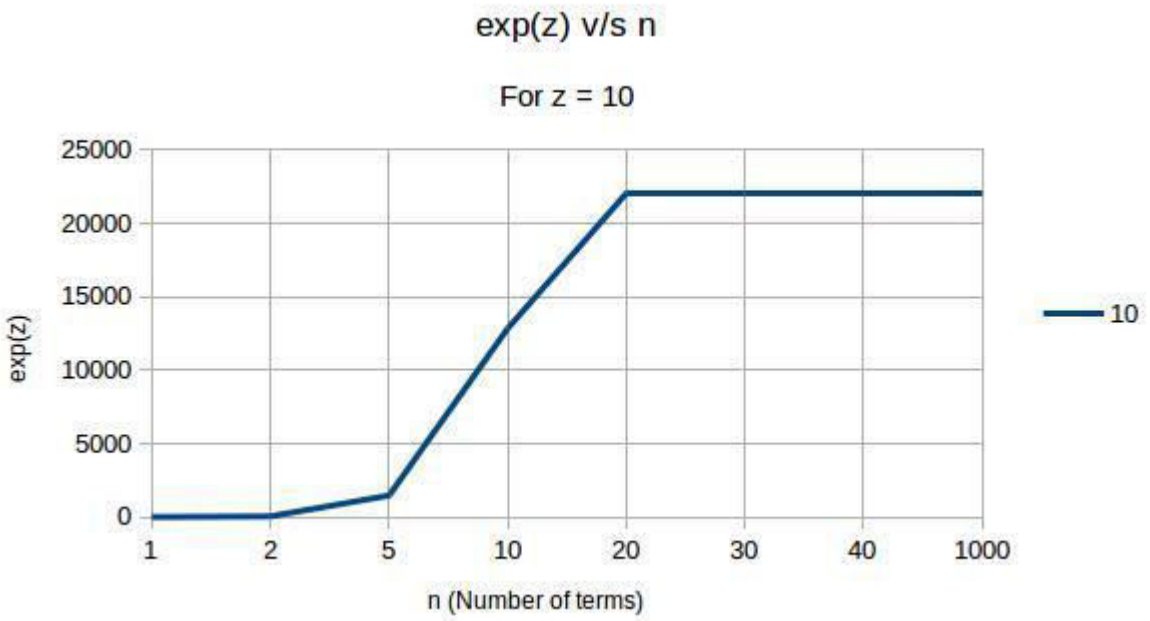

Cosine: Error v/s n
For varying z = [5, 90)
Error (in %)
50
45
40
35
30
25
20
15
10
5
0
1
2
5
50
150
500
1000
n (Number of terms)
5
15
30
45
60
75

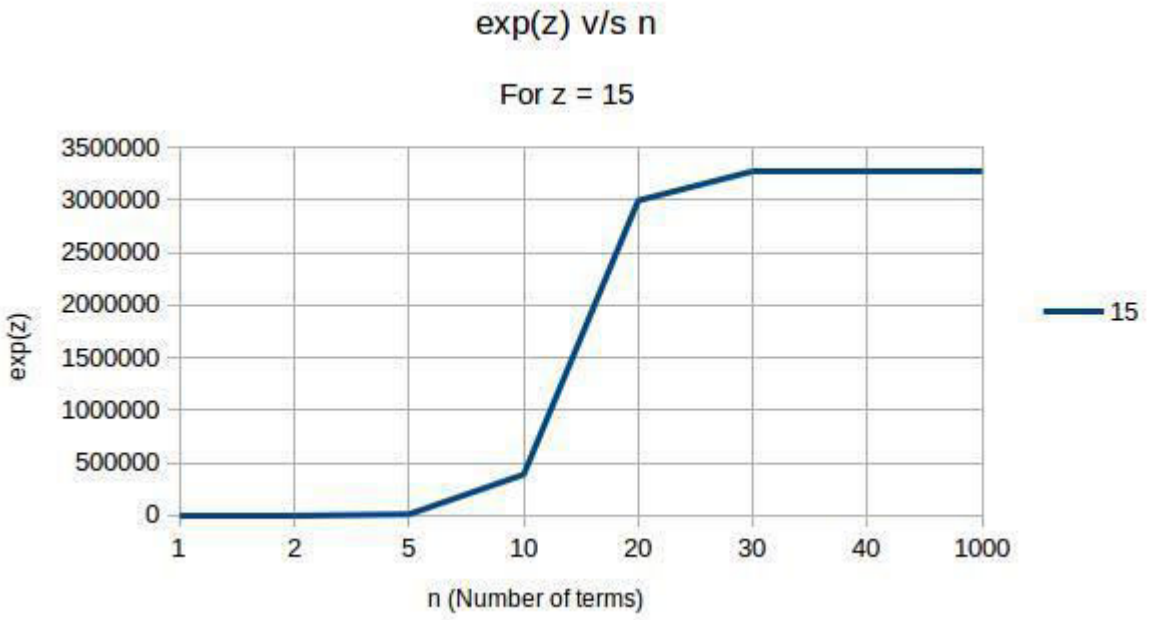


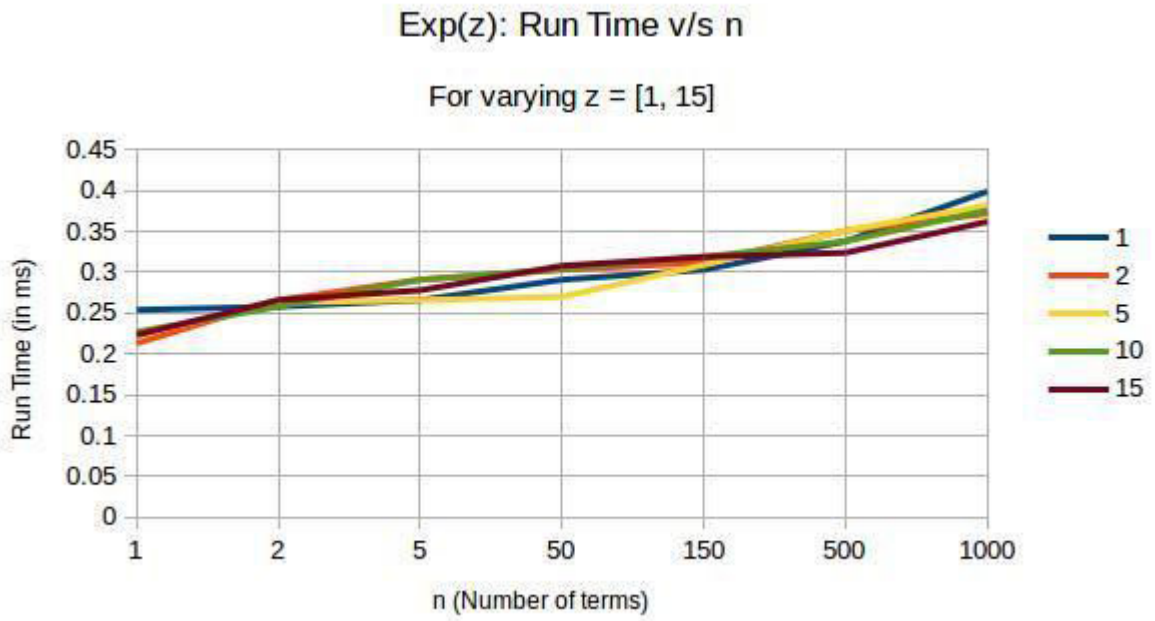
C. Exponential Series

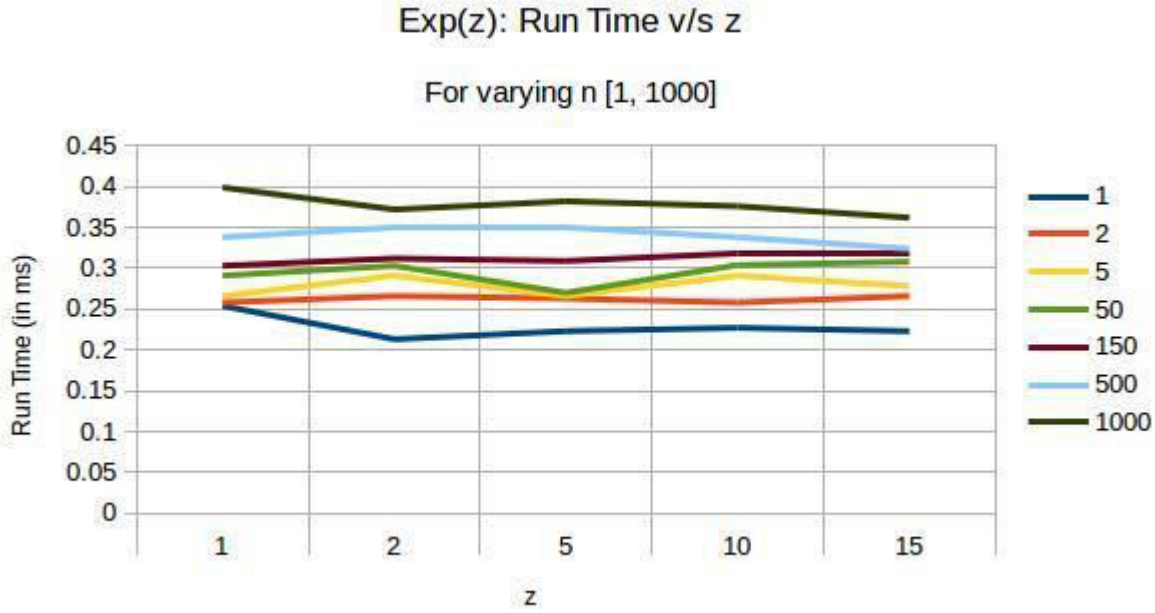

Exp(z) v/s n
For varying z = {1, 2}
exp(z)
0
1
2
3
4
5
6
7
8
1
2
5
10
20
30
40
1000
n (Number of terms)
1
2

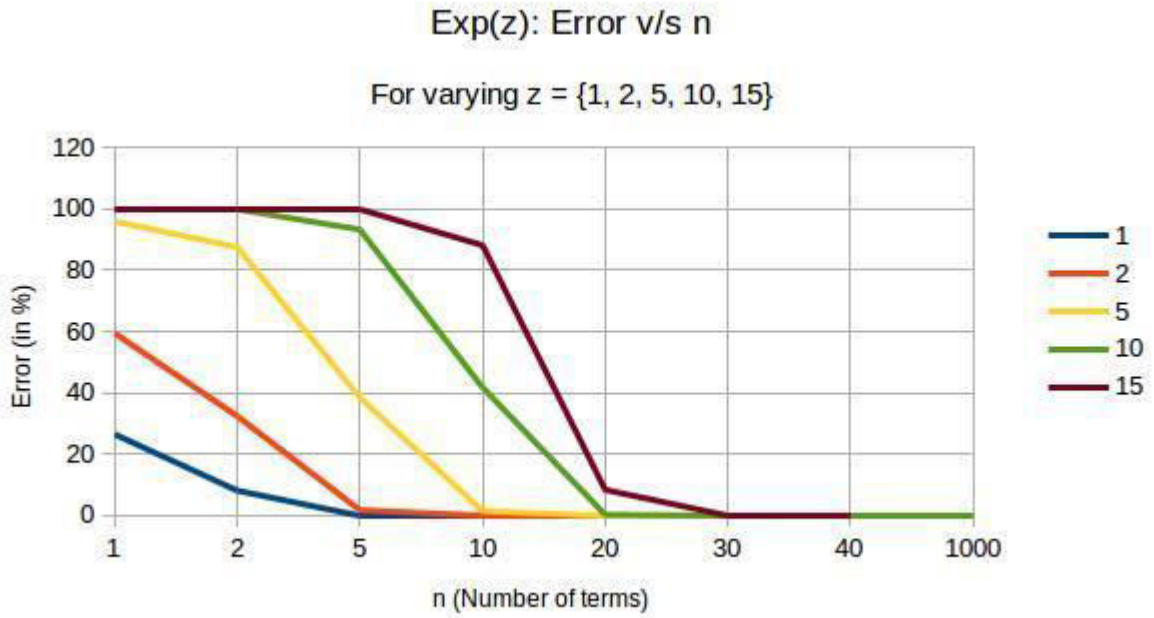

exp(z) v/s n
For z = 5
exp(z)
0
20
40
60
80
100
120
140
160
1
2
5
10
20
30
40
1000
n (Number of terms)
5


exp(z) v/s n
For z = 10
25000
20000
15000
10000
5000
0
exp(z)
1
2
5
10
20
30
40
1000
n (Number of terms)
10


exp(z) v/s n
For z = 15
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
exp(z)
1
2
5
10
20
30
40
1000
n (Number of terms)
15


Exp(z): Run Time v/s n
For varying z = [1, 15]
Run Time (in ms)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
5
50
150
500
1000
n (Number of terms)
1
2
5
10
15


Exp(z): Run Time v/s z
For varying n [1, 1000]
Run Time (in ms)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
5
10
15
z
1
2
5
50
150
500
1000



CONCLUSION

Dependency of Accuracy: As n, number of terms (can be varied by varying number of iterations) increases, accuracy increases, and converge to a constant value for large n.
Dependency of Run Time: As number of terms increases, run time also increases.
Accuracy-Time relation: From above two observations, it can be inferred that, as accuracy increases, run time also increases.

5) Image Compressor:-

SPECIFICATIONS

This program is impelmented in C++ and compresses an image given in .png aur .jpg format to smaller size image of .jpg format. When the quality factor of this code is varied we can see a change in Runtime and Accuracy of code (measured in terms of PSNR of original image and compressed image) side by side. The quality factor is a parameter in this code which defines the quality of compressed image, the more the quality factor the more is the quality of compressed image and hence more is the PSNR with original image. We have studied the variation of Quality Factor with Runtime and PSNR for 2 images(1 in PNG format and other in JPG). The following is the results we get:-

About PSNR:The PSNR block computes the peak signal-to-noise ratio, in decibels, between two images. This ratio is often used as a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the compressed, or reconstructed image.

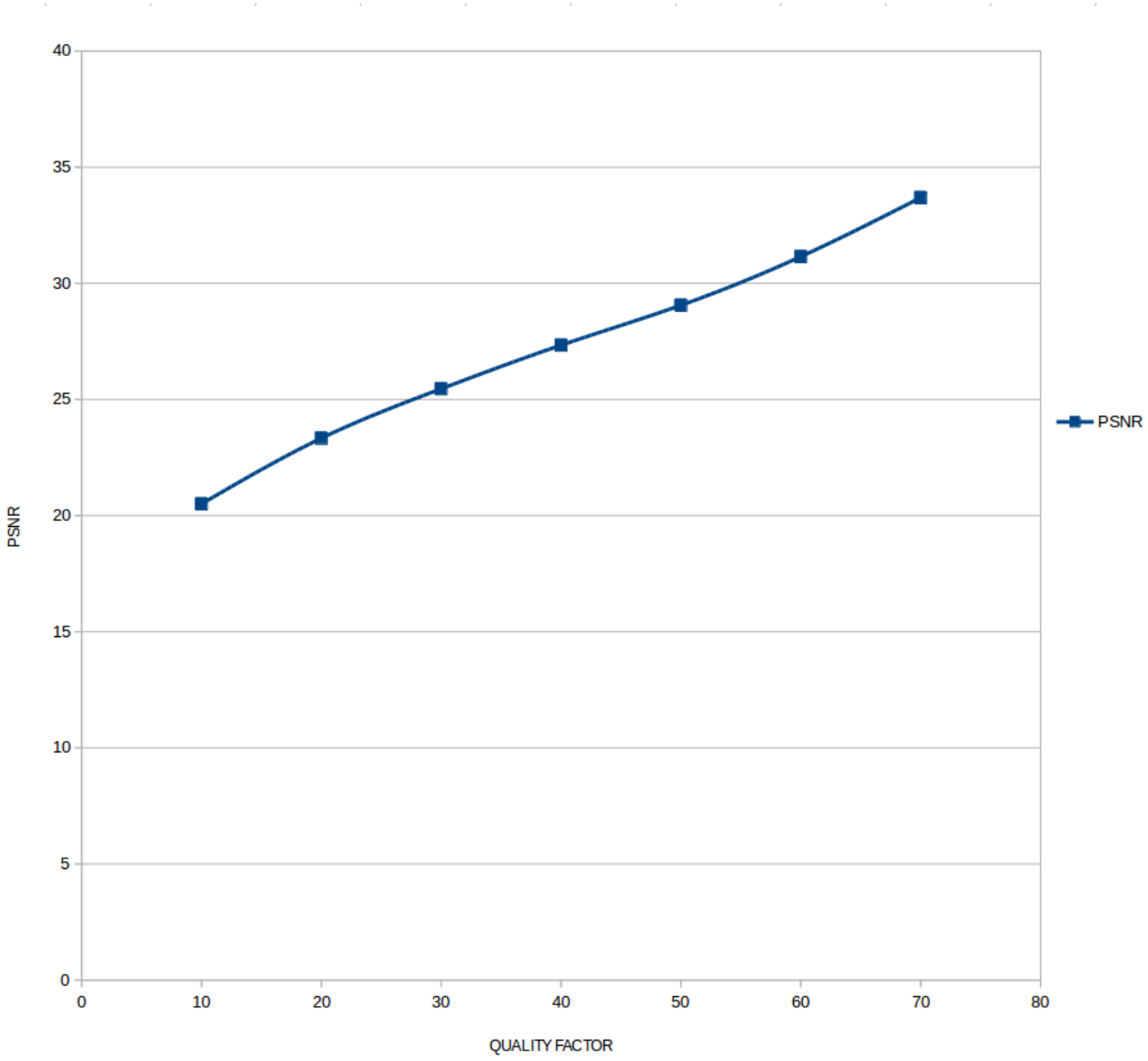
The initial **code and dataset** was taken from:-
https://github.com/kornelski/jpeg-compressor

The code was modified to include to compute and print runtime in millisecond precision using **chrono** library.

RESULTS

IMAGE1

i) Variation of PSNR(Y-axis) and Quality Factor (X-axis):-



ii) Variation of PSNR(Y-axis) and Runtime(ms)(X-axis):-

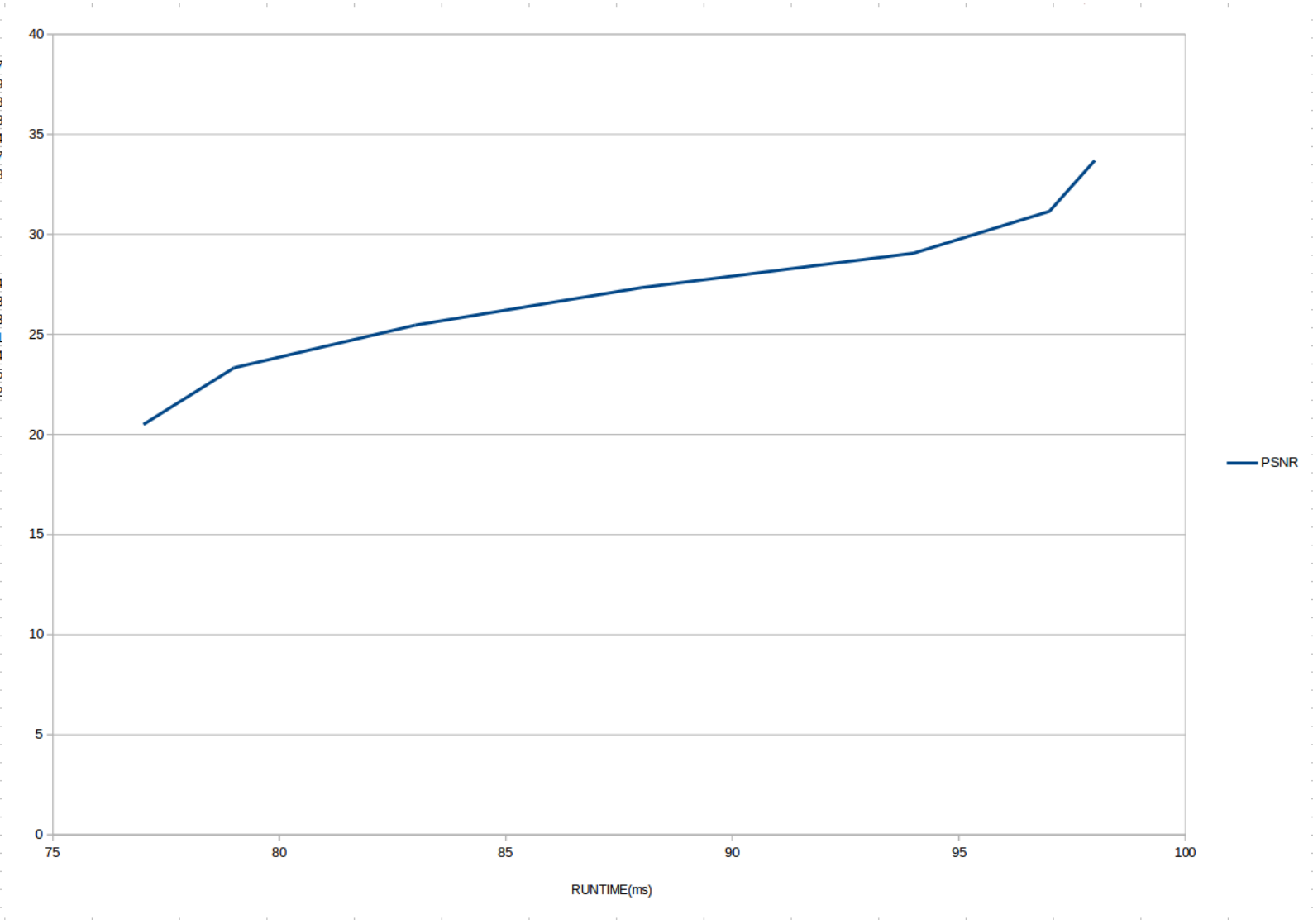
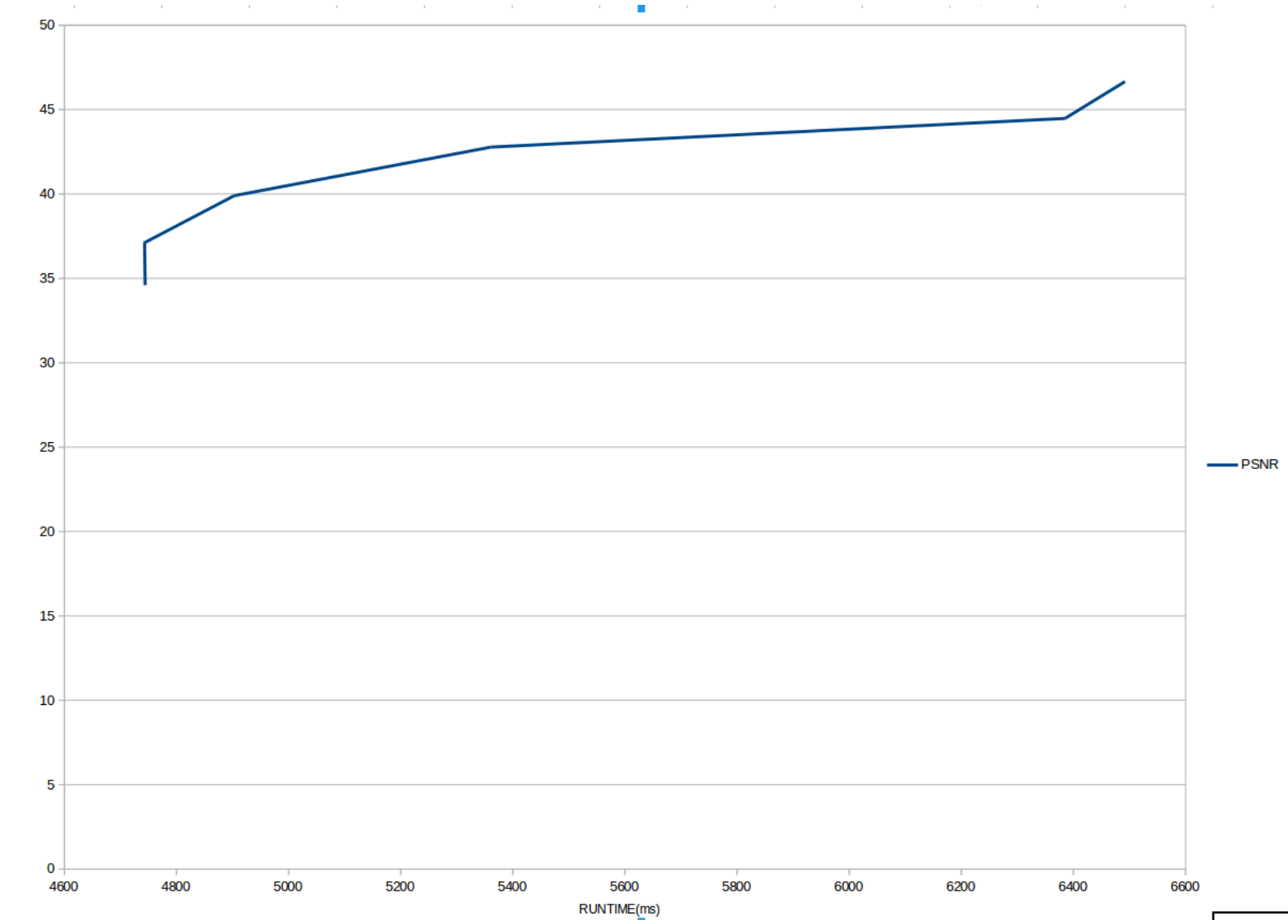
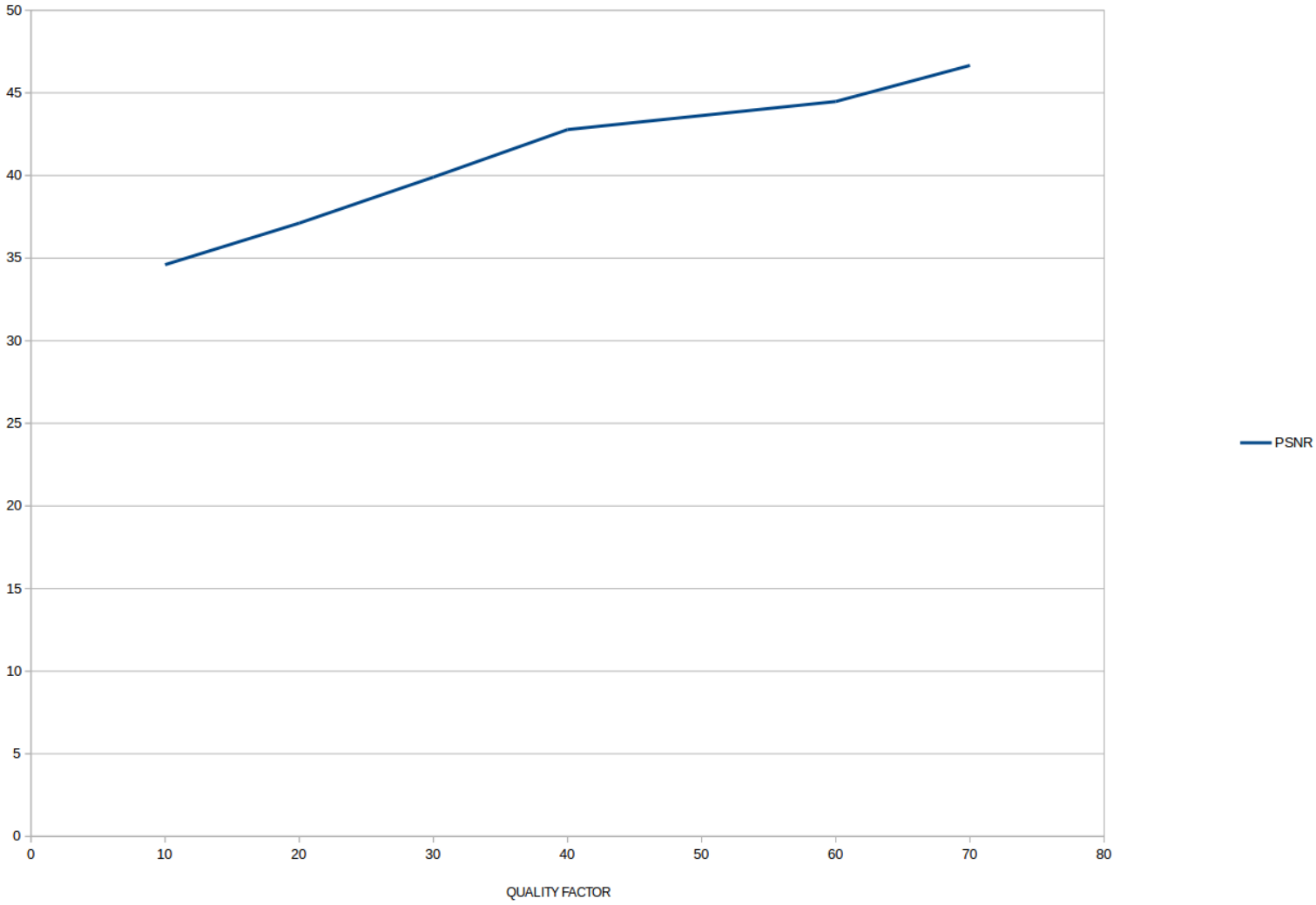


IMAGE2

i) Variation of PSNR(Y-axis) and Quality Factor (X-axis)



ii) Variation of PSNR(Y-axis) and Quality Factor(X-axis):-



CONCLUSION

Both these images follow similar trends that is Quality Factor in inversly proportional to Runtime and Accuracy(PSNR) and Accuracy(PSNR) and Runtime are directly proportional to each other. Hence when we keep high quality factor the time taken to generate such image(as the size is more) will be more and this image will be more similar to original image hence PSNR will be more.

6) Vehicle Detection :-

SPECIFICATIONS

This program is impelmented in C++ and detects the vehicles and their count in a given video for each frame. When the scale factor in this code is varied we can see a change in Runtime and Accuracy of code (measured in terms of recall) side by side. The code uses harr-cascades method of object detection and is implemented using OpenCV library. The scale factor defines the increment in size of sliding window of search after each loop the most commonly used values are between 1.1-1.5. We have studied the variation of Scale Factor with Runtime and Recall for 2 videos. The following is the results we get:-

The initial code was taken from:-
https://github.com/andrewssobral/vehicle_detection_haarcascades

The following changes are made to the code:-

1. The number of inputs of the code were changed, originally the code accepted only a video as a parameter and detected the number of cars in each frame. But now we have to provide a video with the data of number of cars in each frame and the coordinates of cars in each frame in 2 different .txt files.
2. Earlier the code was not made to store the coordinates of detected car it only counted the number of cars detected in each frame but now it is modified to also store the coordinates of each detection.
3. The code is modified to compute the accruacy in terms of three measures which are **Precision, Recall and Estimated Accuracy**. The Estimated Accuracy is only based on count of cars detected and actually present in the frame but Precision and Recall depend on the number of true positives, false positives and false negatives in each frame.

Where **Recall** = tp / tp+fn
and **Precision** = tp / tp+fp

where tp=True Positive, fp=False Positive, fn= False negative for each frame and Overall Precision and Recall are the averages of Precision and Recall for all frames.

4. The code is also modified to compute and print the Running Time of program.
5. Various parameters used in code were earlier fixed in the code and to change them we had to make changes in the code but now they can be taken as input during execution.

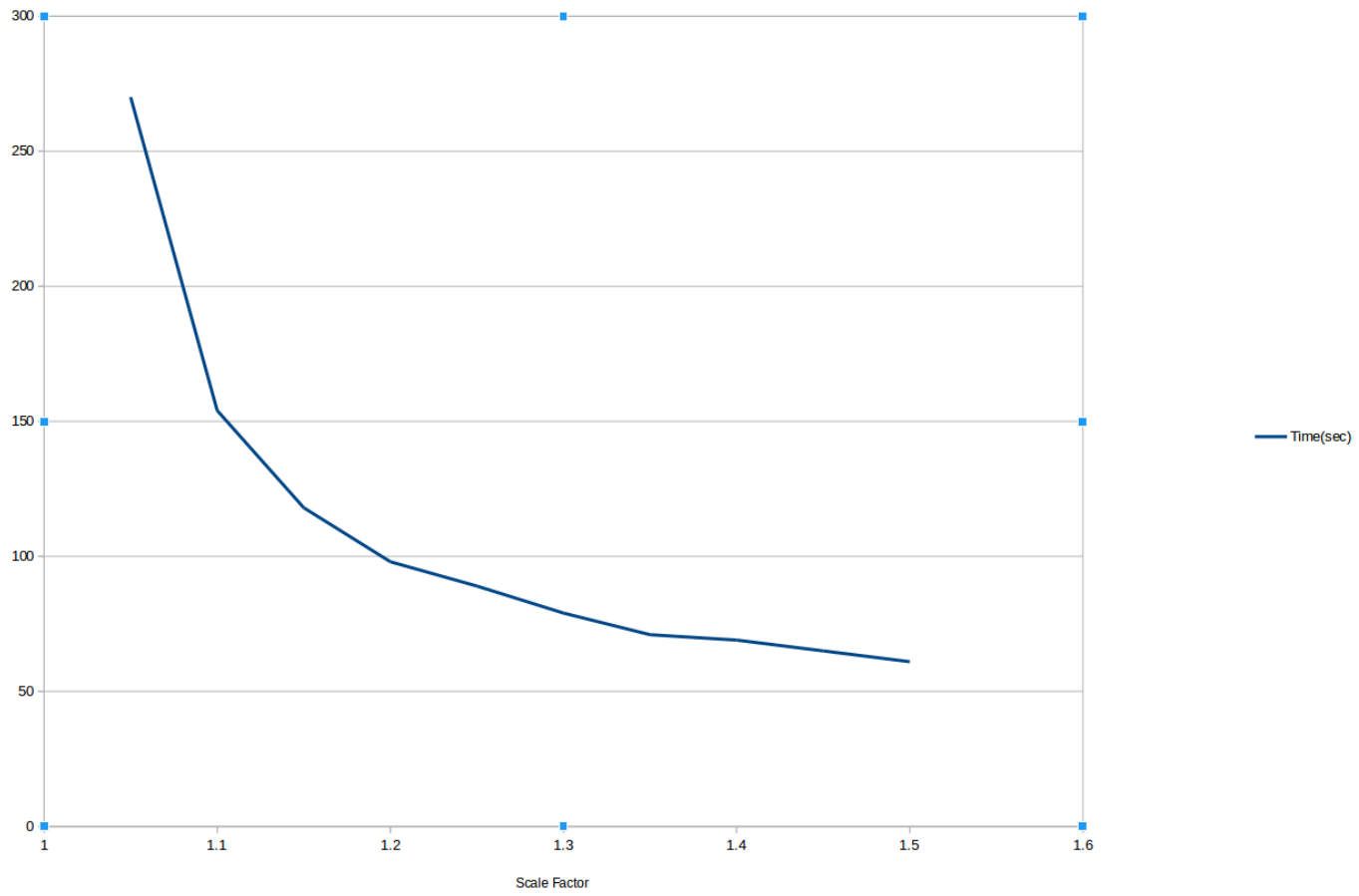
DATASET:-The Dataset used is from **LISA** and taken from:-

http://cvrr.ucsd.edu/LISA/vehicledetection.html .

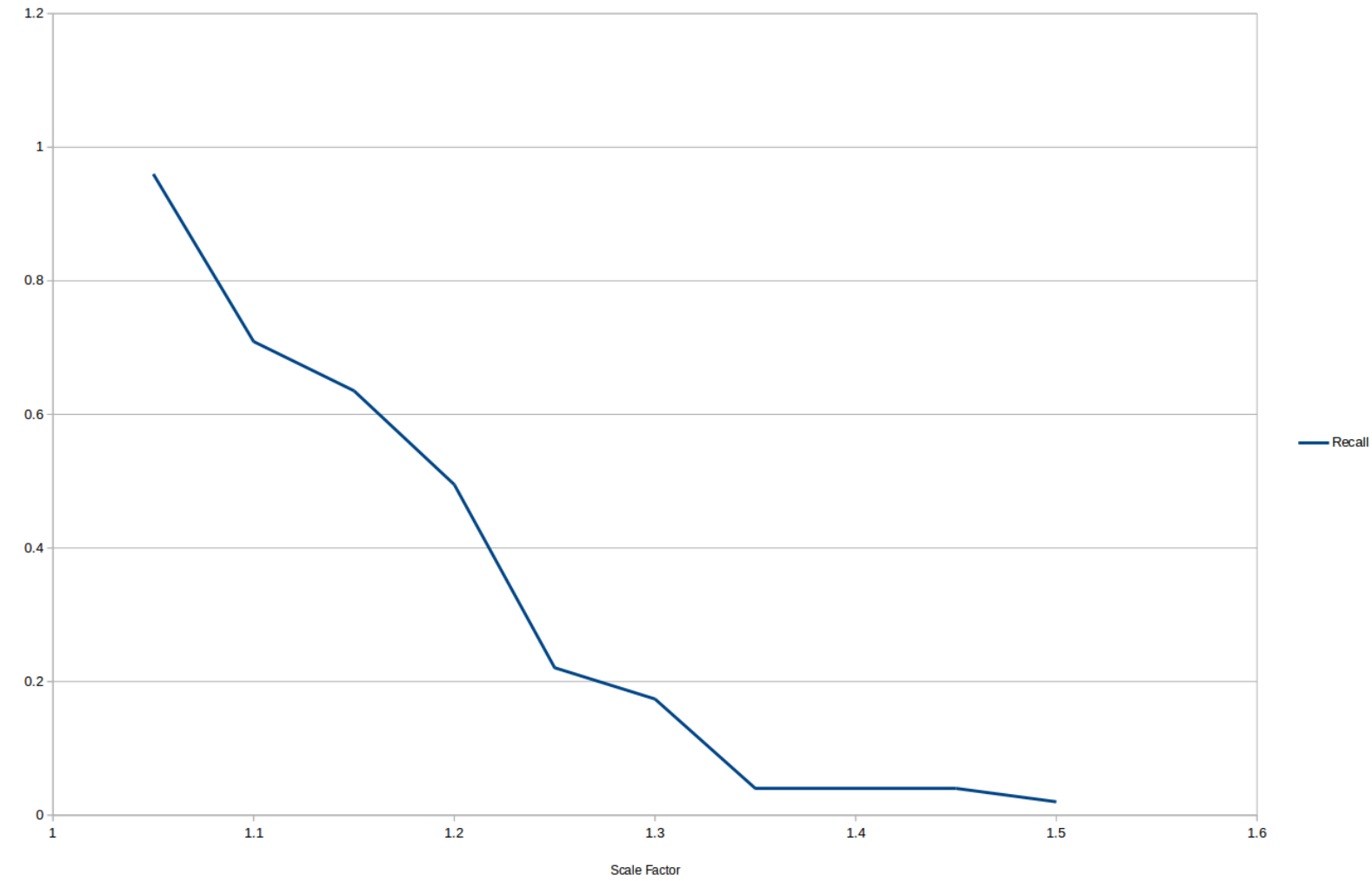
RESULTS

Video1

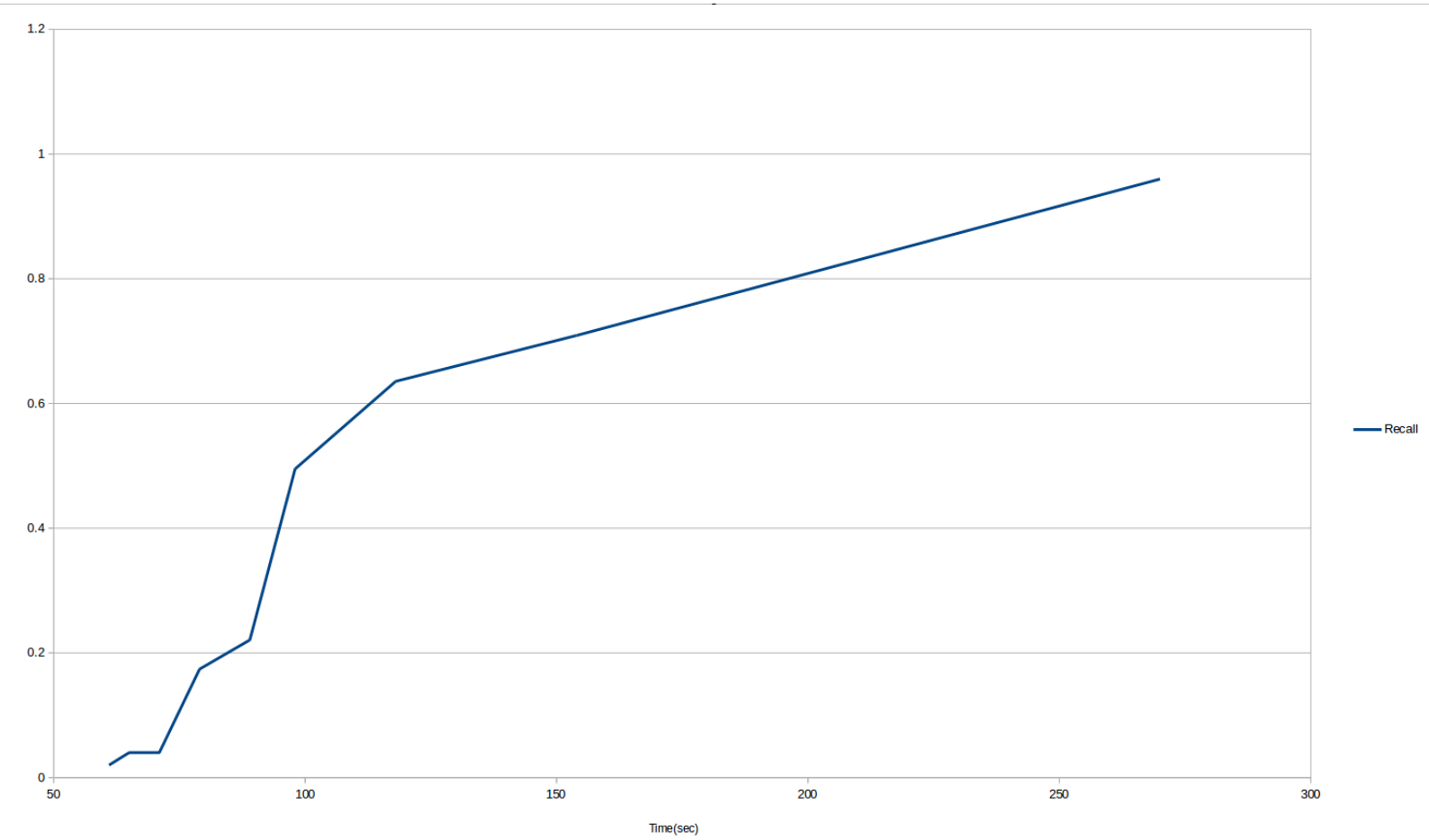
i) Variation of Runtime(sec) (Y-axis) and Scale Factor (X-axis):-



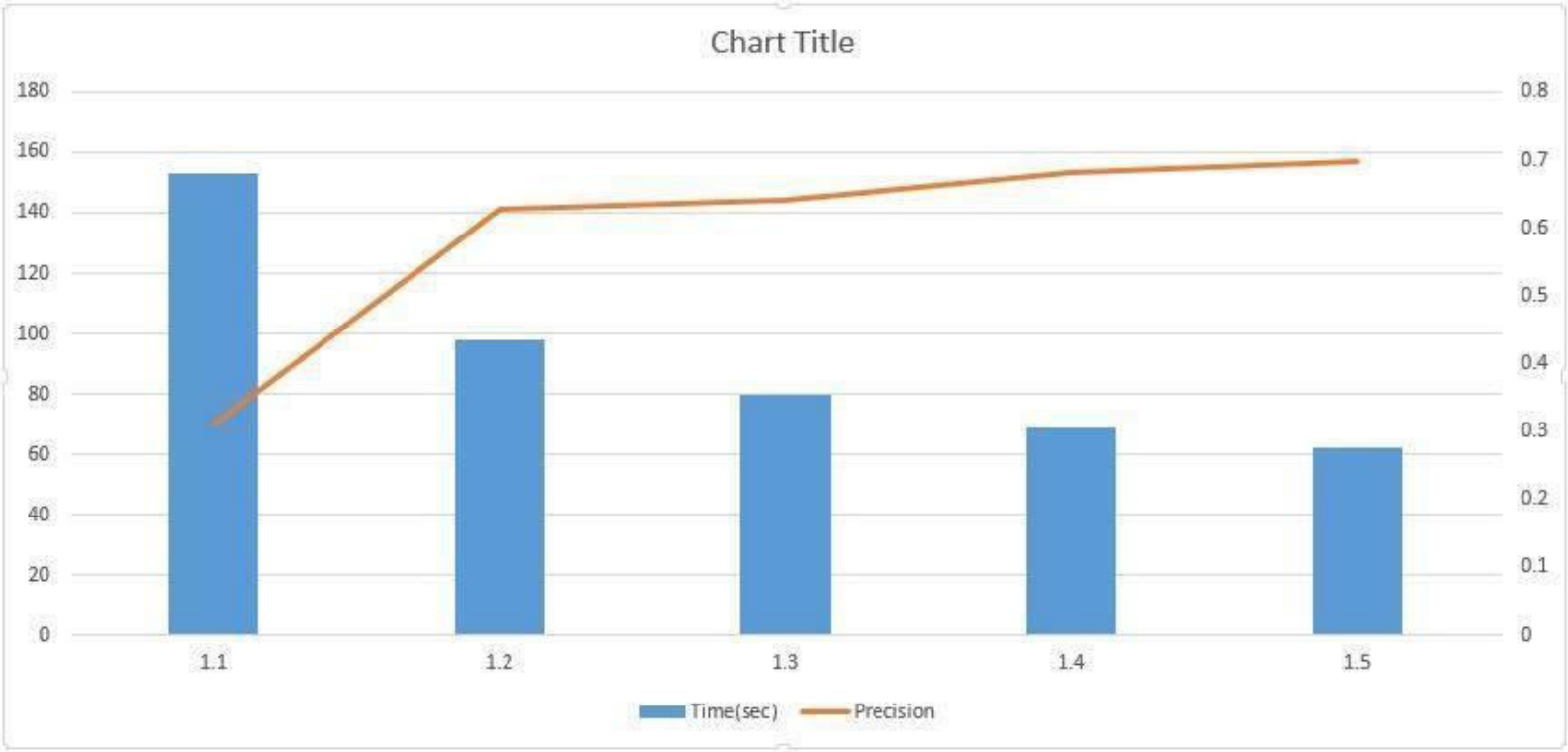
ii) Variation of Accuracy(Recall)(Y-axis) and Scale Factor(X-axis):-



iii) Variation of Accuracy(Recall) (Y-axis)and Runtime(sec)(X-axis):-



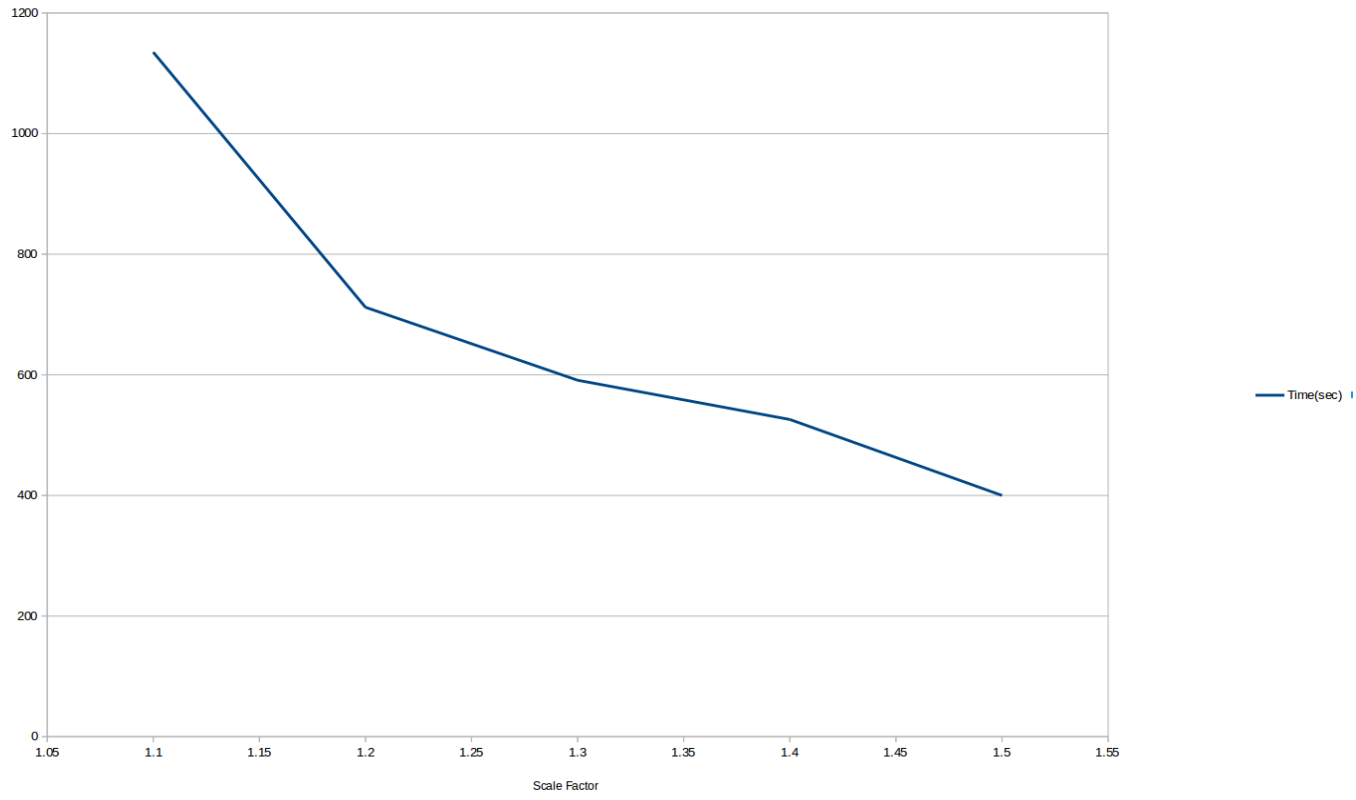
iv) Variation of Precision and Runtime(sec)(Y-axis) V/s ScaleFactor(X-axis):-



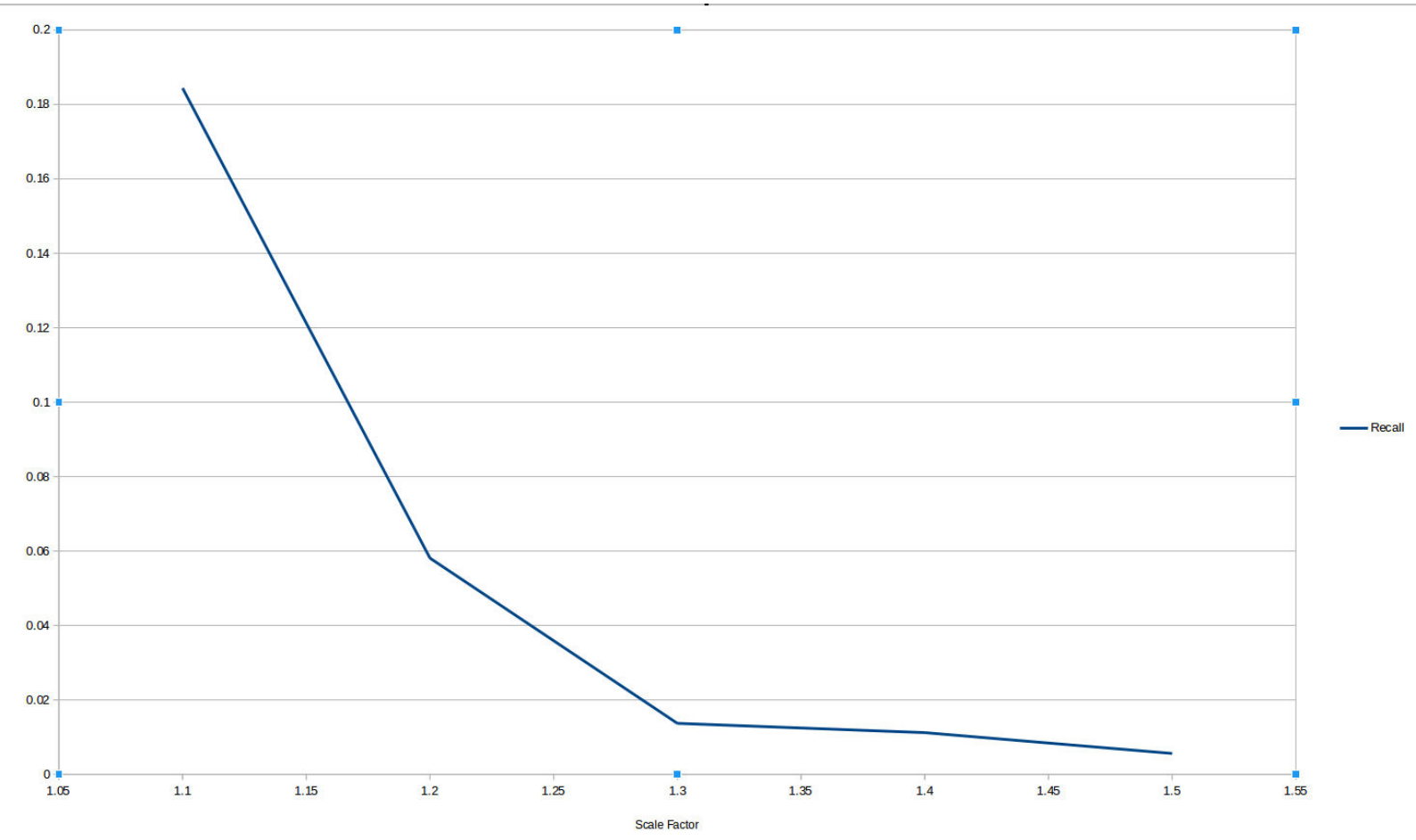
Scale Factor

Video 2

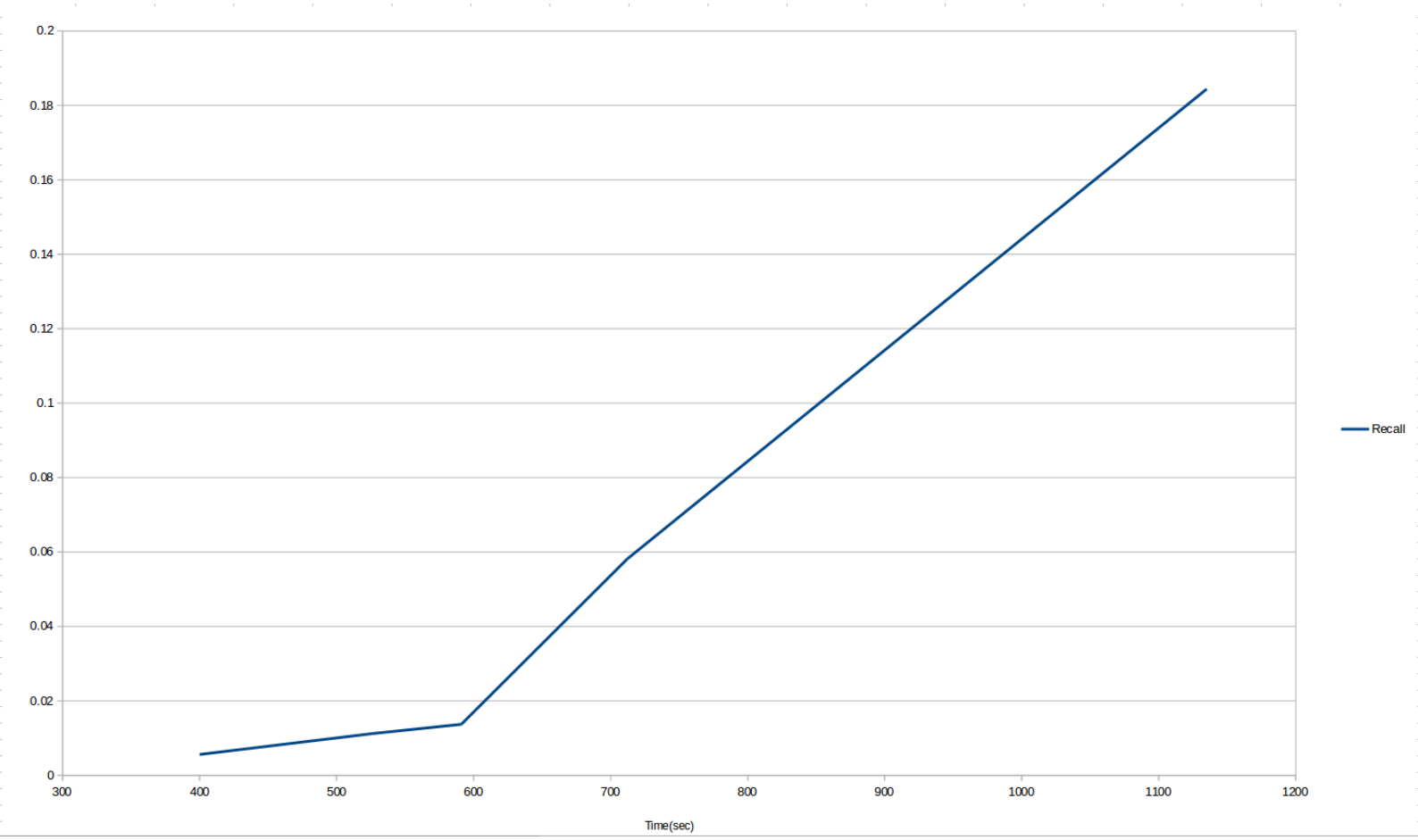
i) Variation of Runtime(sec) (Y-axis) and Scale Factor (X-axis)



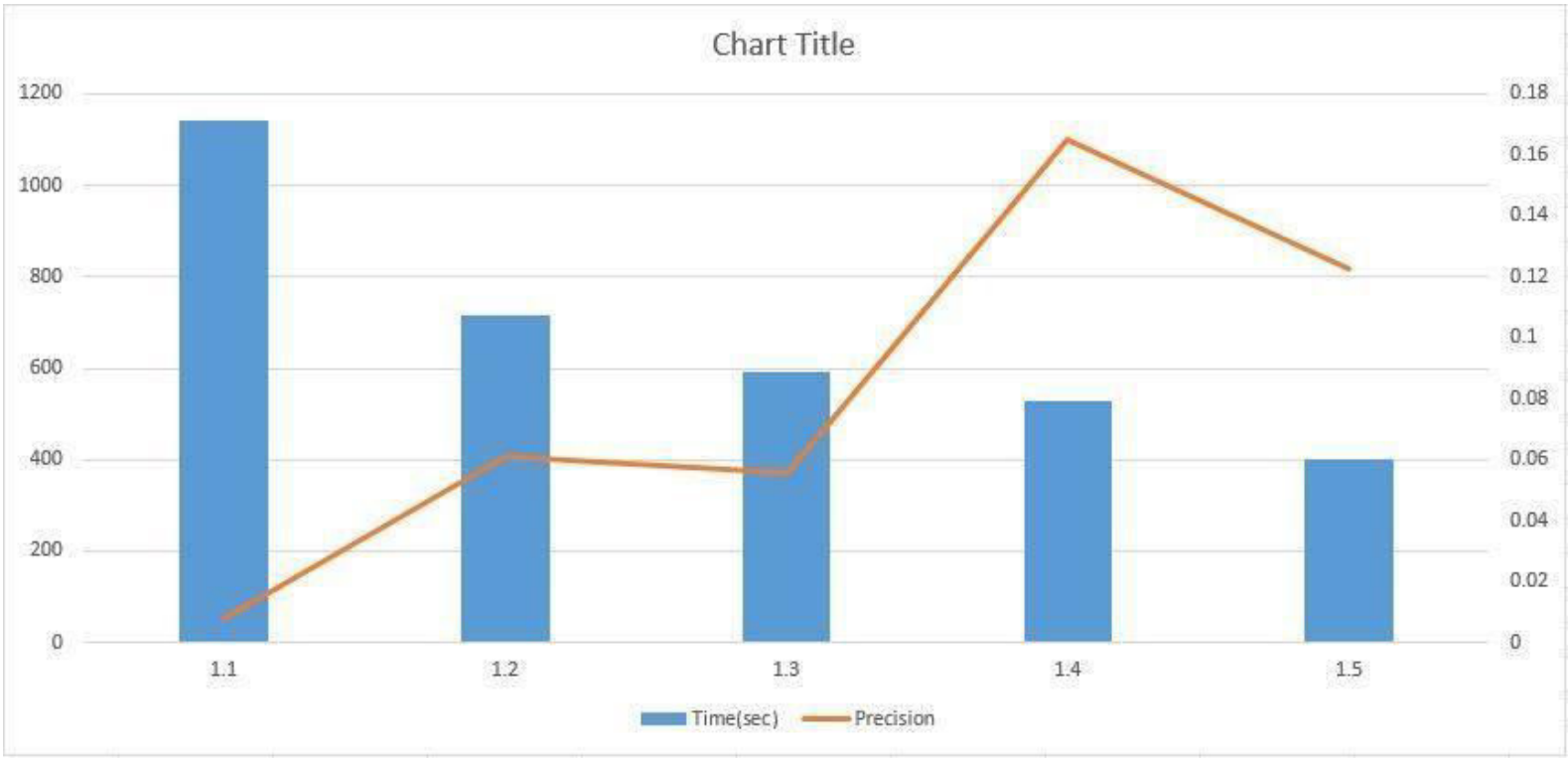
ii) Variation of Accuracy(Recall)(Y-axis) and Scale Factor(X-axis):-



iii) Variation of Accuracy(Recall) (Y-axis)and Runtime(sec)(X-axis):-



iv) Variation of Precision and Runtime(sec)(Y-axis) V/s ScaleFactor(X-axis):-



CONCLUSION

Both these videos follow similar trends that is Scale Factor in inversly proportional to Runtime and Accuracy and Accuracy and Runtime are directly proportional to each other. Hence when we do more fine search

time taken would be more but the result will also be more accurate and vice-versa.