CS386-Artificial Intelligence Lab

Report and Observations : Group 12

Sahil Jindal 110020043

Abhishek Gupta 110040067

Rohan Gyani 110040001

Mridul Ravi Jain 110040083

Back Propagation on Feed Forward Neural Networks

Report and Observations

Assignment Specifications

- Implement back propagation (BP) on feed forward neural n/w (FFNN). Give FFNNs for:
  - *2-input XOR*
  - *2-input NAND*
  - *5-input palindrome*
  - *5-input majority*
  - *5-input parity*
  - *Digit recogniser*
- Choose the learning rate judiciously.
- Study convergence time, local minima, saturation, effect of initialization, effect of learning rate and momentum factor.
- 1 and 0 decisions are based on the output being above the high water mark or being below the low water mark.

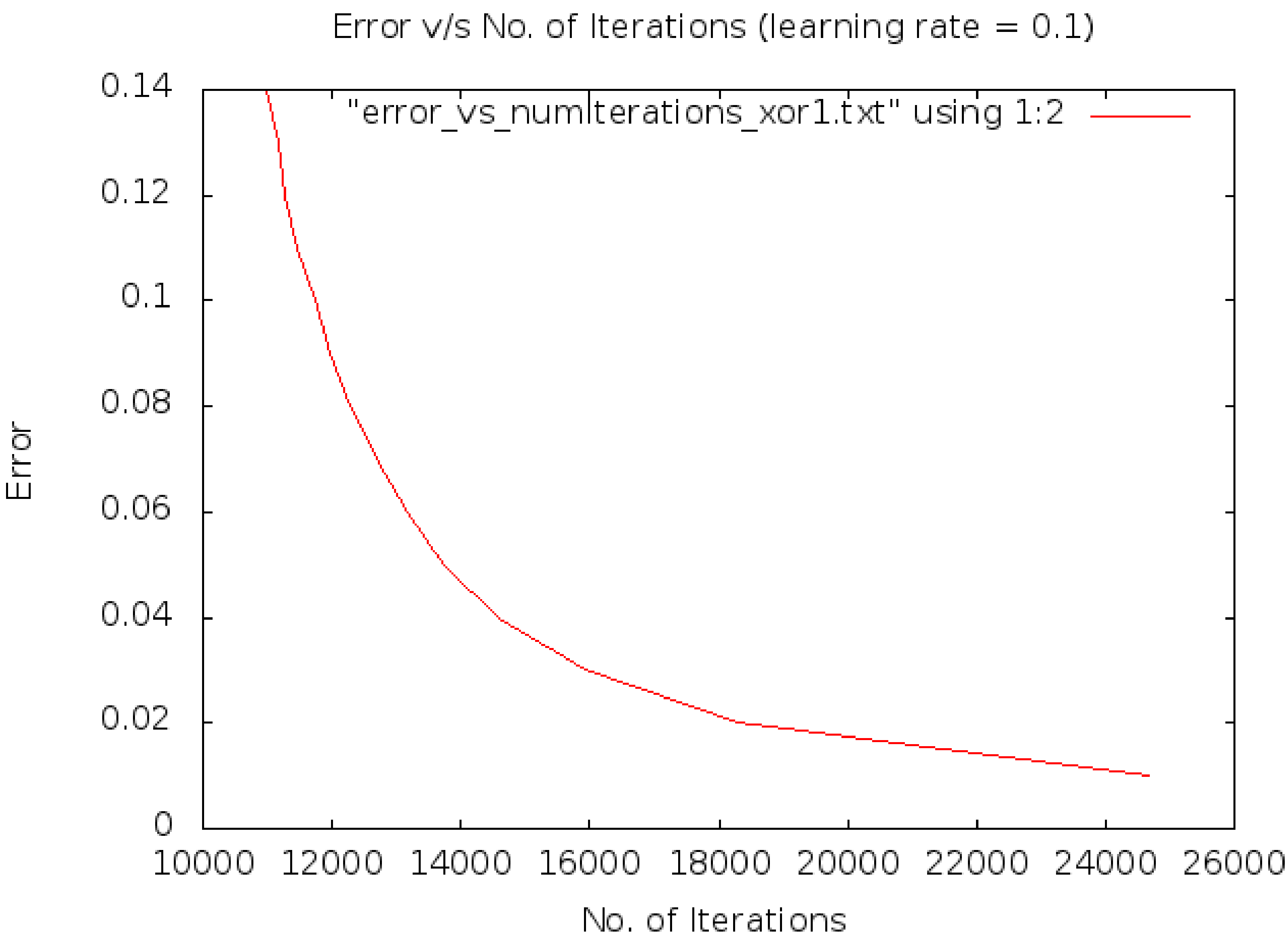
FFNN Pseudo Code

- Apply the inputs to the network and work out the output.
- Calculate errors of output neurons and then calculate Total Sum of Squares Error
- while(Total Sum of Squares Error > Threshold)
    - Change output layer weights
    - Calculate errors for hidden layer neurons
    - Change hidden layer weights(backpropagation)
    - Apply the inputs to the network and work out the output.
    - Calculate errors of output neurons and then calculate Total Sum of Squares Error

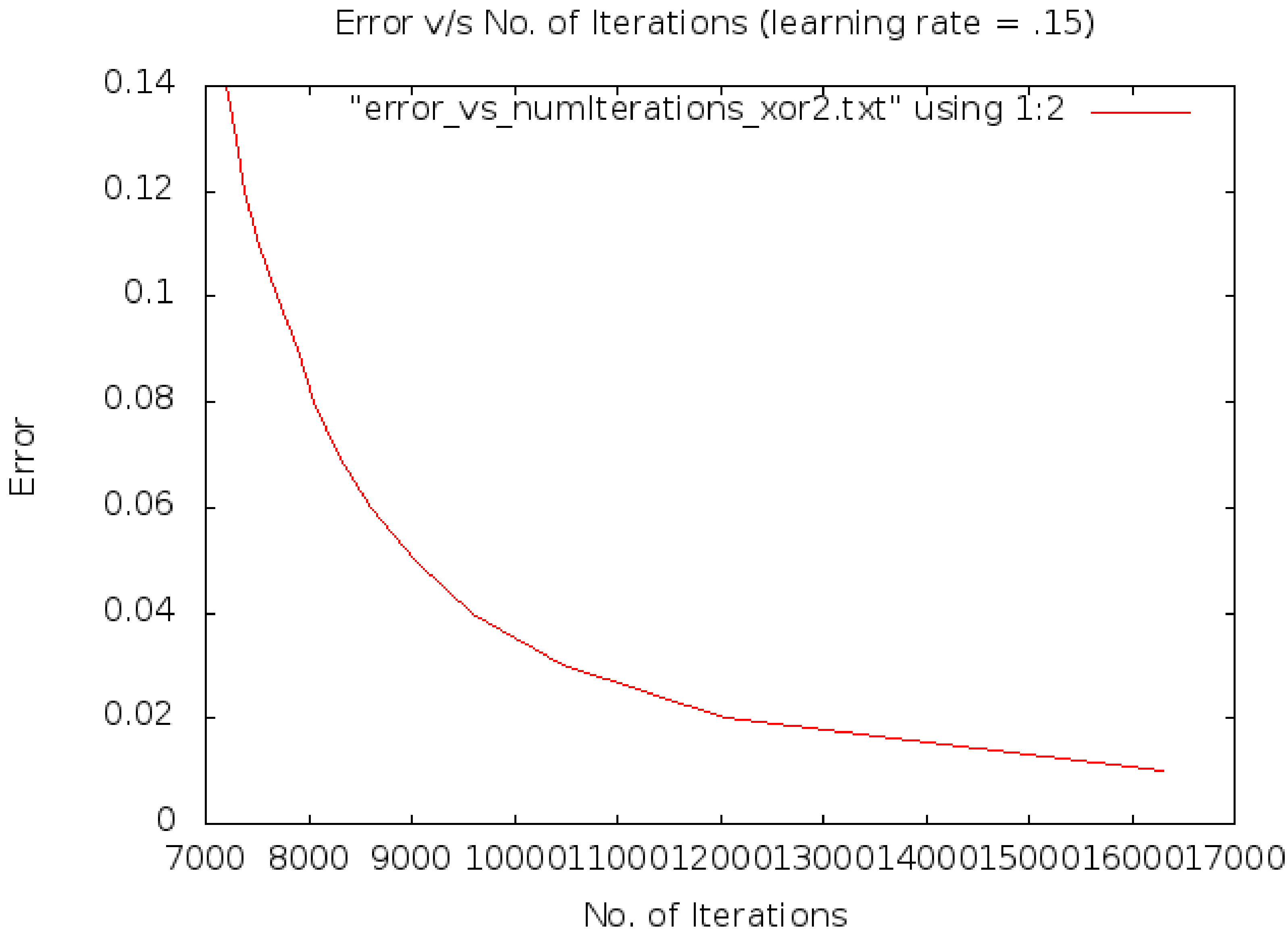
Contents

- Error v/s No of Iterations
  - Effect of Learning Rate
  - Effect of Momentum factor
- Graphs for
  - *2-input XOR*
  - *2-input NAND*
  - *5-input palindrome*
  - *5-input majority*
  - *5-input parity*
  - *Digit recogniser*
- *Functionality of Hidden Layer Neurons*
  - *2-input XOR*
  - *3-input Palindrome*
  - *5-input Palindrome*

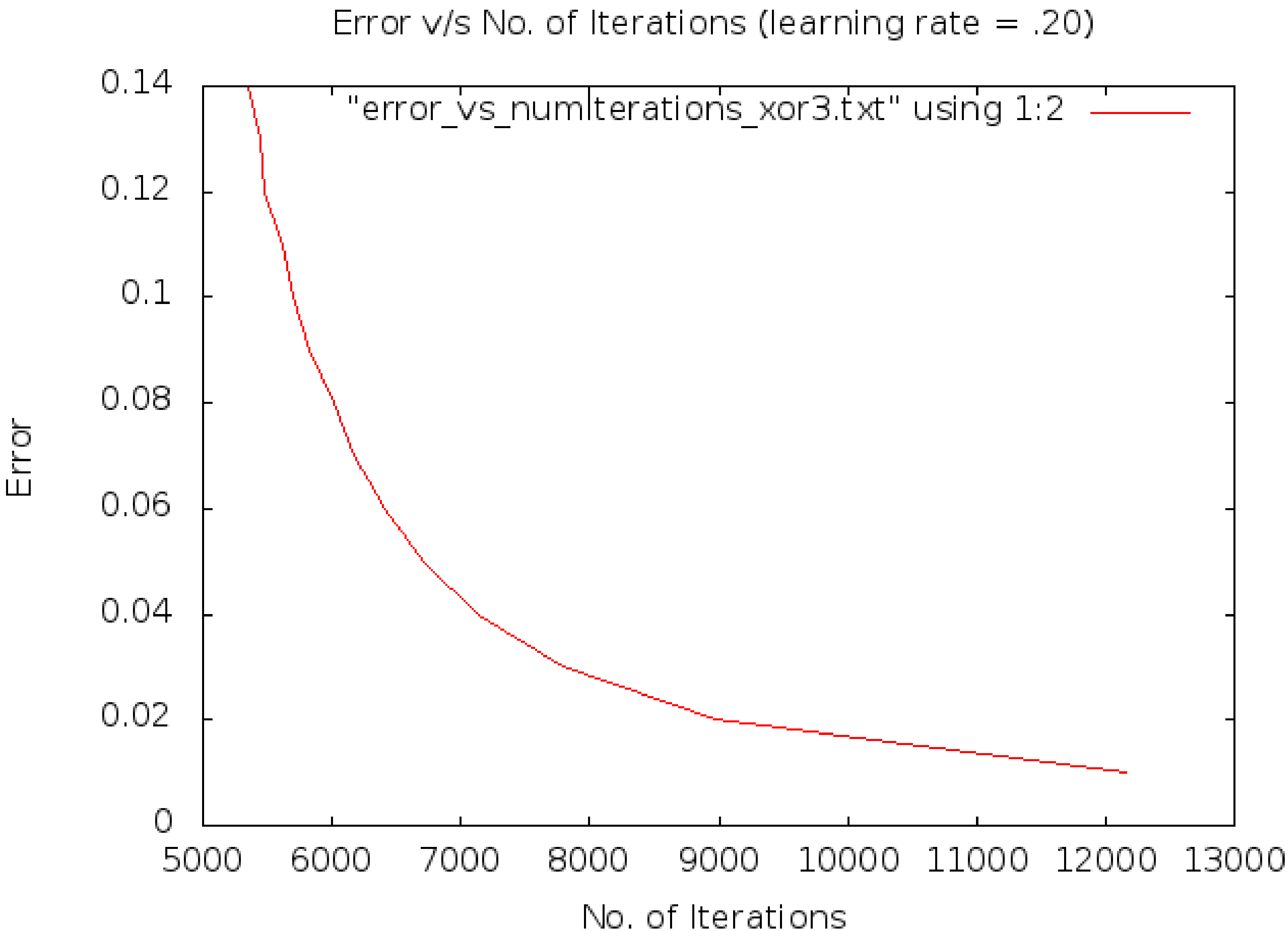
Effect of Learning Rate (XOR)



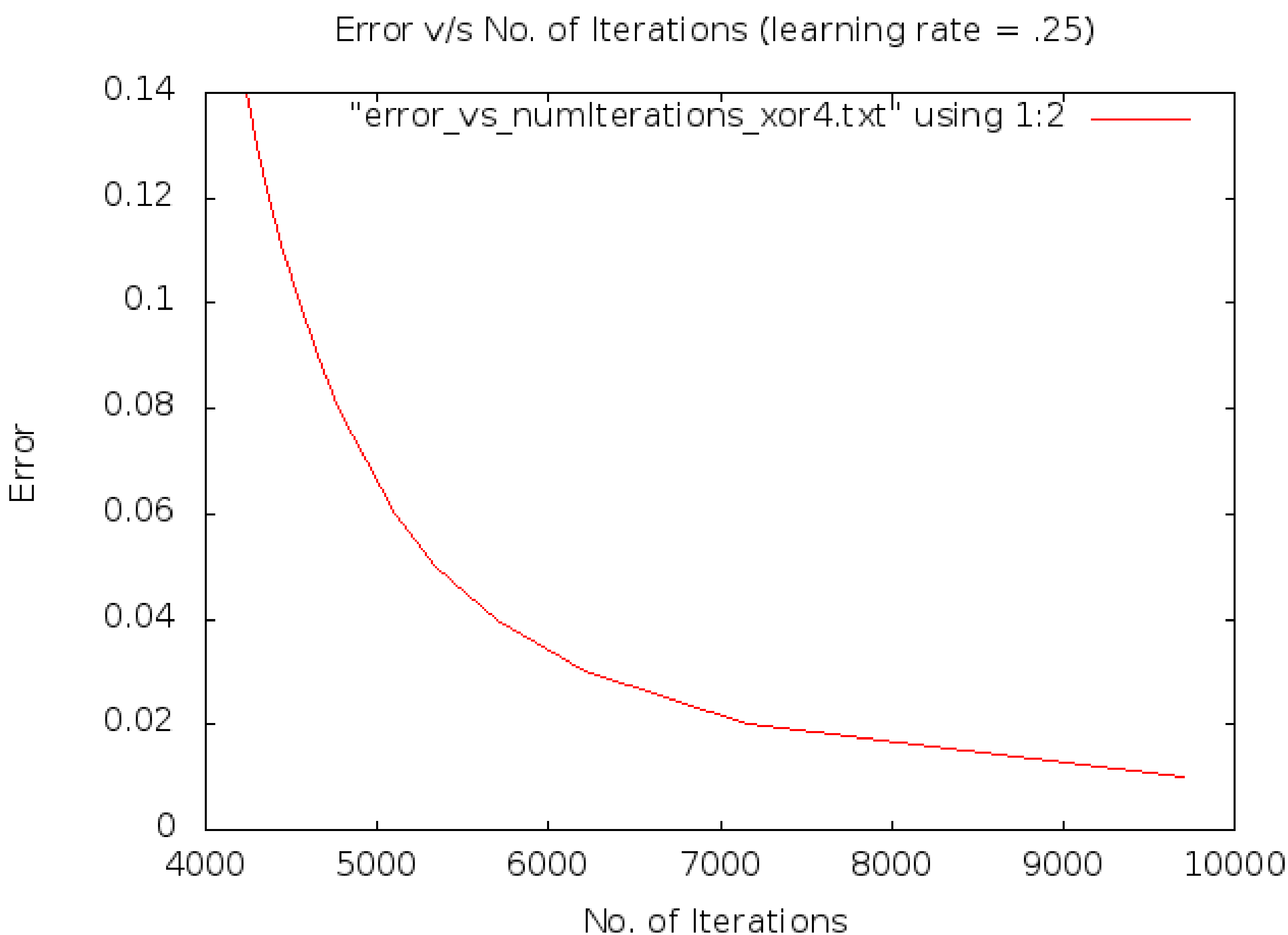
Effect of Learning Rate (XOR)



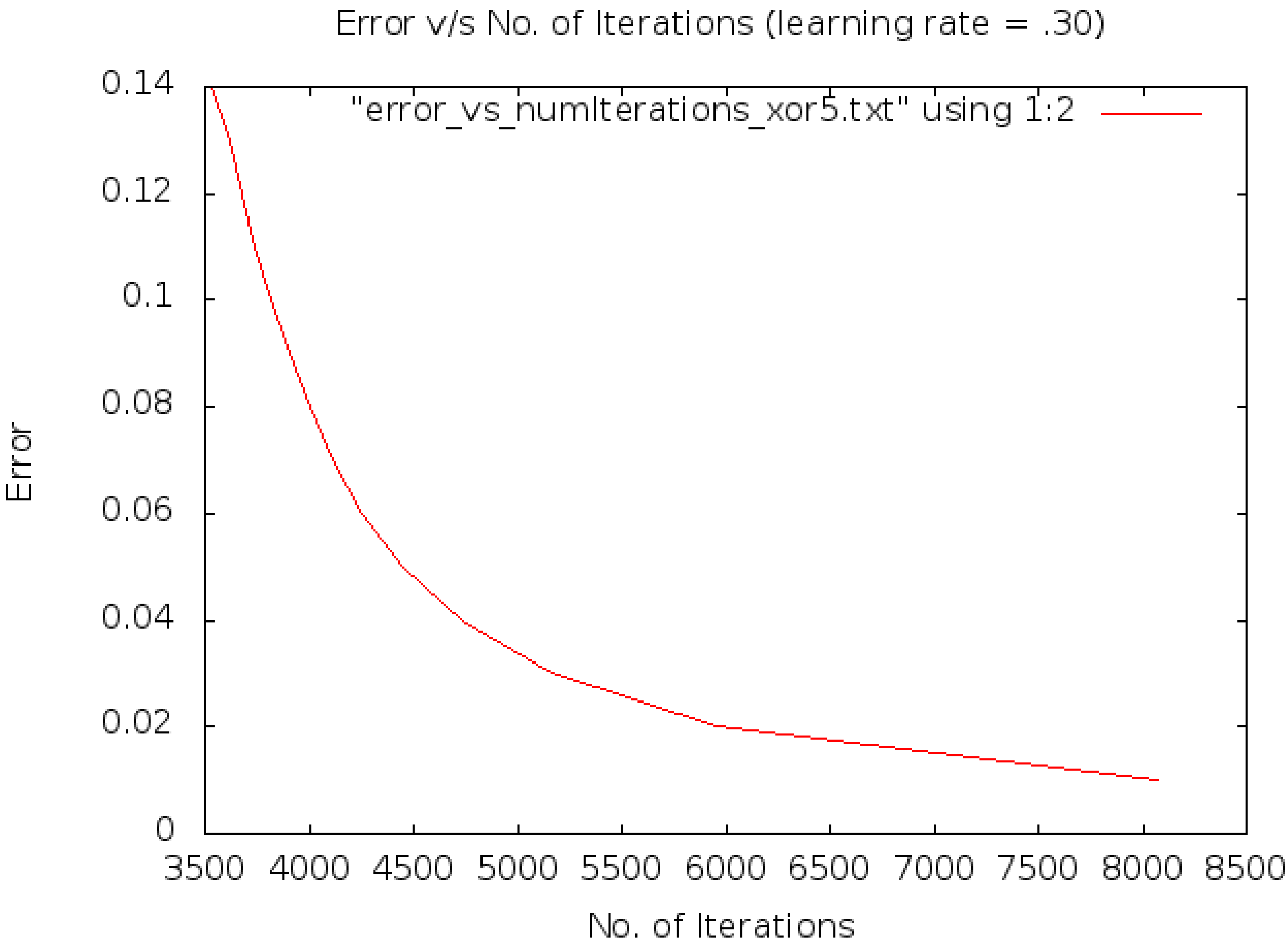
Effect of Learning Rate (XOR)



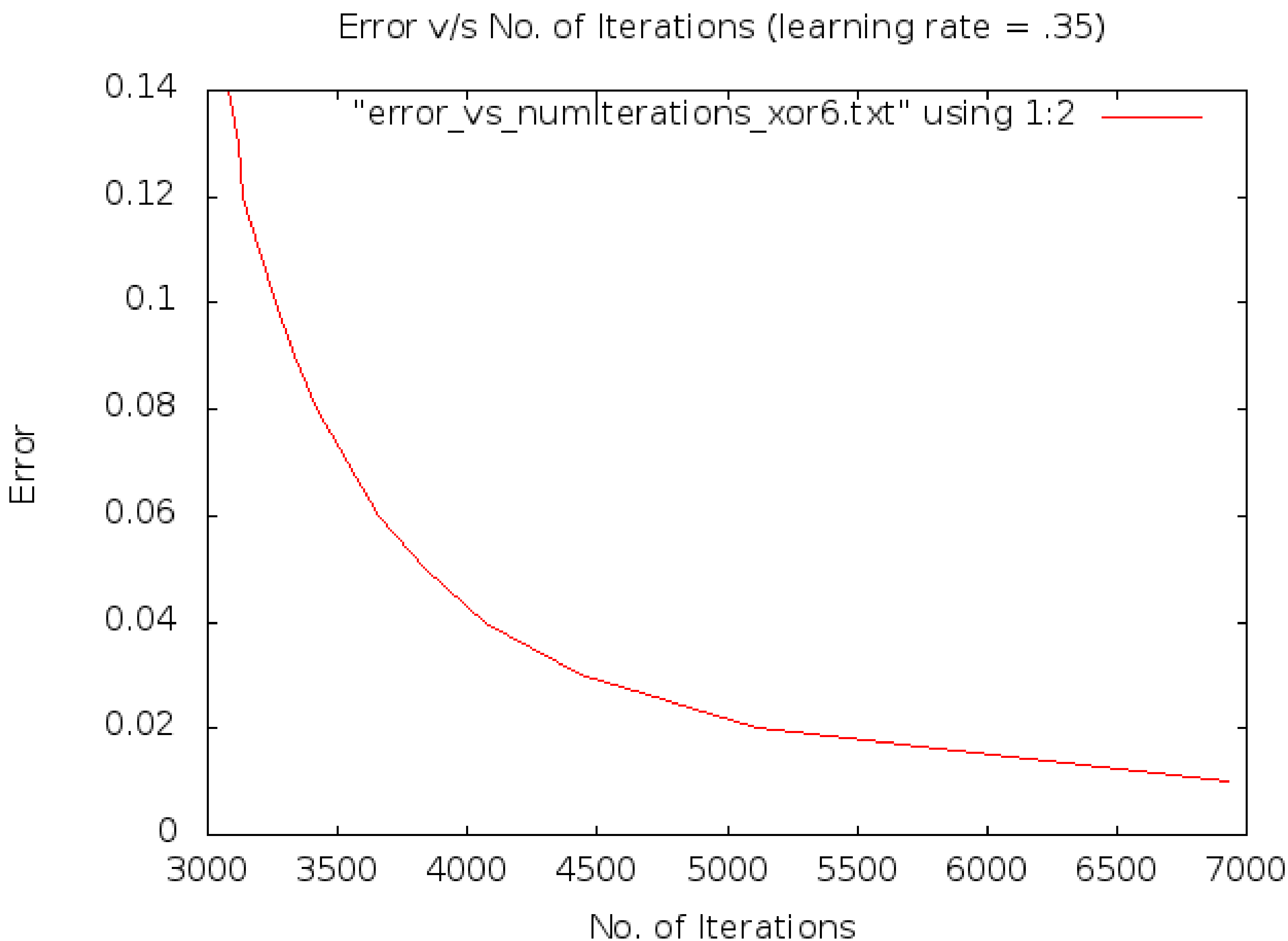
Effect of Learning Rate (XOR)



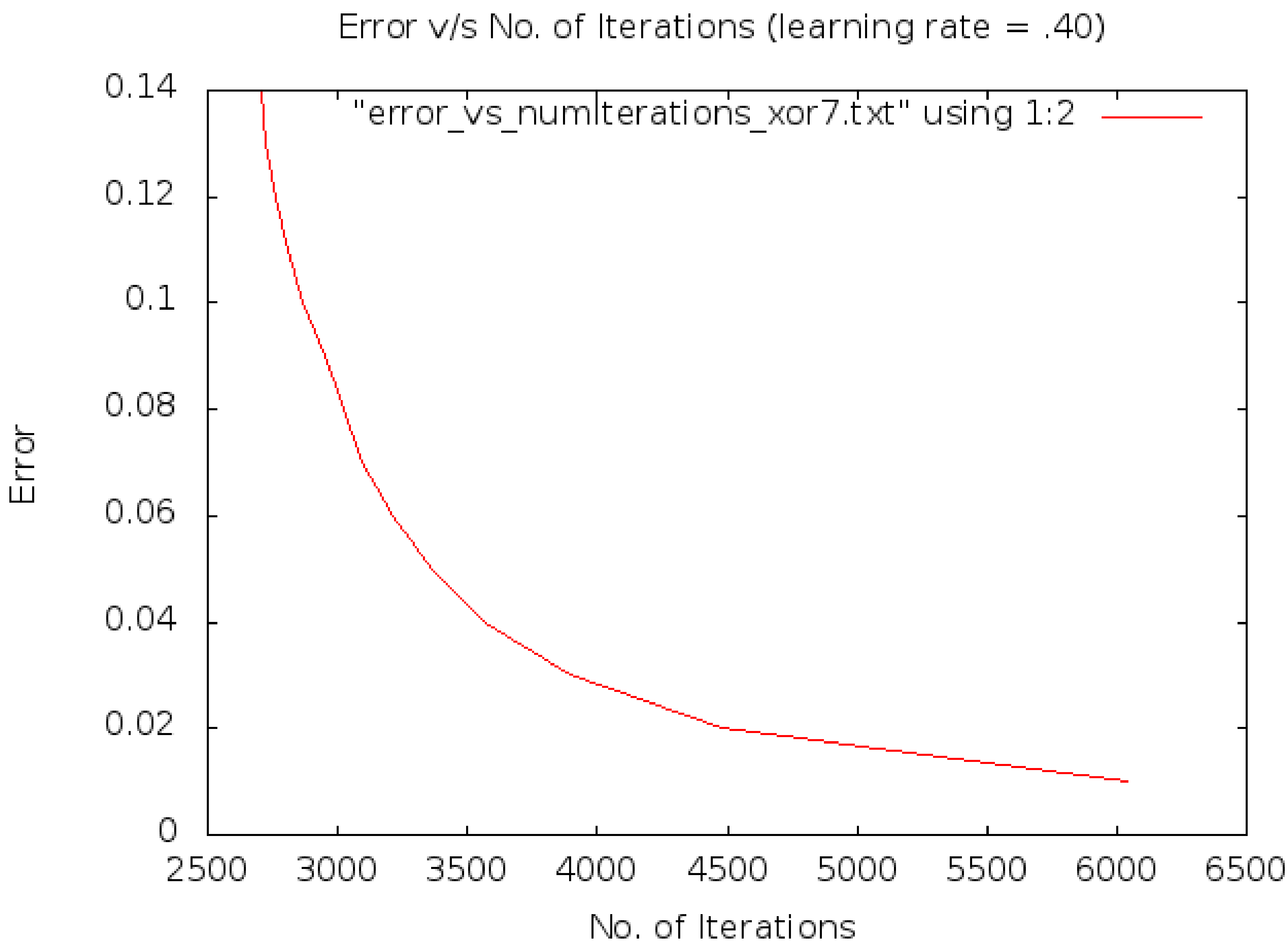
Effect of Learning Rate (XOR)



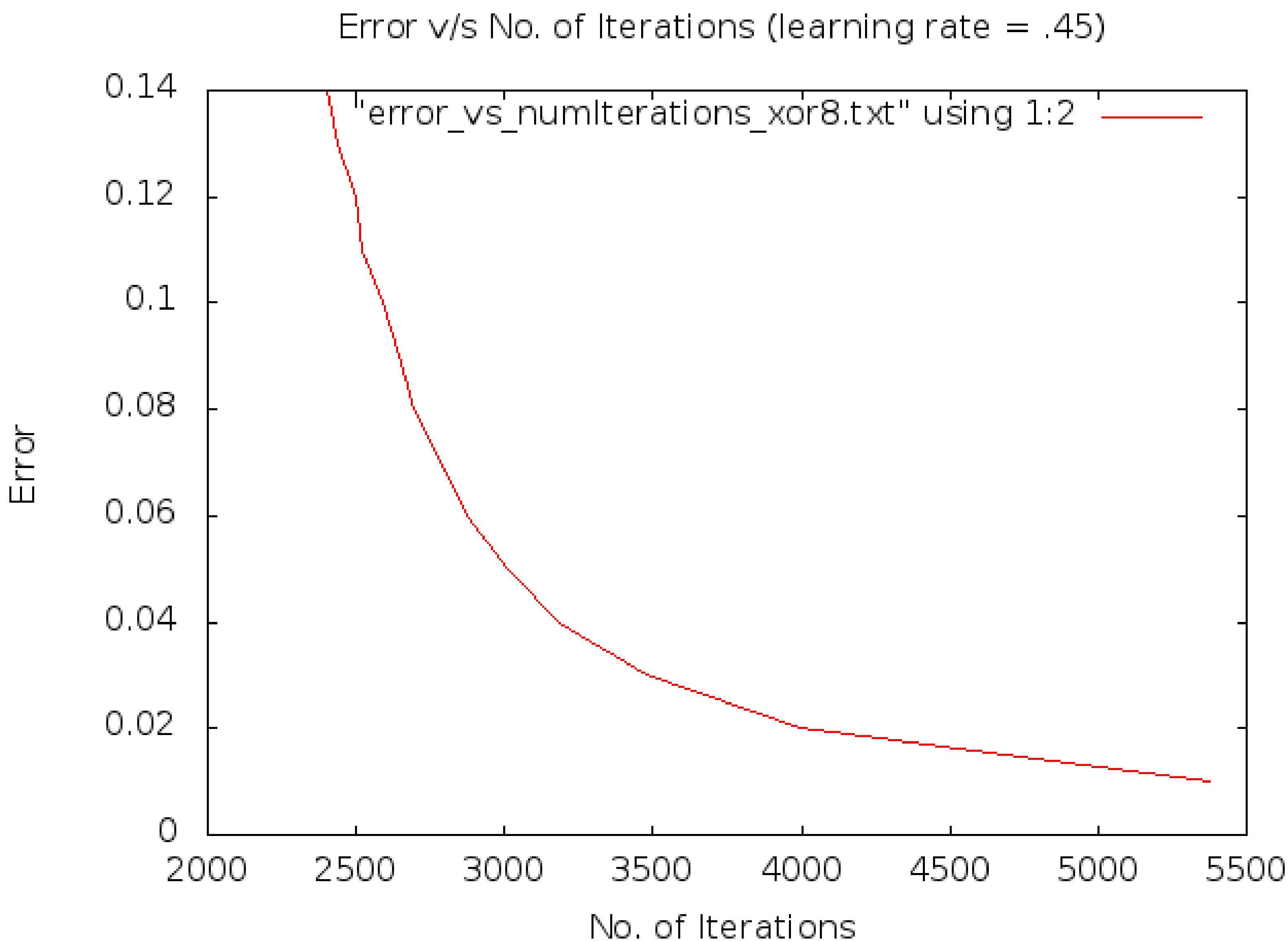
Effect of Learning Rate (XOR)



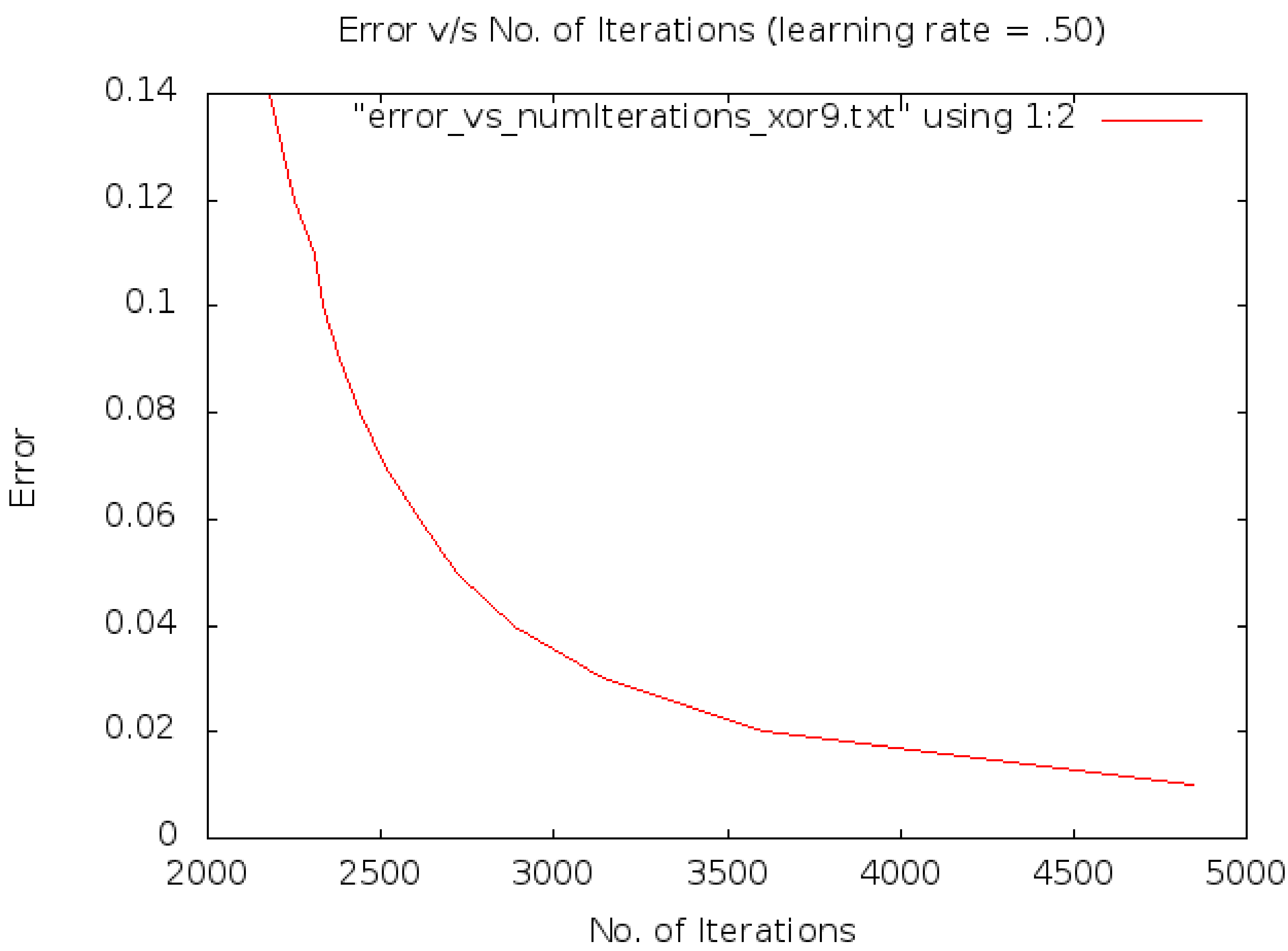
Effect of Learning Rate (XOR)



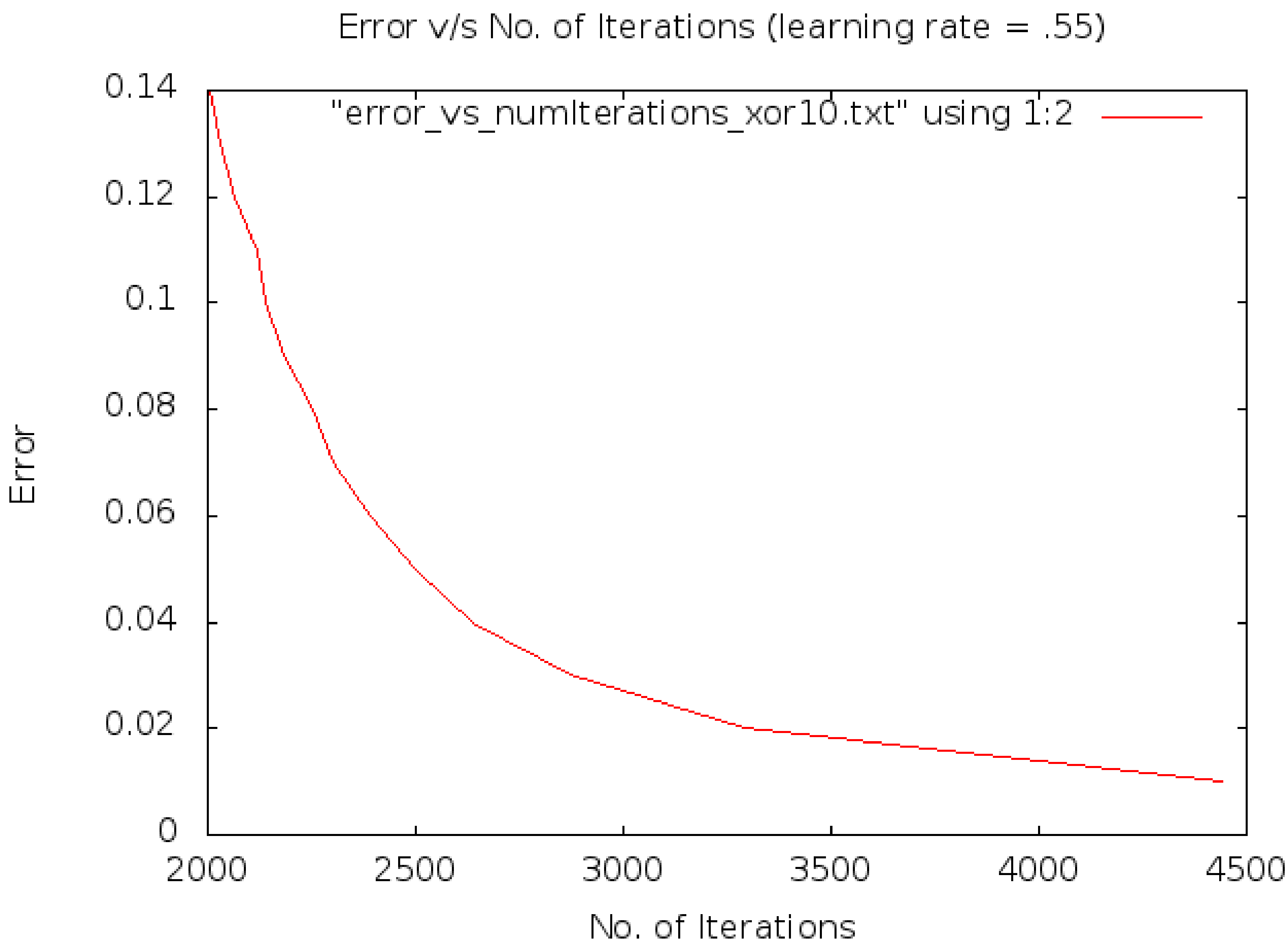
Effect of Learning Rate (XOR)



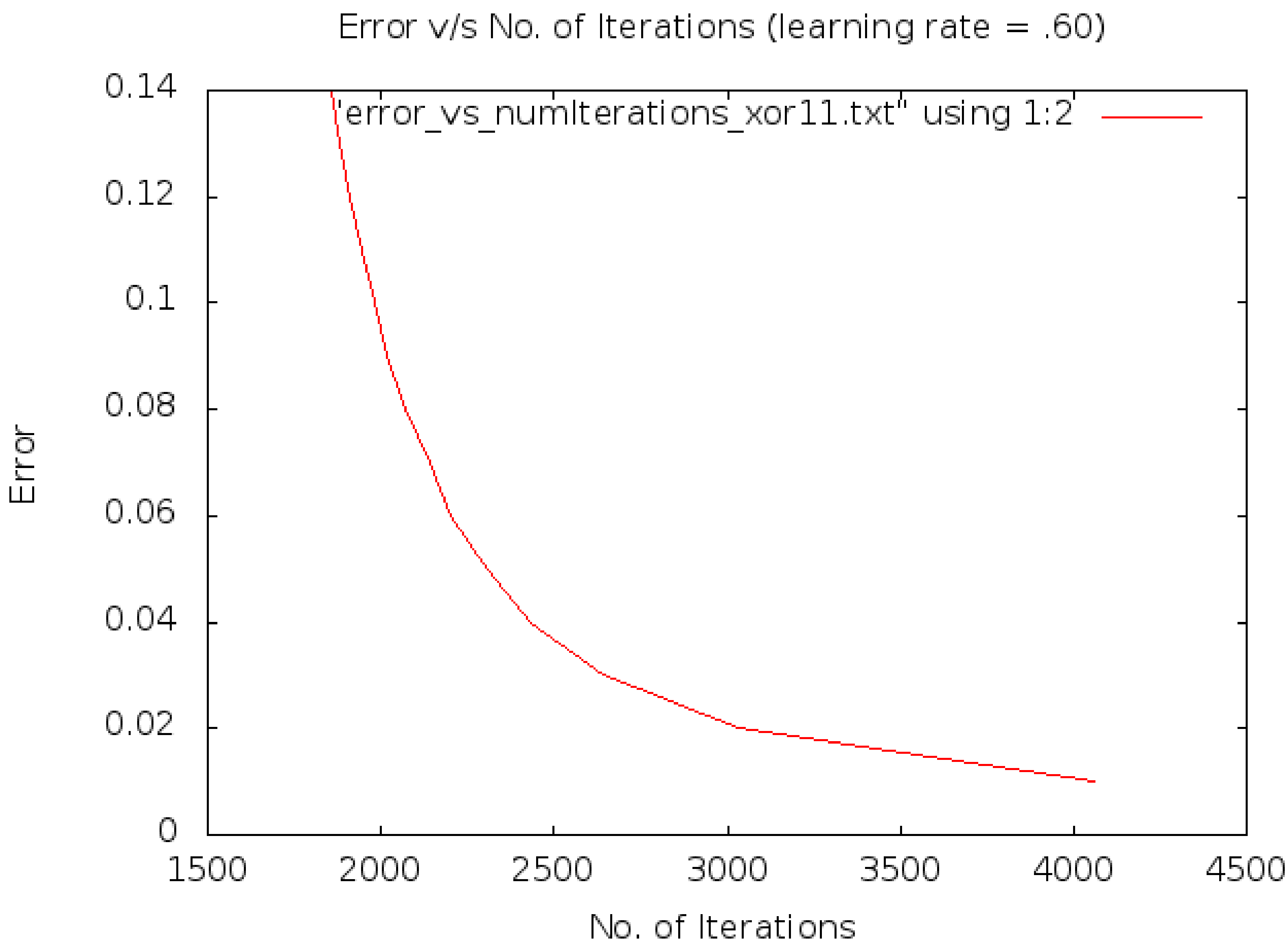
Effect of Learning Rate (XOR)



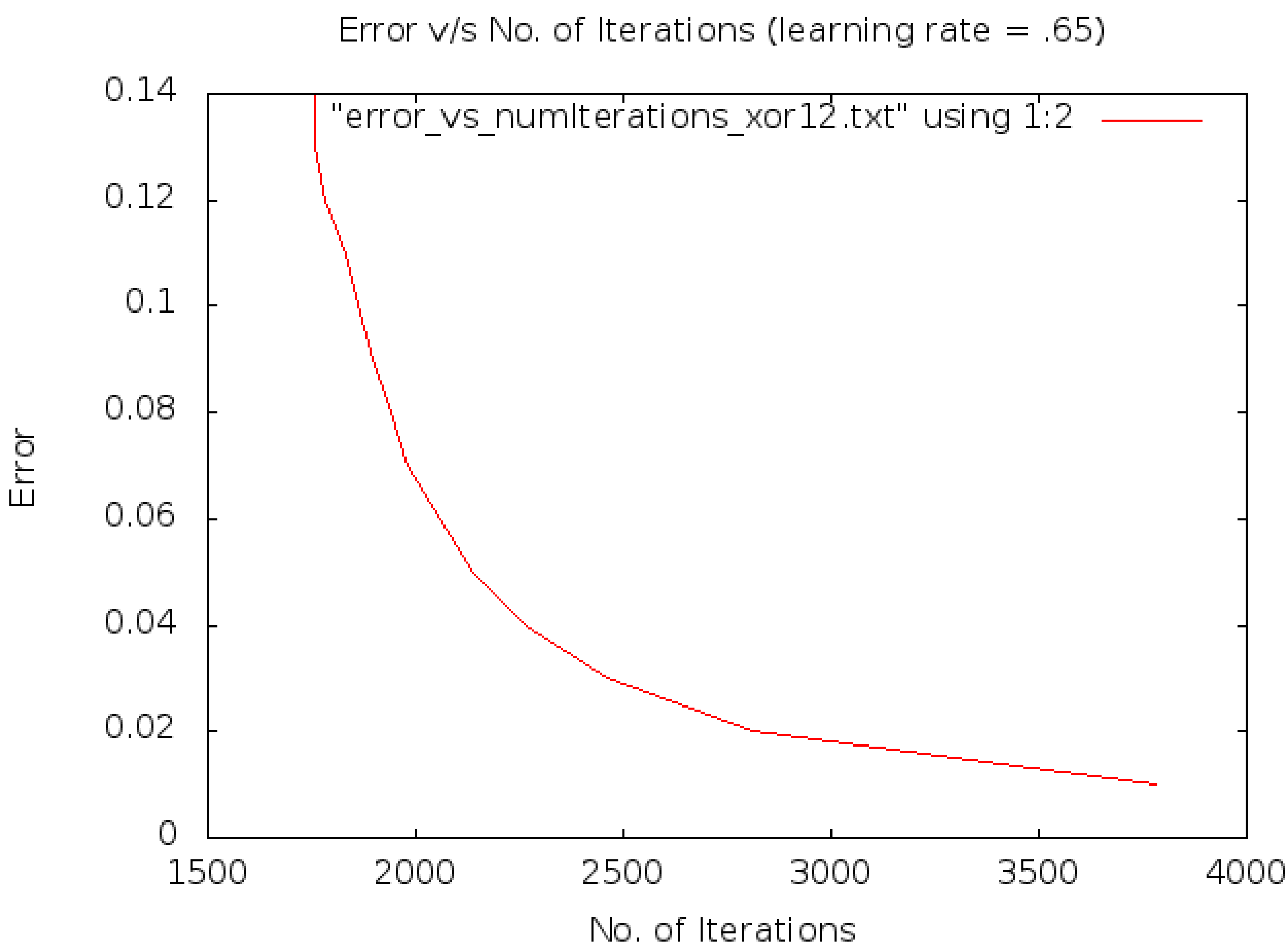
Effect of Learning Rate (XOR)



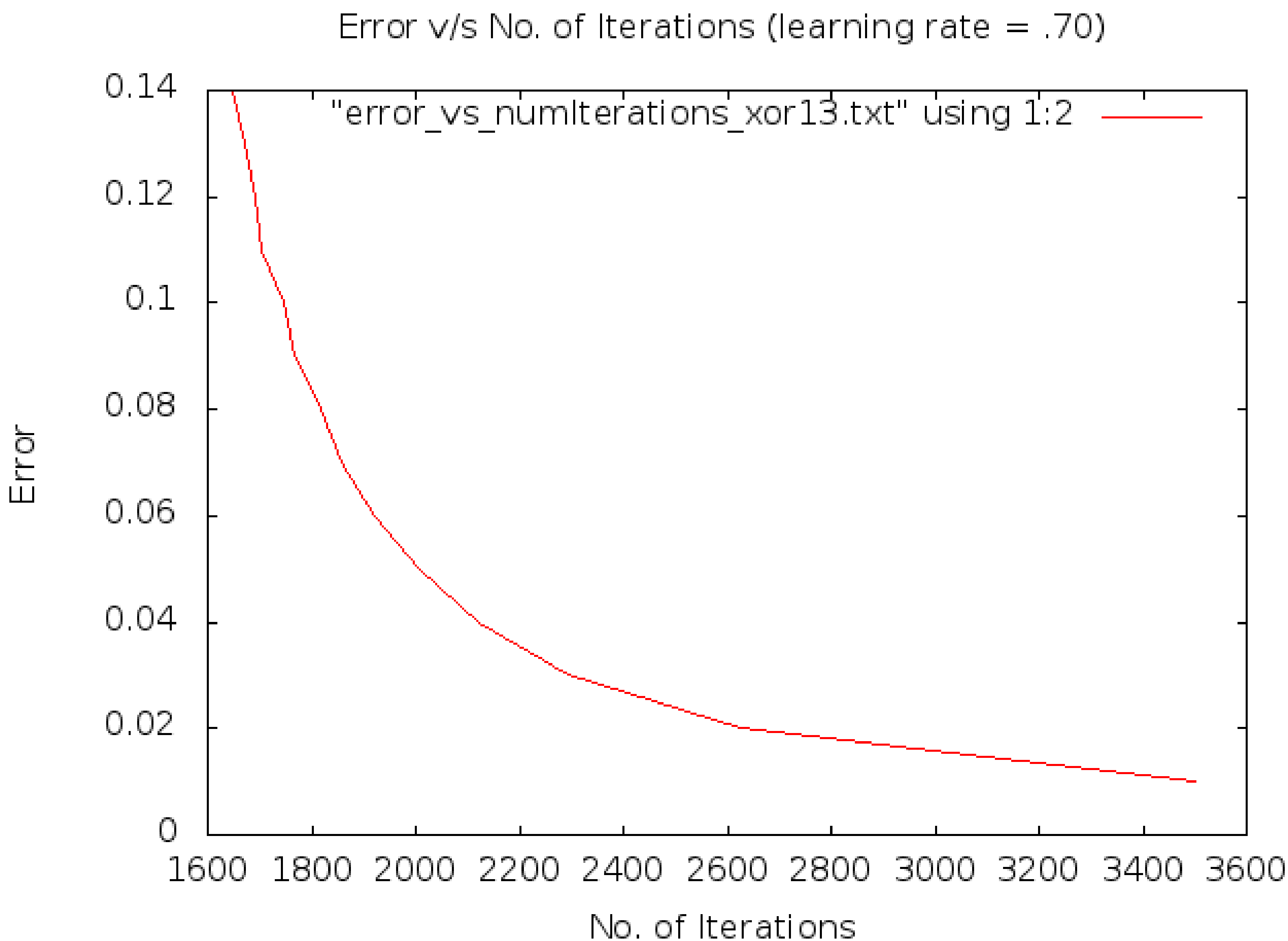
Effect of Learning Rate (XOR)



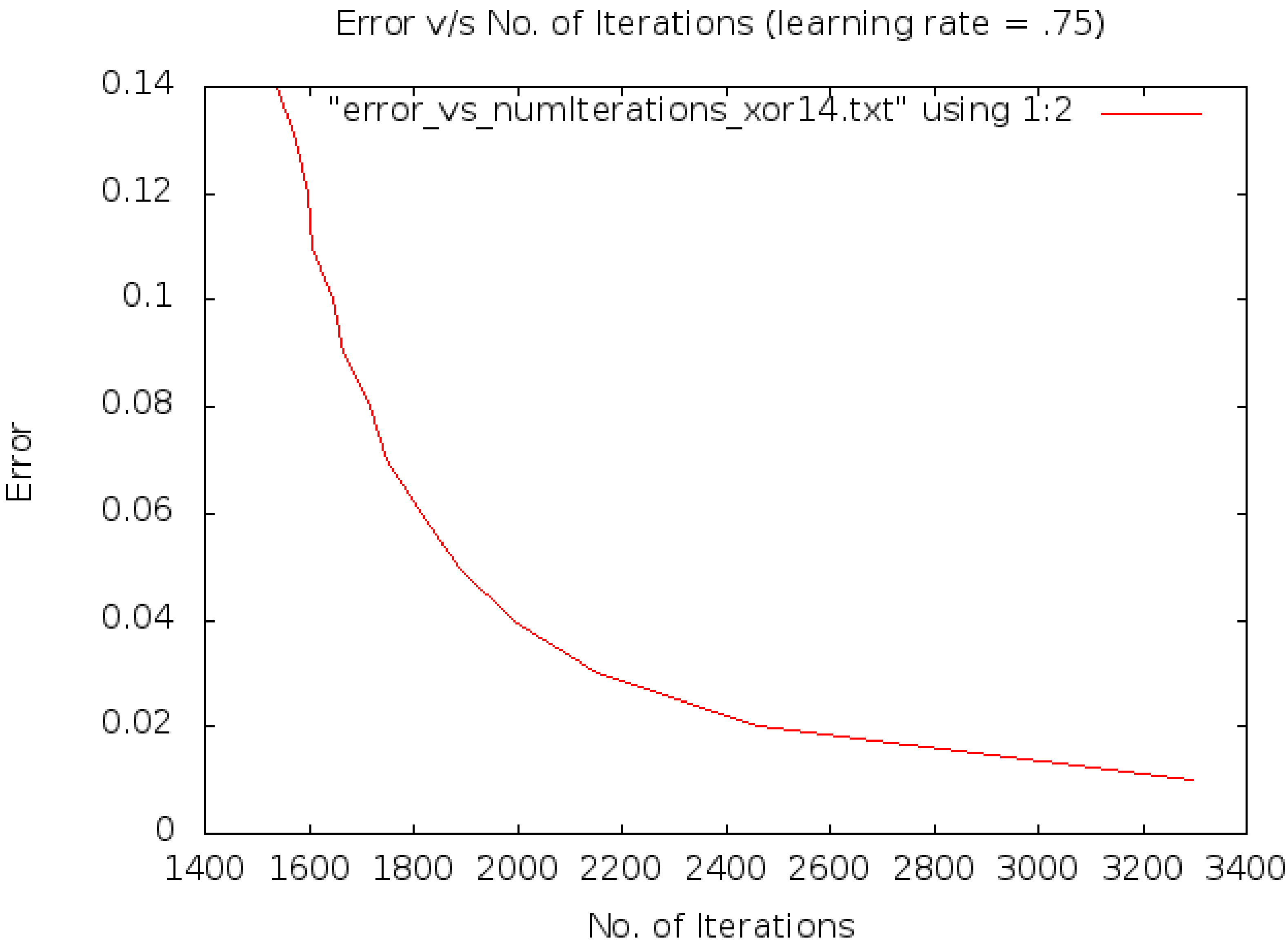
Effect of Learning Rate (XOR)



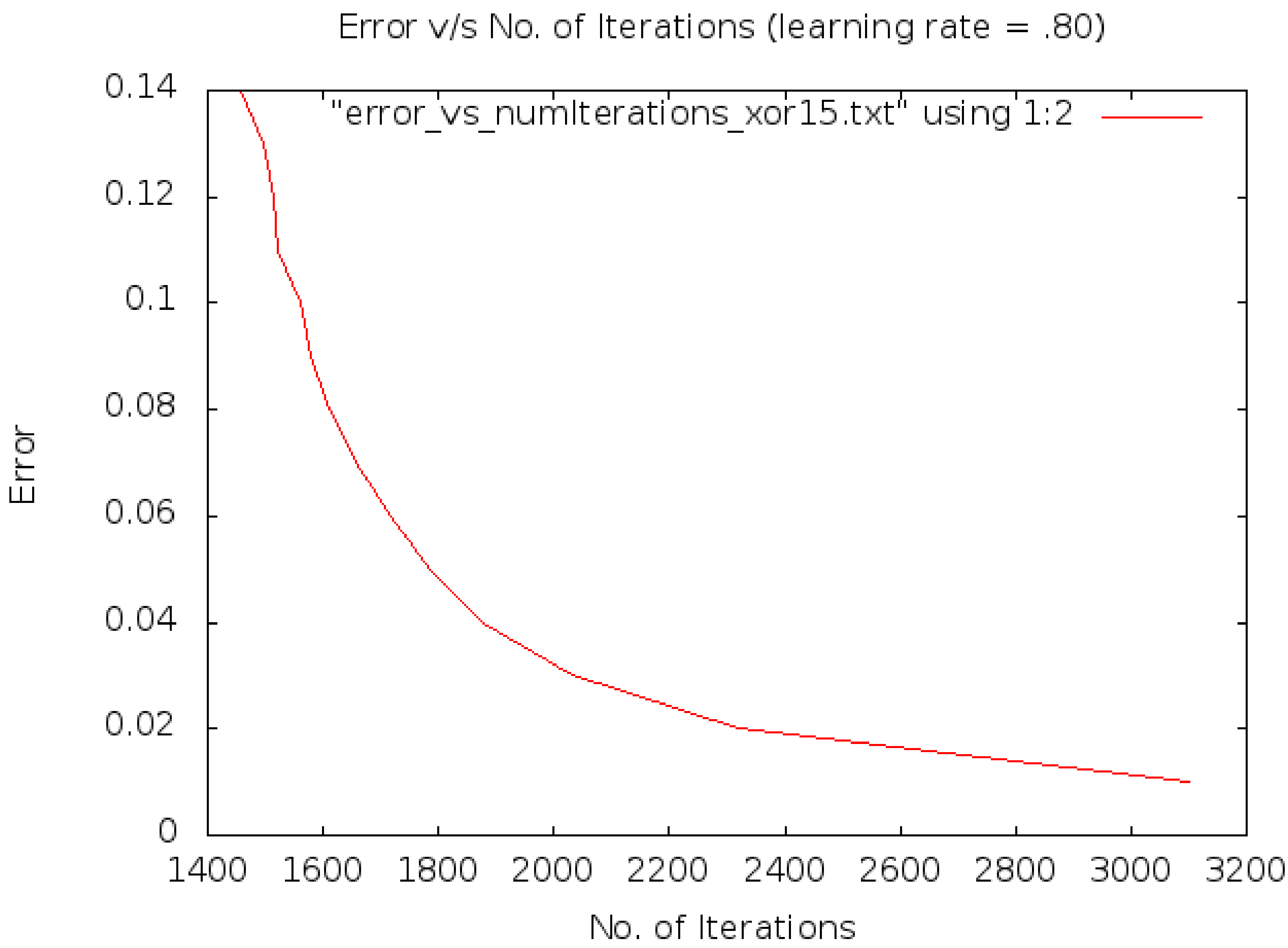
Effect of Learning Rate (XOR)



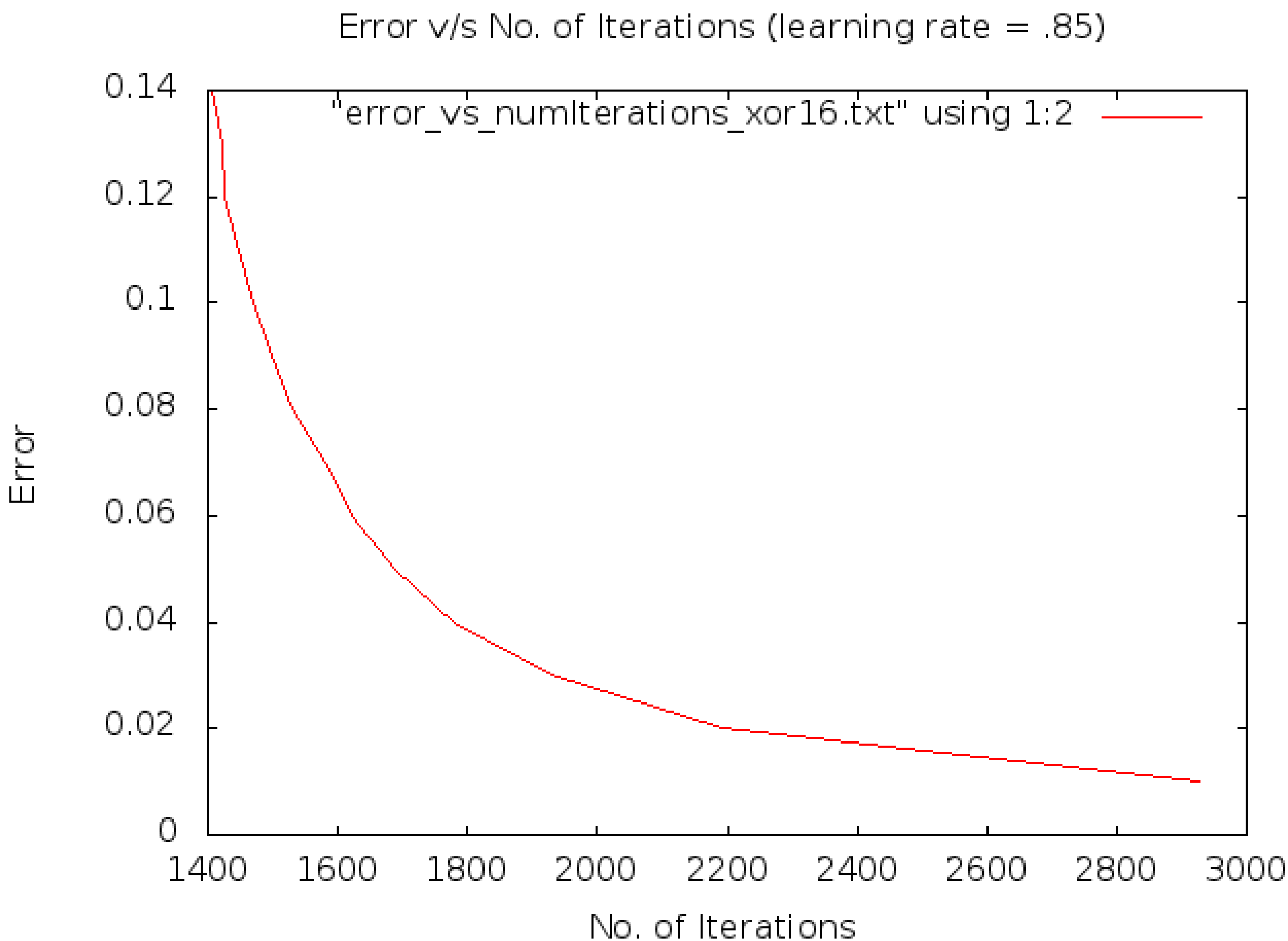
Effect of Learning Rate (XOR)



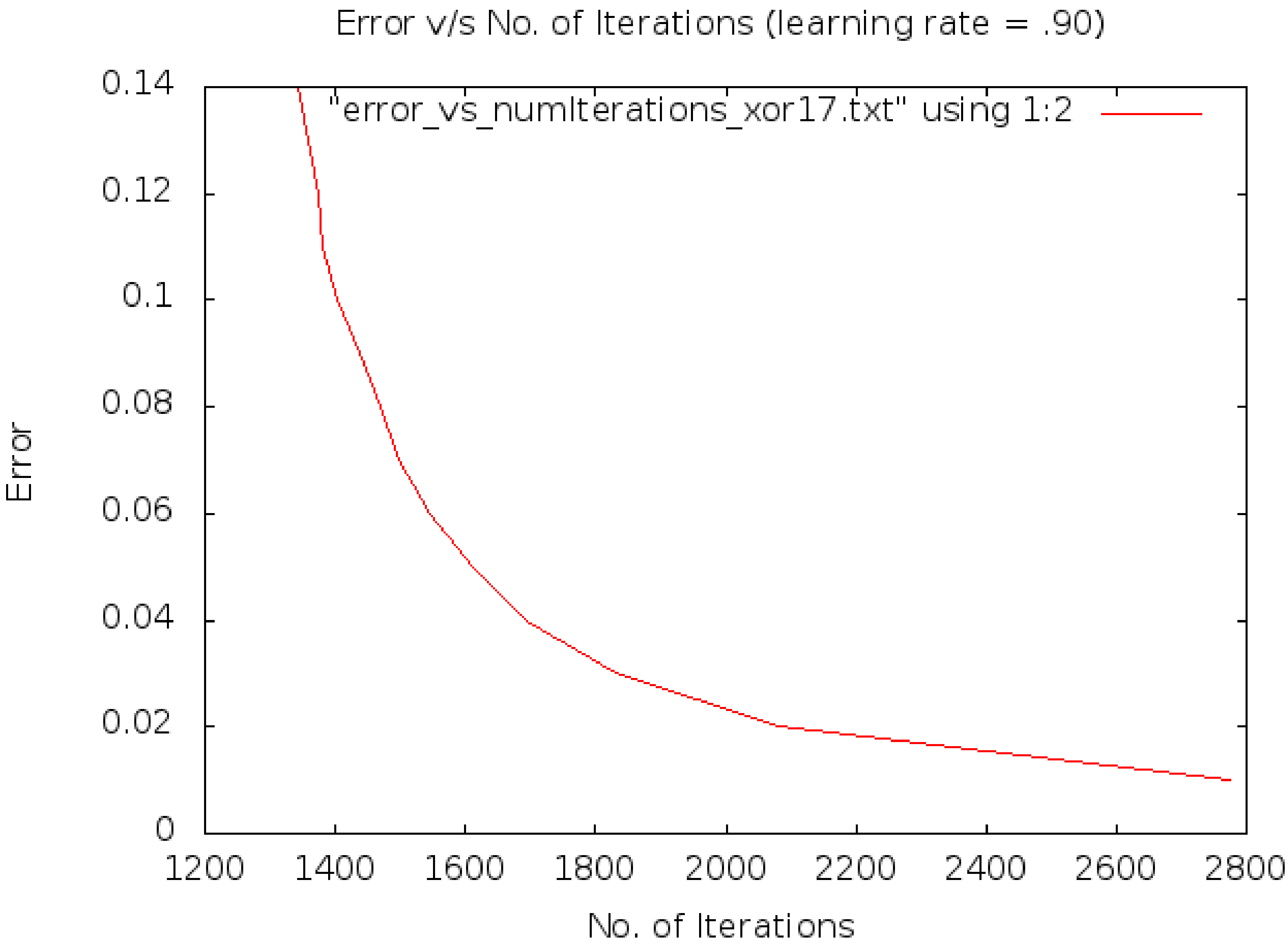
Effect of Learning Rate (XOR)



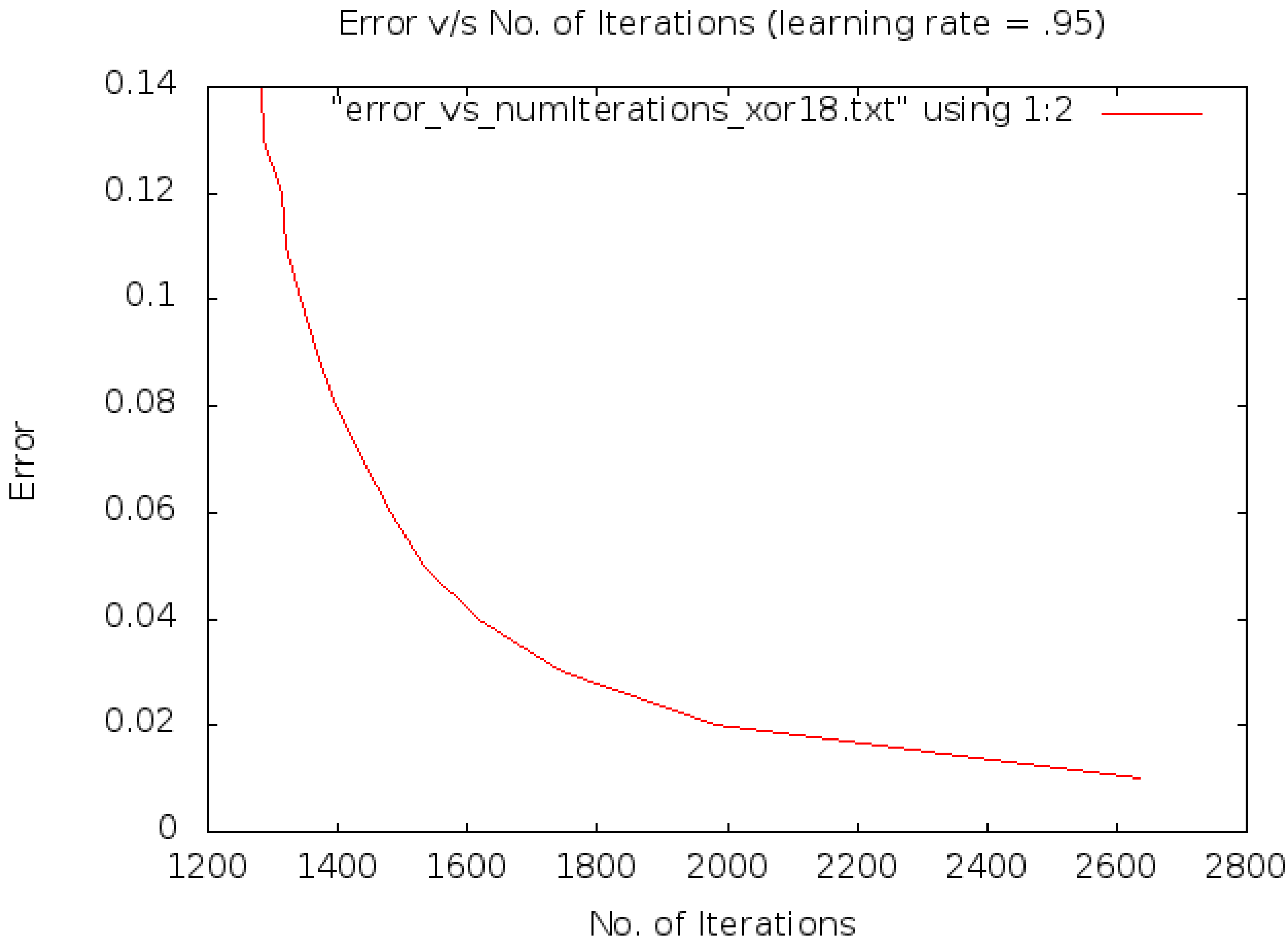
Effect of Learning Rate (XOR)



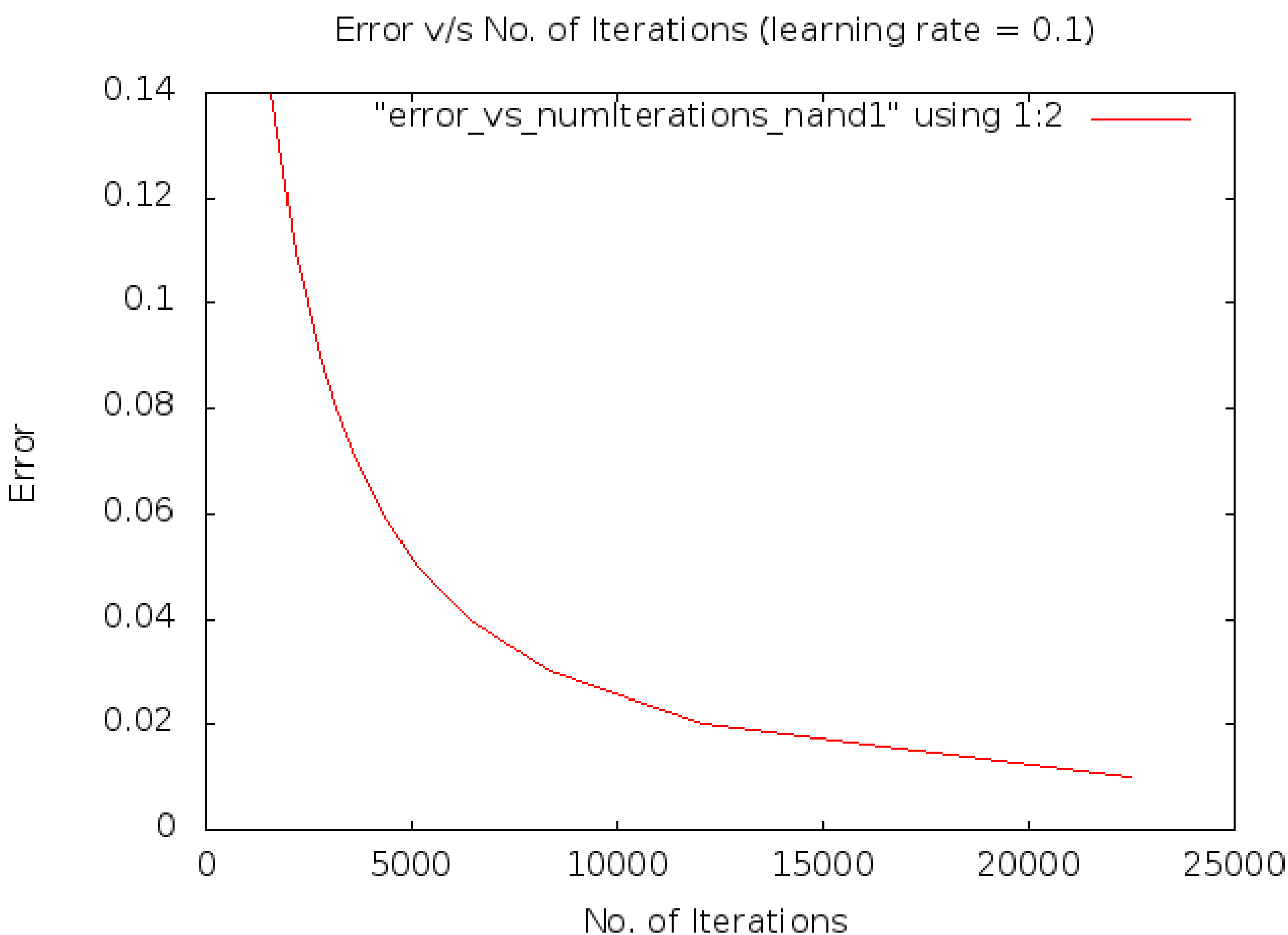
Effect of Learning Rate (XOR)



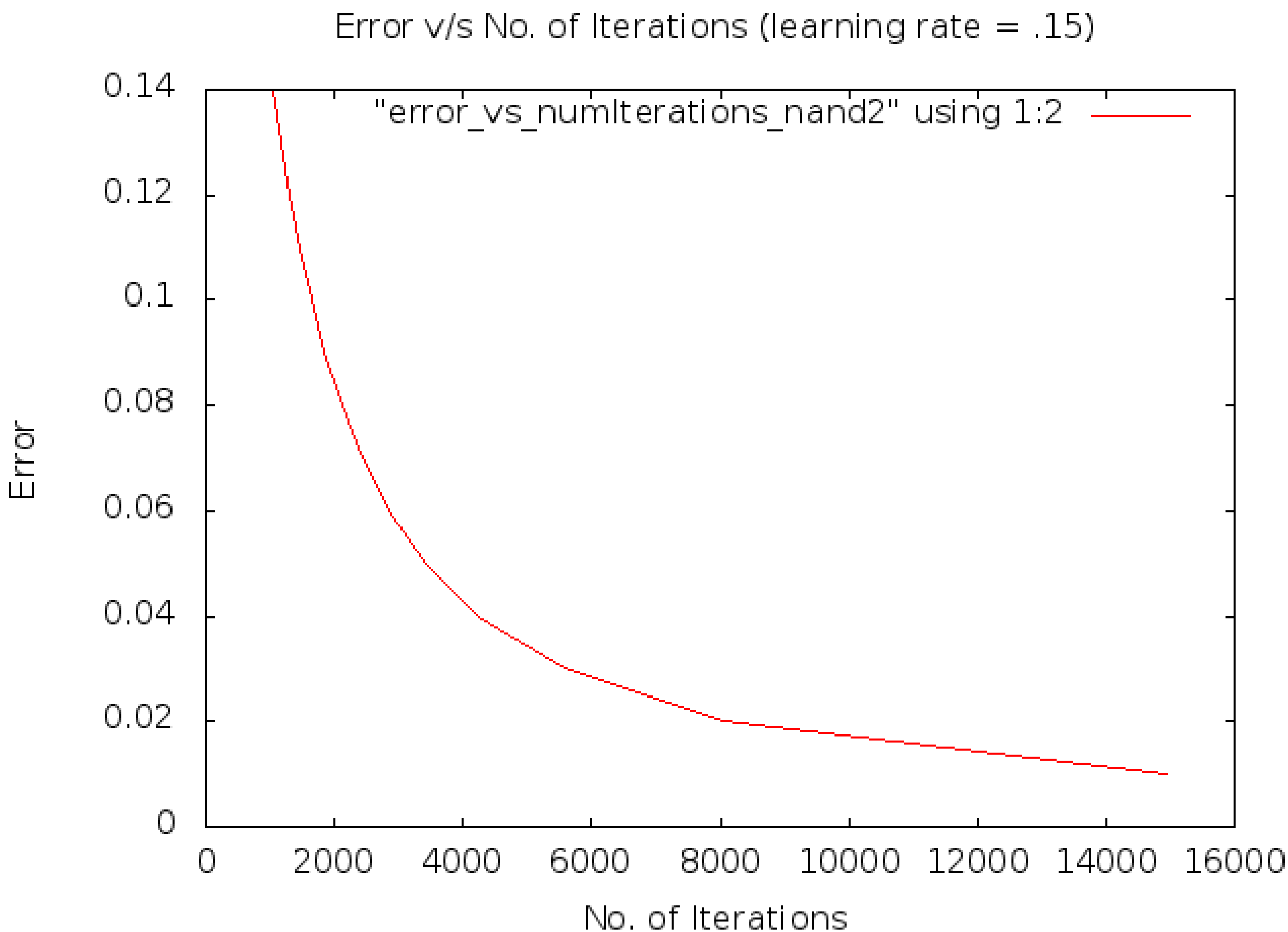
Effect of Learning Rate (XOR)



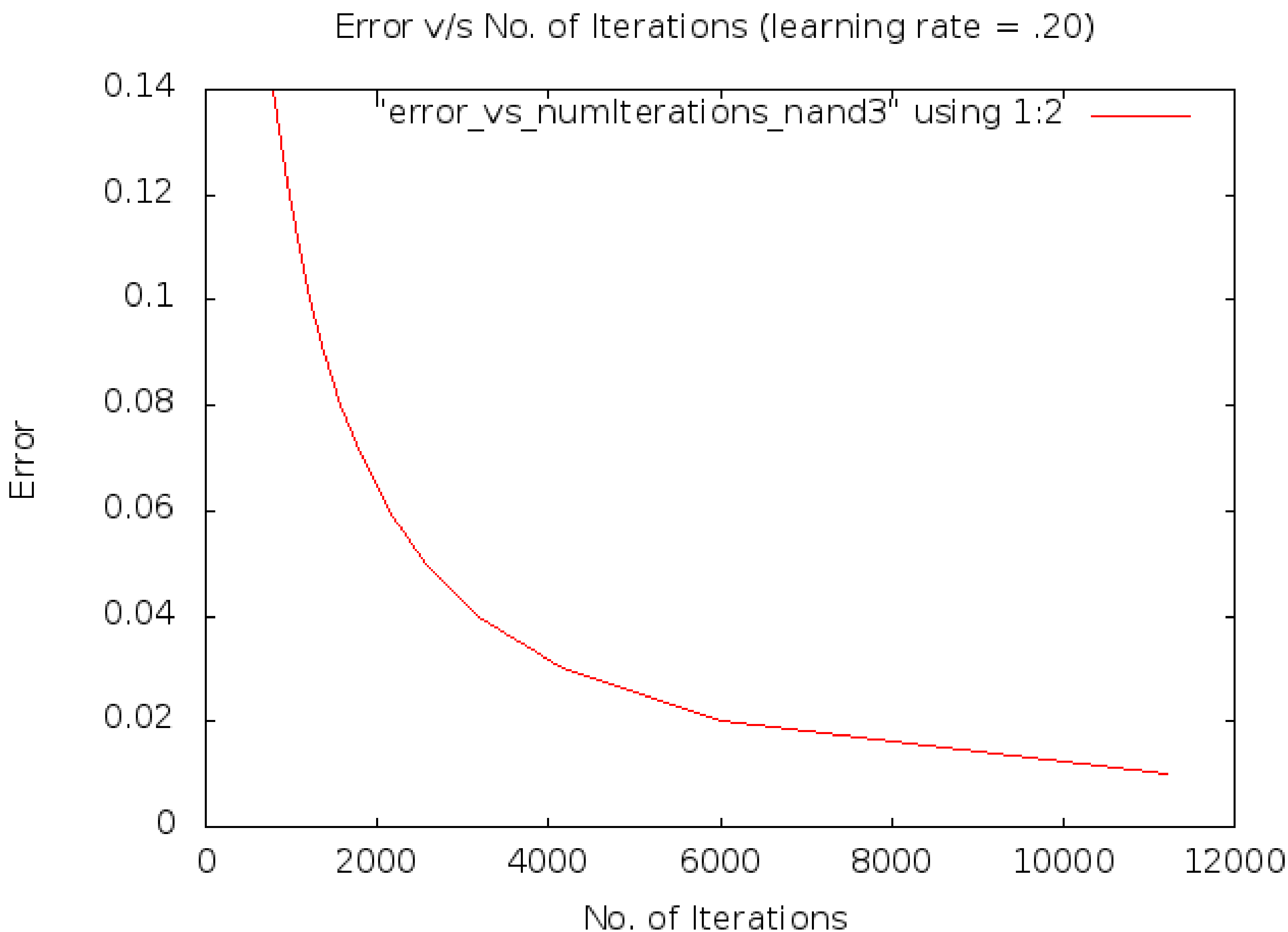
Effect of Learning Rate (NAND)



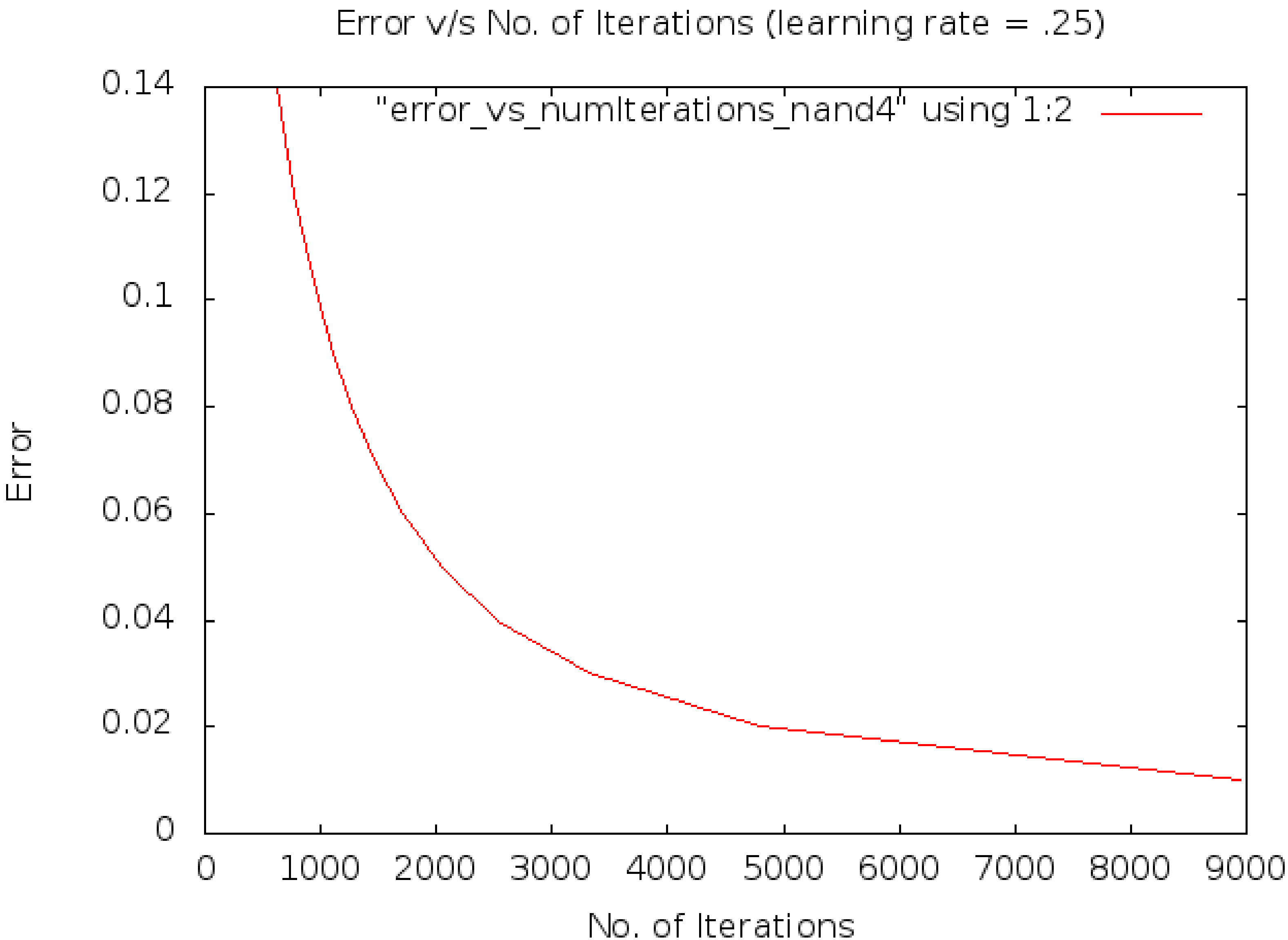
Effect of Learning Rate (NAND)



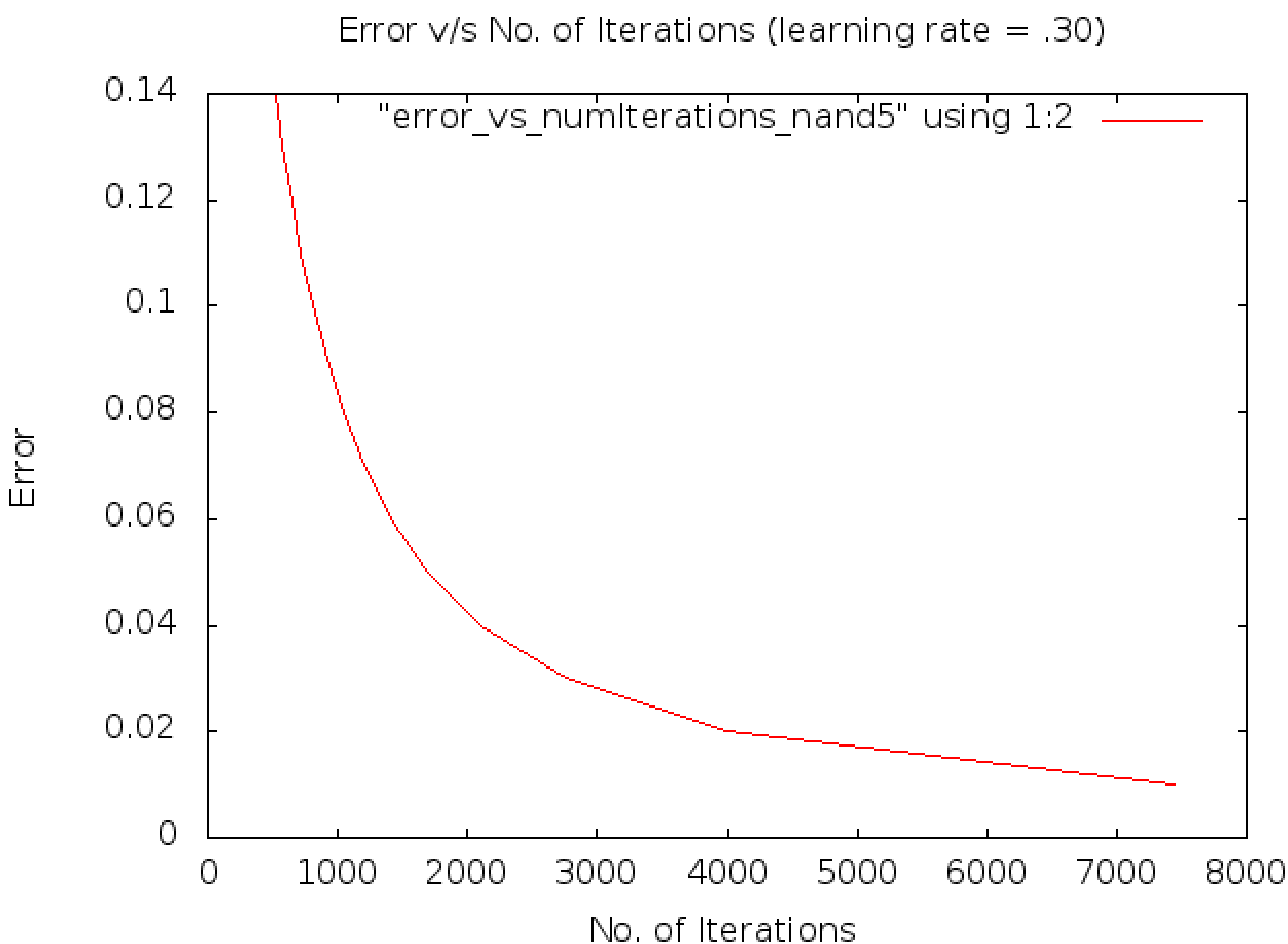
Effect of Learning Rate (NAND)



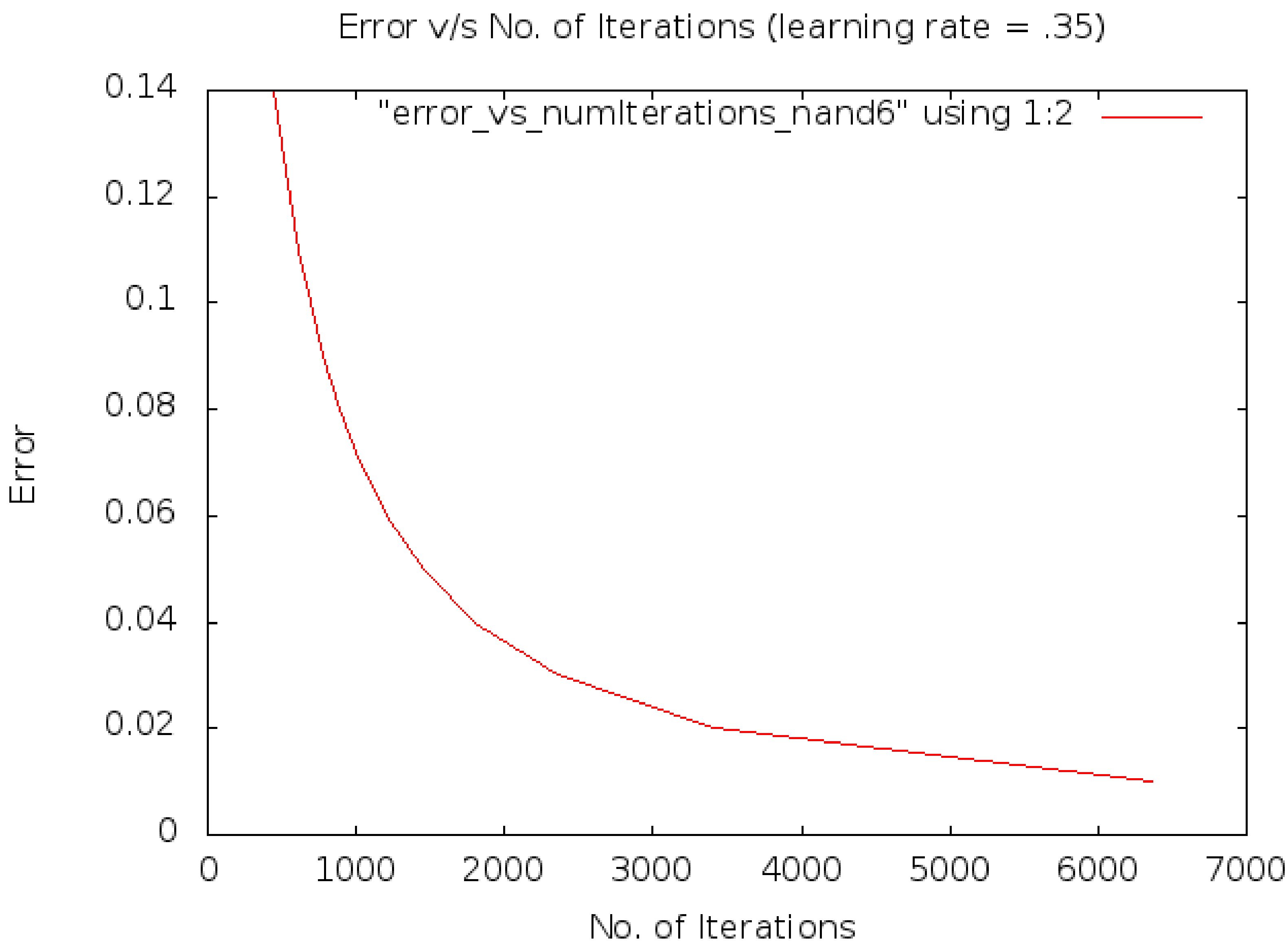
Effect of Learning Rate (NAND)



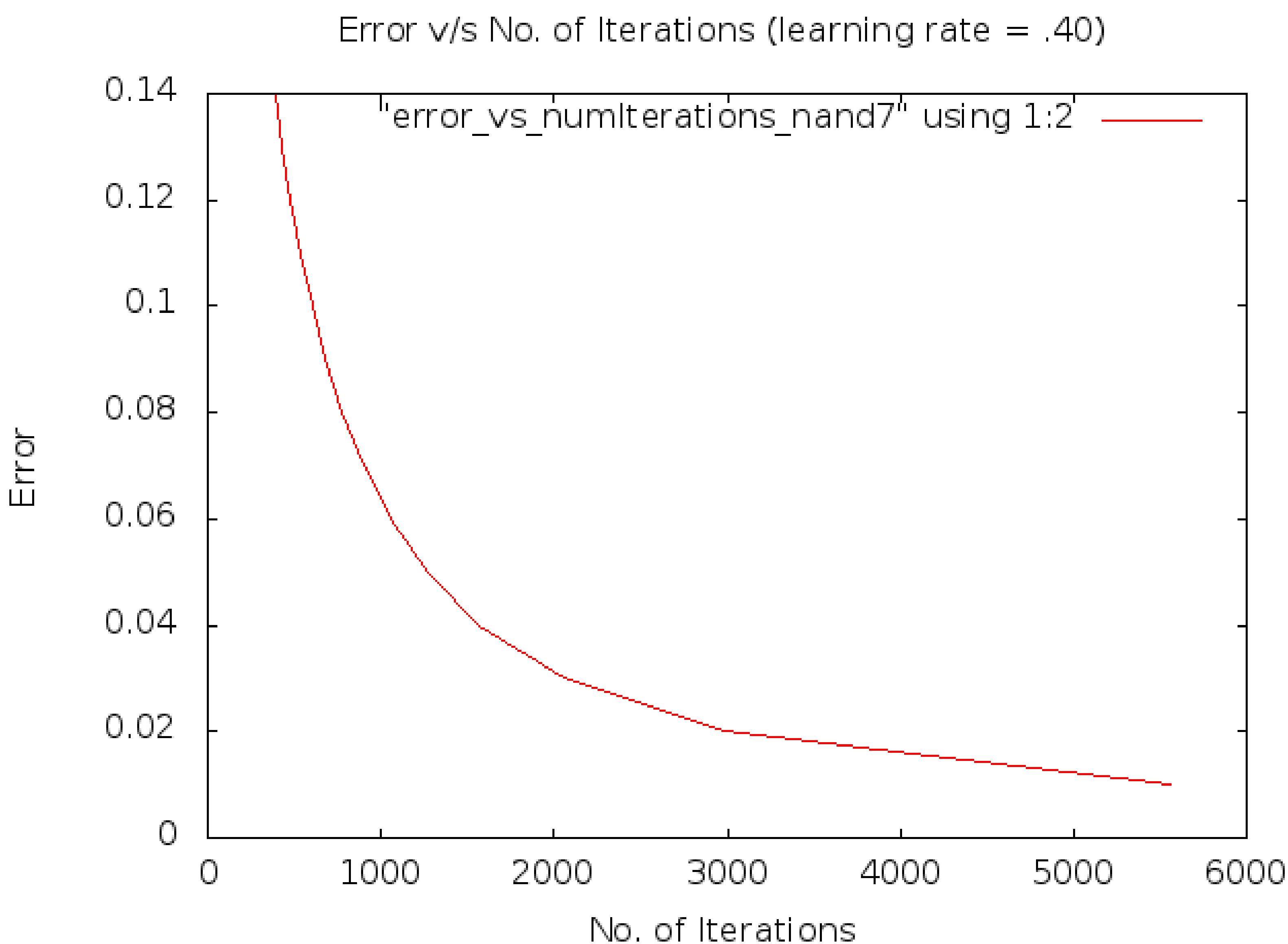
Effect of Learning Rate (NAND)



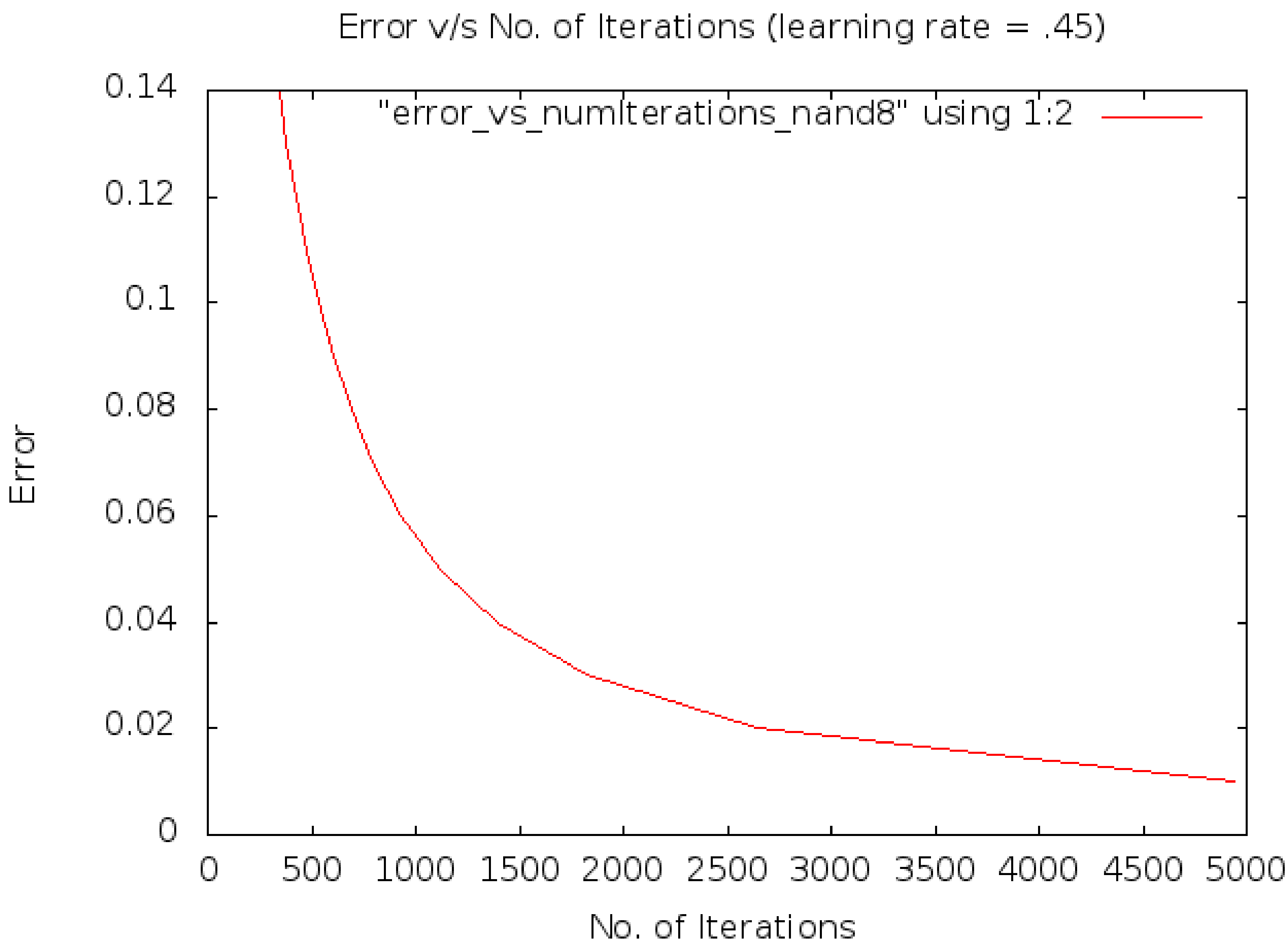
Effect of Learning Rate (NAND)



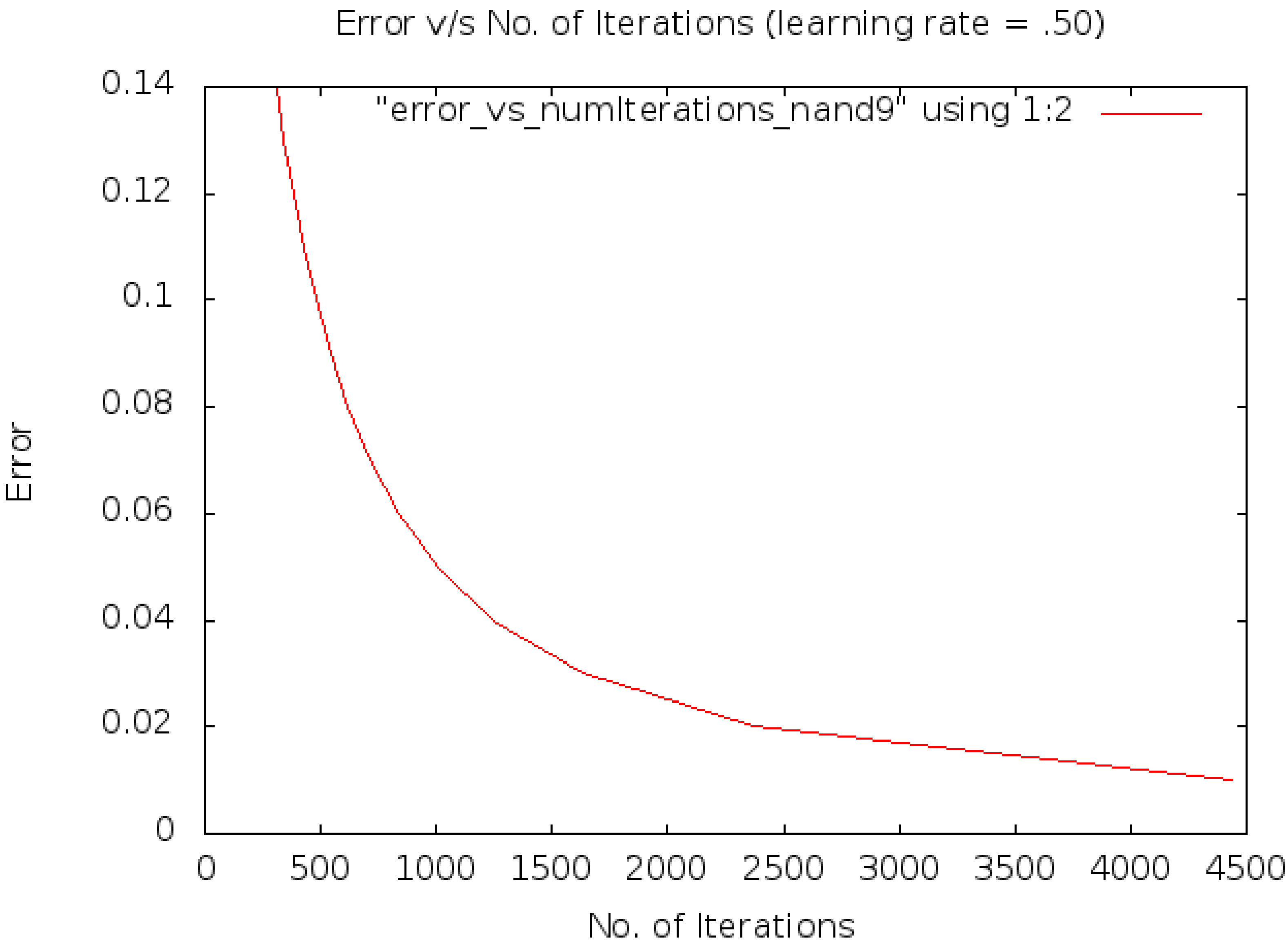
Effect of Learning Rate (NAND)



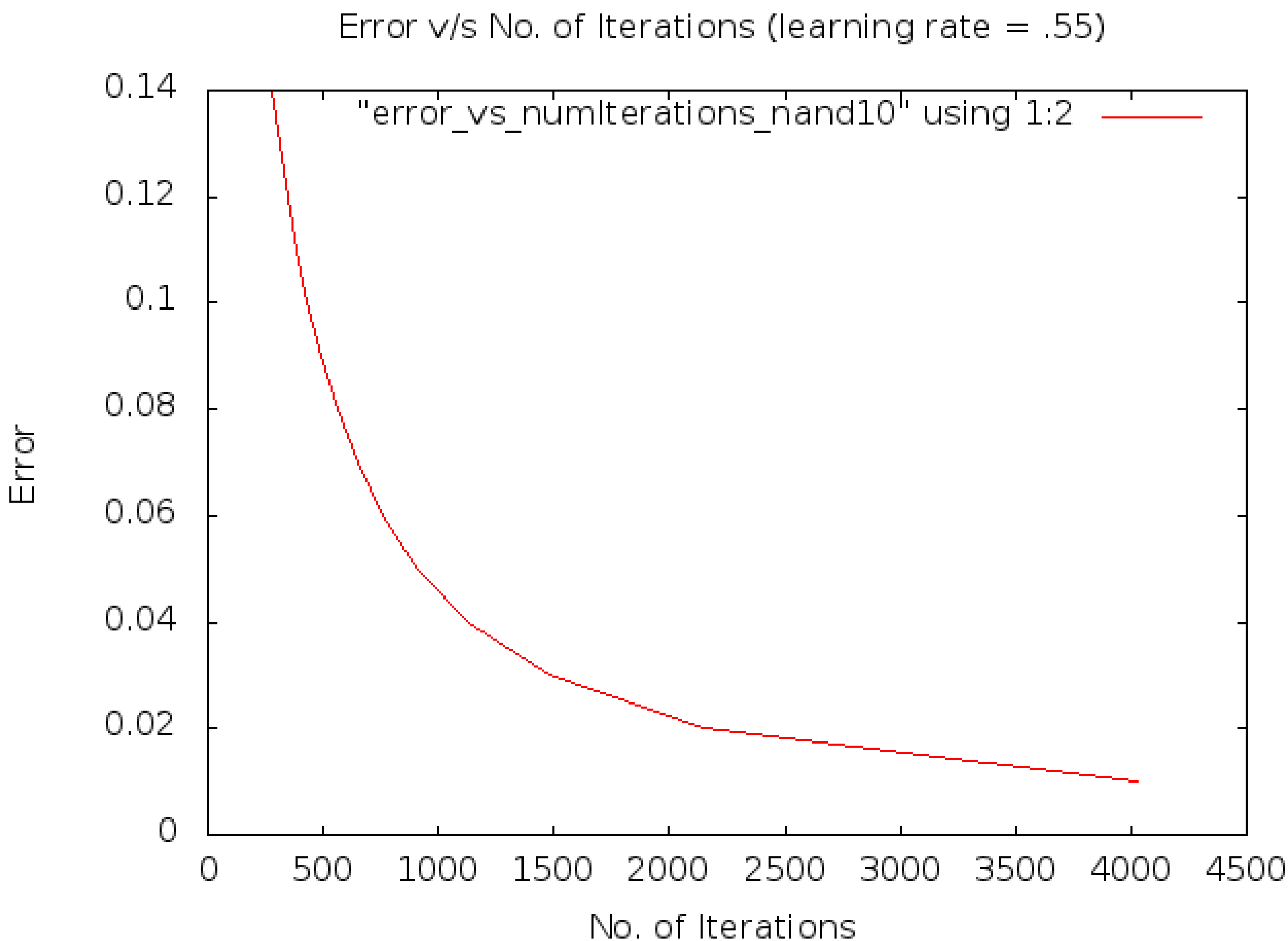
Effect of Learning Rate (NAND)



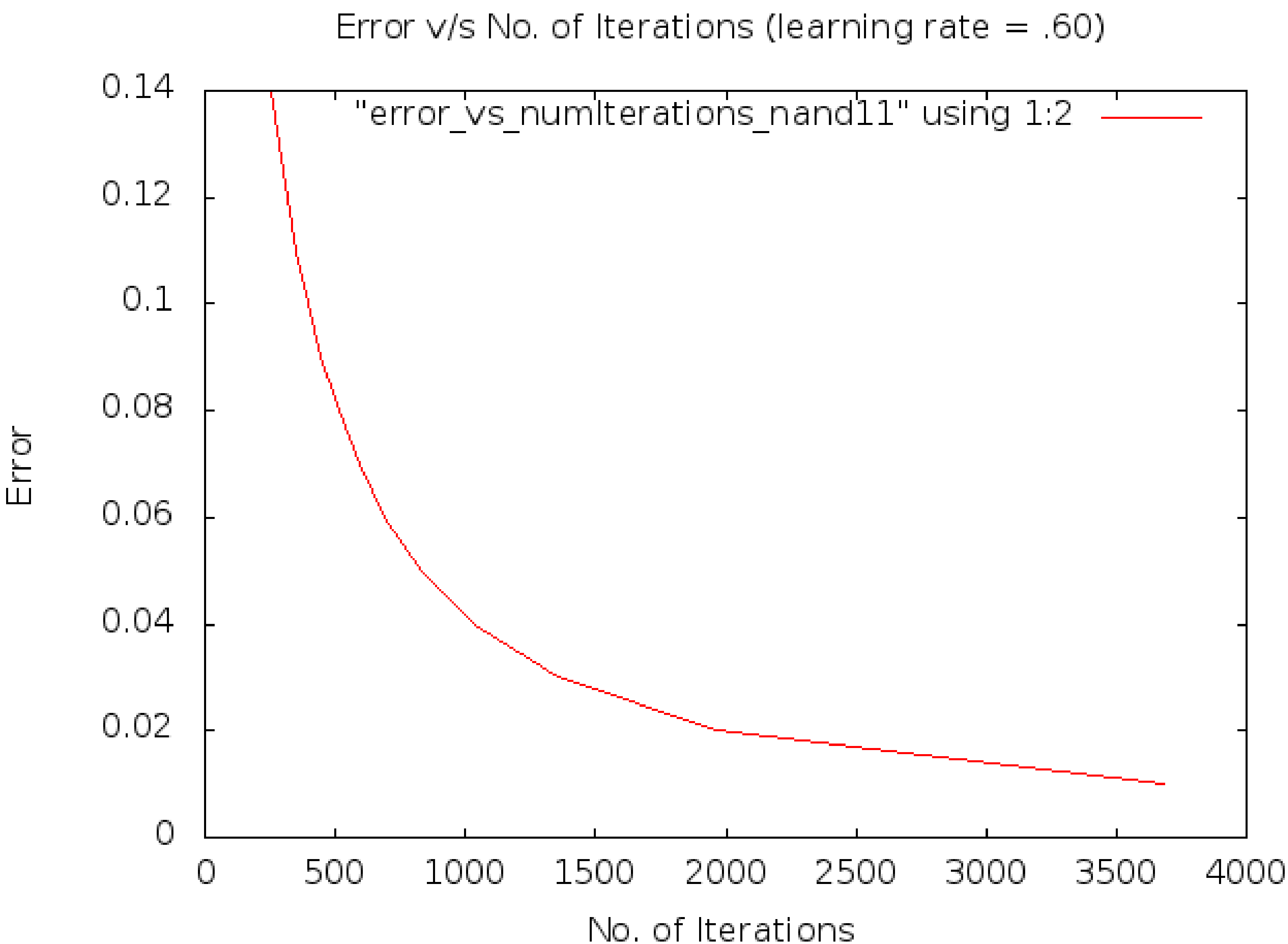
Effect of Learning Rate (NAND)



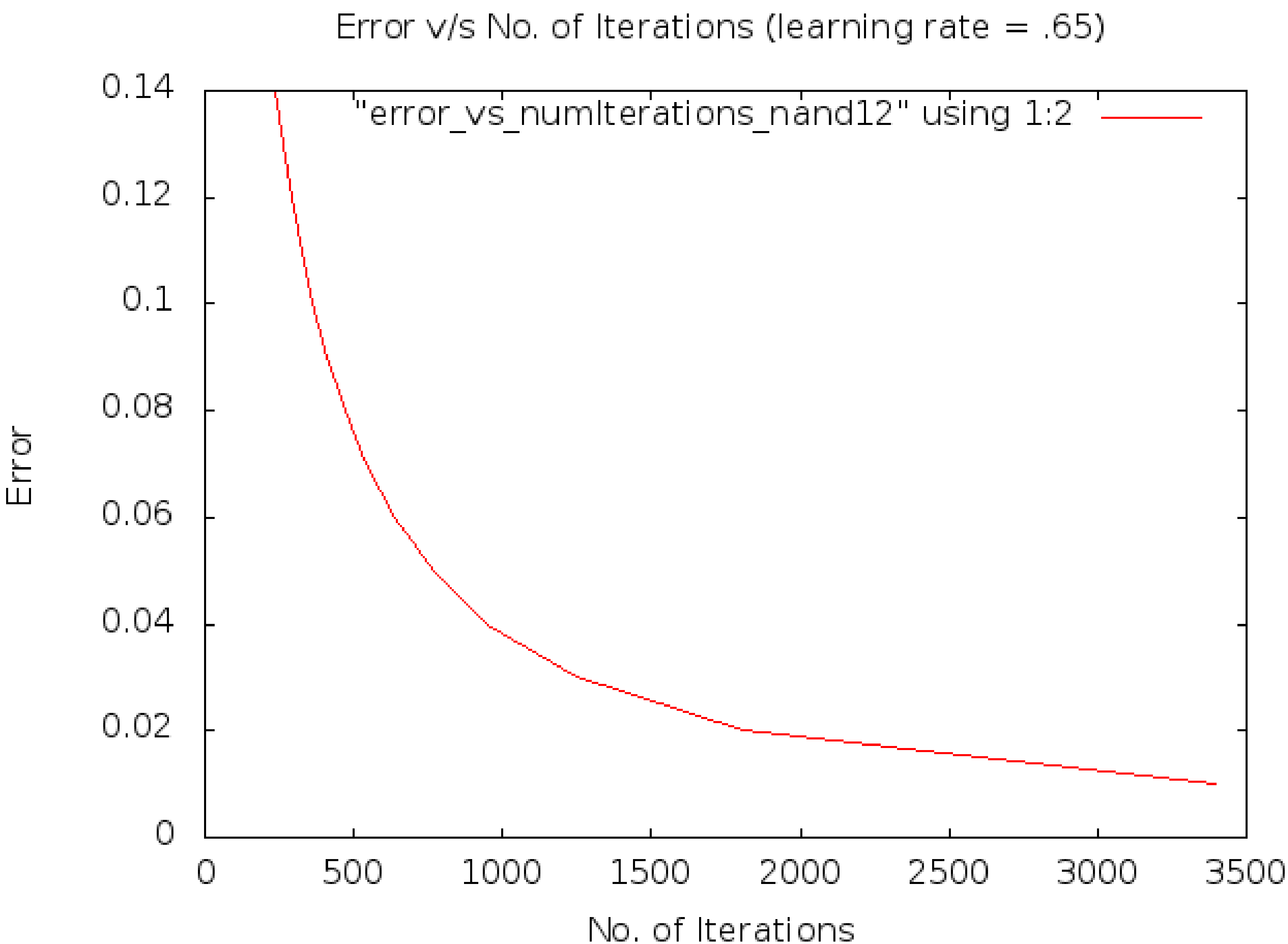
Effect of Learning Rate (NAND)



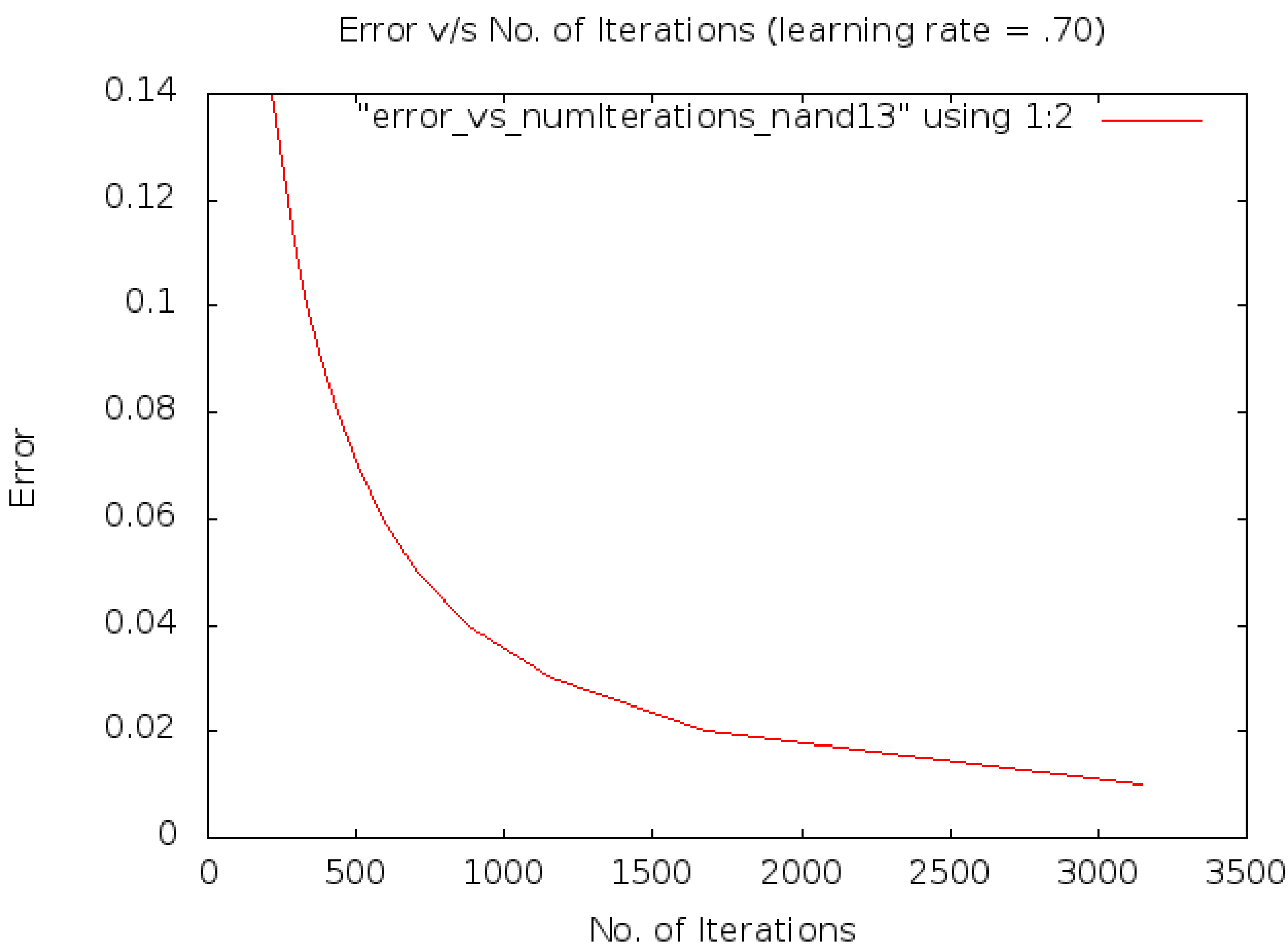
Effect of Learning Rate (NAND)



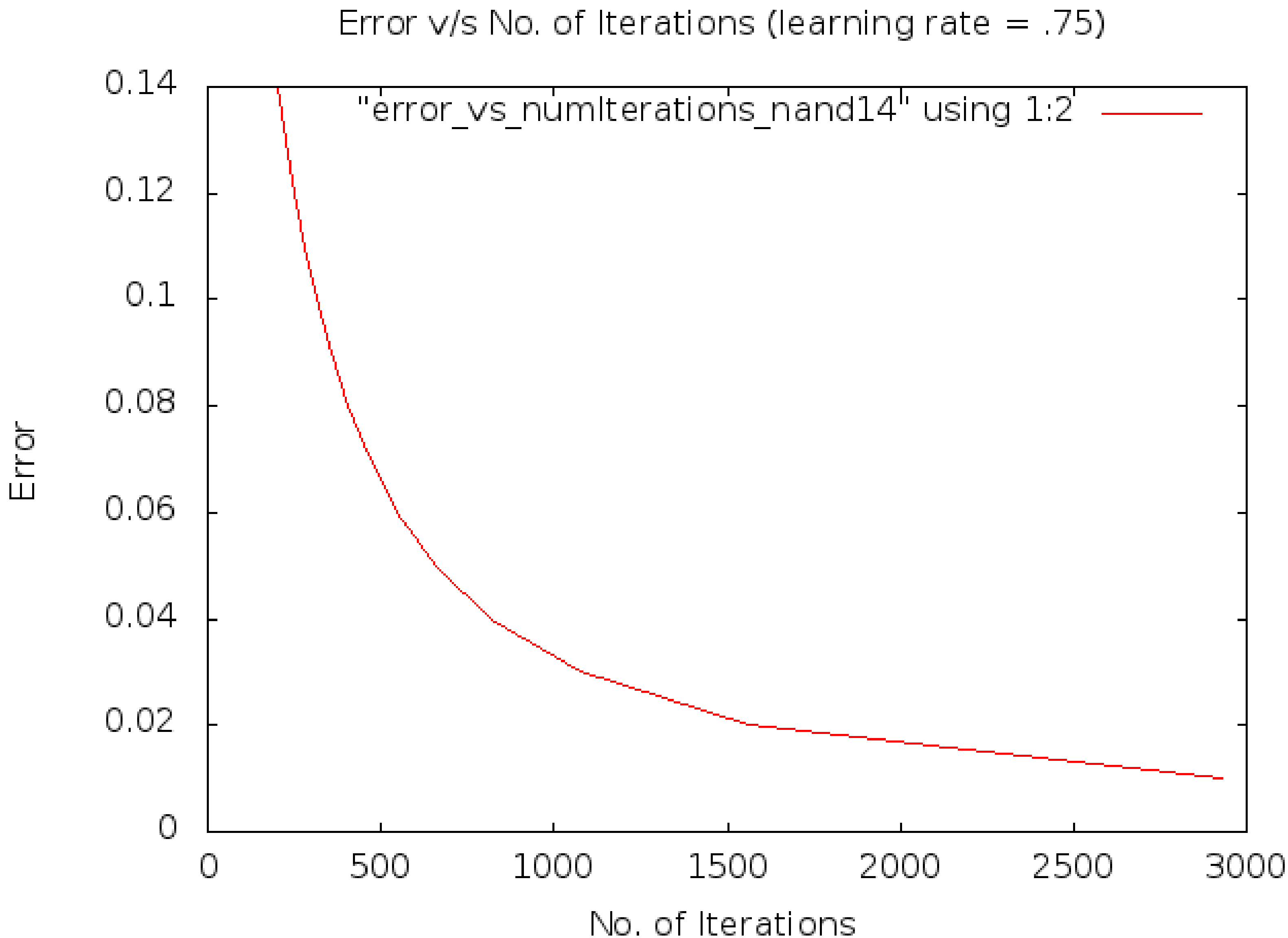
Effect of Learning Rate (NAND)



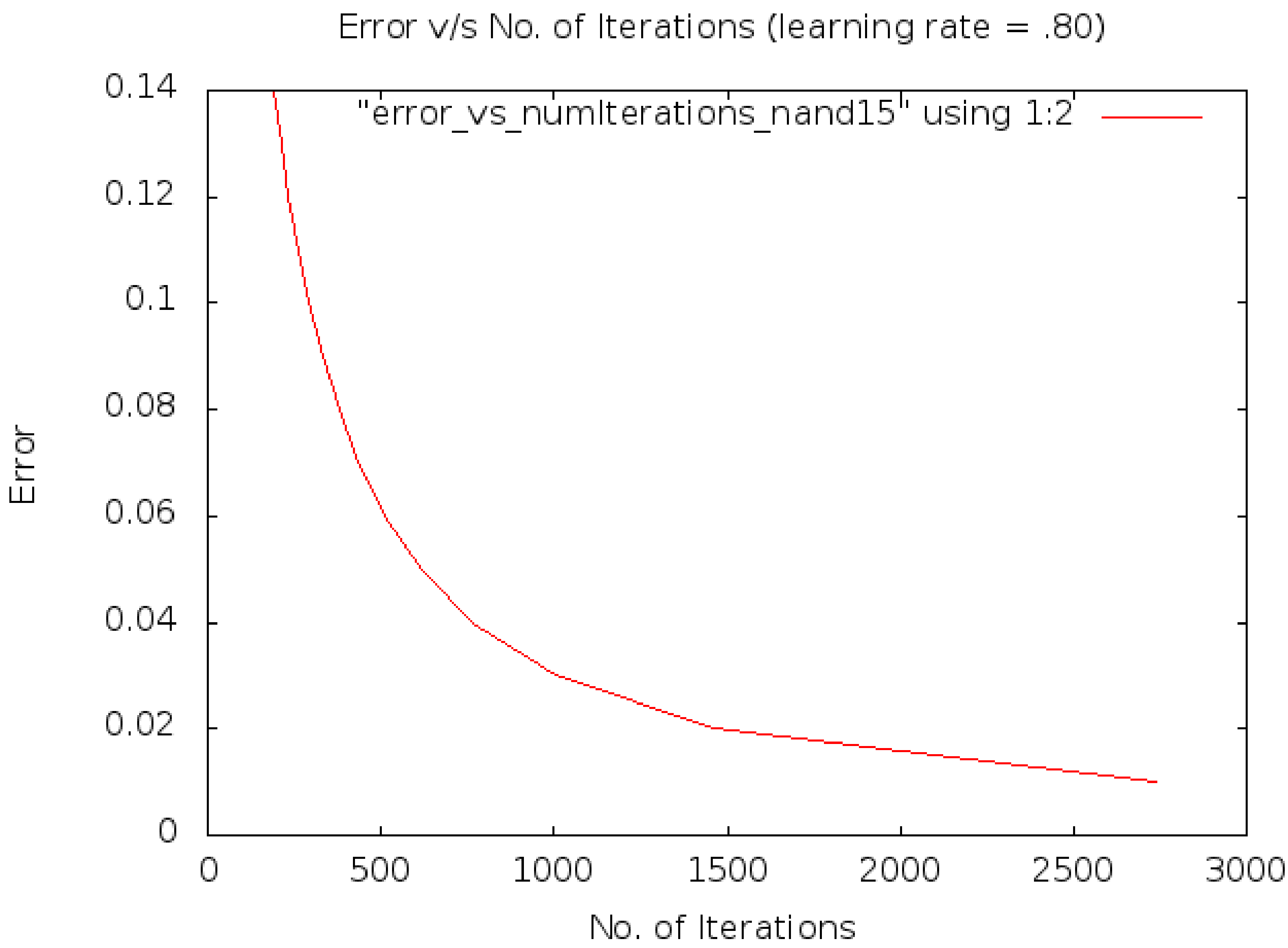
Effect of Learning Rate (NAND)



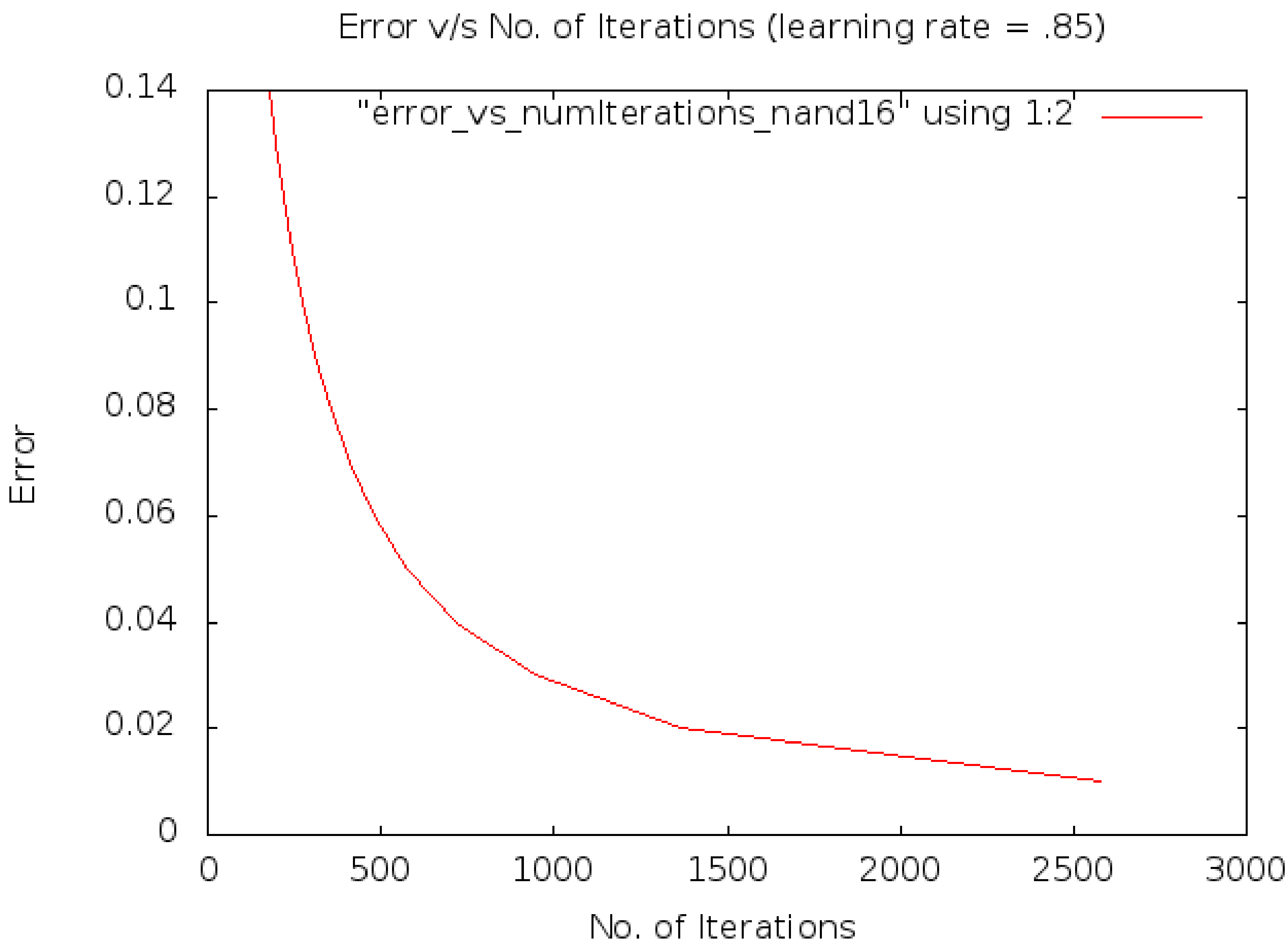
Effect of Learning Rate (NAND)



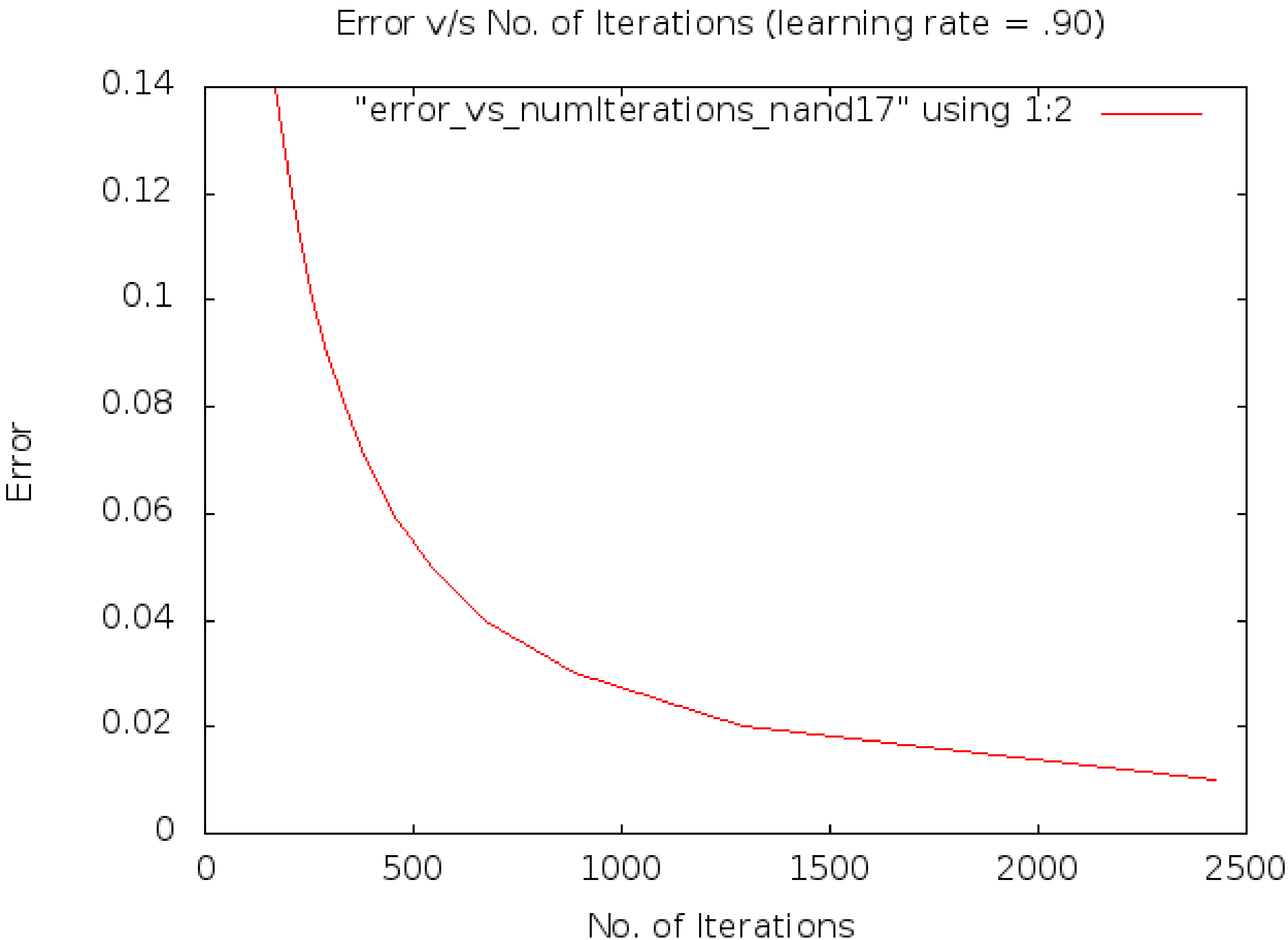
Effect of Learning Rate (NAND)



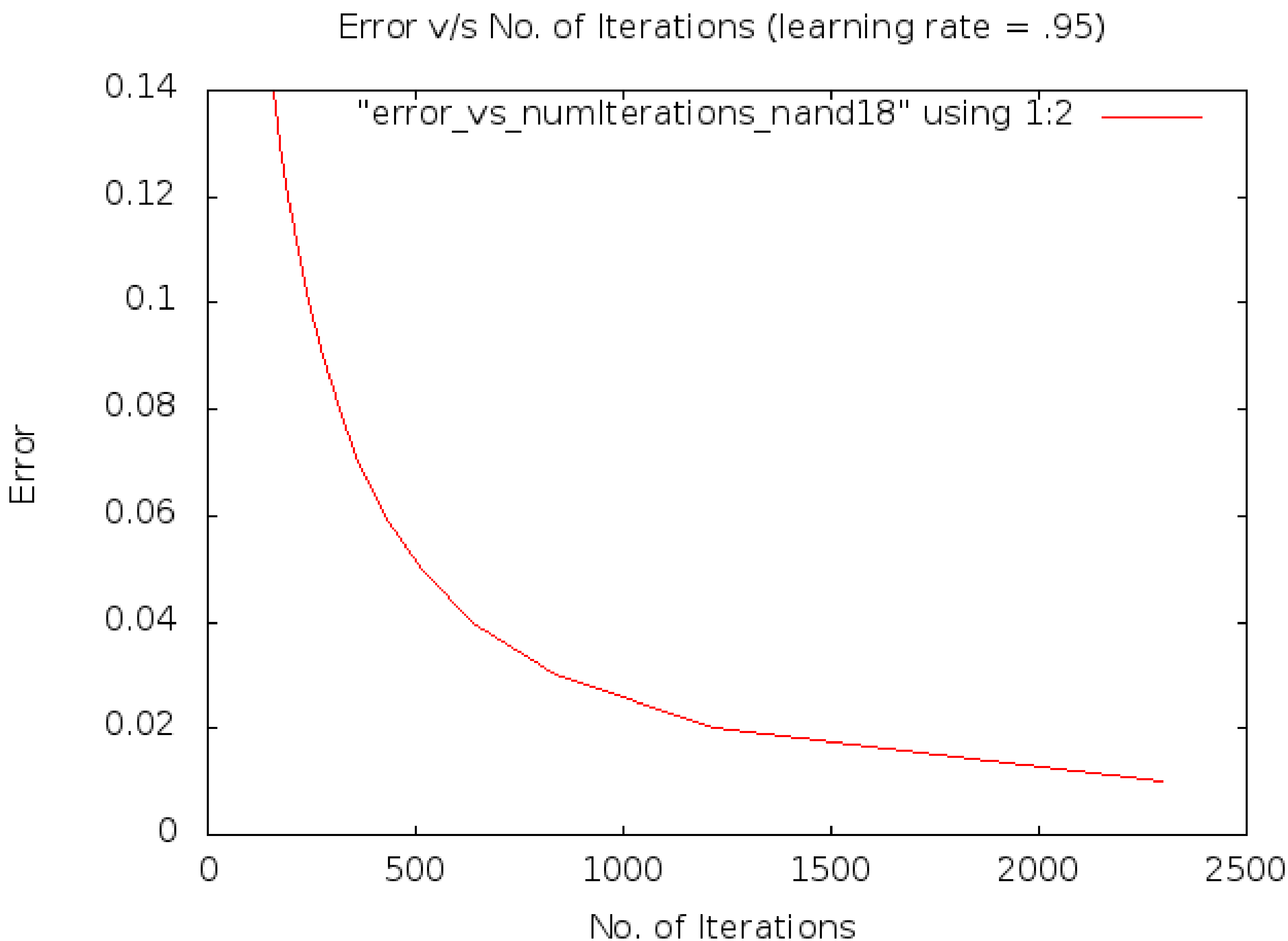
Effect of Learning Rate (NAND)



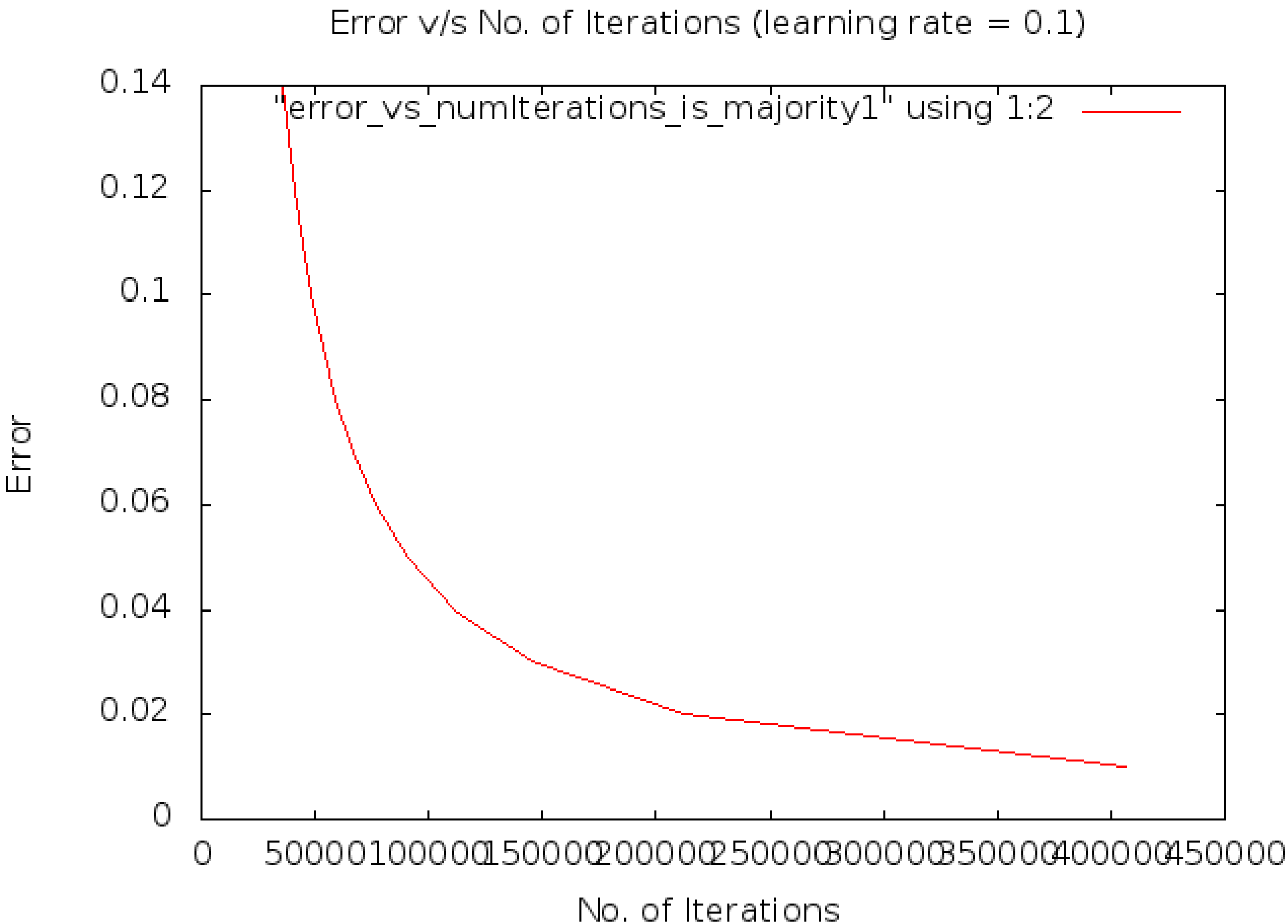
Effect of Learning Rate (NAND)



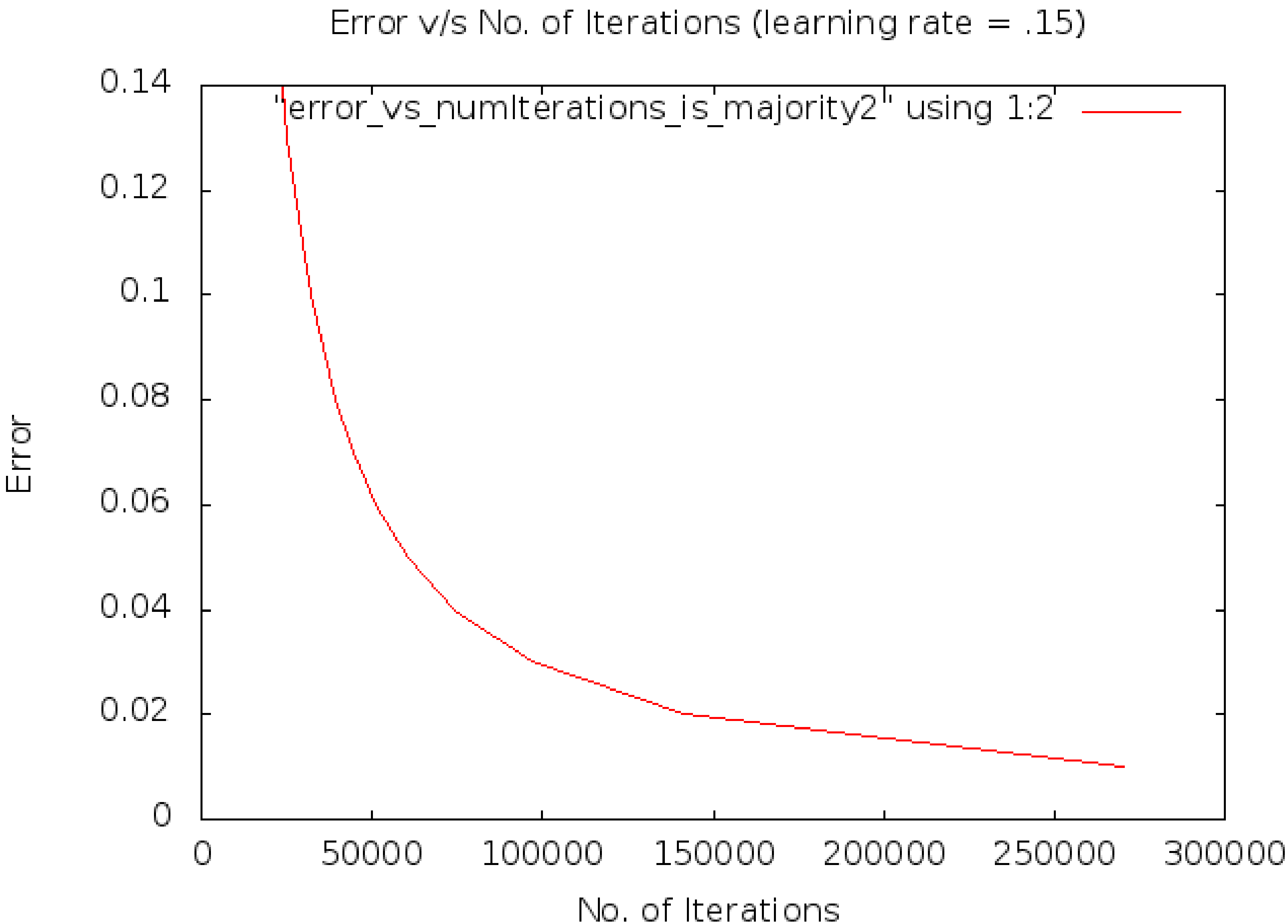
Effect of Learning Rate (NAND)



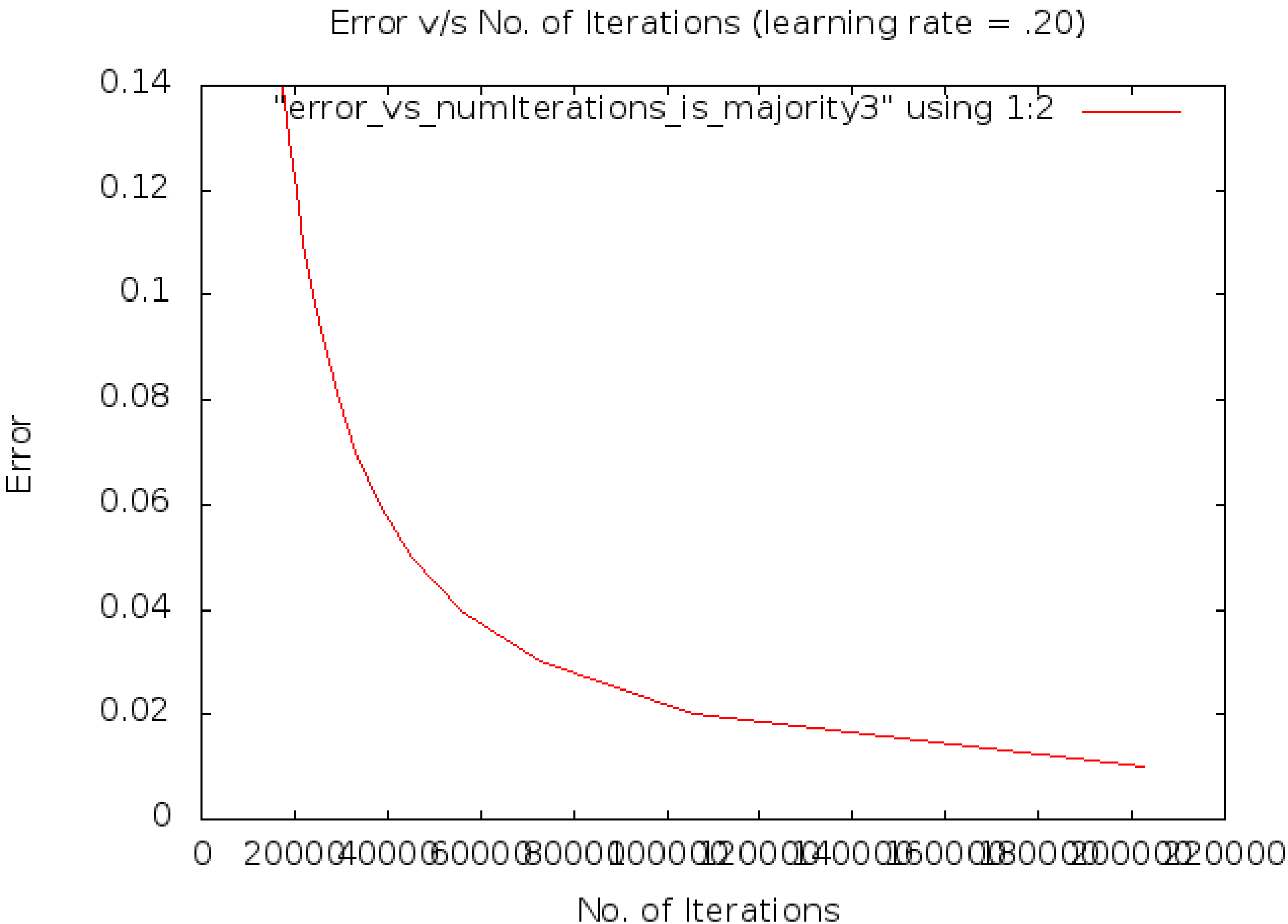
Effect of Learning Rate (Majority)



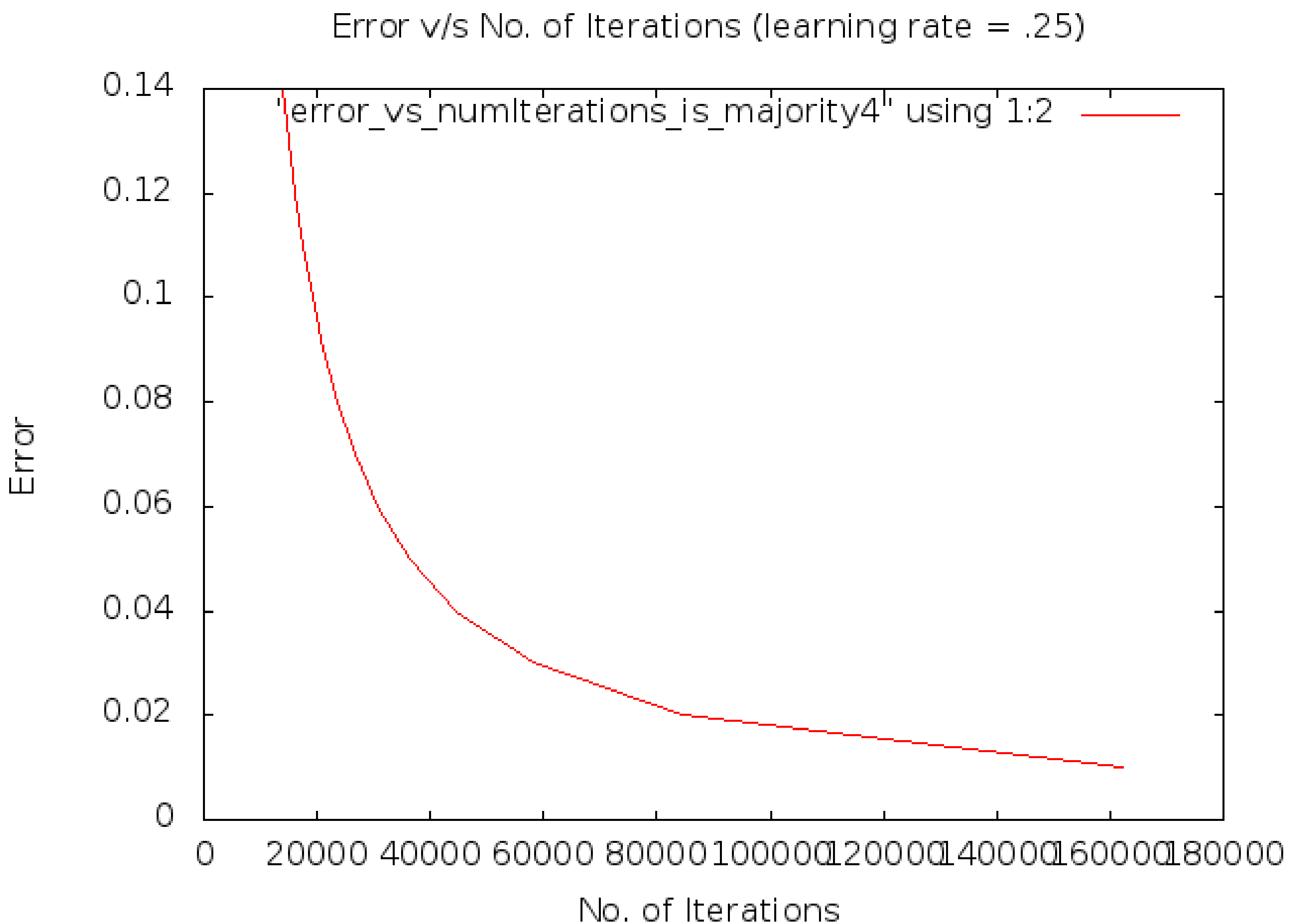
Effect of Learning Rate (Majority)



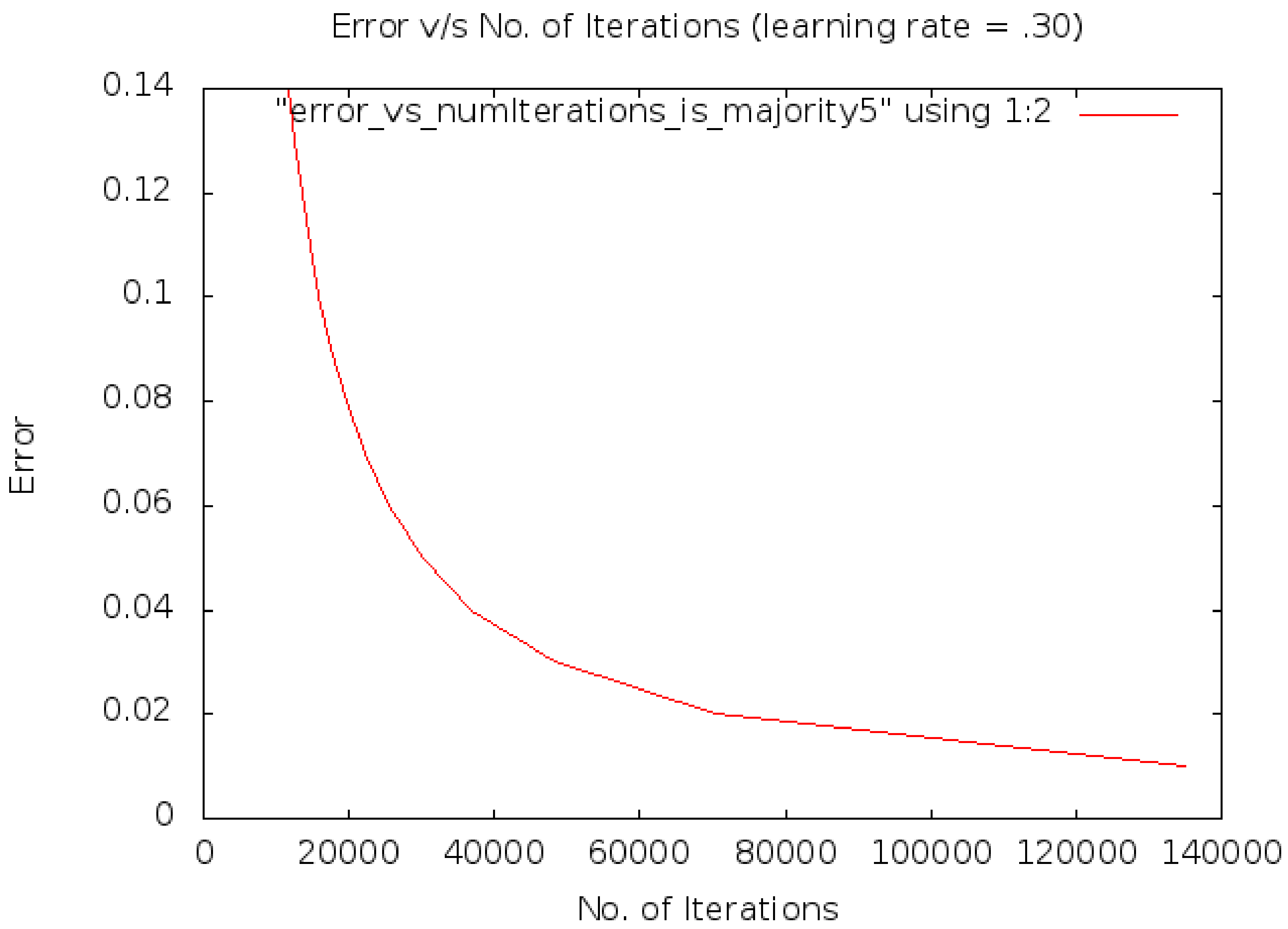
Effect of Learning Rate (Majority)



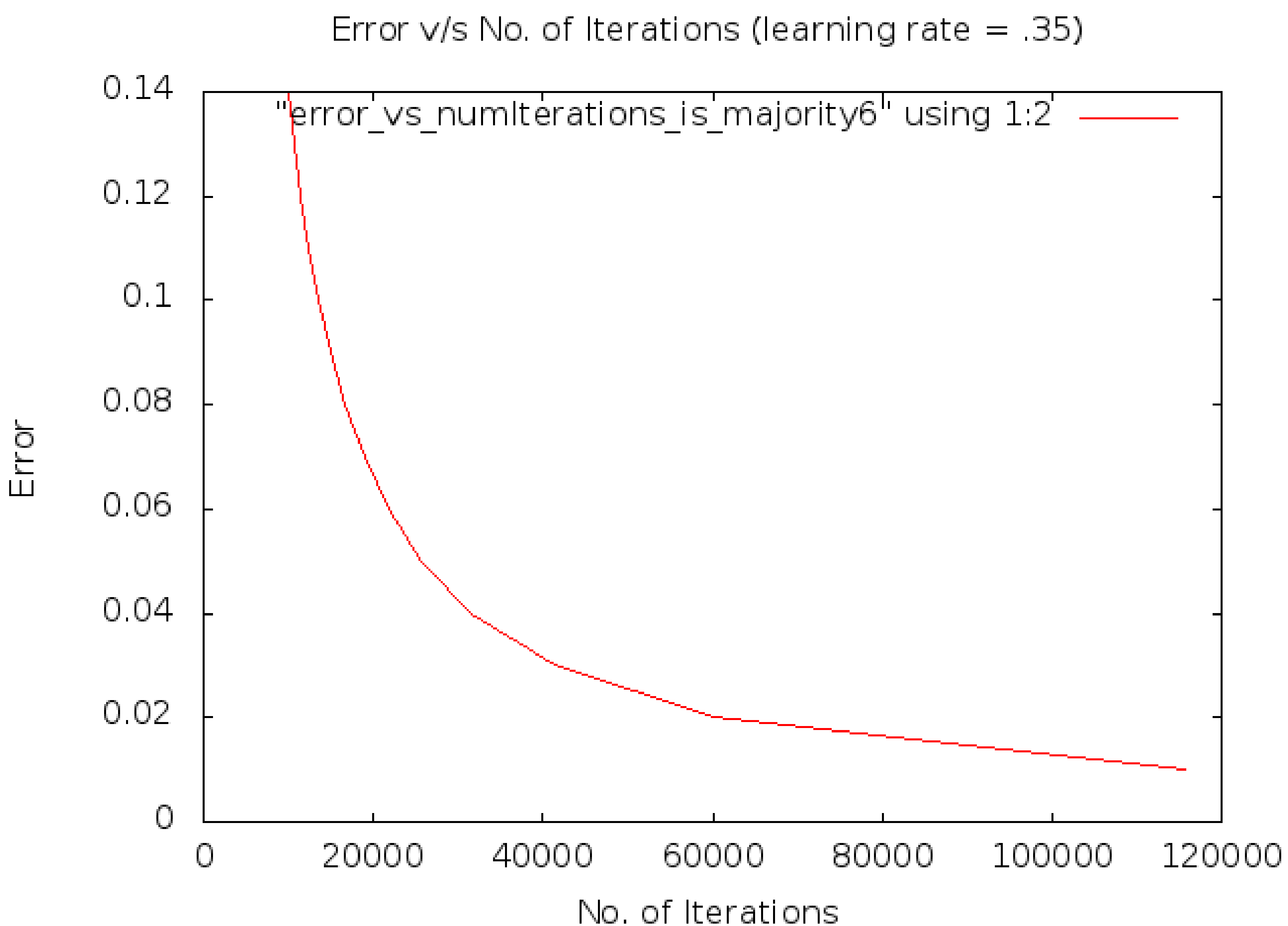
Effect of Learning Rate (Majority)



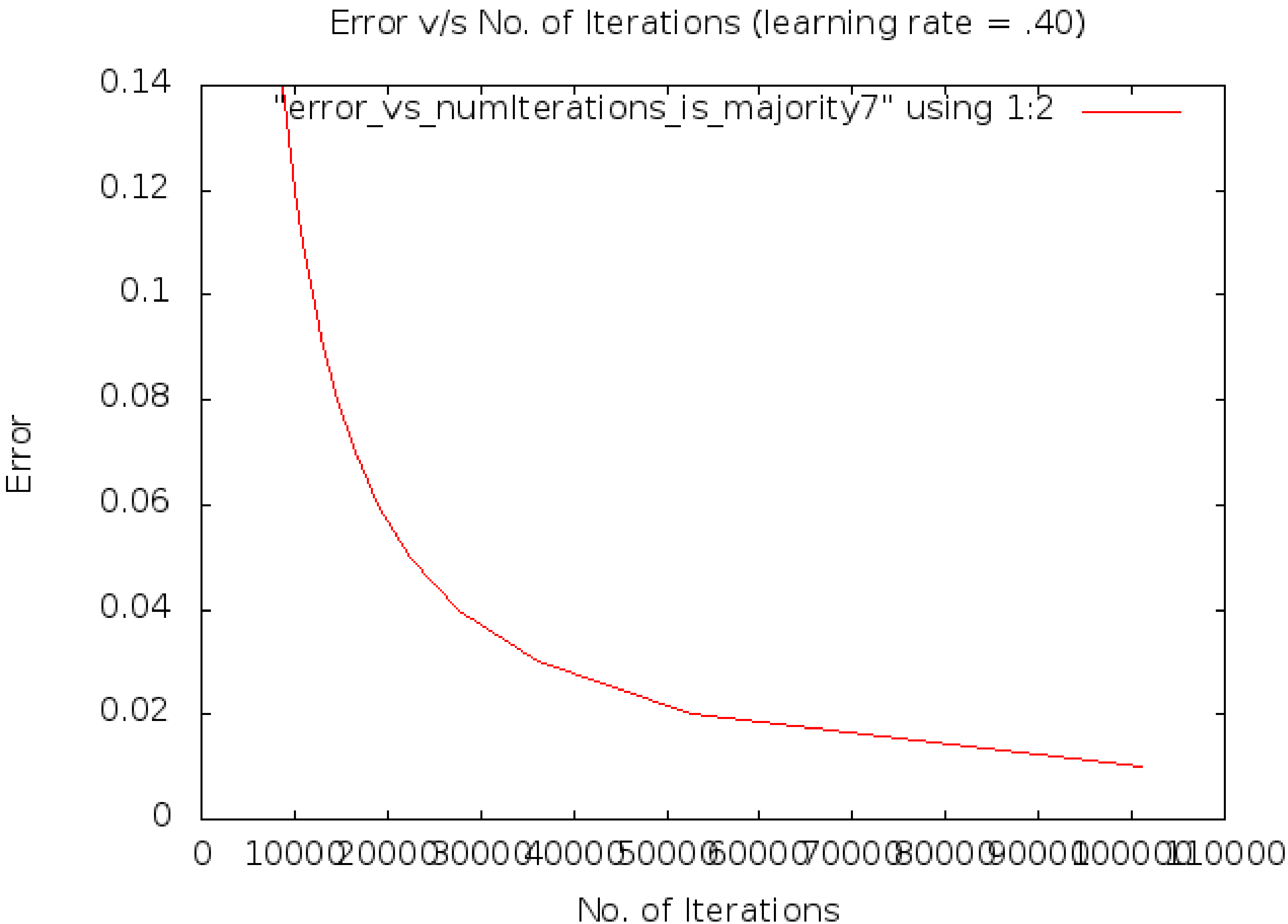
Effect of Learning Rate (Majority)



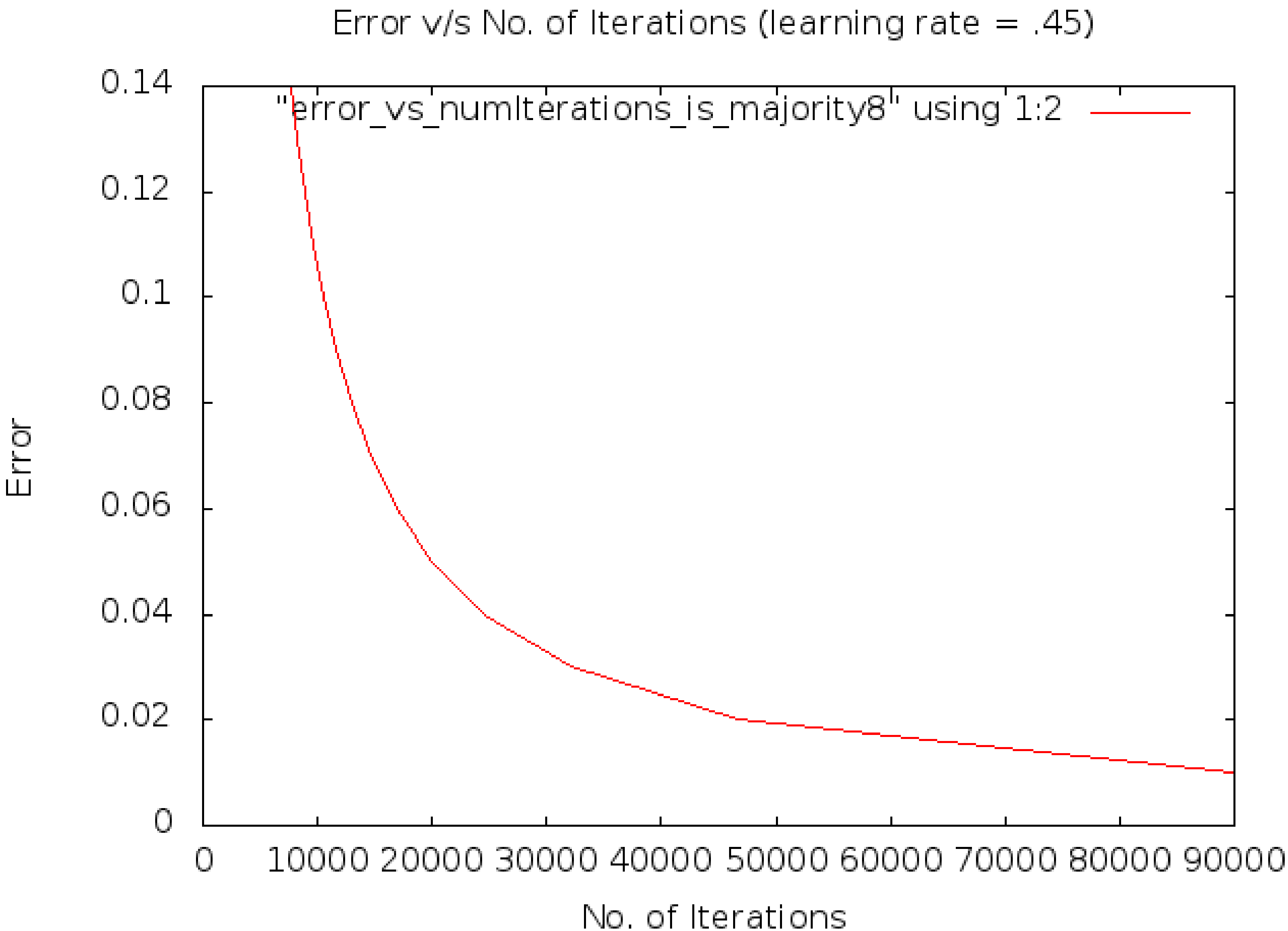
Effect of Learning Rate (Majority)



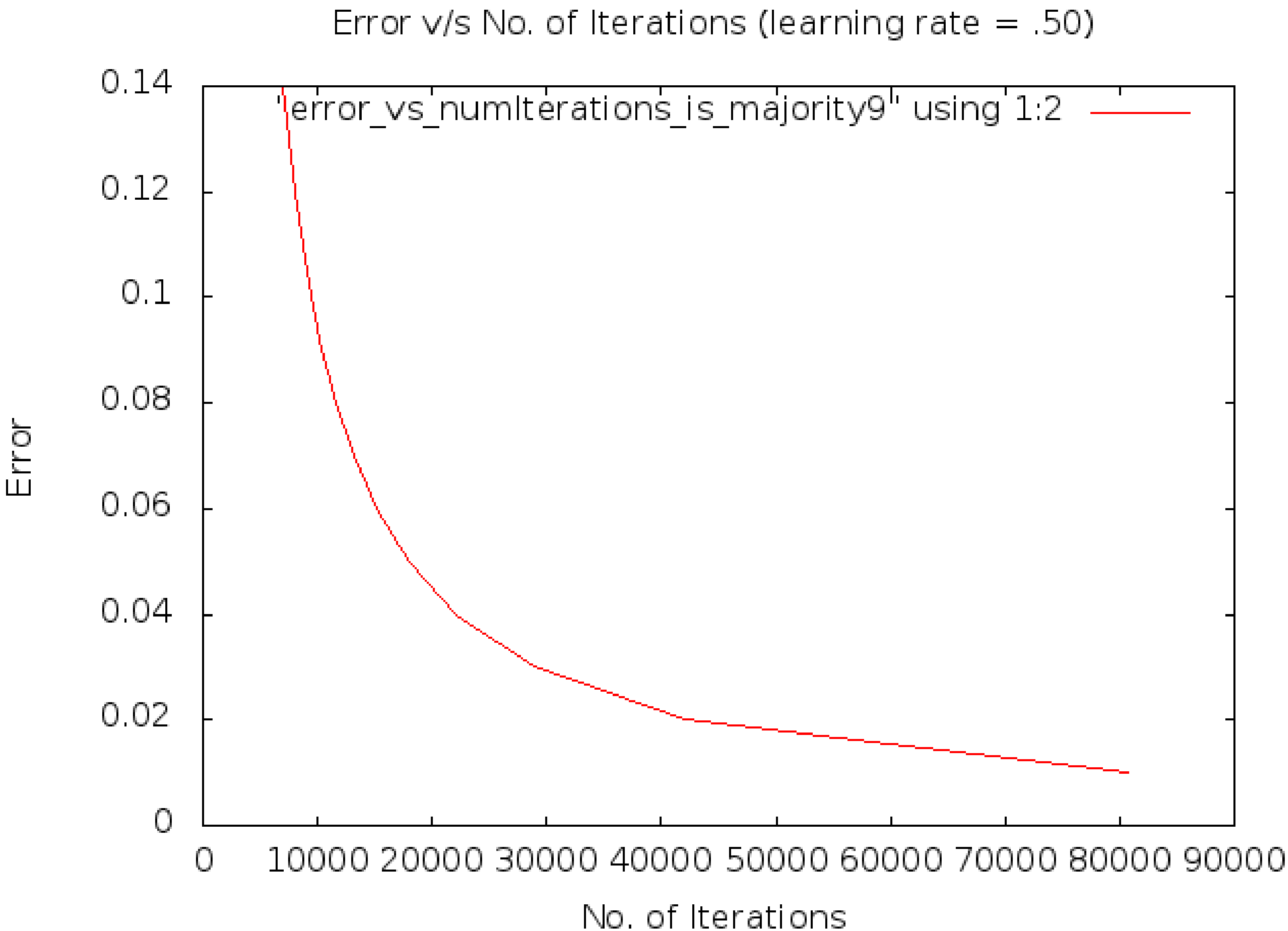
Effect of Learning Rate (Majority)



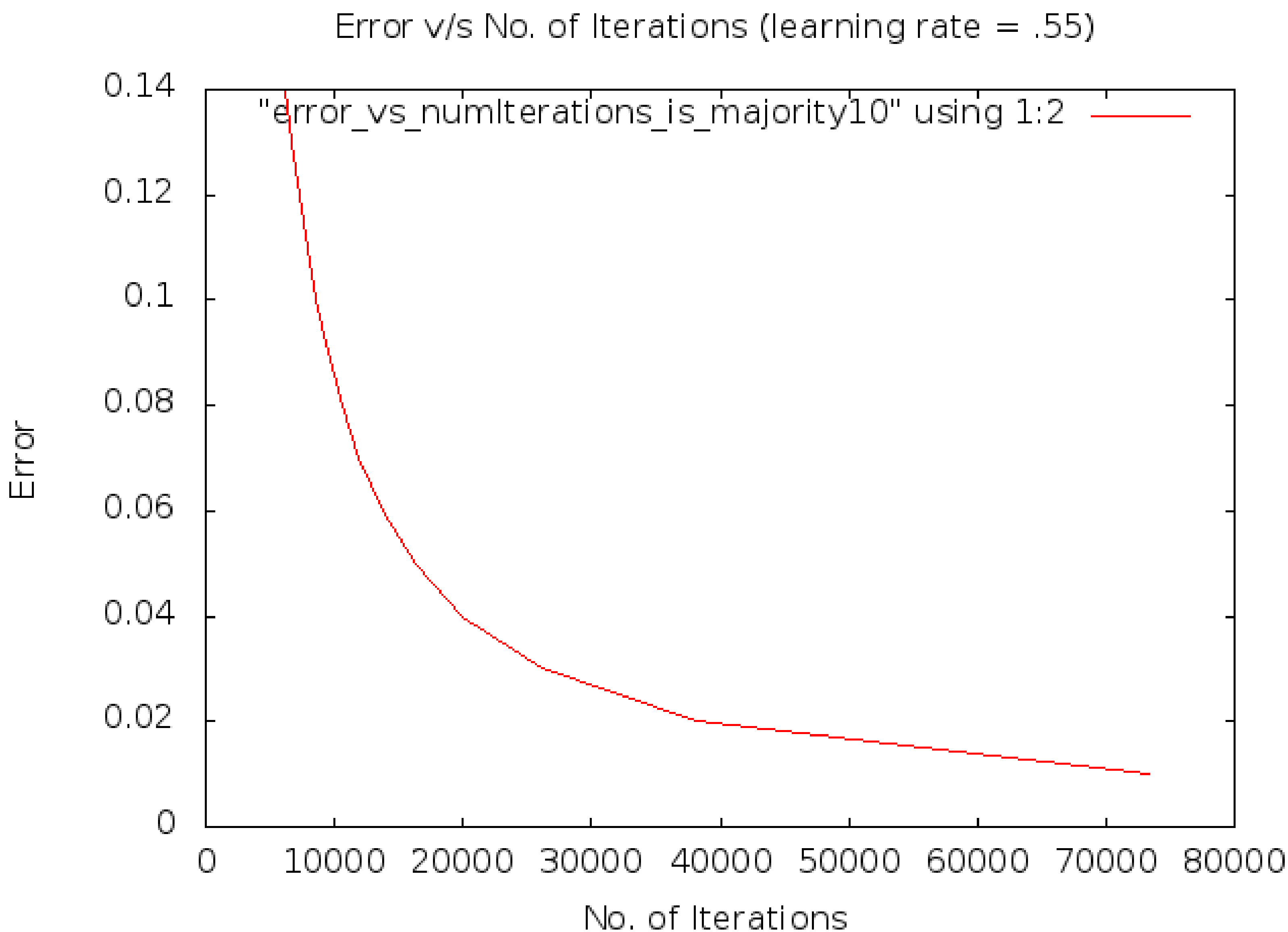
Effect of Learning Rate (Majority)



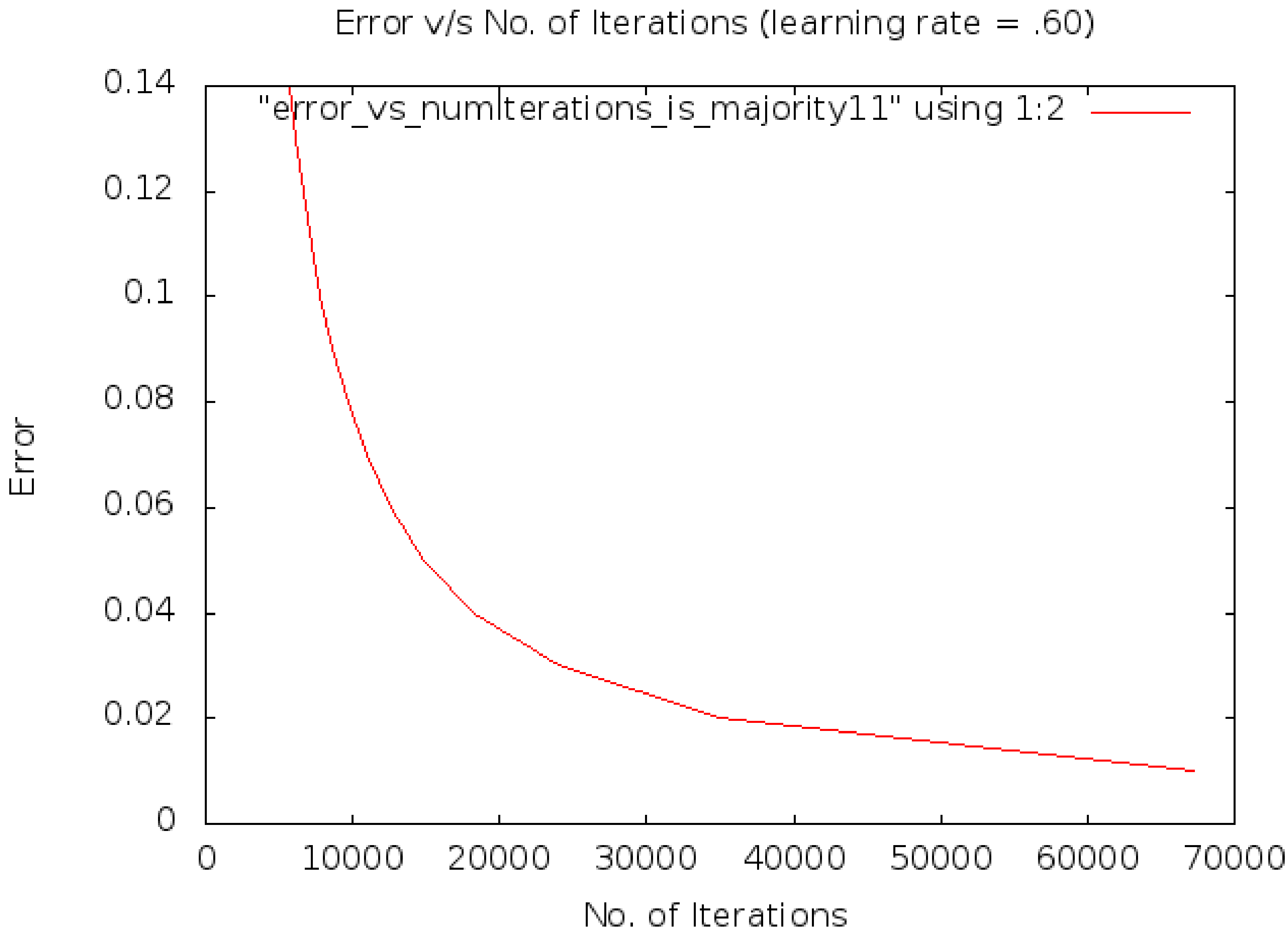
Effect of Learning Rate (Majority)



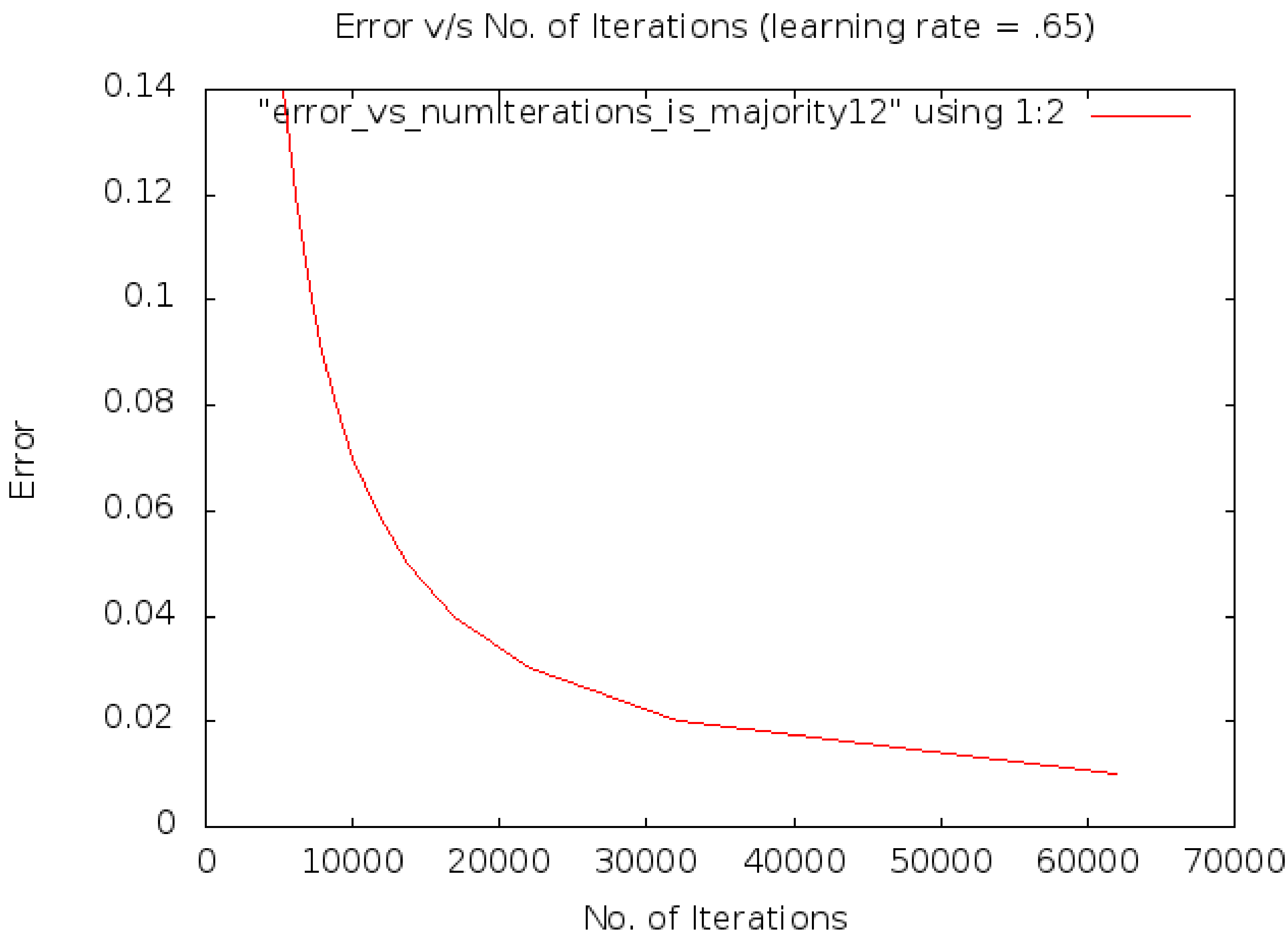
Effect of Learning Rate (Majority)



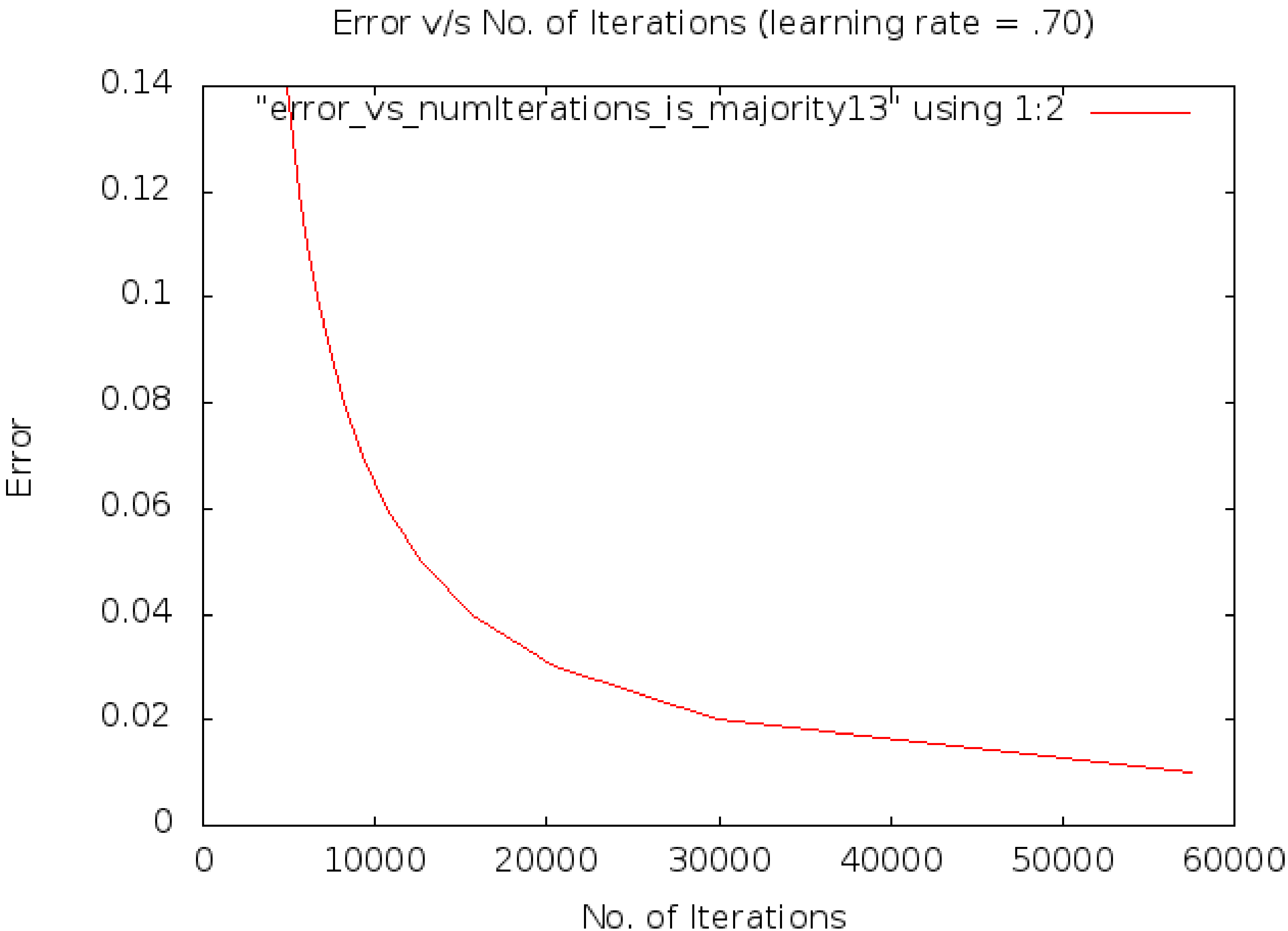
Effect of Learning Rate (Majority)



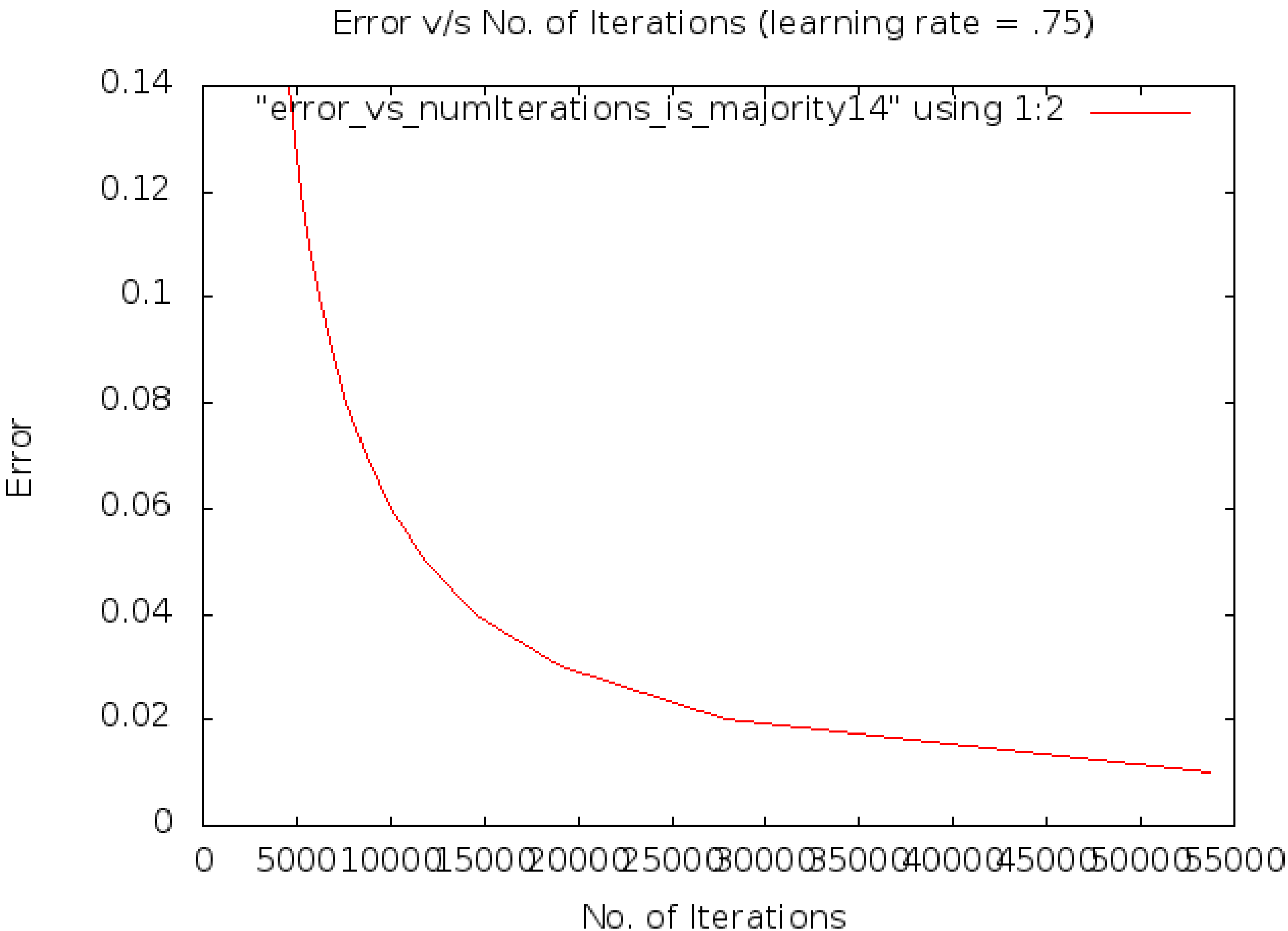
Effect of Learning Rate (Majority)



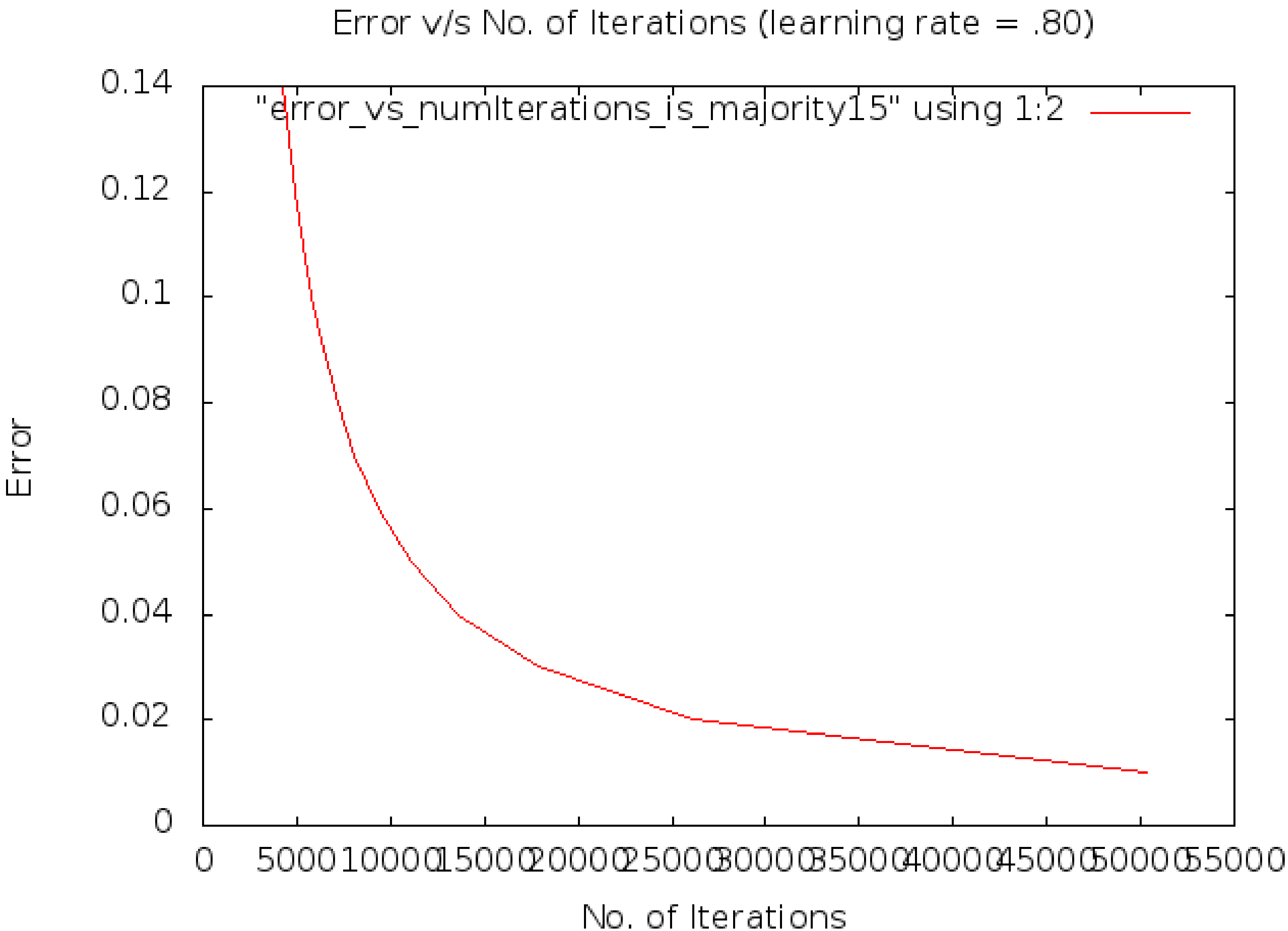
Effect of Learning Rate (Majority)



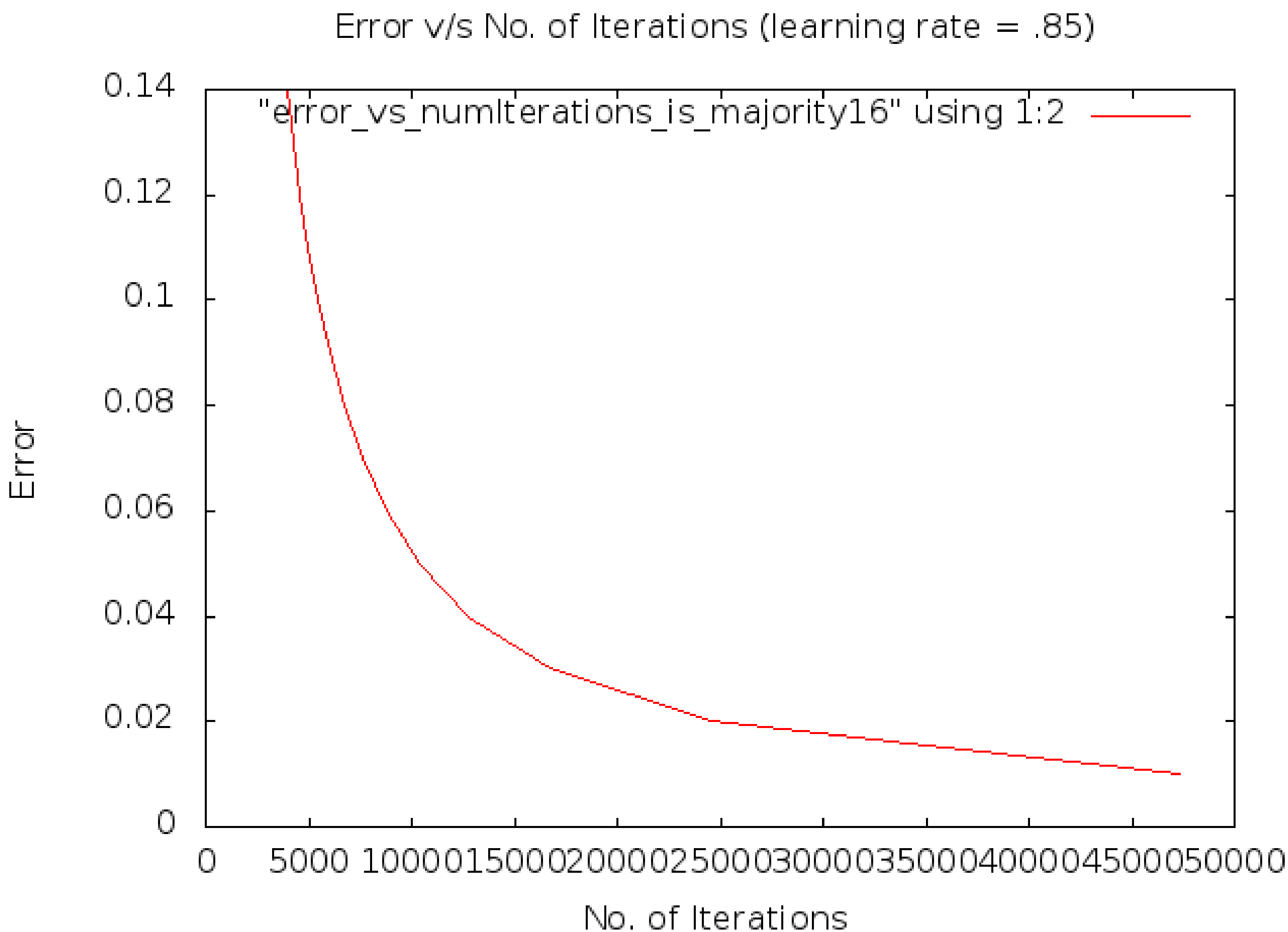
Effect of Learning Rate (Majority)



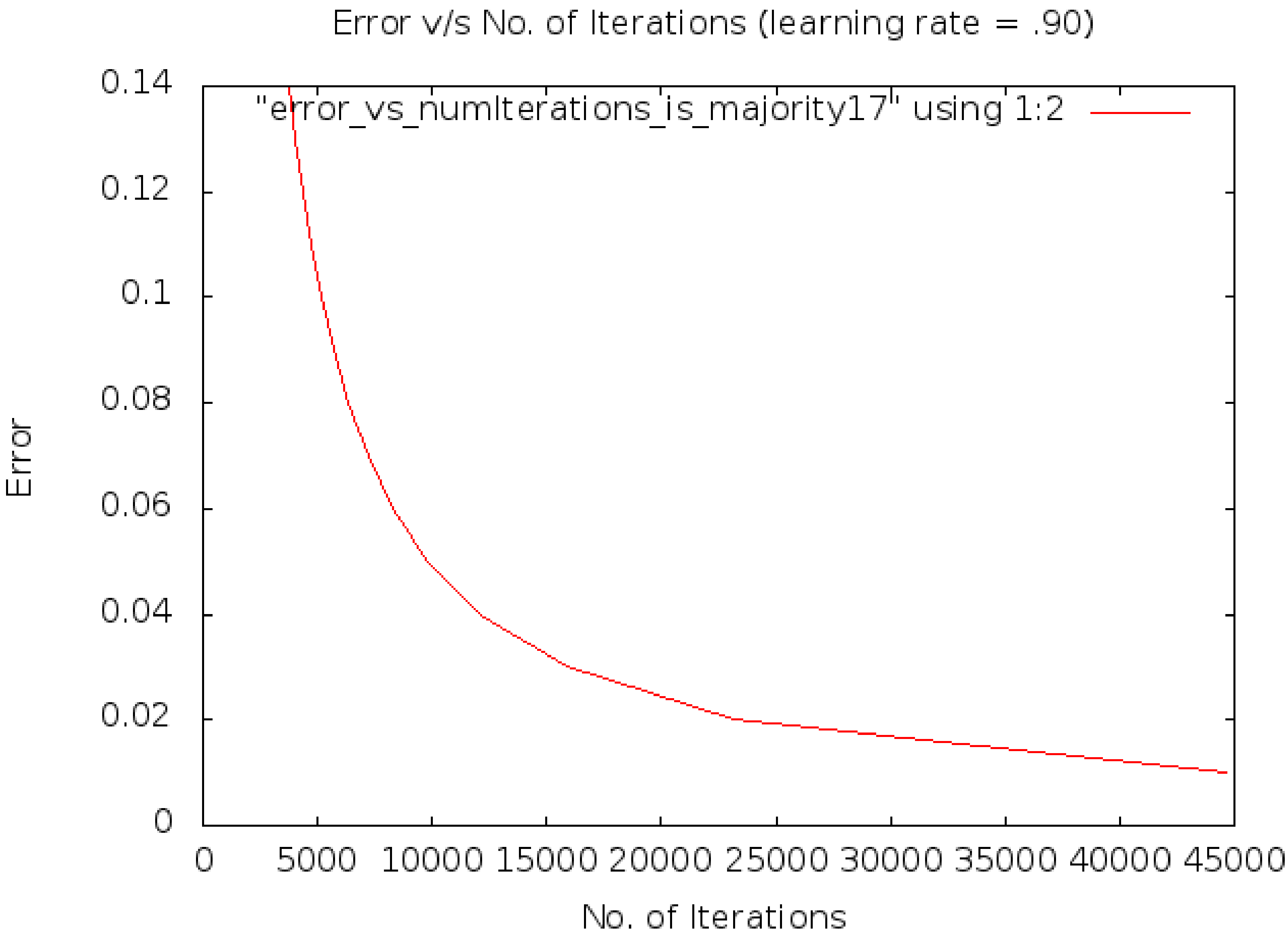
Effect of Learning Rate (Majority)



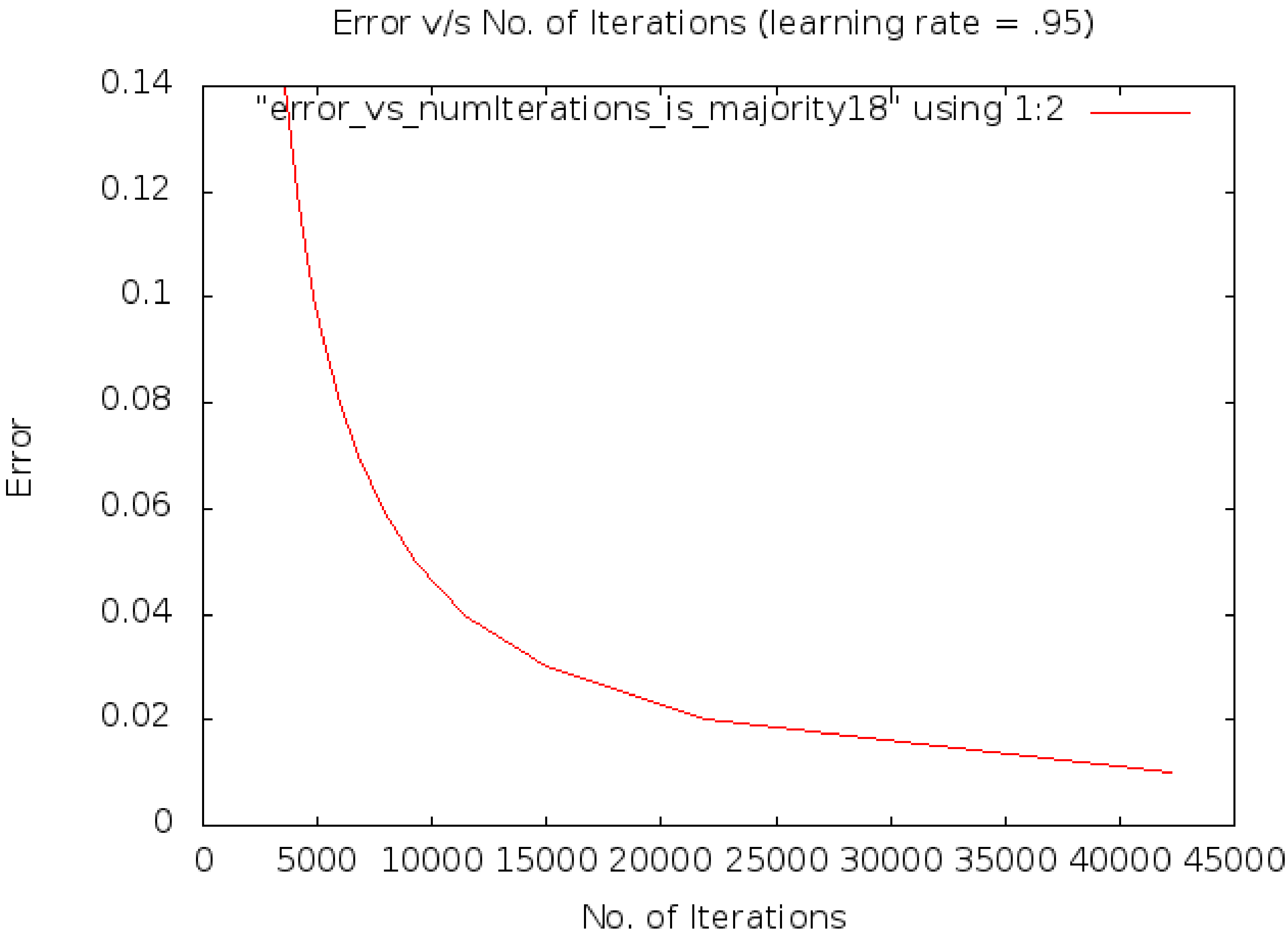
Effect of Learning Rate (Majority)



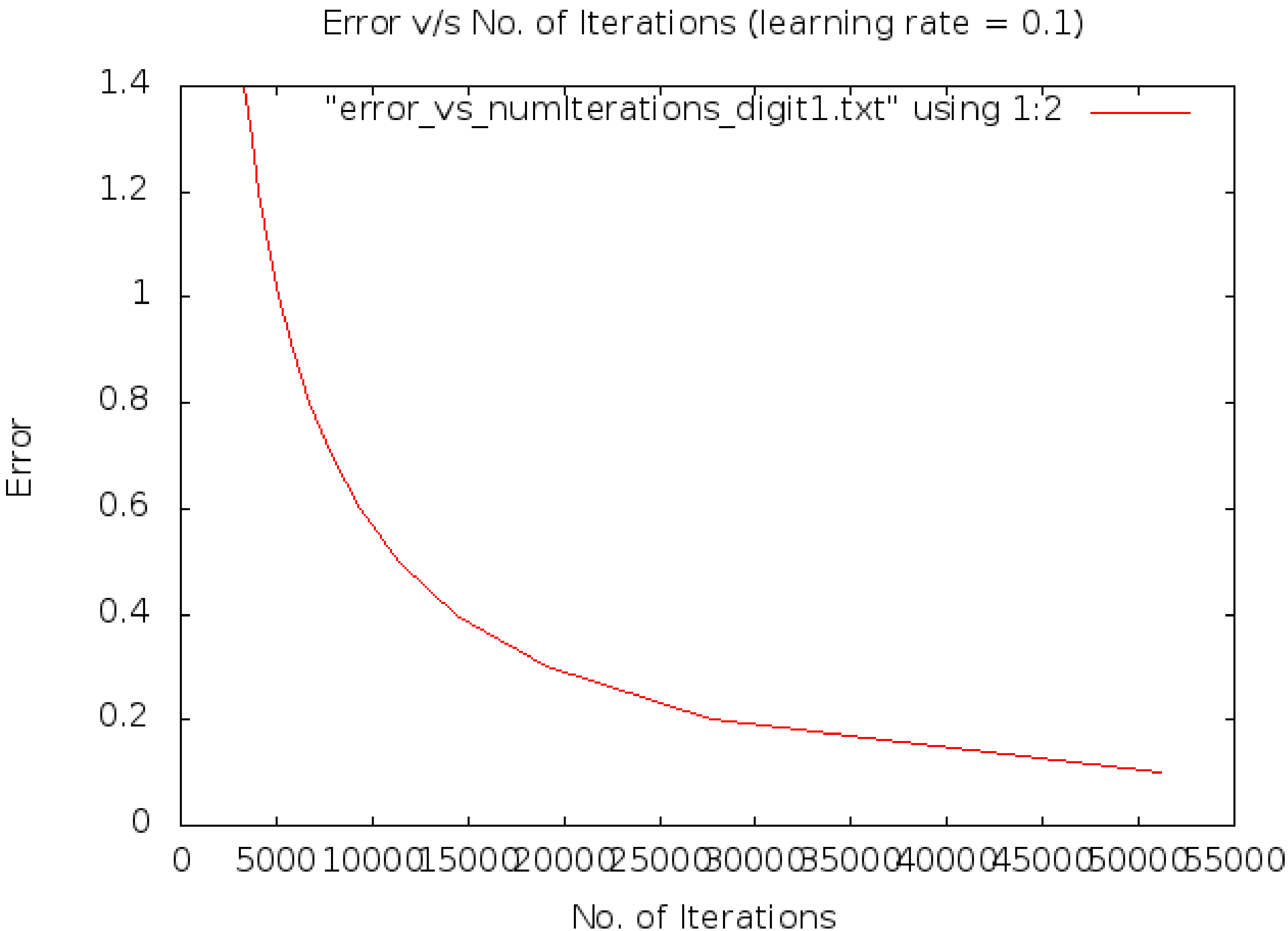
Effect of Learning Rate (Majority)



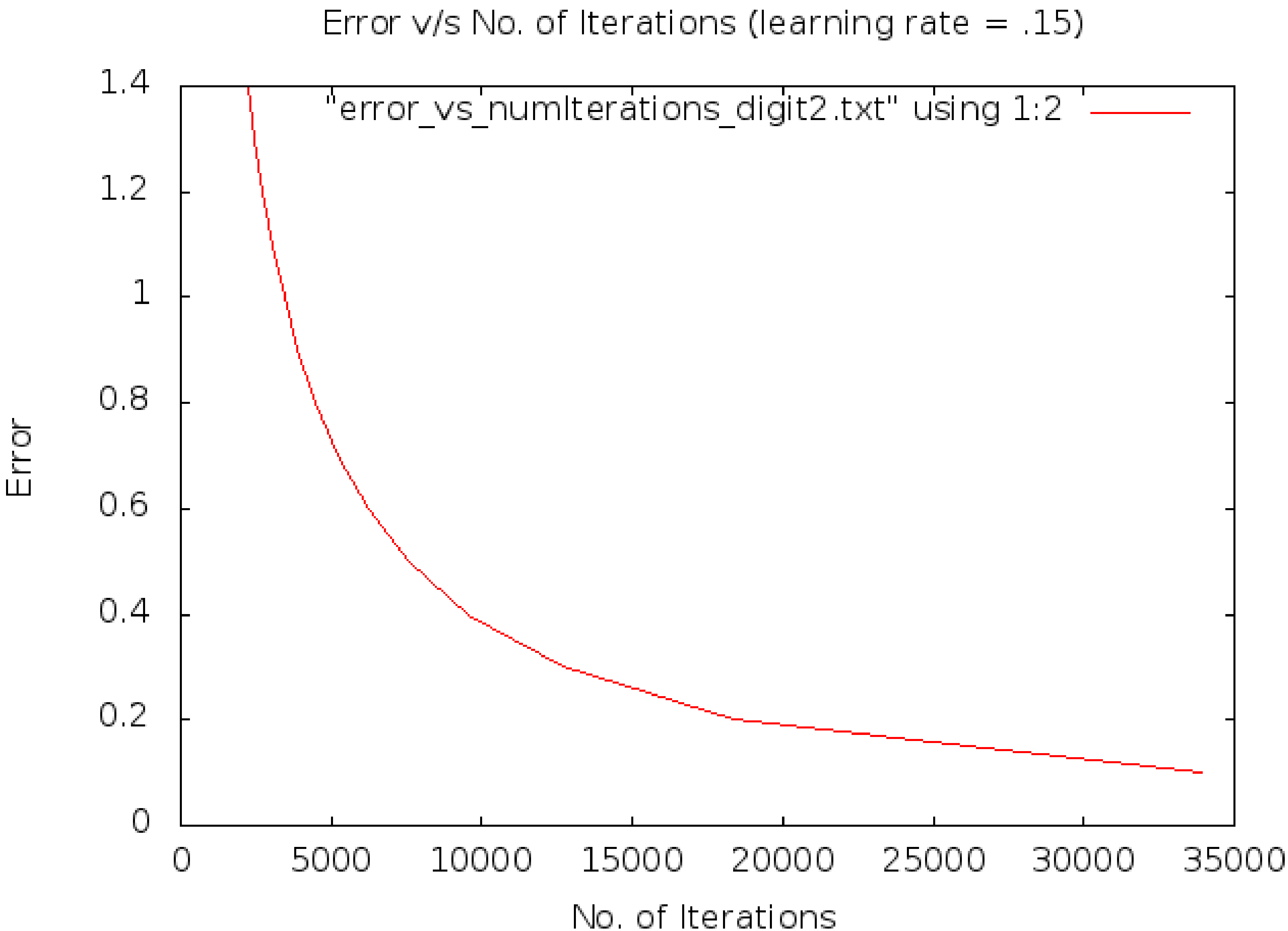
Effect of Learning Rate (Majority)



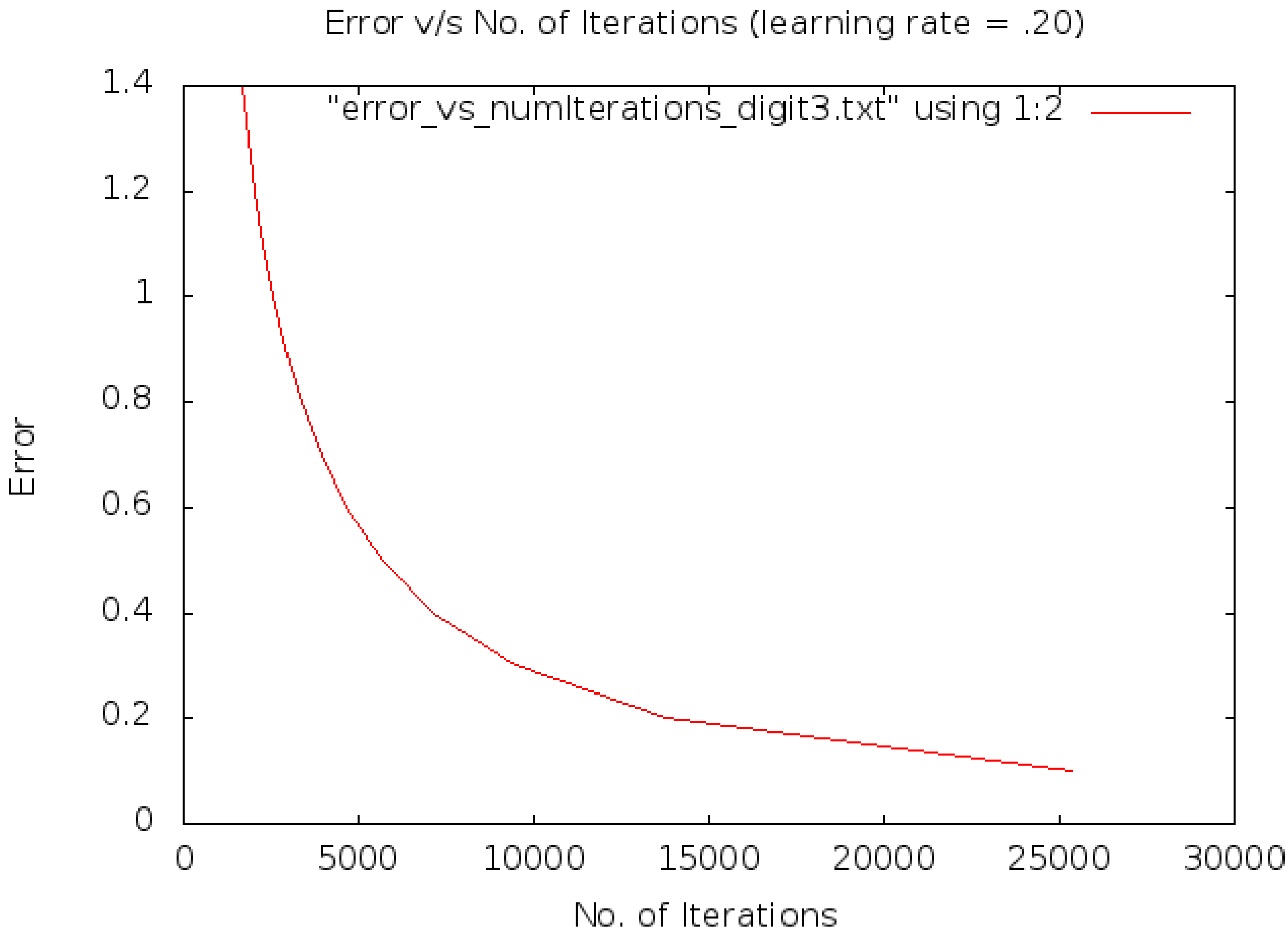
Effect of Learning Rate (Digit Recogniser)



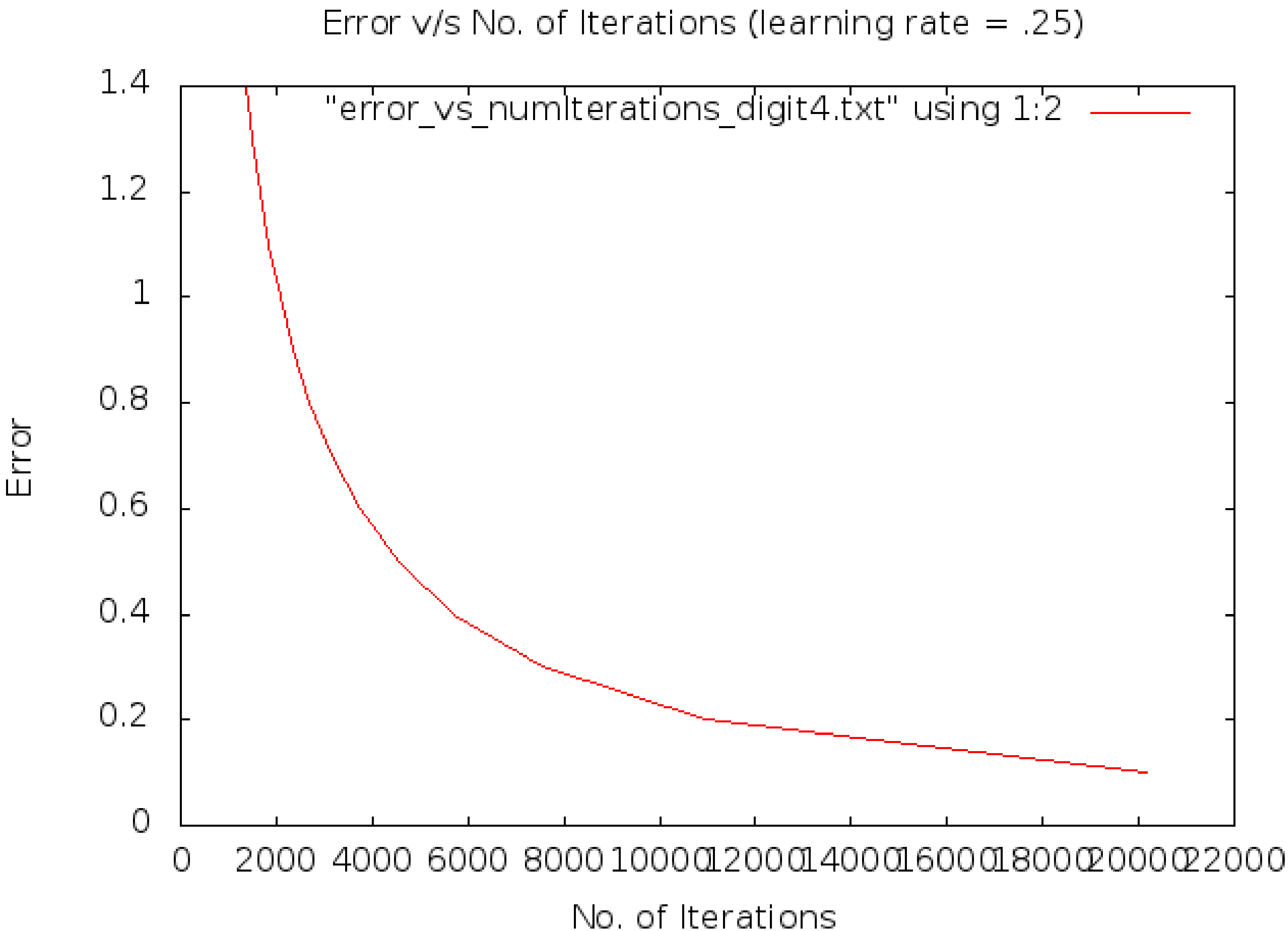
Effect of Learning Rate (Digit Recogniser)



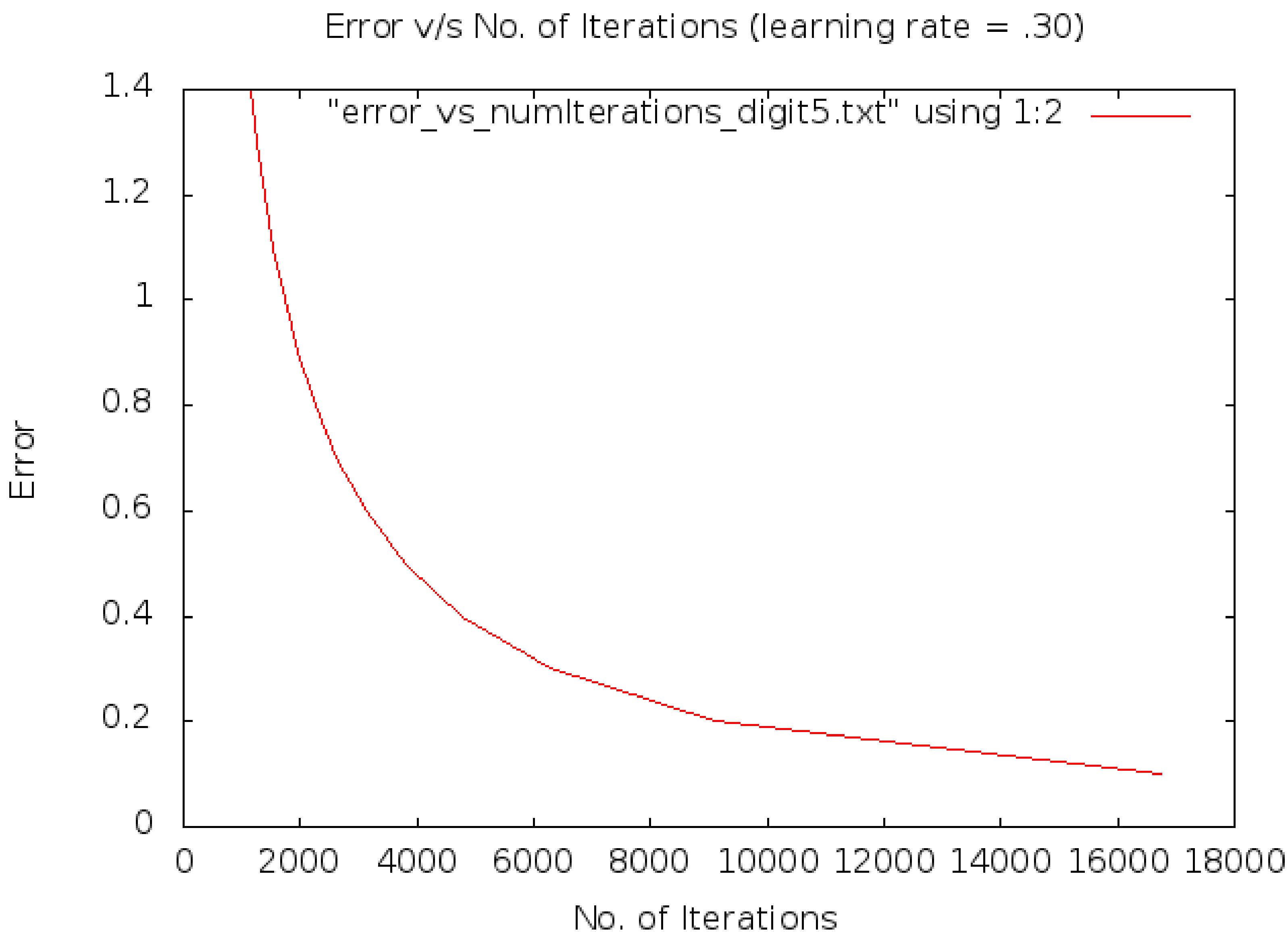
Effect of Learning Rate (Digit Recogniser)



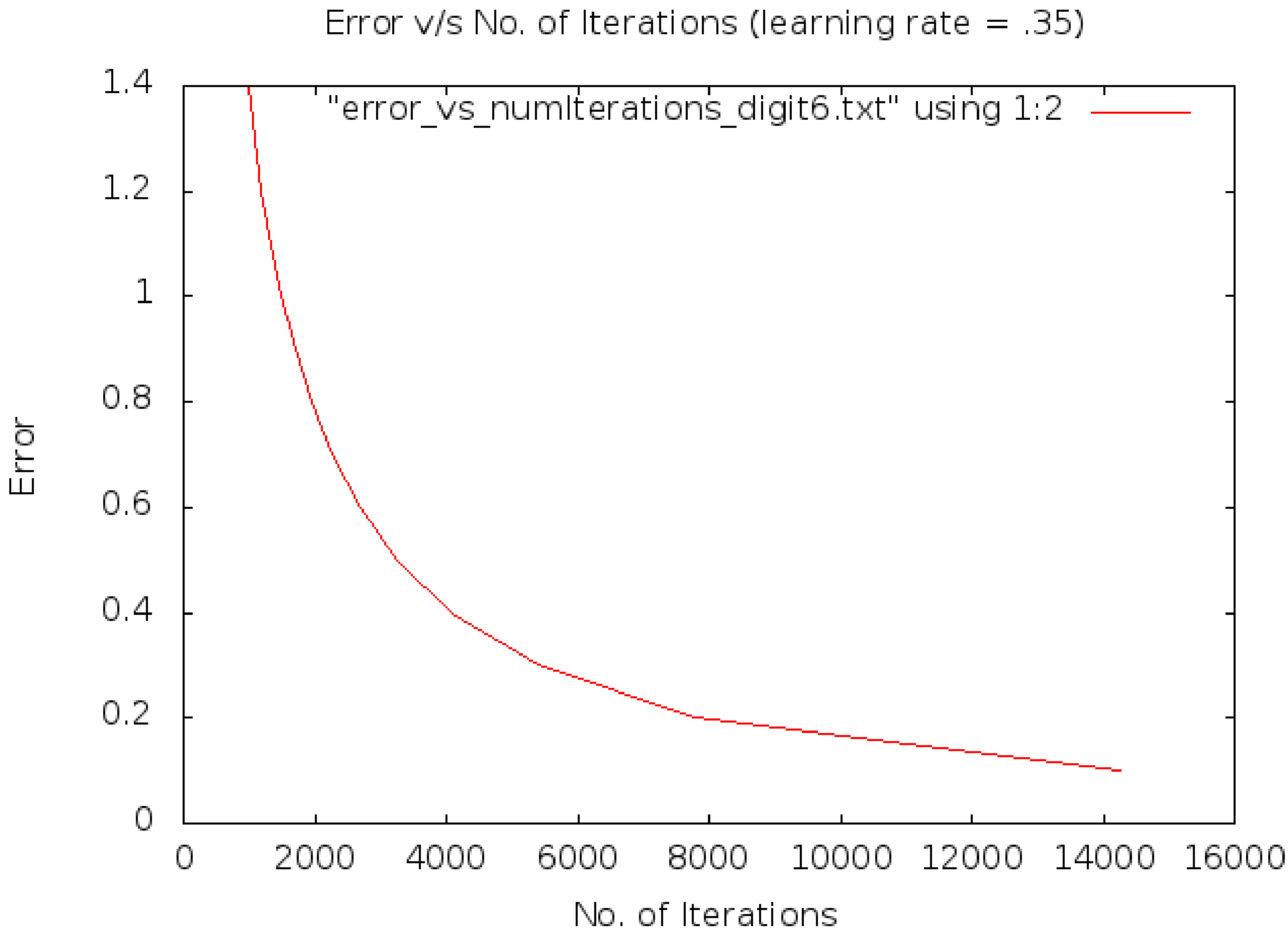
Effect of Learning Rate (Digit Recogniser)



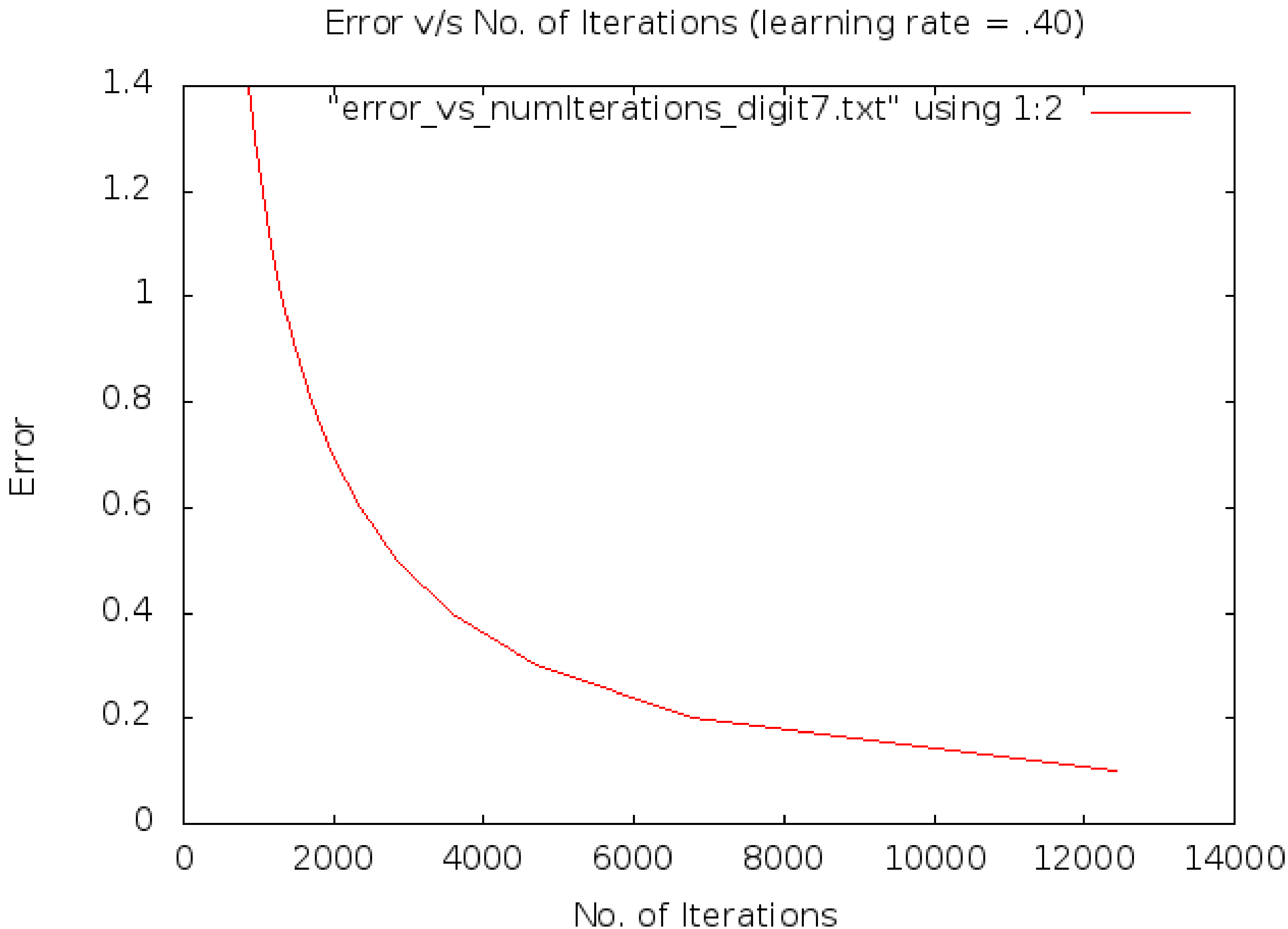
Effect of Learning Rate (Digit Recogniser)



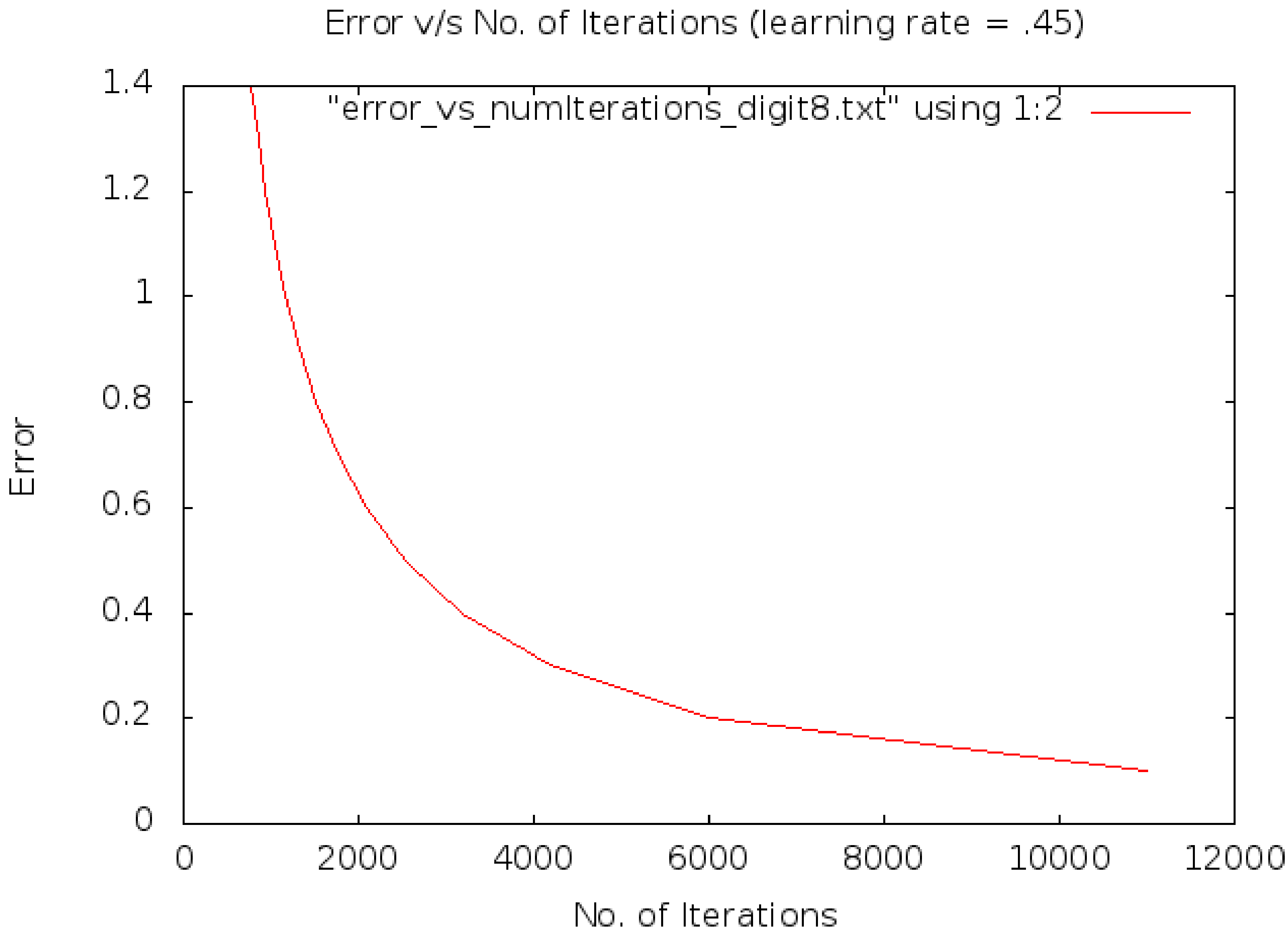
Effect of Learning Rate (Digit Recogniser)



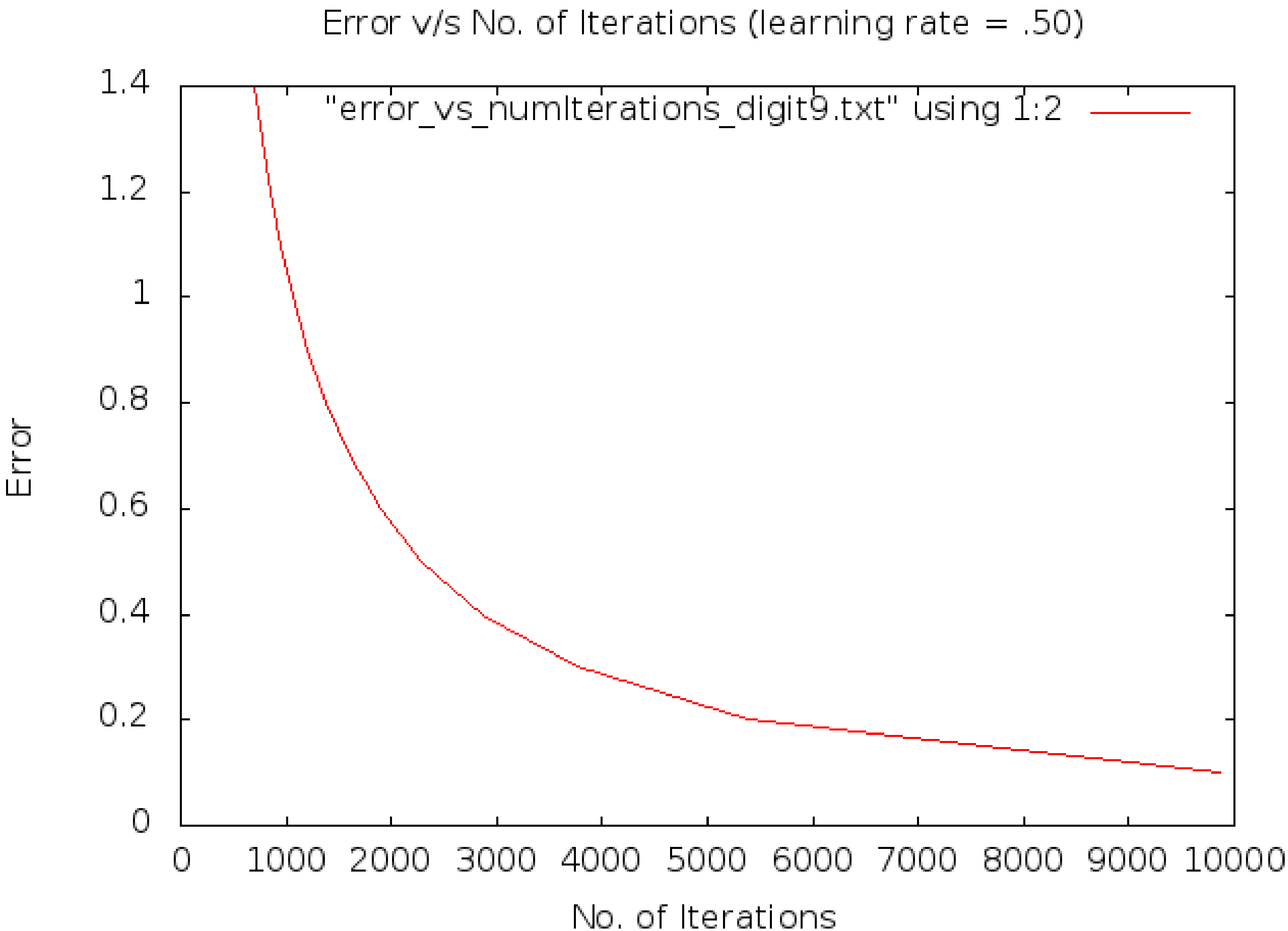
Effect of Learning Rate (Digit Recogniser)



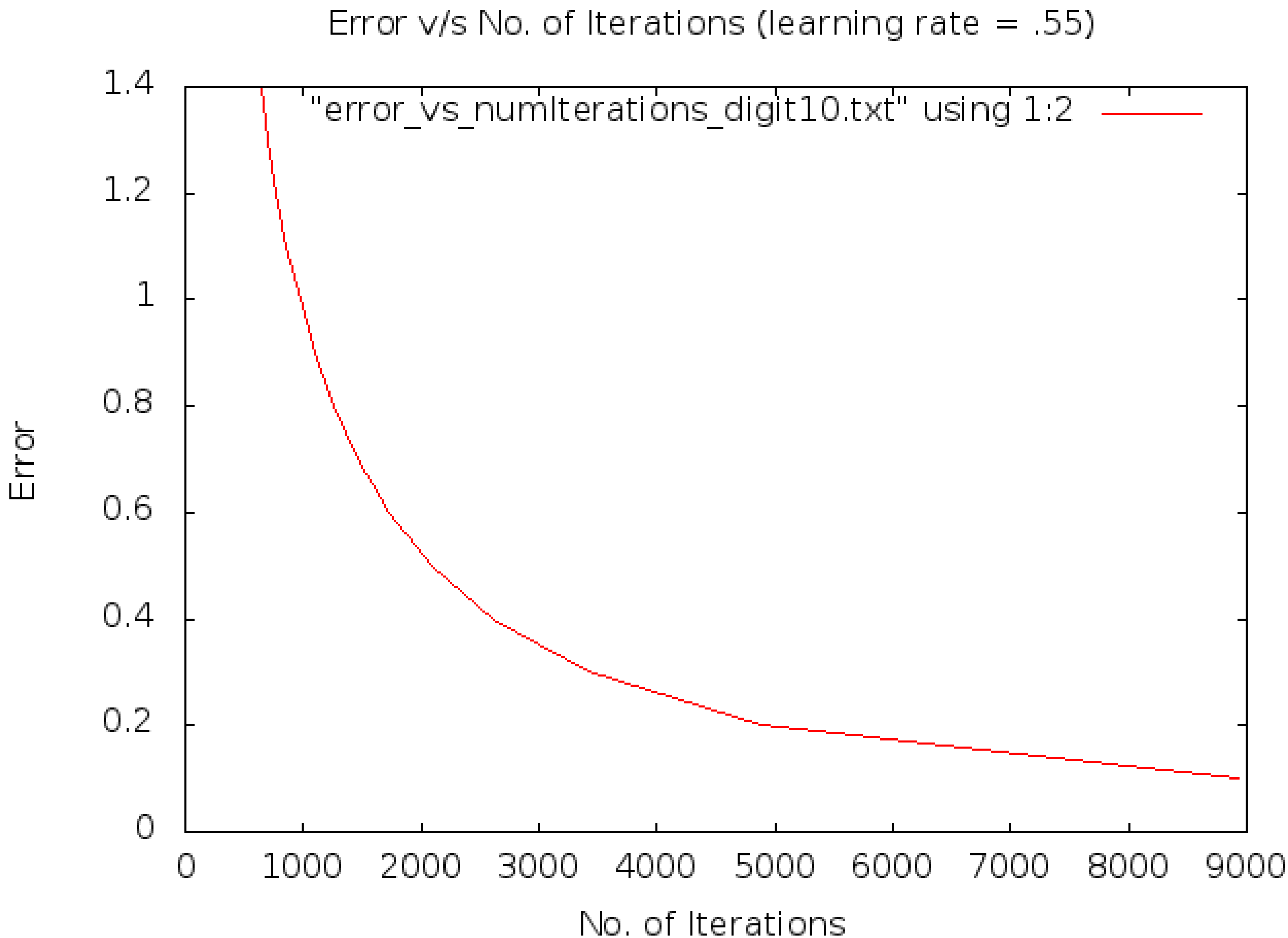
Effect of Learning Rate (Digit Recogniser)



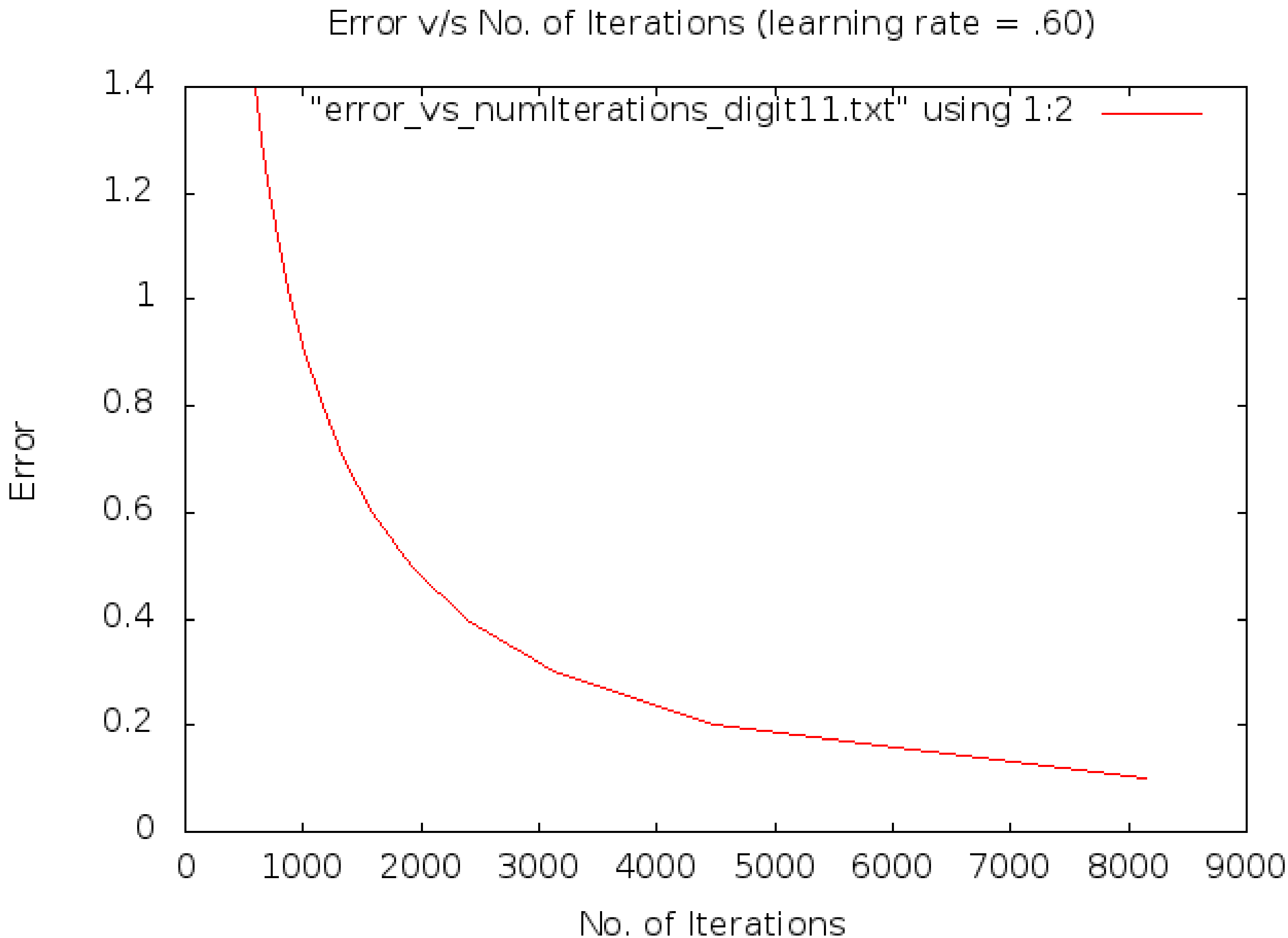
Effect of Learning Rate (Digit Recogniser)



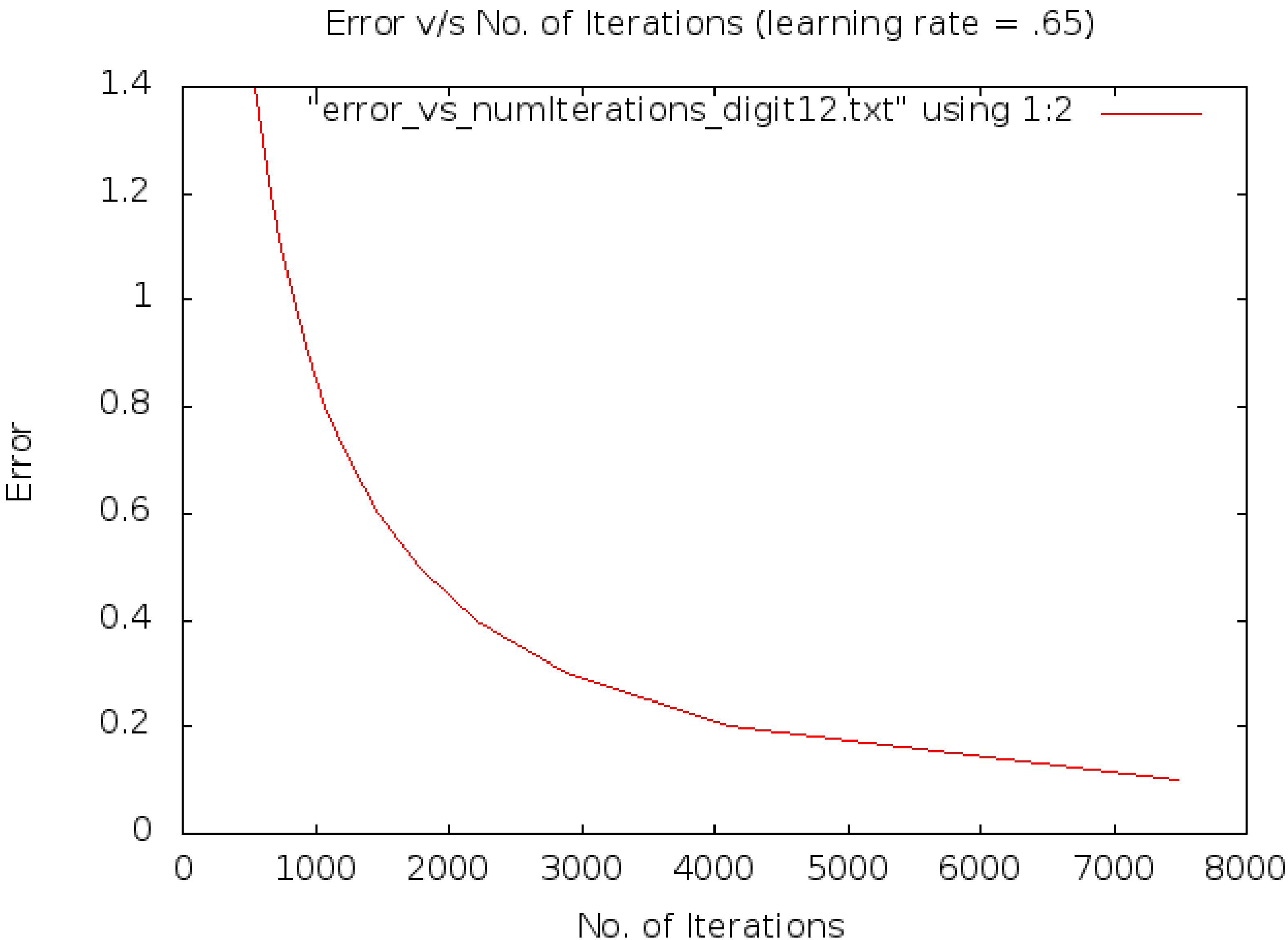
Effect of Learning Rate (Digit Recogniser)



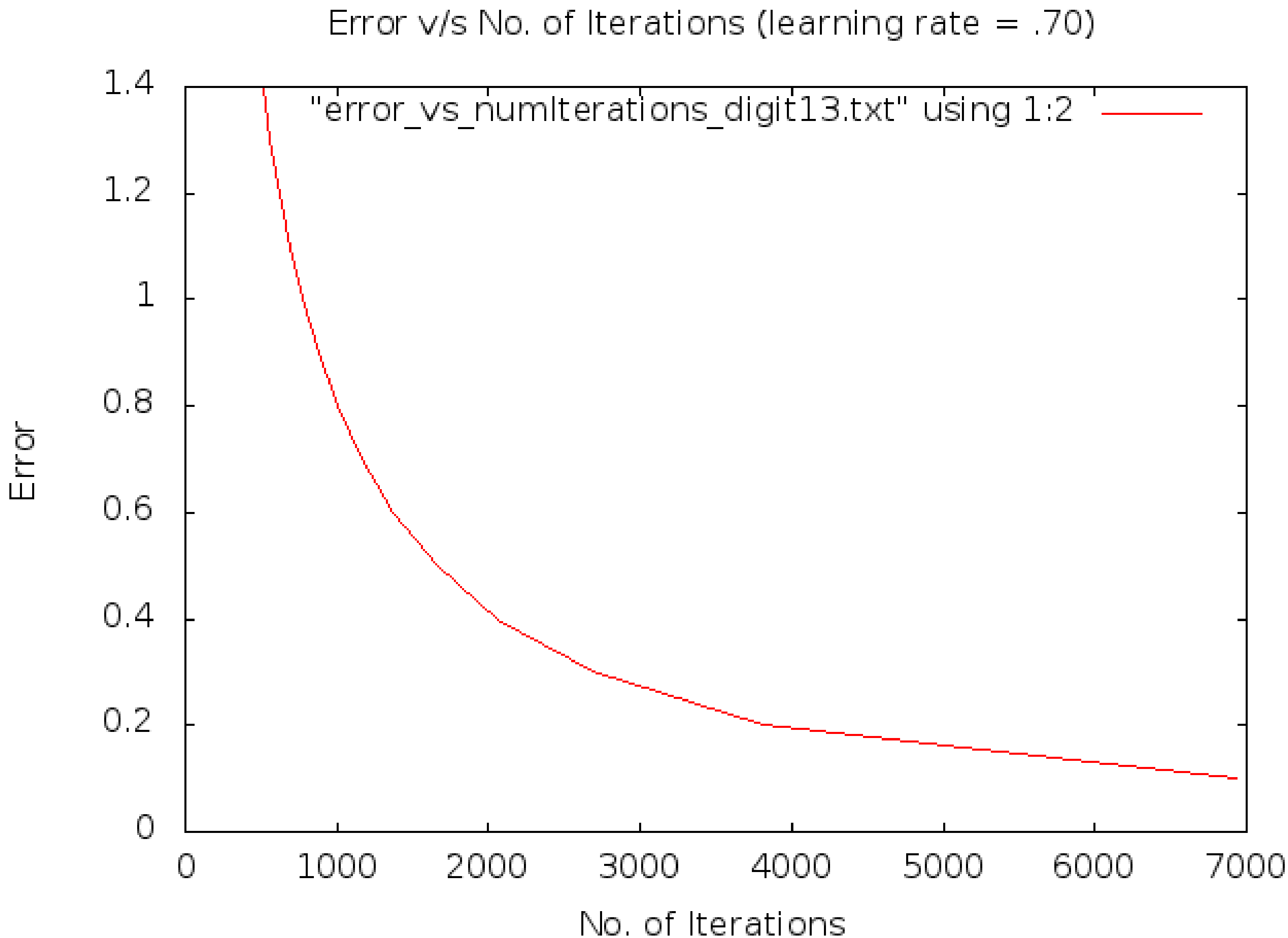
Effect of Learning Rate (Digit Recogniser)



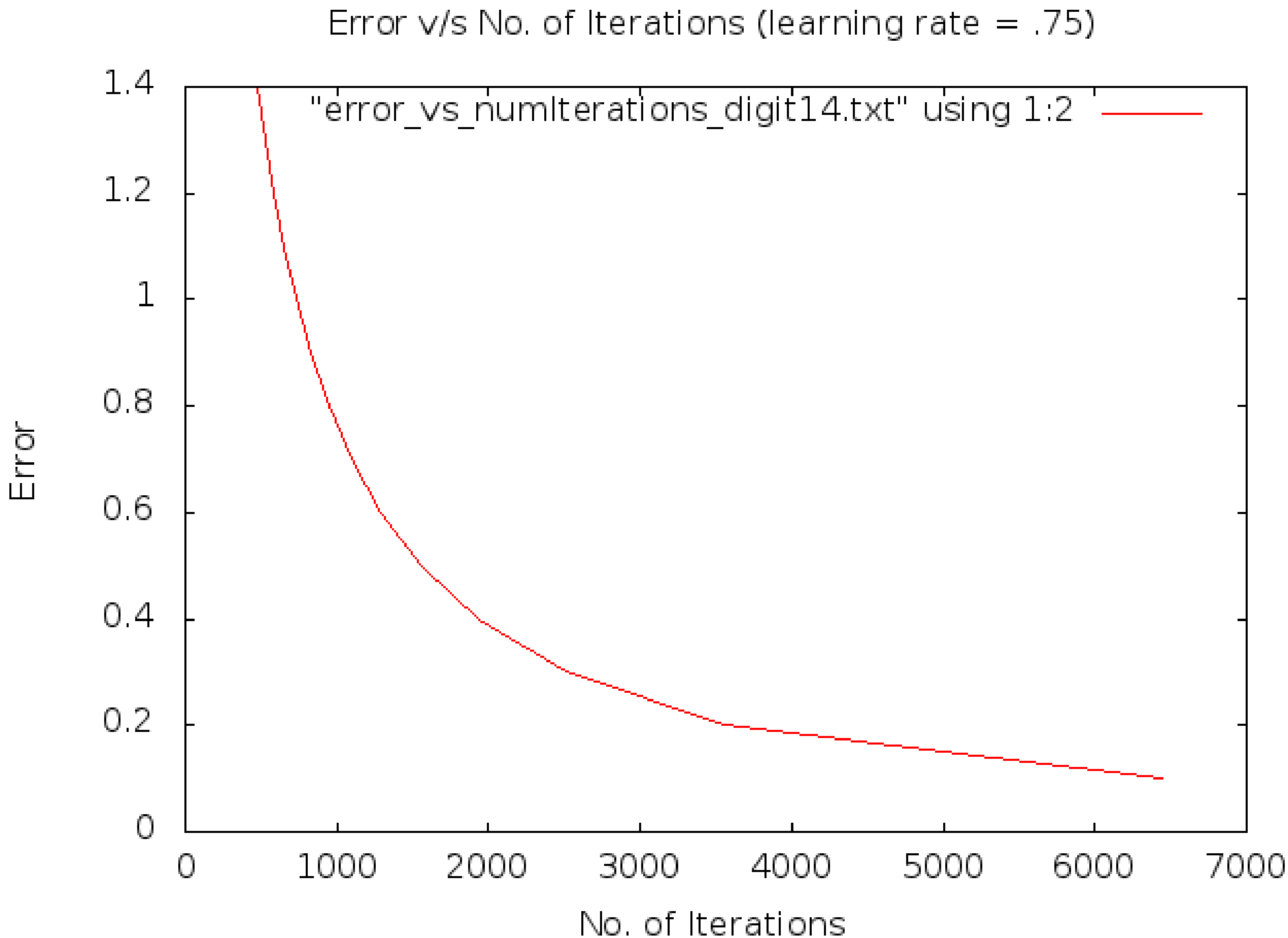
Effect of Learning Rate (Digit Recogniser)



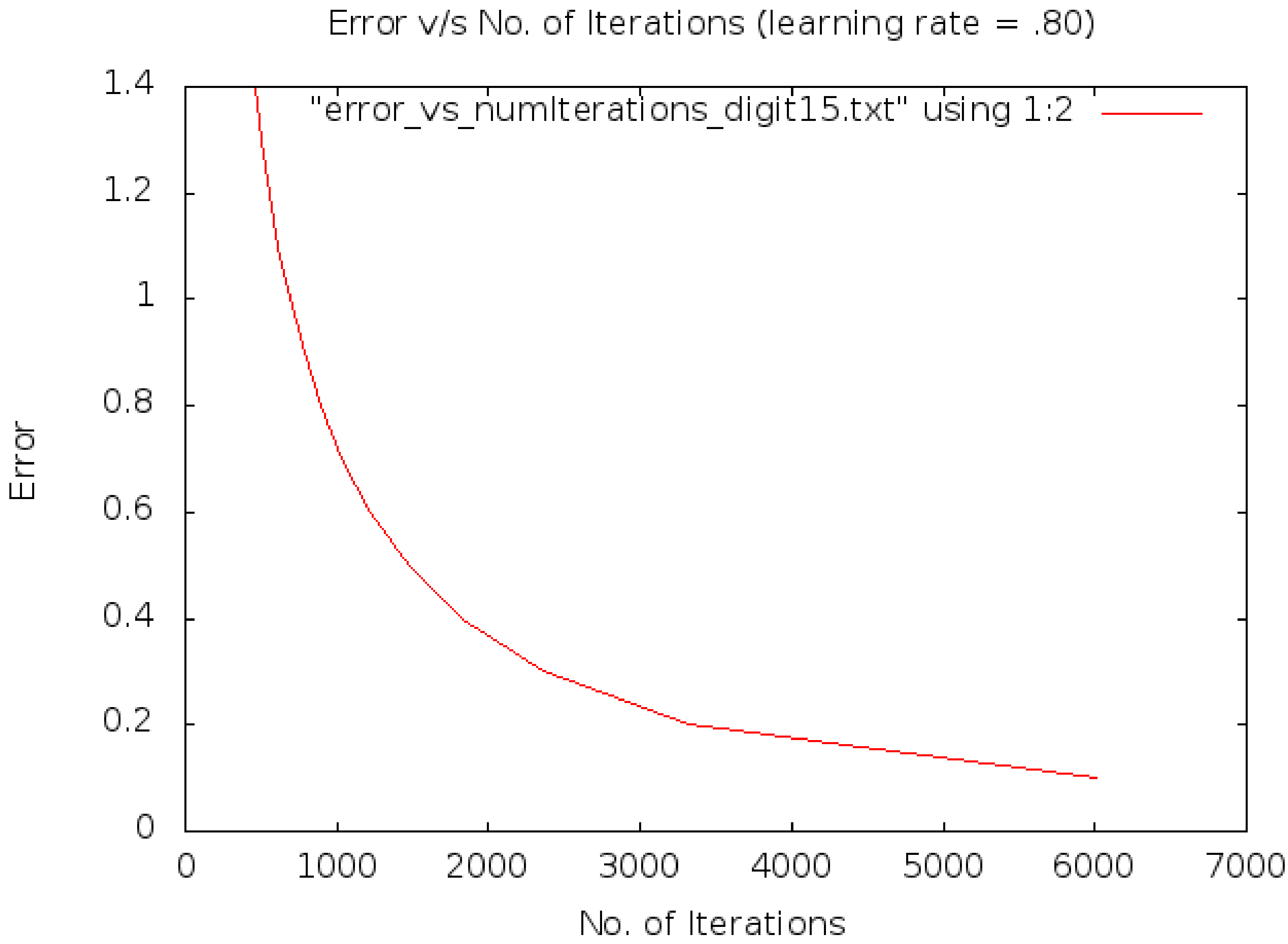
Effect of Learning Rate (Digit Recogniser)



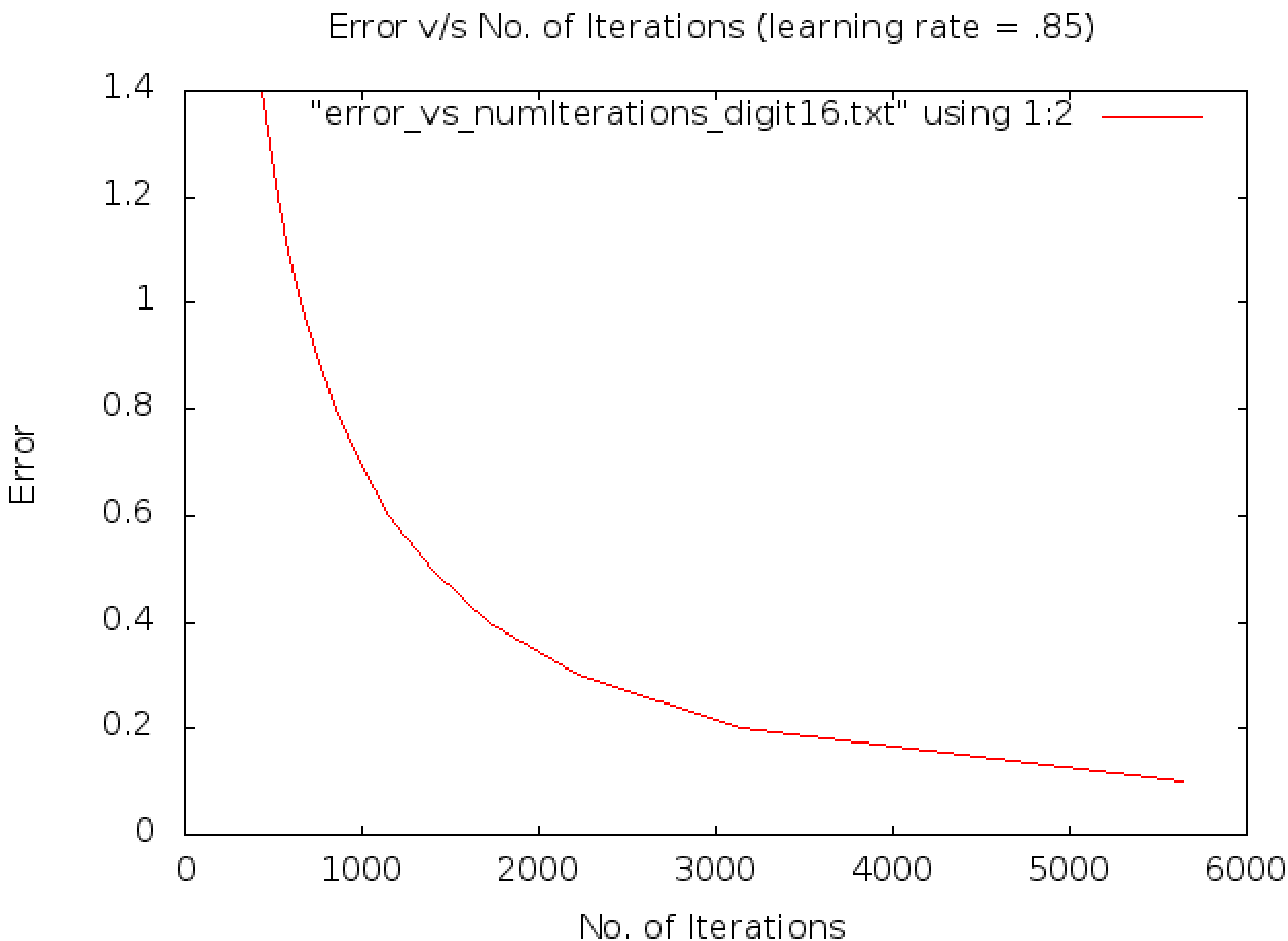
Effect of Learning Rate (Digit Recogniser)



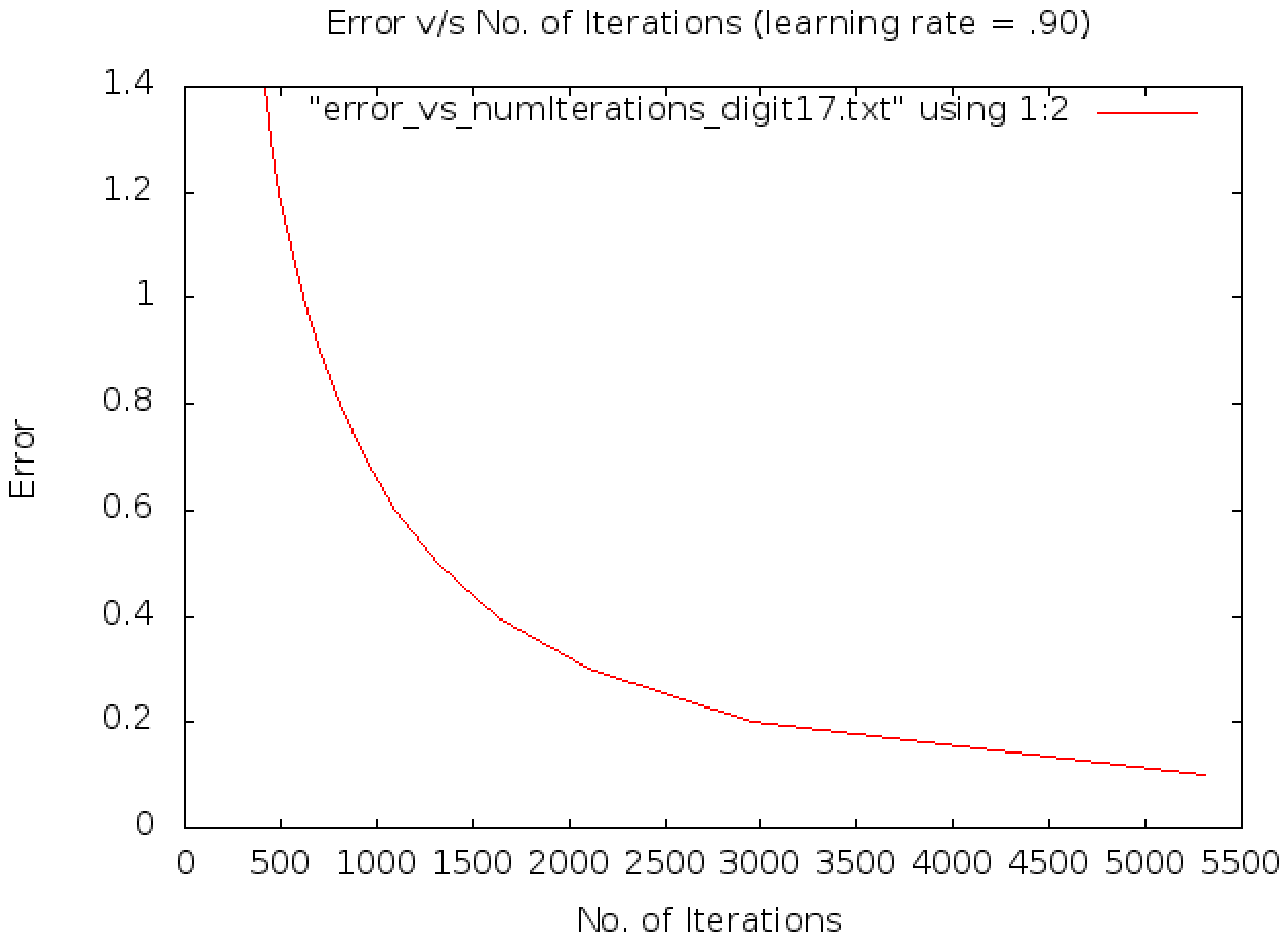
Effect of Learning Rate (Digit Recogniser)



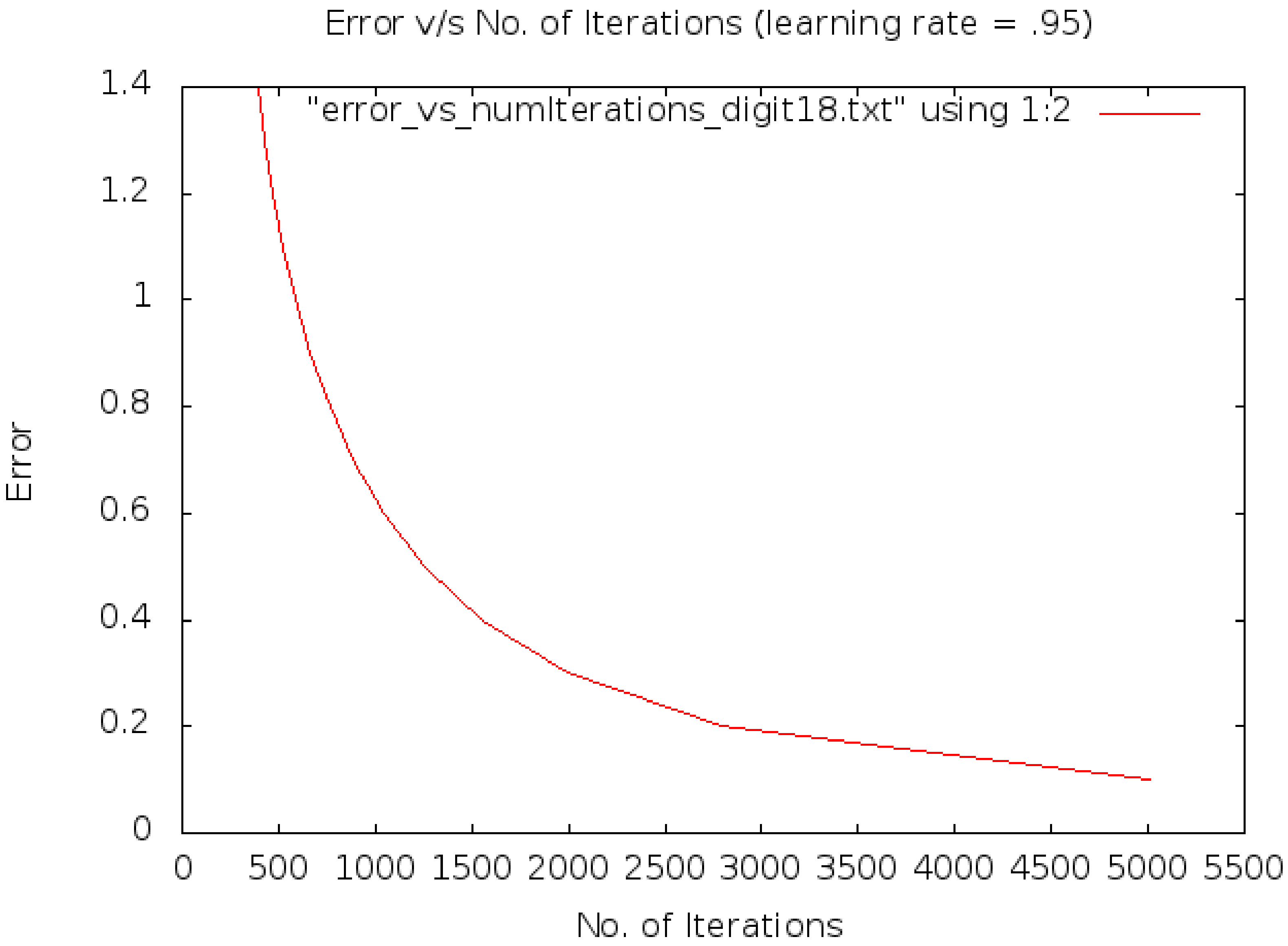
Effect of Learning Rate (Digit Recogniser)



Effect of Learning Rate (Digit Recogniser)

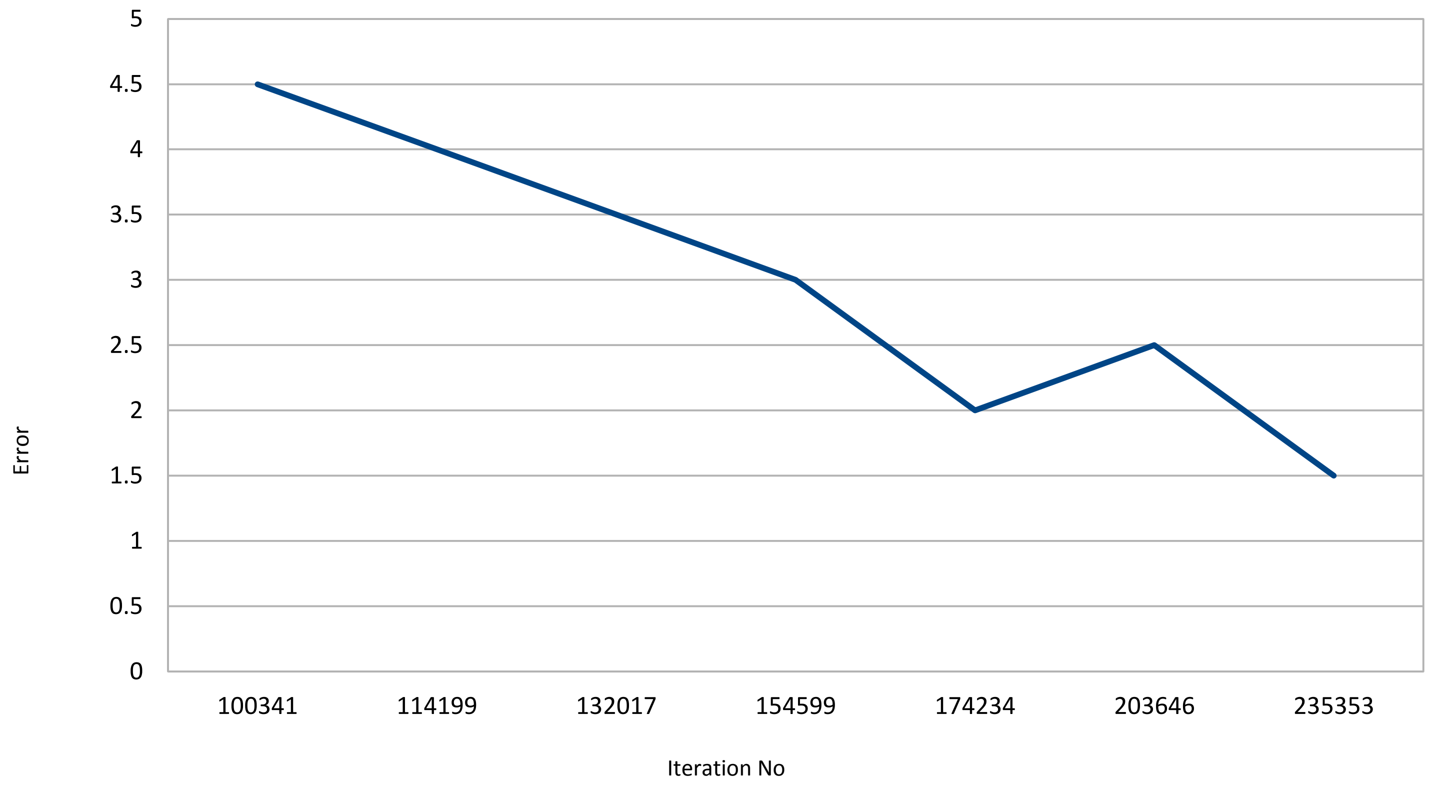


Effect of Learning Rate (Digit Recogniser)



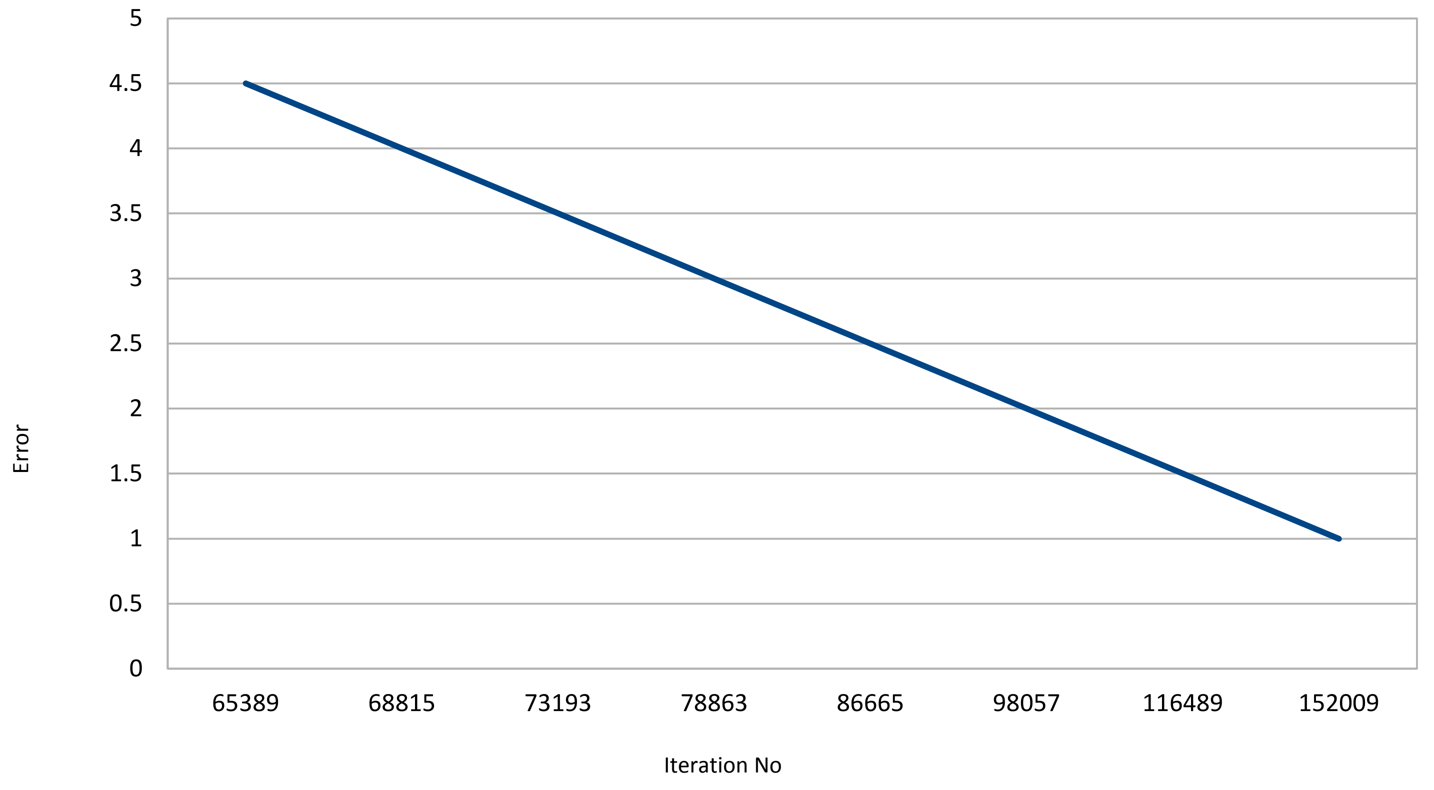
Effect of Learning Rate (Parity)

Eta=0.1 Momentum Factor=0.05

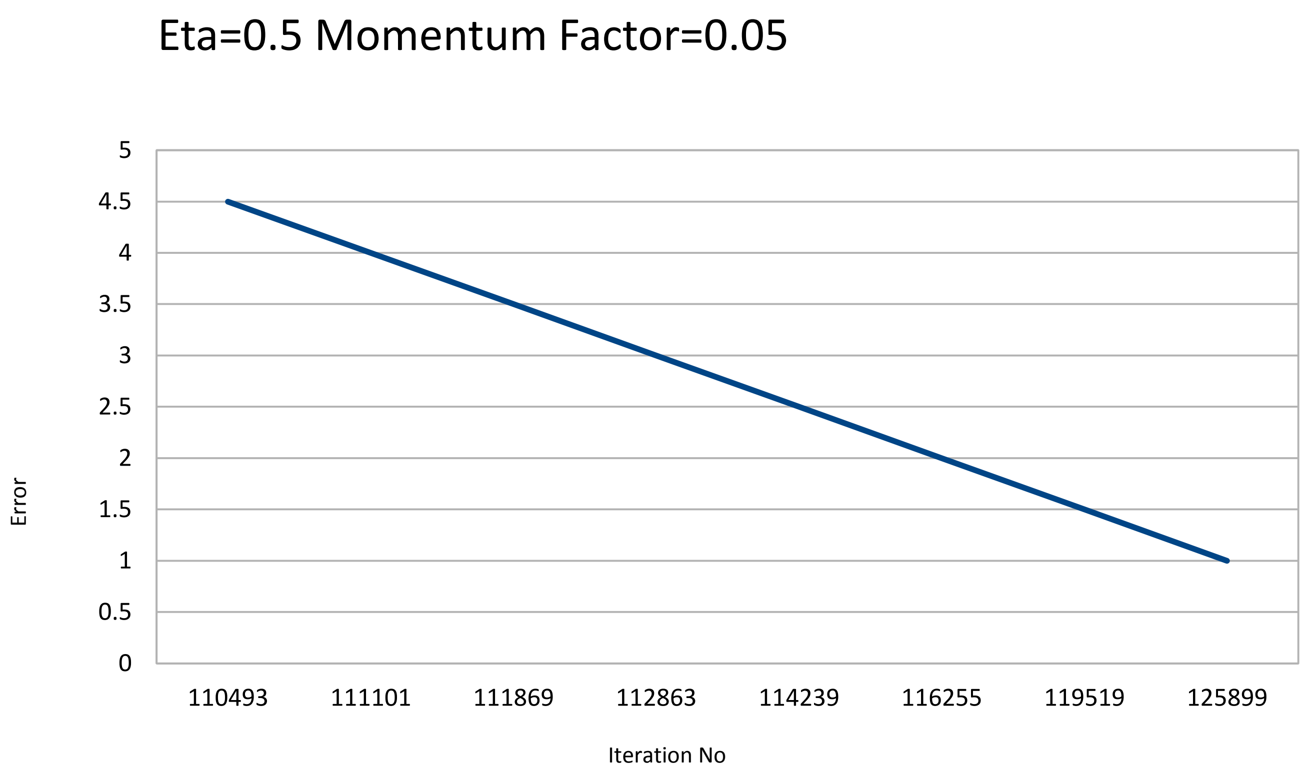


Effect of Learning Rate (Parity)

Eta=0.3 Momentum Factor=0.05

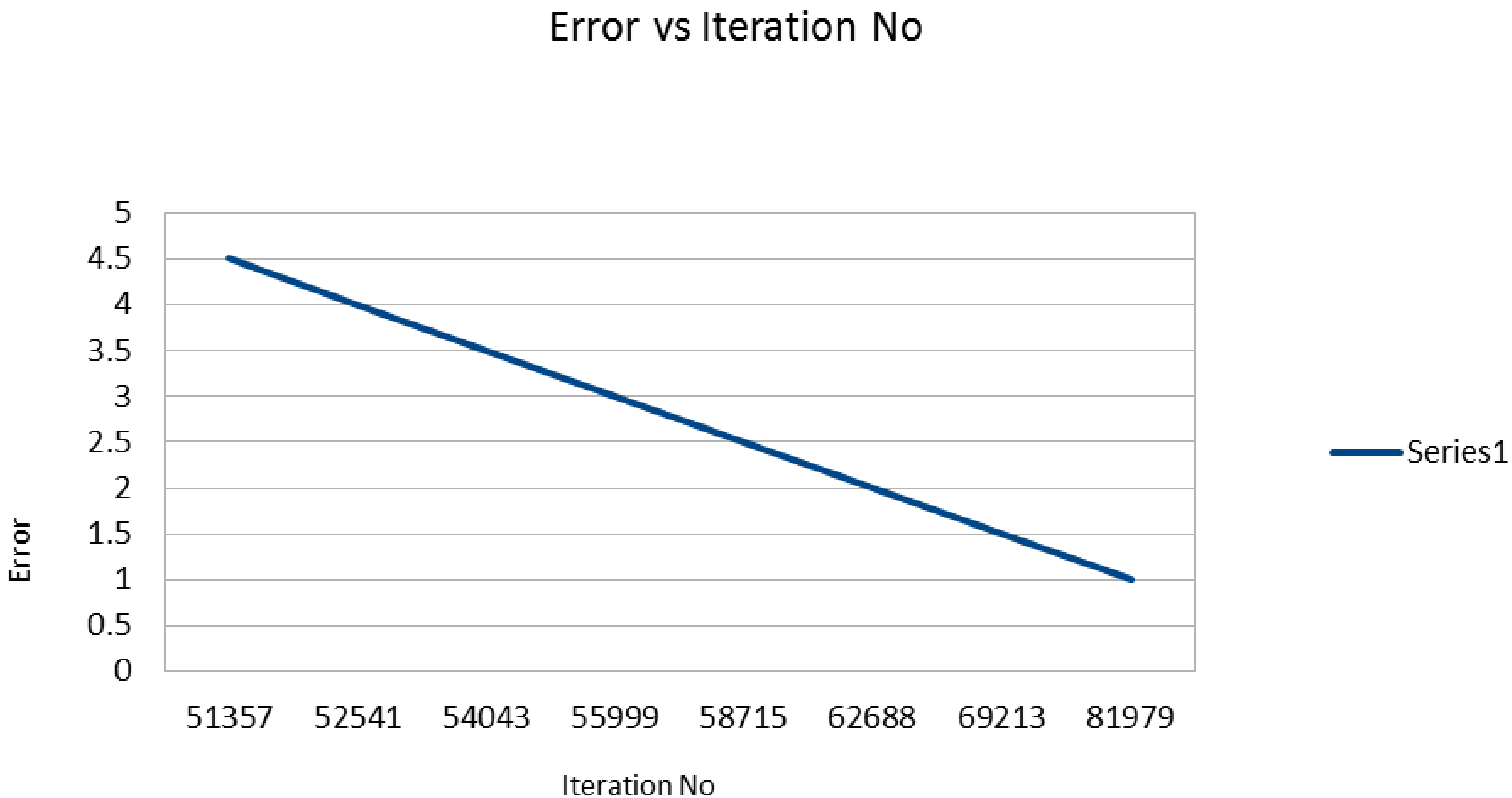


Effect of Learning Rate (Parity)



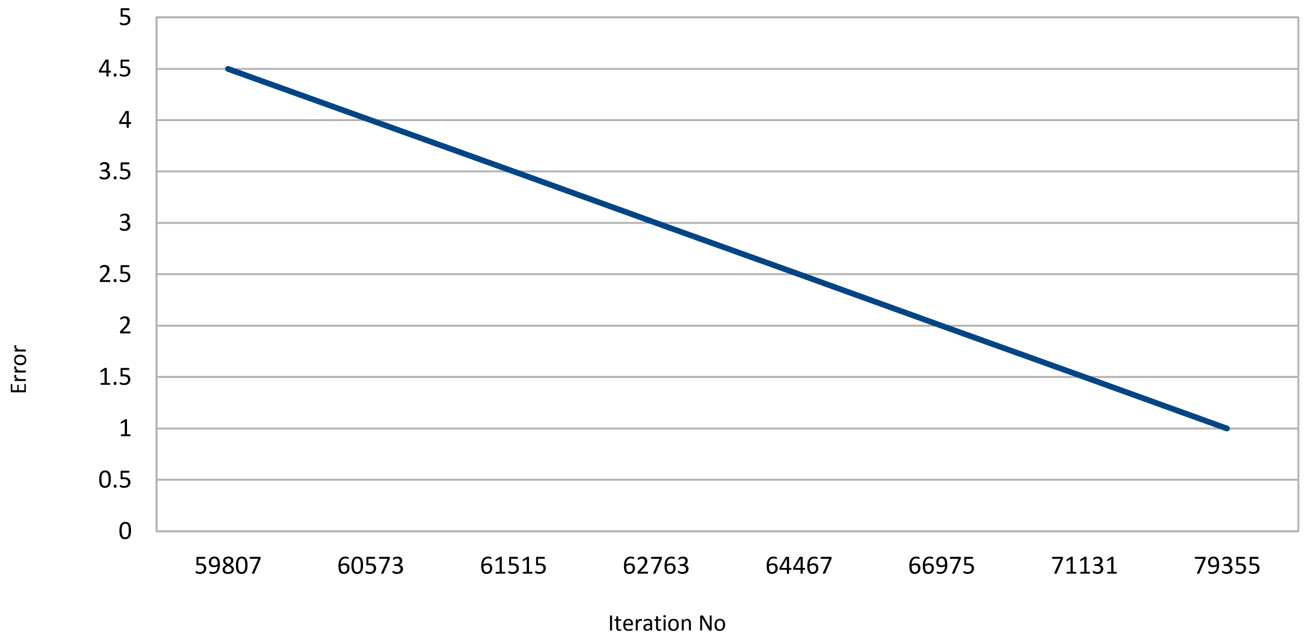
Effect of Learning Rate (Parity)

Eta=0.7, Momentum Factor=0.05

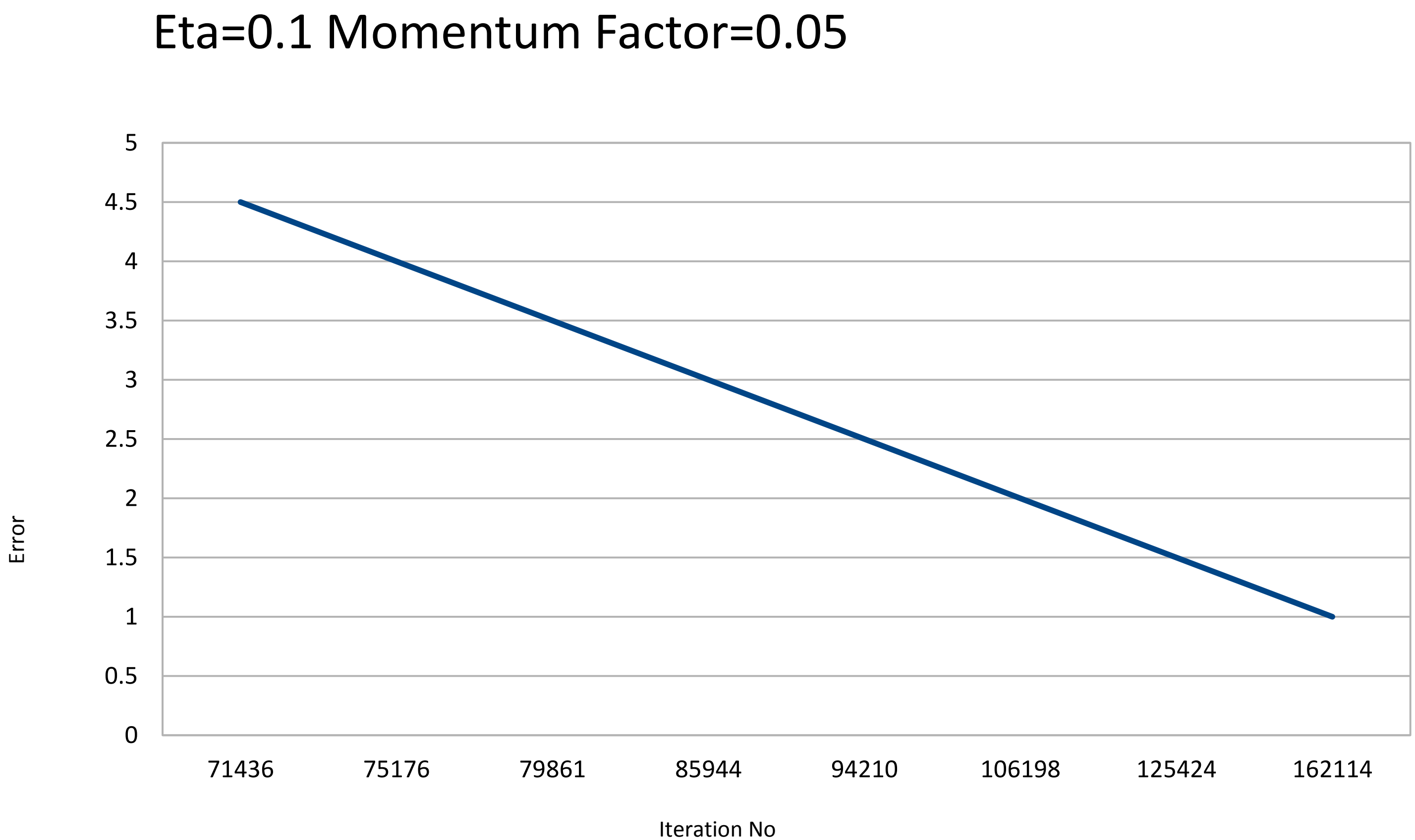


Effect of Learning Rate (Parity)

Eta=0.9, Momentum Factor=0.05

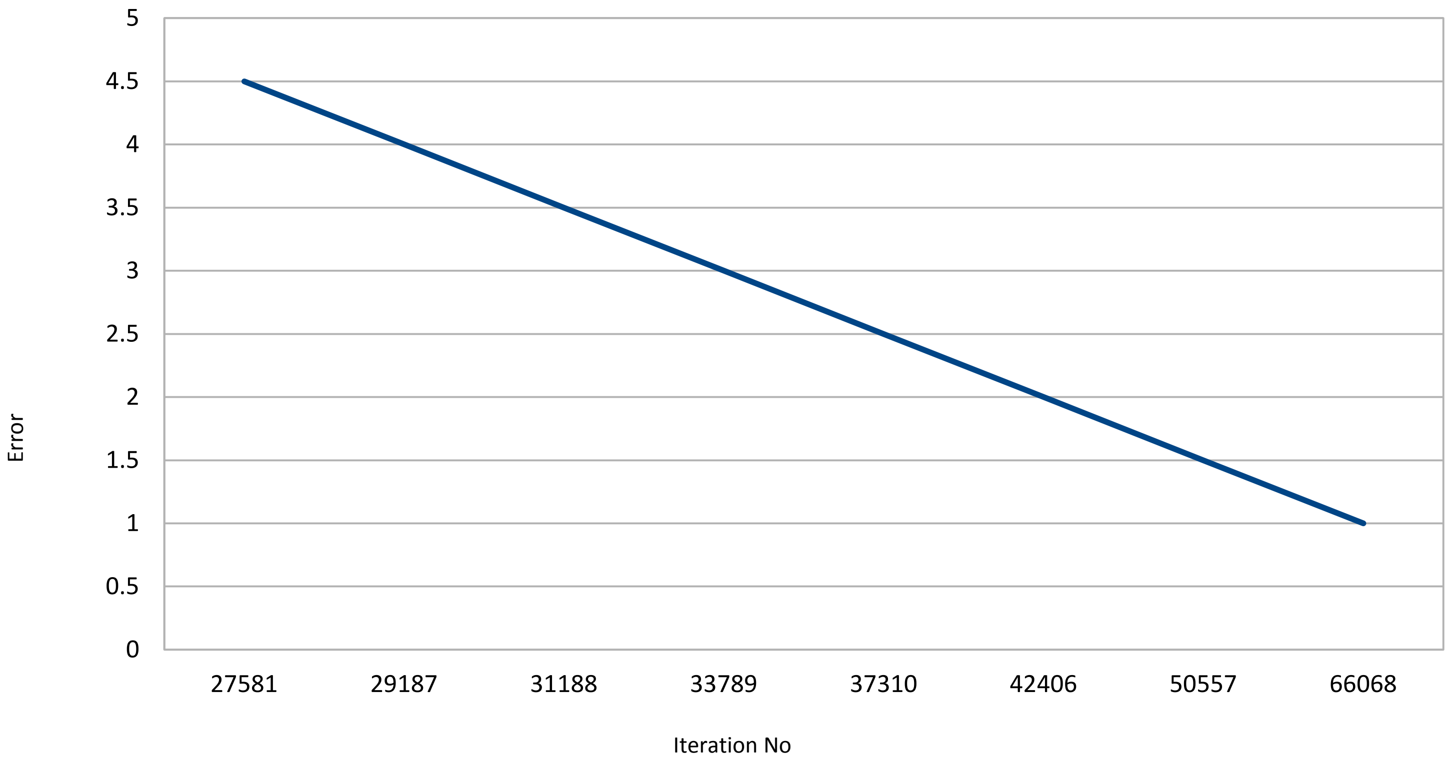


Effect of Learning Rate (Palindrome)

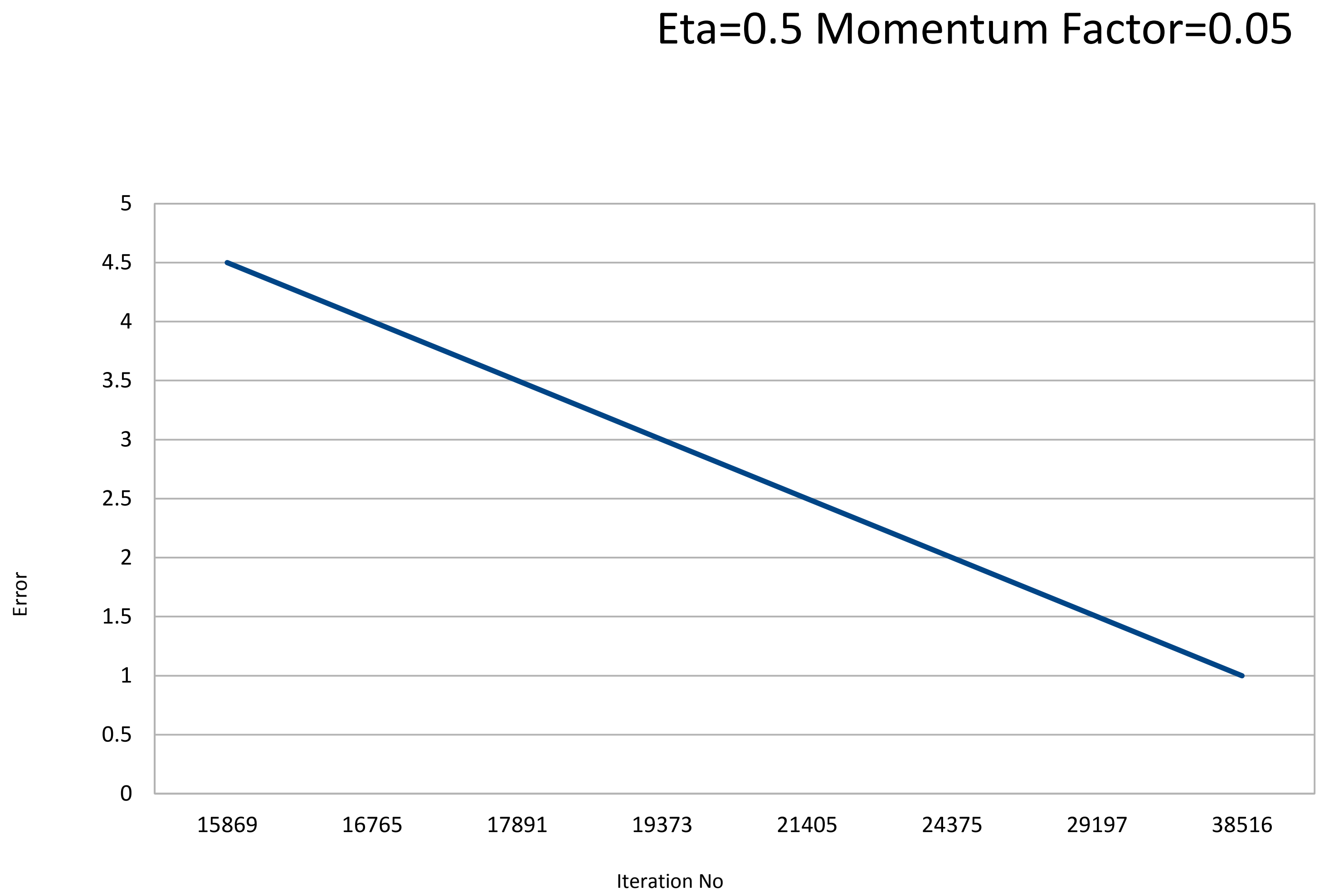


Effect of Learning Rate (Palindrome)

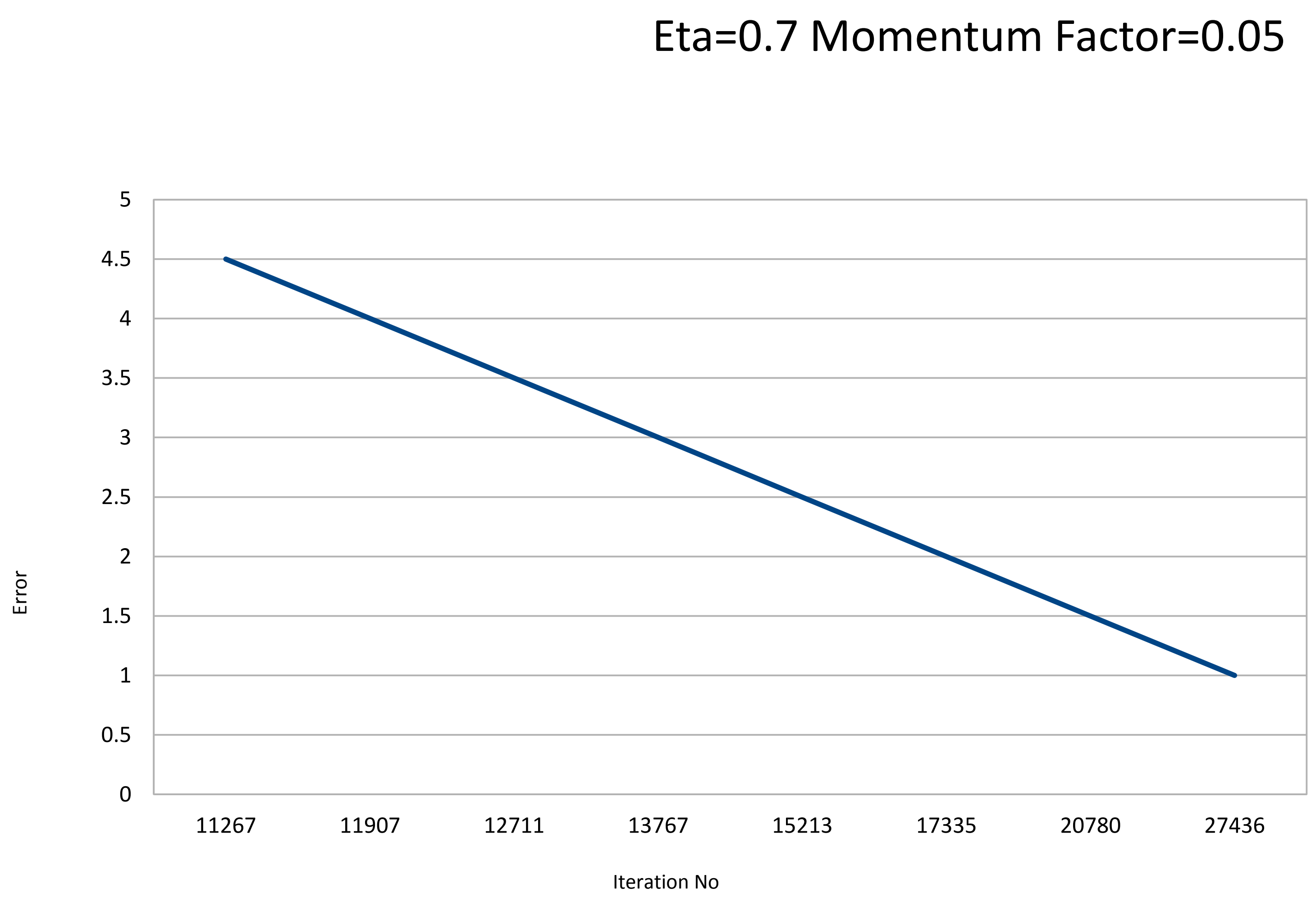
Eta=0.3 Momentum Factor=0.05



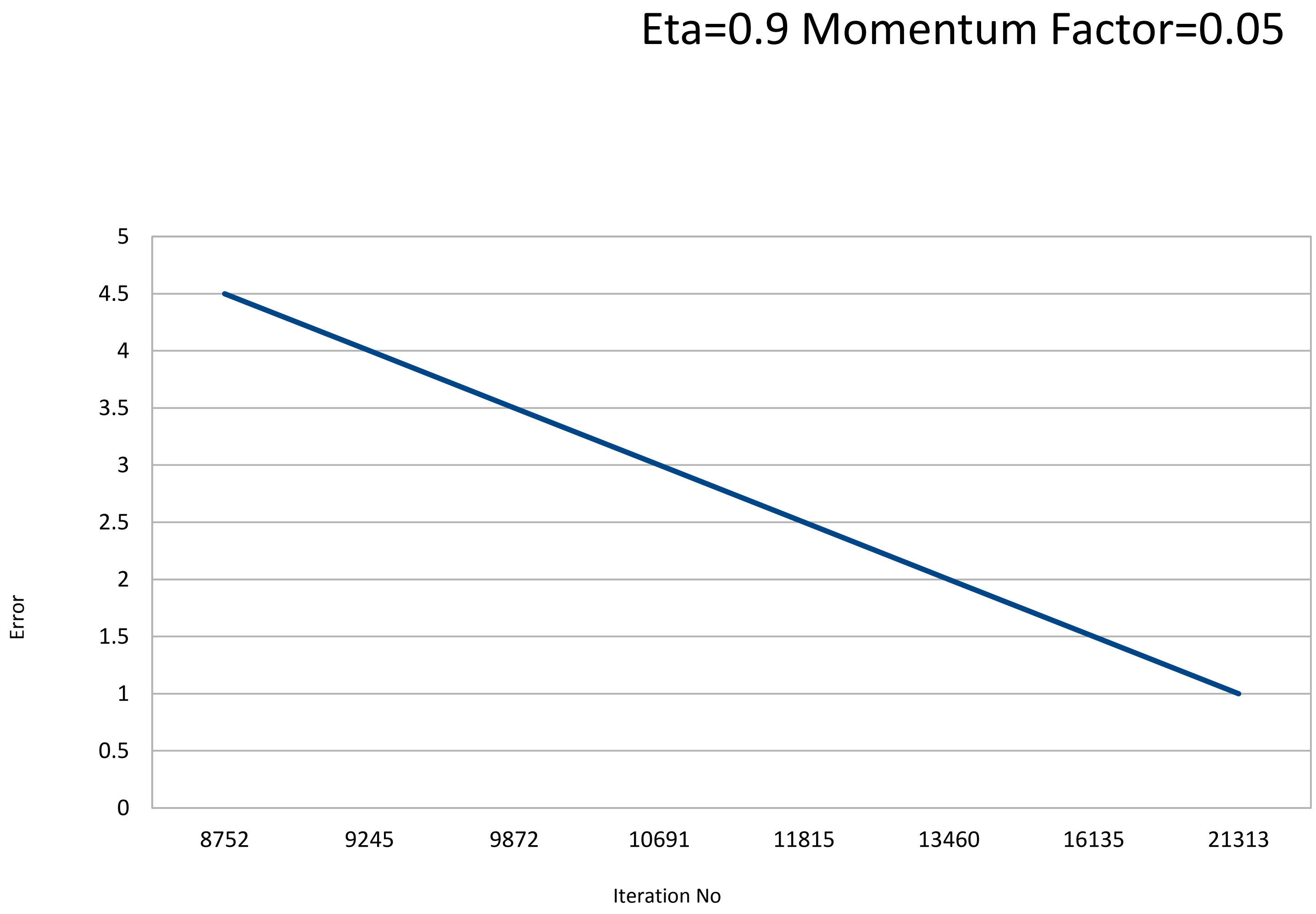
Effect of Learning Rate (Palindrome)



Effect of Learning Rate (Palindrome)



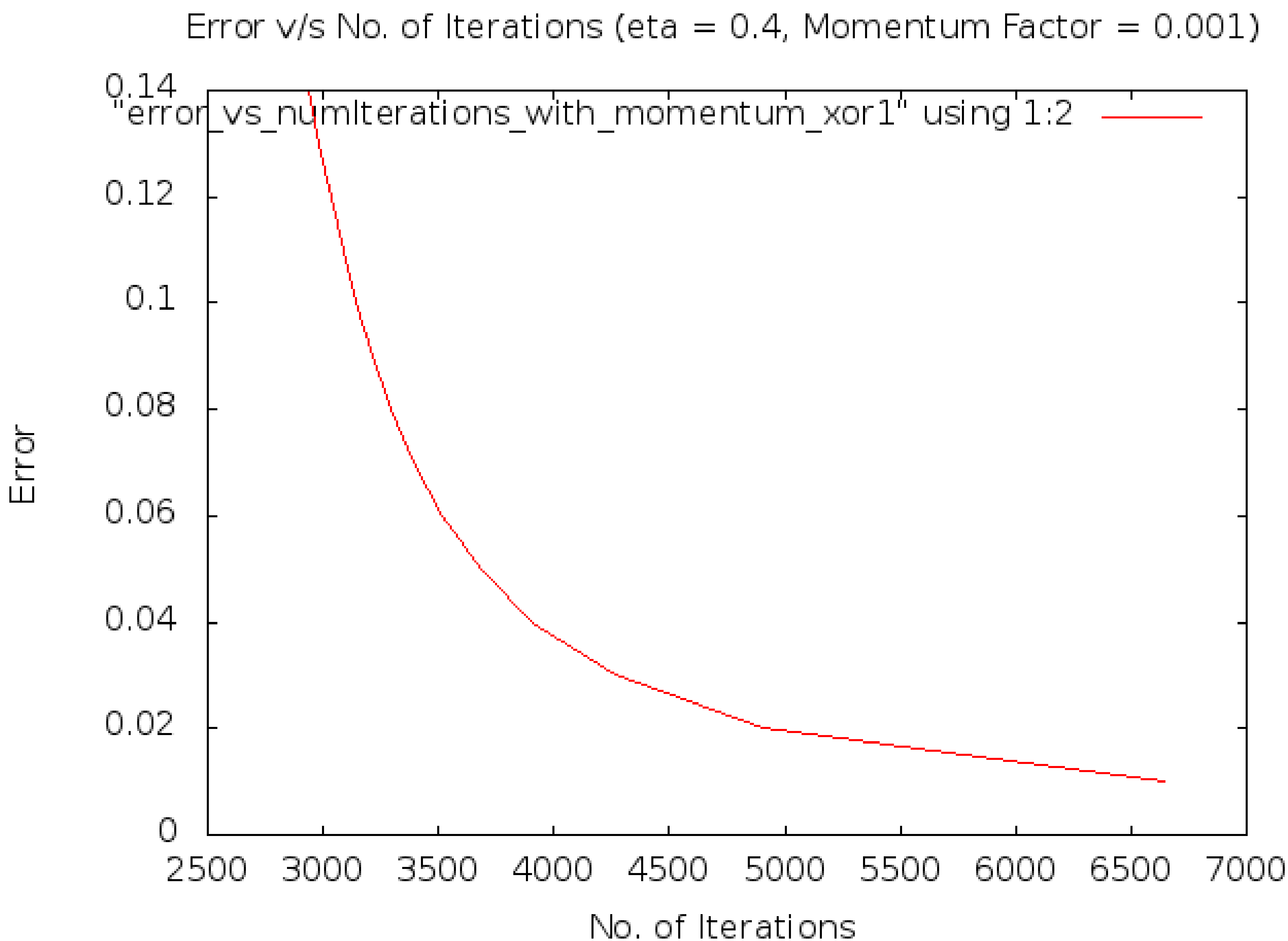
Effect of Learning Rate (Palindrome)



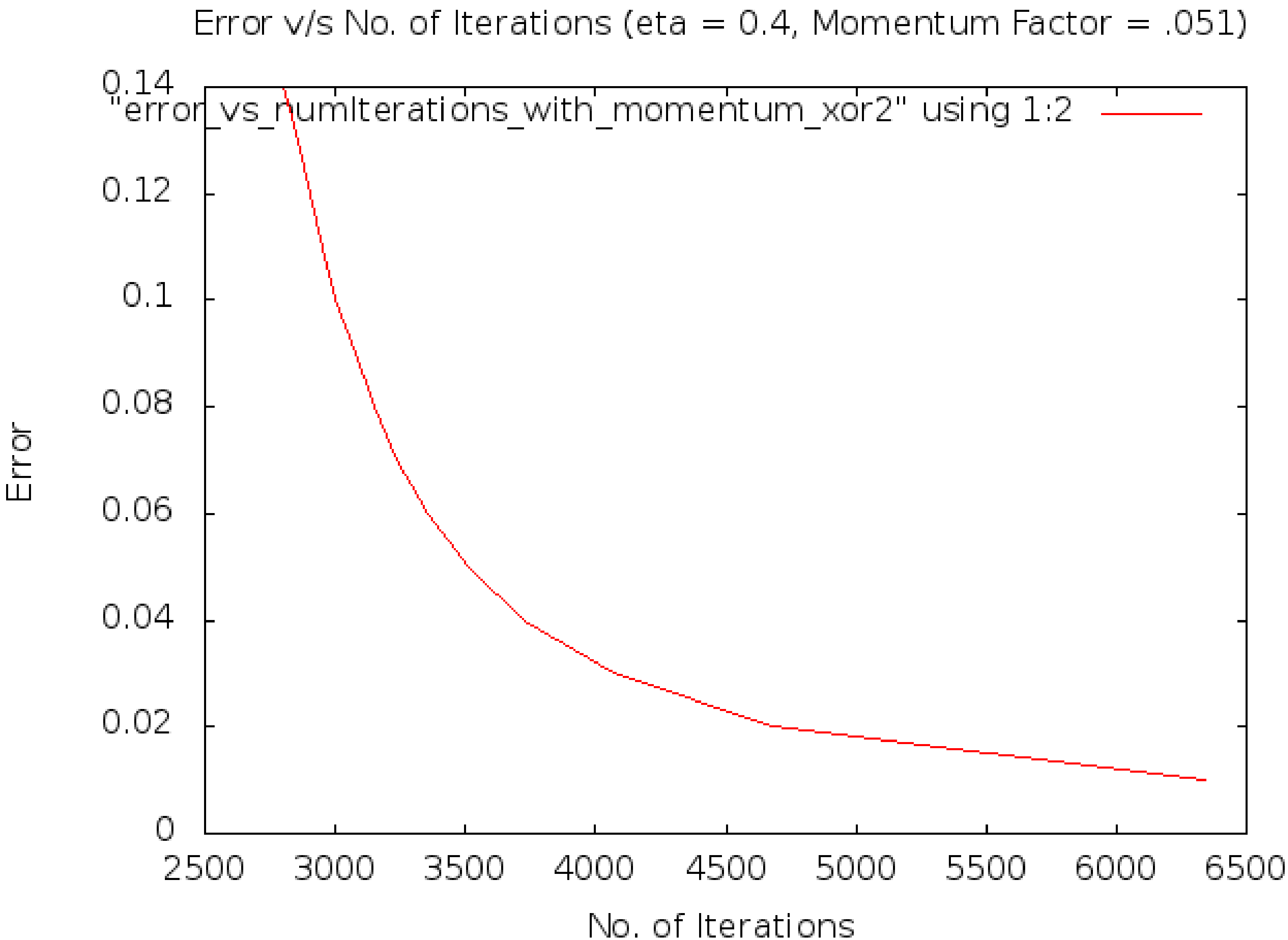
Effect of Learning Rate

- We varied the learning rate from 0.10 to 0.95 in intervals of 0.05 and observed the effect of learning rate.
- In general, as the learning rate increases, no of iterations required to converge to a solution decreases.
- The speed of convergence of a network can be improved by increasing the learning rate.
- Unfortunately, increasing learning rate usually results in increasing network instability, with weight values oscillating erratically as they converge on a solution.
- Higher values of learning rate may provide faster convergence on a solution, but may also increase instability and may lead to a failure to converge.

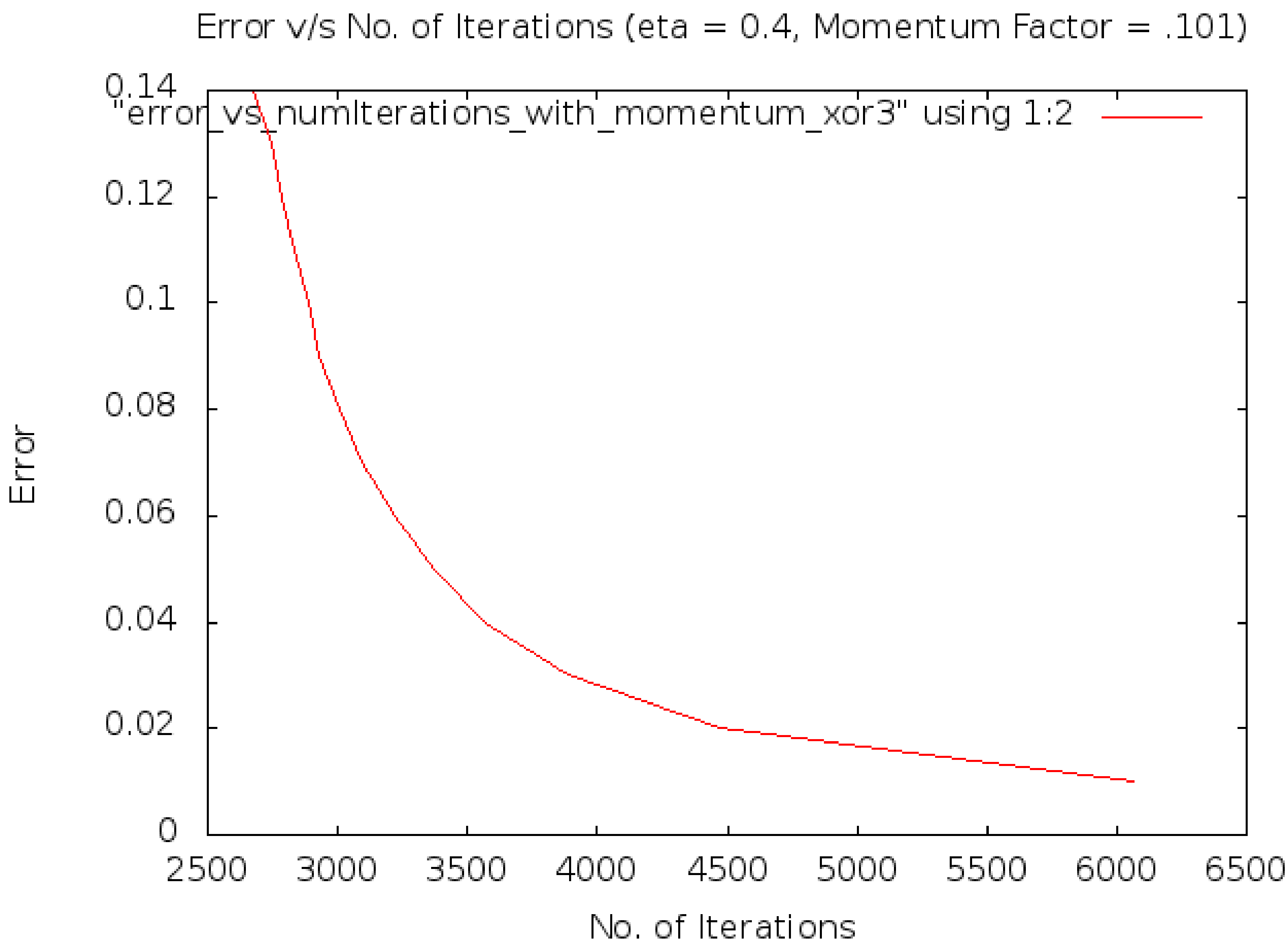
Effect of Momentum Factor (XOR)



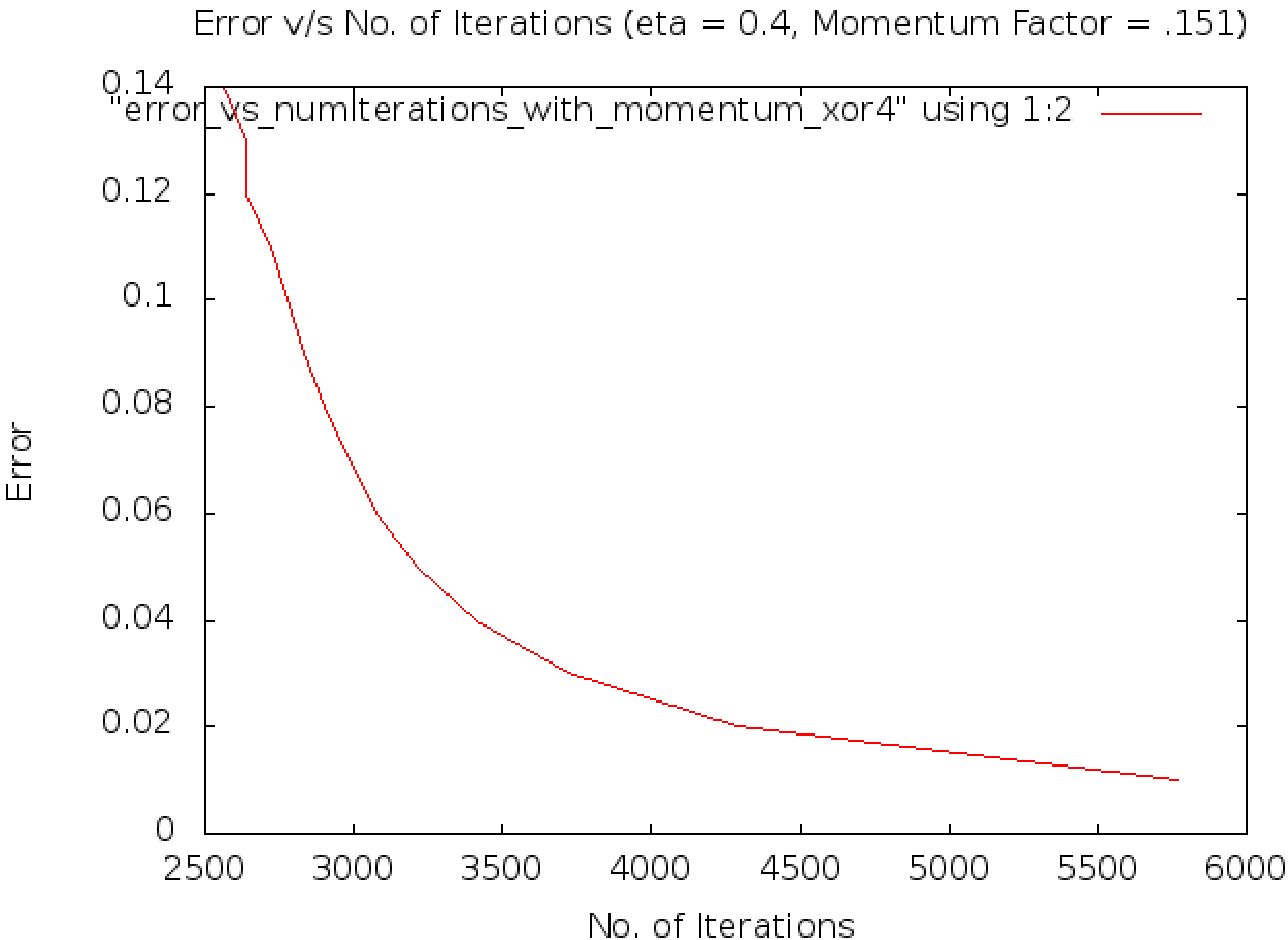
Effect of Momentum Factor (XOR)



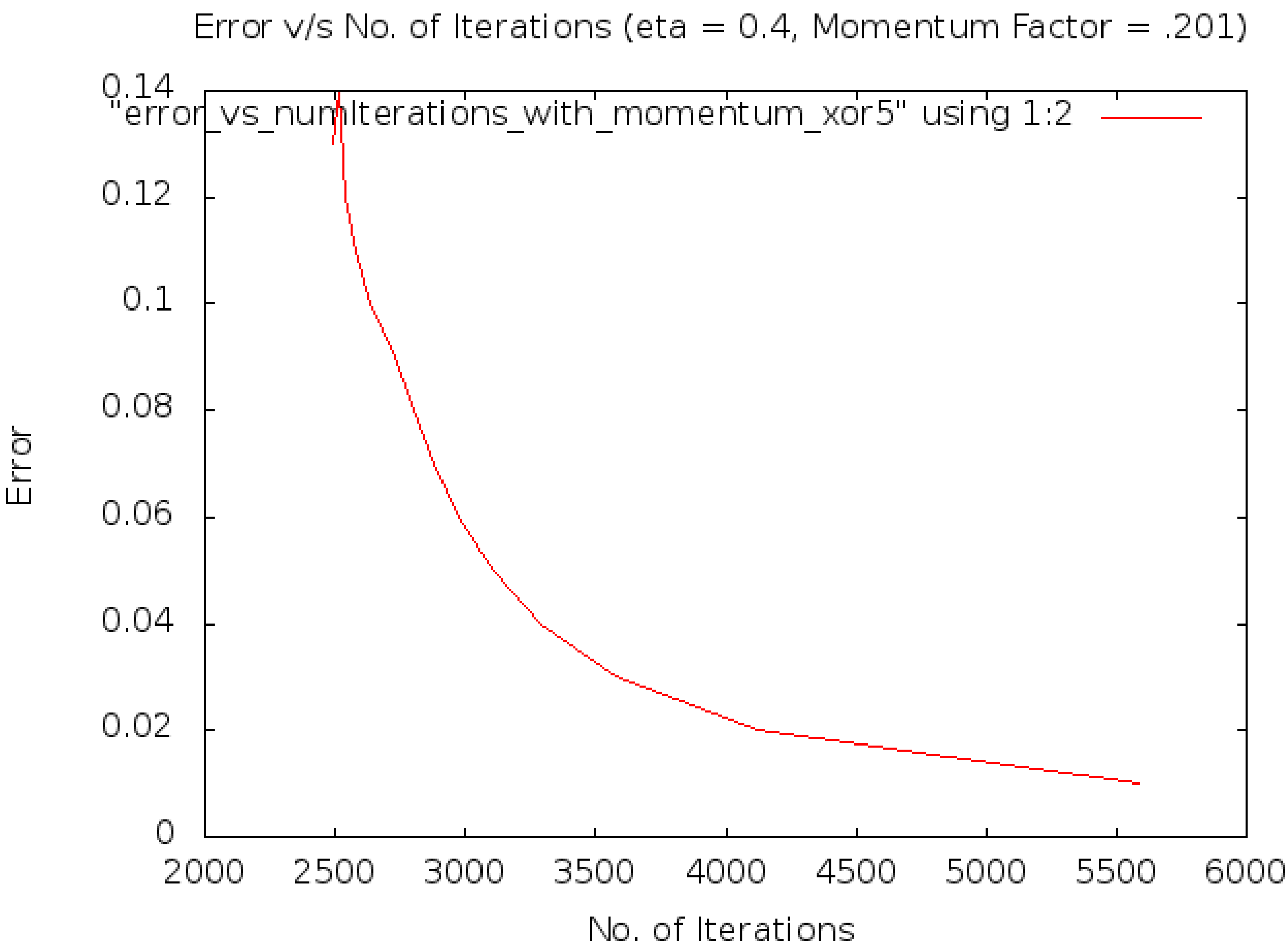
Effect of Momentum Factor (XOR)



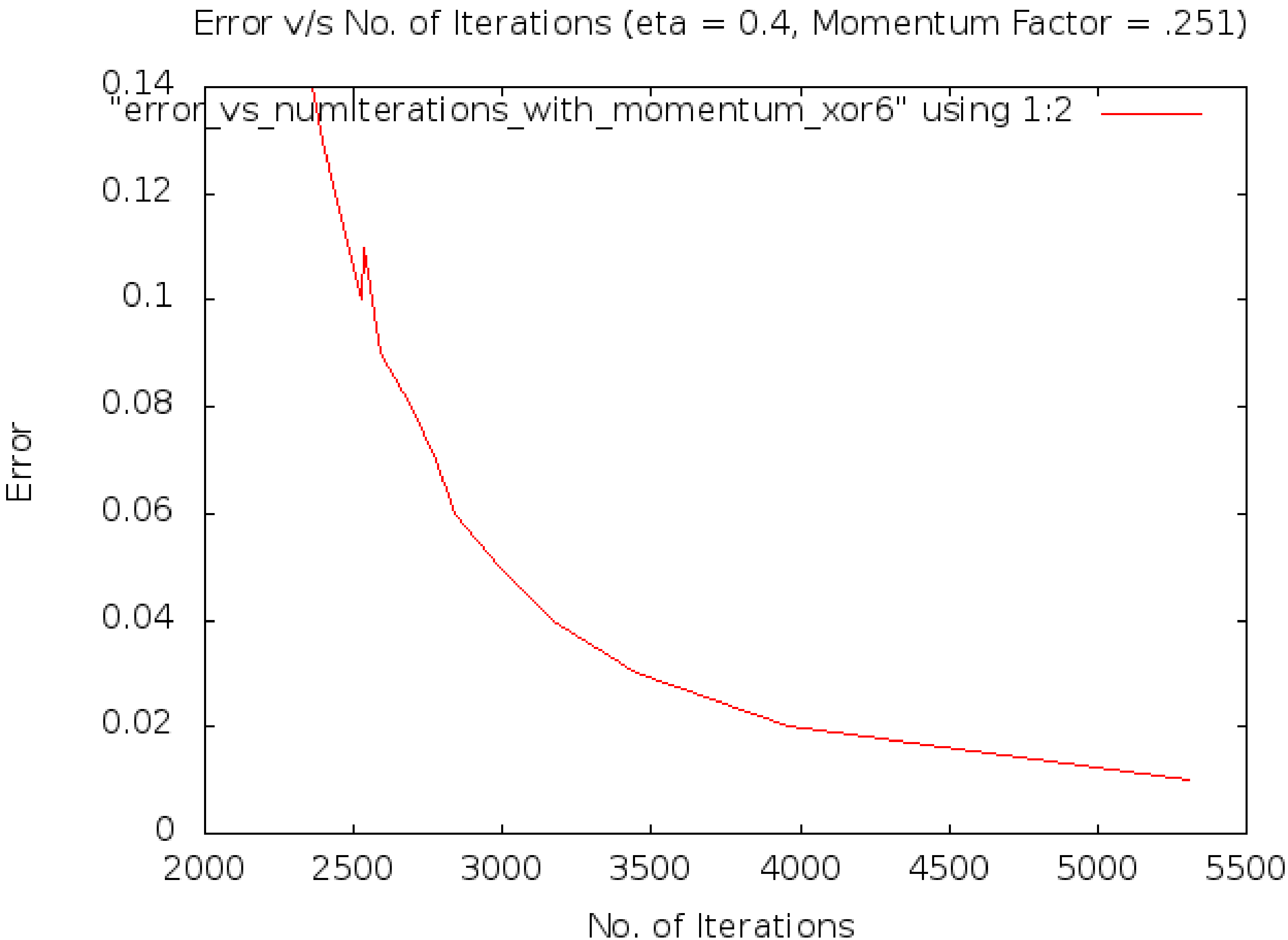
Effect of Momentum Factor (XOR)



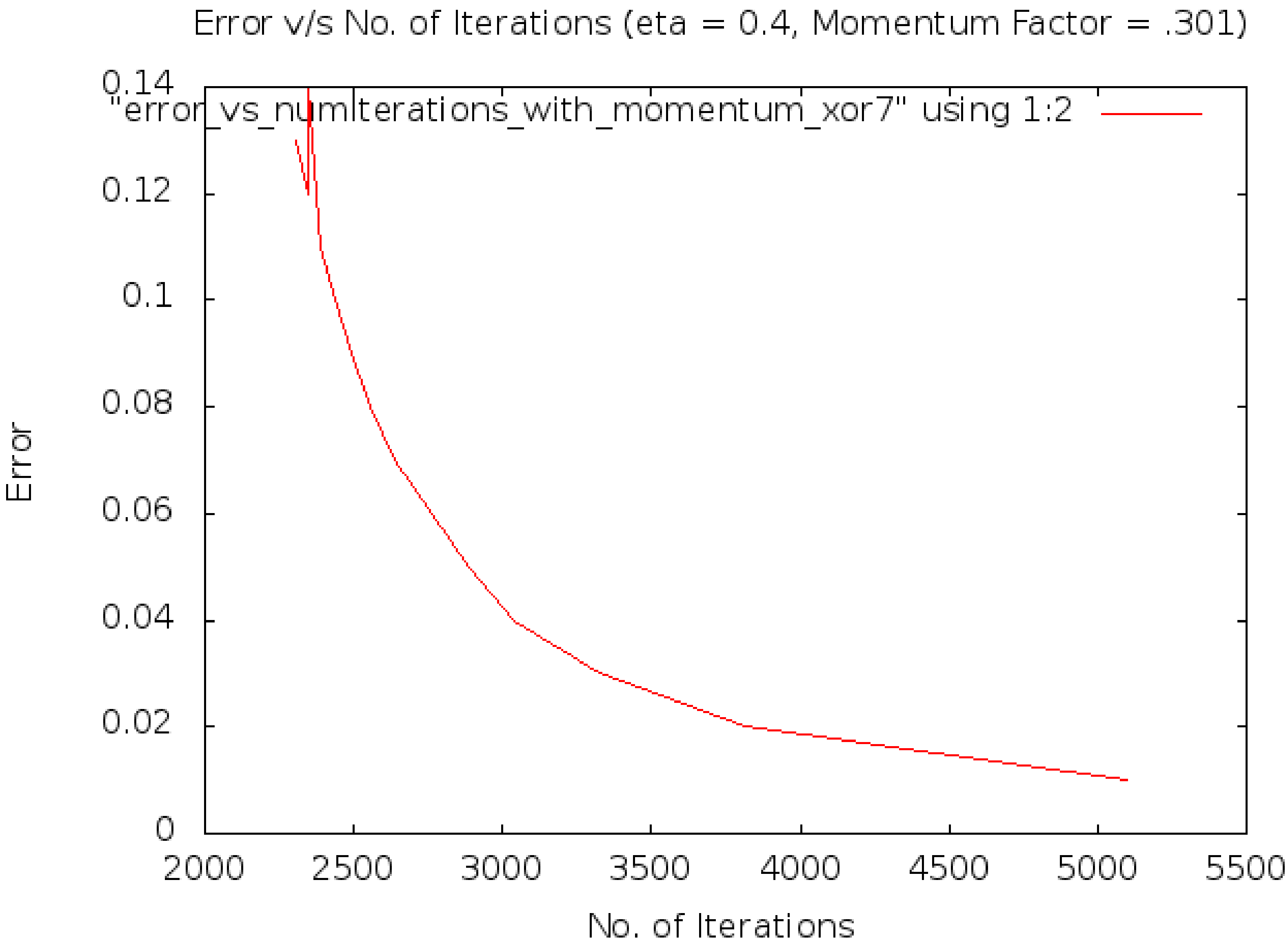
Effect of Momentum Factor (XOR)



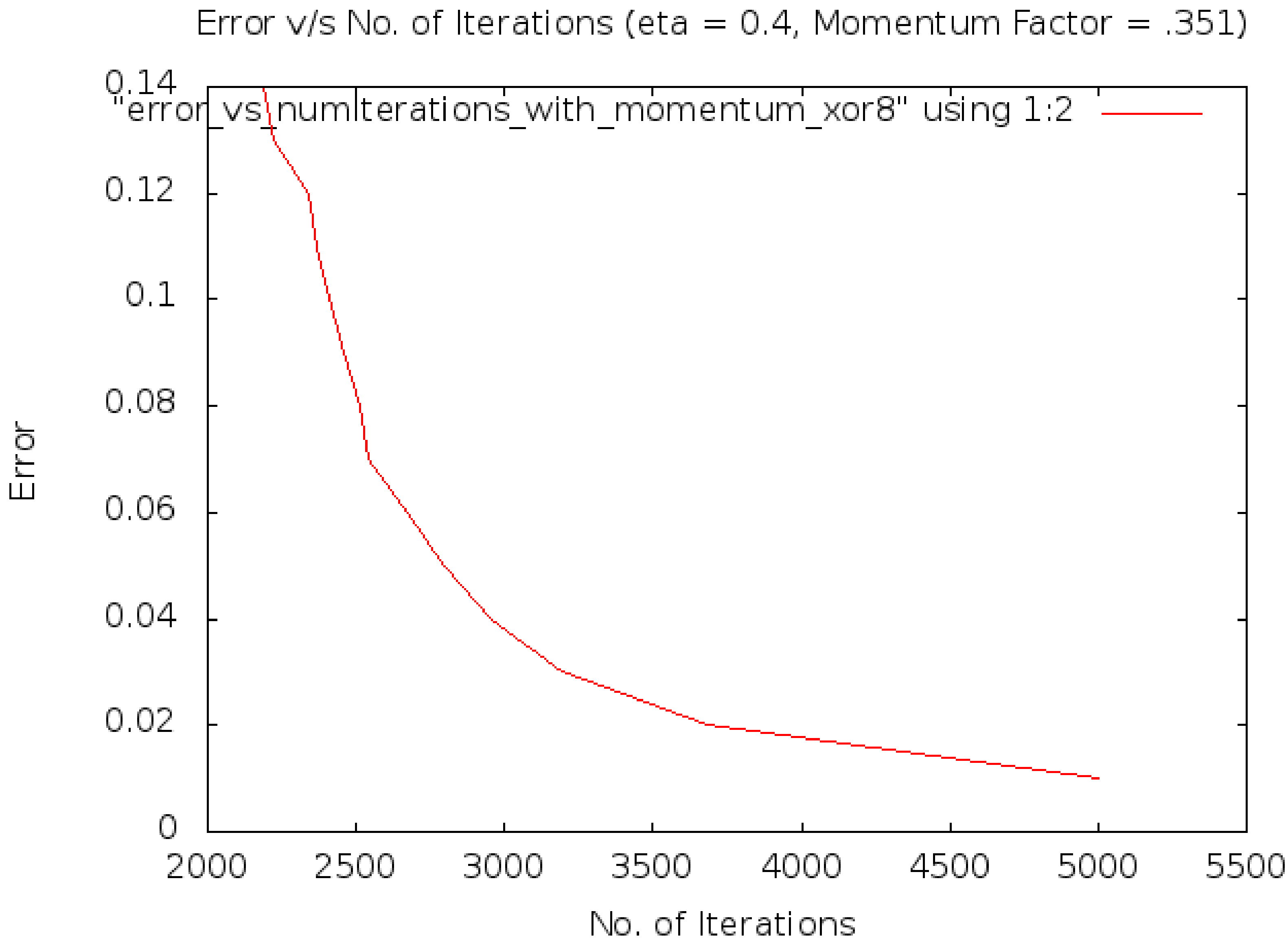
Effect of Momentum Factor (XOR)



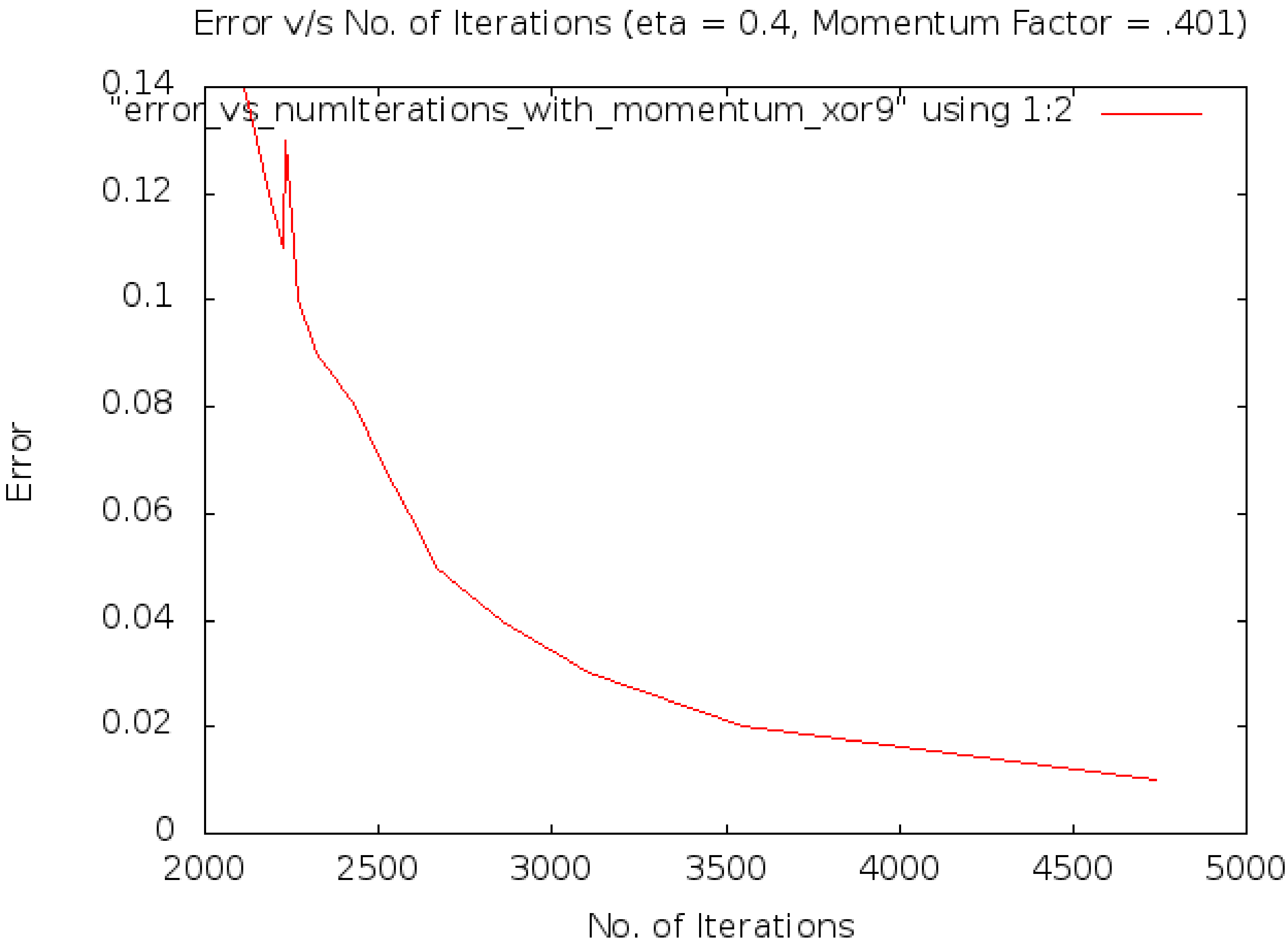
Effect of Momentum Factor (XOR)



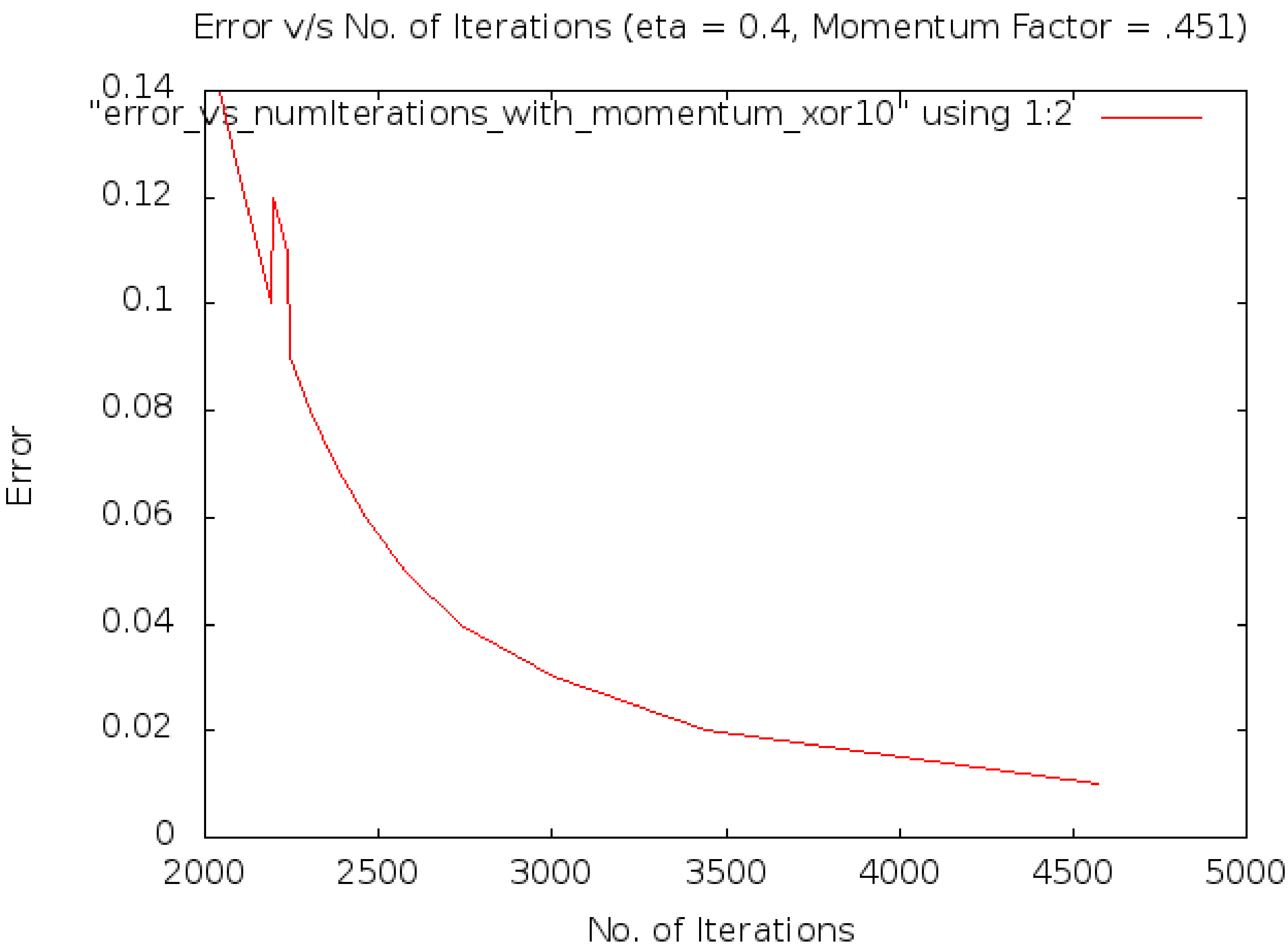
Effect of Momentum Factor (XOR)



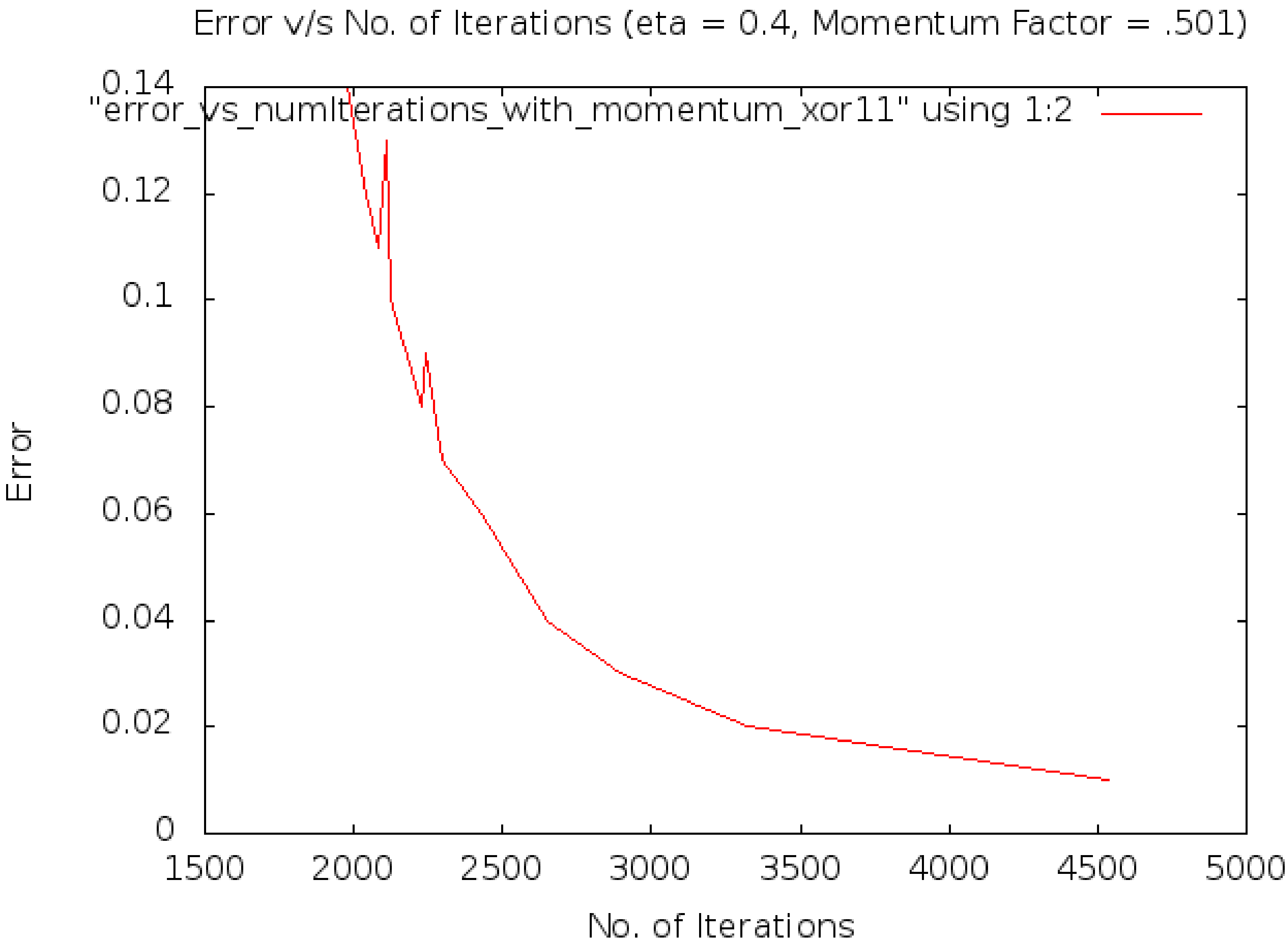
Effect of Momentum Factor (XOR)



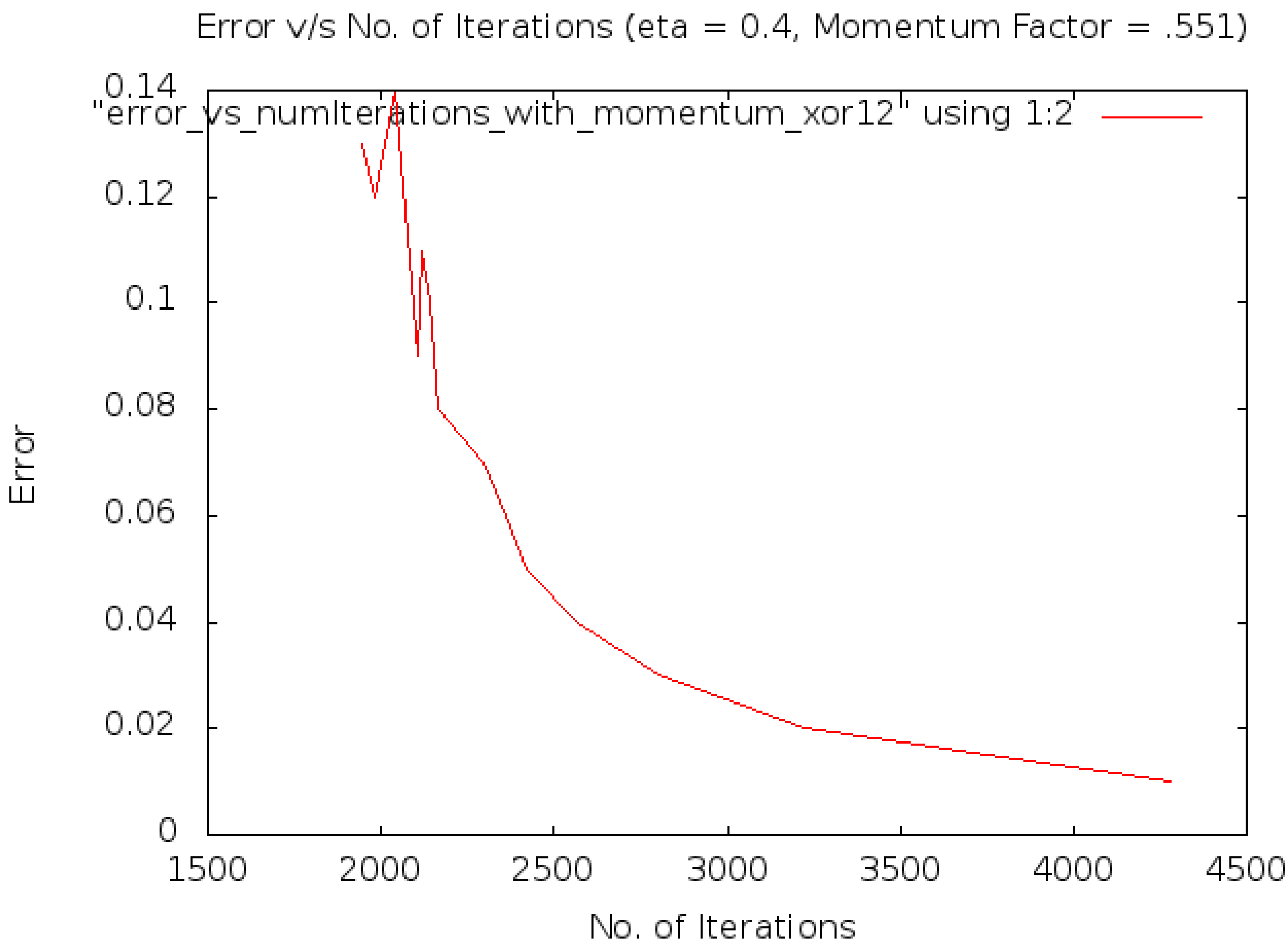
Effect of Momentum Factor (XOR)



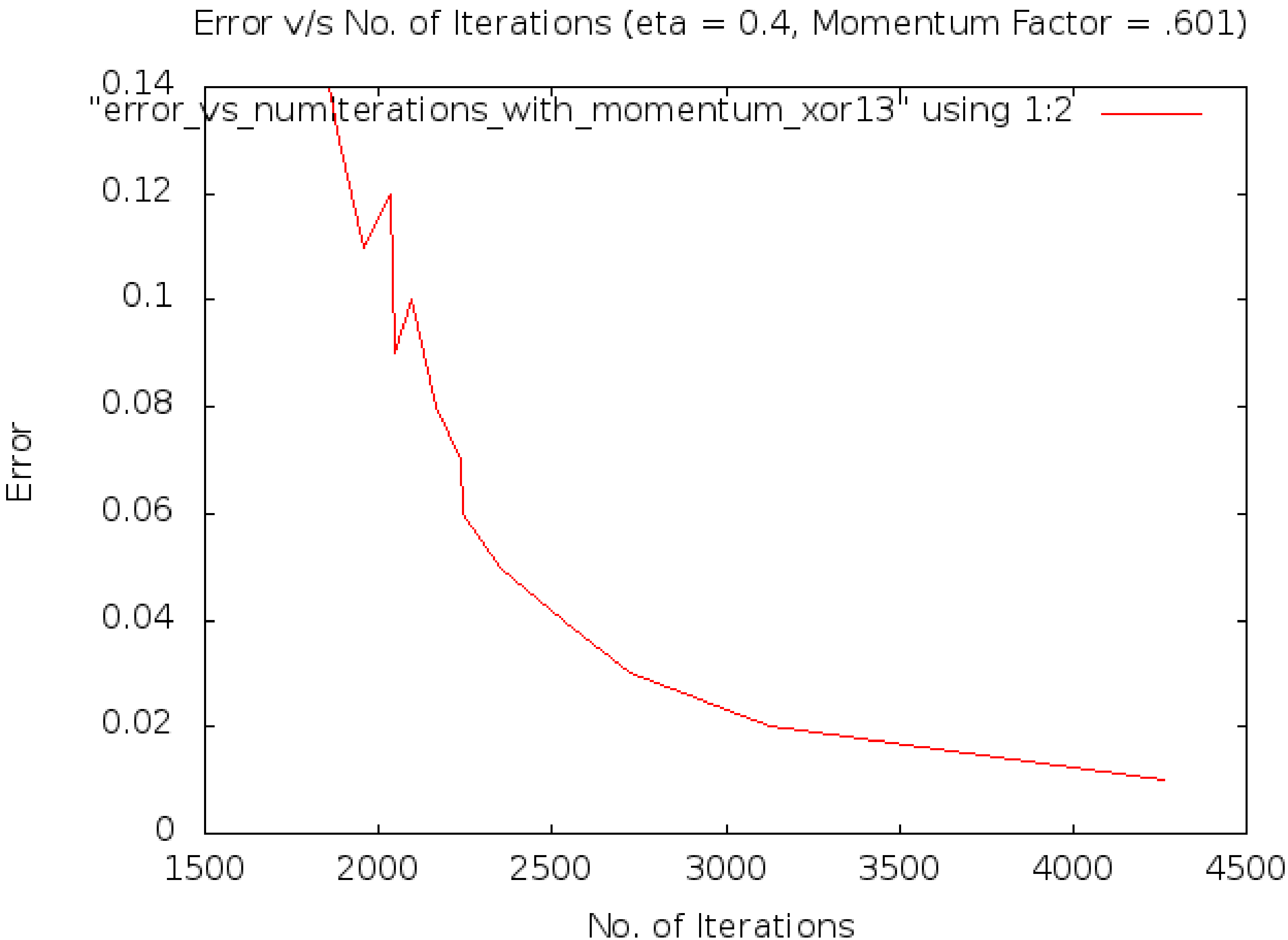
Effect of Momentum Factor (XOR)



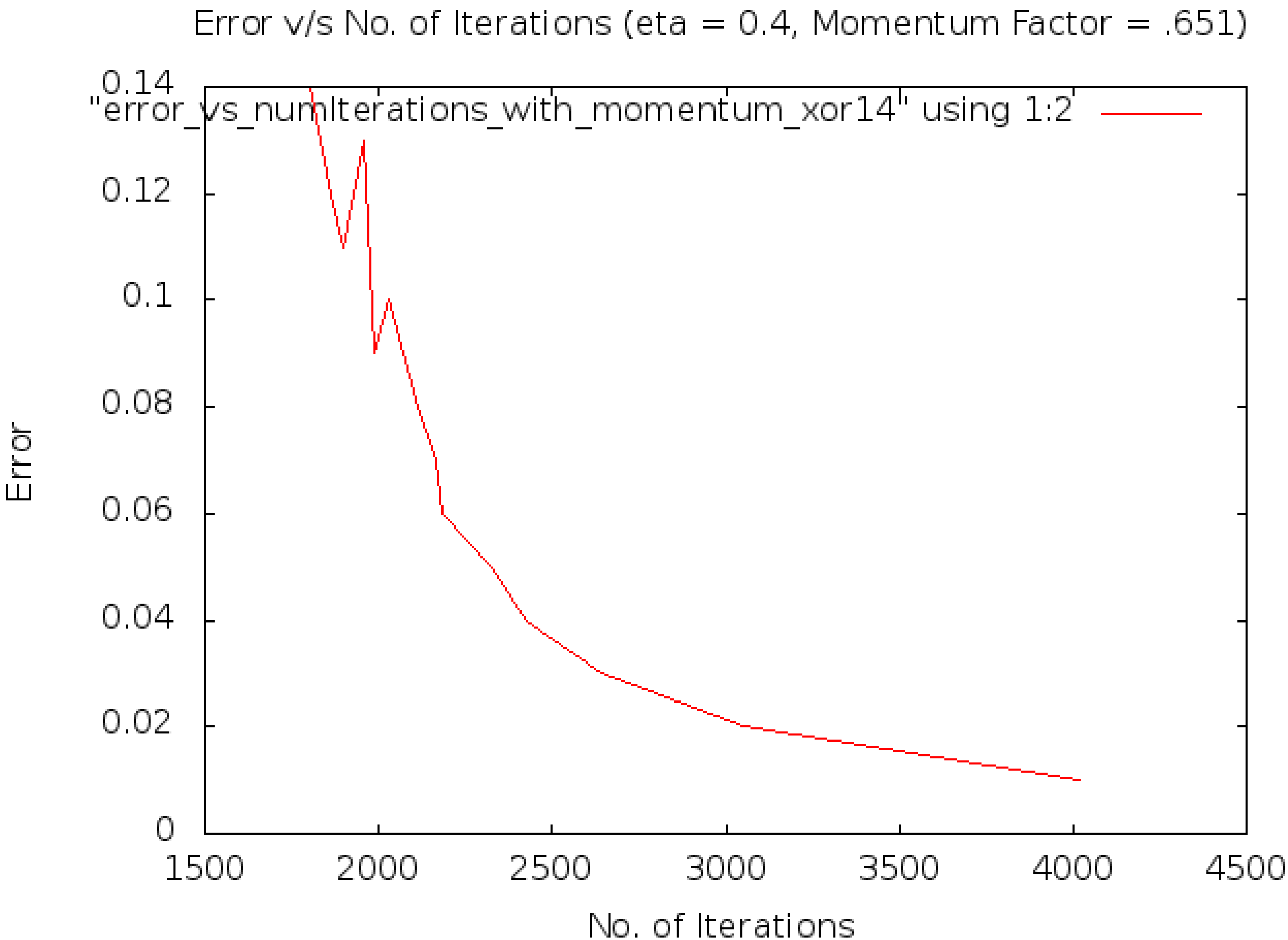
Effect of Momentum Factor (XOR)



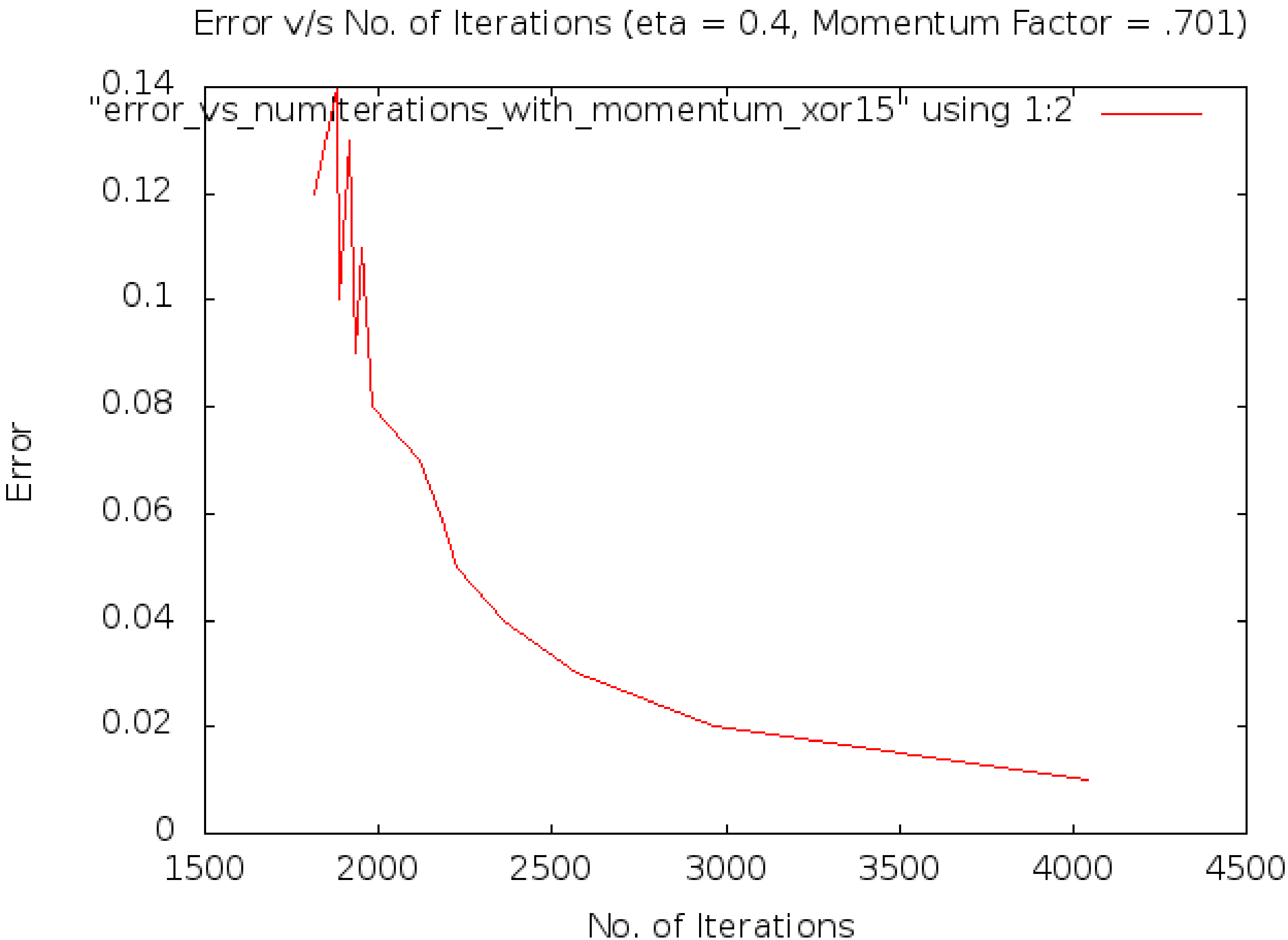
Effect of Momentum Factor (XOR)



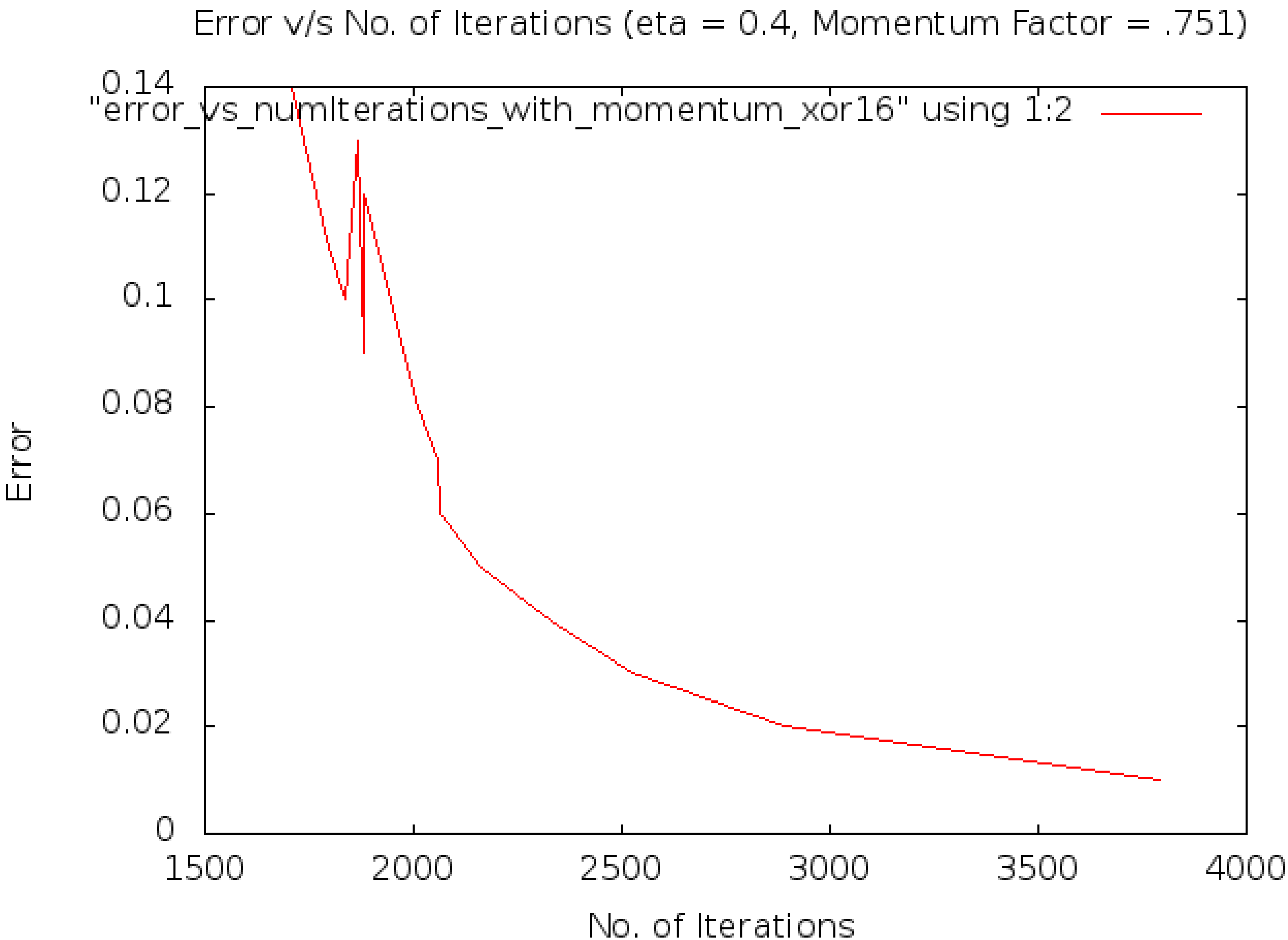
Effect of Momentum Factor (XOR)



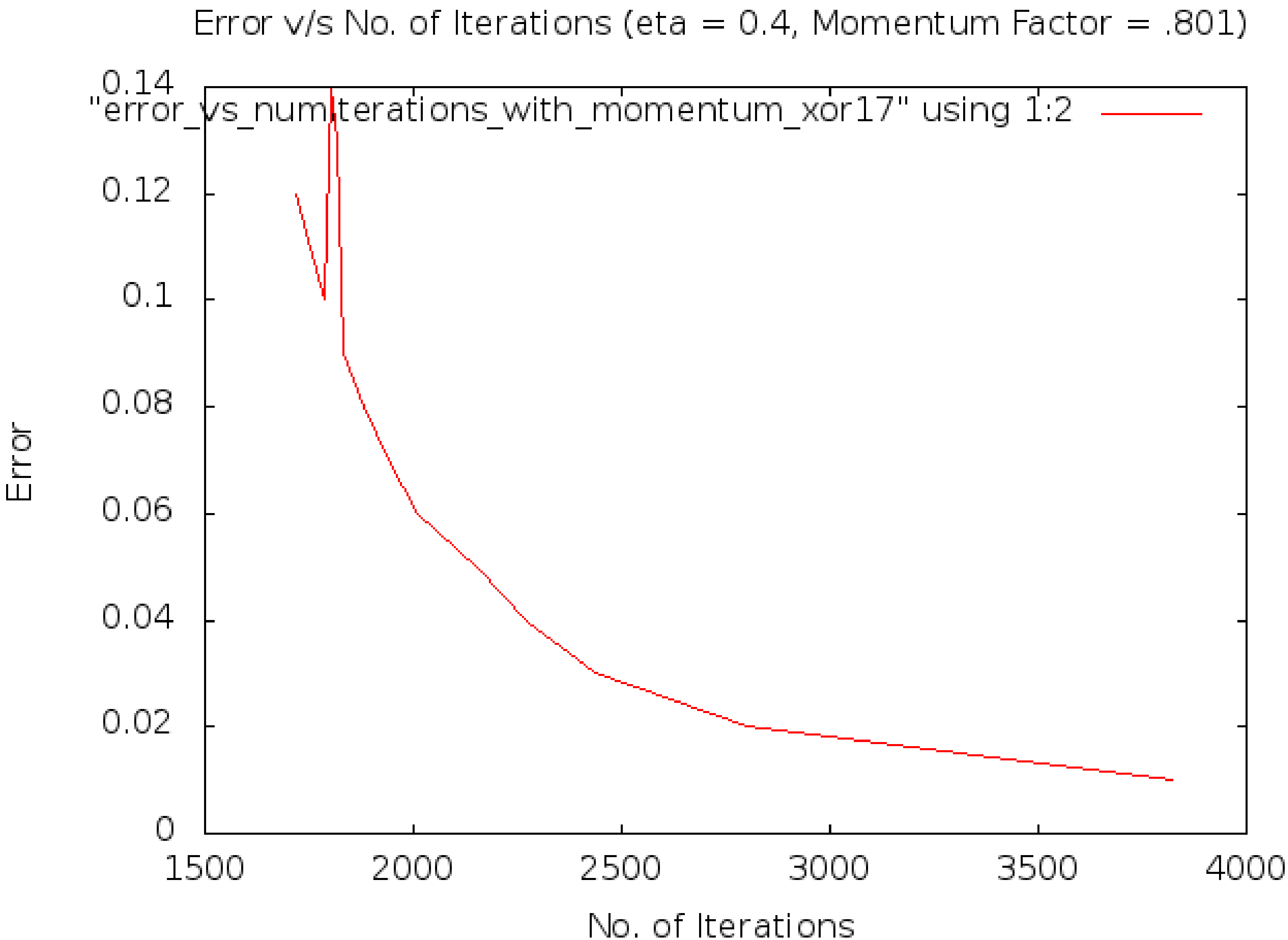
Effect of Momentum Factor (XOR)



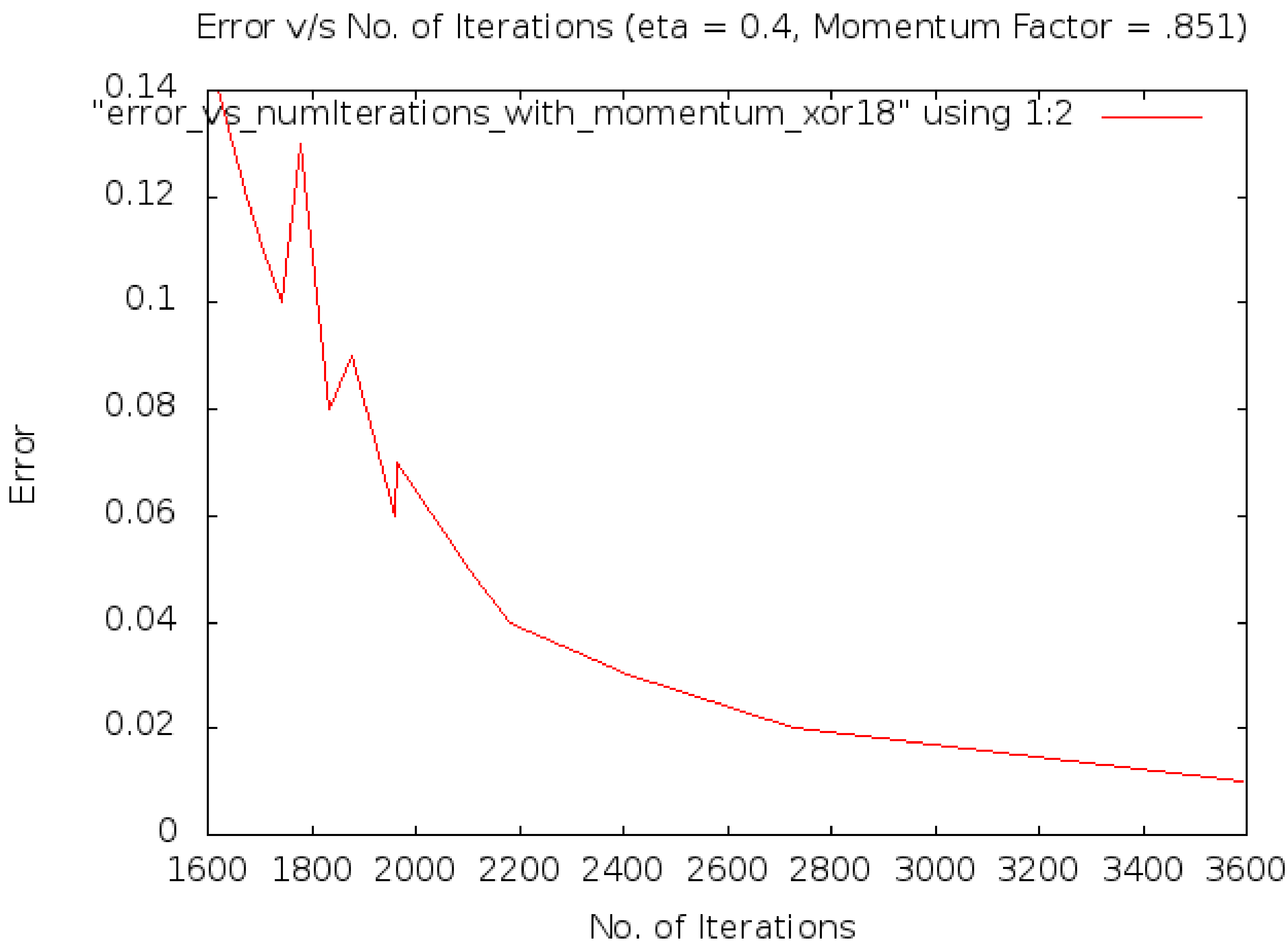
Effect of Momentum Factor (XOR)



Effect of Momentum Factor (XOR)

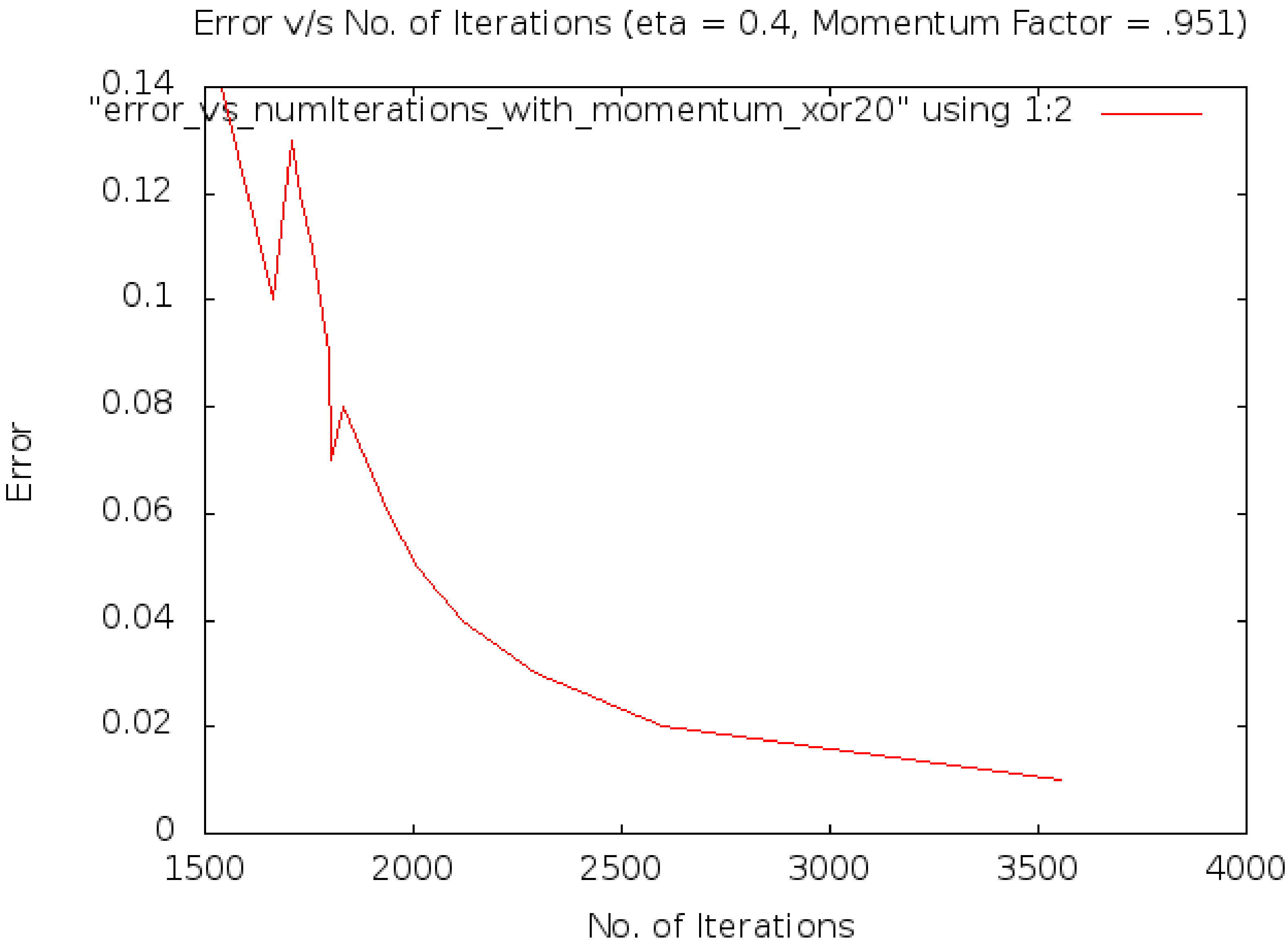


Effect of Momentum Factor (XOR)

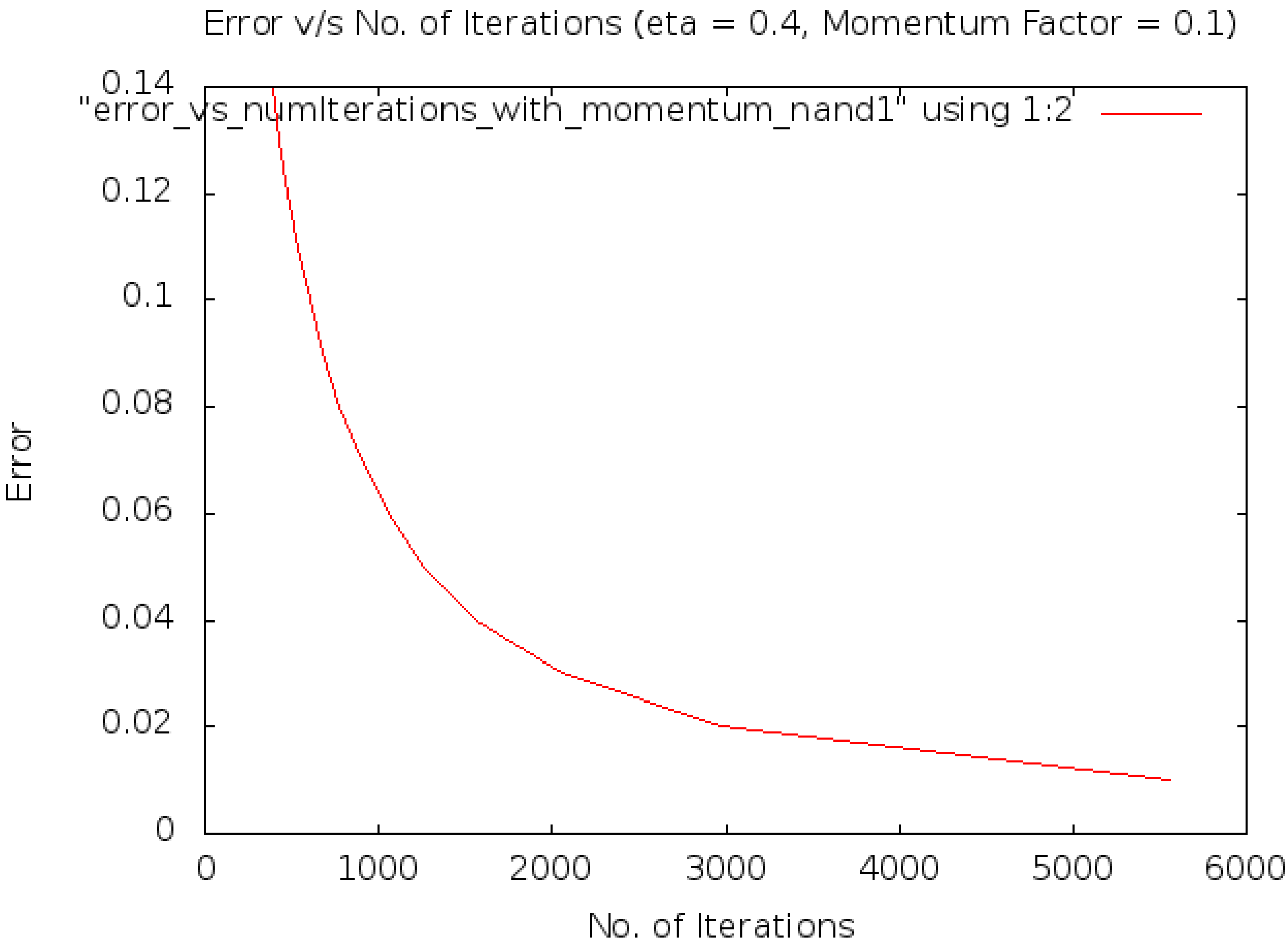


Effect of Momentum Factor (XOR)

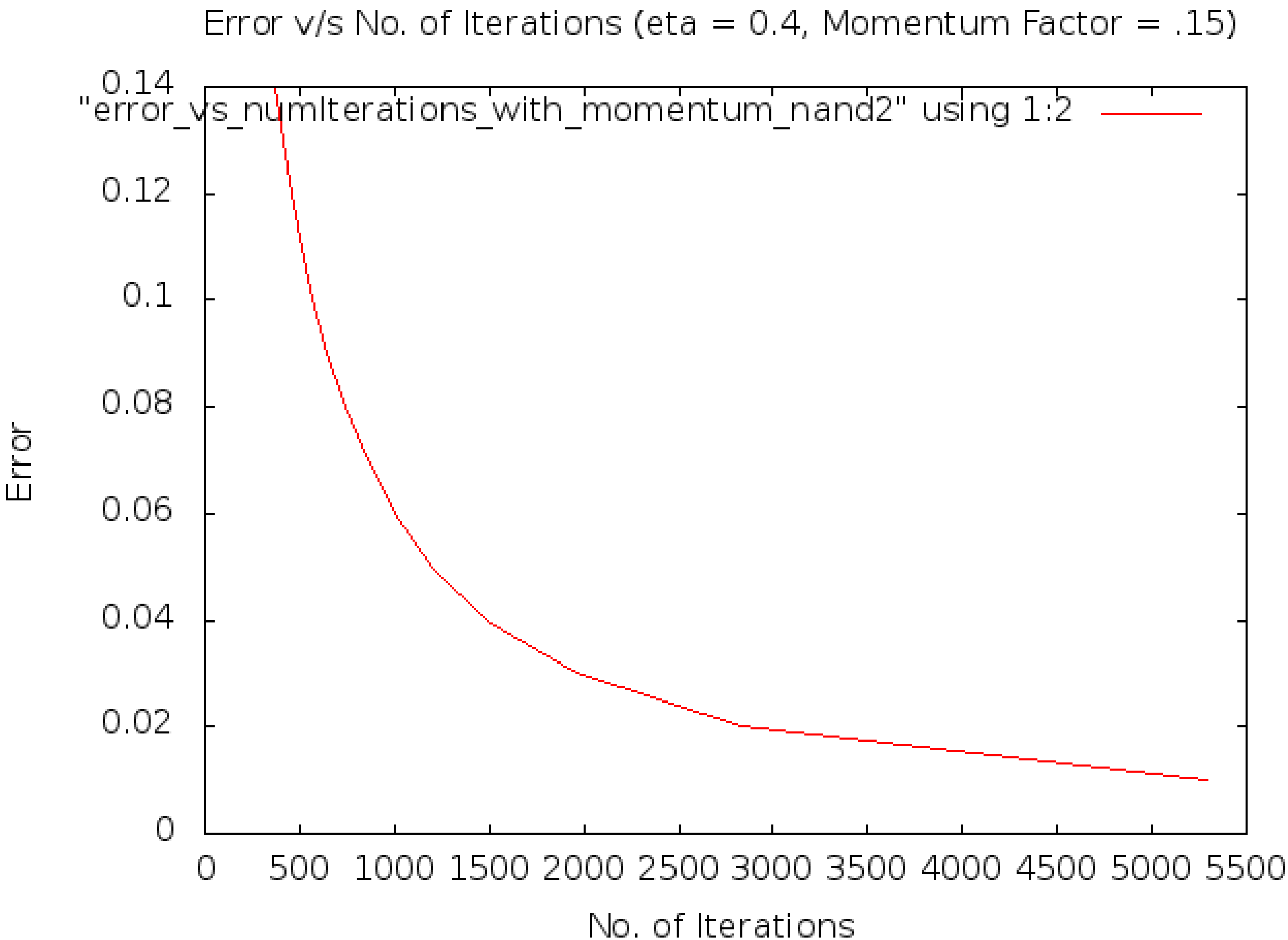
Effect of Momentum Factor (XOR)



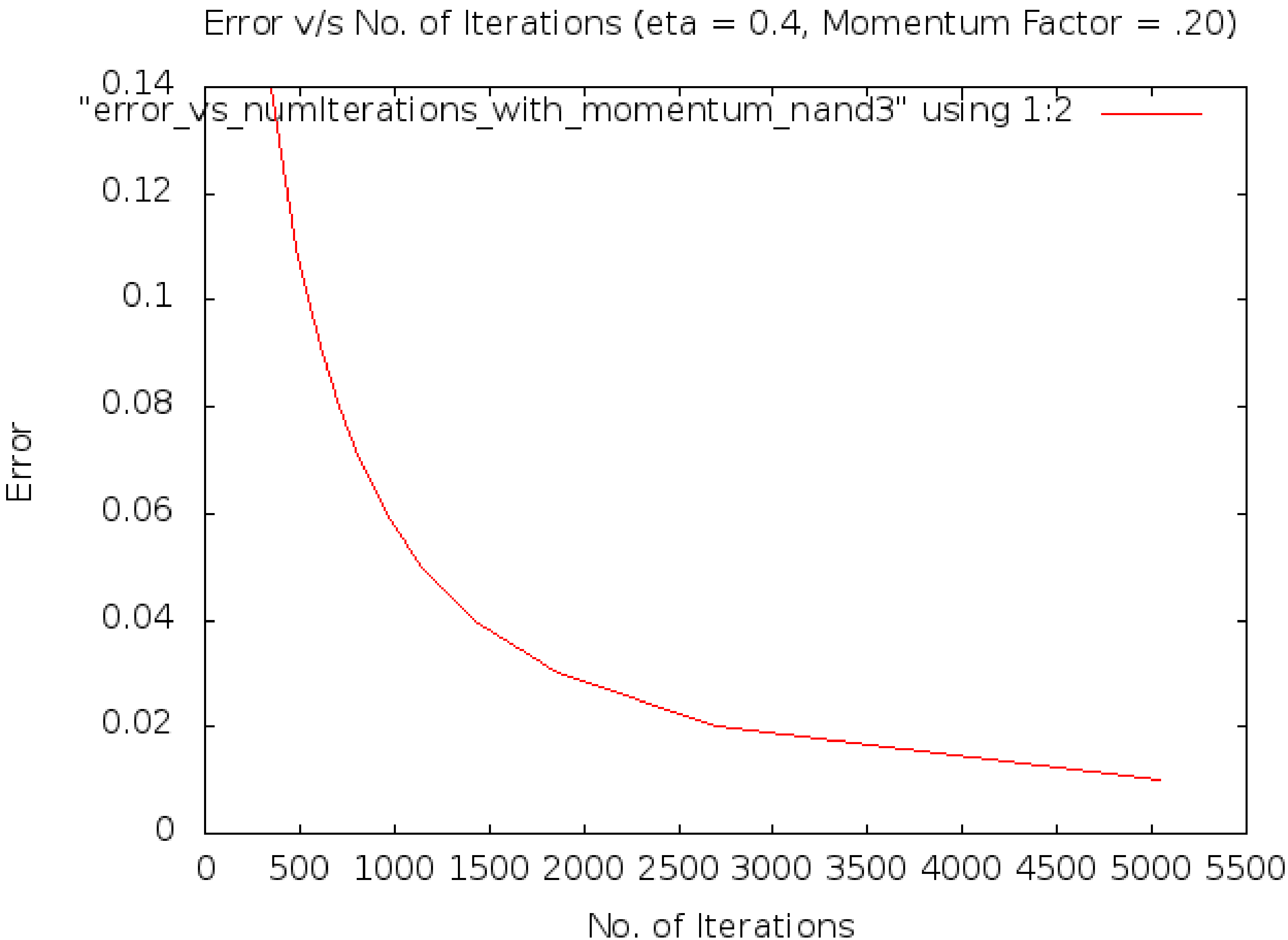
Effect of Momentum Factor (NAND)



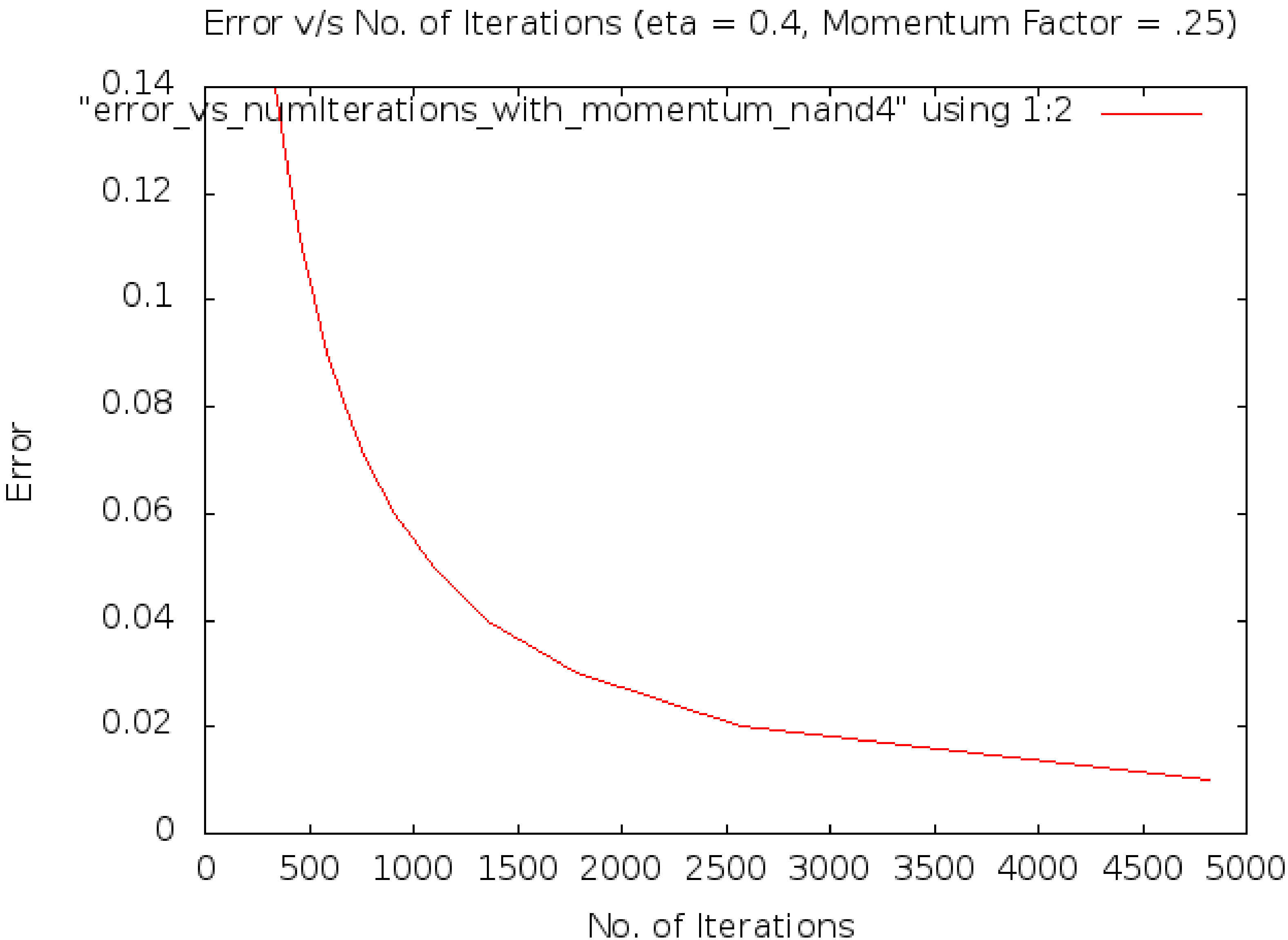
Effect of Momentum Factor (NAND)



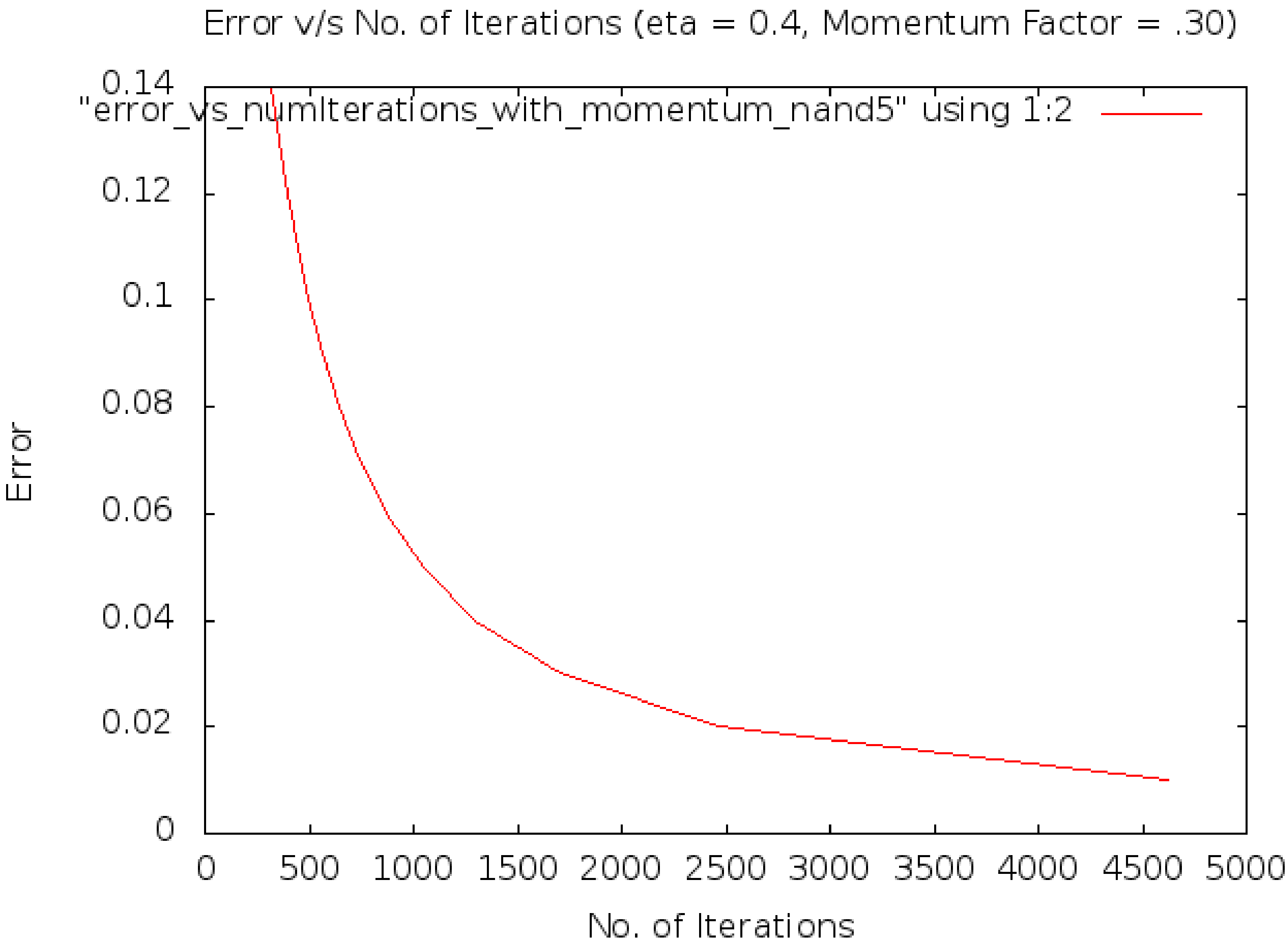
Effect of Momentum Factor (NAND)



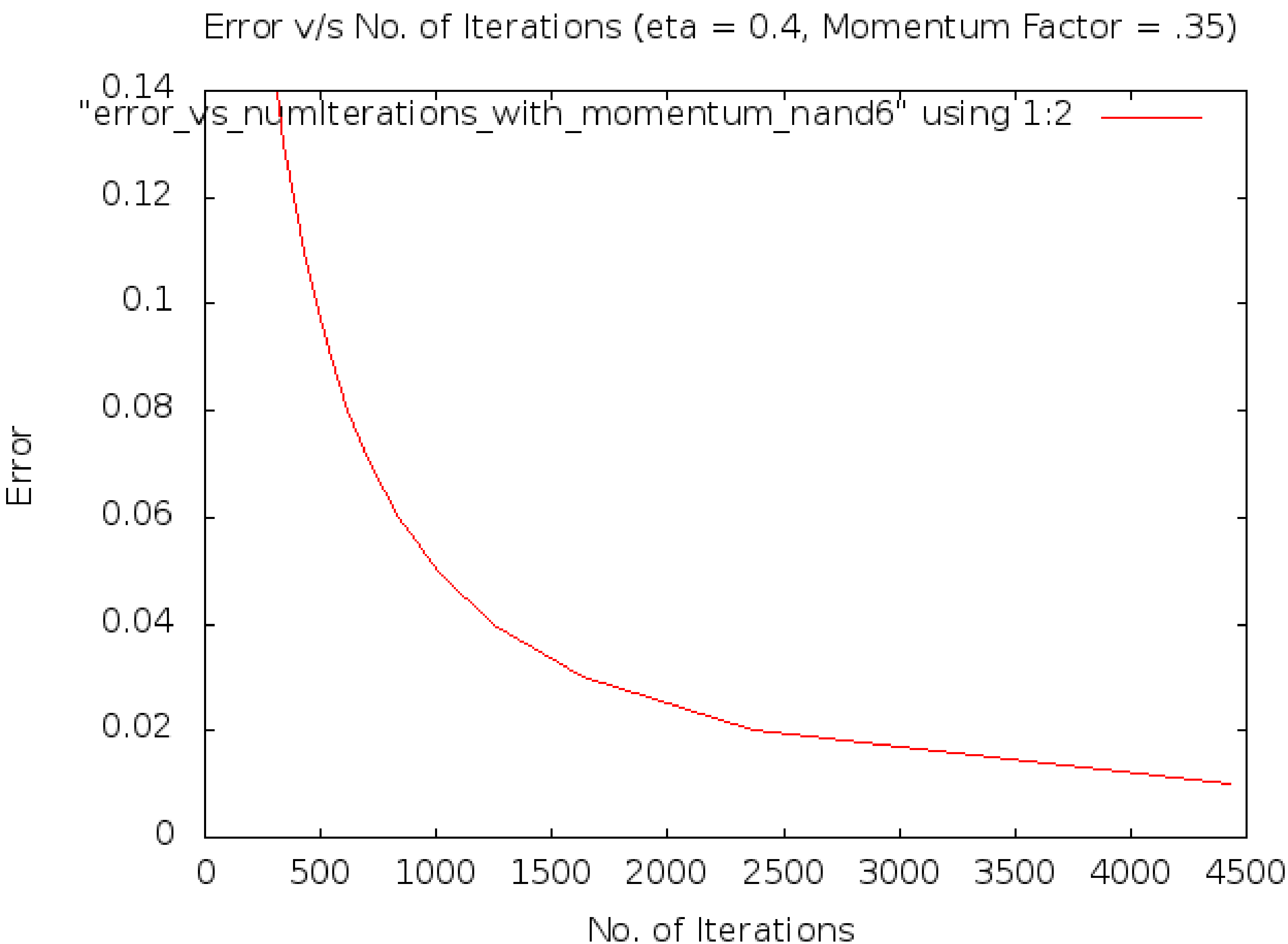
Effect of Momentum Factor (NAND)



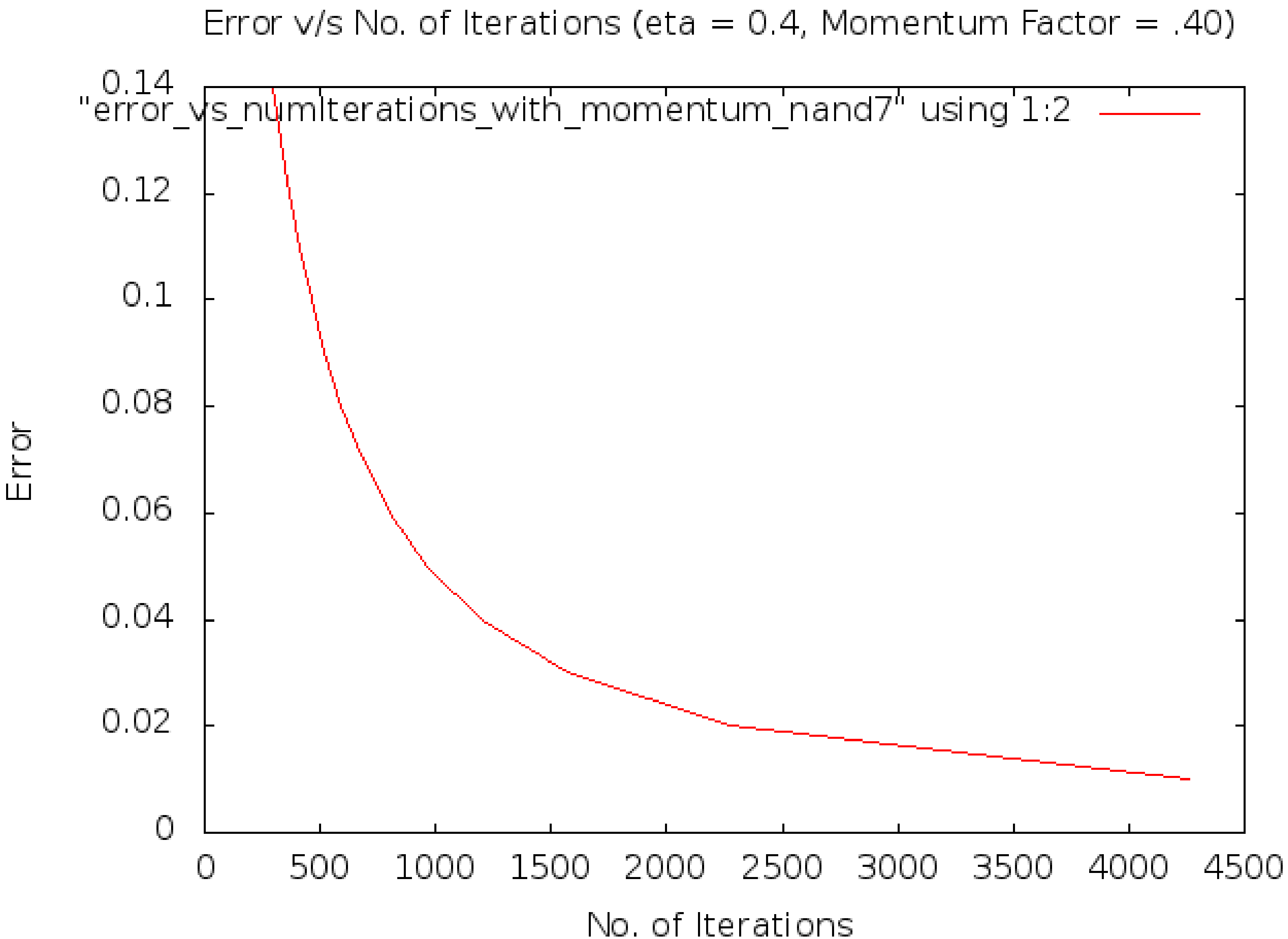
Effect of Momentum Factor (NAND)



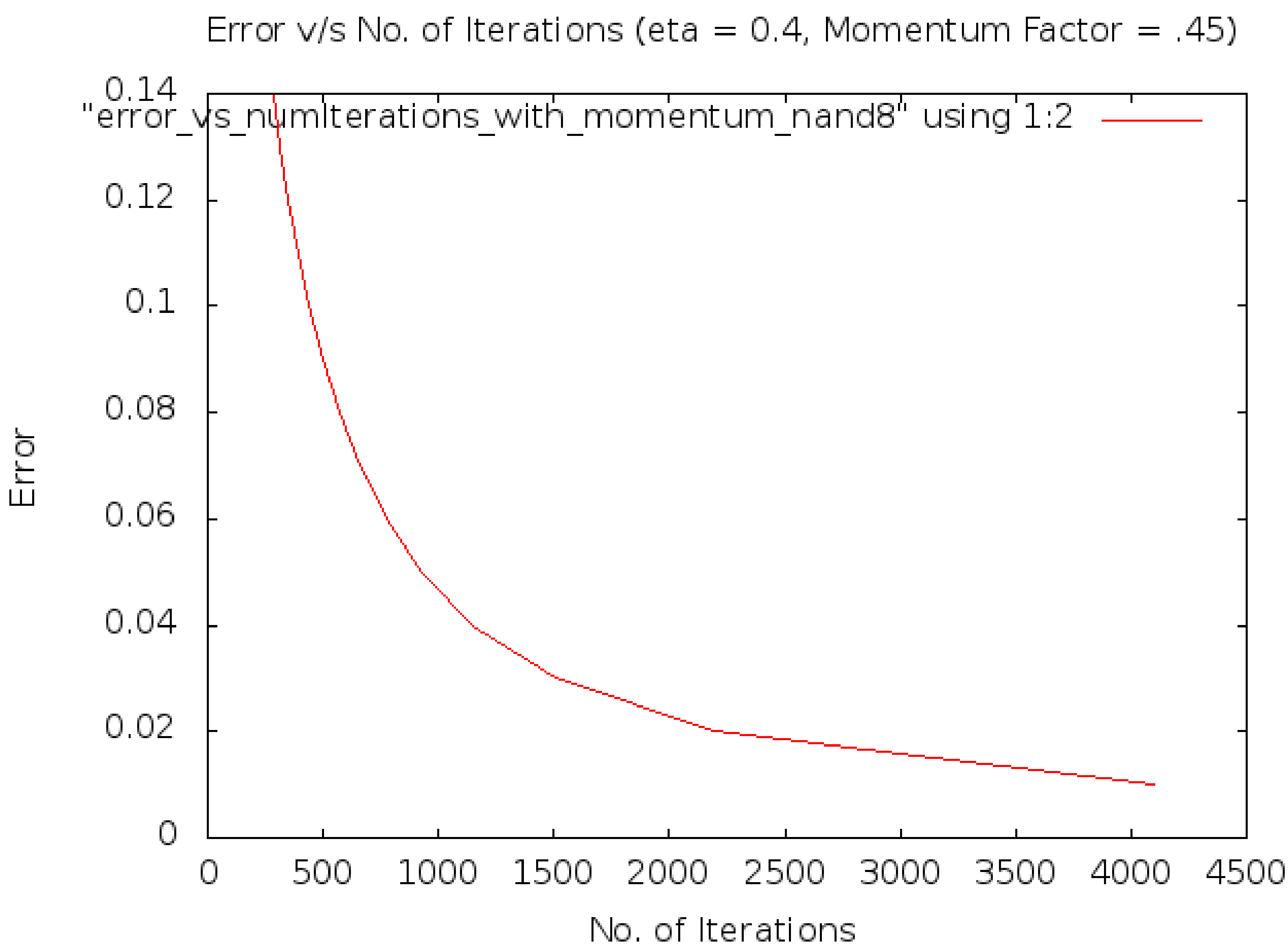
Effect of Momentum Factor (NAND)



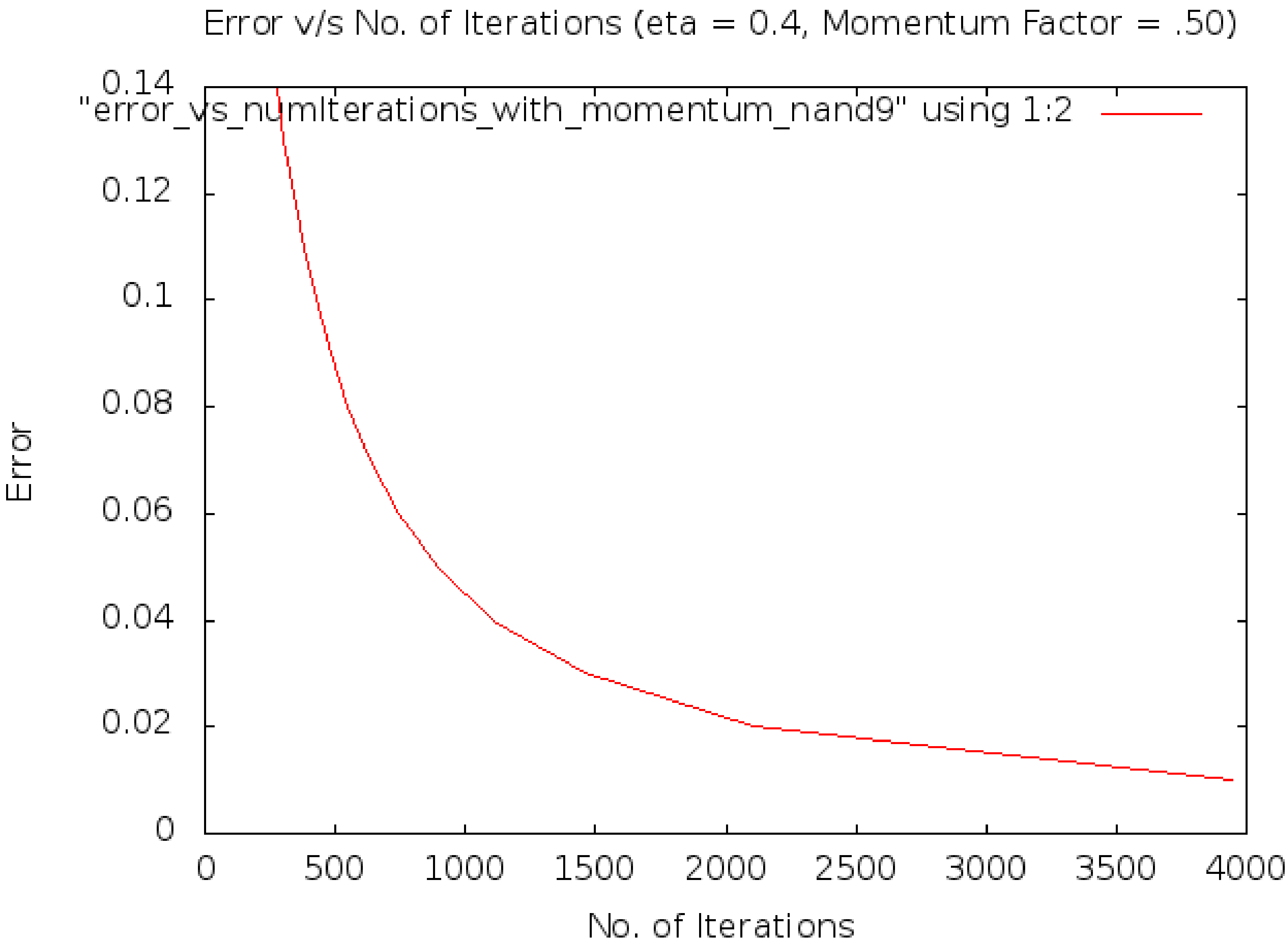
Effect of Momentum Factor (NAND)



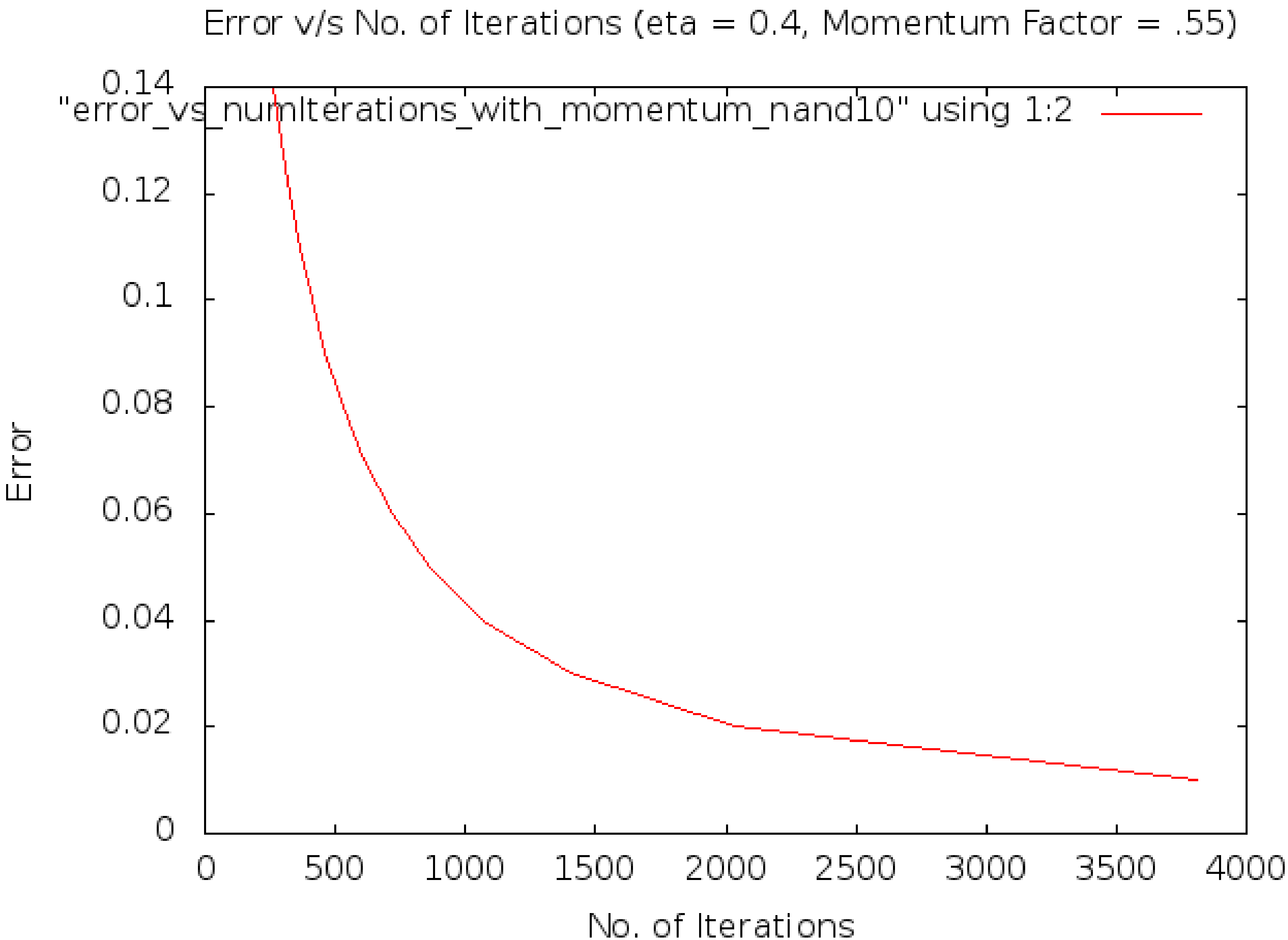
Effect of Momentum Factor (NAND)



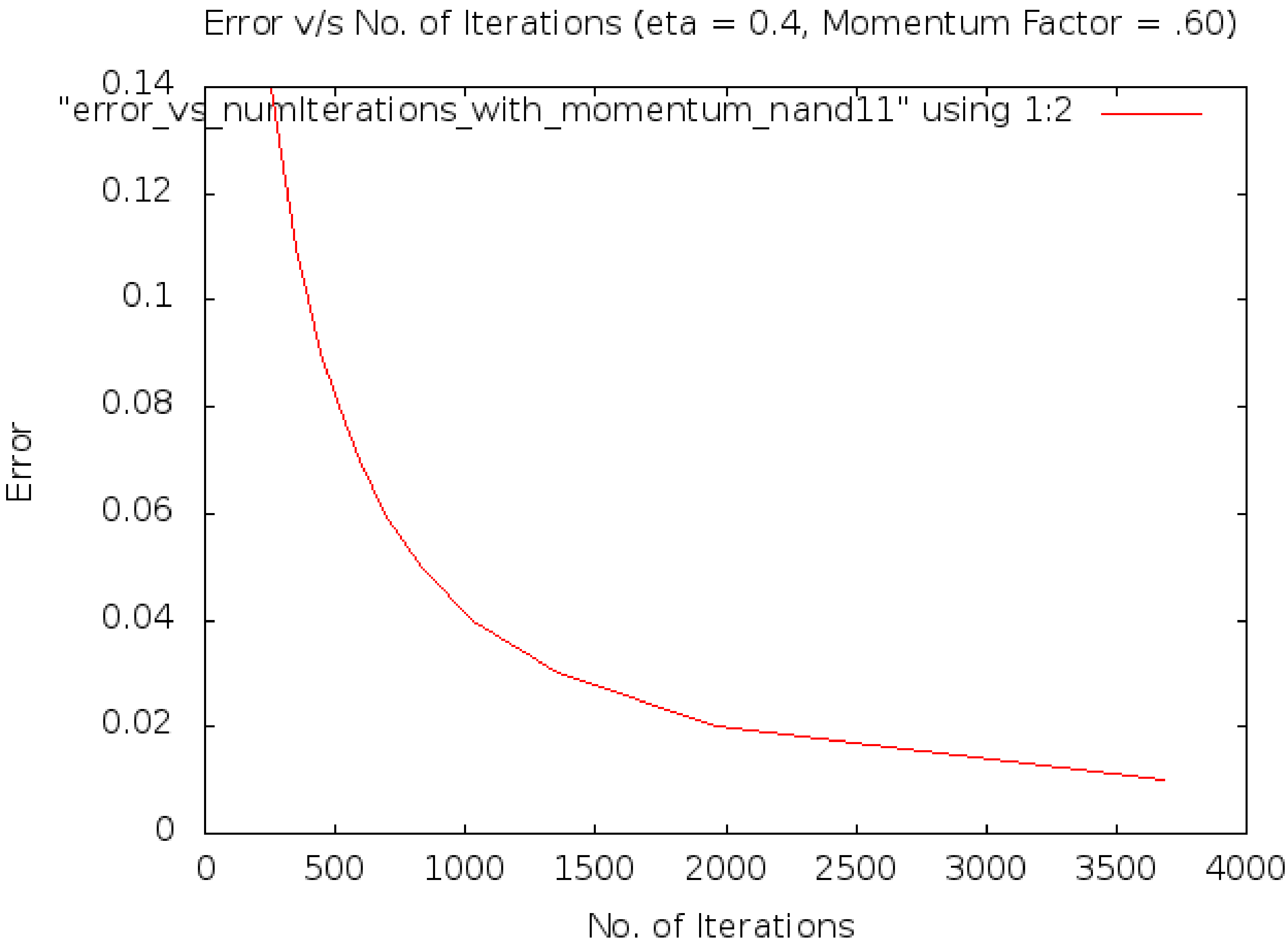
Effect of Momentum Factor (NAND)



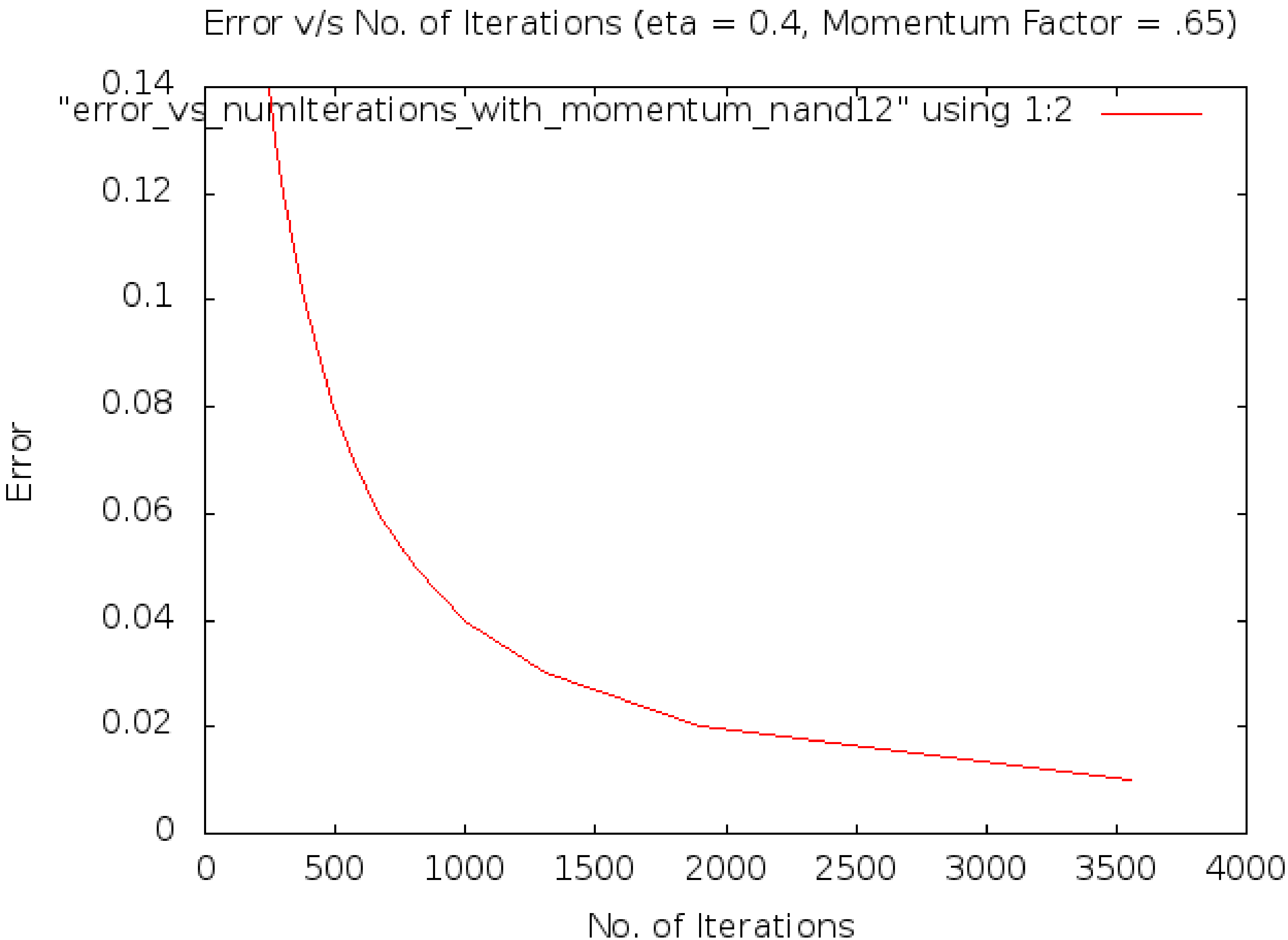
Effect of Momentum Factor (NAND)



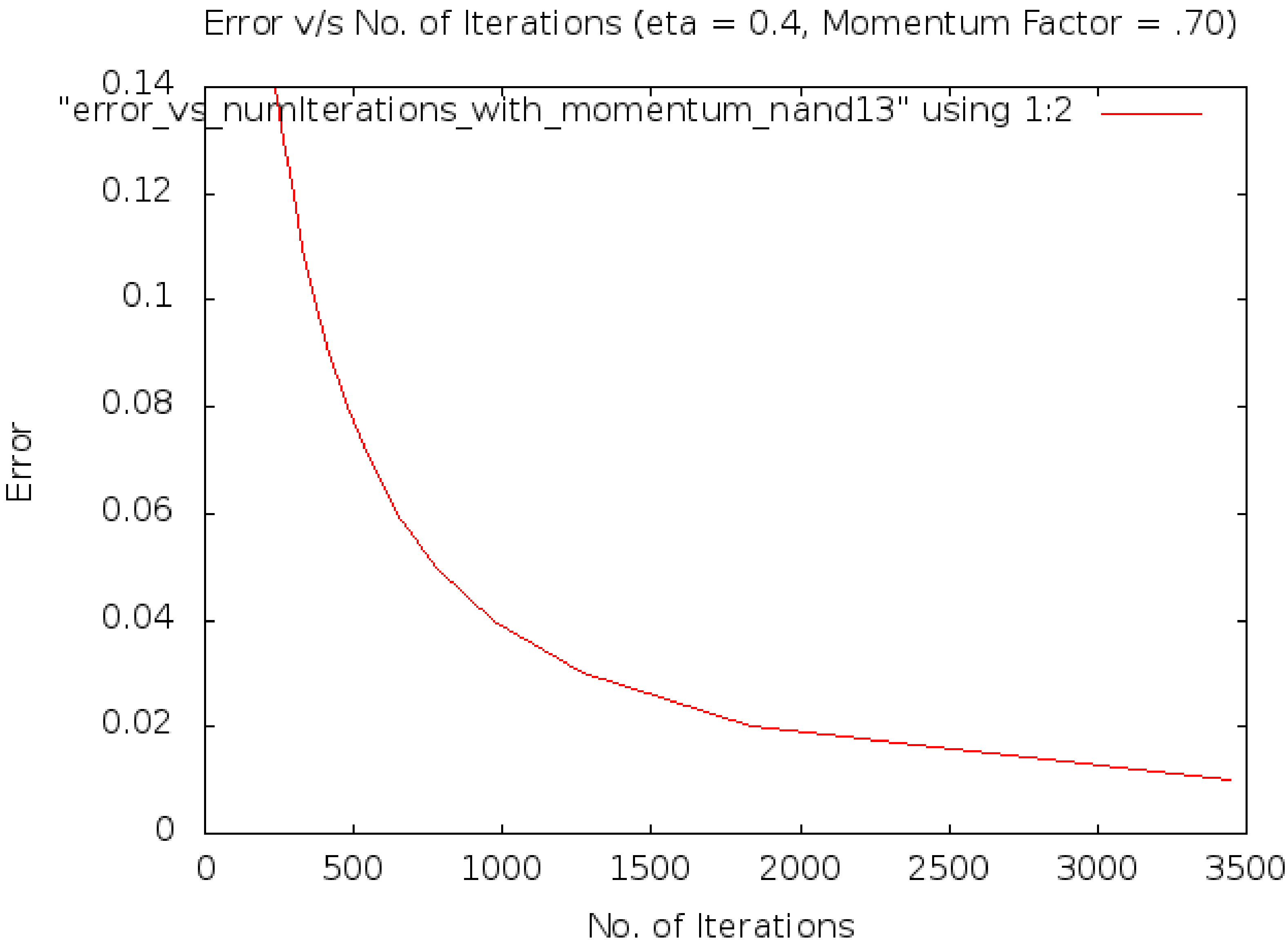
Effect of Momentum Factor (NAND)



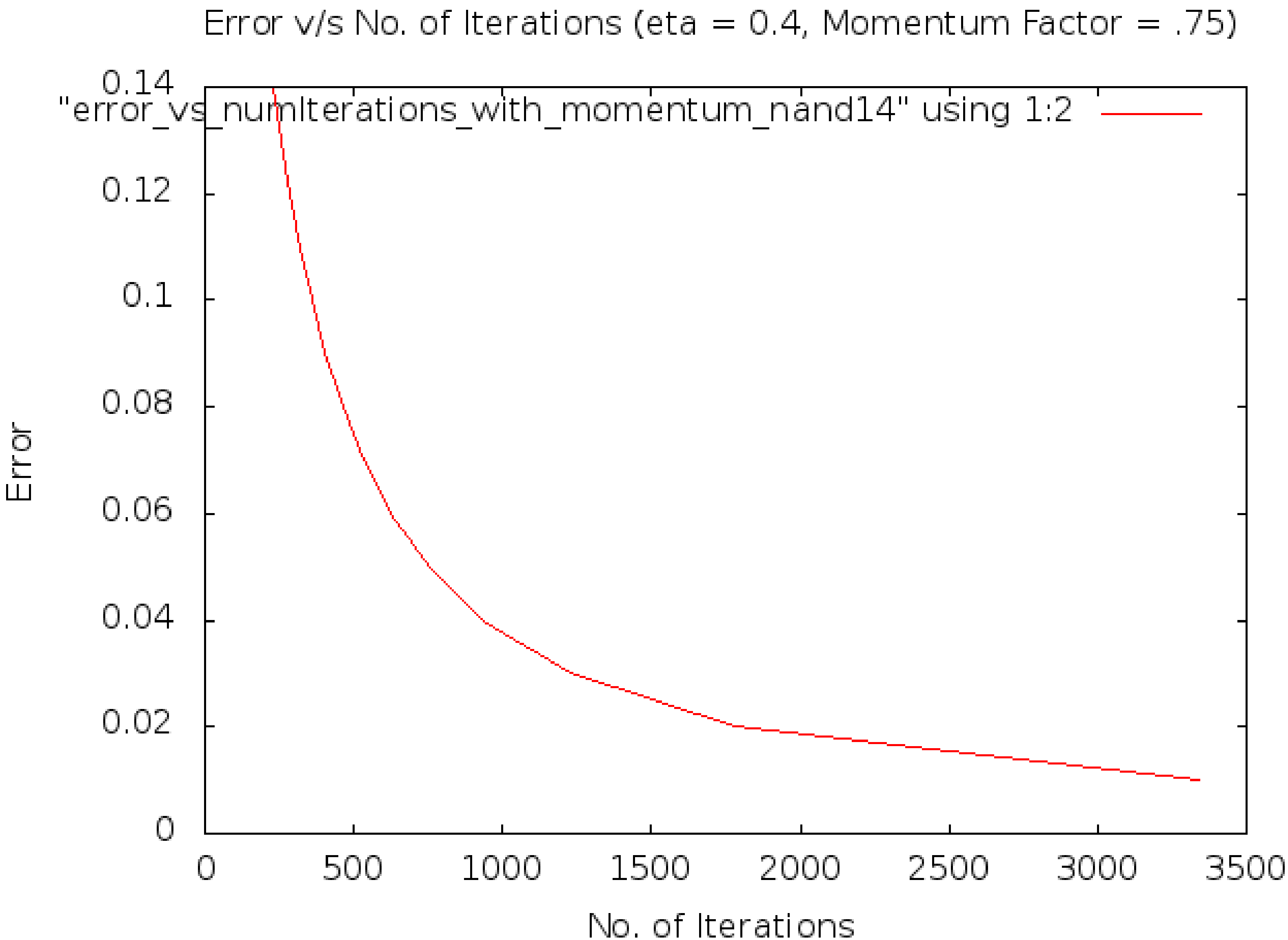
Effect of Momentum Factor (NAND)



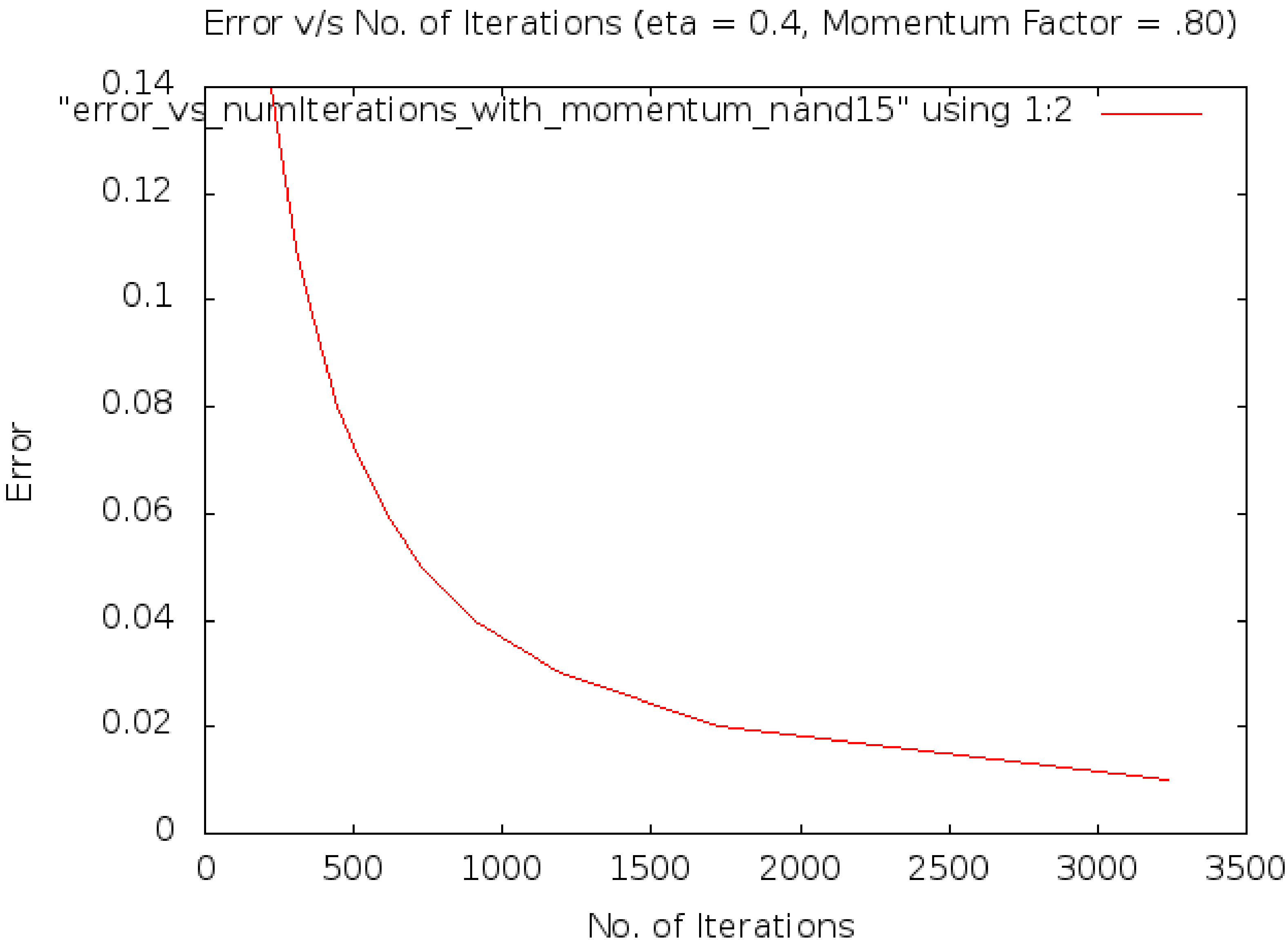
Effect of Momentum Factor (NAND)



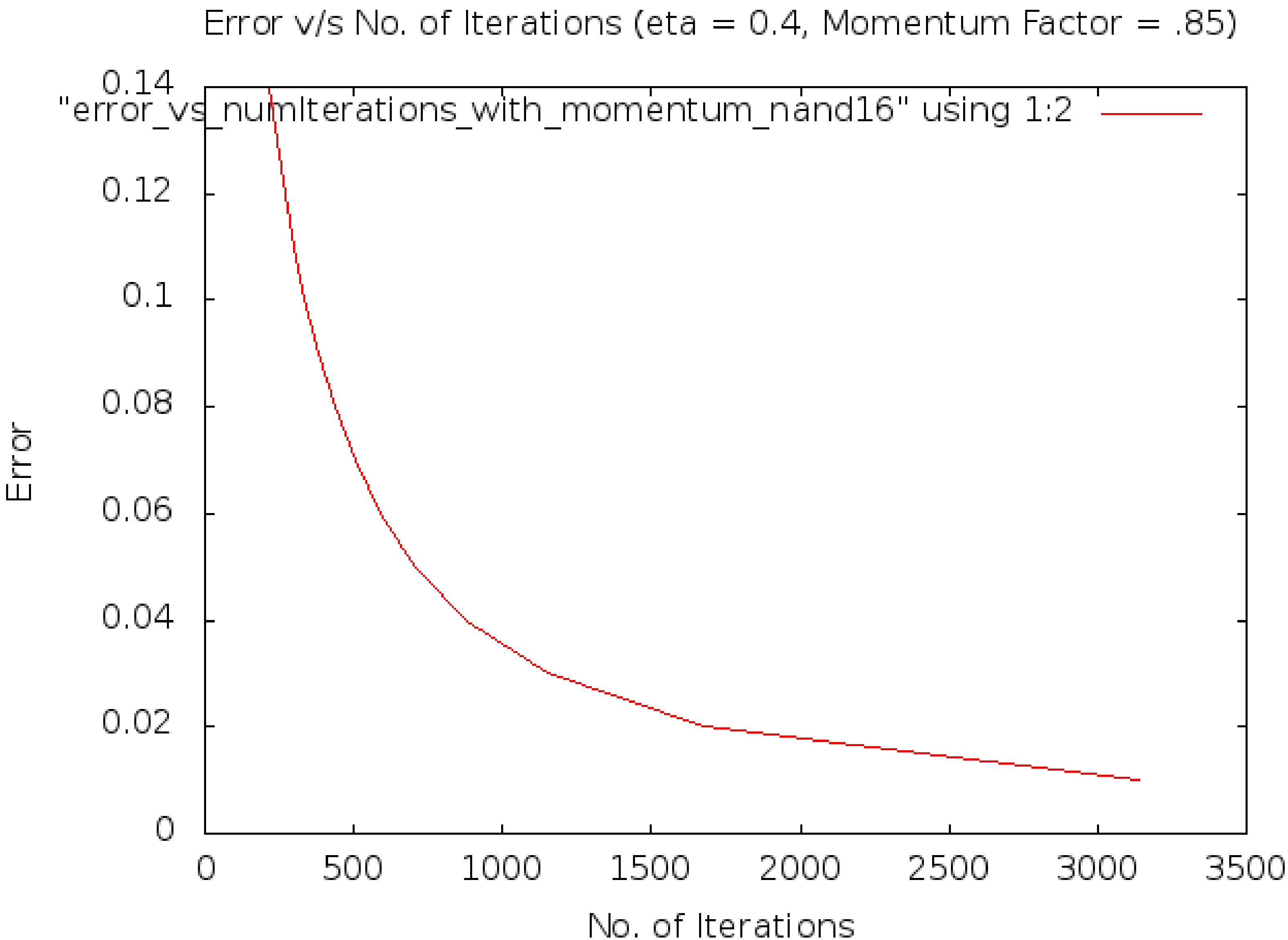
Effect of Momentum Factor (NAND)



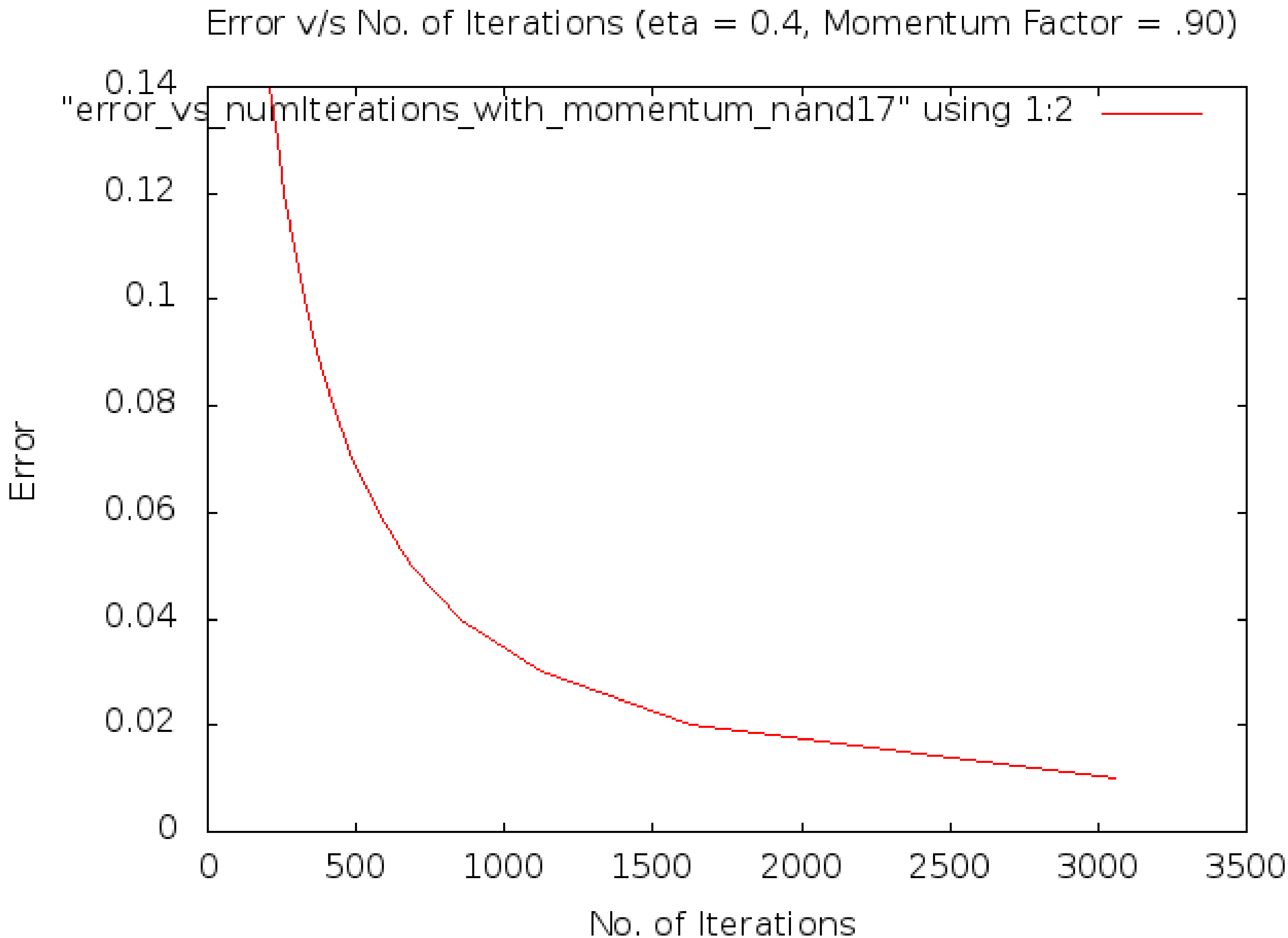
Effect of Momentum Factor (NAND)



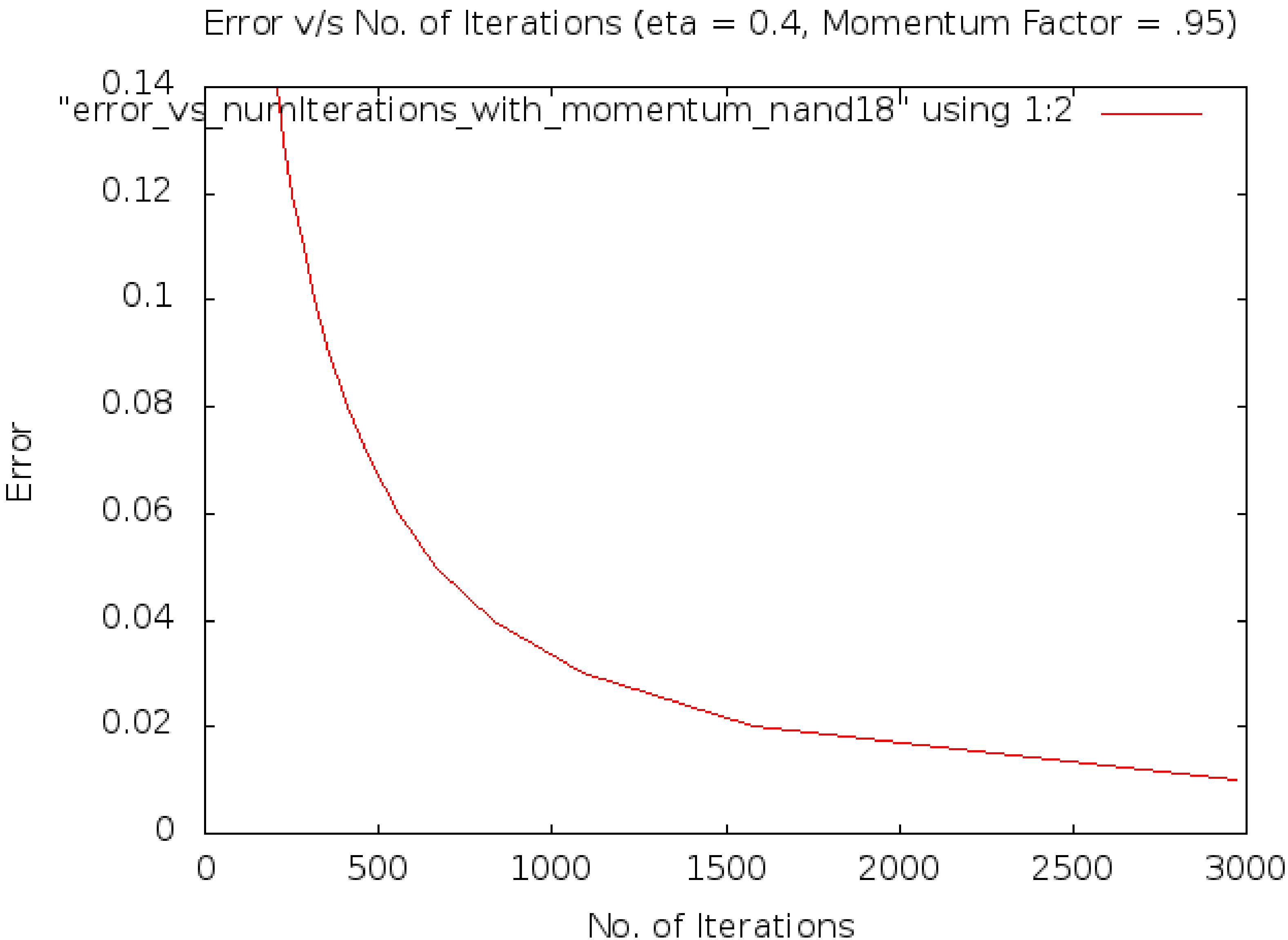
Effect of Momentum Factor (NAND)



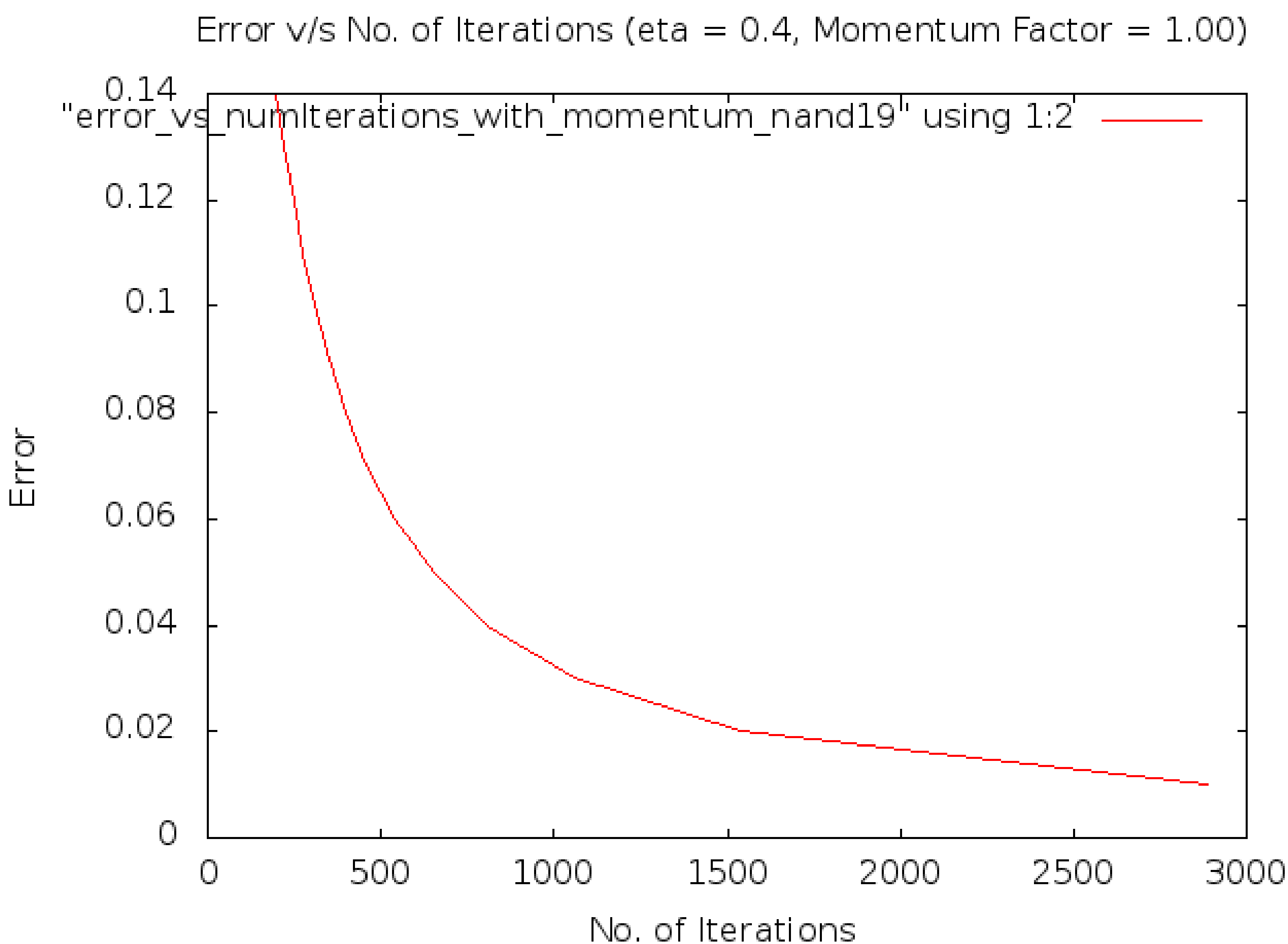
Effect of Momentum Factor (NAND)



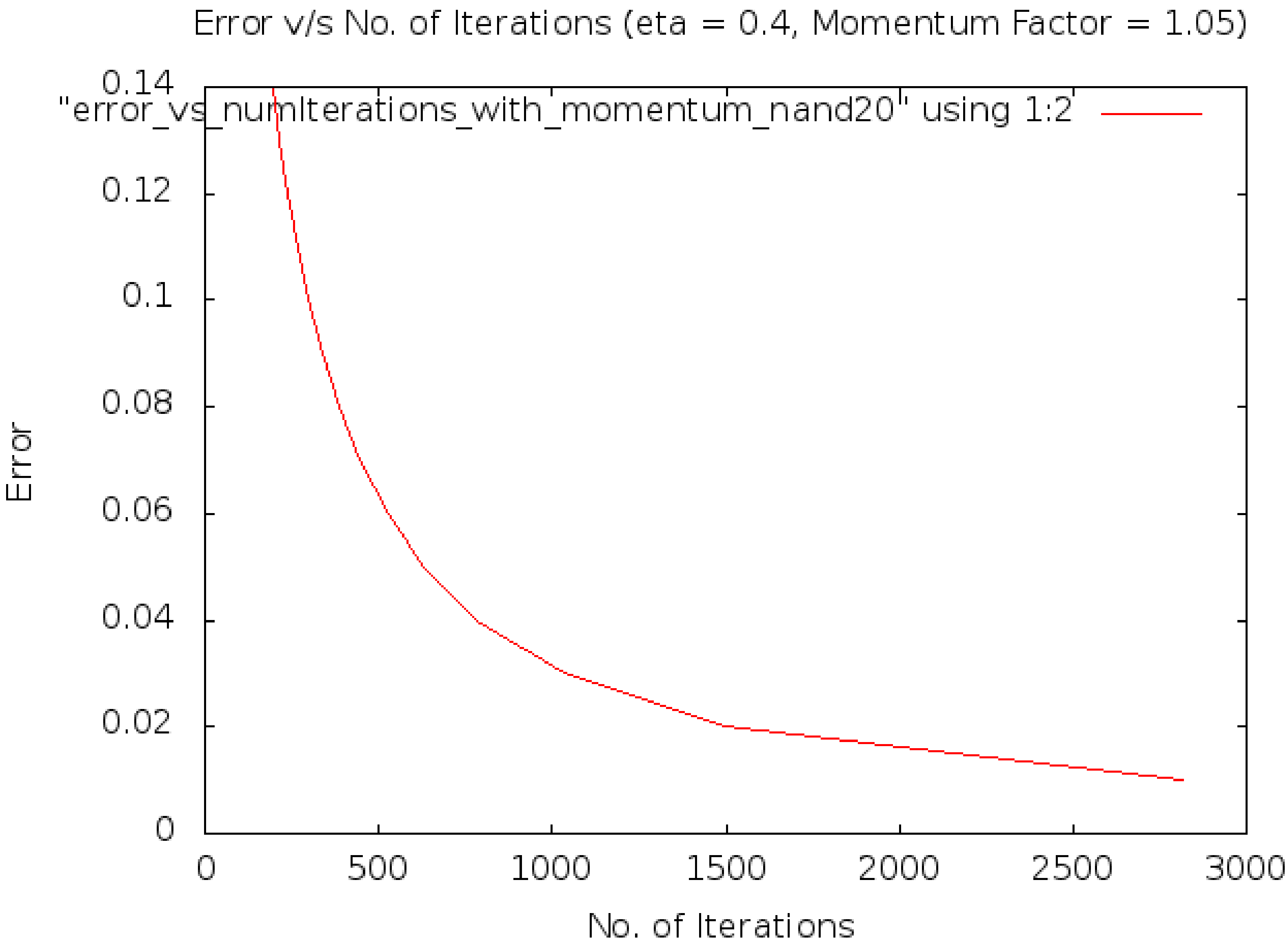
Effect of Momentum Factor (NAND)



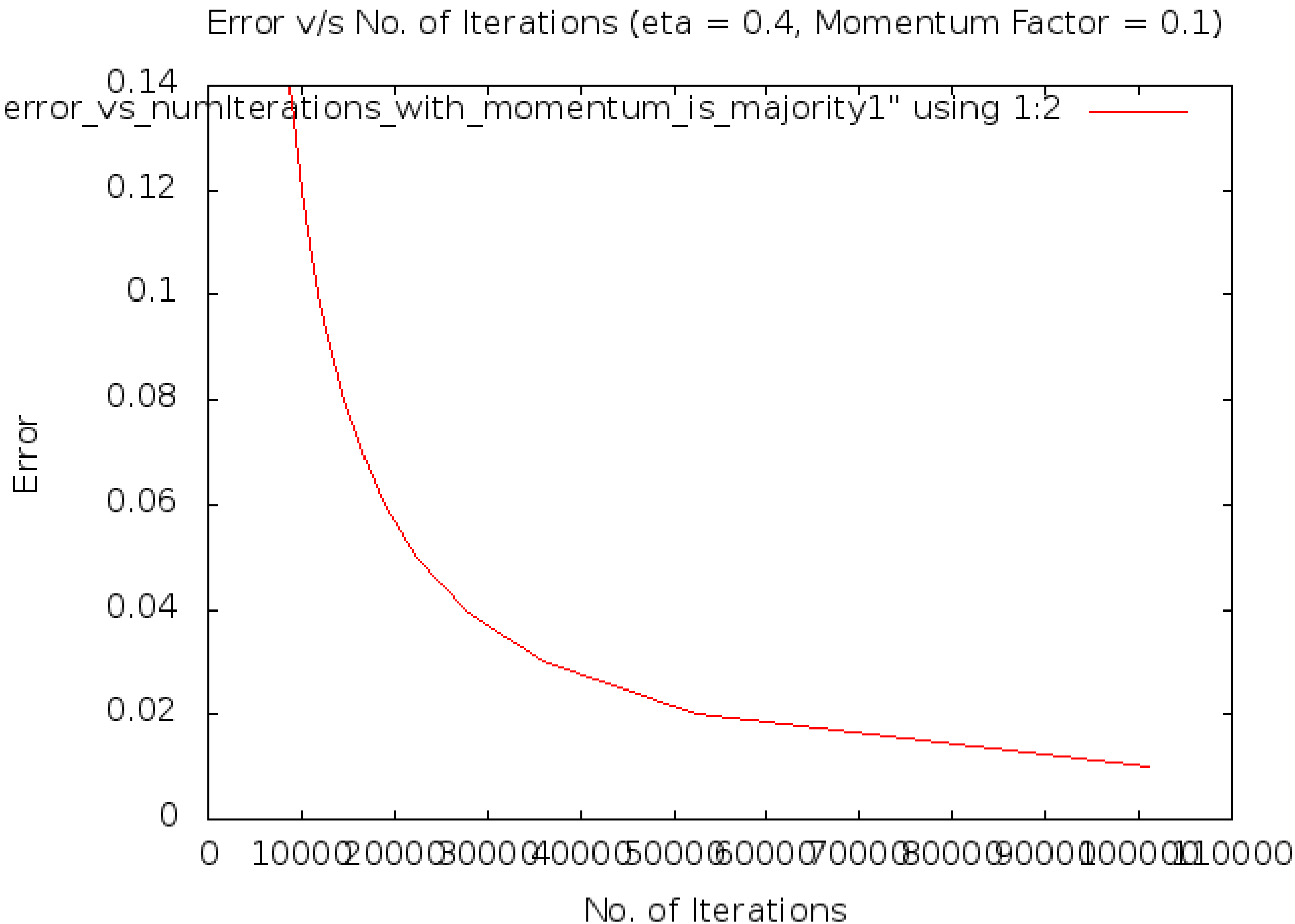
Effect of Momentum Factor (NAND)



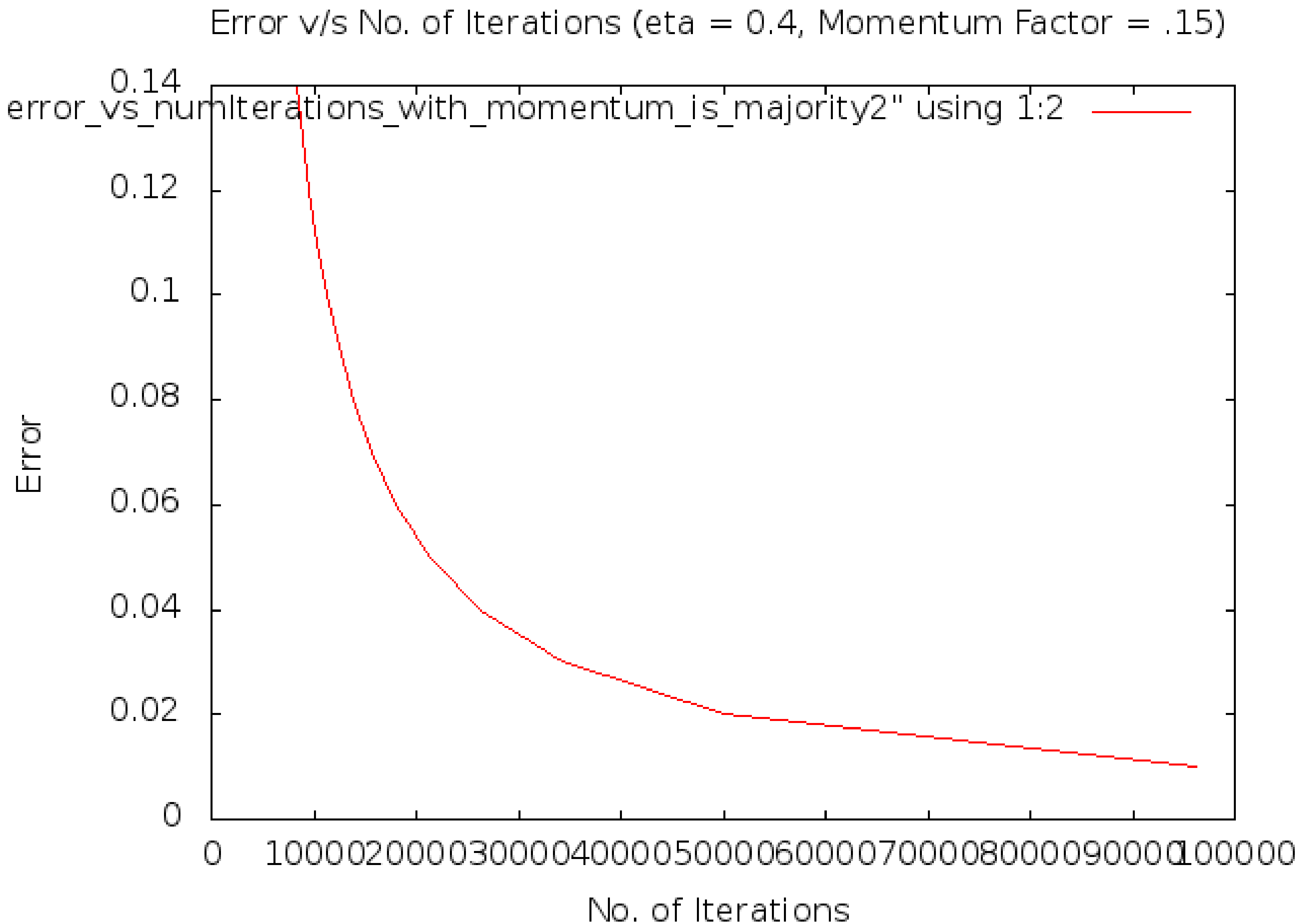
Effect of Momentum Factor (NAND)



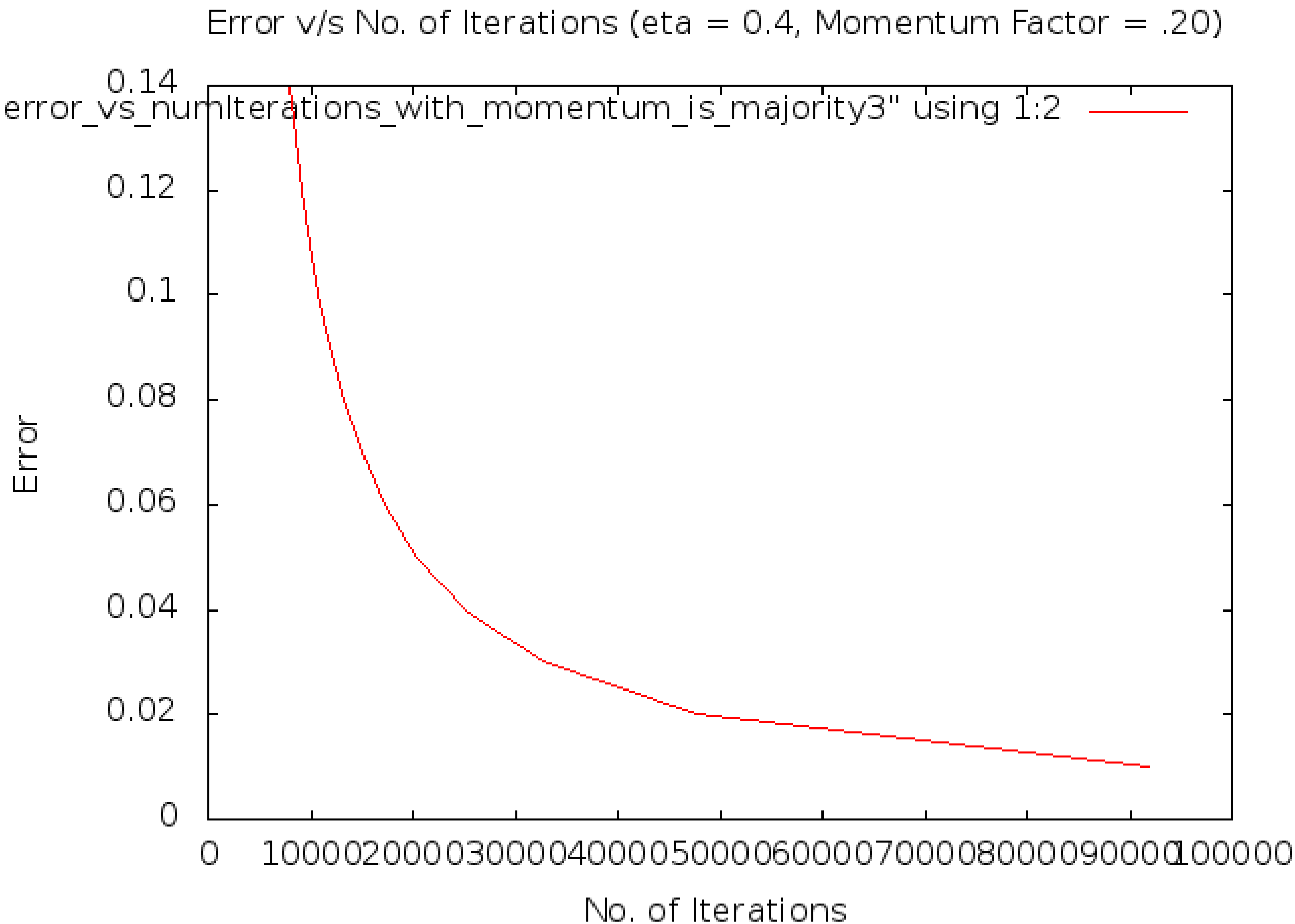
Effect of Momentum Factor (Majority)



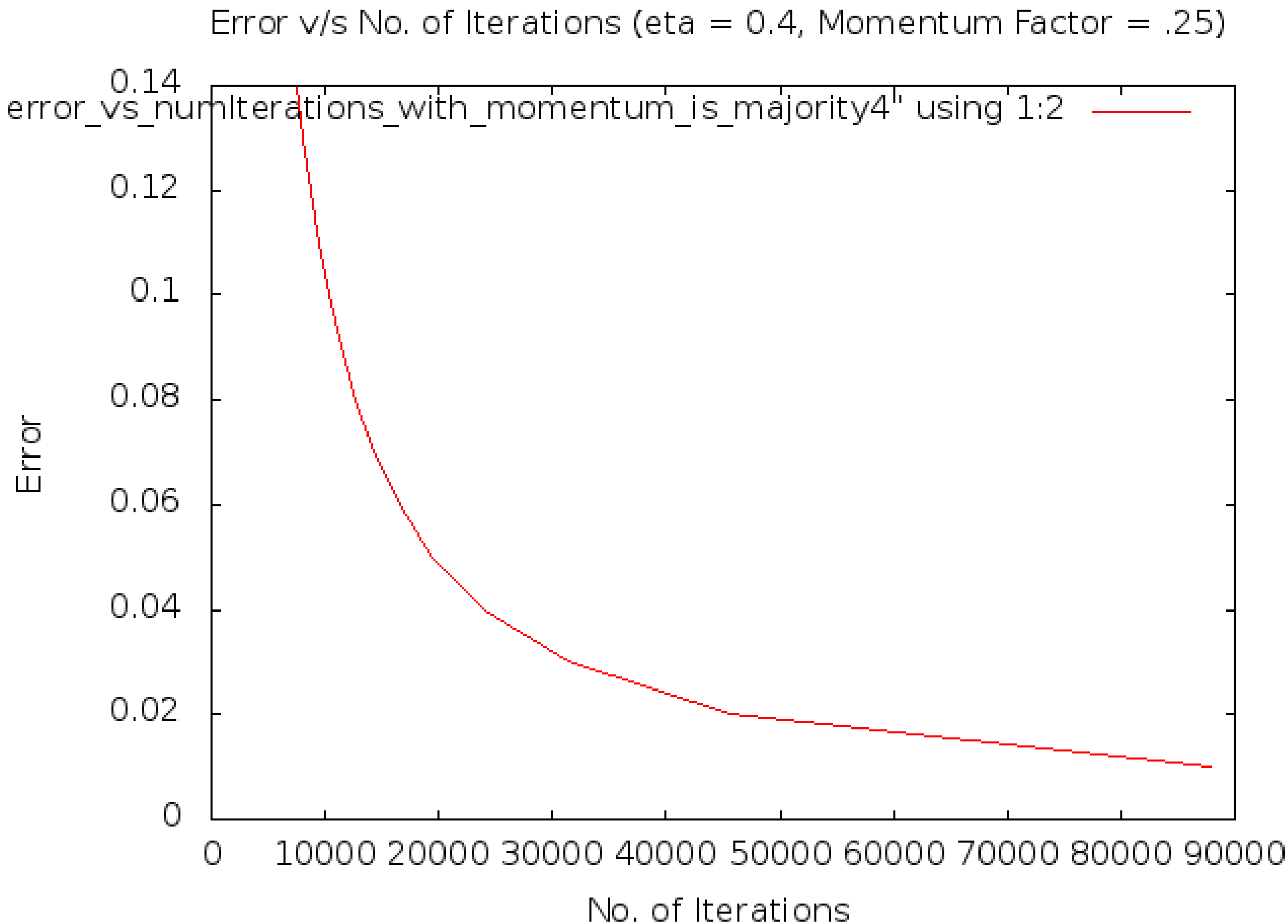
Effect of Momentum Factor (Majority)



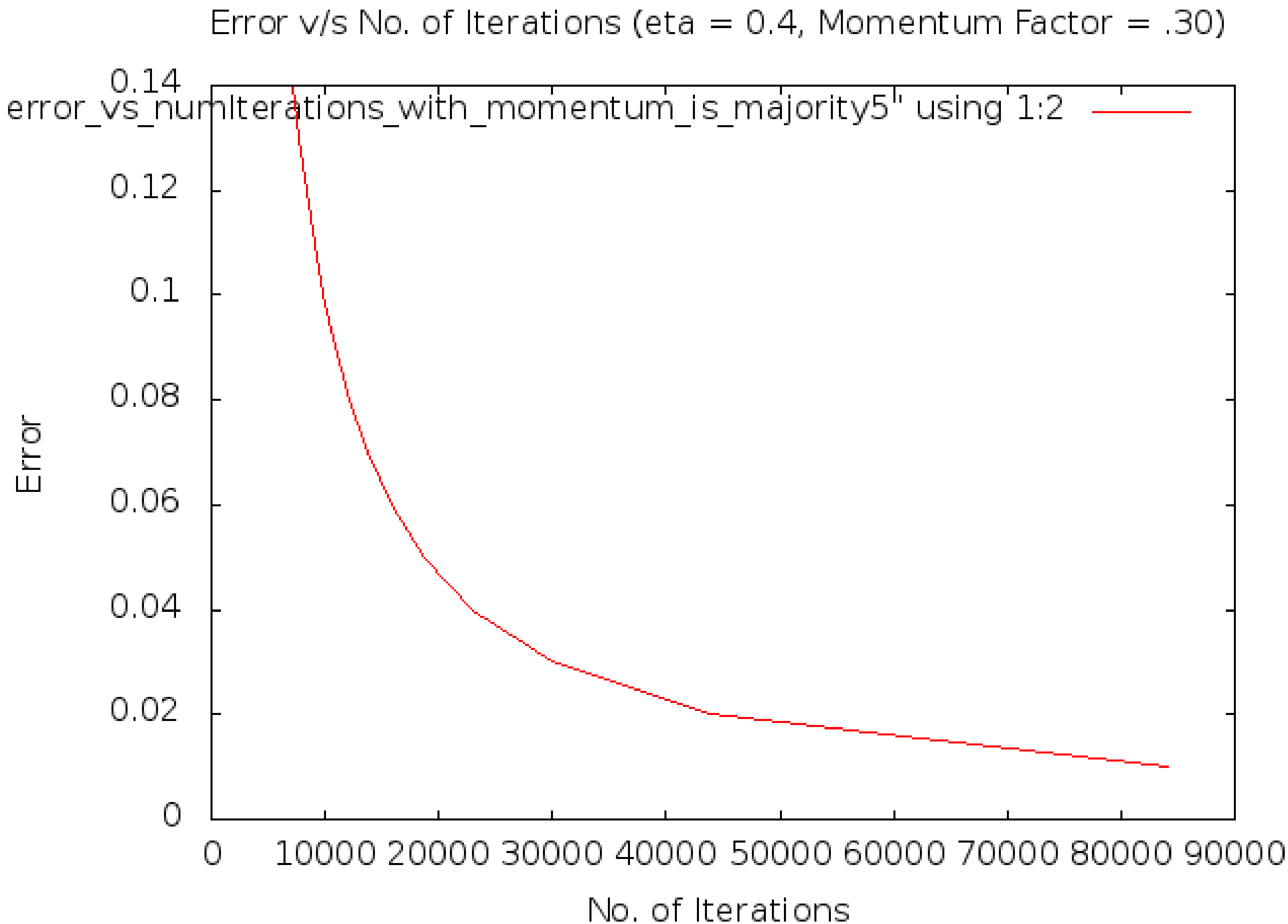
Effect of Momentum Factor (Majority)



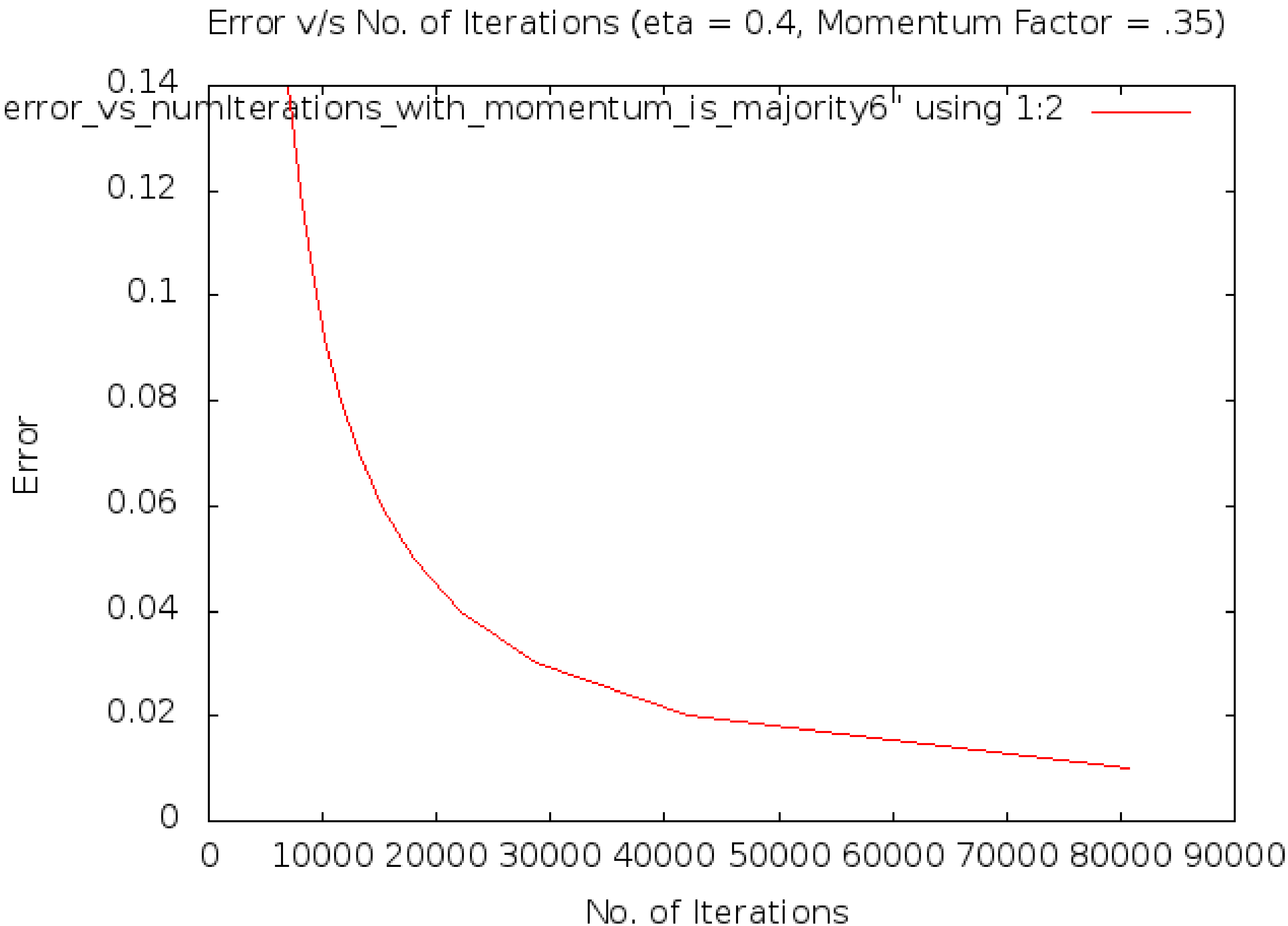
Effect of Momentum Factor (Majority)



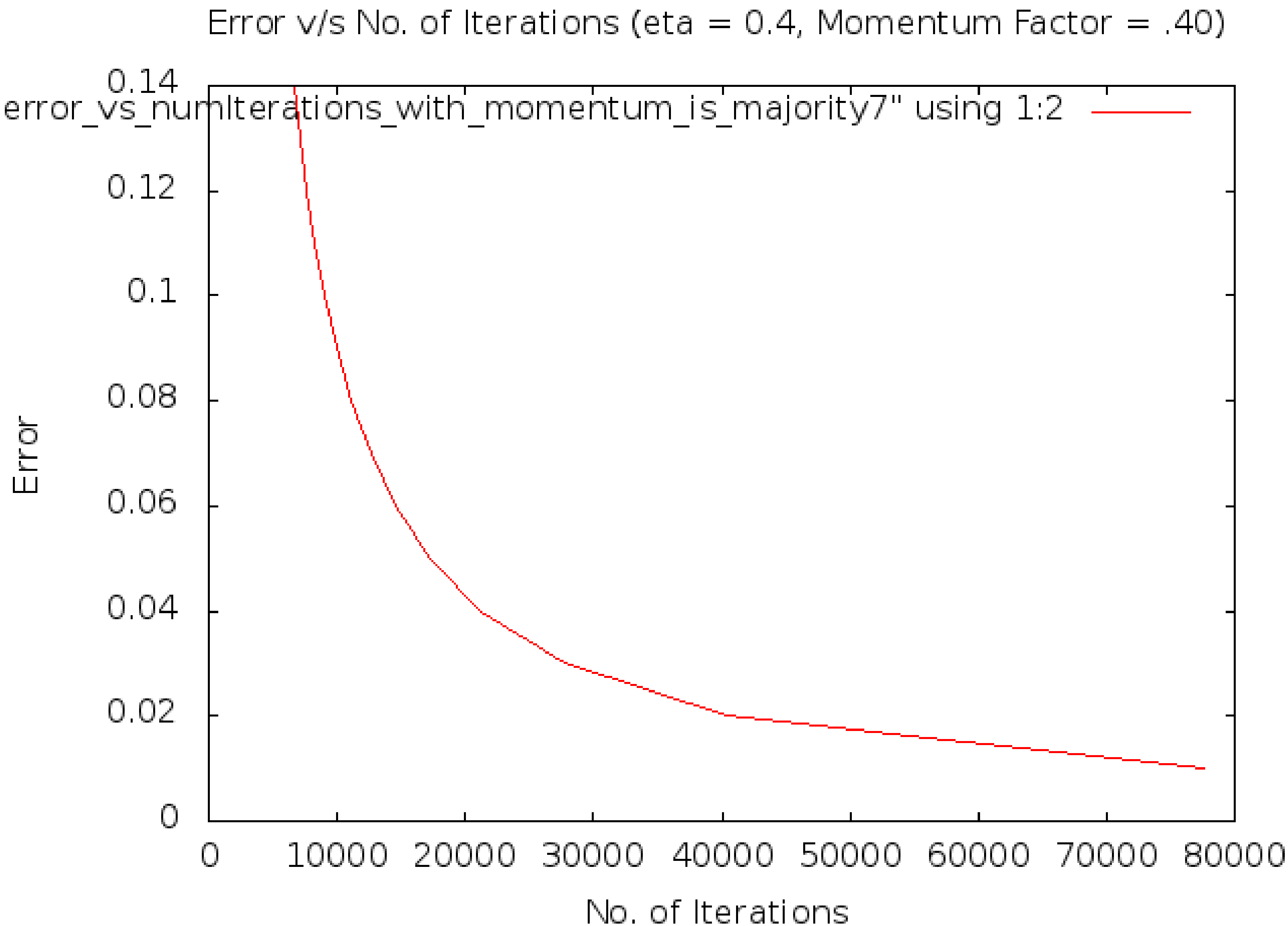
Effect of Momentum Factor (Majority)



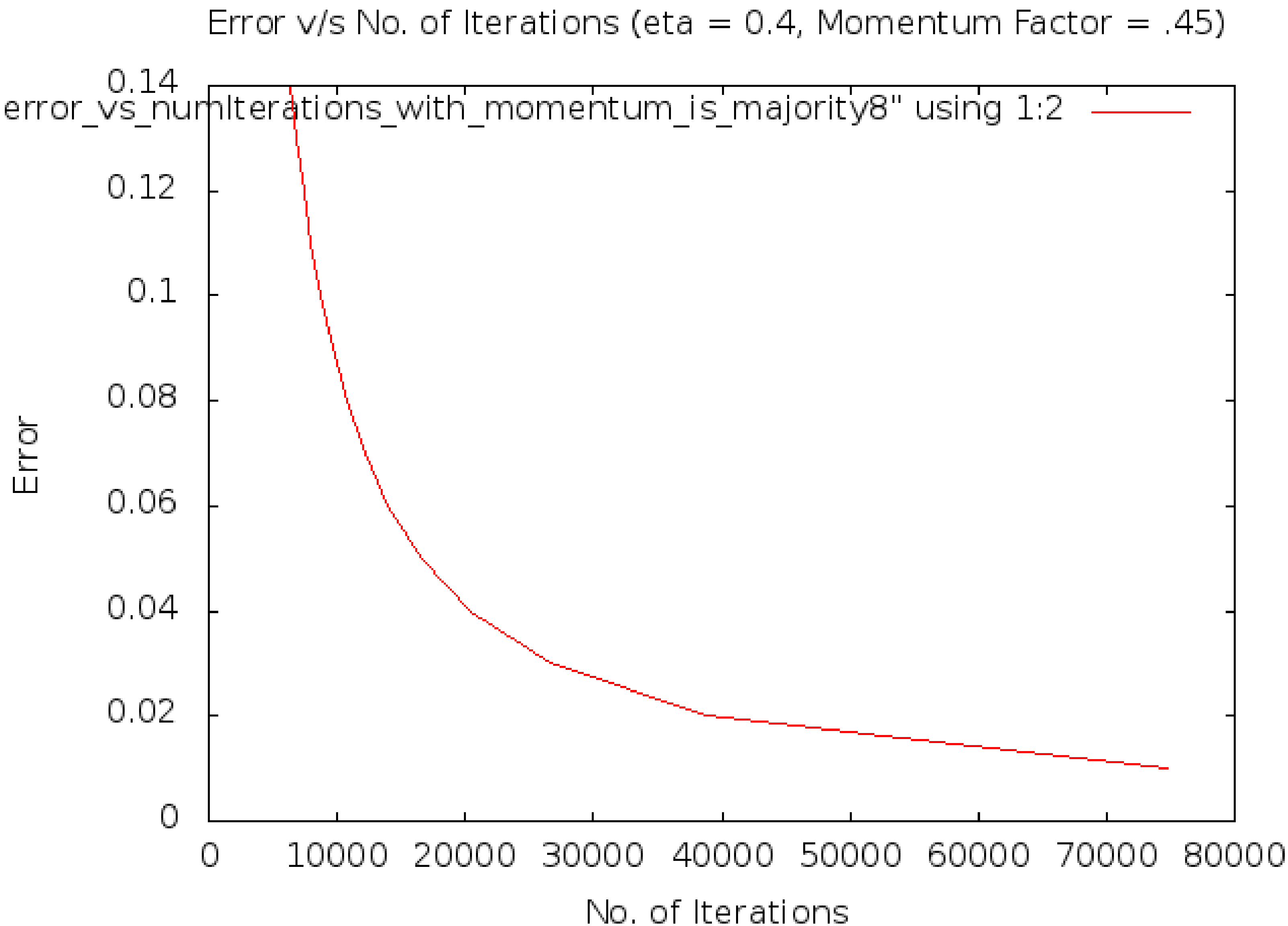
Effect of Momentum Factor (Majority)



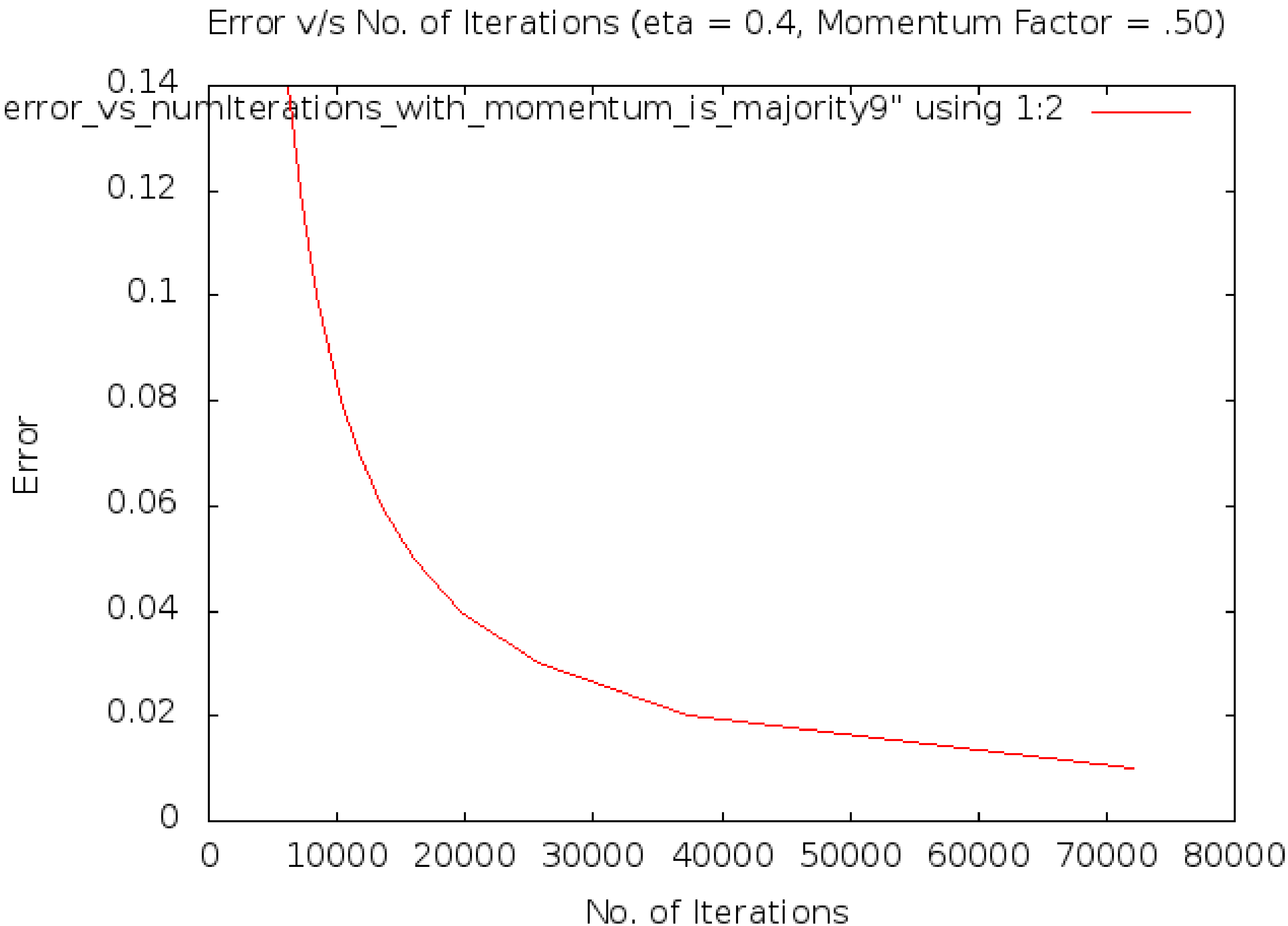
Effect of Momentum Factor (Majority)



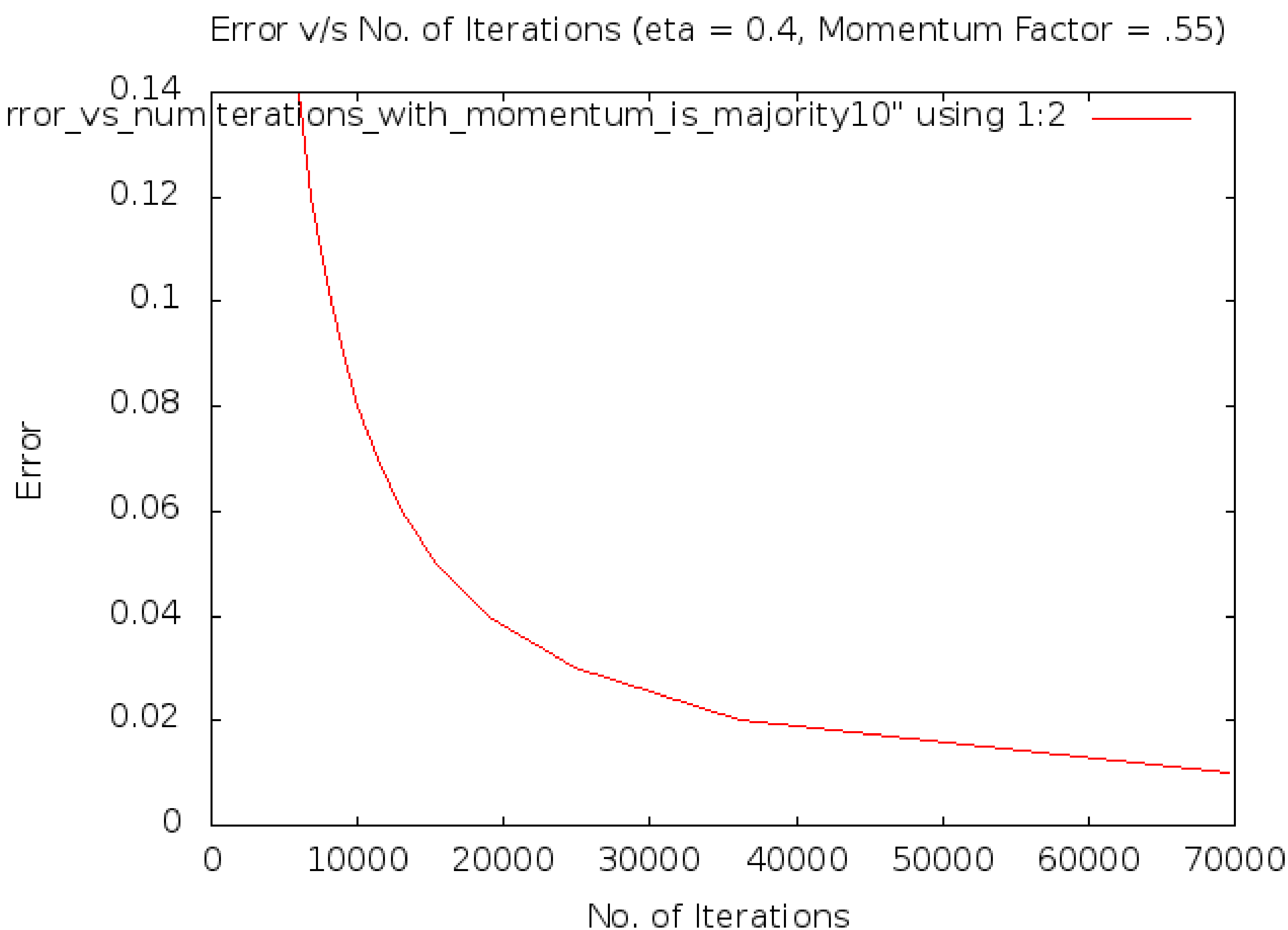
Effect of Momentum Factor (Majority)



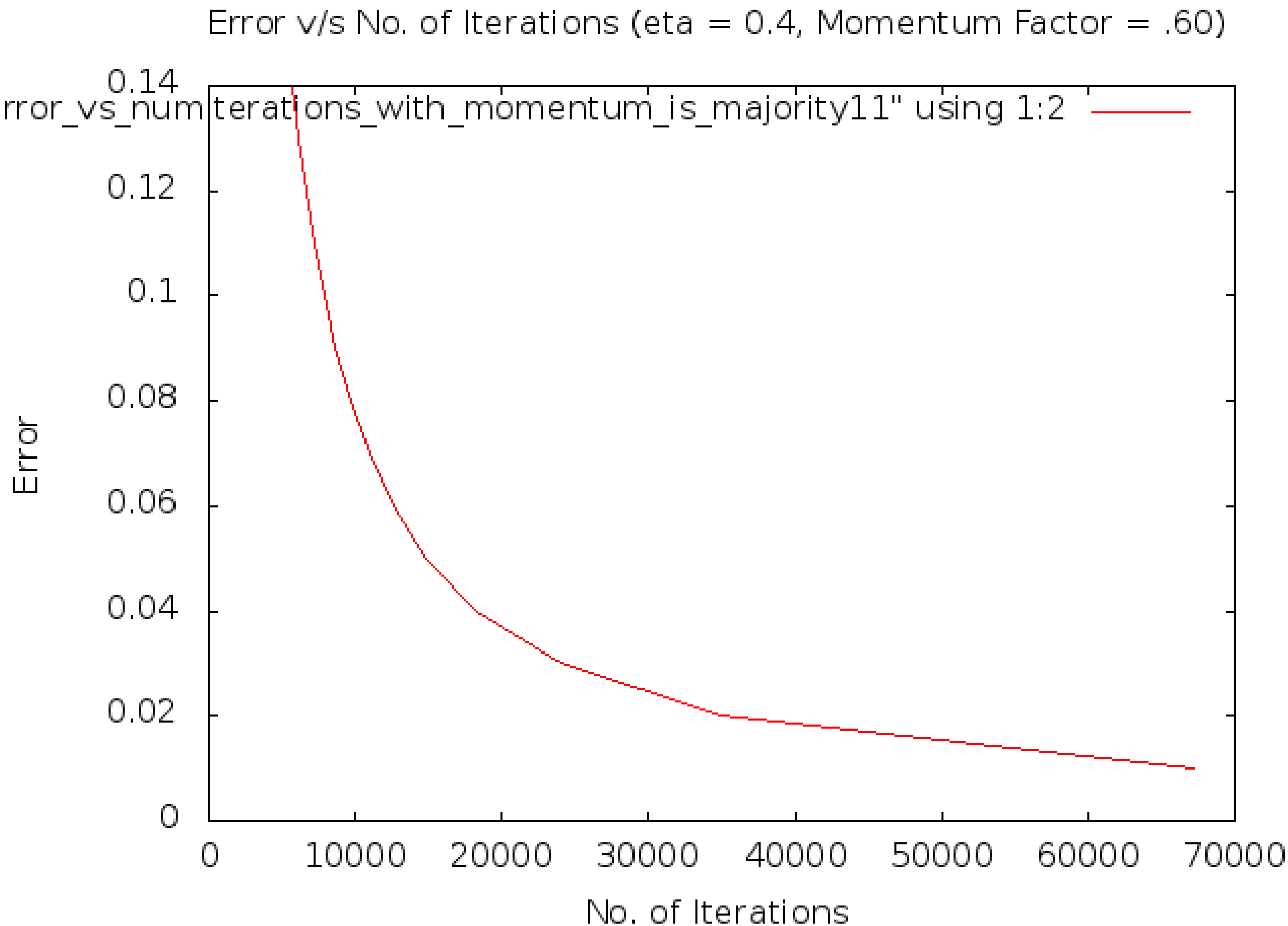
Effect of Momentum Factor (Majority)



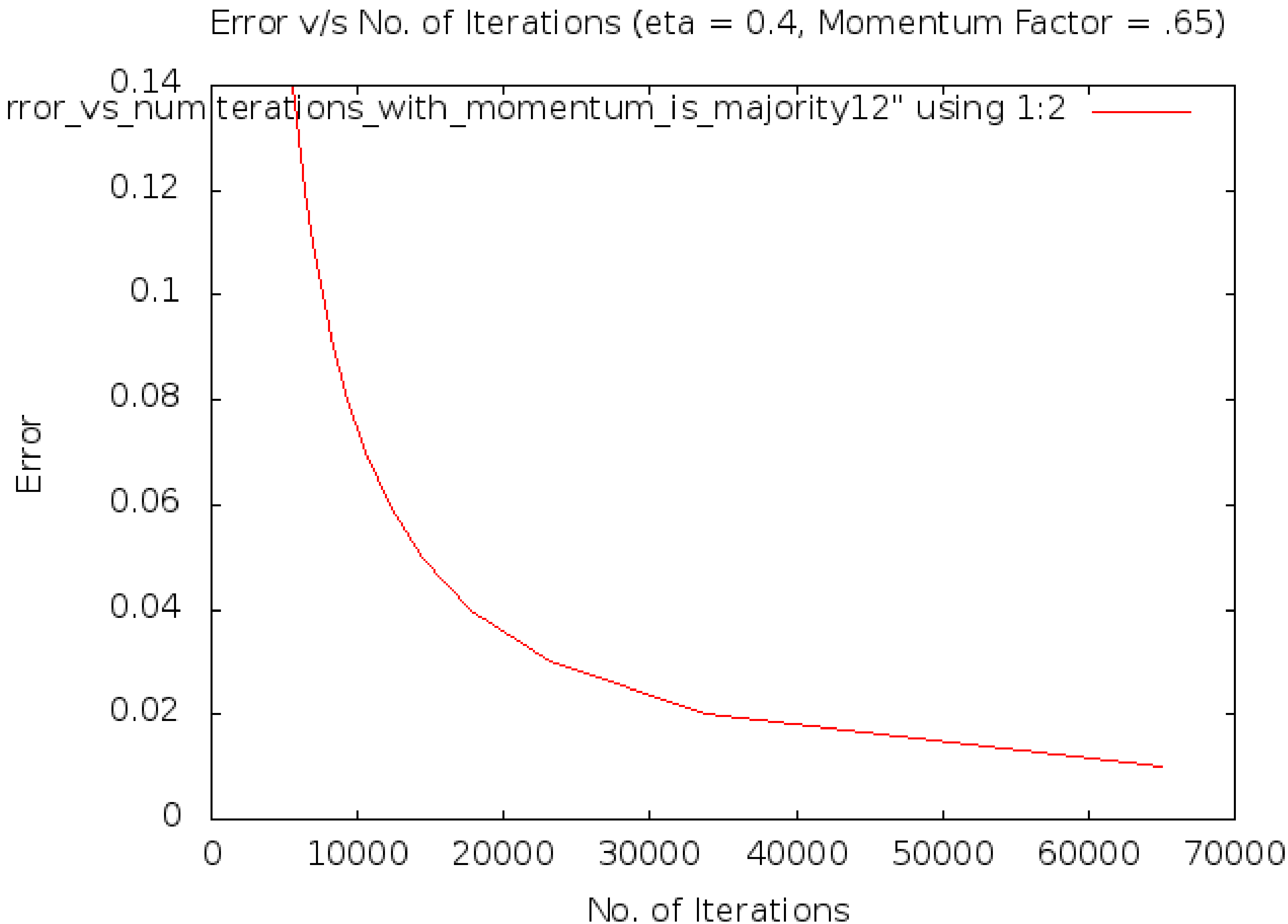
Effect of Momentum Factor (Majority)



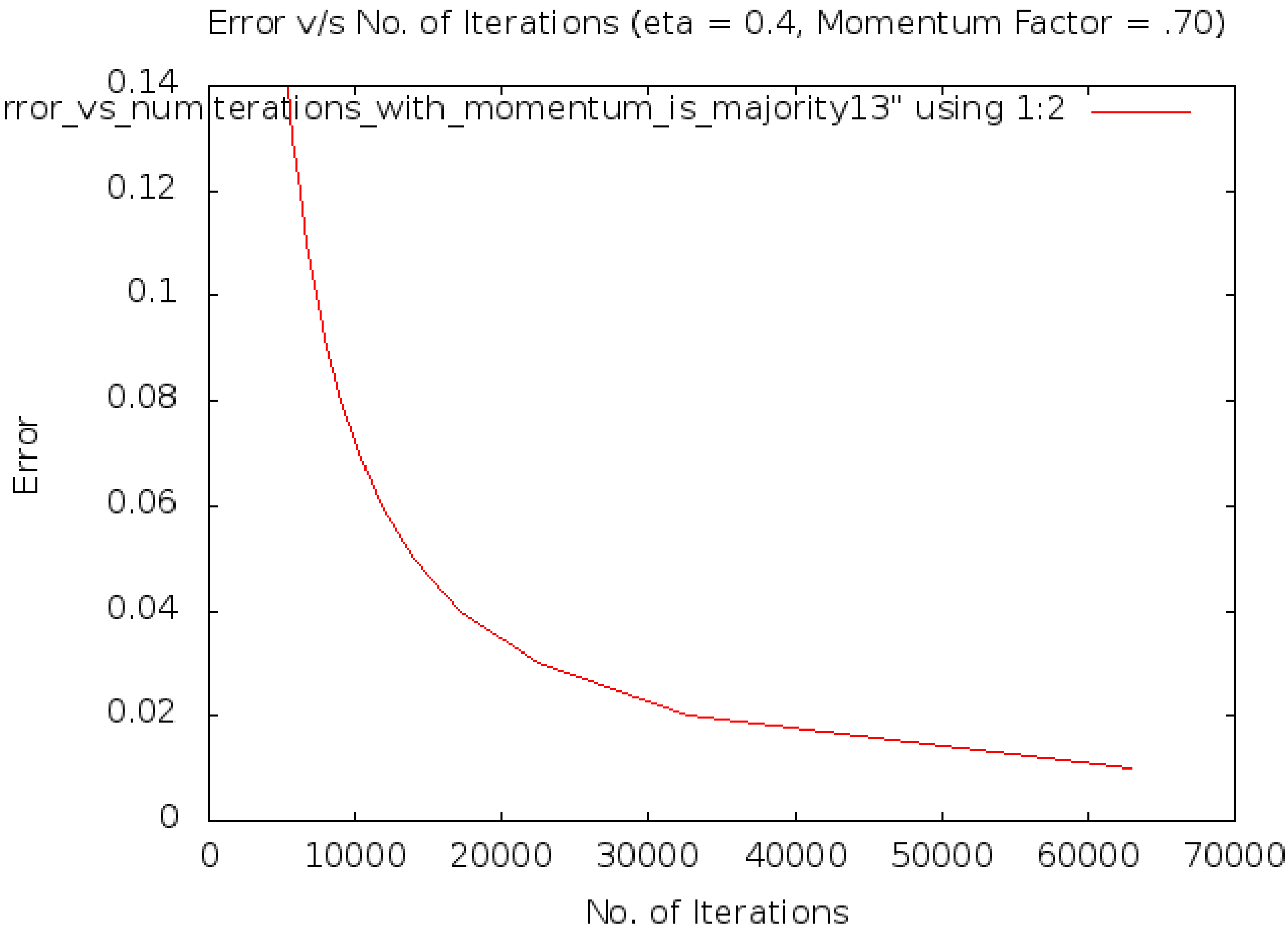
Effect of Momentum Factor (Majority)



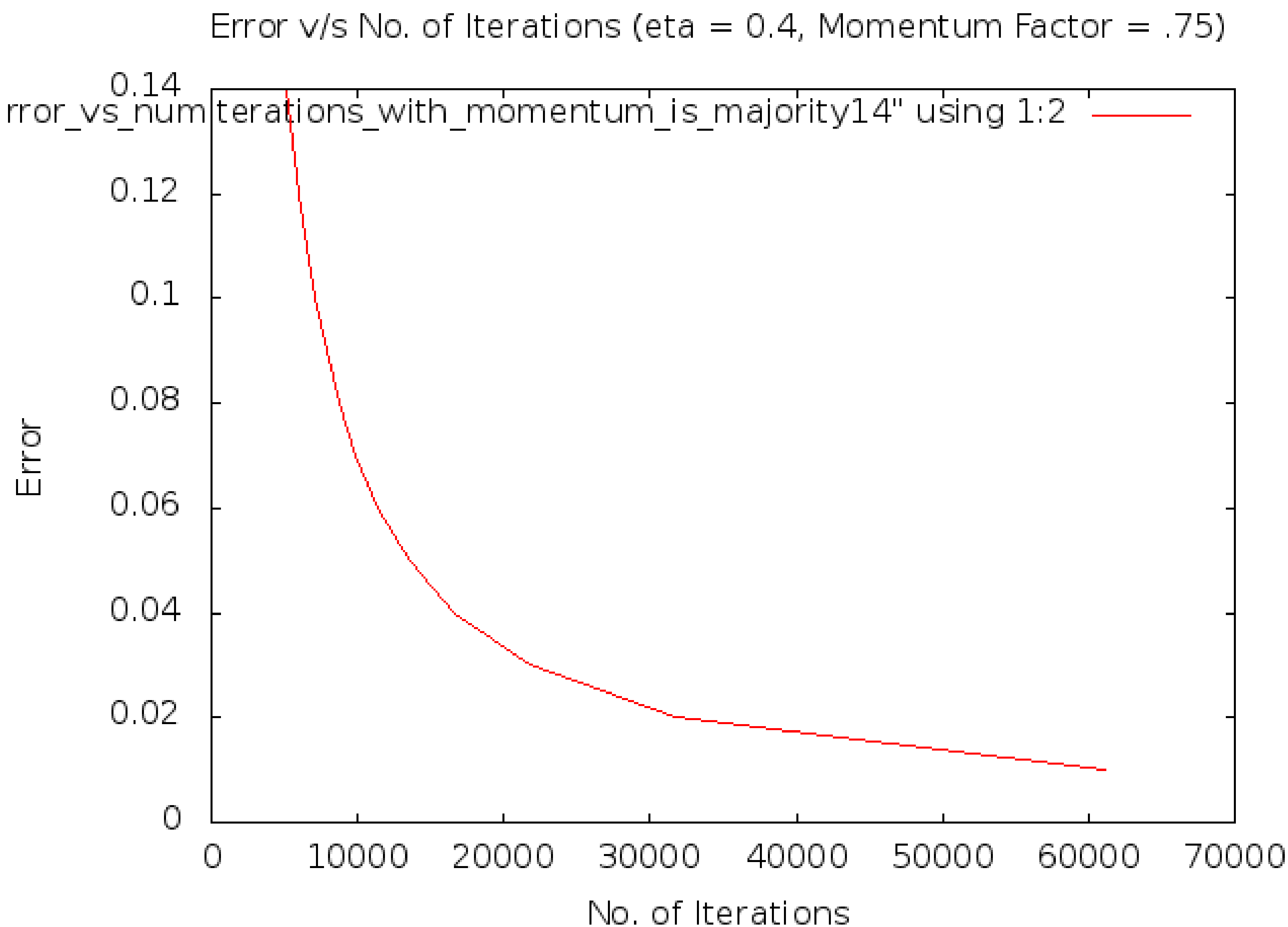
Effect of Momentum Factor (Majority)



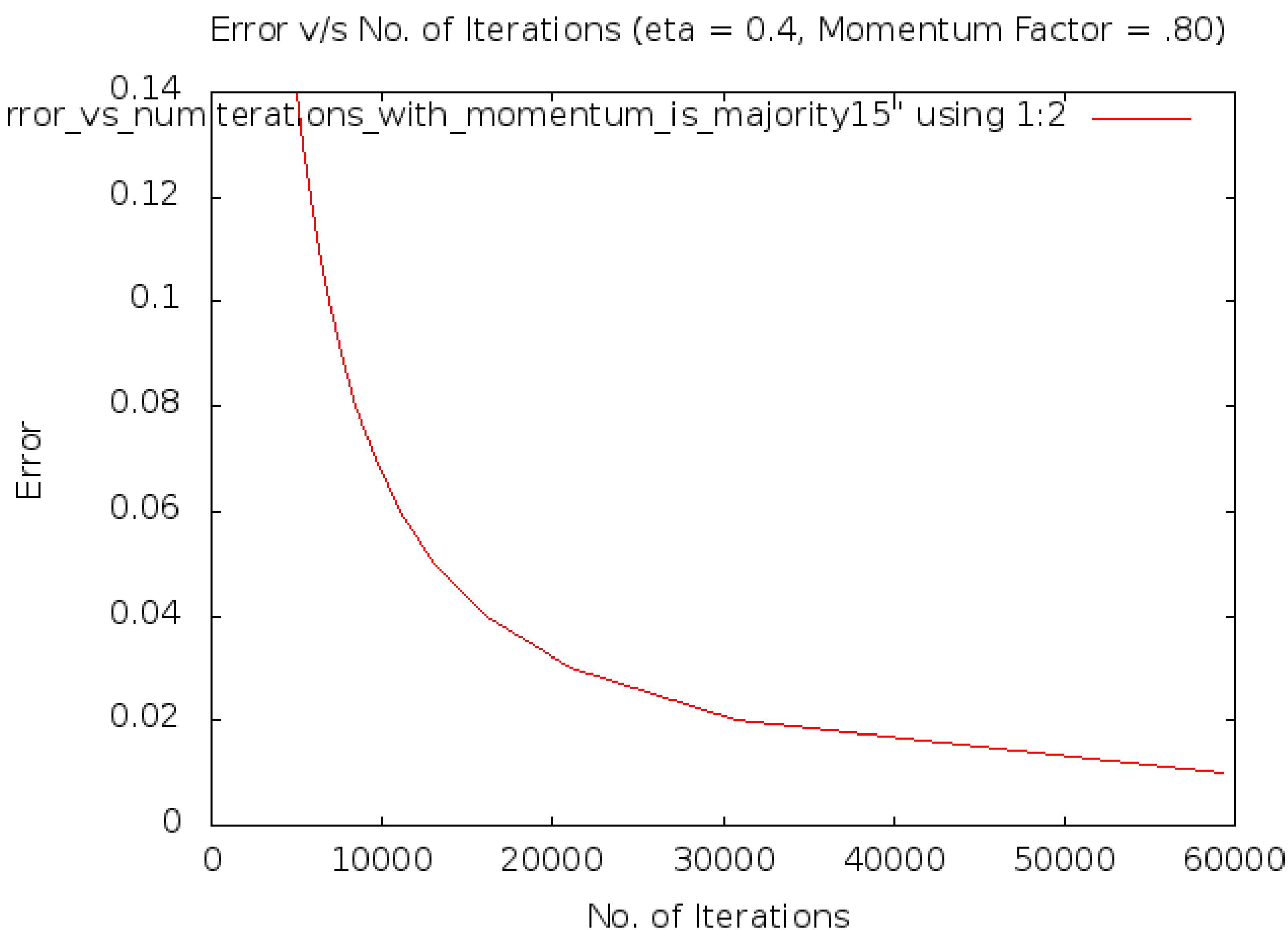
Effect of Momentum Factor (Majority)



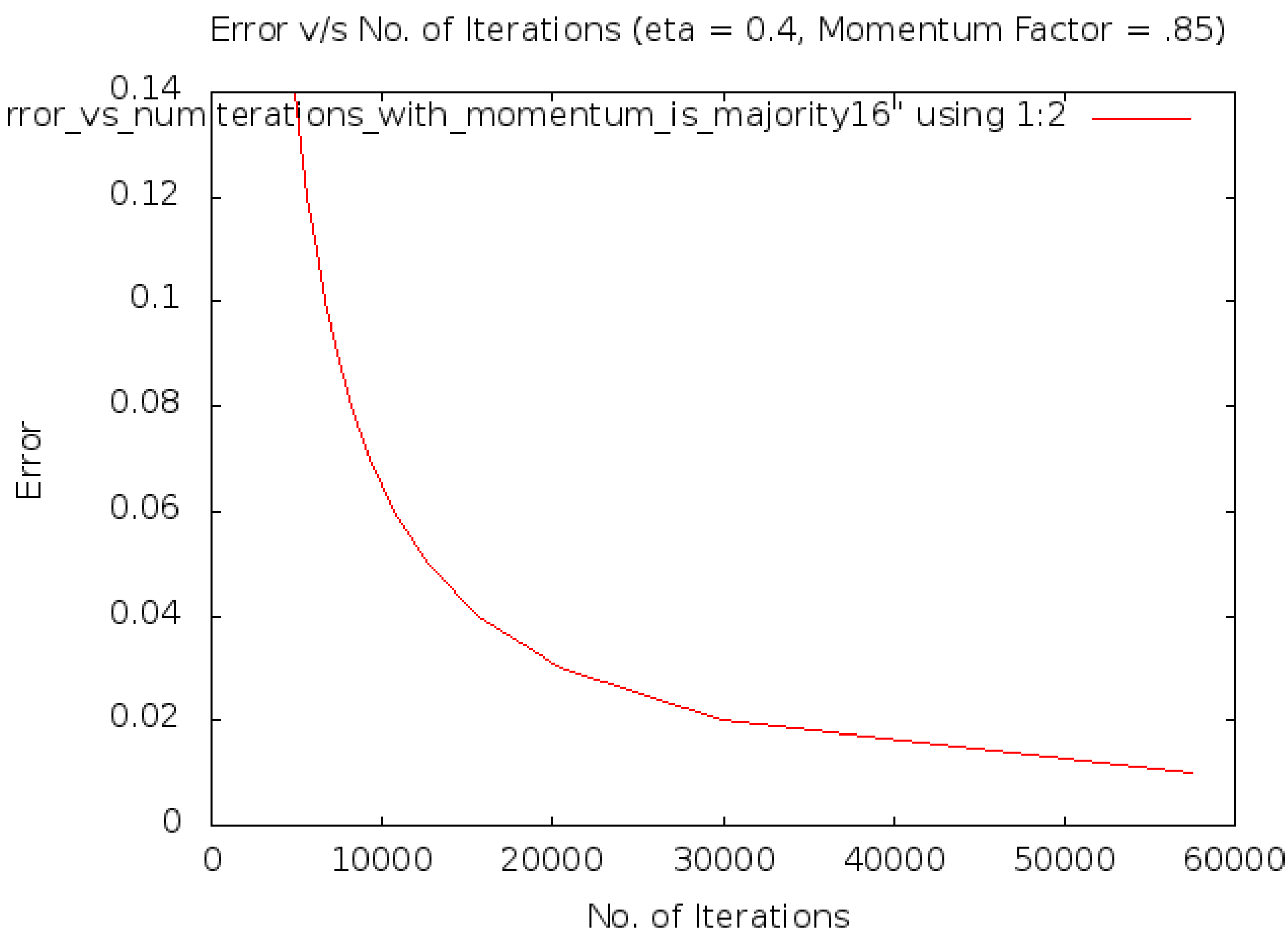
Effect of Momentum Factor (Majority)



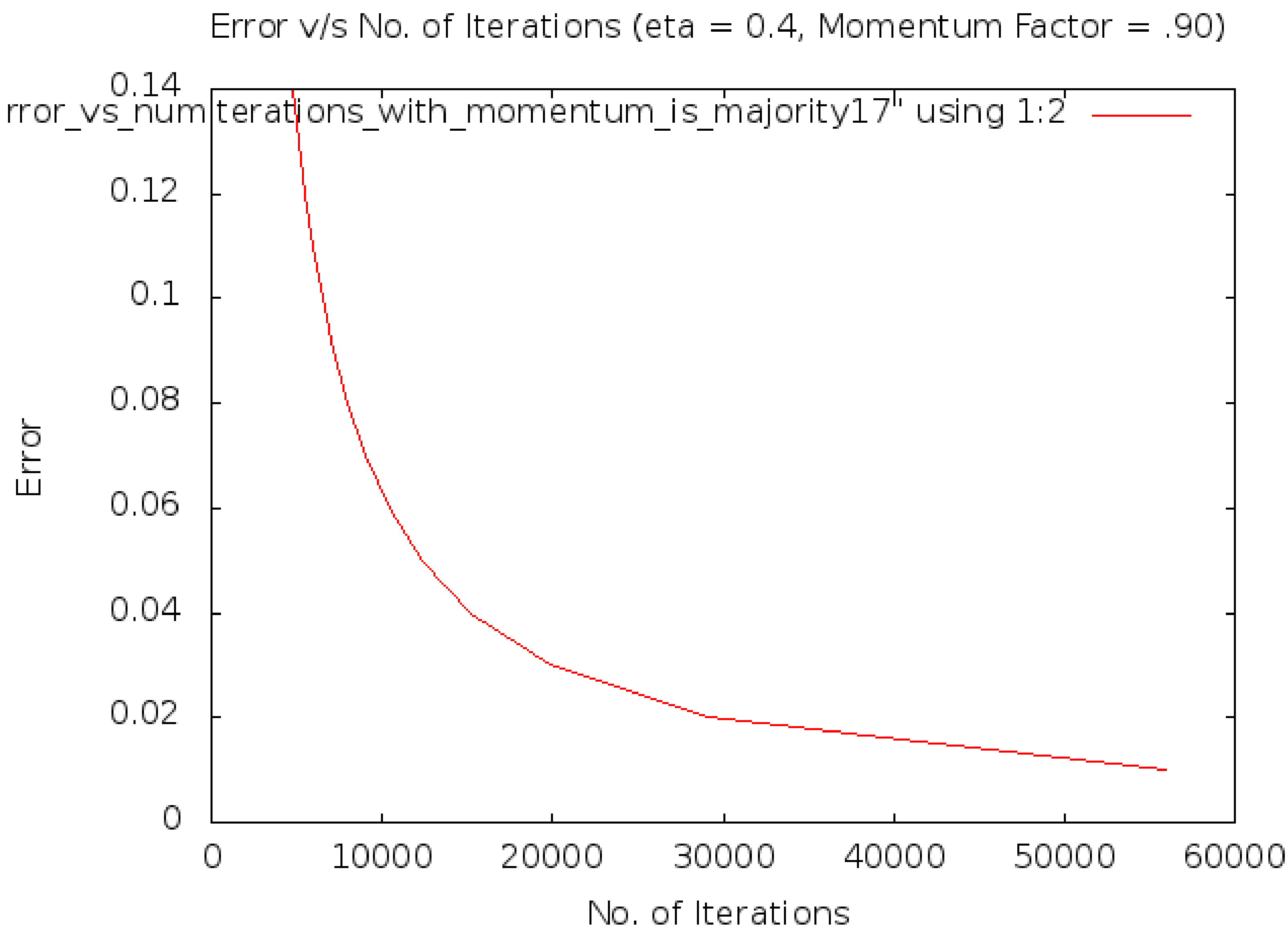
Effect of Momentum Factor (Majority)



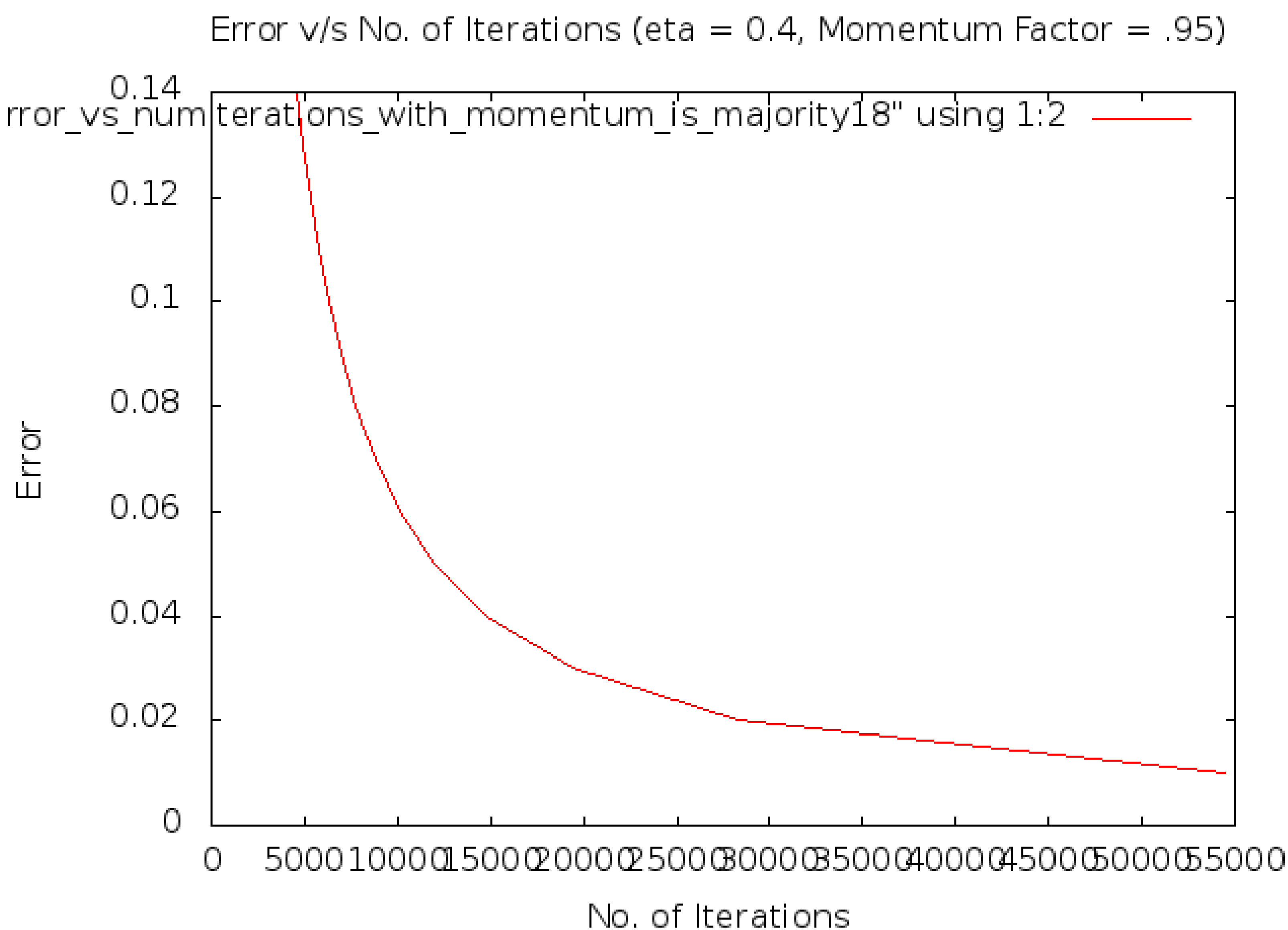
Effect of Momentum Factor (Majority)



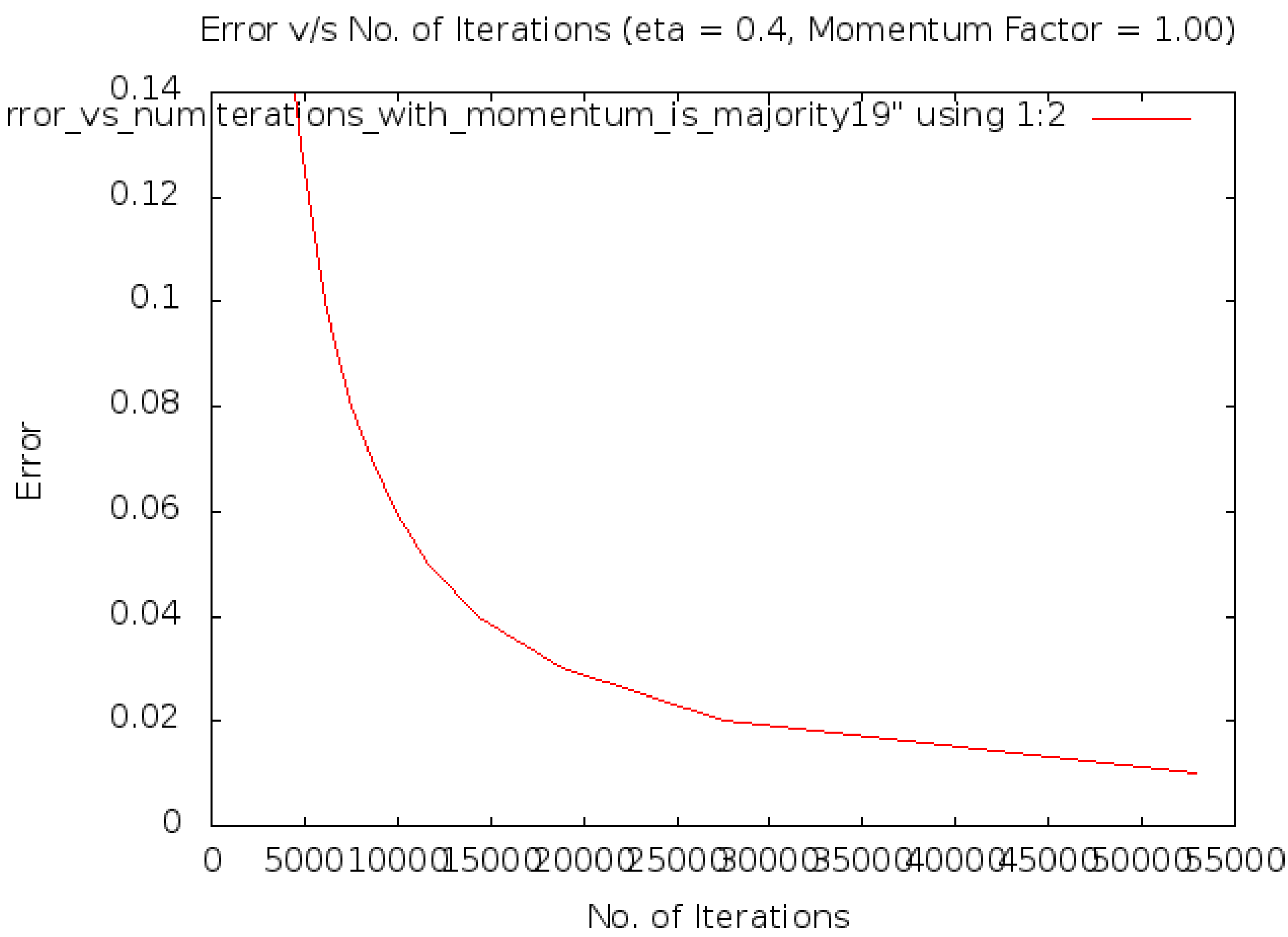
Effect of Momentum Factor (Majority)



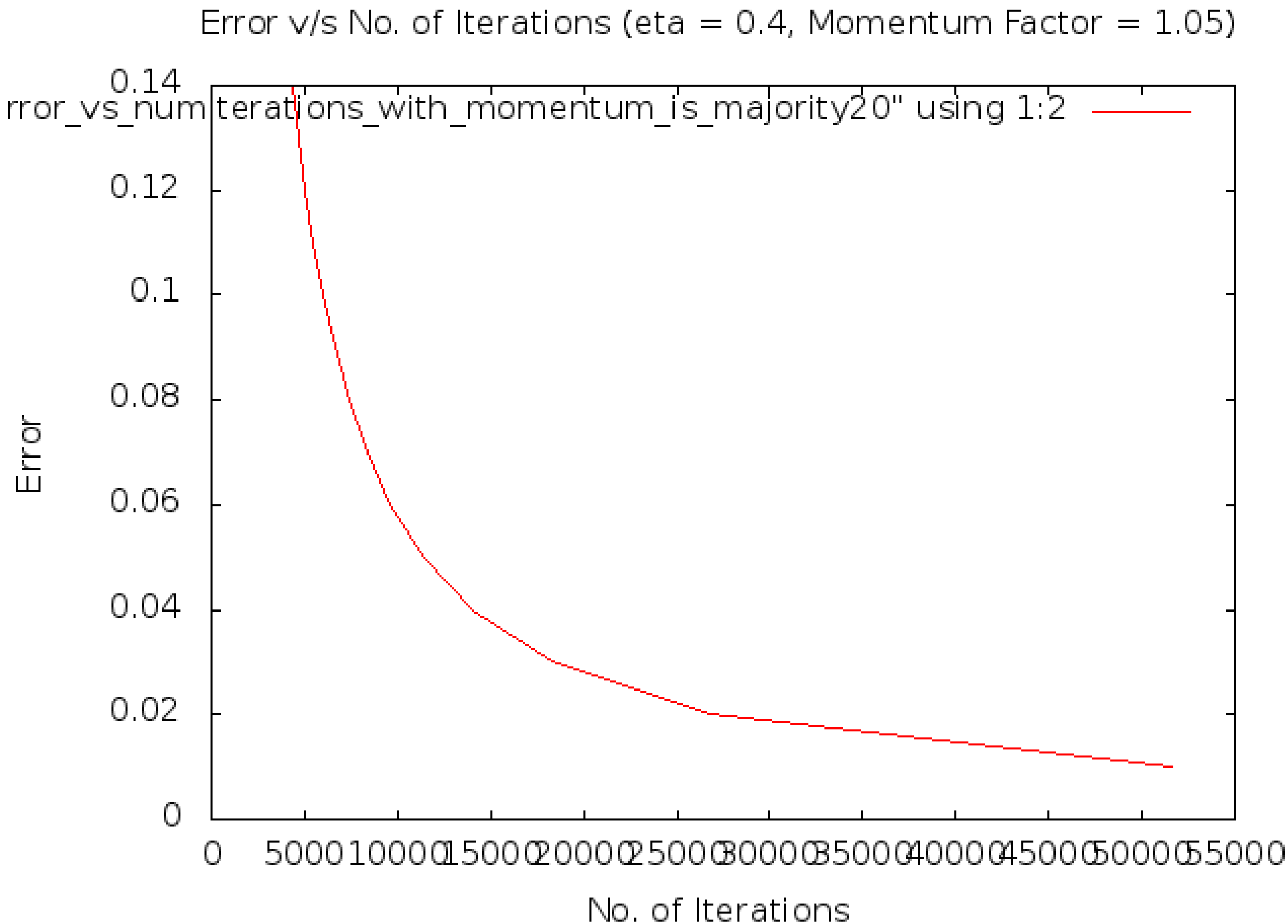
Effect of Momentum Factor (Majority)



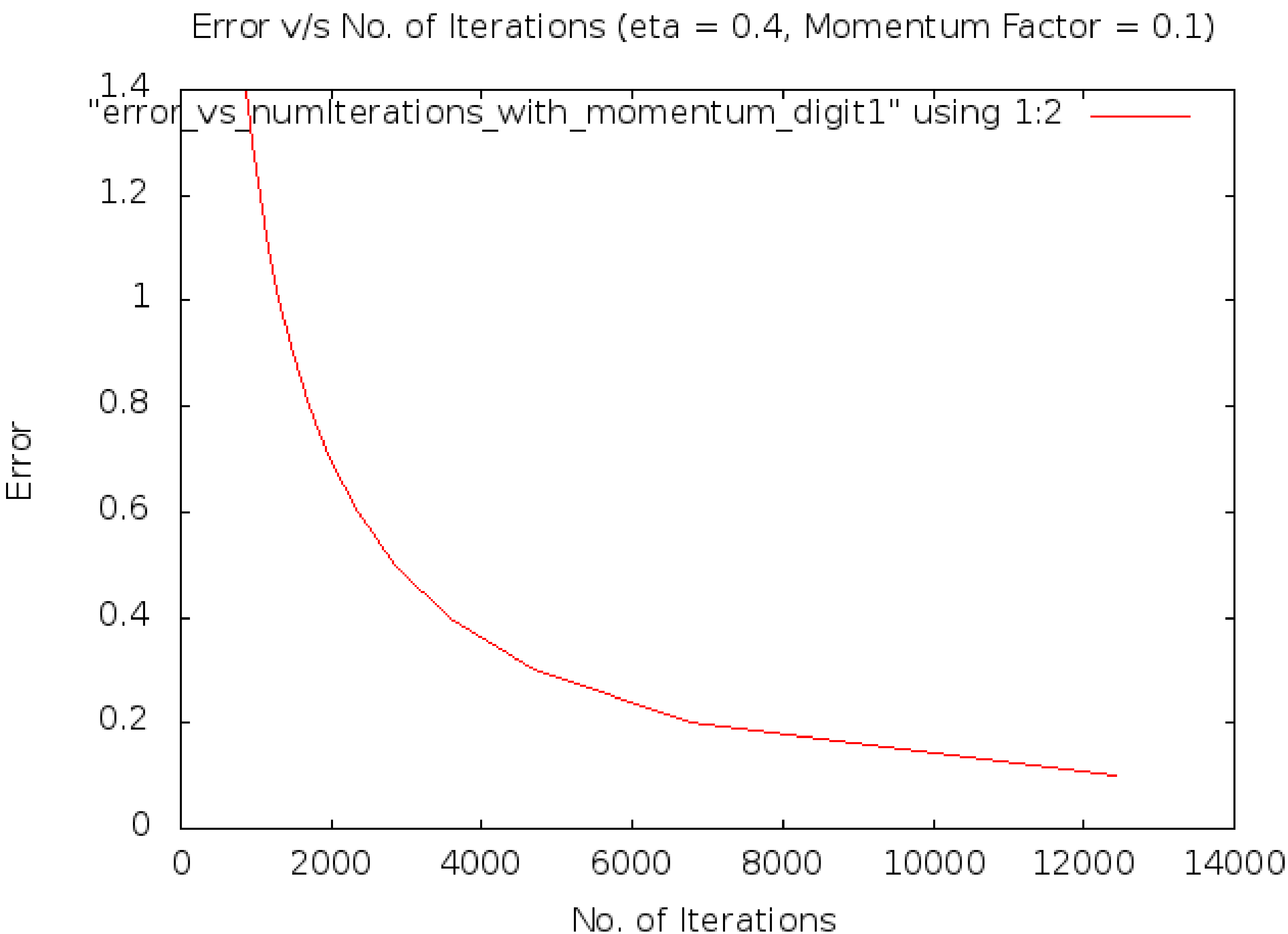
Effect of Momentum Factor (Majority)



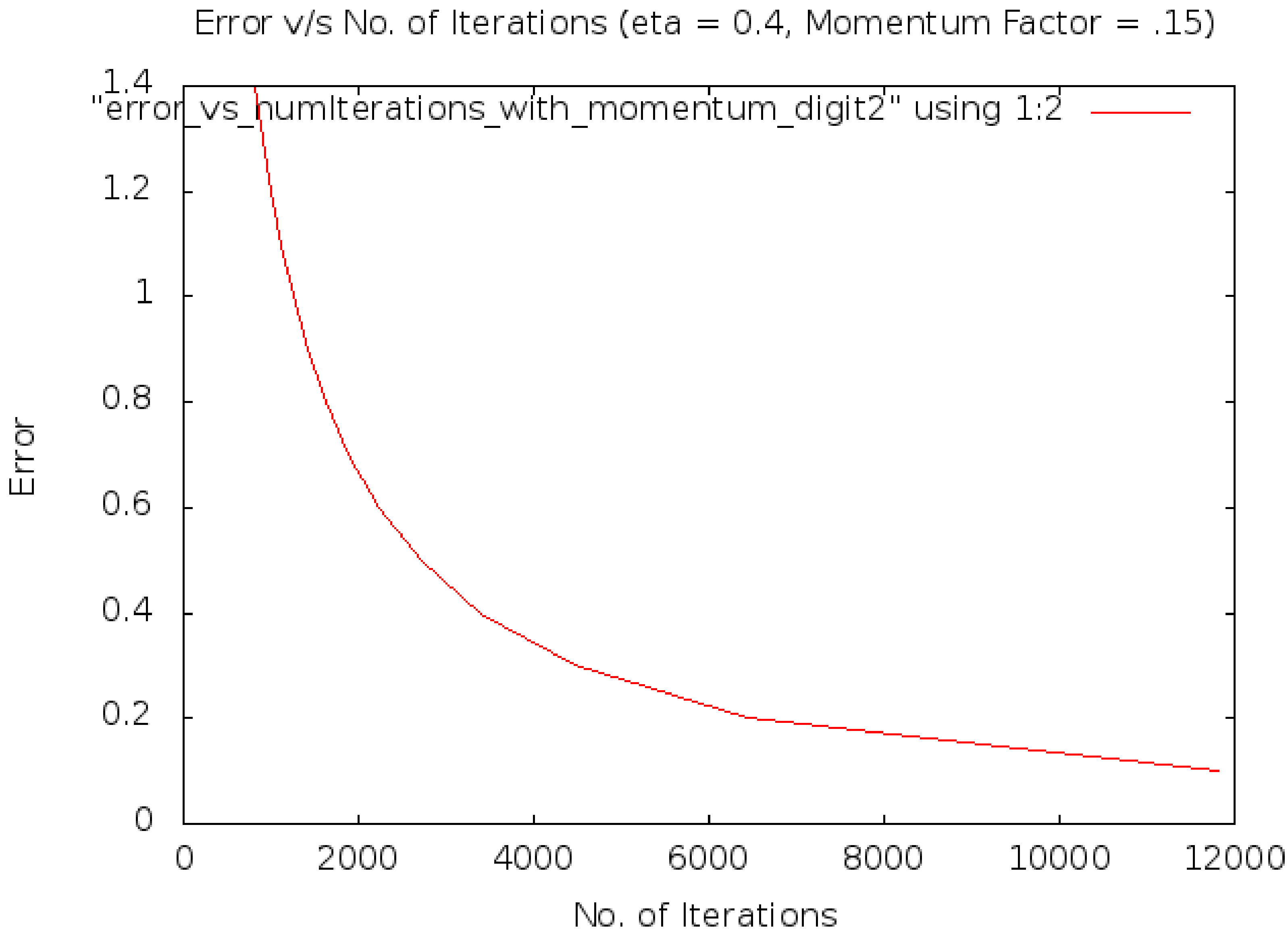
Effect of Momentum Factor (Majority)



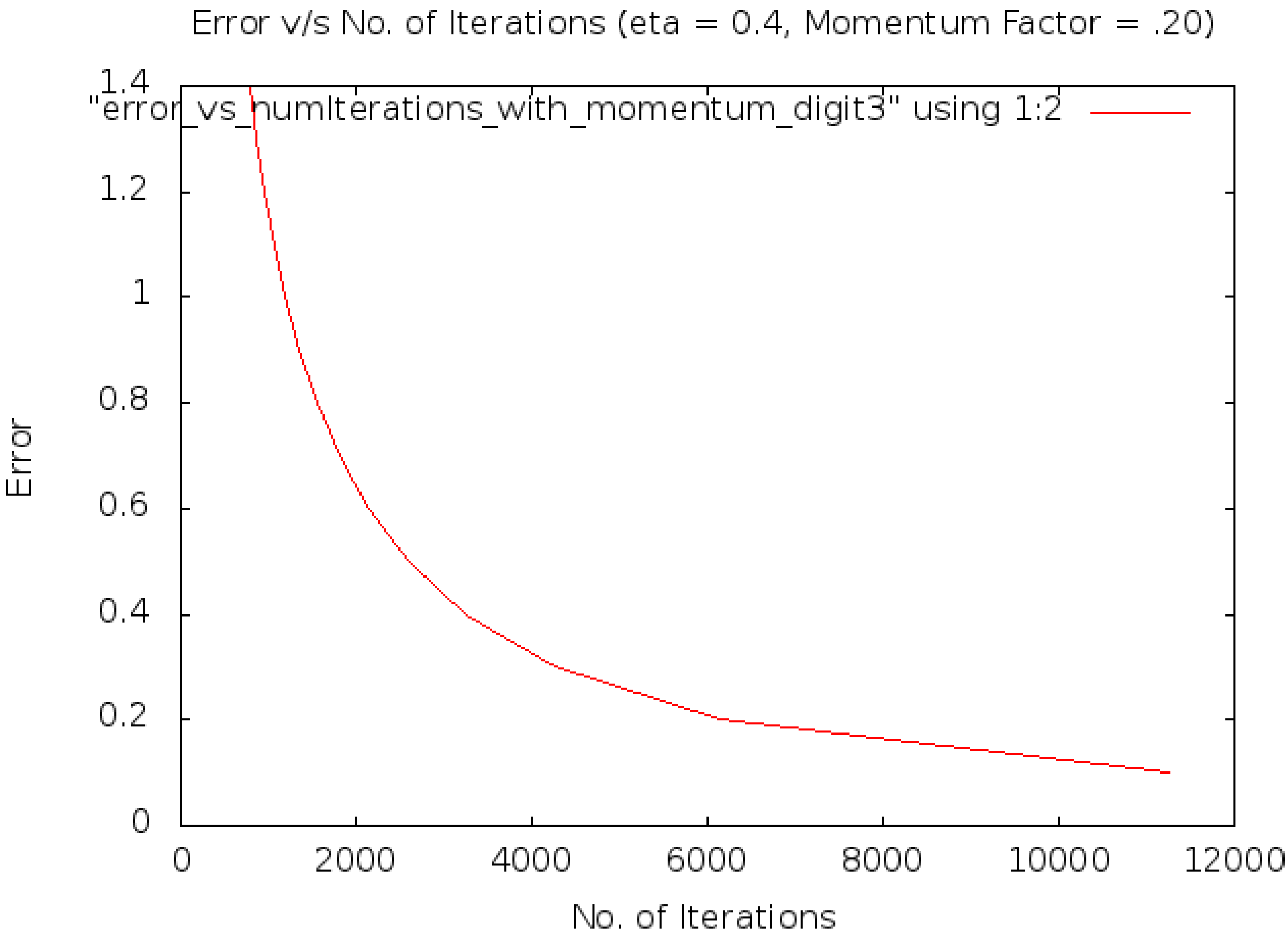
Effect of Momentum Factor (Digit Recogniser)



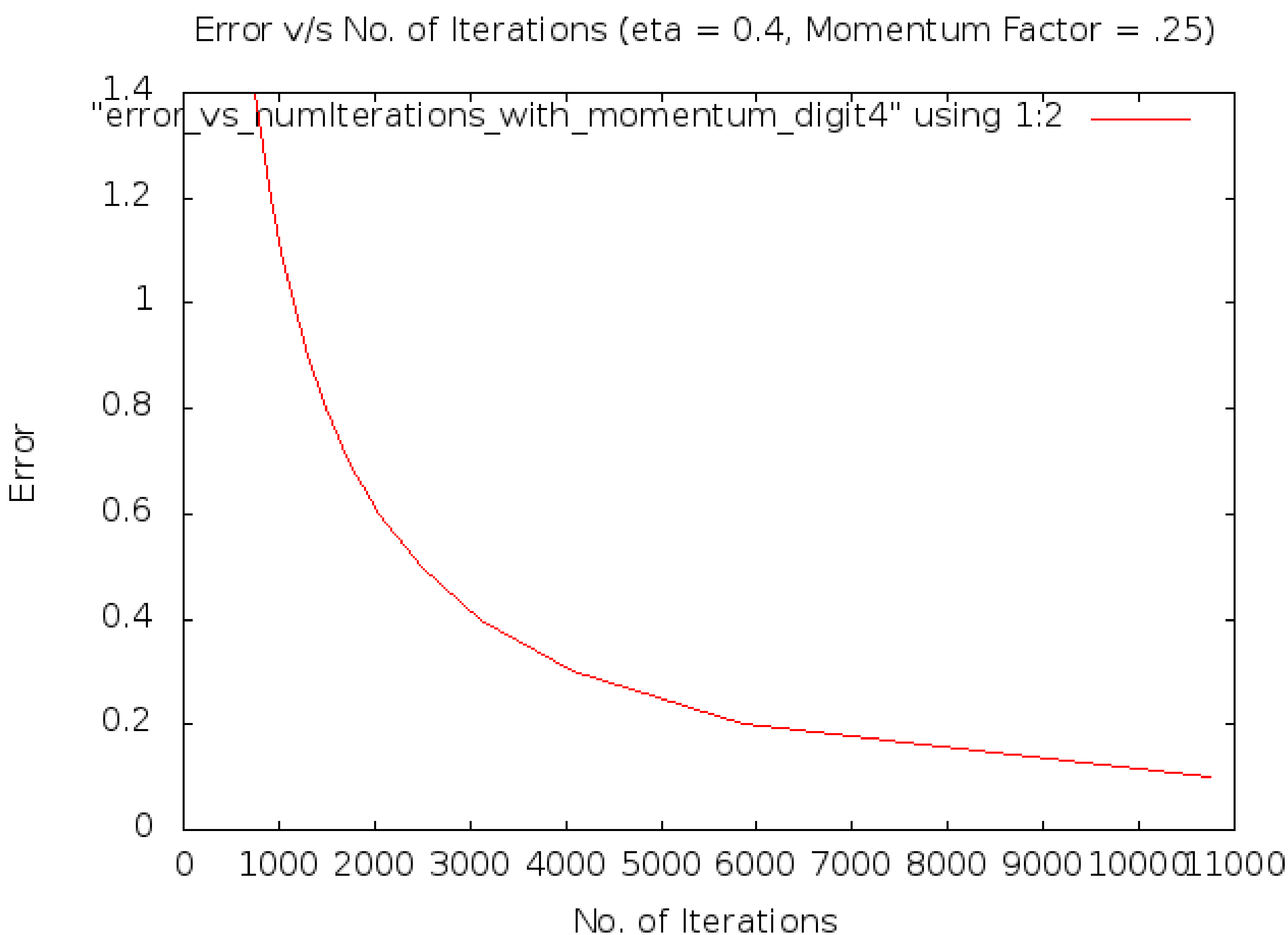
Effect of Momentum Factor (Digit Recogniser)



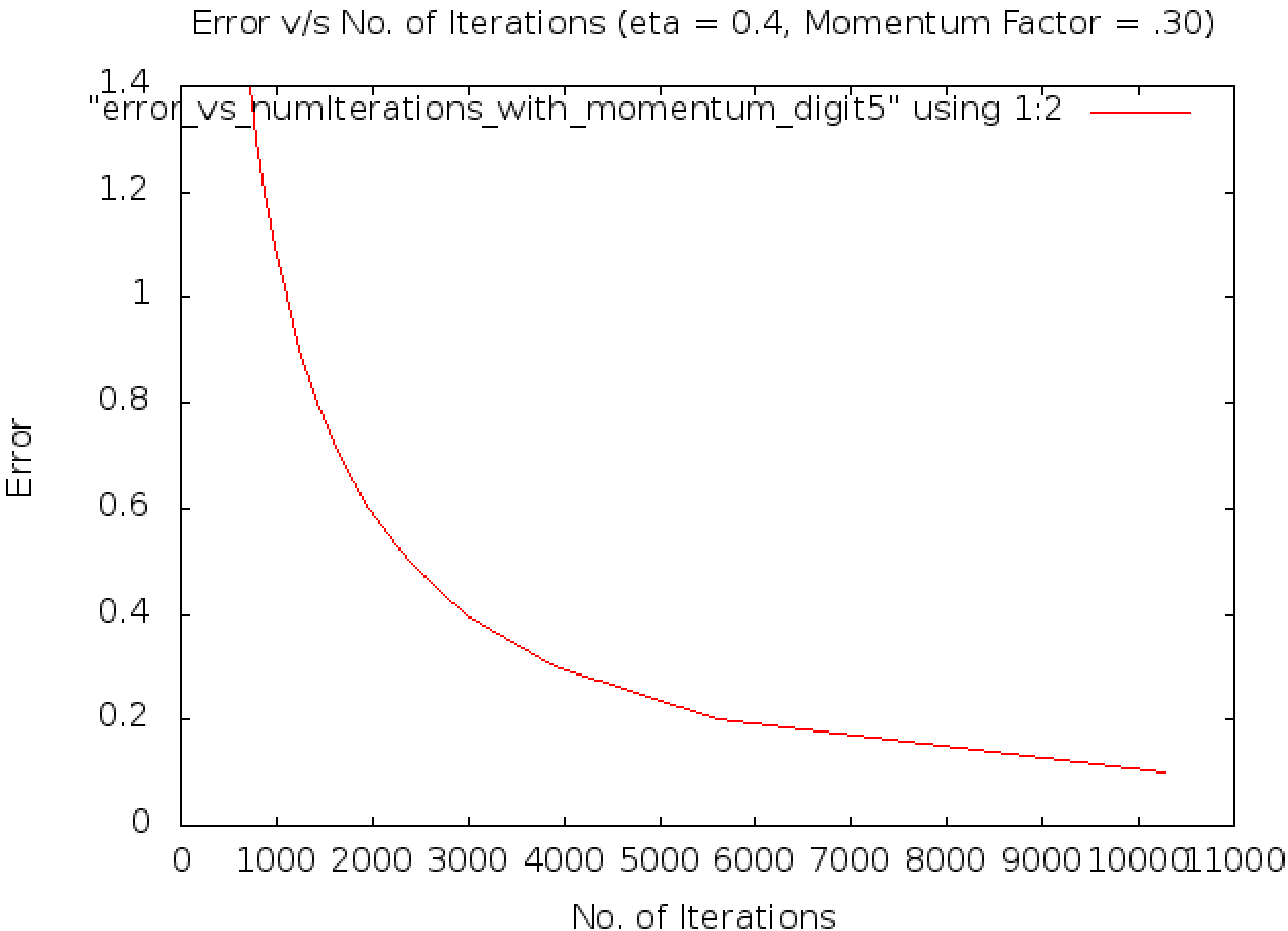
Effect of Momentum Factor (Digit Recogniser)



Effect of Momentum Factor (Digit Recogniser)

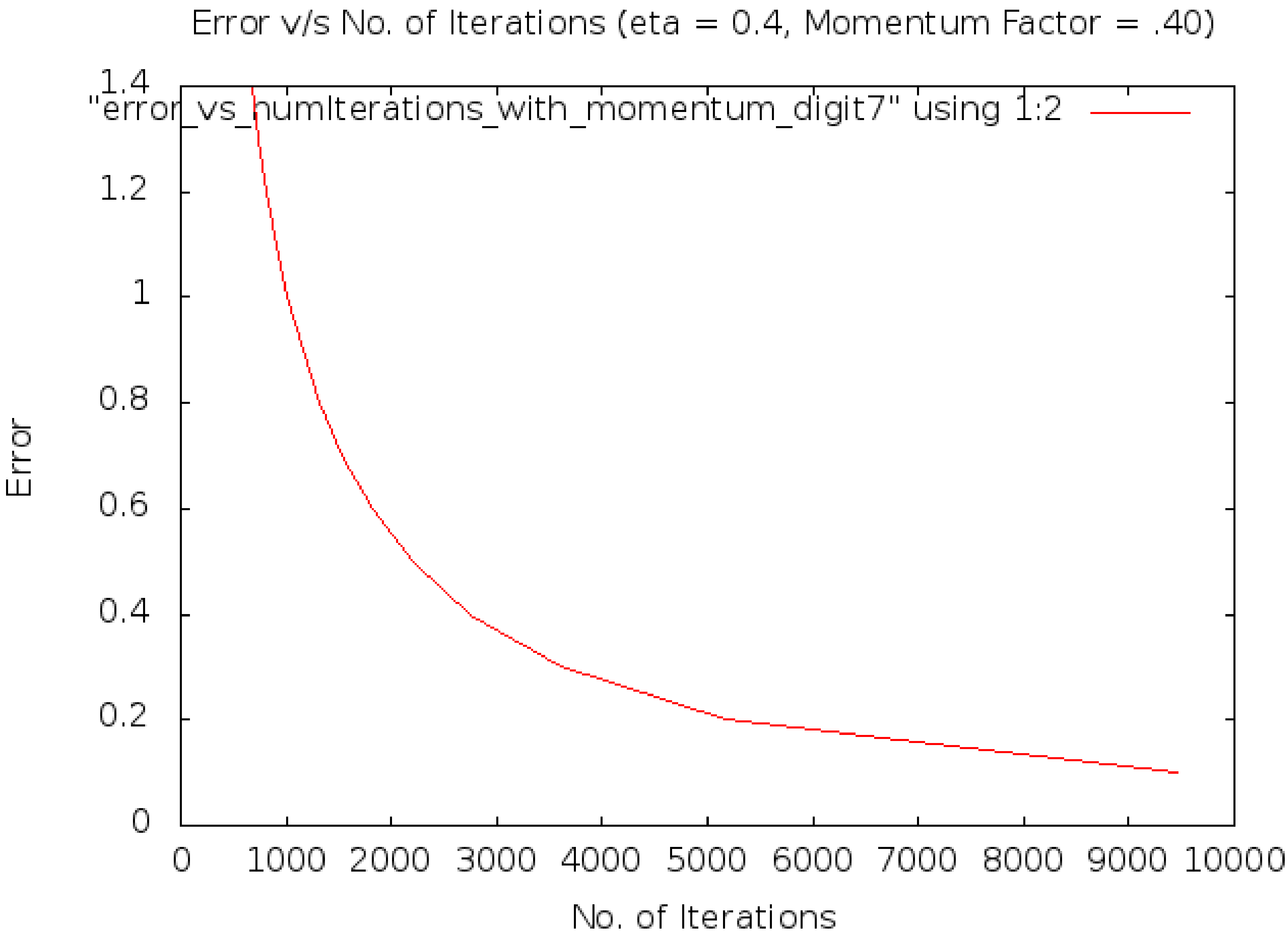


Effect of Momentum Factor (Digit Recogniser)

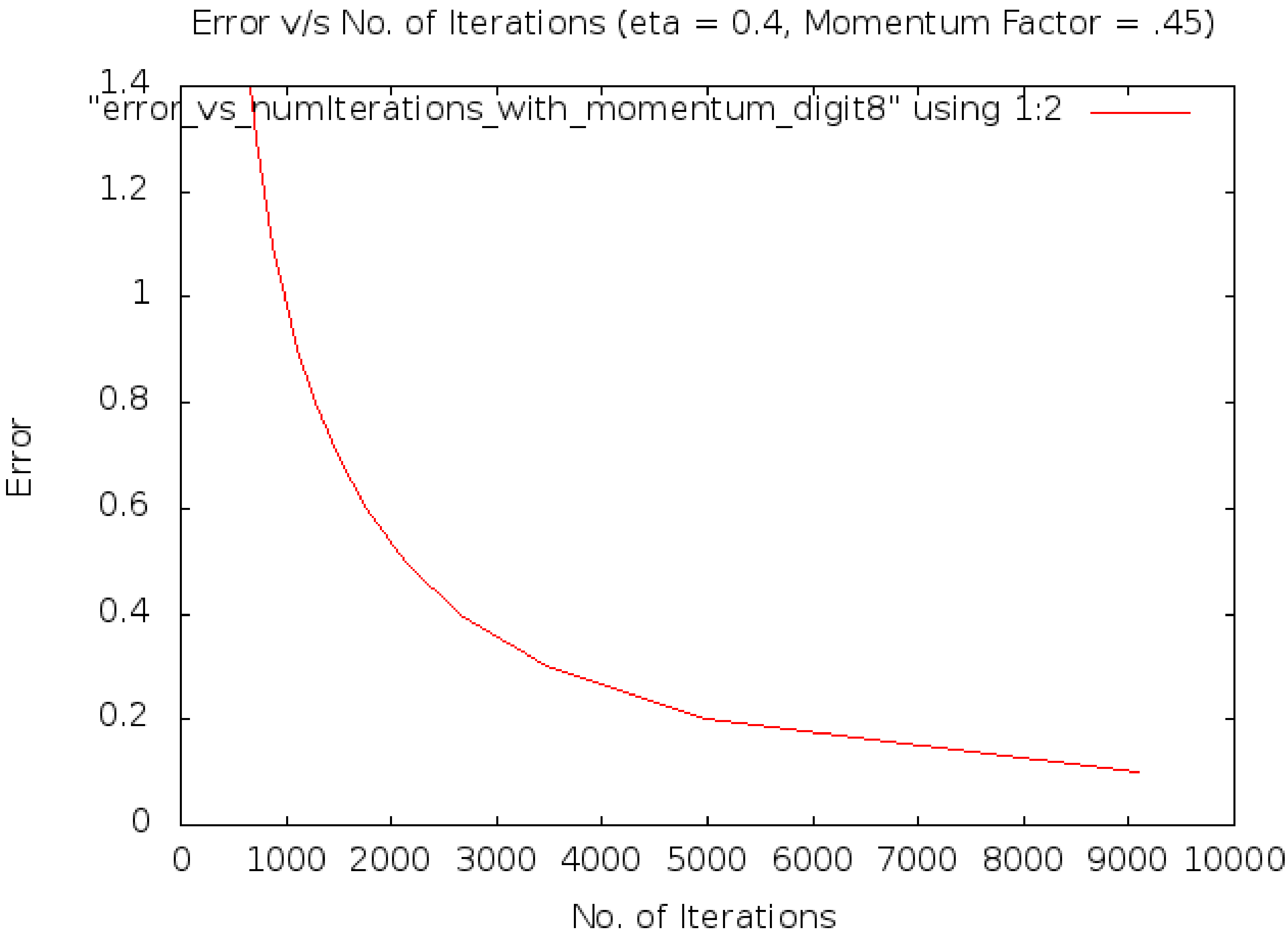


Effect of Momentum Factor (Digit Recogniser)

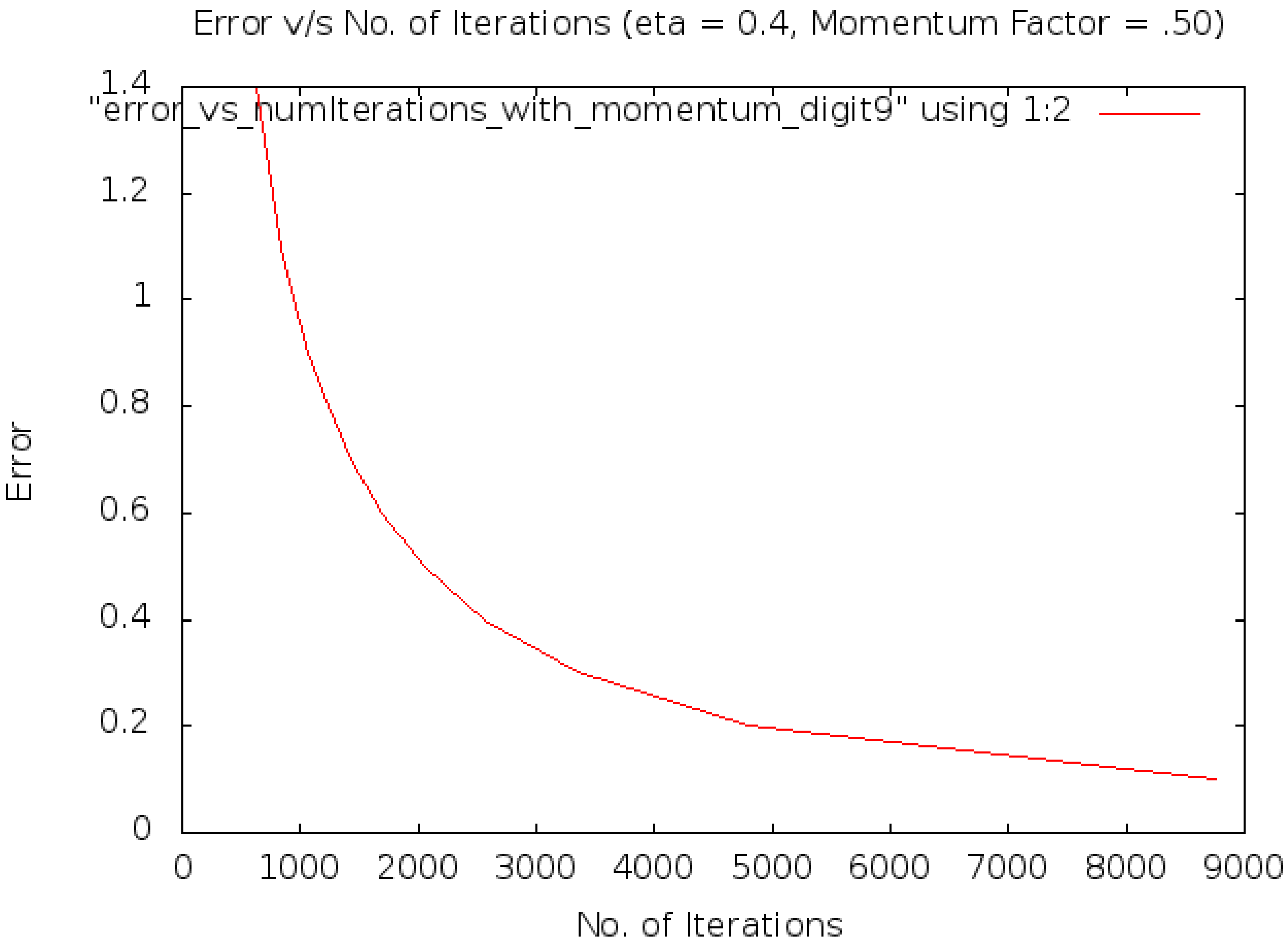
Effect of Momentum Factor (Digit Recogniser)



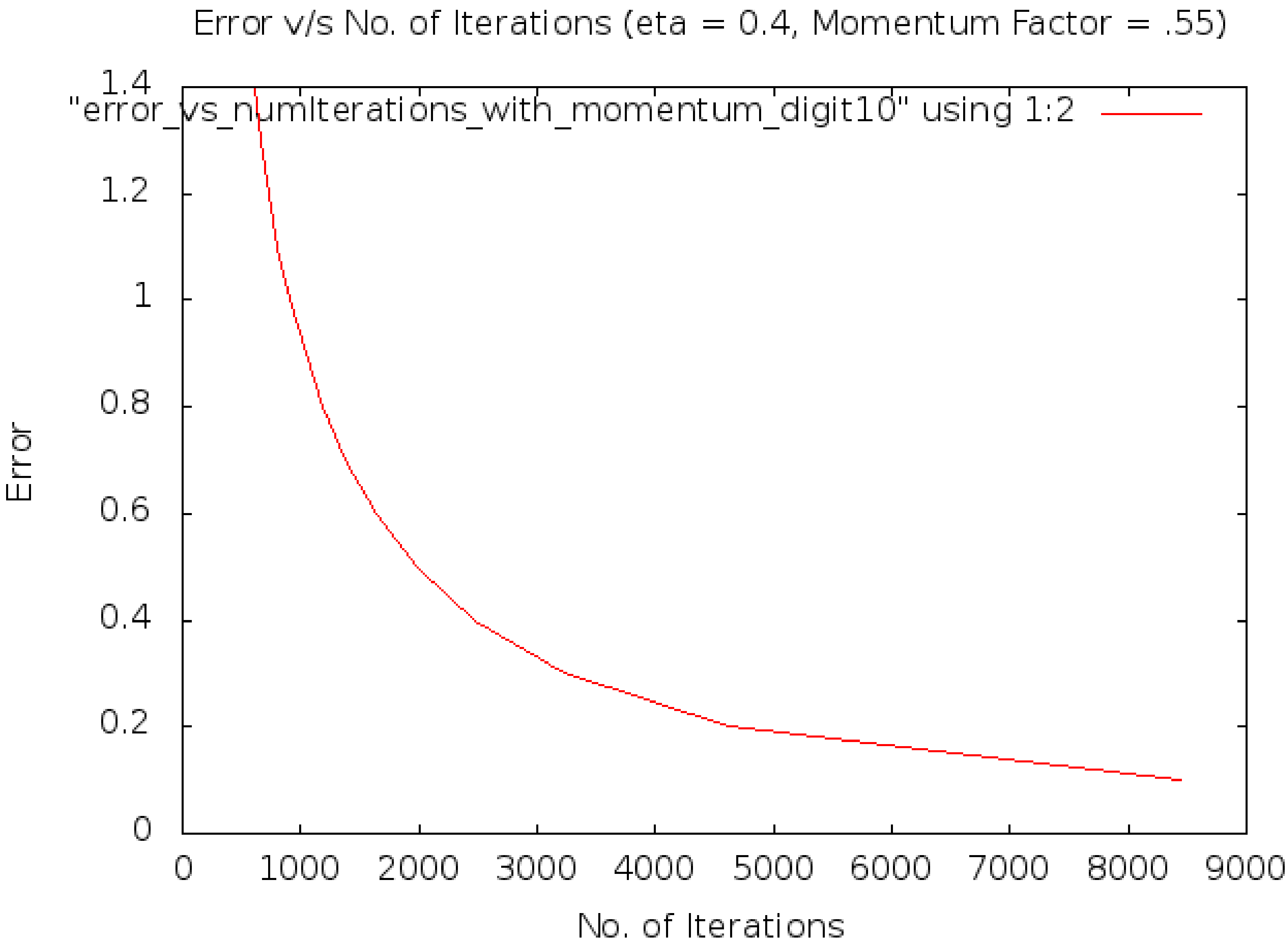
Effect of Momentum Factor (Digit Recogniser)



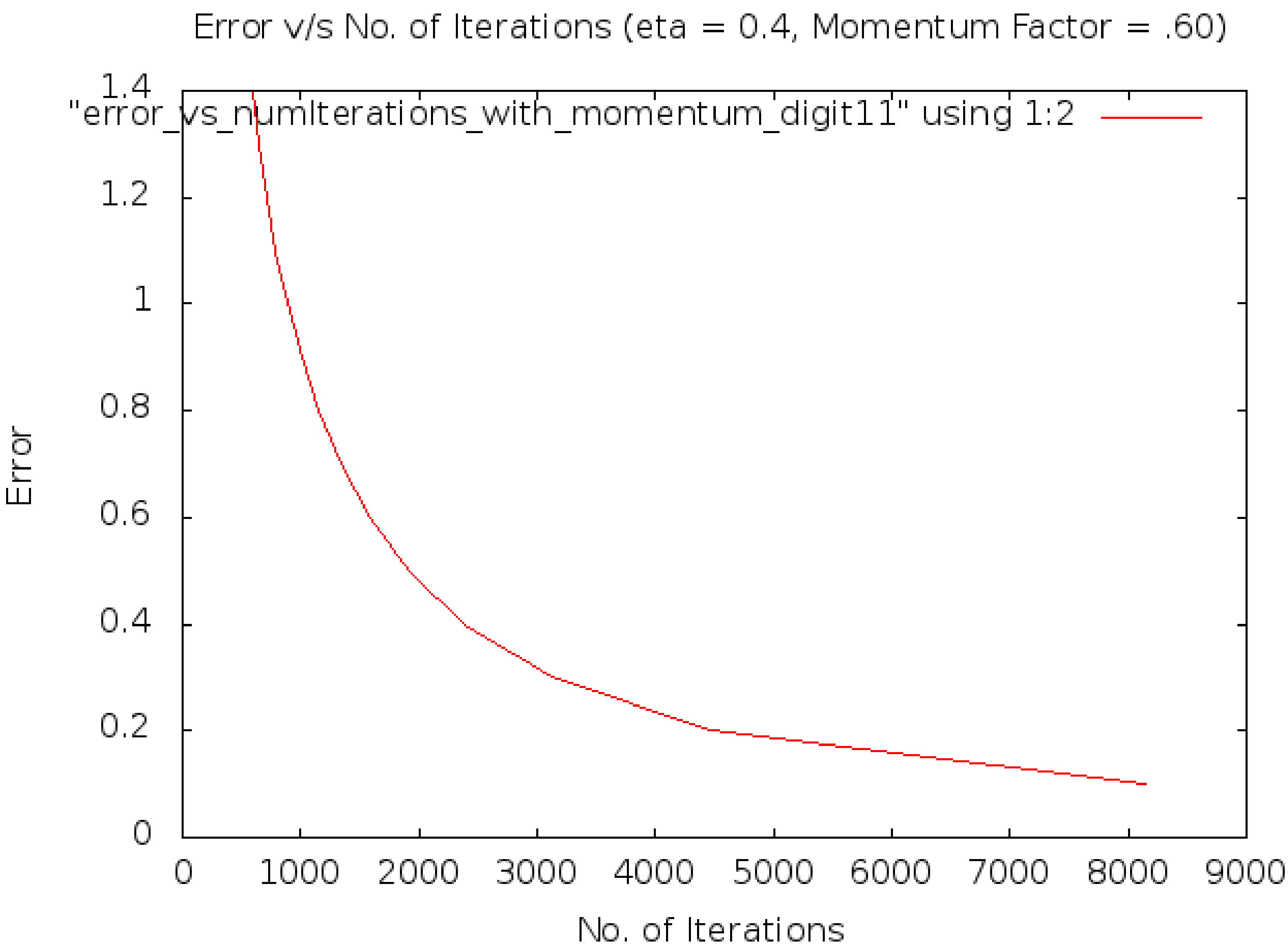
Effect of Momentum Factor (Digit Recogniser)



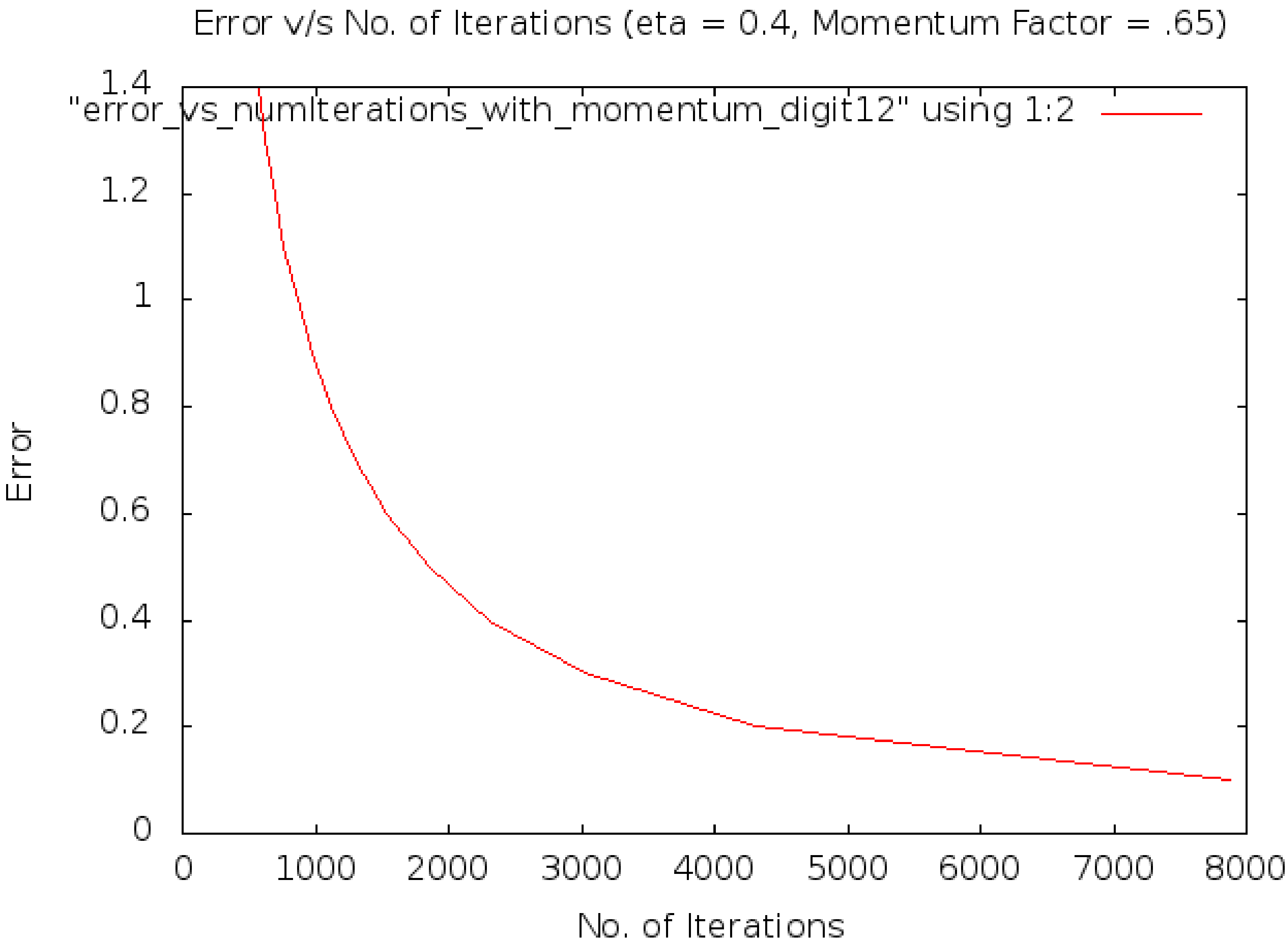
Effect of Momentum Factor (Digit Recogniser)



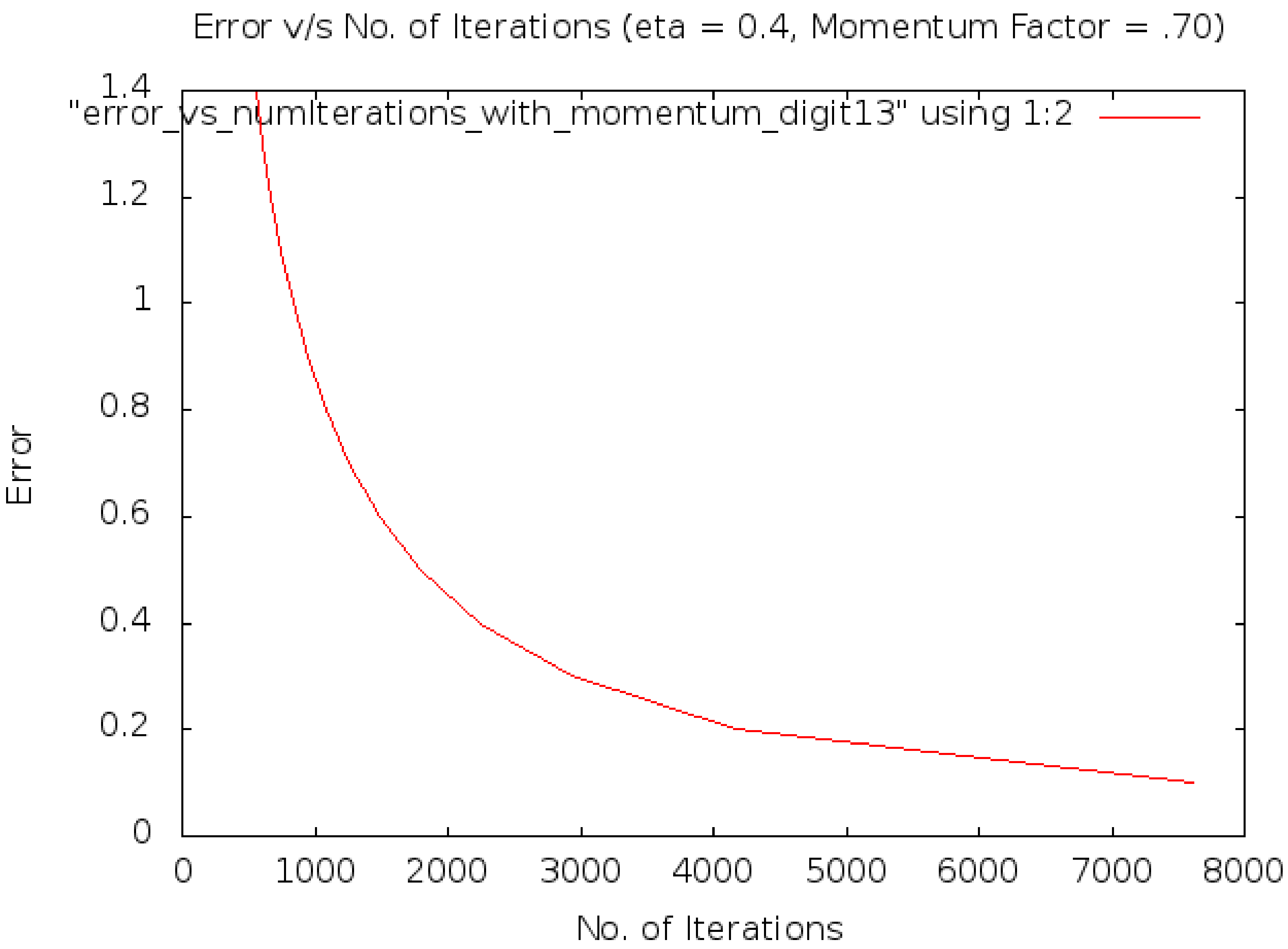
Effect of Momentum Factor (Digit Recogniser)



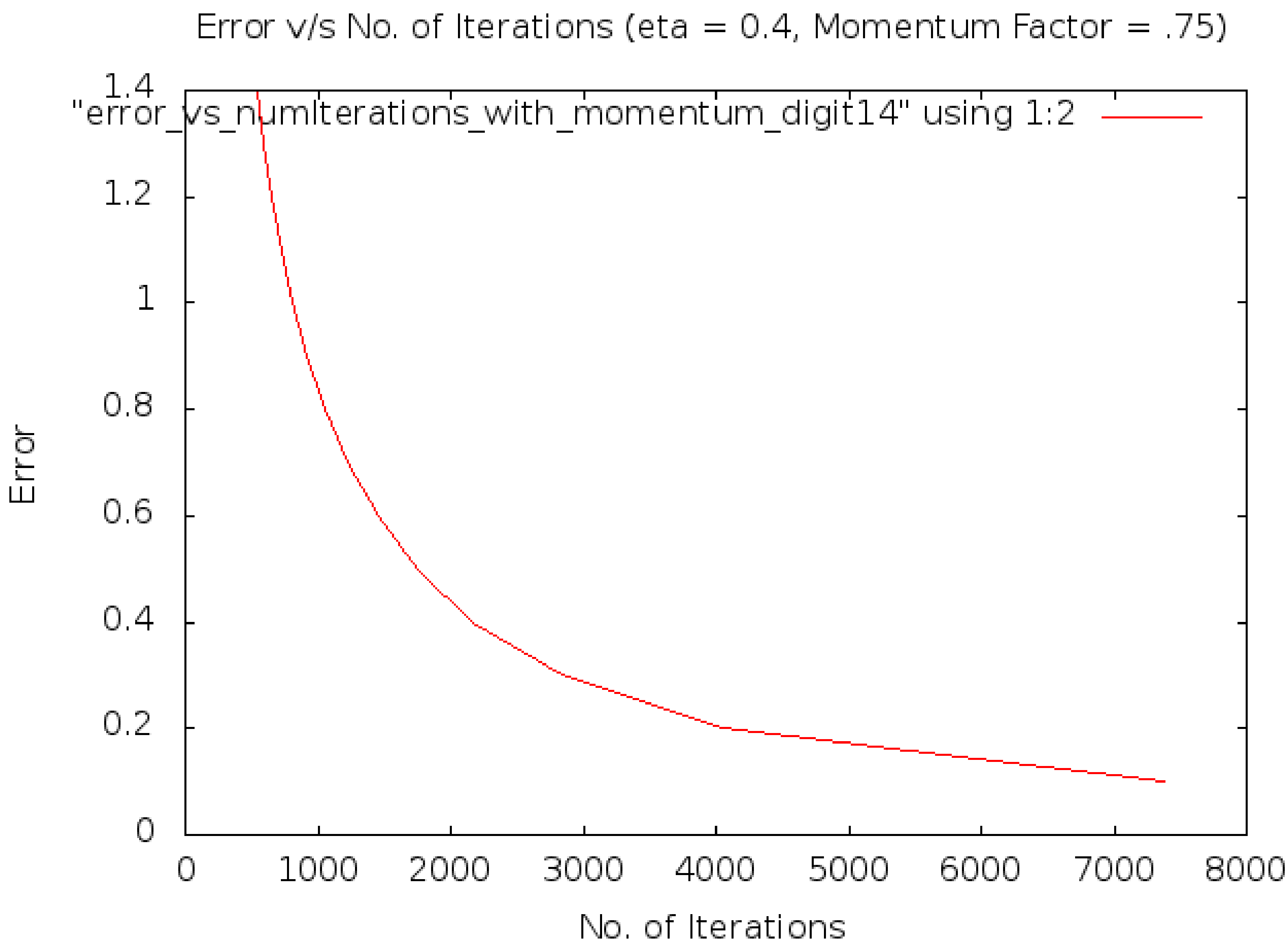
Effect of Momentum Factor (Digit Recogniser)



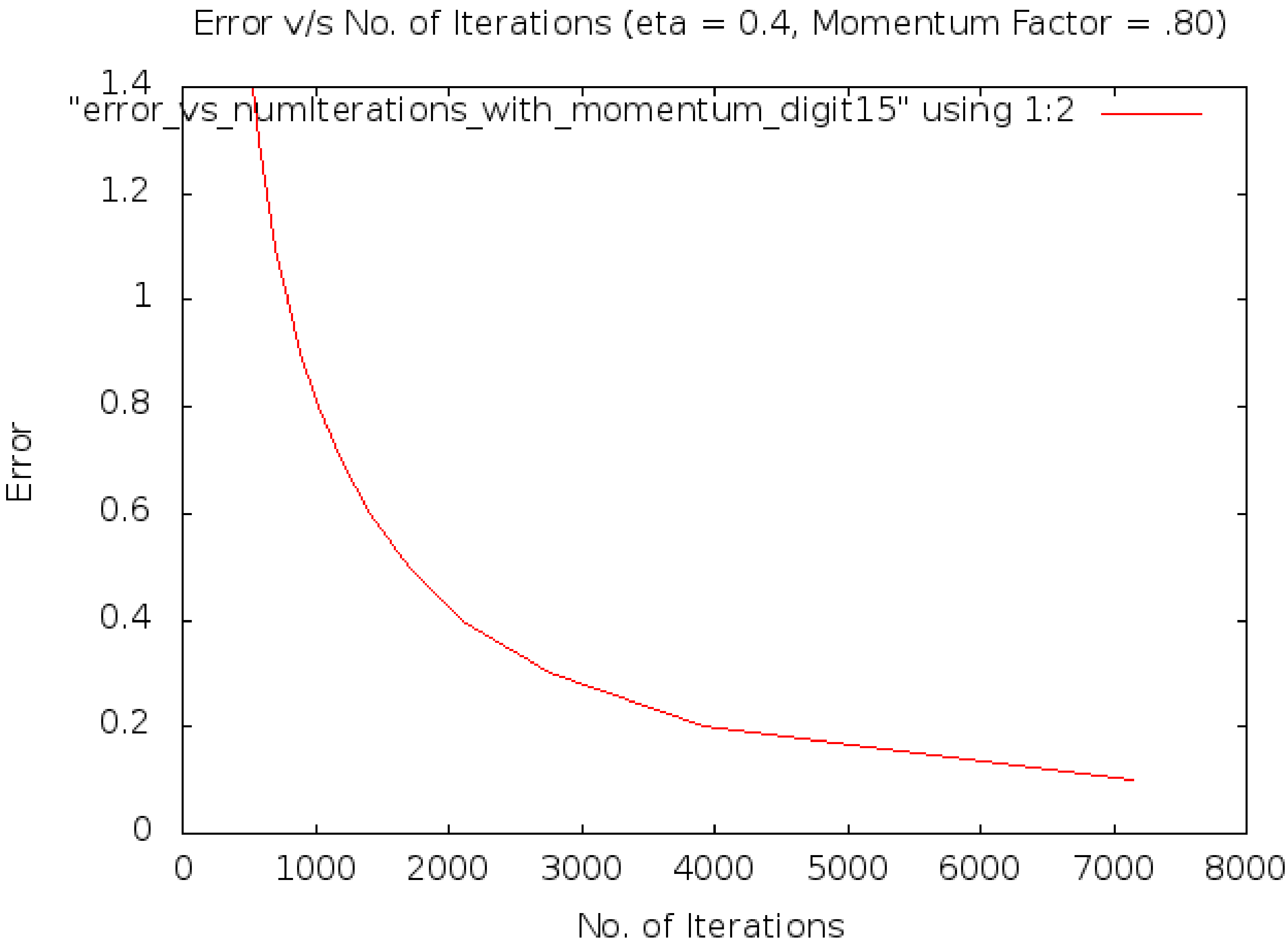
Effect of Momentum Factor (Digit Recogniser)



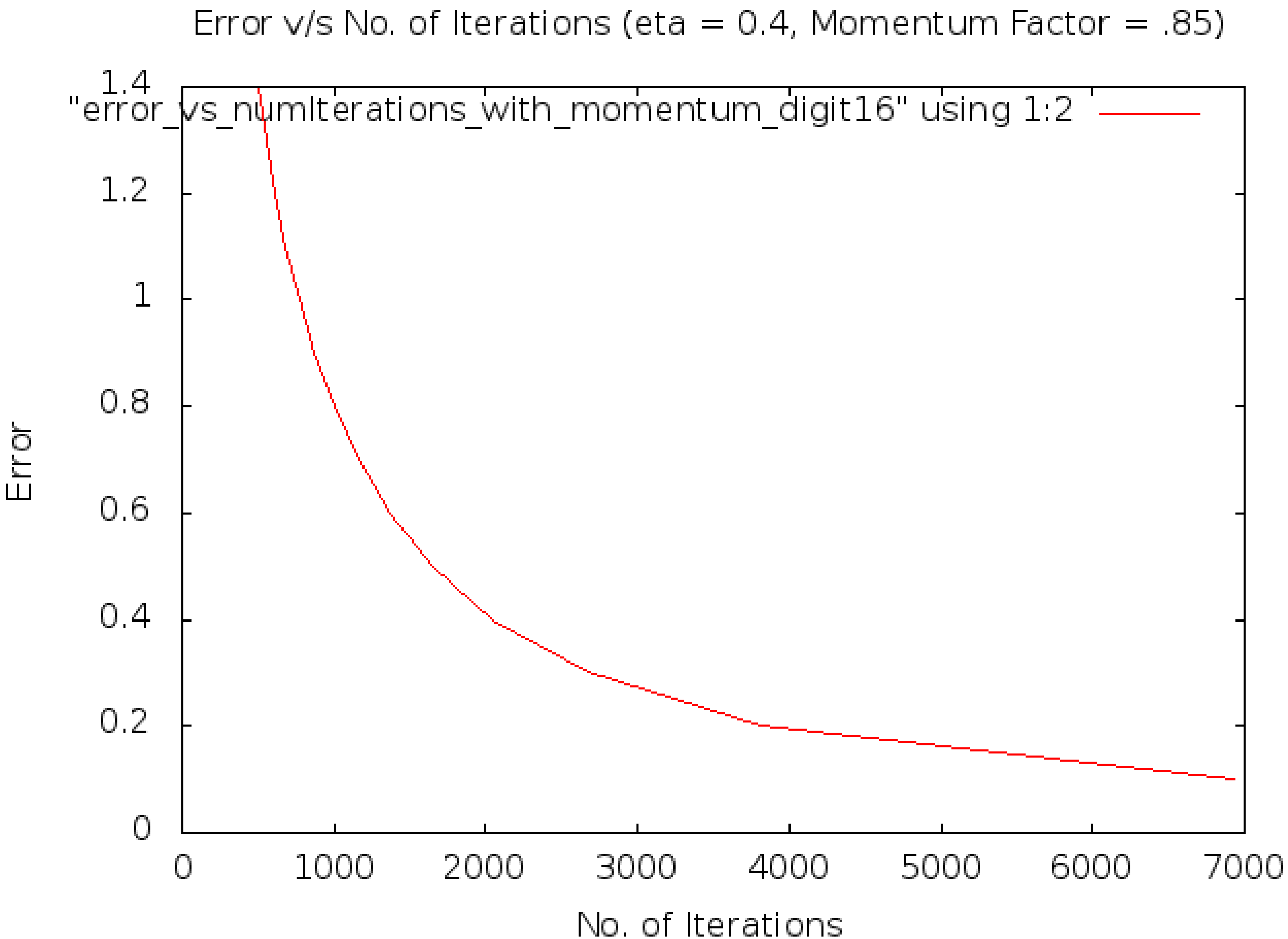
Effect of Momentum Factor (Digit Recogniser)



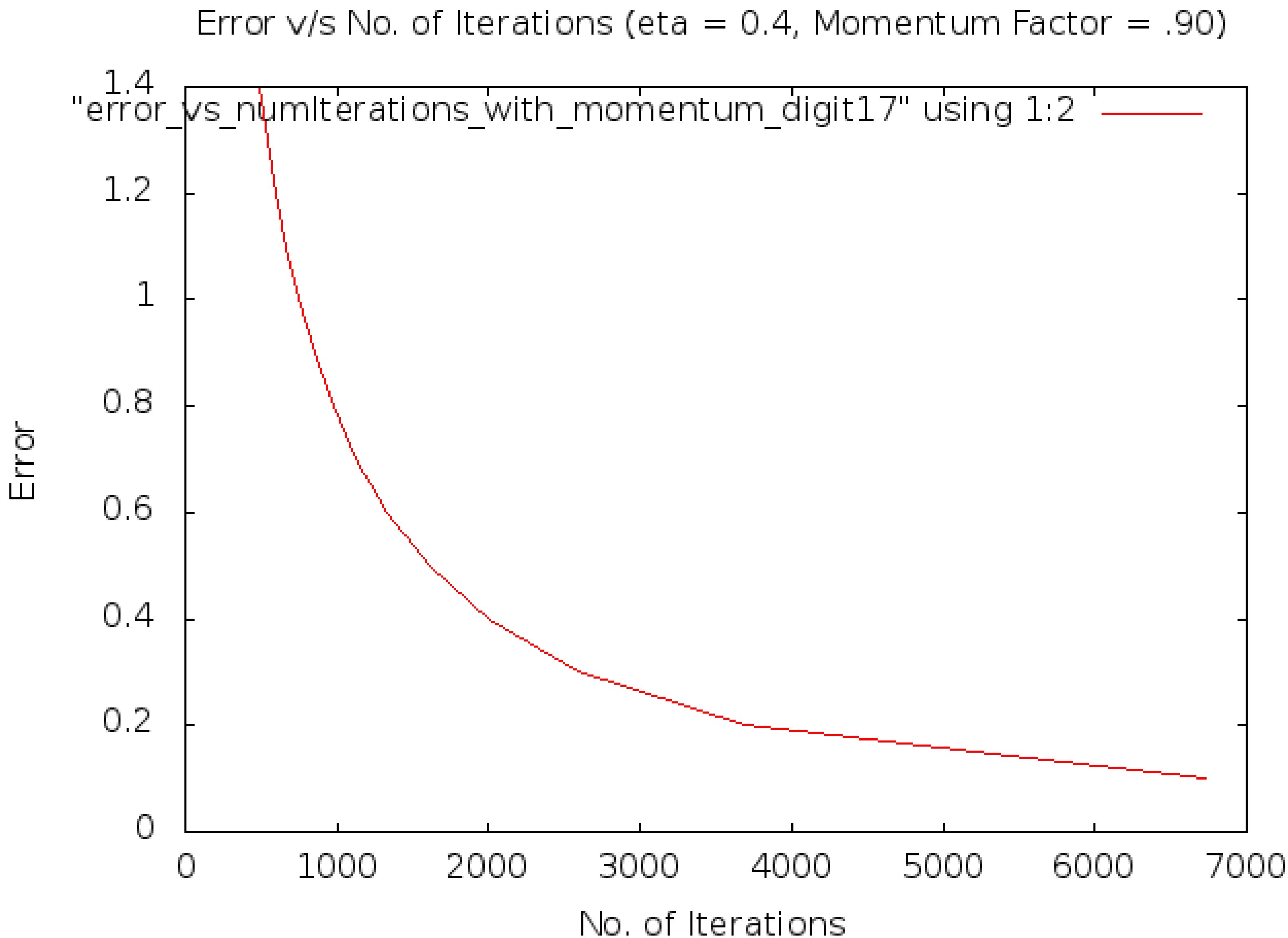
Effect of Momentum Factor (Digit Recogniser)



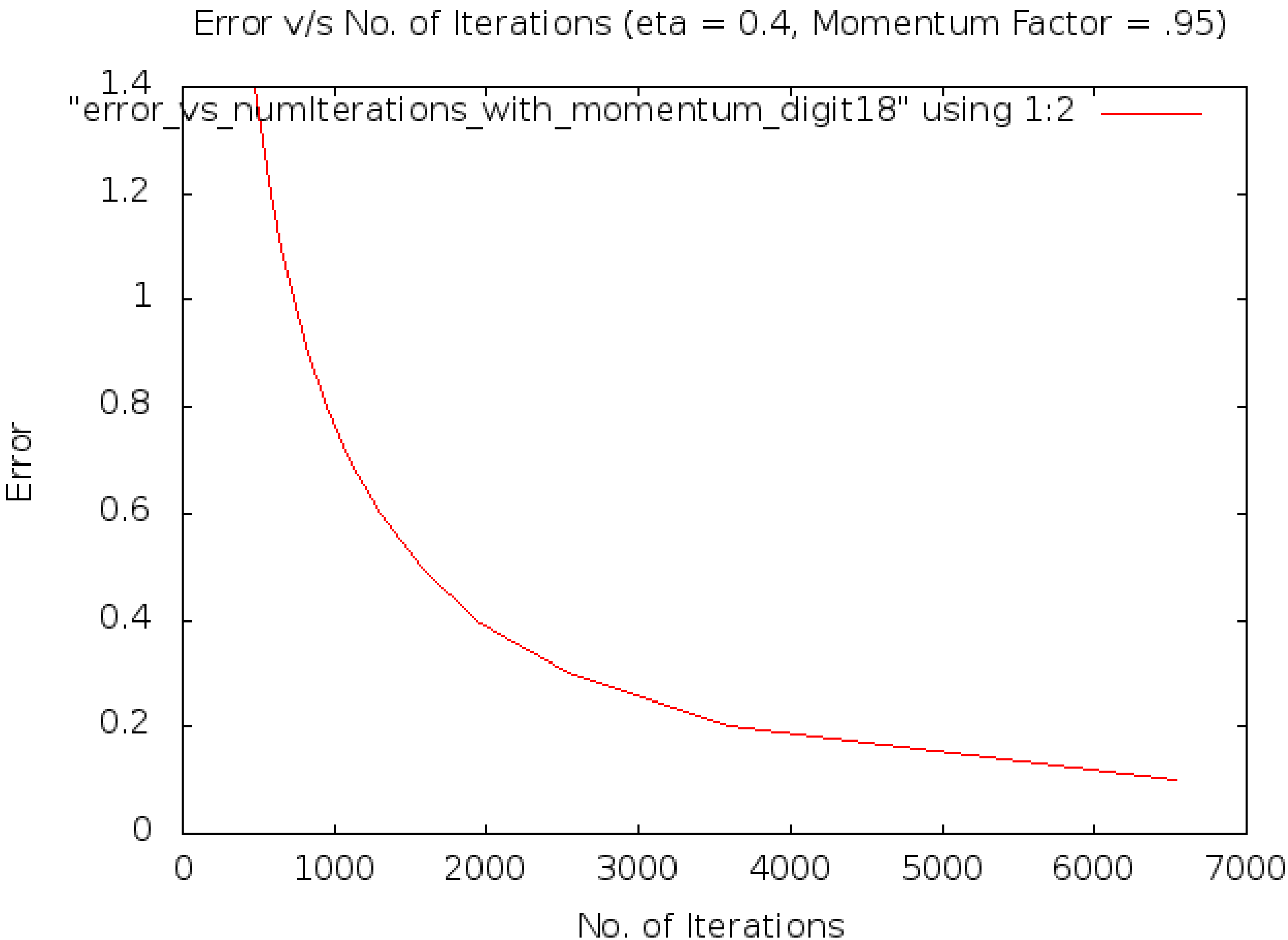
Effect of Momentum Factor (Digit Recogniser)



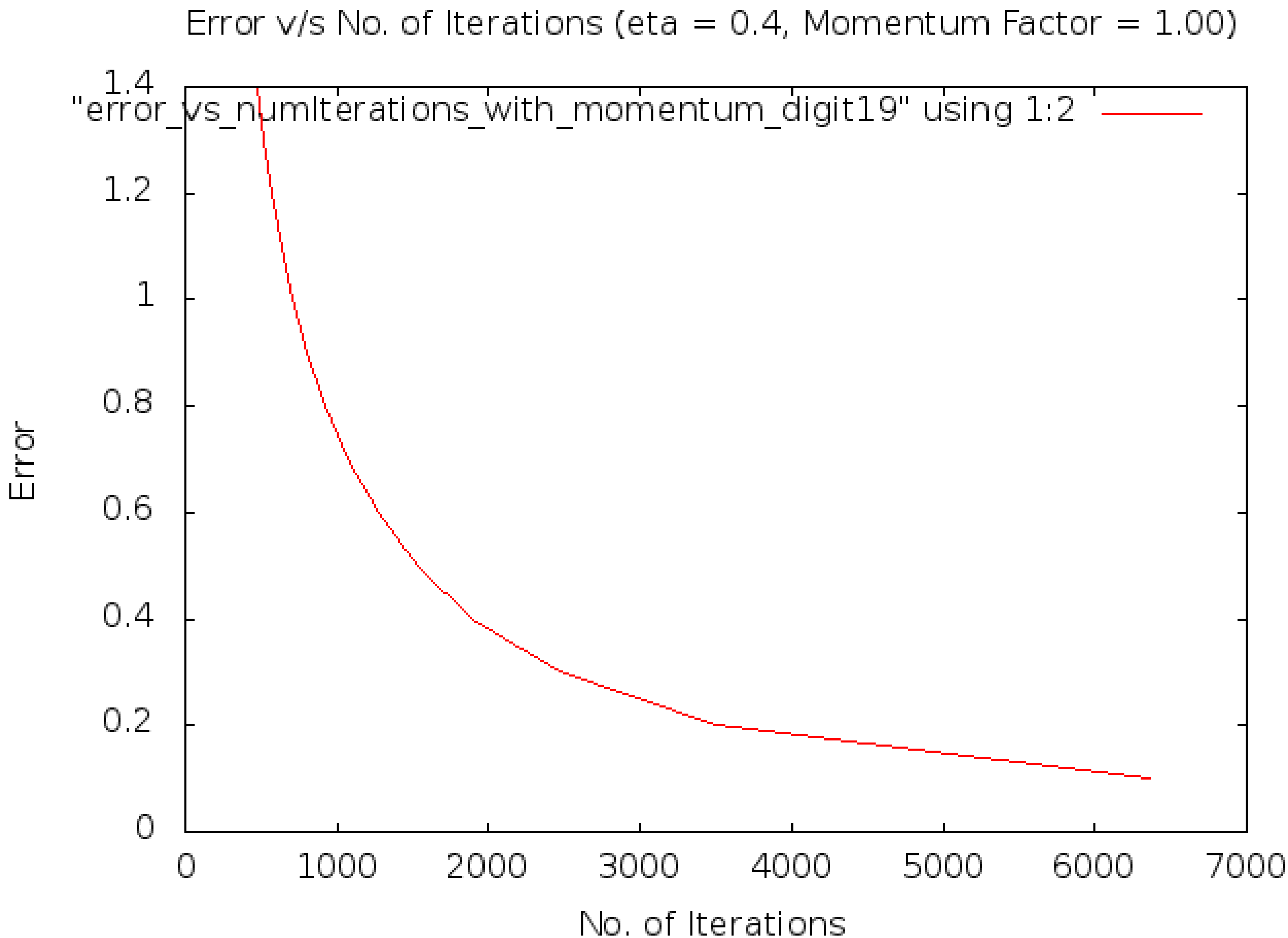
Effect of Momentum Factor (Digit Recogniser)



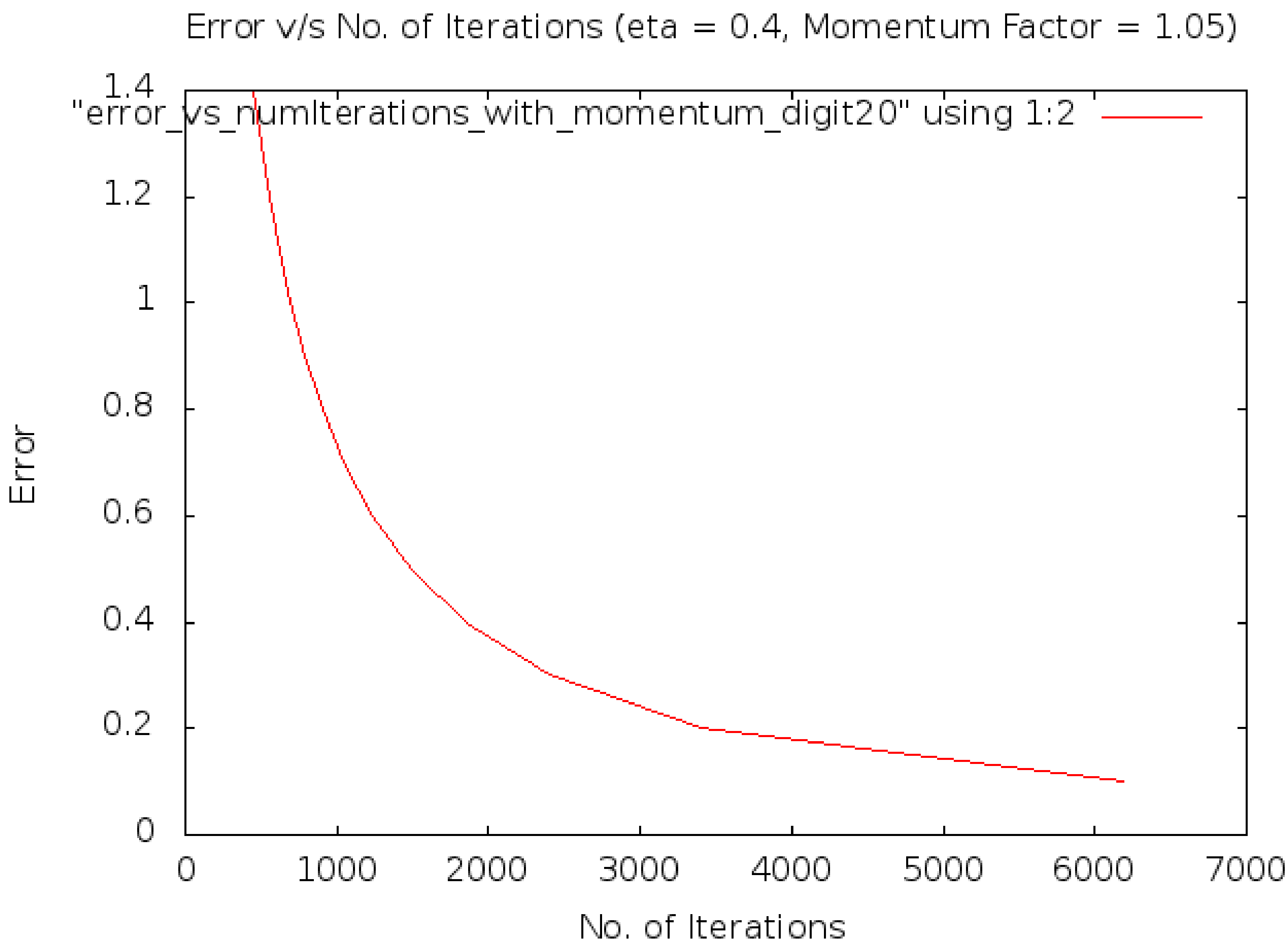
Effect of Momentum Factor (Digit Recogniser)



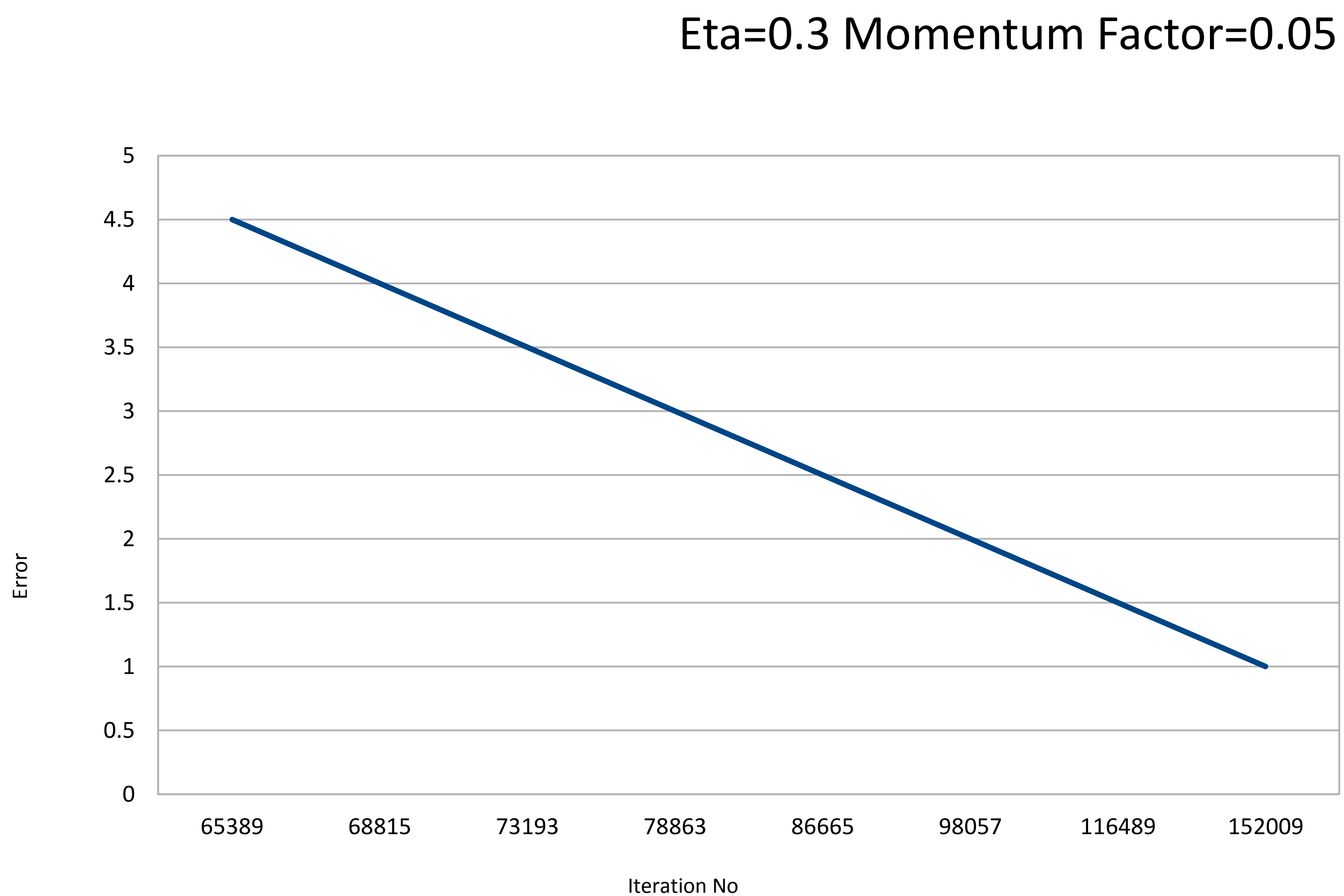
Effect of Momentum Factor (Digit Recogniser)



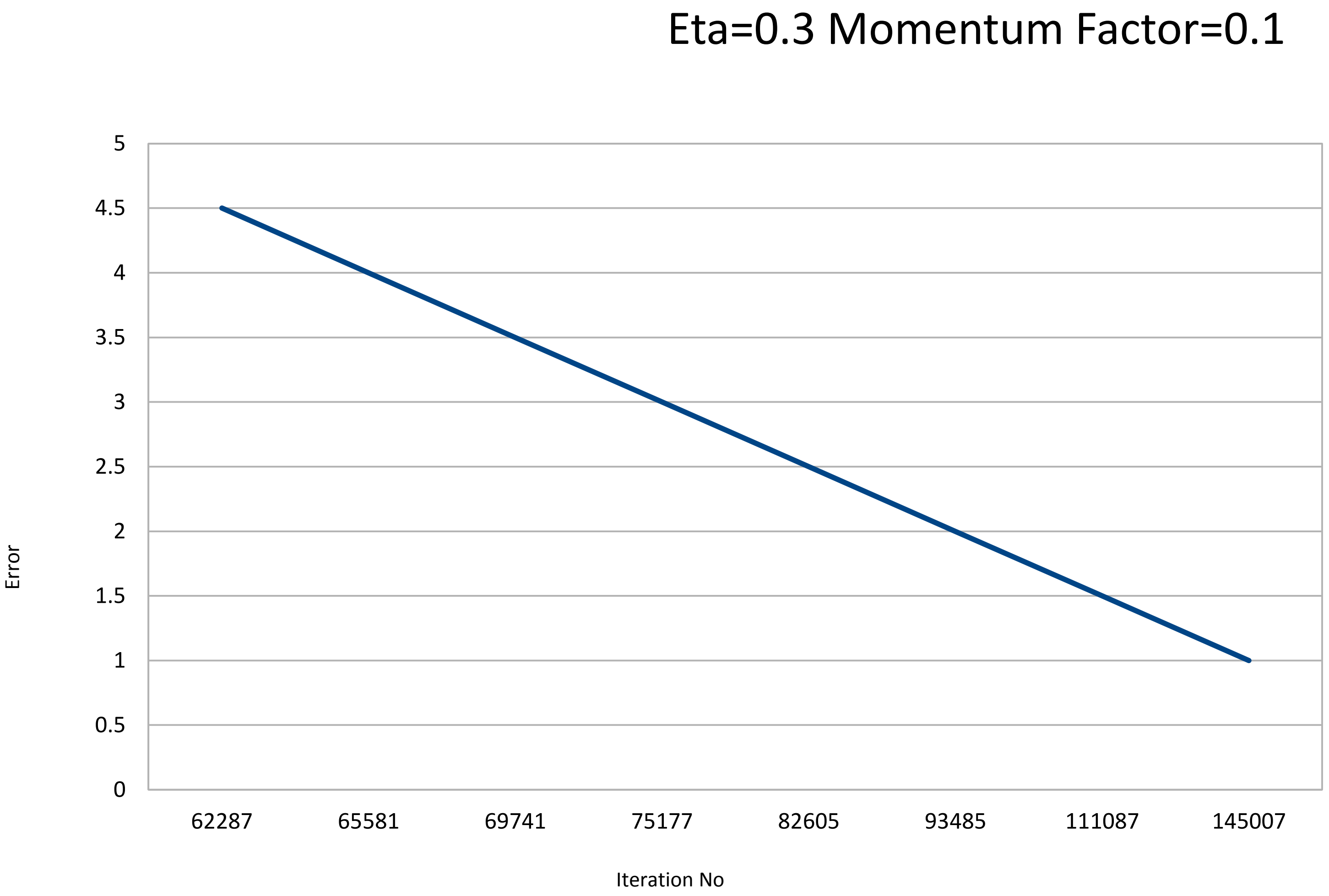
Effect of Momentum Factor (Digit Recogniser)



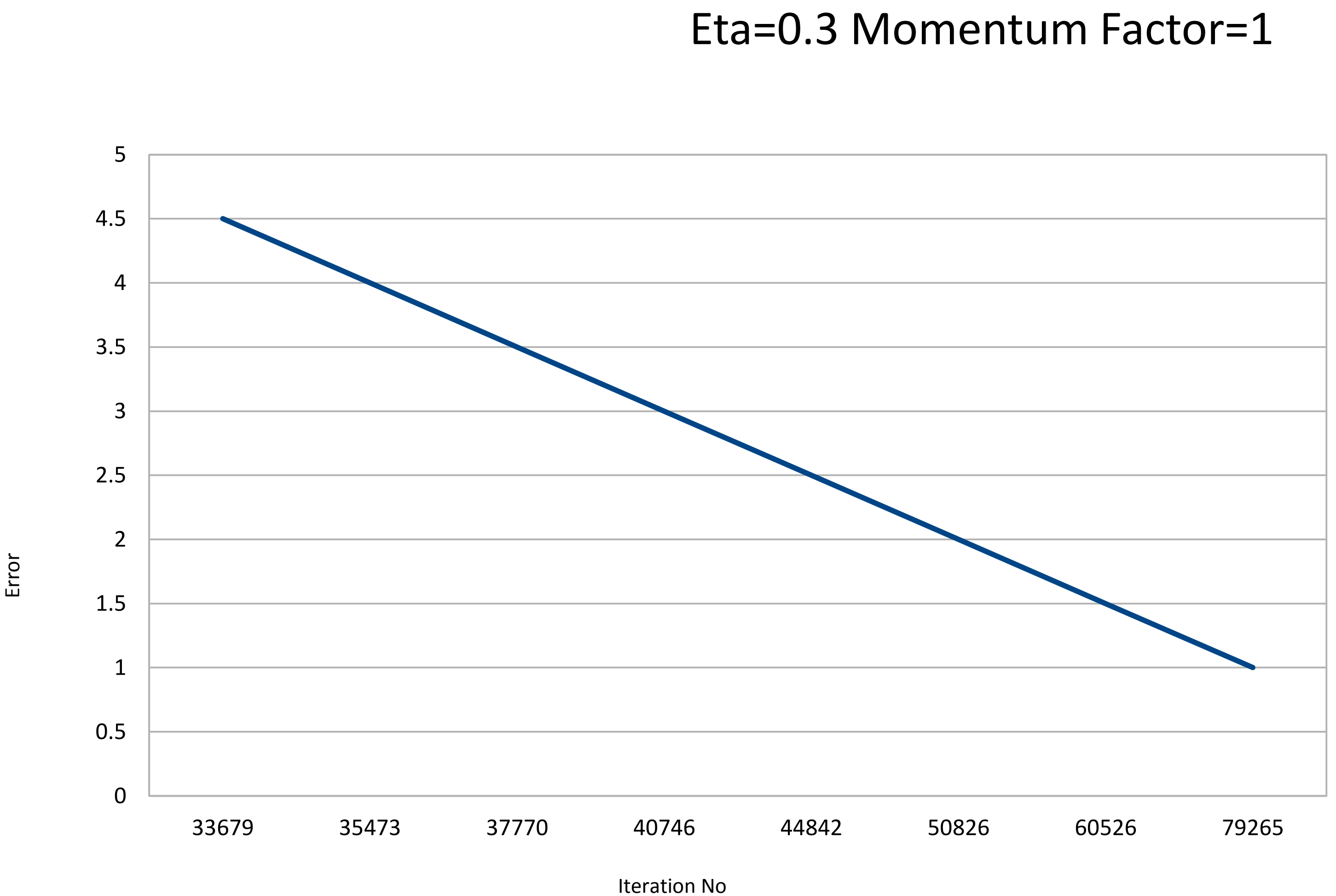
Effect of Momentum Factor (Parity)



Effect of Momentum Factor (Parity)

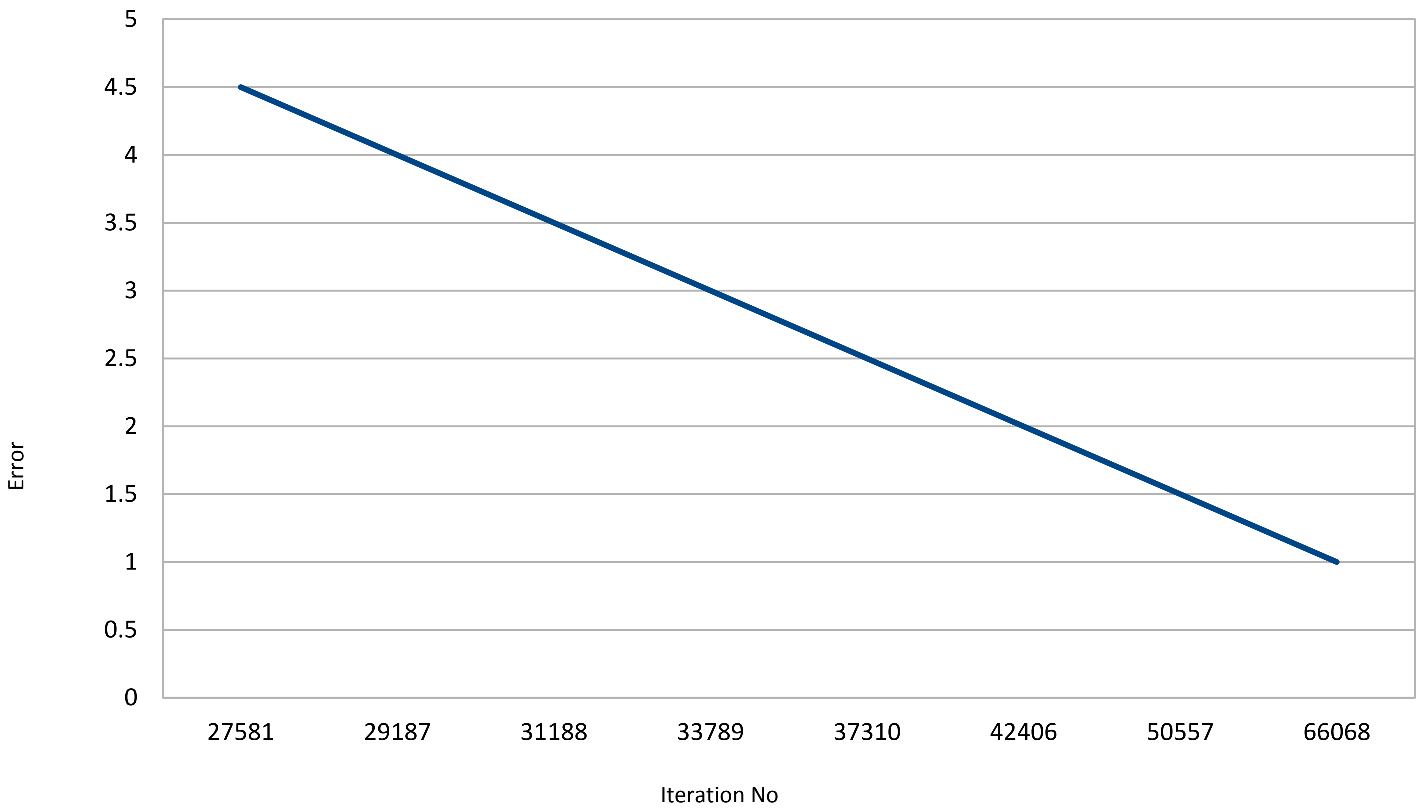


Effect of Momentum Factor (Parity)



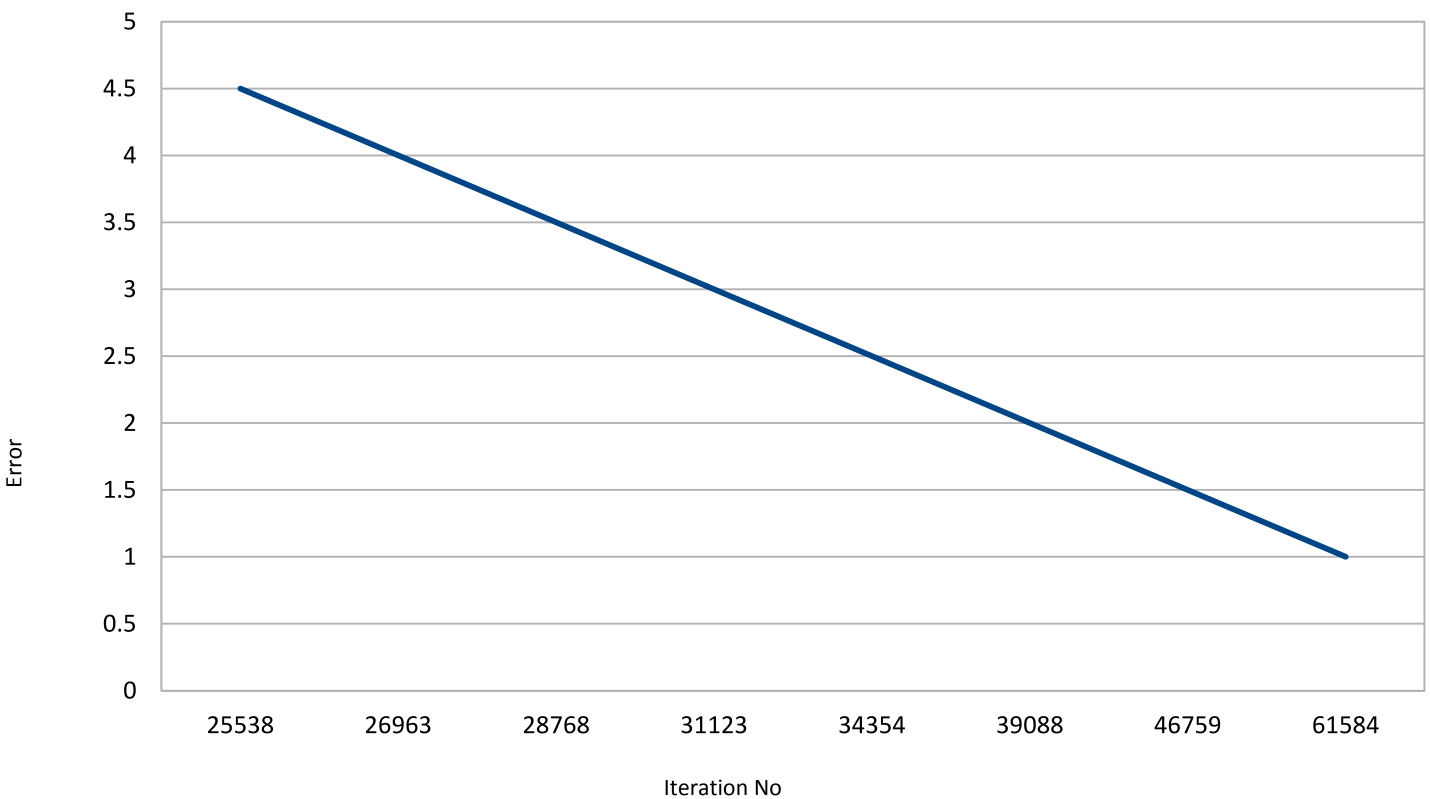
Effect of Momentum Factor(Palindrome)

Eta = 0.3 Momentum Factor = 0.05

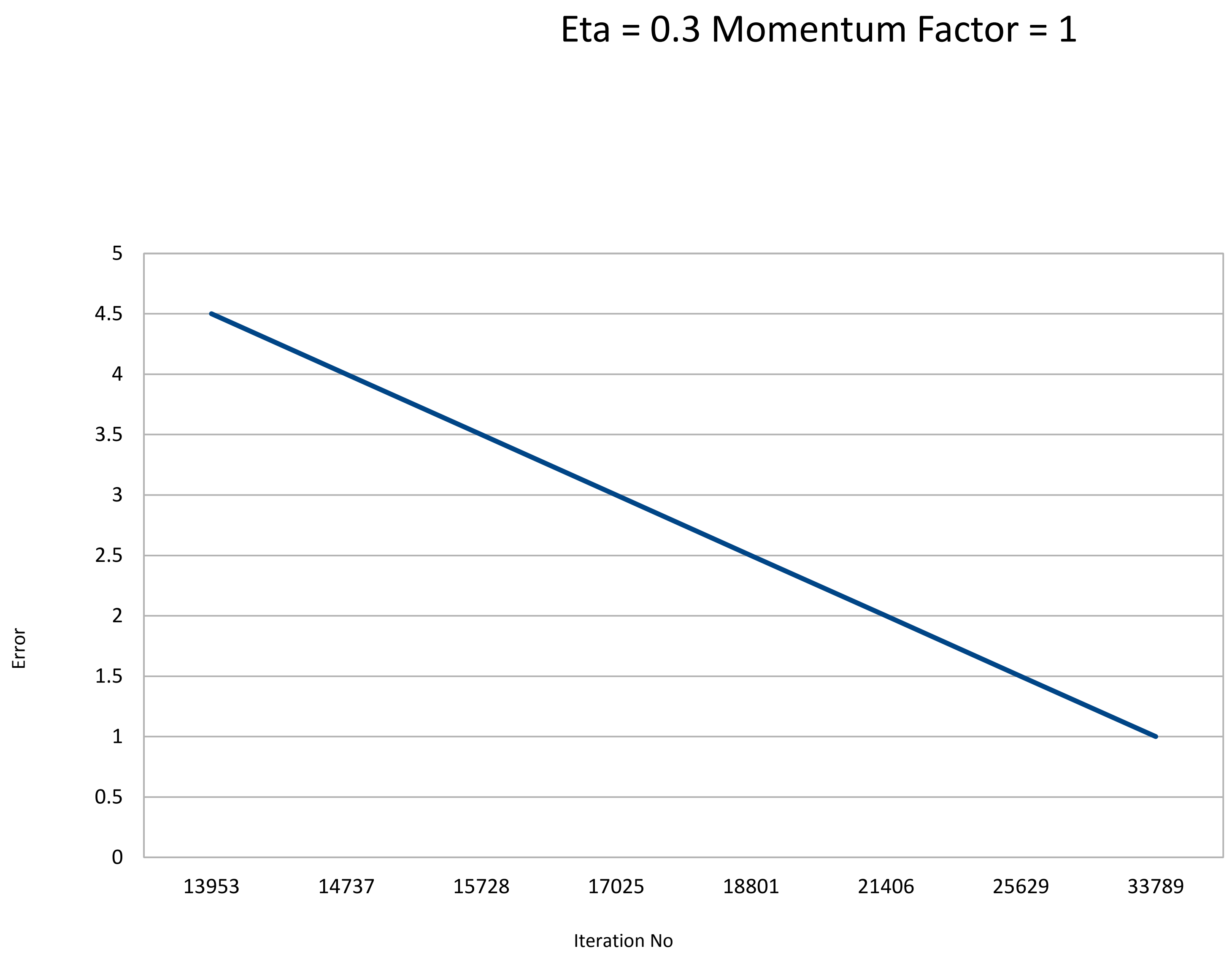


Effect of Momentum Factor(Palindrome)

Eta = 0.3 Momentum Factor = 0.1



Effect of Momentum Factor(Palindrome)



Effect of Momentum Factor

- We observed the effect of increasing the momentum too from close to 0 (0. 001) to close to 1(1.05) in steps of 0.05
- In general, as the momentum factor increased the no of iterations required to converge to a solution also decreased.
- We also observed that as the momentum factor increases, iteration-to-iteration variation in error was greatly increased.

Functionality of Hidden Layer Neurons

2-Input XOR

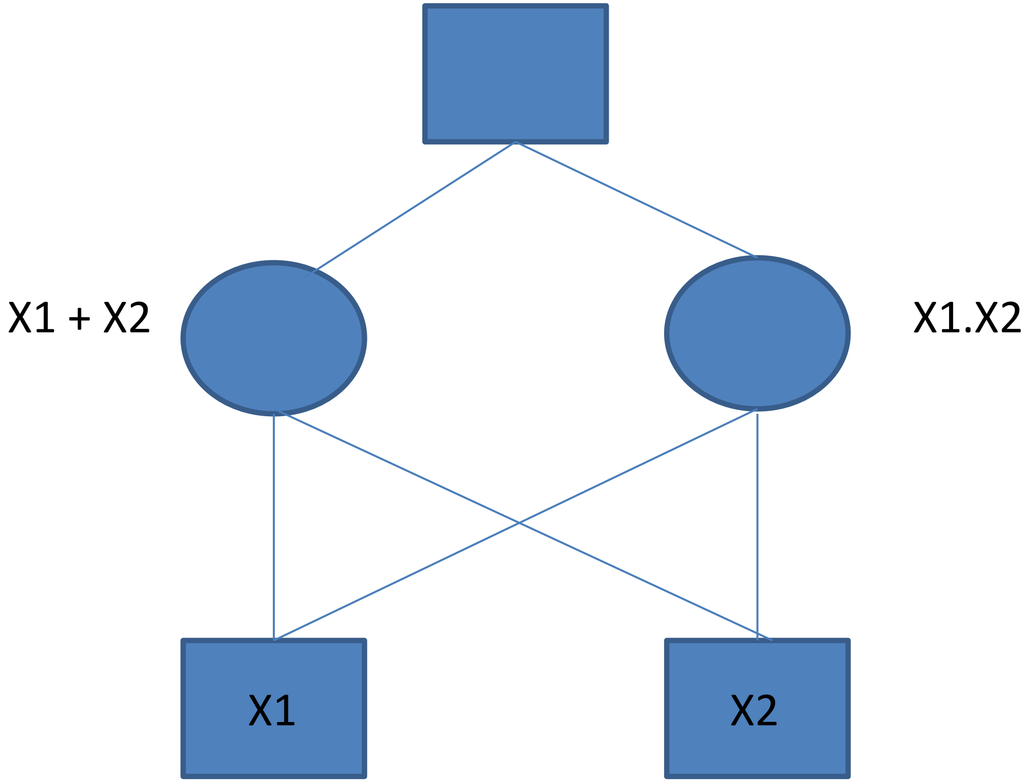
Input1	Input2	HiddenLayerOp1	HiddenLayerOp2	Output
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

- HiddenLayerOp1 = Input1 OR Input2
- HiddenLayerOp2 = Input1 AND Input2
- Output = HiddenLayerOp1 AND (NOT (HiddenLayerOp2))

Functionality of Hidden Layer Neurons

2-Input XOR

$$(X1 + X2) \, . \, \overline{(X1.X2)} \quad = X1.\overline{X2} + X2.\overline{X1}$$



Functionality of Hidden Layer Neurons

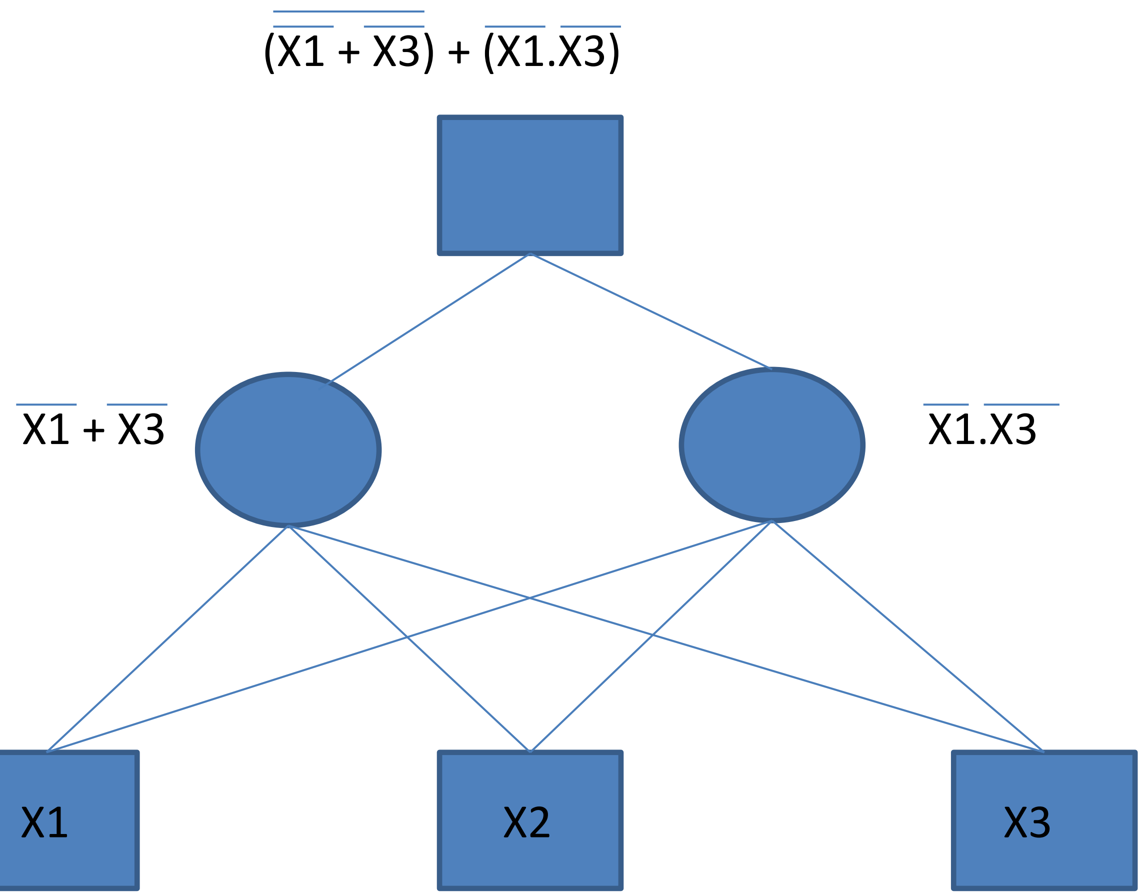
3-Input Palindrome

Input1	Input2	Input3	HiddenLayerOp1	HiddenLayerOp2	Output
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	1	0	0
1	1	1	0	0	1

- HiddenLayerOp1: NOT(Input1) OR NOT(Input3)
- HiddenLayerOp2 : NOT(Input1) AND NOT(Input3)
- Output = NOT(HiddenLayerOp1) OR HiddenLayerOp1

Functionality of Hidden Layer Neurons

3-Input Palindrome



For Palindrome, the X2 should not matter. Interestingly, X2 is not even taken into consideration in hidden layer.

Twitter Sentiment Analysis

Results & Observations

Assignment Specifications

- Give a neural network for recognizing the sentiments of tweets.
- Download tweets, do feature engineering on them.
- A naïve feature vector is the set of words in the tweets.
- Collect all the words in the tweets, sort them, remove duplicates.
- Each tweet will be represented by a 1/0 vector depending on the presence/absence of the word in the tweet.

Feature Engineering On Tweets

- We need to construct a feature vector large enough so that it can capture the essential sentiment of a tweet.
- A naïve feature vector is the set of all words in tweets. This has a large size.
- If our feature vector is too large, learning take long time and network learns unessential features which are not important for a sentiment.
- So, we consider only those words of the tweets that may constitute a sentiment and remove everything else.

Decreasing the Feature Vector Size

- Initially, total number of words in all tweets = 24680
- Convert all the words to lower case.
- Remove all Numbers.
- Remove special characters at start and at end of words such as

  ? , . [ ] ; _ ! = | - & ‘ # URLs
- Don't put certain common words, symbols, numbers in the vector of words such as:

  the, is, a, an, this, that, are my with, I’m, he, she, our, were, can, do, had….
- Finally we remove those words that occur only once in the entire set of words in the tweets because they are highly unlikely to contribute to a sentiment
- This results in a lot of unnecessary words being removed from our feature vector and finally we get a feature vector whose size is 1958

Methodology

- We used a neural network with 3 output neurons and 1958 input neurons. (No hidden layer)
- We performed 5-fold cross validation on the given tweet corpus.
- We divided the tweets into 5 partitions & used 4 partitions for training the neural network and the remaining one for measuring accuracy
- Coded in C++, it took approx. 6 min to converge to a solution for training and testing on a single partition.
- Learning rate was kept at 0.5, Momentum Factor = 0.

Results Obtained (5-fold)

- Set 1 : 56.25%
- Set 2 : 47.25%
- Set 3 : 48%
- Set 4 : 53.25%
- Set 5 : 55.8603%

- Here, Set 'n' corresponds to n^{th} partition for testing and remaining for training.

Testing & Training of FFNN on IRIS Data Set

Results & Observations

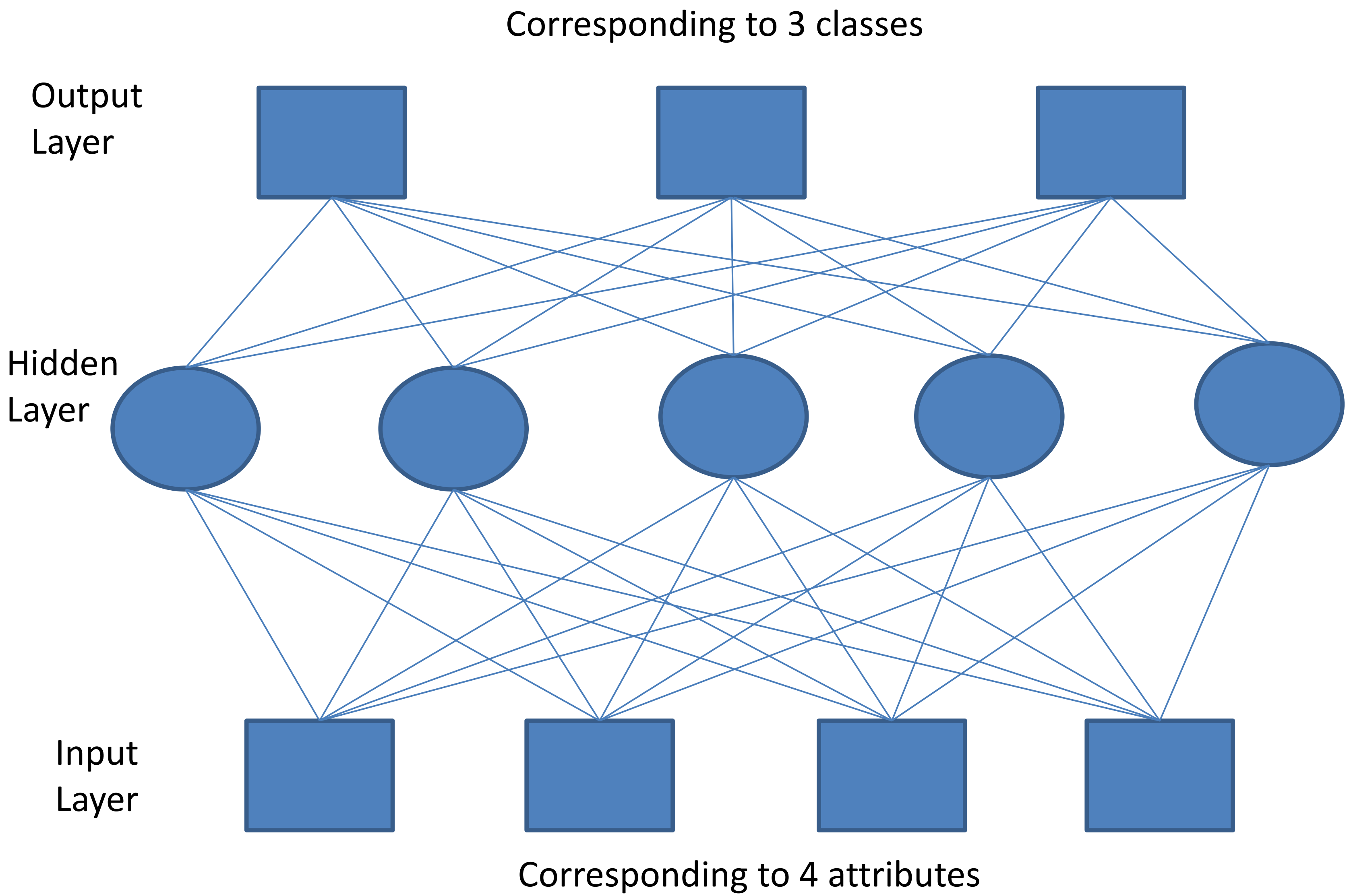
Assignment Specifications

- Download any classification benchmark data from ML repositories (University of California at Irwine).
- Train and test FFNN on such data. Of particular note is a classic problem called IRIS data.

Data Set Specifications

- Number of Instances: 150
- Classes:
    - Iris Setosa
    - Iris Versicolour
    - Iris Virginica
- Number of Attributes: 4 numeric, predictive attributes
- Class Distribution: 33.3% for each of 3 classes

Neural Network Structure



Testing & Training Methodology

- We performed N-fold cross validation with N=5.
- We divided our classification data into 5 partitions & used 4 partitions for training the neural network and the remaining one for measuring accuracy.
- We also observed the effect of changing the order of data set elements i.e. we shuffled the elements of training data set so that instances of data belonging to a single class do not occur together.

Result-Using 5 Fold Validation Data Partition-1

- Initialization - random between -0.5 to 0.5
- Accuracy – 1 (30/30)
- Snapshot of Result →

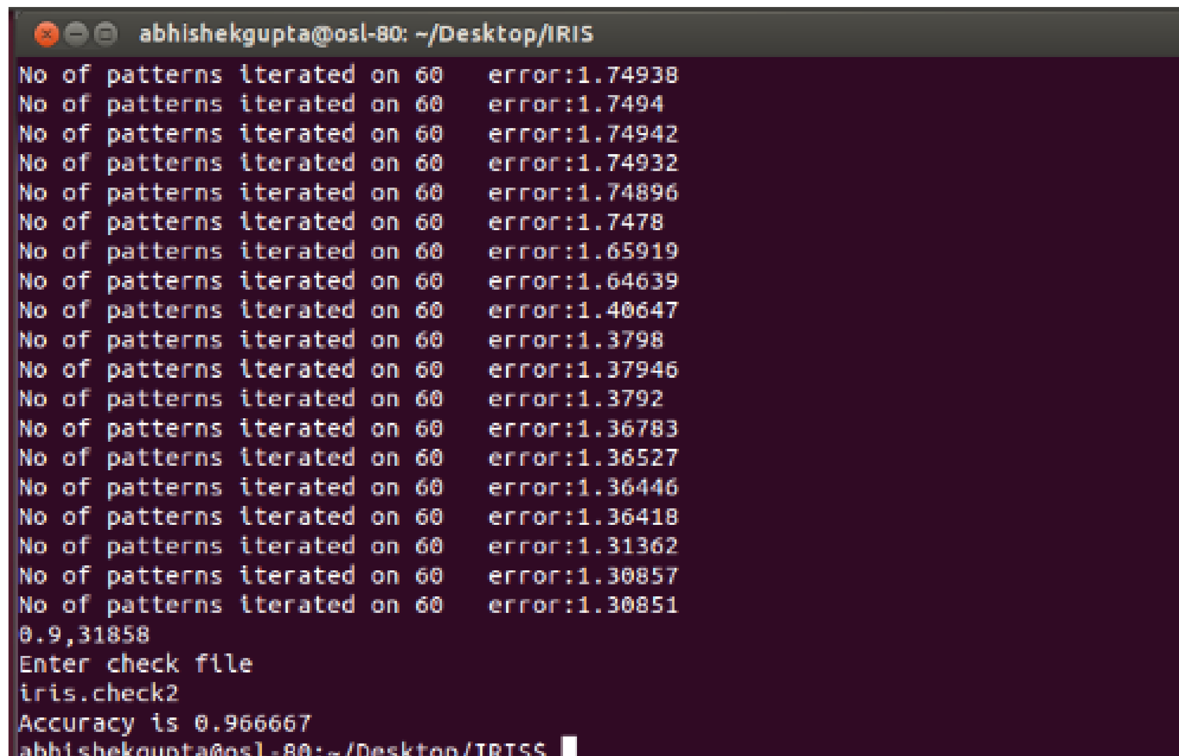
Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%

```
abhishekgupta@osl-80: ~/Desktop/IRIS
No of patterns iterated on 113  error:1.64576
No of patterns iterated on 113  error:1.64557
No of patterns iterated on 113  error:1.62536
No of patterns iterated on 113  error:1.59694
No of patterns iterated on 113  error:1.57587
No of patterns iterated on 113  error:1.57408
No of patterns iterated on 113  error:1.57382
No of patterns iterated on 113  error:1.57323
No of patterns iterated on 113  error:1.56924
No of patterns iterated on 113  error:1.56884
No of patterns iterated on 113  error:1.55462
No of patterns iterated on 113  error:1.54476
No of patterns iterated on 113  error:1.53734
No of patterns iterated on 113  error:1.51177
No of patterns iterated on 117  error:1.50023
No of patterns iterated on 118  error:1.52273
No of patterns iterated on 118  error:1.51377
No of patterns iterated on 118  error:1.513
No of patterns iterated on 118  error:1.50026
1.5,545994
Enter check file
iris.check1
Accuracy is 1
abhishekgupta@osl-80:~/Desktop/IRIS$
```

Result-Using 5 Fold Validation Data Partition-2

- Initialization - random between -0.5 to 0.5
- Accuracy – 0.96667 (29/30)
- Snapshot of Result

Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%

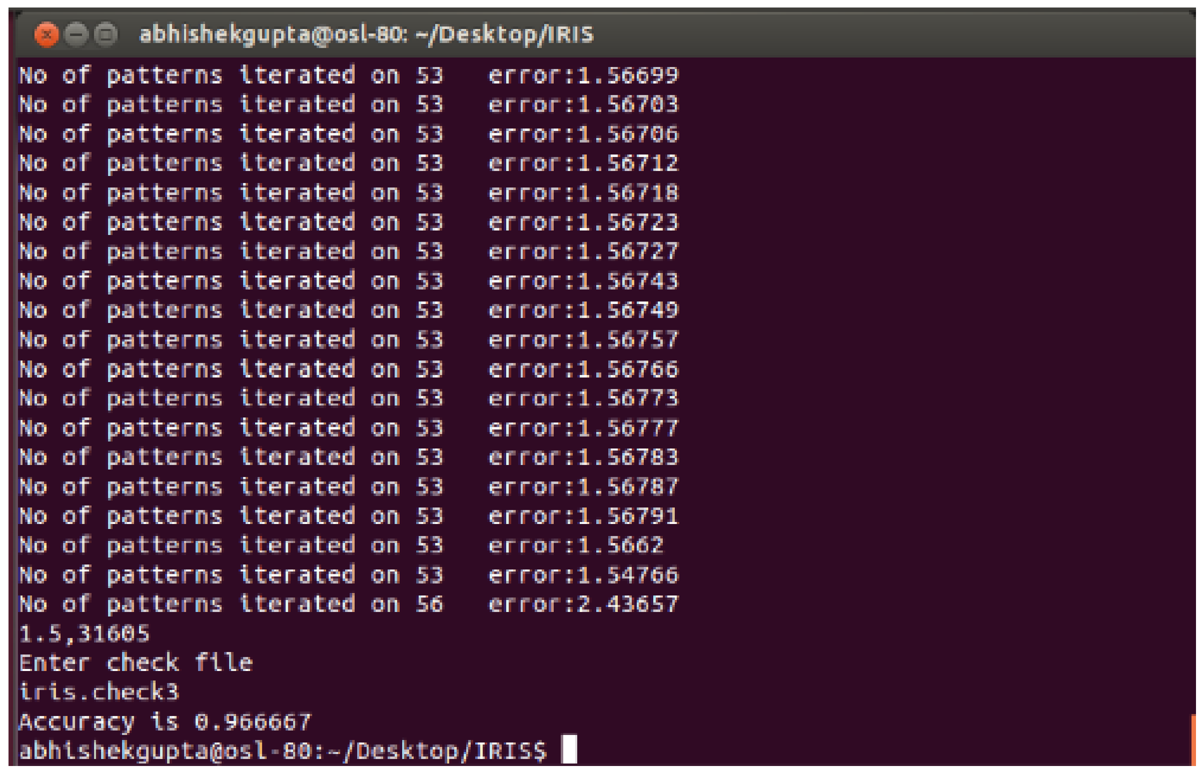


```
abhishekgupta@osl-80: ~/Desktop/IRIS
No of patterns iterated on 60   error:1.74938
No of patterns iterated on 60   error:1.7494
No of patterns iterated on 60   error:1.74942
No of patterns iterated on 60   error:1.74932
No of patterns iterated on 60   error:1.74896
No of patterns iterated on 60   error:1.7478
No of patterns iterated on 60   error:1.65919
No of patterns iterated on 60   error:1.64639
No of patterns iterated on 60   error:1.40647
No of patterns iterated on 60   error:1.3798
No of patterns iterated on 60   error:1.37946
No of patterns iterated on 60   error:1.3792
No of patterns iterated on 60   error:1.36783
No of patterns iterated on 60   error:1.36527
No of patterns iterated on 60   error:1.36446
No of patterns iterated on 60   error:1.36418
No of patterns iterated on 60   error:1.31362
No of patterns iterated on 60   error:1.30857
No of patterns iterated on 60   error:1.30851
0.9,31858
Enter check file
iris.check2
Accuracy is 0.966667
abhishekgupta@osl-80:~/Desktop/IRIS$
```

Result-Using 5 Fold Validation Data Partition-3

- Initialization - random between -0.5 to 0.5
- Accuracy – 0.96667 (29/30)
- Snapshot of Result →

Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%

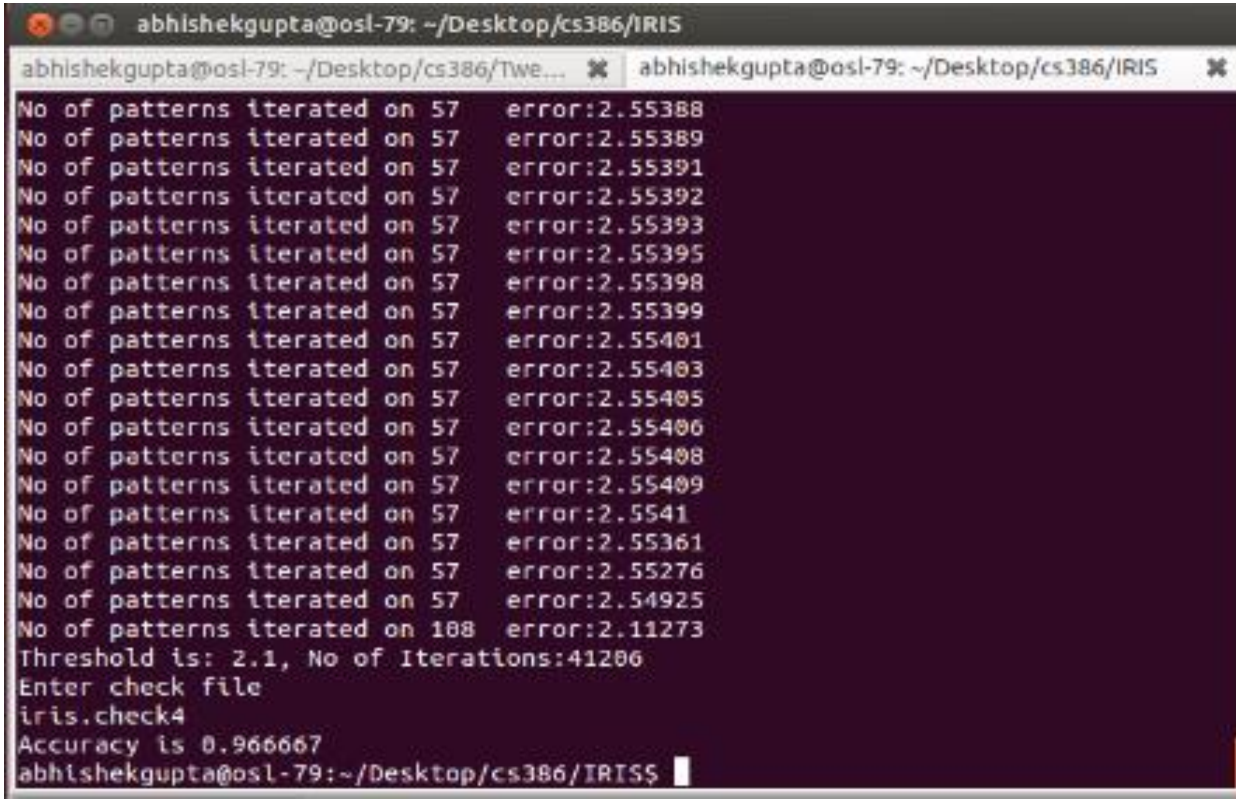


```
abhishekgupta@osl-80: ~/Desktop/IRIS
No of patterns iterated on 53   error:1.56699
No of patterns iterated on 53   error:1.56703
No of patterns iterated on 53   error:1.56706
No of patterns iterated on 53   error:1.56712
No of patterns iterated on 53   error:1.56718
No of patterns iterated on 53   error:1.56723
No of patterns iterated on 53   error:1.56727
No of patterns iterated on 53   error:1.56743
No of patterns iterated on 53   error:1.56749
No of patterns iterated on 53   error:1.56757
No of patterns iterated on 53   error:1.56766
No of patterns iterated on 53   error:1.56773
No of patterns iterated on 53   error:1.56777
No of patterns iterated on 53   error:1.56783
No of patterns iterated on 53   error:1.56787
No of patterns iterated on 53   error:1.56791
No of patterns iterated on 53   error:1.5662
No of patterns iterated on 53   error:1.54766
No of patterns iterated on 56   error:2.43657
1.5,31605
Enter check file
iris.check3
Accuracy is 0.966667
abhishekgupta@osl-80:~/Desktop/IRIS$
```

Result-Using 5 Fold Validation Data Partition-4

- Initialization - random between -0.5 to 0.5
- Accuracy – 0.96667 (29/30)
- Snapshot of Result →

Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%



Result-Using 5 Fold Validation Data Partition-5

- Initialization - random between -0.5 to 0.5
- Accuracy – 1 (30/30)
- Snapshot of Result →

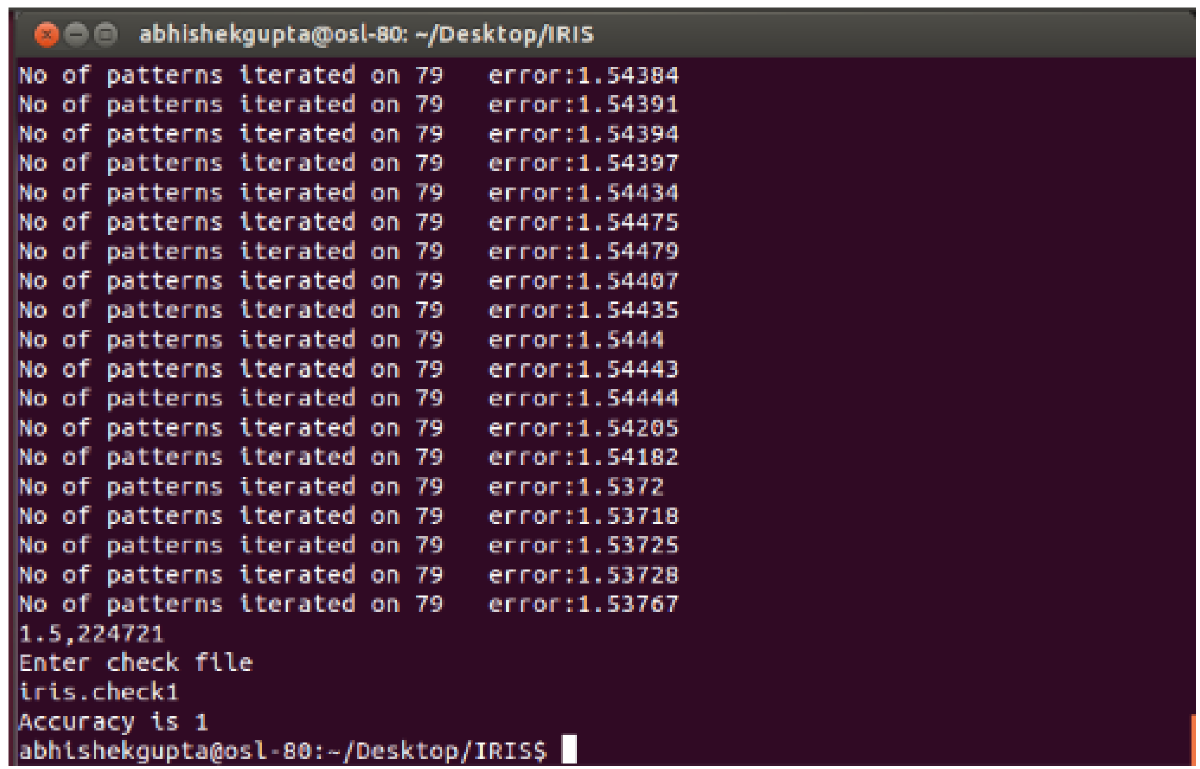
Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%

```
abhishekgupta@osl-80: ~/Desktop/IRIS
No of patterns iterated on 101  error:1.70963
No of patterns iterated on 101  error:1.70963
No of patterns iterated on 103  error:2.19497
No of patterns iterated on 103  error:2.1621
No of patterns iterated on 103  error:2.14959
No of patterns iterated on 103  error:2.14853
No of patterns iterated on 103  error:2.14853
No of patterns iterated on 103  error:2.14802
No of patterns iterated on 103  error:2.11097
No of patterns iterated on 103  error:2.10214
No of patterns iterated on 103  error:2.1017
No of patterns iterated on 103  error:2.0981
No of patterns iterated on 103  error:2.09779
No of patterns iterated on 103  error:2.09778
No of patterns iterated on 103  error:2.09778
No of patterns iterated on 103  error:2.07553
No of patterns iterated on 103  error:2.07267
No of patterns iterated on 103  error:2.01906
No of patterns iterated on 103  error:1.933
1.5,663819
Enter check file
iris.check5
Accuracy is 1
abhishekgupta@osl-80:~/Desktop/IRIS$
```

Shuffled Data Set Partition-1

- Initialization - random between -0.25 to 0.25
- Accuracy – 1 (30/30)
- Snapshot of Result →

Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%

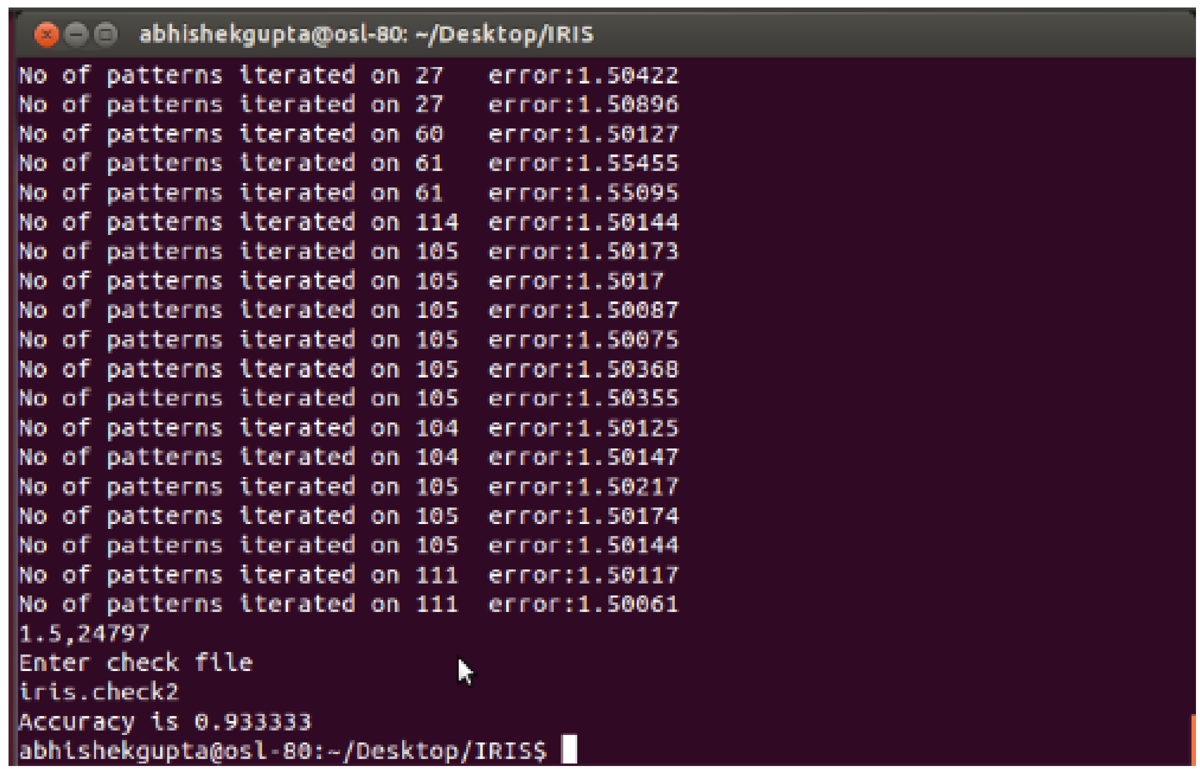


```
abhishekgupta@osl-80: ~/Desktop/IRIS
No of patterns iterated on 79   error:1.54384
No of patterns iterated on 79   error:1.54391
No of patterns iterated on 79   error:1.54394
No of patterns iterated on 79   error:1.54397
No of patterns iterated on 79   error:1.54434
No of patterns iterated on 79   error:1.54475
No of patterns iterated on 79   error:1.54479
No of patterns iterated on 79   error:1.54407
No of patterns iterated on 79   error:1.54435
No of patterns iterated on 79   error:1.5444
No of patterns iterated on 79   error:1.54443
No of patterns iterated on 79   error:1.54444
No of patterns iterated on 79   error:1.54205
No of patterns iterated on 79   error:1.54182
No of patterns iterated on 79   error:1.5372
No of patterns iterated on 79   error:1.53718
No of patterns iterated on 79   error:1.53725
No of patterns iterated on 79   error:1.53728
No of patterns iterated on 79   error:1.53767
1.5,224721
Enter check file
iris.check1
Accuracy is 1
abhishekgupta@osl-80:~/Desktop/IRIS$
```

Shuffled Data Set Partition-2

- Initialization - random between -0.25 to 0.25
- Accuracy – 0.93333 (28/30)
- Snapshot of Result

Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%



Shuffled Data Set Partition-3

- Initialization - random between -0.25 to 0.25
- Accuracy – 0.966667 (29/30)
- Snapshot of Result →

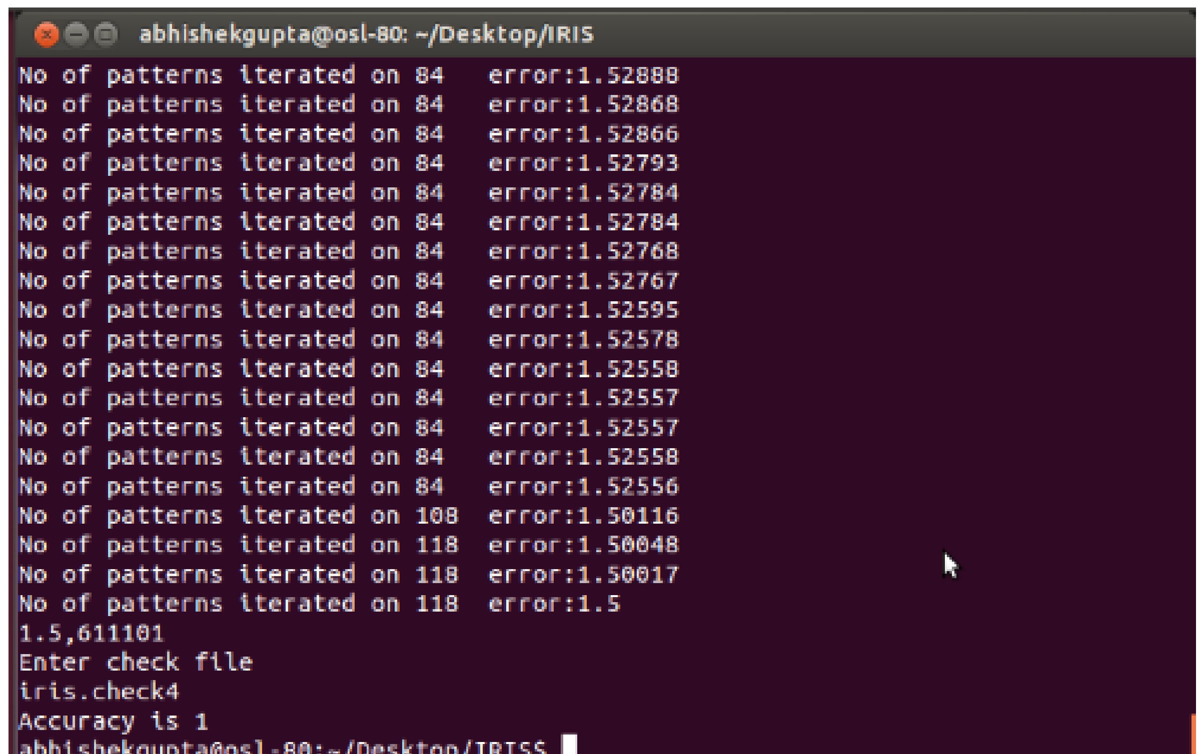
Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%

```
abhishekgupta@osl-80: ~/Desktop/IRIS
No of patterns iterated on 71   error:1.99863
No of patterns iterated on 71   error:1.9987
No of patterns iterated on 71   error:1.99869
No of patterns iterated on 71   error:1.9987
No of patterns iterated on 71   error:1.99871
No of patterns iterated on 71   error:1.99891
No of patterns iterated on 71   error:1.99891
No of patterns iterated on 71   error:1.99802
No of patterns iterated on 71   error:1.9979
No of patterns iterated on 71   error:1.99791
No of patterns iterated on 71   error:1.99428
No of patterns iterated on 71   error:1.99393
No of patterns iterated on 71   error:1.99269
No of patterns iterated on 71   error:1.98739
No of patterns iterated on 34   error:1.7247
No of patterns iterated on 34   error:2.01174
No of patterns iterated on 34   error:2.01172
No of patterns iterated on 34   error:2.01174
No of patterns iterated on 34   error:2.01167
1.5,29229
Enter check file
iris.check3
Accuracy is 0.966667
abhishekgupta@osl-80:~/Desktop/IRIS$
```

Shuffled Data Set Partition-4

- Initialization - random between -0.25 to 0.25
- Accuracy – 1 (30/30)
- Snapshot of Result

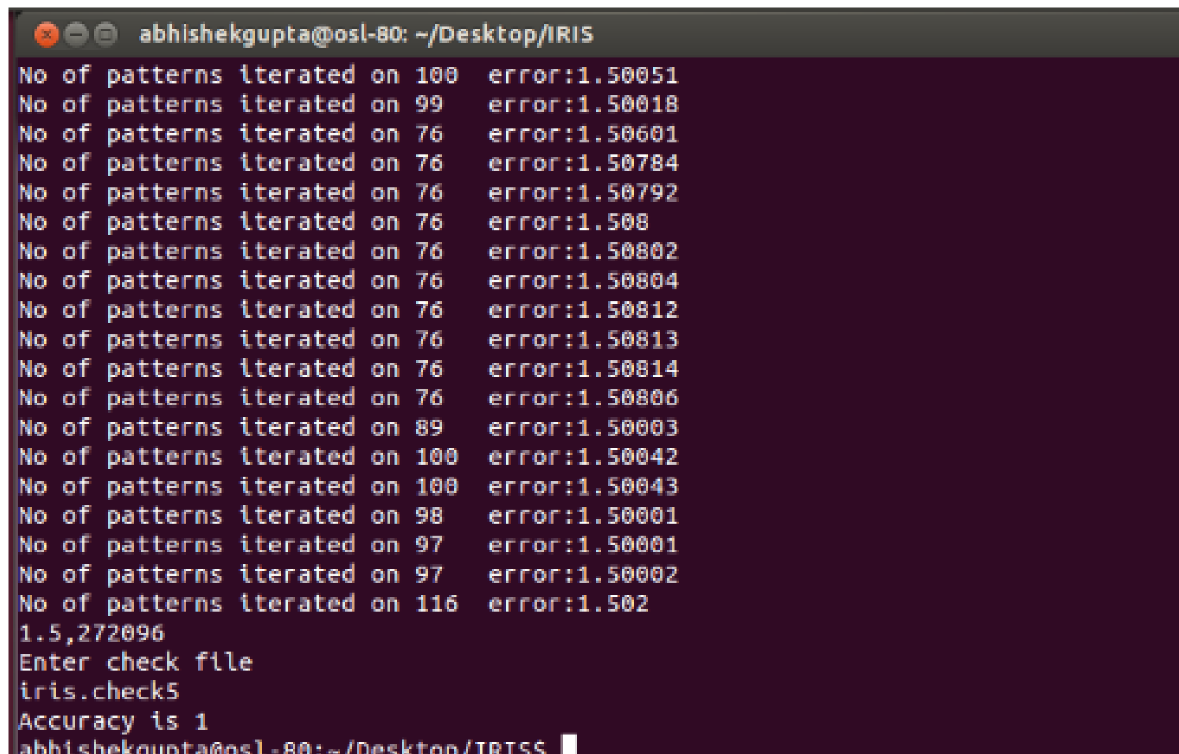
Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%



Shuffled Data Set Partition-5

- Initialization - random between -0.25 to 0.25
- Accuracy – 1 (30/30)
- Snapshot of Result →

Momentum Factor : 0.1
Learning Rate : 0.1
Threshold Error: 50%



Observations

- A shuffled data sets usually leads to better learning than cyclic, fixed orders of training patterns. However, it also takes more time to train the Network.
- An ordered presentation of the training cases to the network can cause weights/error to move very erratically over the error surface.
- Any given series of an individual class' training patterns potentially causes the network to move in weight-space in a direction that is very different from the overall desired direction

Implementation of the A-star Algorithm

Observations & Analysis

Assignment Specifications

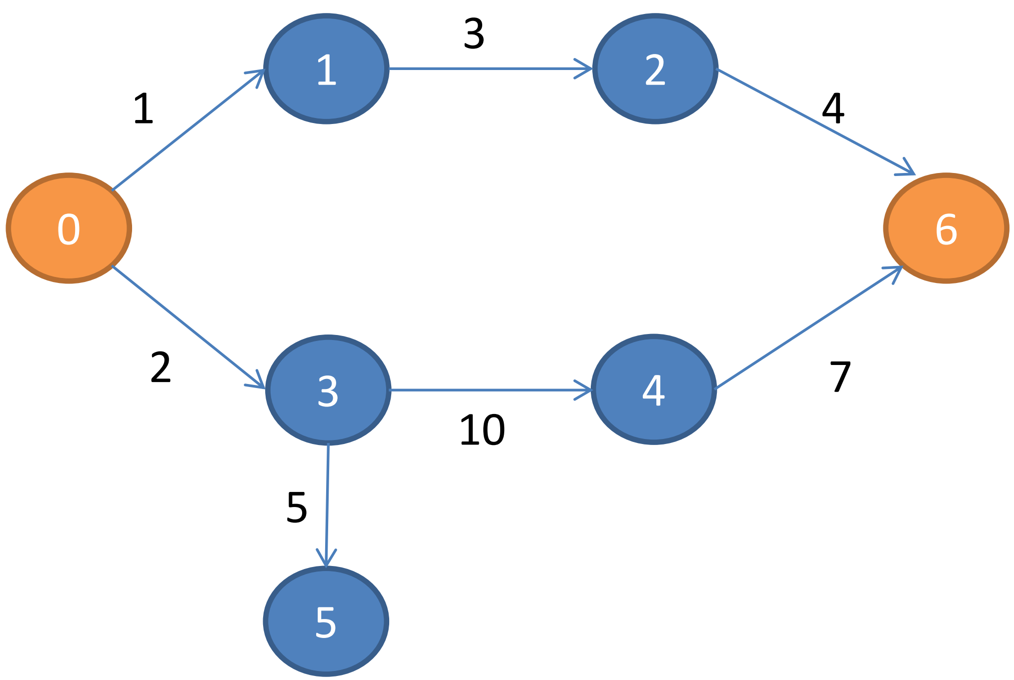
- Code A*; keep it general enough to be able to adapt to any search problem.
- Write modules for open and closed list management. Similarly for parent pointer redirection.
- Verify experimentally the intuition, "better heuristic performs better".
- Verify that "if $h(n)>h^*(n)$, for all n, A* may find the goal faster, but may discover a suboptimal path".
- Verify that monotone restriction is satisfied, parent pointer redirection for nodes on closed list is not needed.
- Come up with new heuristics for 8-puzzle and missionaries and cannibals; establish their admissibility and monotonicity or otherwise and measure performance.
- Carry out bidirectional A* search S-->G and G-->S.

A* Pseudo-Code

```
create the open list of nodes, initially containing only our starting node
create the closed list of nodes, initially empty

while (we have not reached our goal) {
        consider the best node in the open list (the node with the lowest f value)

        if (this node is the goal) {
                then we're done
        }
        else {
                move the current node to the closed list and consider all of its neighbors

                for (each neighbor) {
                        if (this neighbor is in the closed list and our current g value is lower) {
                                update the neighbor with the new, lower, g value
                                change the neighbor's parent to our current node
                        }
                        else if (this neighbor is in the open list and our current g value is lower) {
                                update the neighbor with the new, lower, g value
                                change the neighbor's parent to our current node
                        }
                        else this neighbor is not in either the open or closed list {
                                add the neighbor to the open list and set its g value
                        }
                }
        }
}
```

A-star Algorithm On a Simple Graph

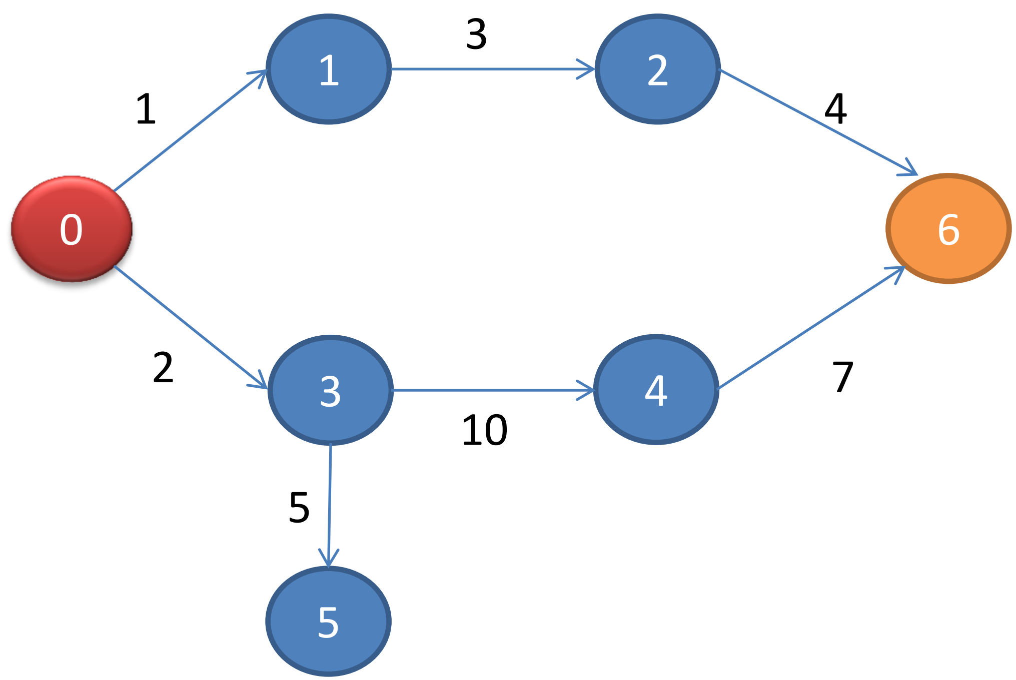
- **Without parent pointer redirection**



Start Node: 0
Goal Node: 6
h(i)= 0 for i=1,2…7

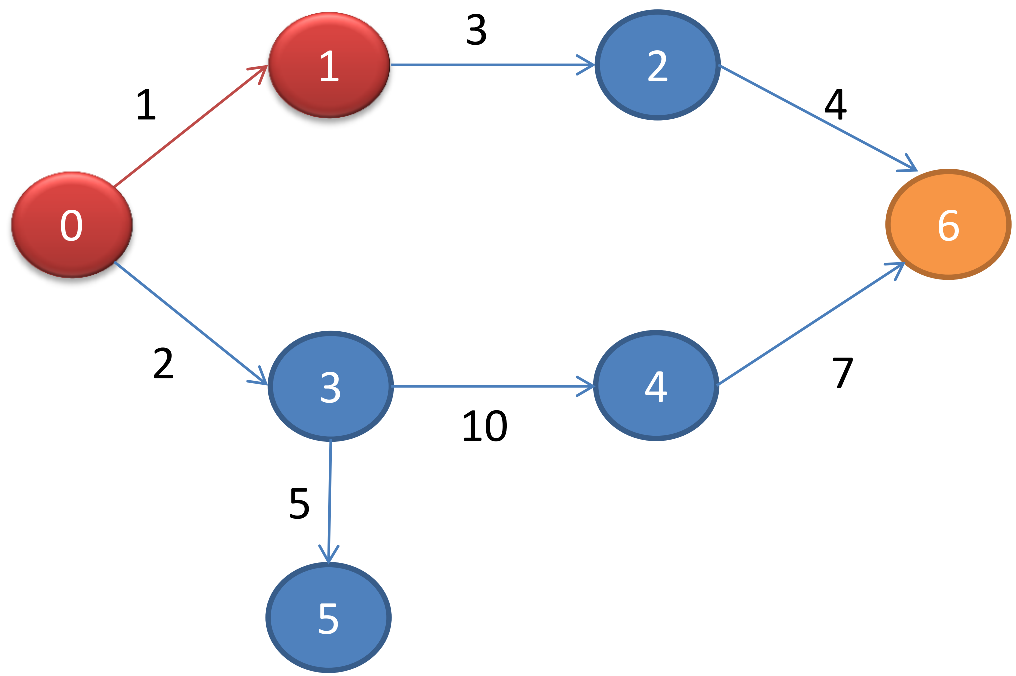
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0



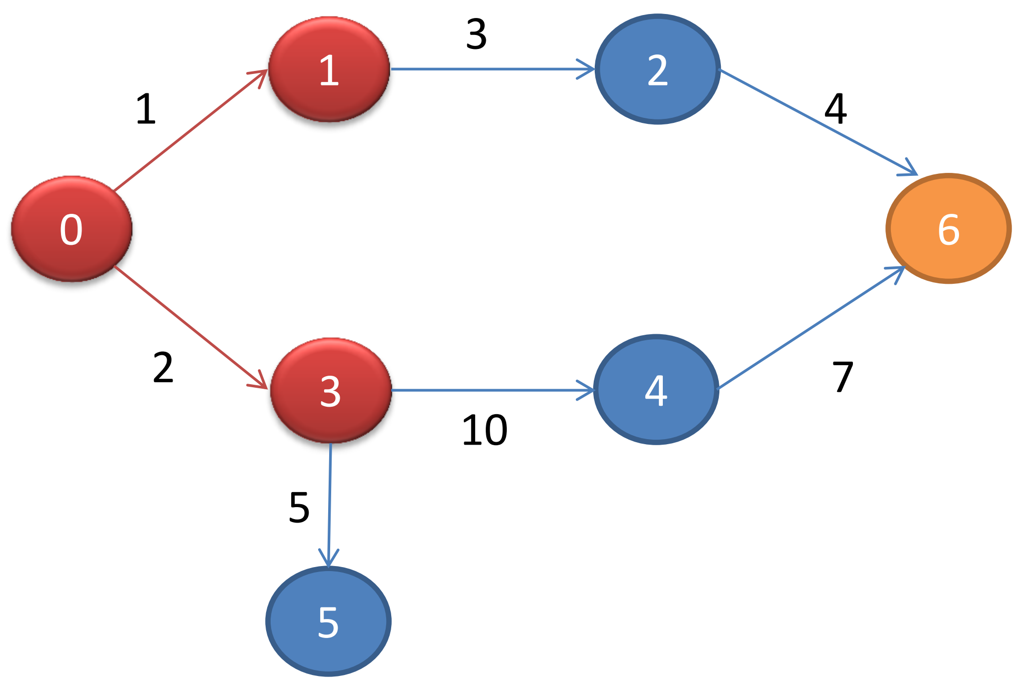
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1



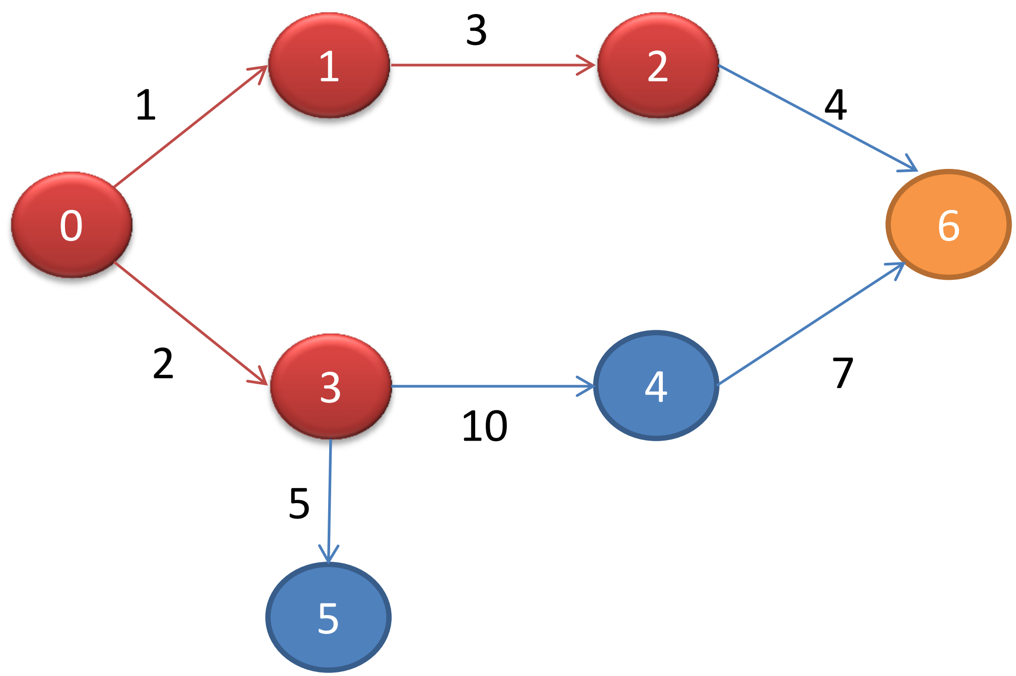
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1
  - Iteration 3 : Node 3



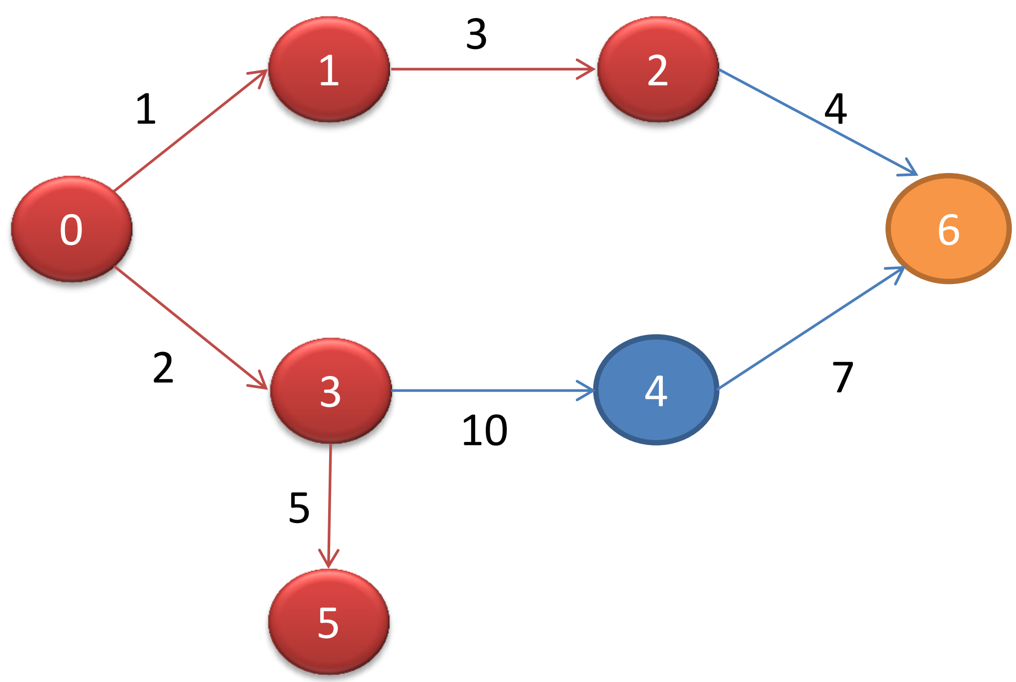
A-star Algorithm On a Simple Graph

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1
  - Iteration 3 : Node 3
  - Iteration 4 : Node 2



A-star Algorithm On a Simple Graph

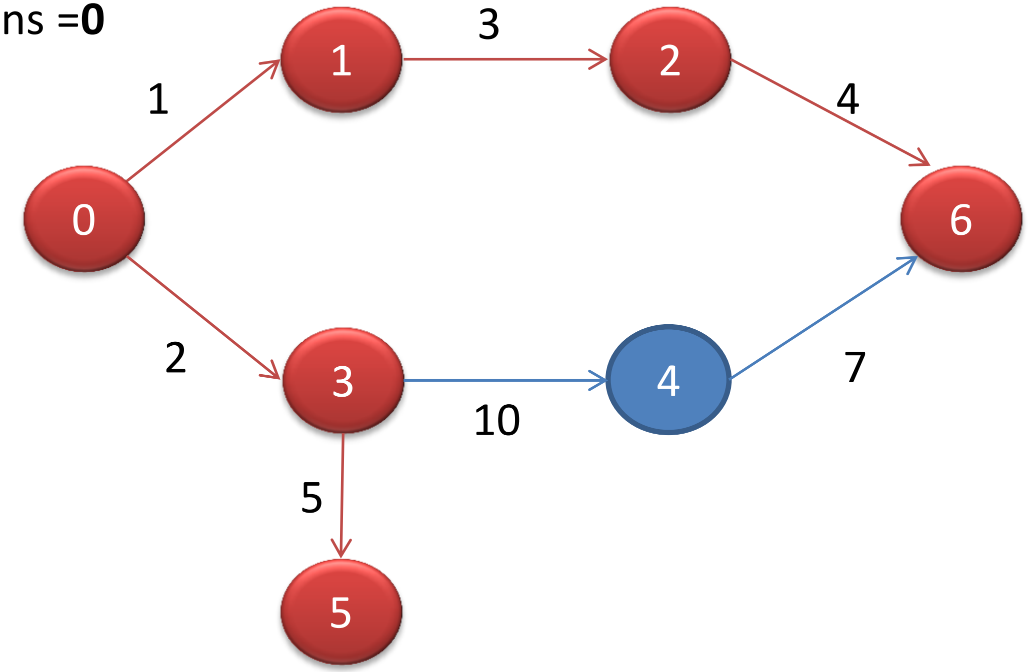
- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2 : Node 1
  - Iteration 3 : Node 3
  - Iteration 4 : Node 2
  - Iteration 5 : Node 5



A-star Algorithm On a Simple Graph

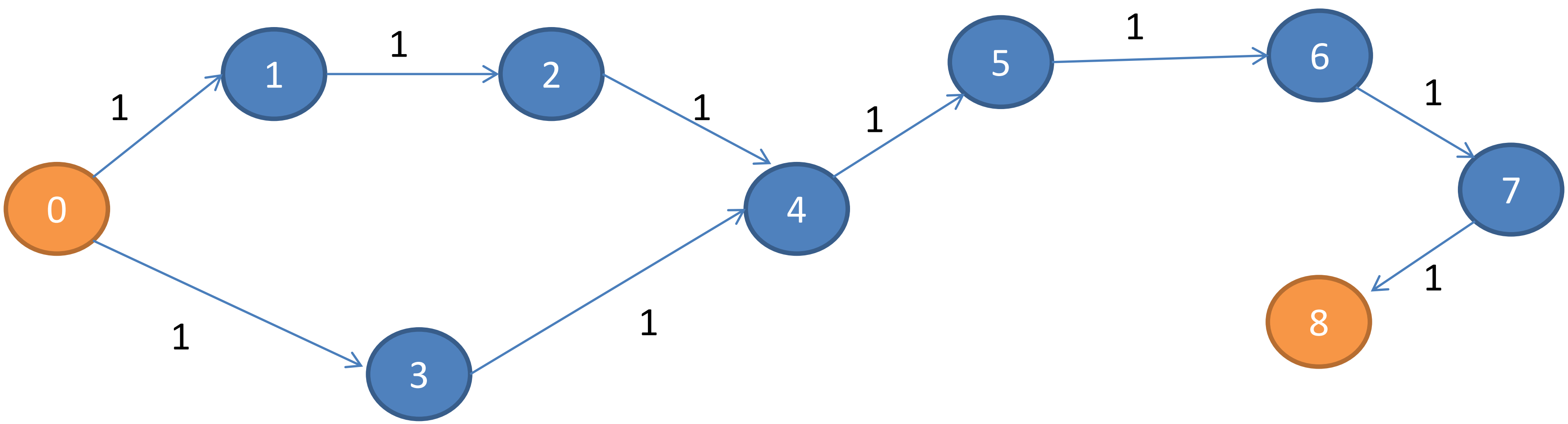
Final Output

- The optimal path is: **0 1 2 6**
- Optimal path cost is **8**
- Number of iterations taken by the A* Algo =**5**
- Number of Parent Pointer Redirections =**0**



Parent Pointer Redirection

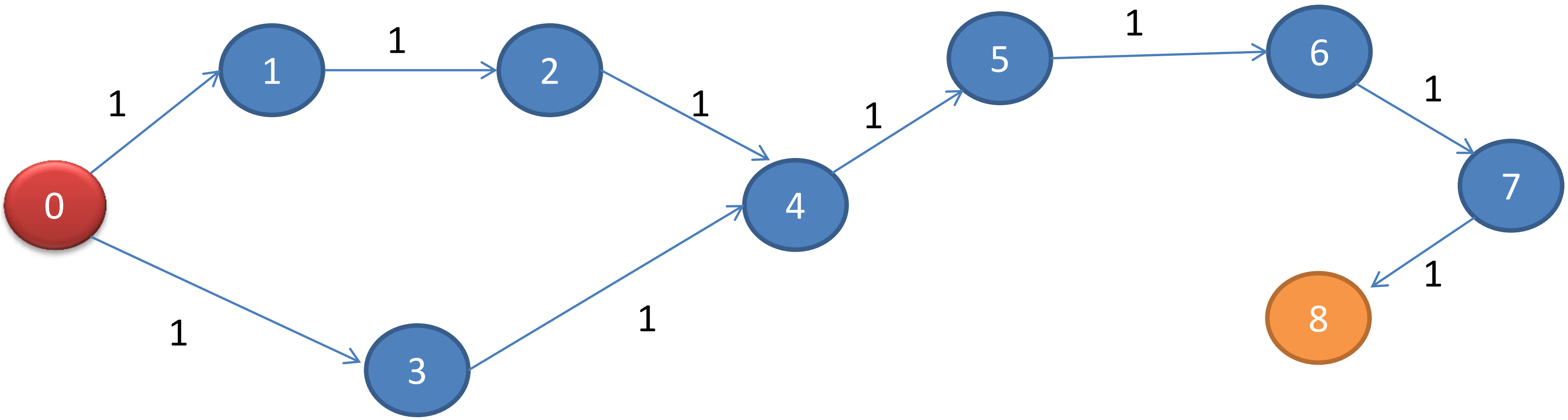
- **We shall use the following graph & heuristic, to see a case where parent pointer redirection takes place.**



Start Node: 0 Goal Node: 8 h(i) = 5 for i=3
0 otherwise

Parent Pointer Redirection

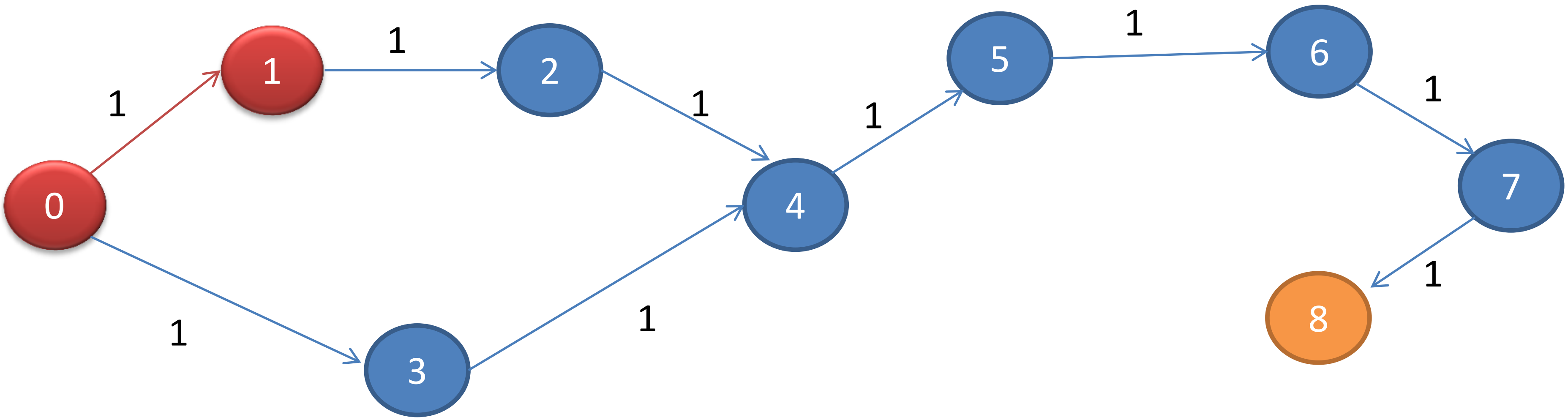
- **Iteration 1:**



The node chosen to be expanded from the open list: 0

Parent Pointer Redirection

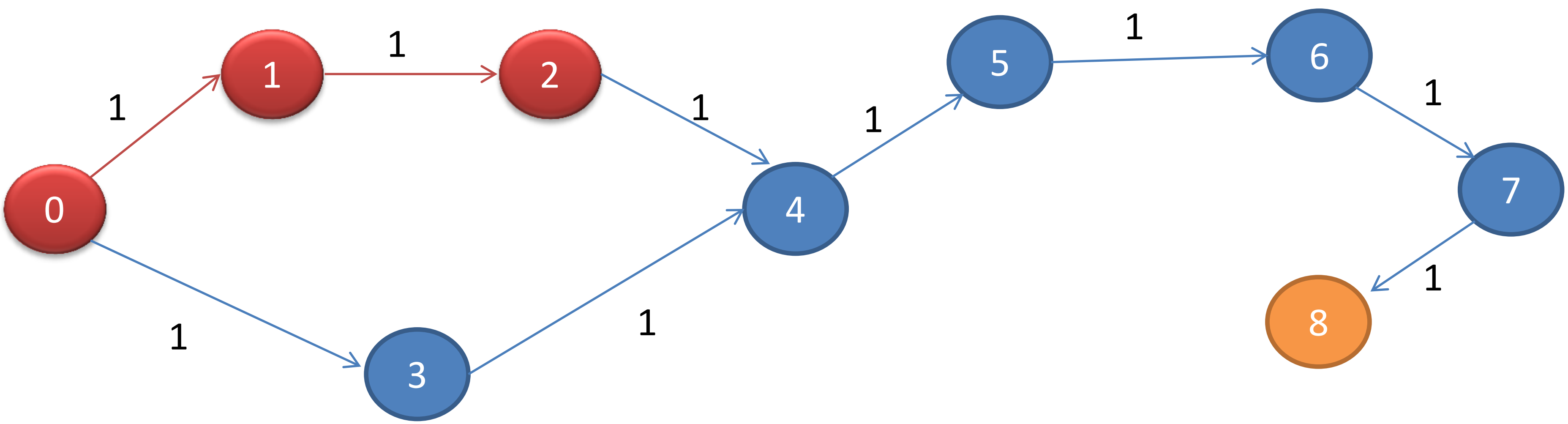
- **Iteration 2:**



The node chosen to be expanded from the open list: 1

Parent Pointer Redirection

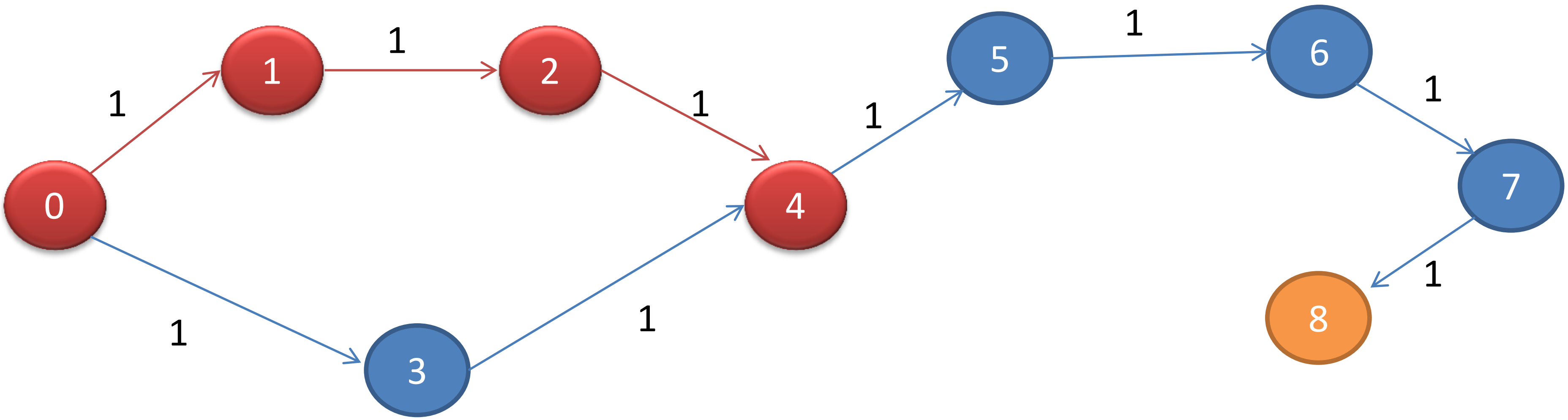
- **Iteration 3:**



The node chosen to be expanded from the open list: 2

Parent Pointer Redirection

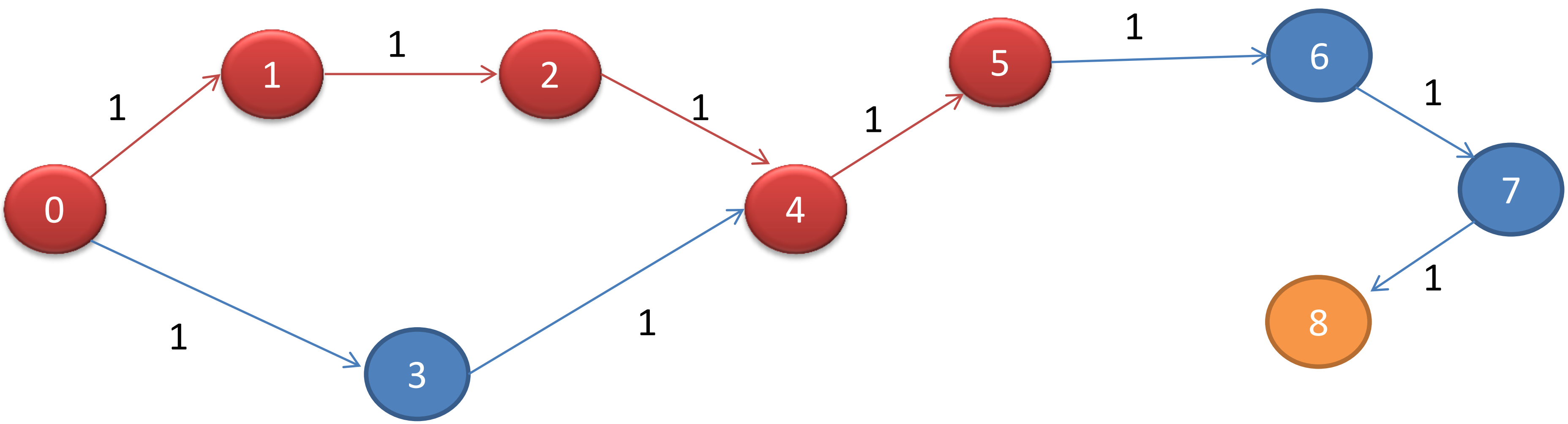
- **Iteration 4:**



The node chosen to be expanded from the open list: 4

Parent Pointer Redirection

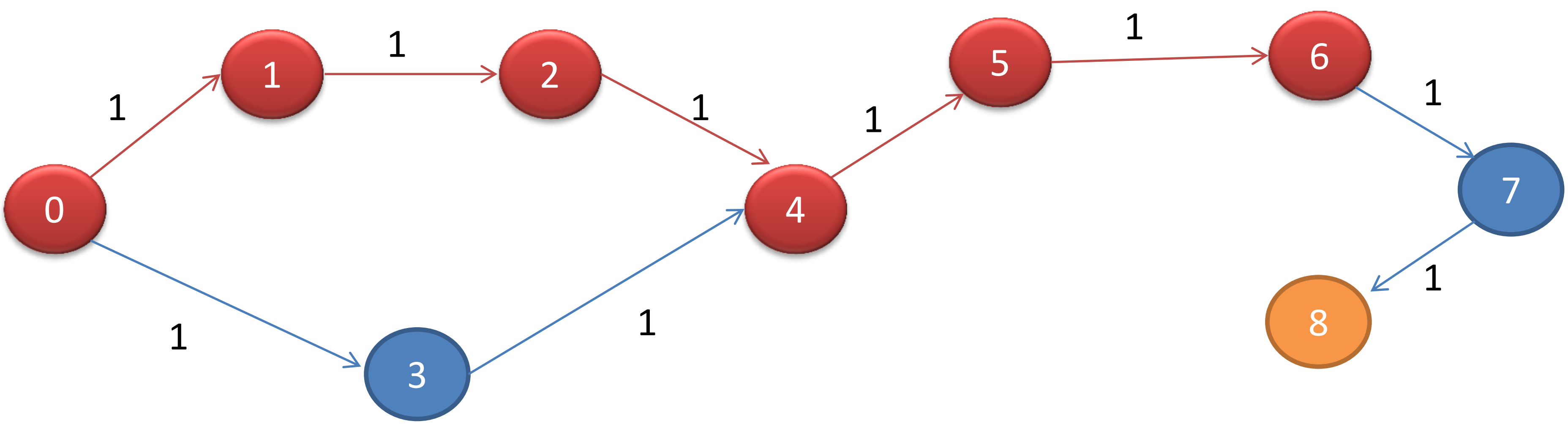
- **Iteration 5:**



The node chosen to be expanded from the open list: 5

Parent Pointer Redirection

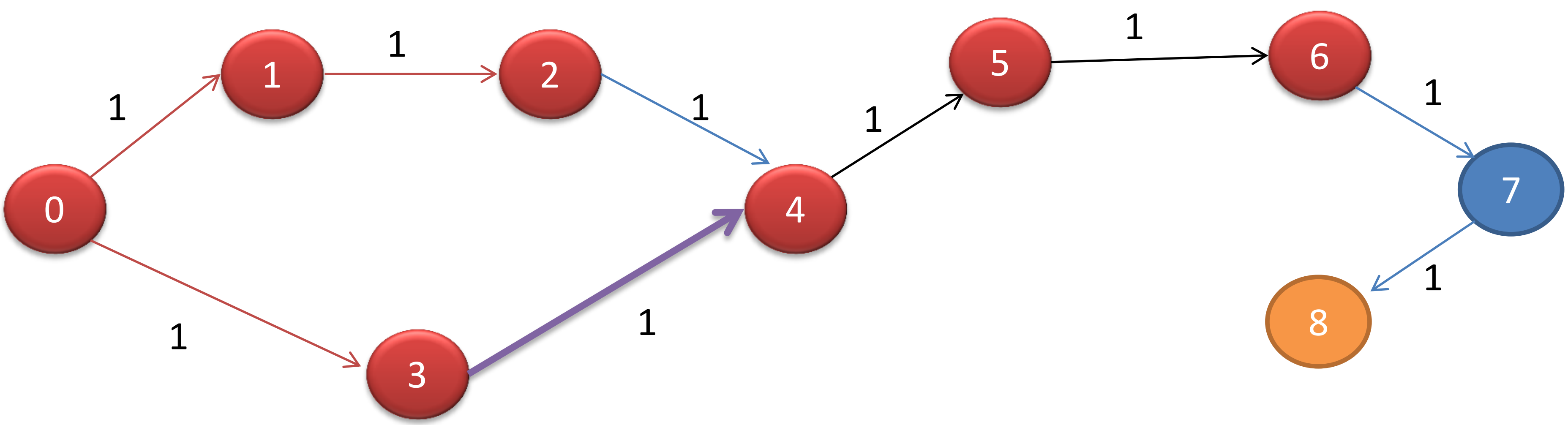
- **Iteration 6:**



The node chosen to be expanded from the open list: 6

Parent Pointer Redirection

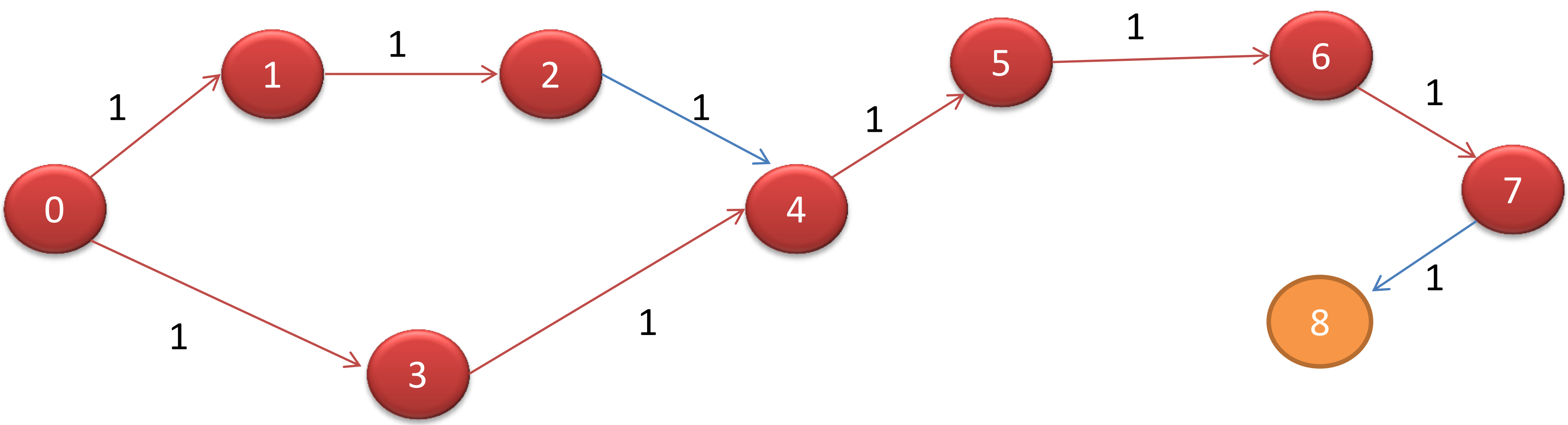
- **Iteration 7:**



The node chosen to be expanded from the open list: 3
1 Parent Pointers Redirected: 3 → 4
g values changed for 3 nodes: 4 5 6

Parent Pointer Redirection

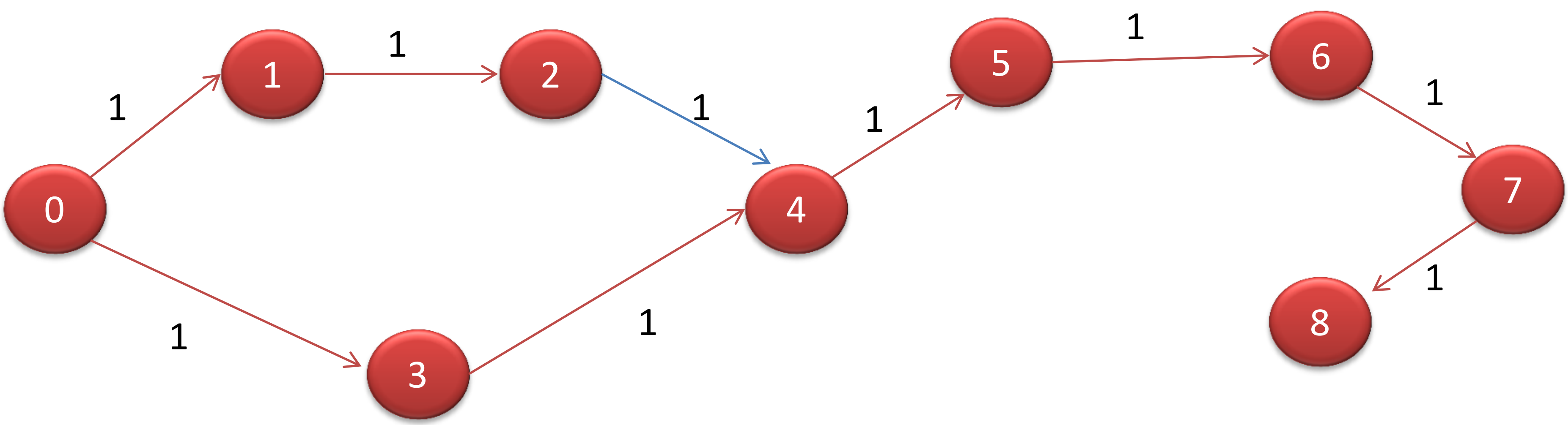
- **Iteration 8:**



The node chosen to be expanded from the open list: 7

Parent Pointer Redirection

- **Final Output:**



- The optimal path is:0 3 4 5 6 7 8
- Optimal path cost is 6
- Number of iterations taken by the A* Algo =8
- Number of Parent Pointer Redirections =3

Better Heuristic Performs Better

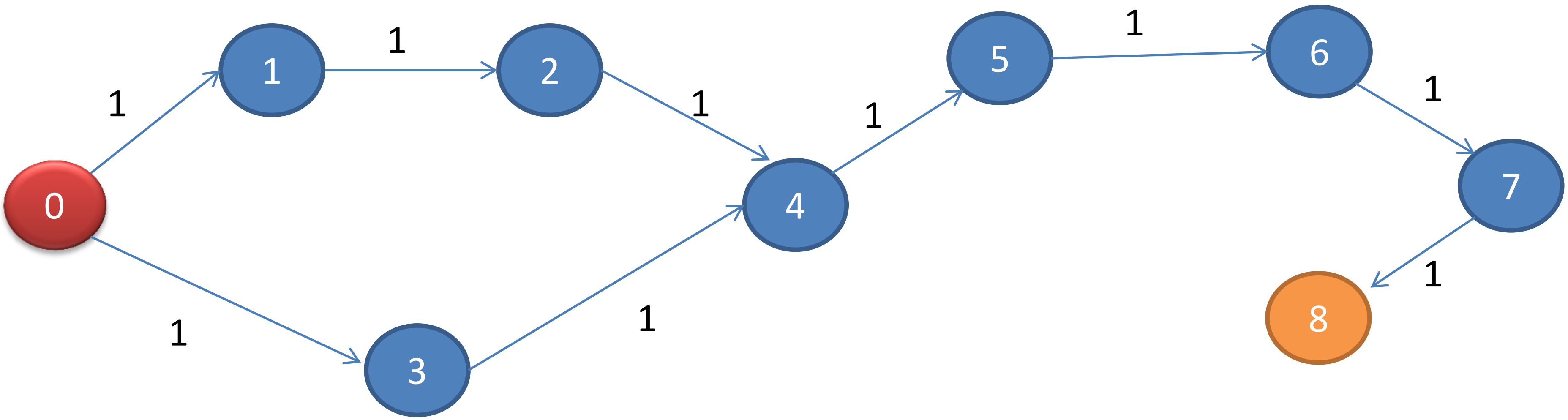
- We now change the heuristic of previous case as follows:
  - $h(0) = 6$
  - $h(1) = 6$
  - $h(2) = 5$
  - $h(3) = 5$
  - $h(4) = 4$

  $h(5) = 3$
  $h(6) = 2$
  $h(7) = 1$
  $h(8) = 0$

- Clearly above heuristic is better than the previous heuristic which was $h(i) = \begin{cases} 5 & \text{for } i=3 \\ 0 & \text{otherwise} \end{cases}$

Better Heuristic Performs Better

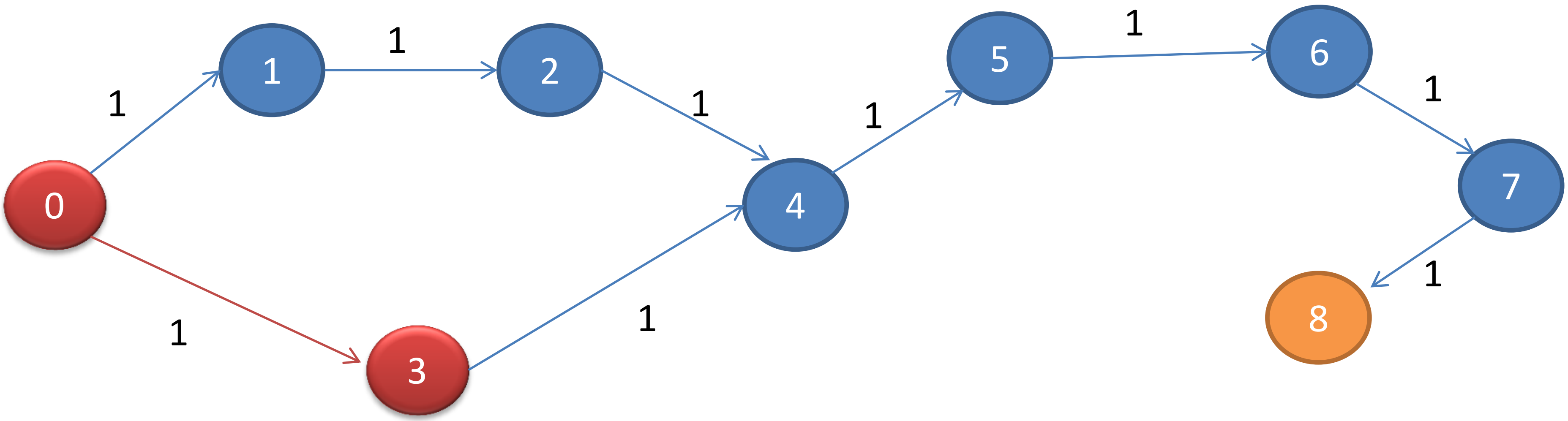
- **Iteration 1:**



The node chosen to be expanded from the open list: 0

Better Heuristic Performs Better

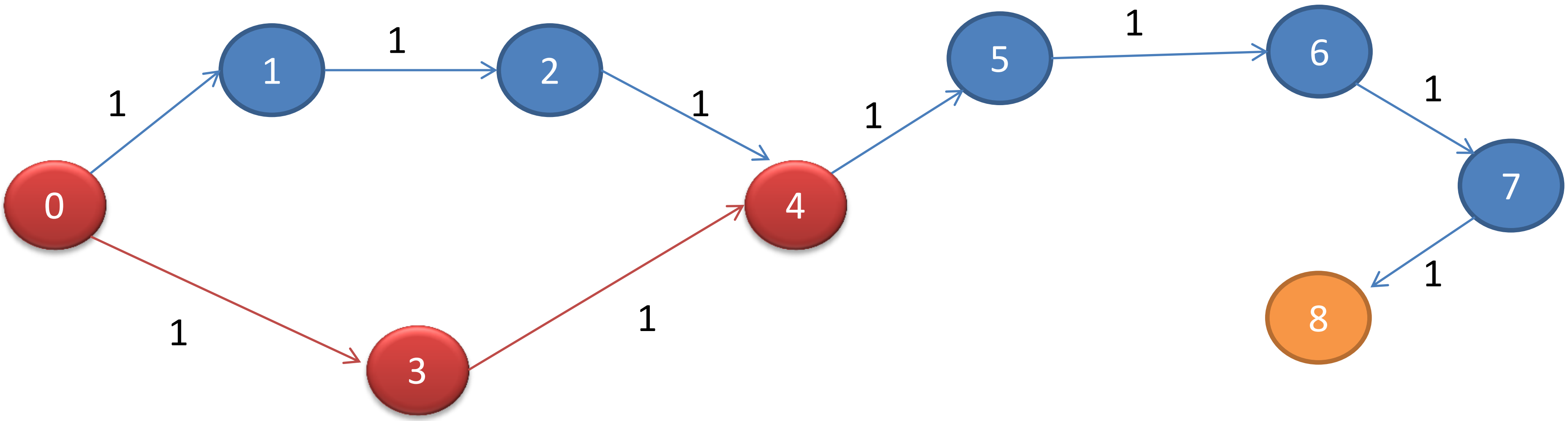
- **Iteration 2:**



The node chosen to be expanded from the open list: 3

Better Heuristic Performs Better

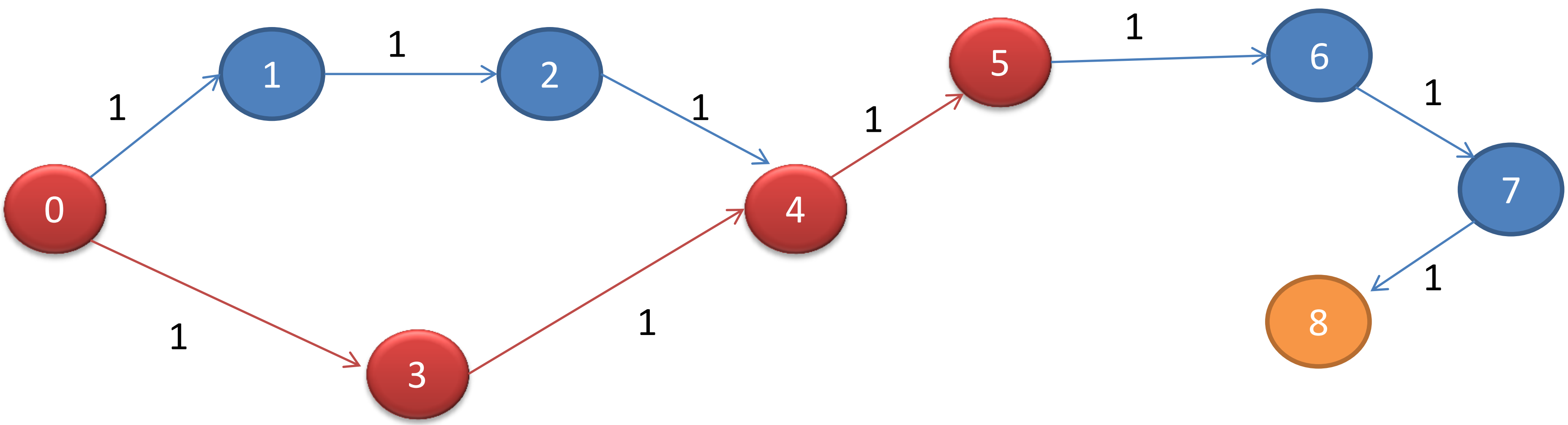
- **Iteration 3:**



The node chosen to be expanded from the open list: 4

Better Heuristic Performs Better

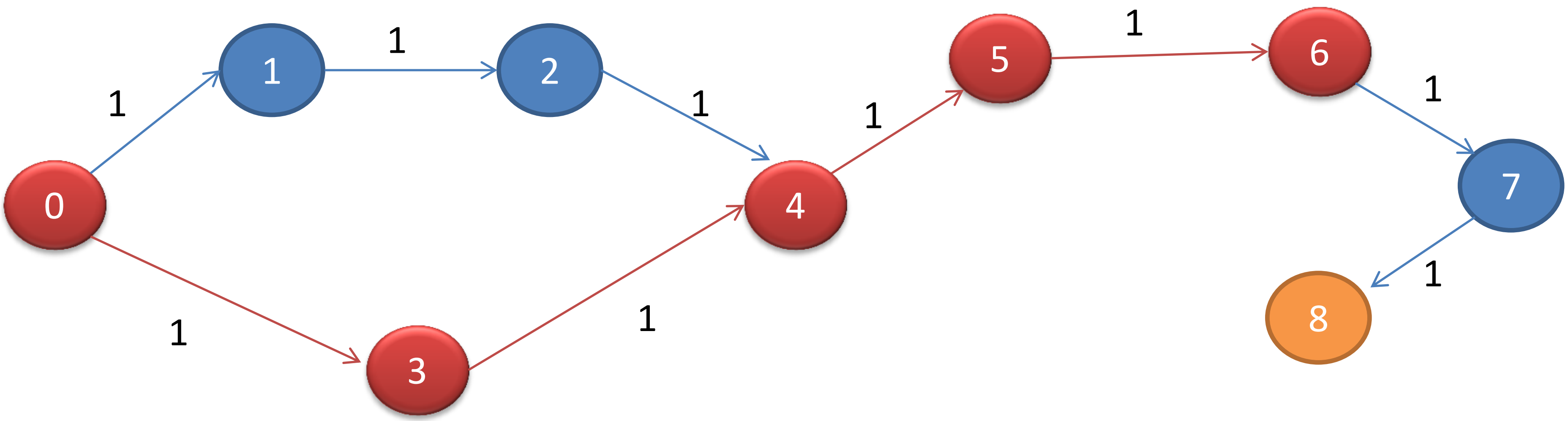
- **Iteration 4:**



The node chosen to be expanded from the open list: 5

Better Heuristic Performs Better

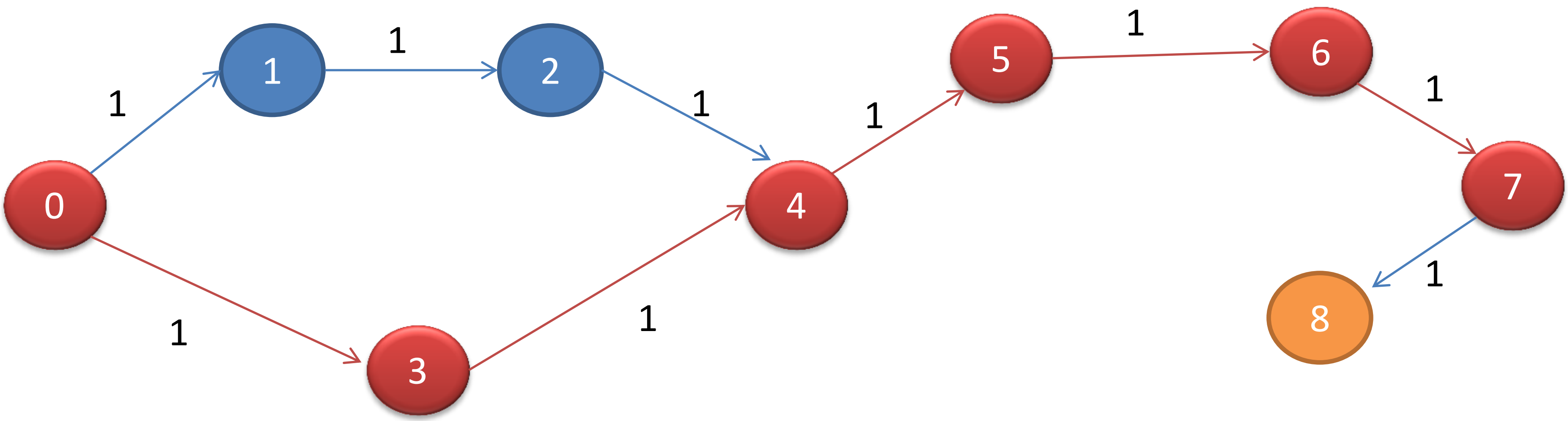
- **Iteration 5:**



The node chosen to be expanded from the open list: 6

Better Heuristic Performs Better

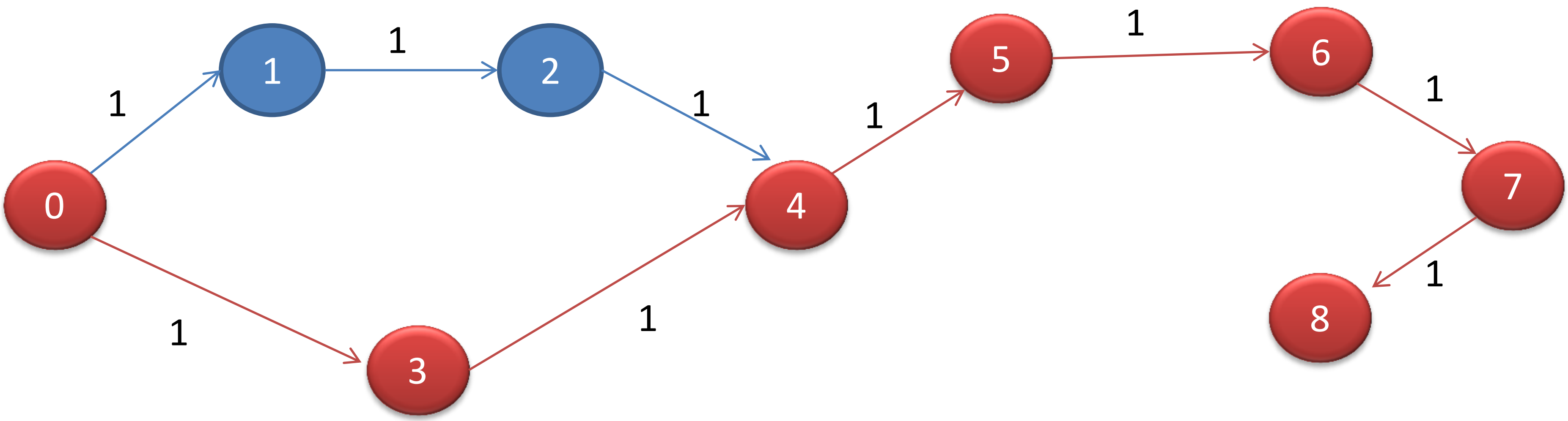
- **Iteration 6:**



The node chosen to be expanded from the open list: 7

Better Heuristic Performs Better

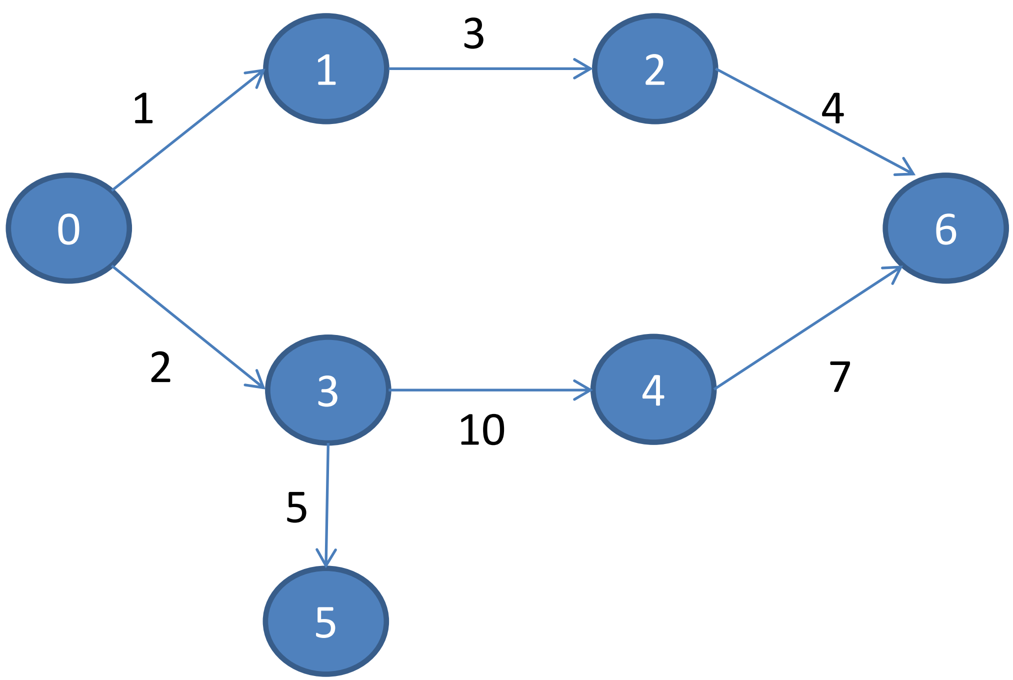
- **Final Output:**



- The optimal path **is:0 3 4 5 6 7 8**
- Optimal path cost is **6**
- Number of iterations taken by the A* Algo =**6(as opposed to 8 previously)**
- Number of Parent Pointer Redirections =**0**
- **So we see that with a better heuristic A* algorithms converges faster**

h greater than h*

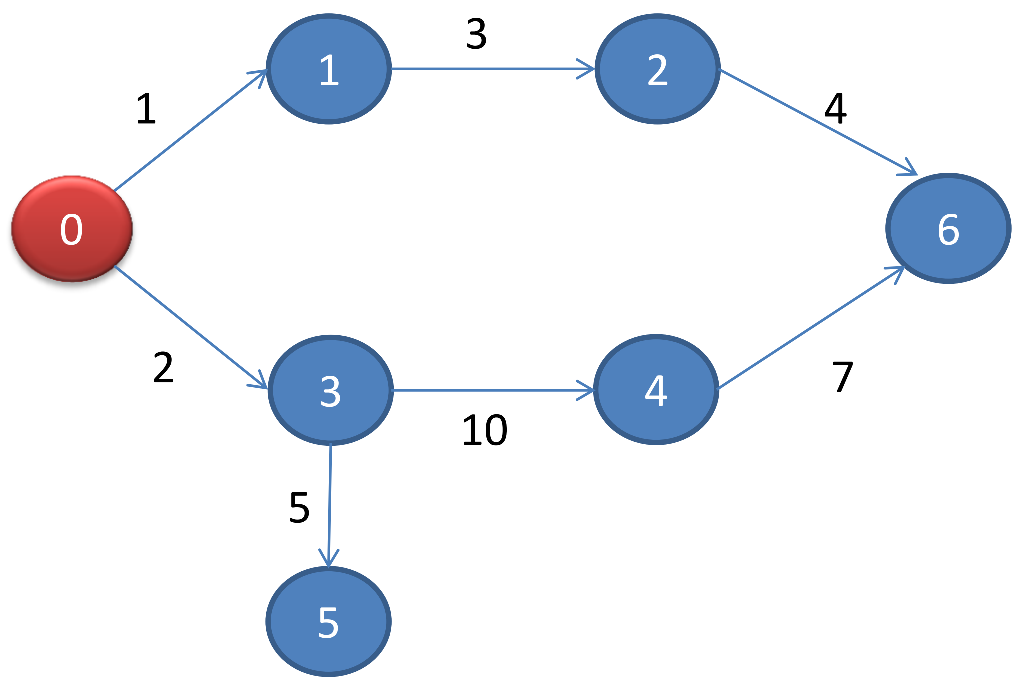
- We now verify that “if h(n)>h*(n)”, for all n, A* may find the goal faster, but may discover a suboptimal path.
- We consider the graph used initially and run the A* algorithm on it using a heuristic such that h > h*(n) for all n



Start Node: 0
Goal Node: 6
h(1)= 20
h(1)= 30
h(1)= 40
h(1)= 10
h(1)= 10
h(1)= 10
h(1)= 0

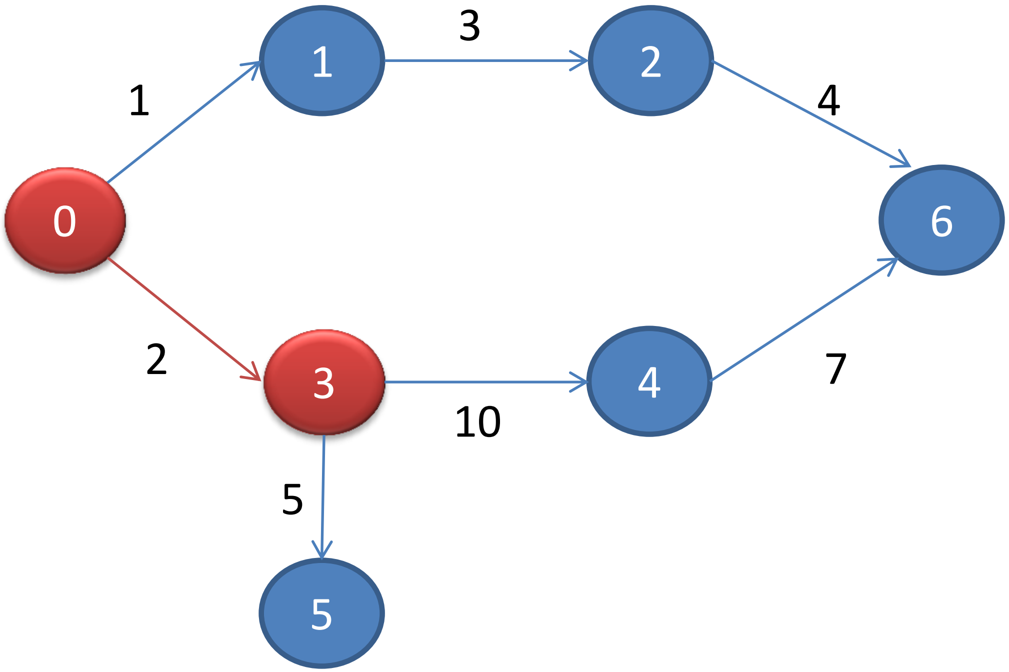
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0



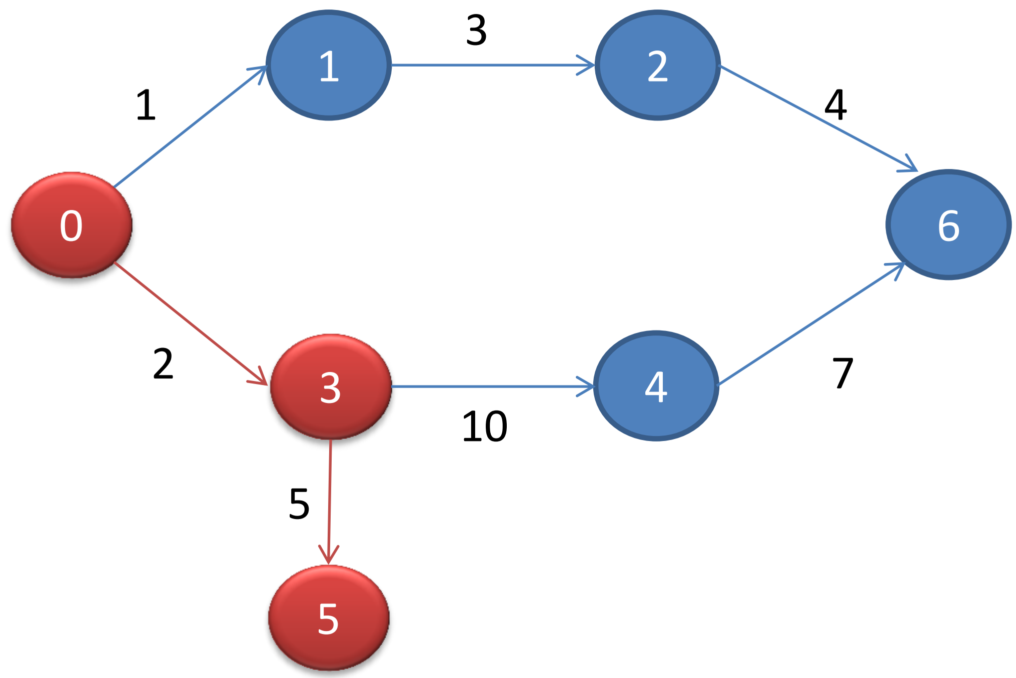
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2: Node 3



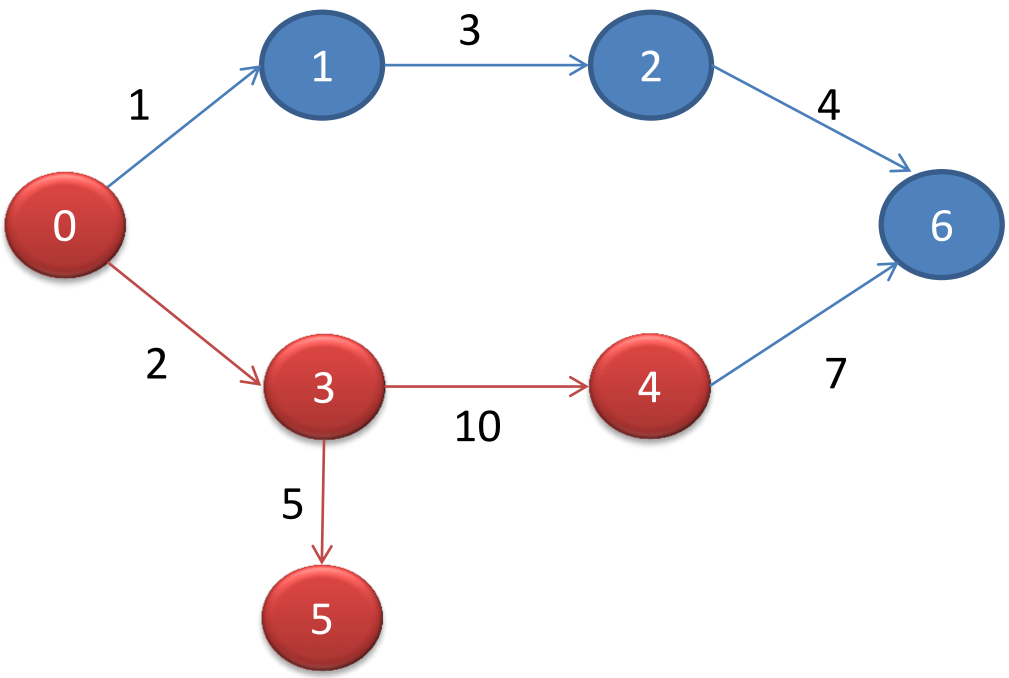
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2: Node 3
  - Iteration 3: Node 5



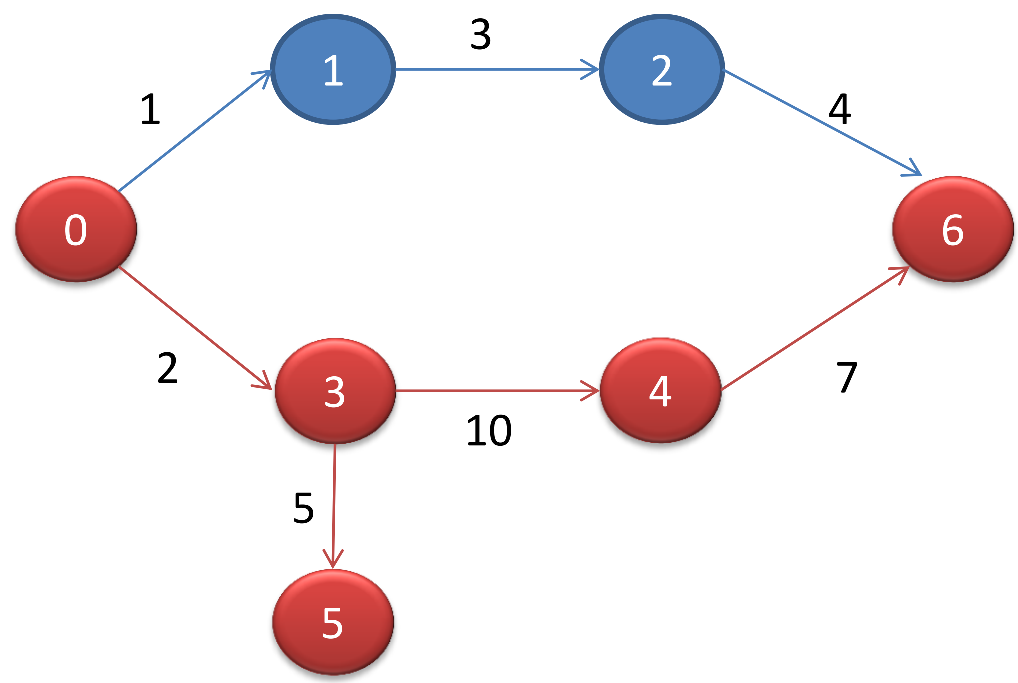
h greater than h*

- **Node picked by algorithm from open list:**
  - Iteration 1: Node 0
  - Iteration 2: Node 3
  - Iteration 3: Node 5
  - Iteration 4: Node 4



h greater than h*

- **Final Output:**
  - Number of iterations taken by the A* Algo =4
  - Number of Parent Pointer Redirections =0
  - Found path is:0 3 4 6
  - Found path cost is 19
  - The discovered path is a suboptimal path.



Monotone Restriction

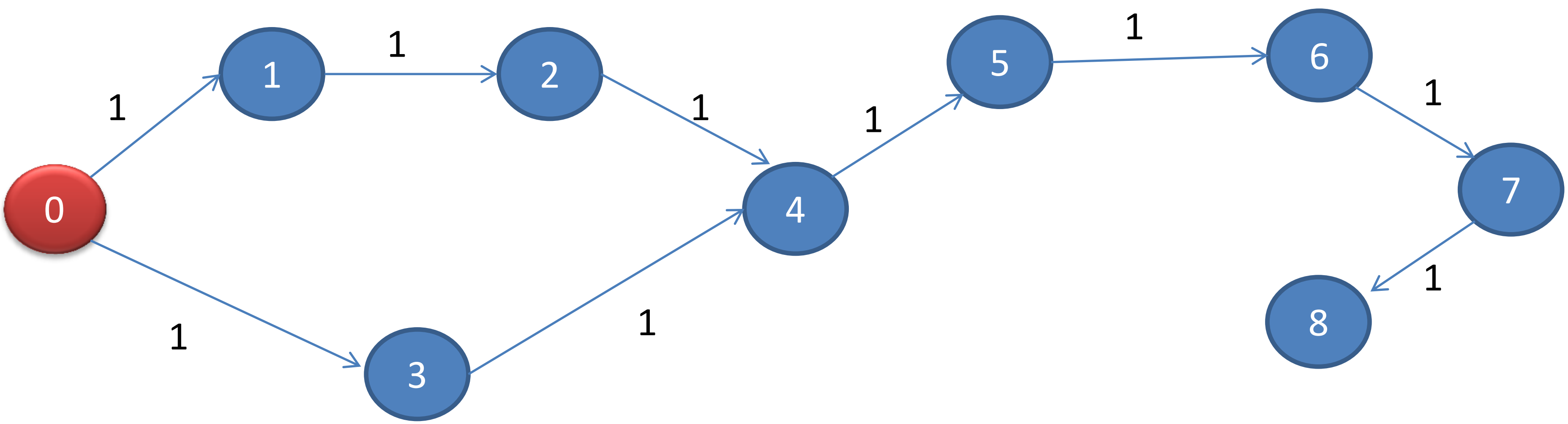
- We now change the heuristic of second case as follows:
  - $h(0) = 6$
  - $h(1) = 6$
  - $h(2) = 5$
  - $h(3) = 5$
  - $h(4) = 4$
  - $h(5) = 3$
  - $h(6) = 2$
  - $h(7) = 1$
  - $h(8) = 0$

- Note that in the above case MR is satisfied i.e.

$$h(\text{parent}) \le h(\text{child}) + C(\text{parent}, \text{child})$$

Monotone Restriction

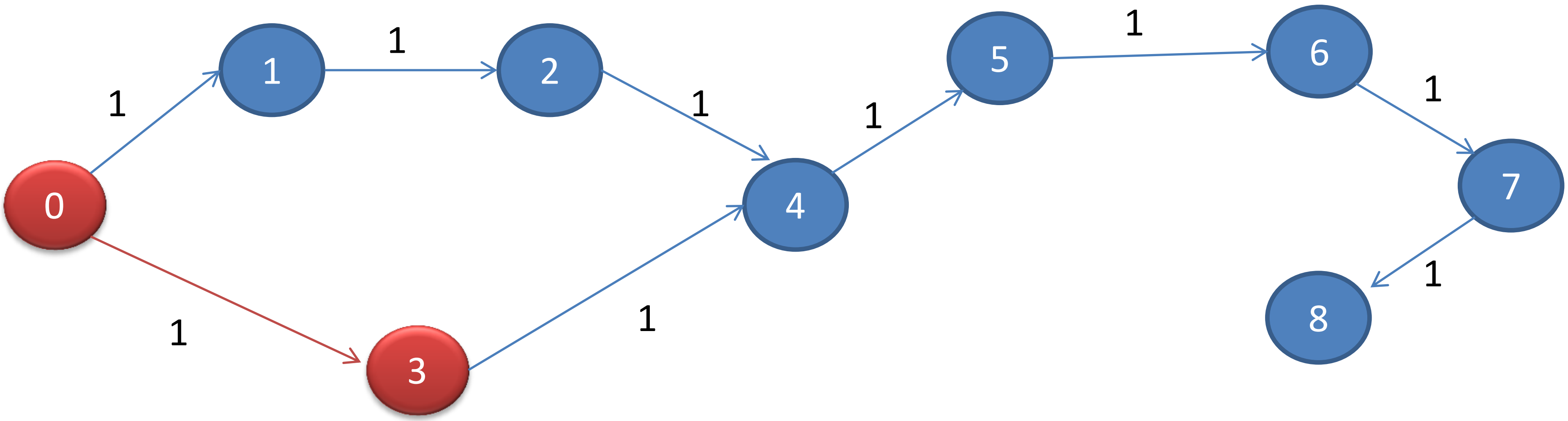
- **Iteration 1:**



The node chosen to be expanded from the open list: 0

Monotone Restriction

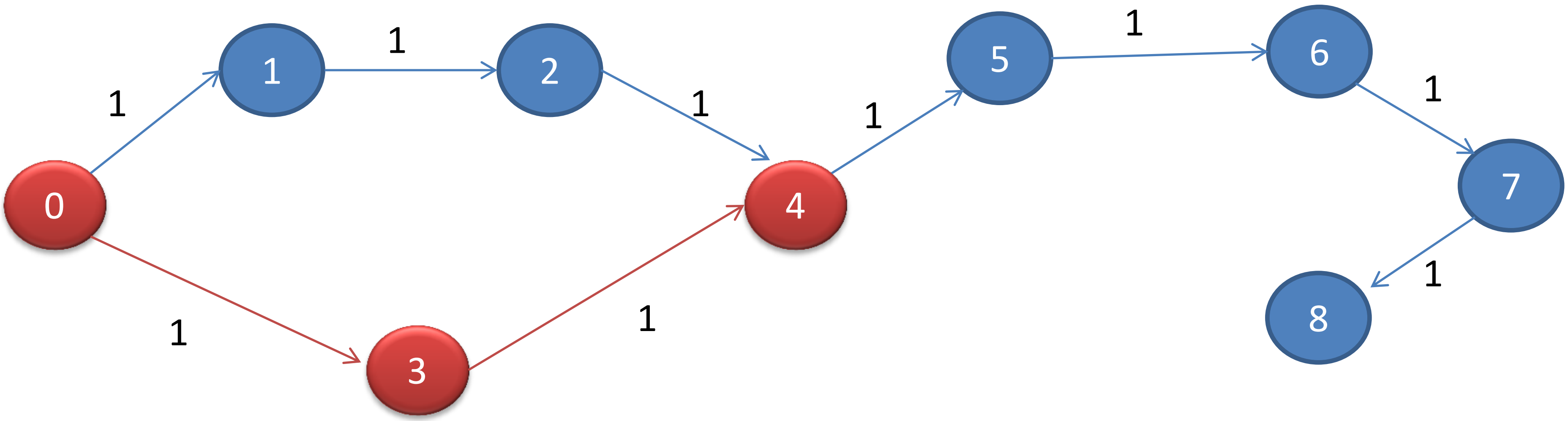
- **Iteration 2:**



The node chosen to be expanded from the open list: 3

Monotone Restriction

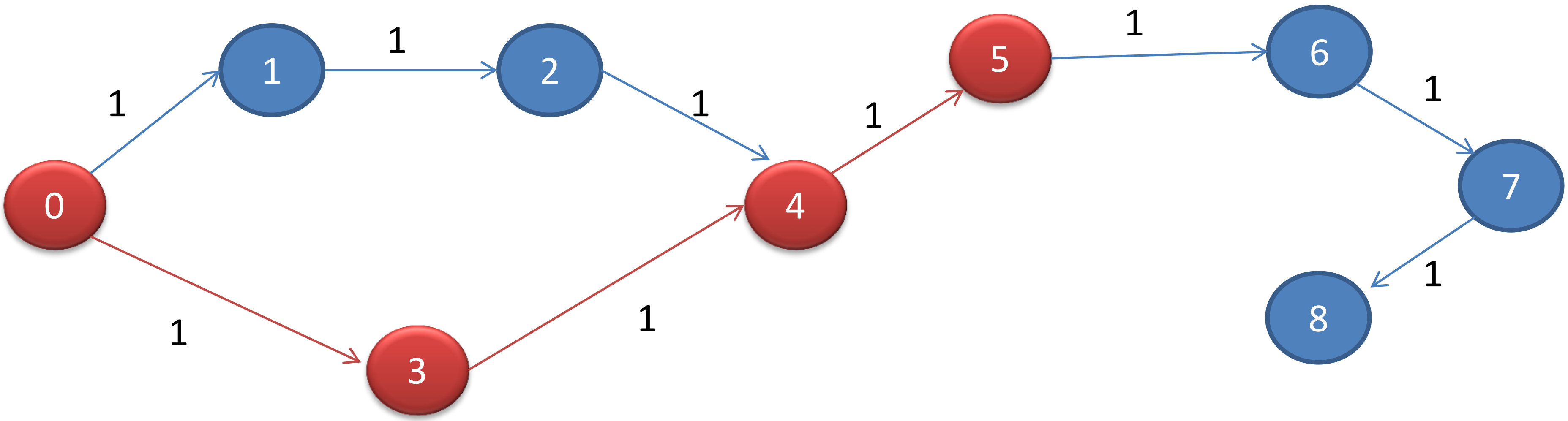
- **Iteration 3:**



The node chosen to be expanded from the open list: 4

Monotone Restriction

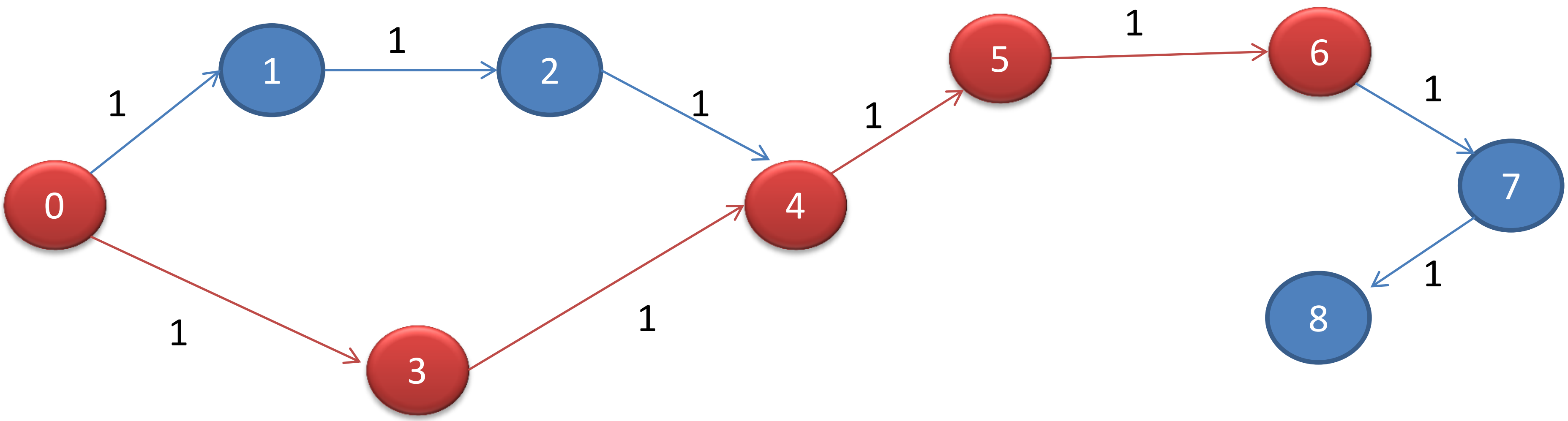
- **Iteration 4:**



The node chosen to be expanded from the open list: 5

Monotone Restriction

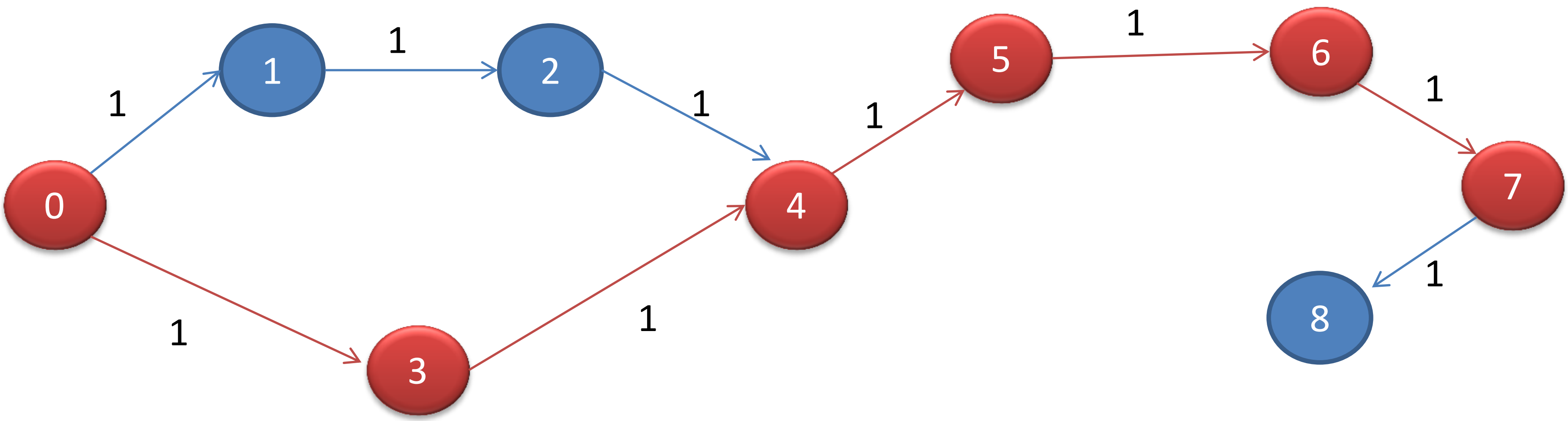
- **Iteration 5:**



The node chosen to be expanded from the open list: 6

Monotone Restriction

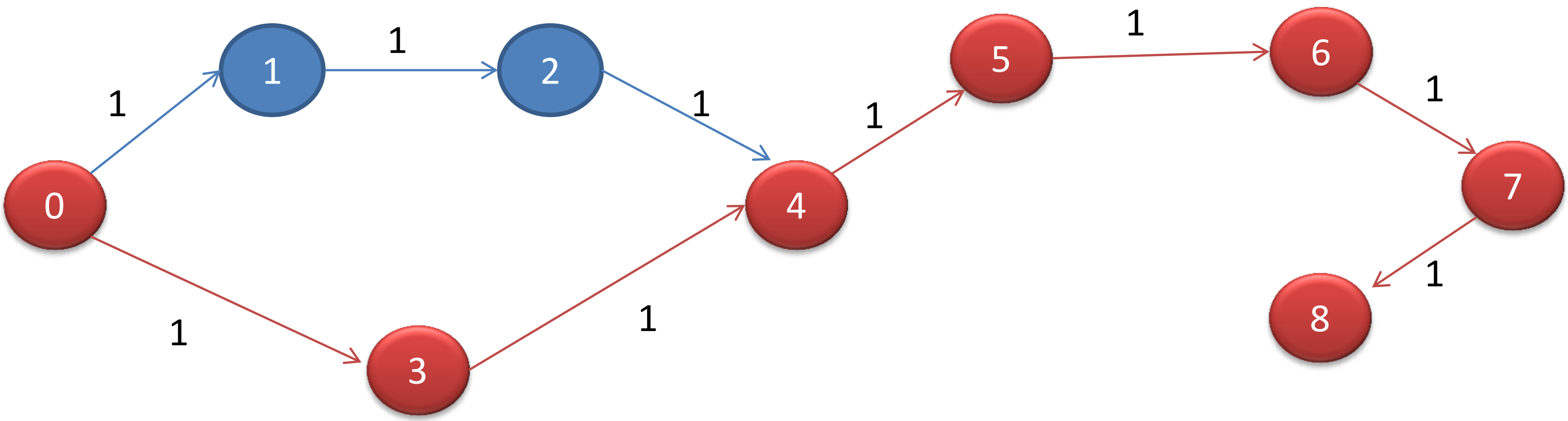
- **Iteration 6:**



The node chosen to be expanded from the open list: 7

Monotone Restriction

- **Final Output:**



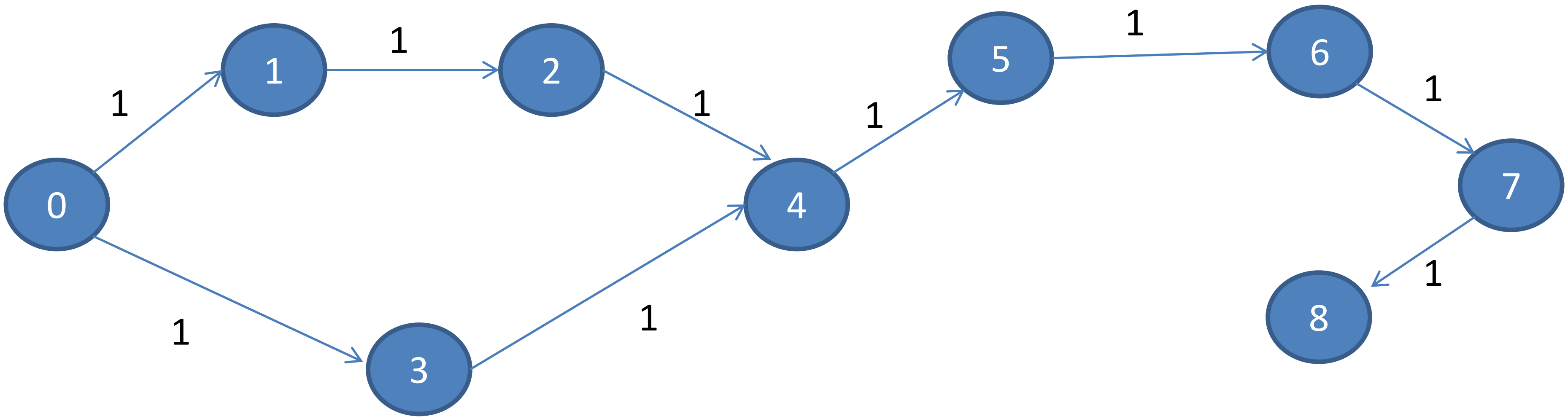
- The optimal path **is:0 3 4 5 6 7 8**
- Optimal path cost is **6**
- Number of iterations taken by the A* Algo =**6**
- Number of Parent Pointer Redirections =**0**
- **So we see that with a when Monotone Restriction is satisfied, parent pointer redirection is not needed**

Bidirectional Search Algorithm

- **Carry out the Forward and Backward search parallely.**
- **For both Forward Search & Backward Search, maintain separate Open Lists and Closed Lists.**
- **Push the Start Node in Open List of Forward Search and Goal Node in Open List of Backward Search.**
- **In each iteration of A* Algorithm,**
  - **Carry out Forward Search.**
  - **Carry out Backward Search.**
  - **Check the intersection of CL of Forward Search & CL of Backward Search.**
  - **If intersection is not NULL, Stop!**
  - **Otherwise carry out another iteration of A* Search**
- **The path discovered is given by :**
  - **Following the parent pointers from Start Node to Intersection Node in Forward Search**
  - **Reversing the direction of Parent Pointers from intersection node to Start node of Backward Search**

Bidirectional Search

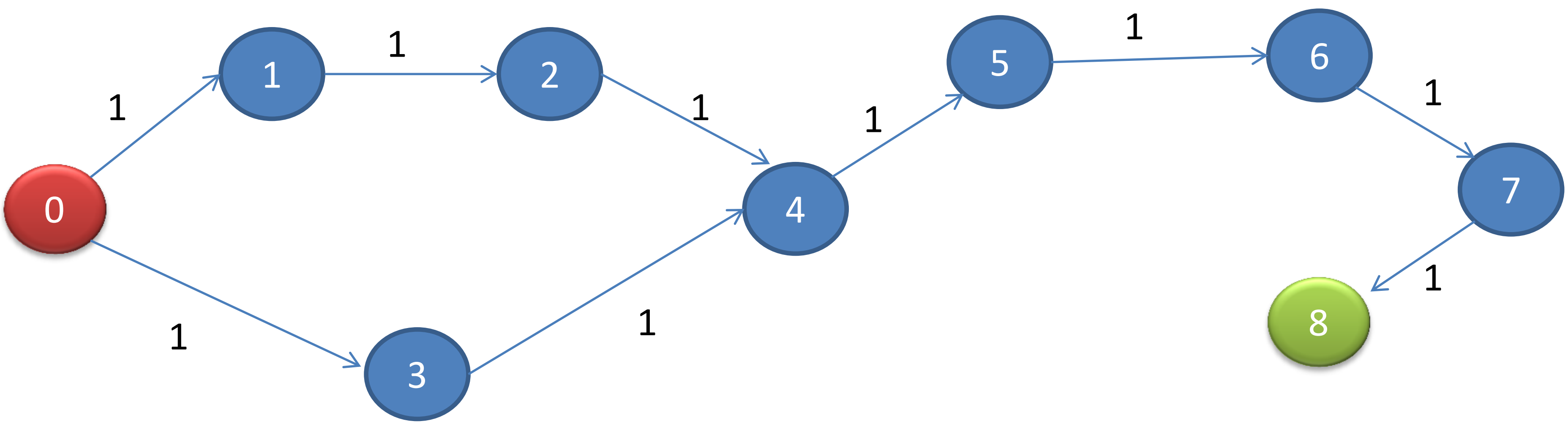
- **We shall use the following graph & heuristic, to do a bidirectional search.**



- **Heuristic** h(i) = 5 for i=3
- 0 otherwise
- **Forward Pass**: Start Node: 0 Goal Node: 8
- **Reverse Pass**: Start Node: 8 Goal Node: 0
- **We chose a node to be expanded from the Open List in forward direction and similarly in backward direction.**

Bidirectional Search

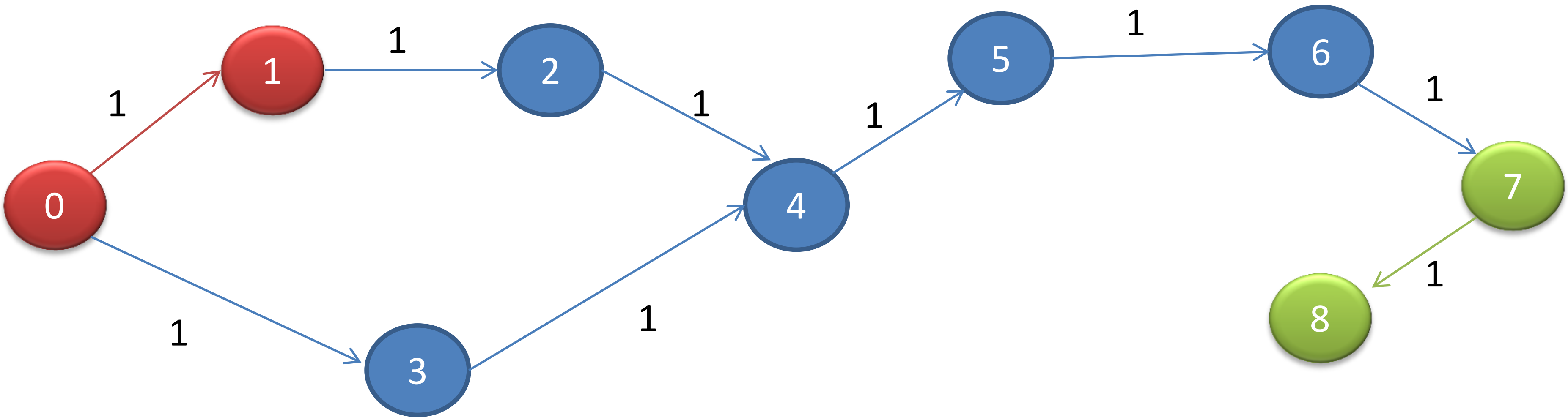
- **Iteration 1:**



The node chosen to be expanded from the open list in FORWARD direction: 0
The node chosen to be expanded from the open list in BACKWARD direction: 8

Bidirectional Search

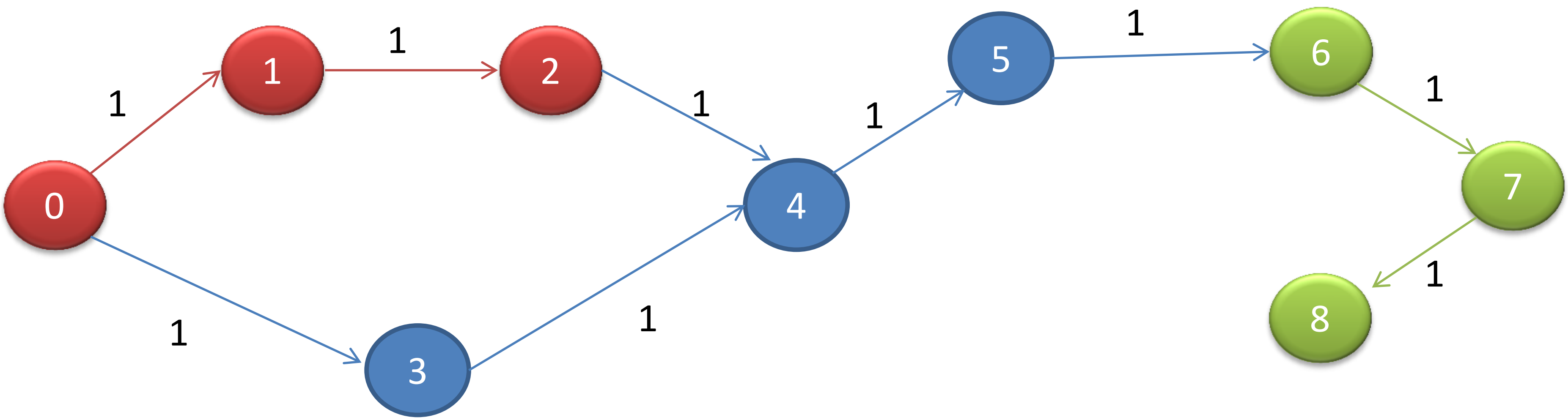
- **Iteration 2:**



The node chosen to be expanded from the open list in FORWARD direction: 1
The node chosen to be expanded from the open list in BACKWARD direction: 7

Bidirectional Search

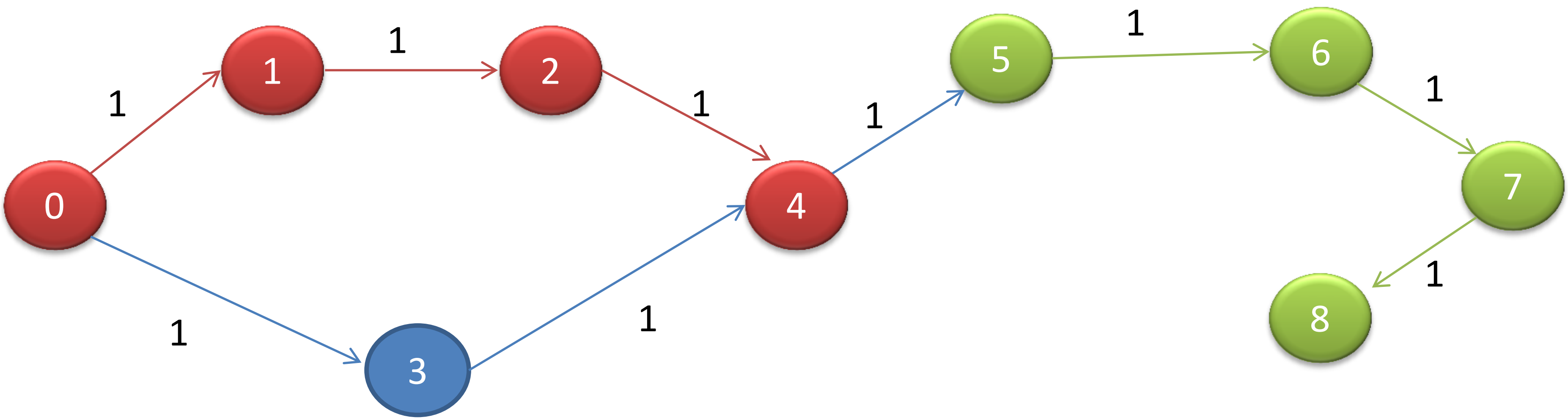
- **Iteration 3:**



The node chosen to be expanded from the open list in FORWARD direction: 2
The node chosen to be expanded from the open list in BACKWARD direction: 6

Bidirectional Search

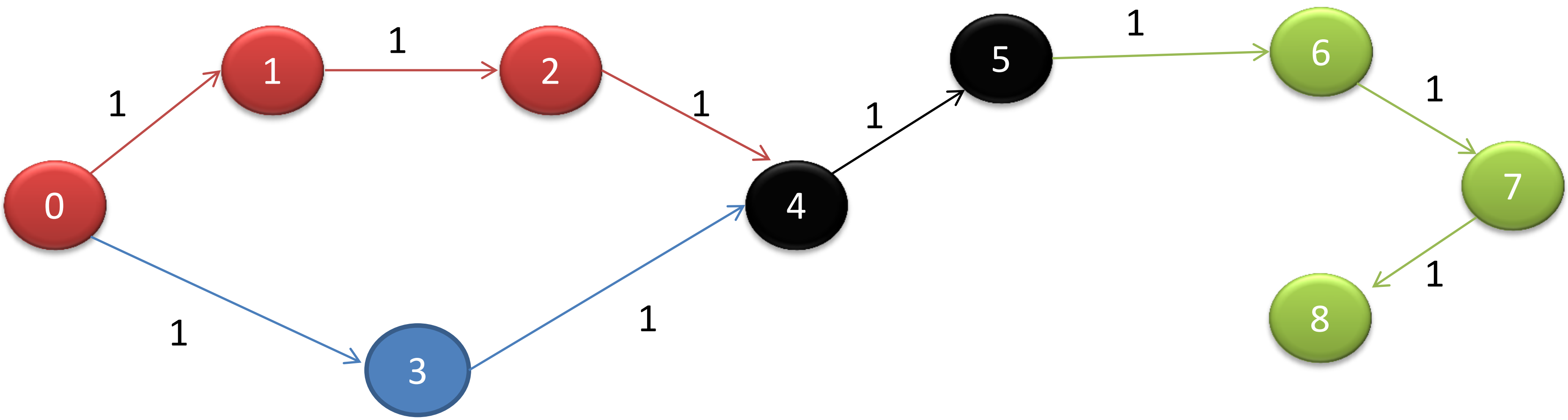
- **Iteration 4:**



The node chosen to be expanded from the open list in FORWARD direction: 4
The node chosen to be expanded from the open list in BACKWARD direction: 5

Bidirectional Search

- **Stop Condition Reached:**



- The search stops when the two searches meet at a common node.
- **The Algorithm takes only 4 iterations to converge.**
- **However the path found is NOT an optimal path.**
- **Cost of discovered path = 7**
- **Cost of optimal path = 6 (via 0-3-4-5-6-7-8)**

Missionaries and Cannibals

- Constraints
  - The boat can carry at most 2 people
  - On no bank should the cannibals outnumber the missionaries

- State : *<#M, #C, P>*
  - *#M* = Number of missionaries on bank *L*
  - *#C* = Number of cannibals on bank *L*
  - *P* = Position of the boat

- *Start State = <3, 3, L>*
- *Goal State = < 0, 0, R >*
- Operations
  - *M2* = Two missionaries take boat
  - *M1* = One missionary takes boat
  - *C2* = Two cannibals take boat
  - *C1* = One cannibal takes boat
  - MC = One missionary and one cannibal takes boat

Missionaries and Cannibals

- **Heuristic, h(n) = 2n-1**
  - **where n=number of people on left bank**
  - **if n = 0 , h = 0 as we have reached the goal state**

- Nodes chosen to be expanded from the open list: 0 1 3 2 4 5 6 7 8 9 10 11 12
- Optimal Path discovered:

  0 1 4 5 6 7 8 9 10 11 12 14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 13
- Number of parent pointer redirections = 0
- Note that this heuristic is Neither Admissible Nor Monotone
- Consider state S = <2,2,L>
  - h*(S) = 5 (optimal path : <2,2,L> - <0,2,R> - <0,3,L> - <0,1,R> - <0,2,L> - <0,0,R>)
  - h(S) = 7 (2*4-1 = 7) Hence Not Admissible
- Also for P = <2,2,L> C = <0,2,L>
  - h(P) = 7 h(C) = 3
  - C(P,C) = 1 Hence Not Monotone

Missionaries and Cannibals

- **Heuristic, h(n) = 2n+1**
  - **where n=number of people on left bank**
  - **if n = 0 , h = 0 as we have reached the goal state**

- Nodes chosen to be expanded from the open list: 0 1 3 2 4 5 6 7 8 9 10 11 12
- Optimal Path discovered:

  0 1 4 5 6 7 8 9 10 11 12 14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 13
- Number of parent pointer redirections = 0
- Note that this heuristic is Neither Admissible Nor Monotone
- Consider state S = <2,2,L>
  - h*(S) = 5 (optimal path : <2,2,L> - <0,2,R> - <0,3,L> - <0,1,R> - <0,2,L> - <0,0,R>)
  - h(S) = 9 (2*4+1 = 9) Hence Not Admissible
- Also for P = <2,2,L> C = <0,2,L>
  - h(P) = 9 h(C) = 5
  - C(P,C) = 1 Hence Not Monotone

Missionaries and Cannibals

- **Heuristic, h(n) = n-1**
  - **where n=number of people on left bank**
  - **if n = 0 , h = 0 as we have reached the goal state**

- Nodes chosen to be expanded from the open list: 0 1 3 2 4 5 6 7 8 9 10 11 12 13
- Optimal Path discovered:

  0 1 4 5 6 7 8 9 10 11 12 14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 13
- Number of parent pointer redirections = 0
- Note that this heuristic is Admissible but Not Monotone
- Consider P = <2,2,L> C = <0,2,L>
  - h(P) = 3 h(C) = 1
  - C(P,C) = 1 Hence Not Monotone

Missionaries and Cannibals

- **Heuristic, h(n) = n/2**
  - **where n=number of people on left bank**
  - **if n = 0 , h = 0 as we have reached the goal state**
- Nodes chosen to be expanded from the open list: 0 3 4 1 2 5 6 7 8 9 10 11 12 13
- Optimal Path discovered:

  0   3   4   5   6   7   8   9   10   11   12   14
- Optimal Path cost = 11
- Number of iterations taken by A* Algorithm = 14
- Number of parent pointer redirections = 0
- This heuristic is Admissible as well as Monotone.

8-Puzzle Problem

2		3
1	8	5
4	7	6

Start State

1	2	3
4	5	6
7	8	

Goal State

- Tile movement represented as the movement of the blank space.
- Operators:
  - L : Blank moves left
  - R : Blank moves right
  - U : Blank moves up
  - D : Blank moves down
  - *C(L) = C(R) = C(U) = C(D) = 1*

8-Puzzle Problem

- Heuristic, h(n) = sum of Manhattan distances of tiles from their destined position

  h(n) = X-dist + Y-dist from Goal State

- The optimal path is:0-1- 4-10-22-39-69-119
- Optimal path cost is 7
- Number of nodes expanded from Open List =7
- Number of Parent Pointer Redirections =0

8-Puzzle Problem

- Heuristic, h(n) = no. of tiles displaced from their destined position.
- The optimal path is:0 1 4 10 22 39 69 119
- Optimal path cost is 7
- Number of nodes expanded from Open List = 8
- Number of Parent Pointer Redirections =0

- Here also we see that ***Manhattan Distance***, which is a better heuristic than ***No. of Displaced Tiles*** heuristic performs better(converges faster).

8-Puzzle Problem(our own heuristic)

- Heuristic, h(n) = Manhattan distances of only the blank tile
- The optimal path is:0-1- 4-10-22-39-69-119
- Optimal path cost is 7
- Number of nodes expanded from Open List =**53**
- Number of Parent Pointer Redirections =0

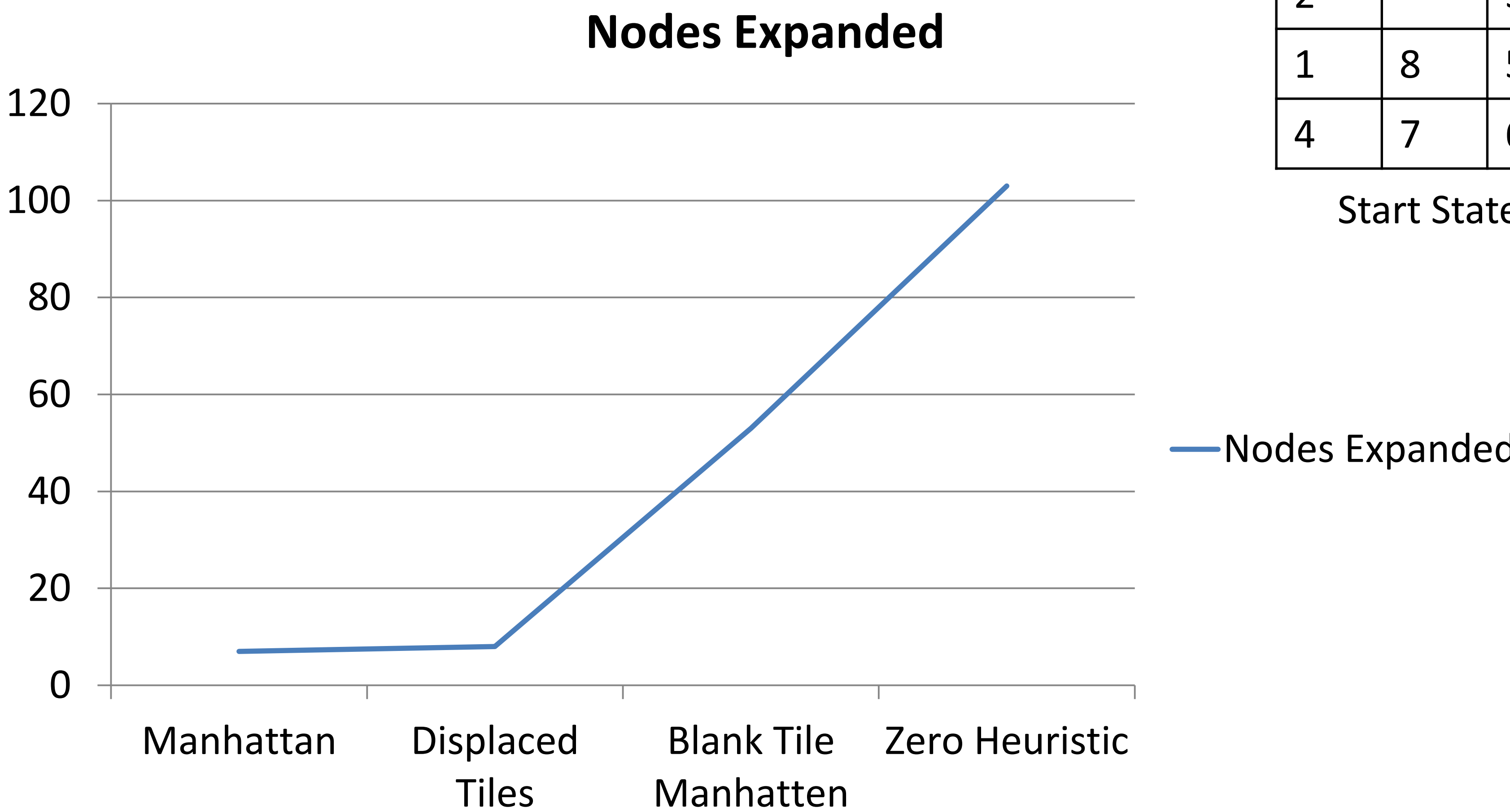
- Now we observe that this heuristic expands a lot higher number of nodes from the Open List as this is a very poor heuristic.

8-Puzzle Problem(our own heuristic)

- Heuristic, h(n) = 0 for all nodes
- The optimal path is:0-1- 4-10-22-39-69-119
- Optimal path cost is 7
- Number of nodes expanded from Open List =103
- Number of Parent Pointer Redirections =0

- Now we observe that this heuristic expands even a lot higher number of nodes from the Open List as this is a very poor heuristic.

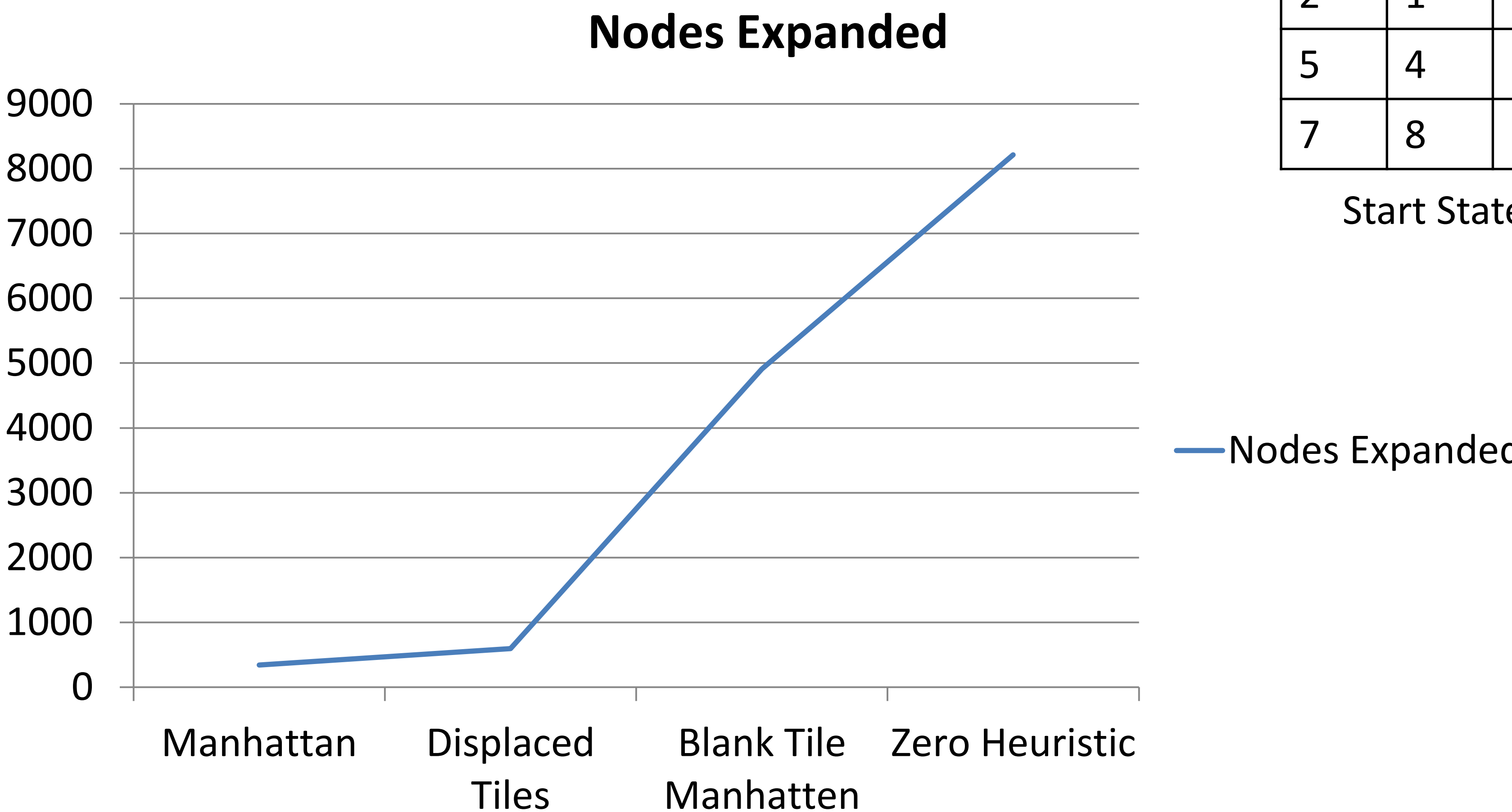
Comparing Heuristics (8-Puzzle)



2		3
1	8	5
4	7	6

Start State

Comparing Heuristics (8-Puzzle)



2	1	3
5	4	6
7	8	

Start State

8-Puzzle Non Reachability

Start State

2	1	3
5	4	7
6	8	

Goal State

1	2	3
4	5	6
7	8	

- We now have a start state from which we can not reach to the goal state.
- However, we can figure the non-reachability before running the A* Algorithm by using the following strategy:

I. Write the start state puzzle in row major form i.e. in above case start state can be written as [2 1 3 5 4 7 6 8].

II. Count the no. of inversions in this array.

III. Repeat the above steps (I) & (II) for goal state. Since Goal State is fixed, row major form is [1 2 3 4 5 6 7 8] & its no of inversions are zero.

IV. Now we Use the following rule:

"Start State with even no of inversions can reach a Goal State with even no. of inversions and Start State with odd no of inversions can reach a Goal State with odd no. of inversions"

V. Since our goal state contains 0 (even) inversions, so **our start state must have even no. of inversions**.

Automatic Theorem Prover

CS386-Assignment

Sahil Jindal	Abhishek Gupta	Rohan Gyani	Mridul Ravi Jain
110020043	110040067	110040001	110040083

Assignment Specifications

- You will have to create an automatic theorem prover for propositional logic.
- The input is any well formed formula in PL.
- The output is yes/no depending on the formula being a theorem or not
- The proof must be SYNTACTIC. You cannot use a truth table.
- Go from 1st principles OR use Deduction theorem
- After outputting the result, you have to DISPLAY the proof path
- You can take human help if stuck in between in the proof. For example, you can ask for a hint as to which axiom will be needed.

Theorem Input Format

- Elements are *propositions* : Capital letters
- Operator is only one : $\rightarrow$ (called implies)
- Special symbol ***f*** (called 'false')
- Two other symbols : '**(**' and '**)**'
- Well formed formula is constructed according to the grammar

  $$WFF \rightarrow P \mid f \mid WFF \rightarrow WFF$$

- Inference rules:
- Modes Ponens
- Axioms:
  - A1: $(A \rightarrow (B \rightarrow A))$
  - A2: $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)))$
  - A3: $(((A \rightarrow f) \rightarrow f) \rightarrow A)$

Techniques Applied

- Every theorem to be proved is brought down to deriving WFF1, WFF2, WFF3….. $\vdash \boldsymbol{f}$

  by applying Deduction Theorem

- Put all existing hypothesis in the proof vector

- All subsequent statements are pushed into this proof vector.

Techniques Applied

- Then in a loop continuously check if one of the following conditions can be applied until proof is reached:
  - Check if Modem Ponens can be applied on any two quantities in the proof vector
  - If there are 2 statements S1 & S2 such that
    - S2: (L→ S1) → R
    - Then apply Axiom1 with **A** as S1 and **B** as L
  - If the LHS of any hypothesis is of the form A → (B →A), then apply Axiom1 on it
  - If any statement is of the form A → (B→C) apply Axiom2
  - If any statement is of the form ((A → *f*)→*f)* apply Axiom3
  - If LHS of any statement is of the form (((A → *f*)→*f*) → A) apply Axiom3
  - If none of the above conditions can be applied, then apply Brute Force and finally ask for Human Help

Brute Force

- Pick all elements from the proof vector of the form:
  - P
  - $(P \rightarrow Q)$
- Construct a vector 'X' of these elements
- Generate statements by plugging in the elements from Vector 'X' in placeholders in Axioms (using all permutations). Put these statements in another vector 'Y'.
- For every statement in vector 'Y' and proof vector, check if Modus Ponen can be applied.
- If MP can be applied, put the corresponding statement from Y and the result of MP in the proof vector.

Human Help

- Human help can be provided in the form of:
  - Applying transitivity:
    $(A \rightarrow B)$ and $(B \rightarrow C) \Rightarrow (A \rightarrow C)$
  - Applying contraposition:
    $(A \rightarrow B) \Rightarrow (\sim B \rightarrow \sim A)$
  - Apply De Morgan's 1st law:
    $(\sim(P \wedge Q)) \Rightarrow (\sim P \vee \sim Q)$
  - Apply De Morgan's 2nd law:
    $(\sim(P \vee Q)) \Rightarrow (\sim P \wedge \sim Q)$
  - Apply:
    $f \rightarrow P$
  - Apply any user specified statement(if provable)
  - Apply axioms
  - Continue.

Final Result

- Whenever applying any standard result using Human help, we also print the corresponding proof.

- In the end, we print a final proof that consists of only the relevant statements.

Circuit Verification

Methods & Explanation

Assignment Specifications

- A circuit is defined by its input, output, gates and connectivity.
- There is circuit specific knowledge and circuit independent knowledge.
- Circuit independent knowledge is properties of gates (AND, OR, NOT etc) and the meaning of connectivity. You should read all this knowledge into the PROLOG memory.
- However, keep the circuit independent knowledge memory resident.
- The final goal is to verify the input and output in each row of the truth table

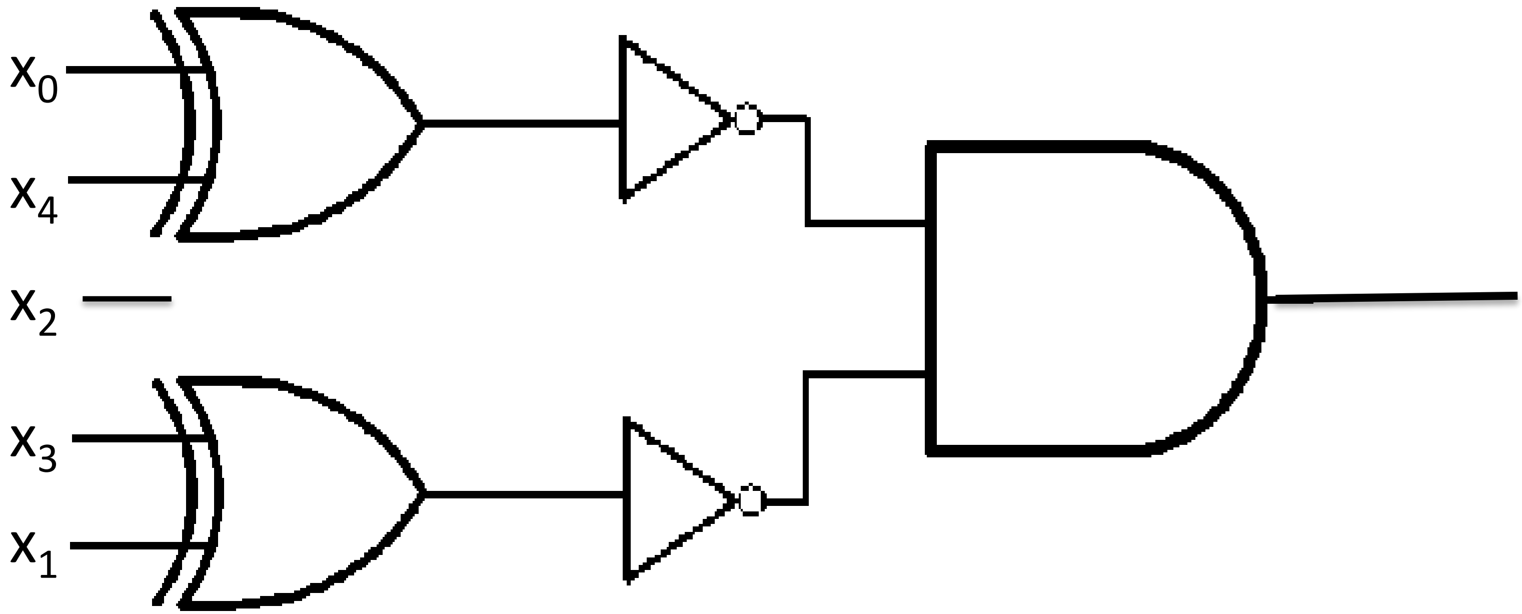
Rules Defined

- type(X, a).
  - a: 'AND', 'OR', 'NOT','XOR'
  - Represents type of gate X
- no_of_inputs(X, val).
  - val : 1, 2, ...
  - Gate X has val inputs
- count_1s(X , num_inputs, val, acc).
  - Counts number of 1s as input to gate X
  - num_inputs: input index to gate X
  - acc: accumulator variable(initialised to zero)
  - val: final result is stored in this variable
- signal(x_n,a).
  - Signal at x_n is a (x_n is input to circuit)

Rules Defined

- in(n,X, val).
  - nth input to gate ‘X’ is ‘val’
- in(n,X).
  - nth input to gate ‘X’
- connected(x1, x2).
  - Used to specify connections in the circuit
  - Eg: connected(x1,in(1,a1))
    - x1 is connected to 1st input of gate a1
- out(X,Val).
  - Output of gate ‘X’ is stored in ‘Val’
- out(X).
  - Output of gate ‘X’

5-input Palindrome Circuit



Output= $\overline{(x_0 \oplus x_4)} \cdot \overline{(x_1 \oplus x_3)}$

5-input Weighted Majority Circuit