Markovian Decision Processes and Reinforcement Learning

Rohit Kelkar {ryk@andrew.cmu.edu}, Vivek Mehta {vivekm@andrew.cmu.edu}

**Abstract - This paper presents the results of analysis of the performance of reinforcement learning algorithms mainly Value iteration, Policy iteration, Q-learning and Prioritized sweeping. It introduces the adaptive learning rate scheme as an enhancement to the existing Q learning algorithm for non stationary environments. The paper also analyzes the performance of prioritized sweeping based on the number of backups.**

**Keywords: ε-greedy policy, value iteration, policy iteration, Q learning, learning rate, prioritized sweeping, backups**

# Reinforcement Learning

## *The Reinforcement Learning Model*

The reinforcement learning problem [1] consists of an agent interacting with the environment. As shown in Figure 1 at each instance the agent receives from the environment some representation of the environment’s current state st Є S, where S represents the state space. Based on the state the agent selects an action a Є A(st), where A(st) is the set of all possible actions available in state st. As a consequence of this action, at the next time step, the agent transitions to a new state st+1 and receives a scalar reward rt+1 Є R. The action in each state is dictated by a policy π which maps each state to an action. The objective of the agent is to find the optimal policy π\* which optimizes some measure of the rewards.



Figure 1: Agent-Environment Interaction in Reinforcement Learning

## *The Markovian property*

The agent makes its decisions as a function of a signal from the environment which notifies the agent about the environment’s current state. The environment’s state at any instant may or may not be a function of all the past states that the agent has been through. If the environments state at time t+1, st+1 can be modeled as a function of the current state st and the current action at then the environment is said to possess the Markovian property [1]. The environment’s state at time t+1 can be defined by specifying only P {st+1 = s’, rt+1 = r | st, at}. If the environment has Markov property then its one step dynamics allow us to predict the next state and expected reward given the knowledge of current state and the action taken in the current state. The predictions made on the basis of the Markov property are as good as the predictions made with the knowledge of the complete history up to that time.

## *Model free and Model based Reinforcement Learning*

Model free algorithms do not assume or build a model of the environment. Example Q learning [2]. The model based algorithms create a model of the environment based on the interactions of the agent with the environment. Example prioritized sweeping [3].

# The Experimental Setup Review Stage

As can be seen in Figure 2 the experimental setup consists of an agent moving in a discrete state space represented by a maze where each state is represented by a cell in the maze. The maze contains terminal states represented by goal states and obstacles represented by walls. The maze is bounded on all sides by walls. If the agent tries to transition from one state to another and hits a wall instead then the agent is penalized and stays in the same state. There is a path cost associated with every transition that the agent makes from one state to another. The aim of the agent is to find that path to the goal state which has least cost associated with it.

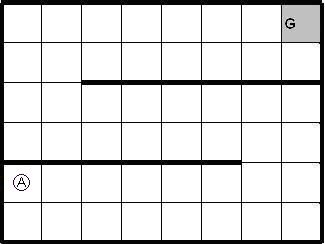


Figure 2: Snapshot of the test bed

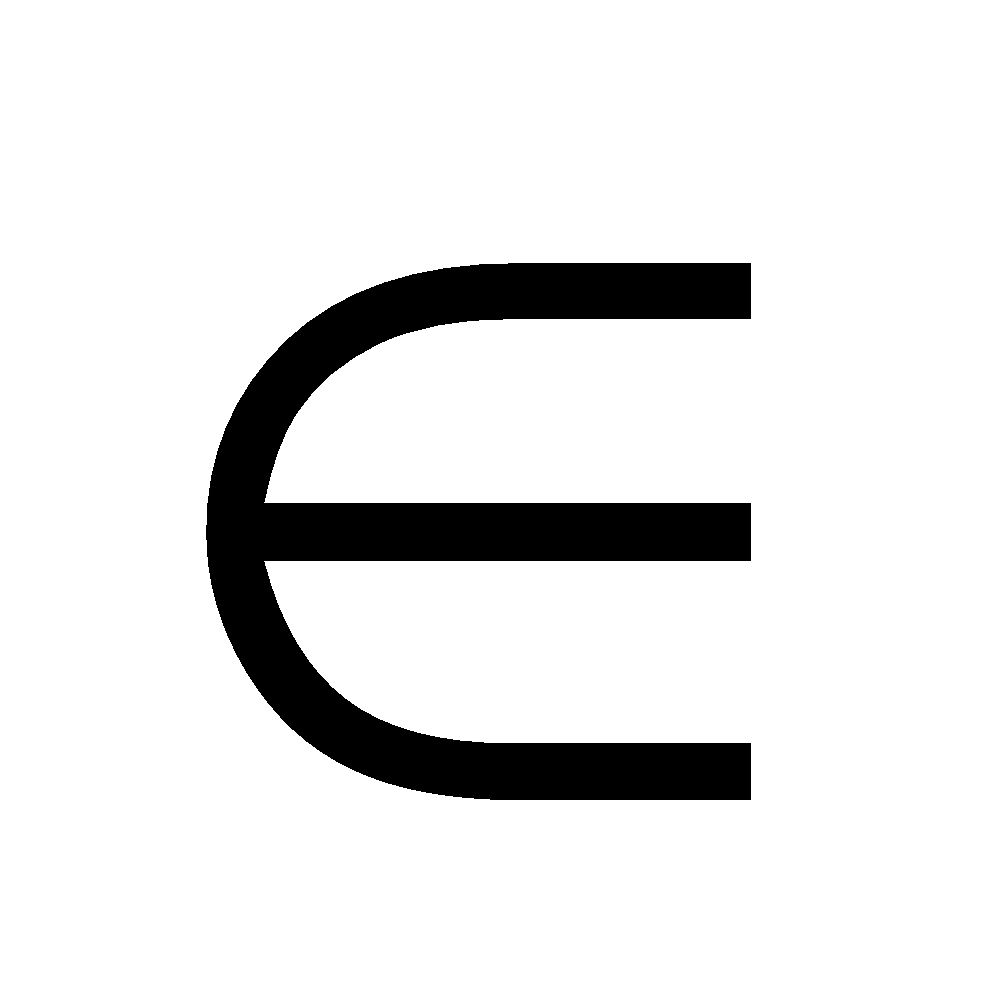
To model the noise in the environment a parameter named ‘pjog’ is used. Each state has a finite number of successors, N. If in a particular state s the agent decides to perform action a then the agent will end up in the valid successor of s with a probability equal to (1-pjog) and end up in any one of the N-1 successors of that state with a probability equal to pjog/ (N-1).

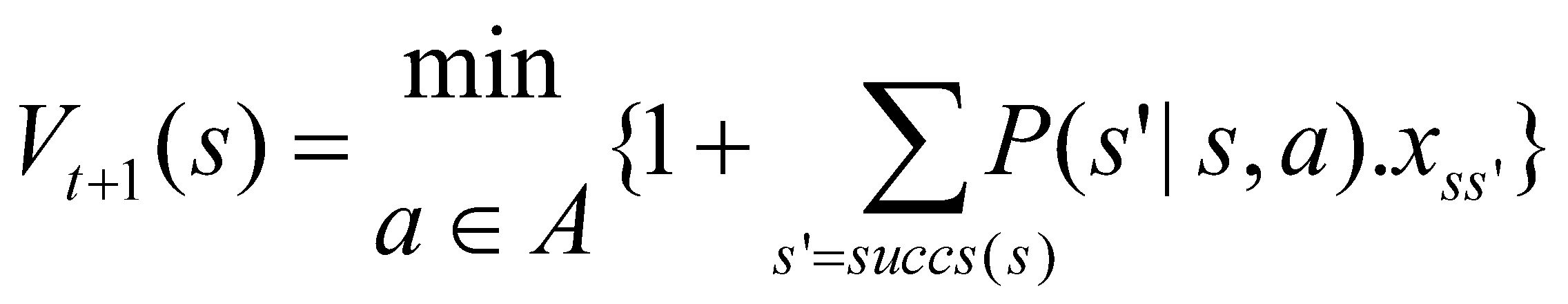
For Q learning and prioritized sweeping another parameter, called ε, is used. This is specifically to implement the ε-greedy policy [1]. Under this policy the agent decides to perform the best action with a probability of (1- ε) and performs any random action with a probability equal to ε/(N-1).

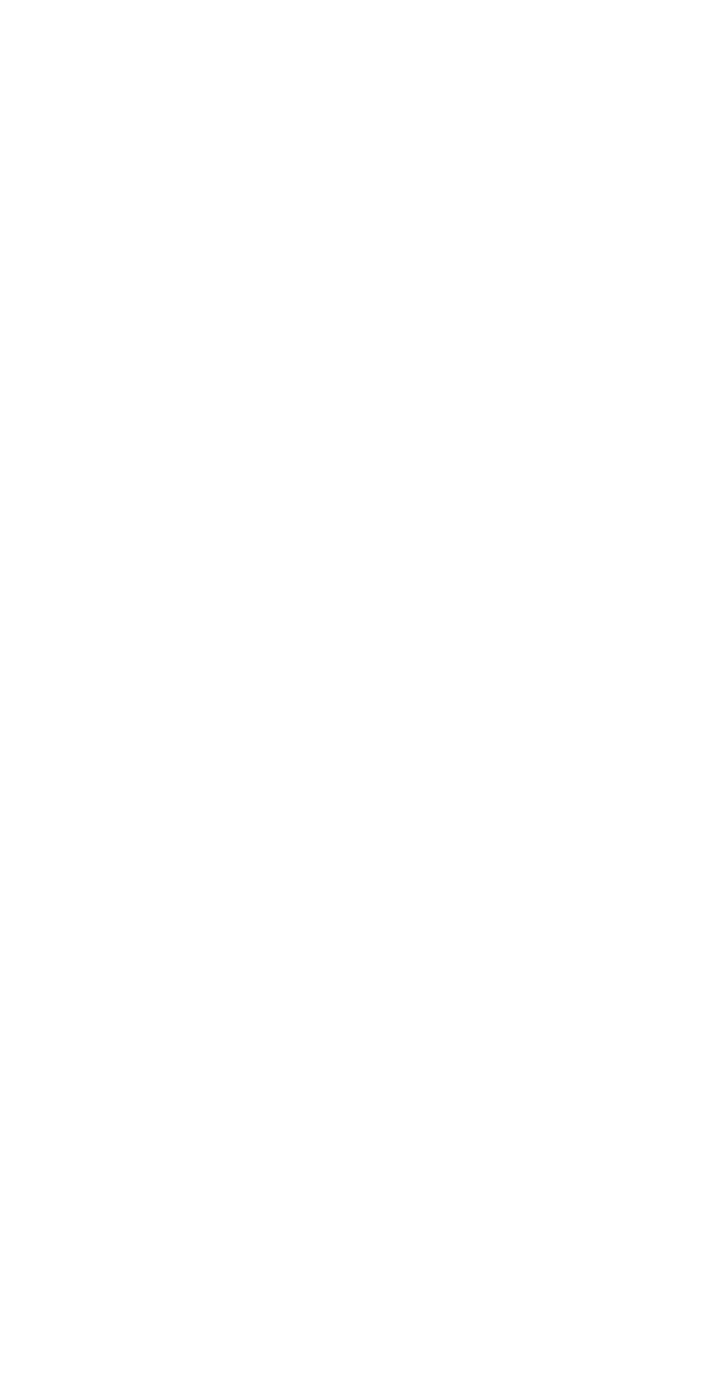
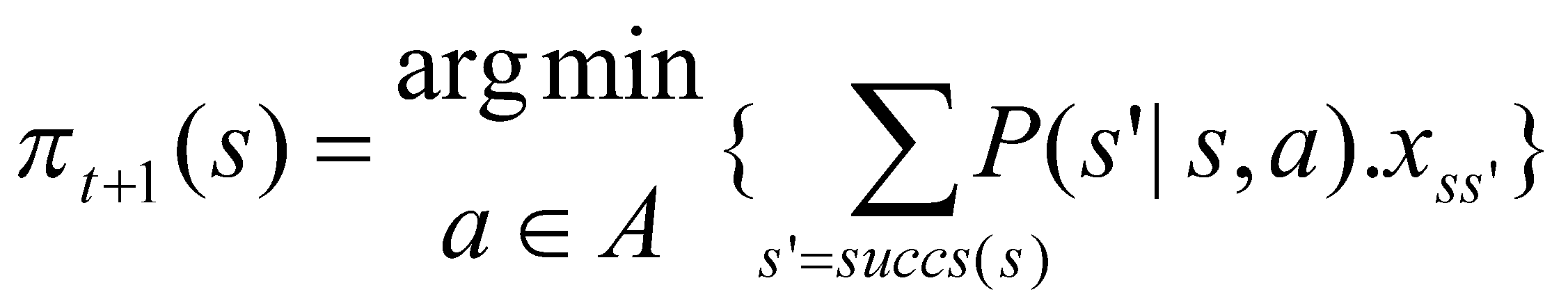
# Value Iteration [2] algorithm applied to the experimental setup

*Initialize V(s) arbitrarily*

*Loop until policy good enough*

*Loop for sS*

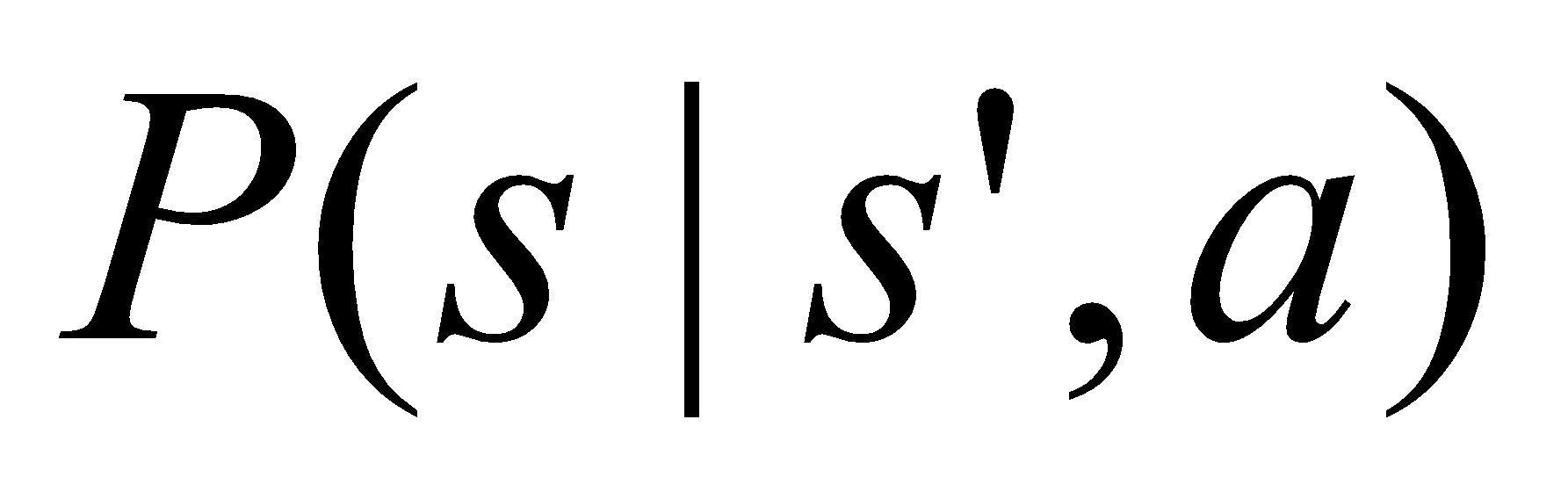


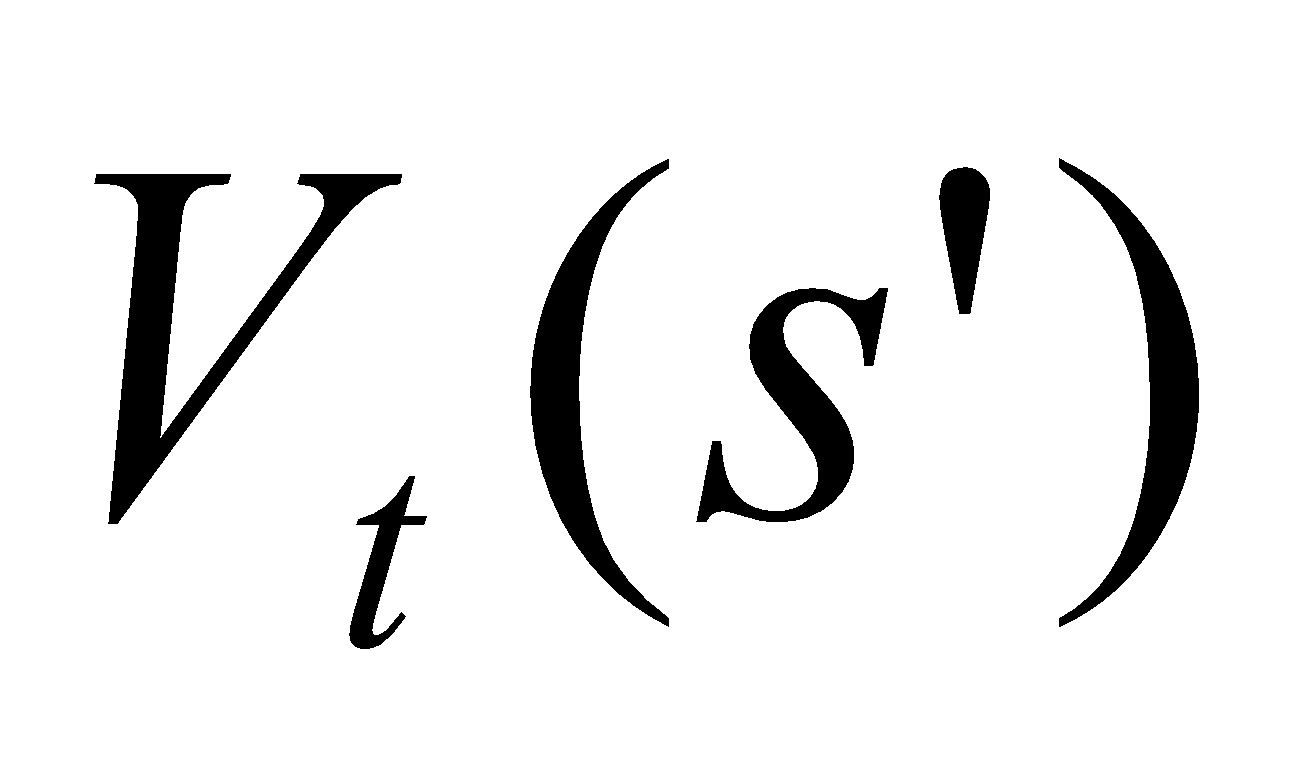


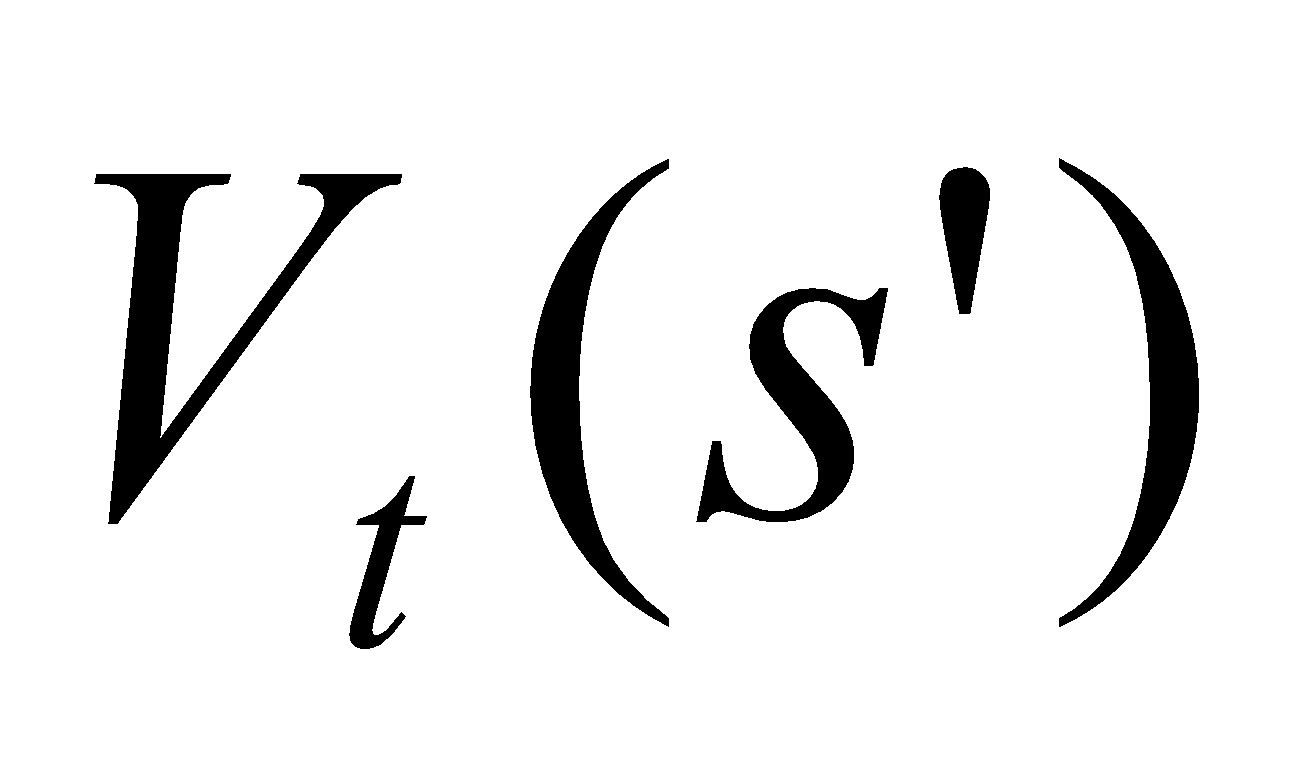
*End Loop*

*End Loop*

*Where,*

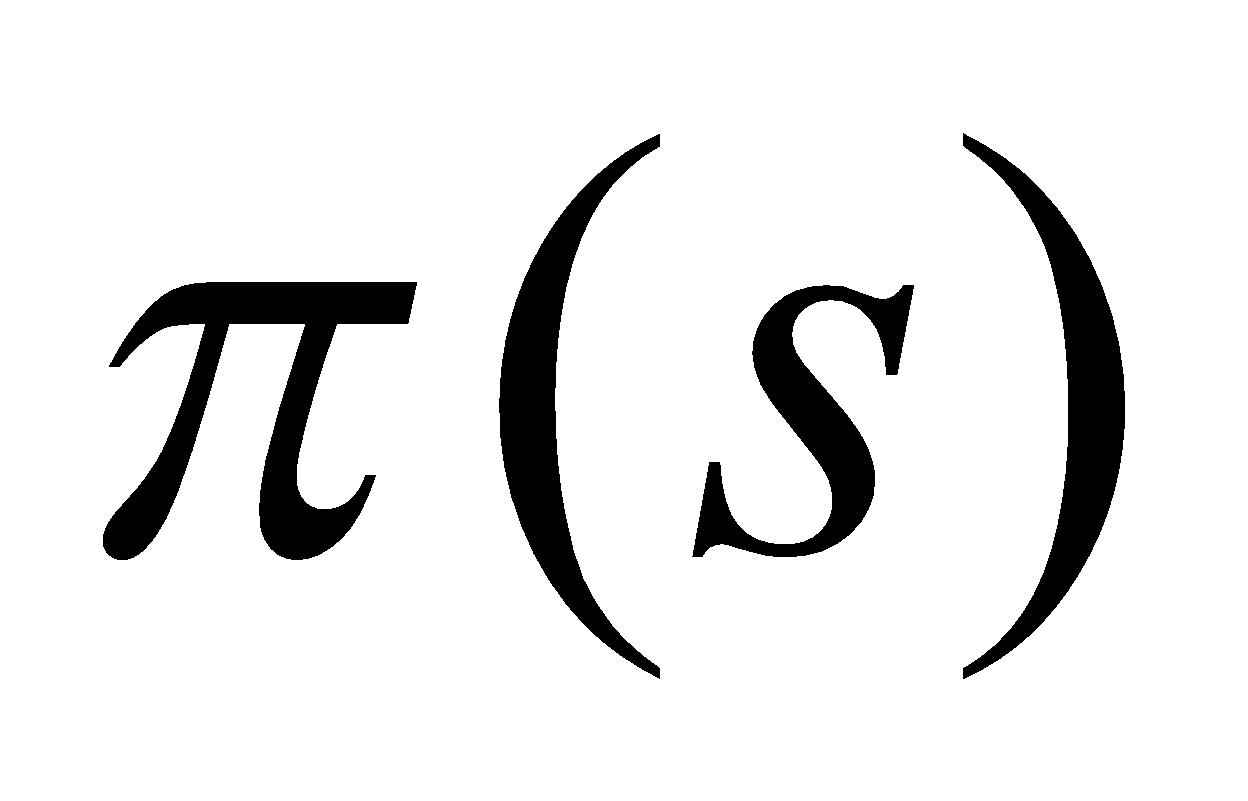
*= probability of transition from state s to state s’ after performing action a*

*xss’=, if transition from s to s’ is safe*

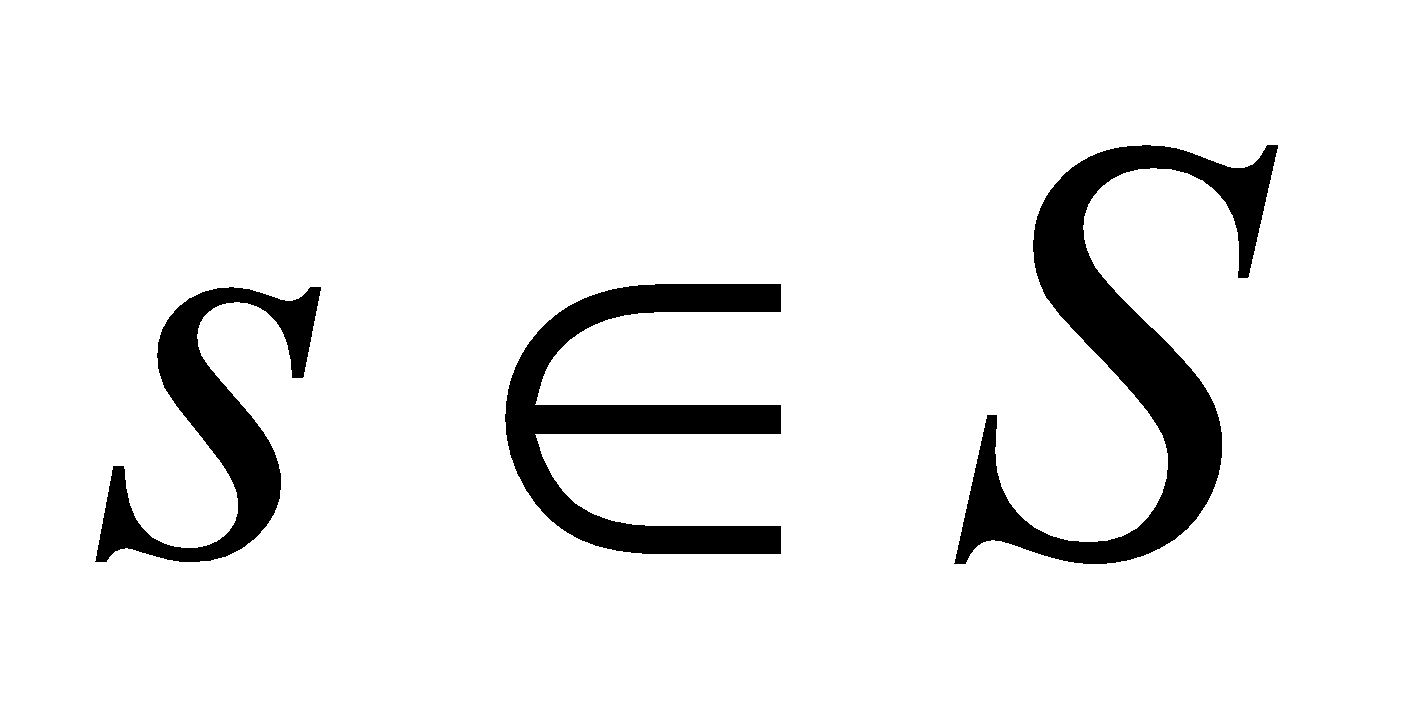
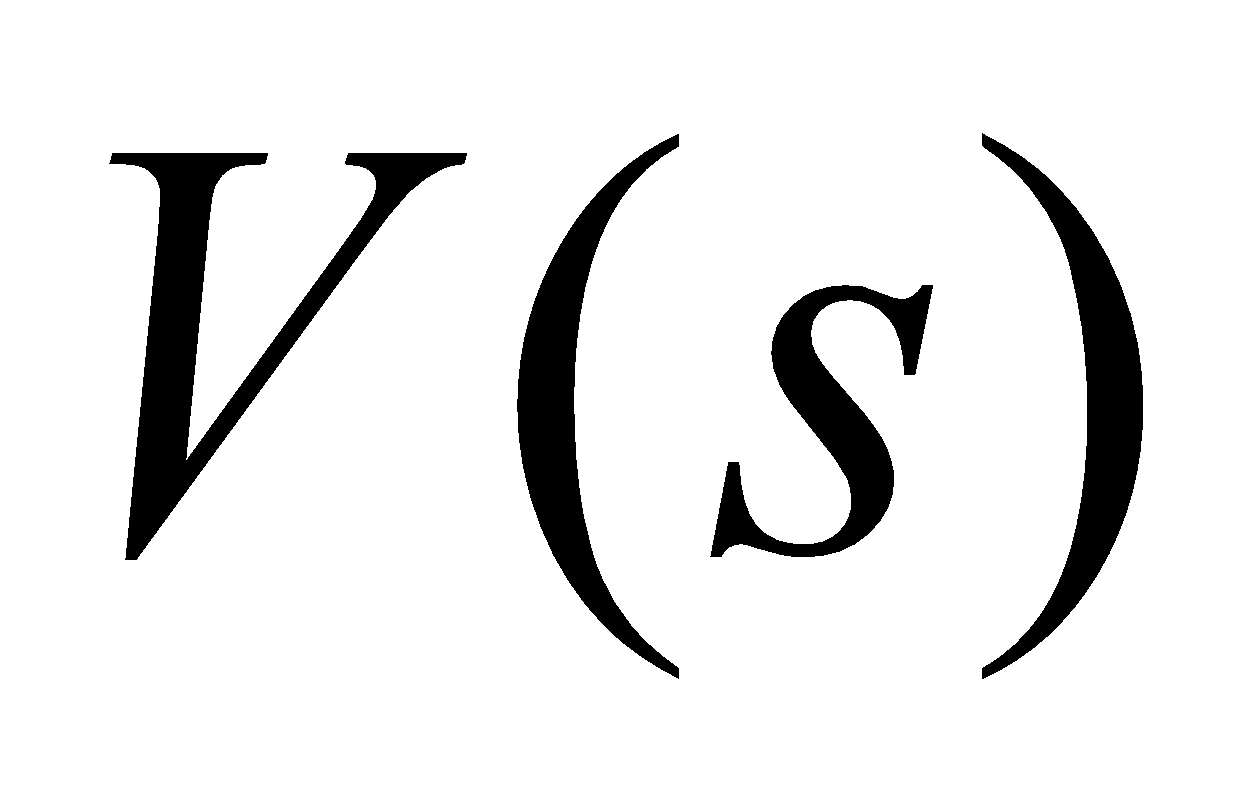
*= penalty+, if agent hits the wall while attempting to make the transition from s to s’*

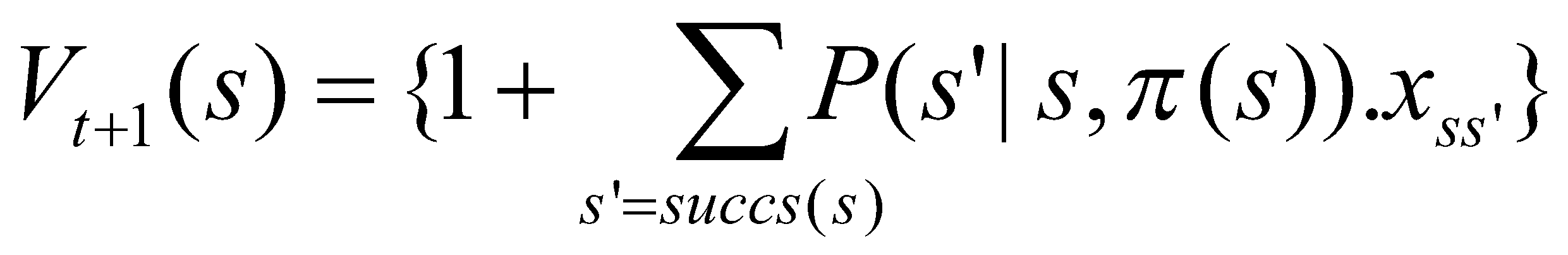
The algorithm terminates when it finds the optimal policy. A good measure of optimal policy is that the maximum of the difference in the values that each state experiences between successive iterations of the algorithm is less than a tiny preset threshold.

# Policy Iteration [2] algorithm applied to the experimental setup

*1.* ***Initialization:*** *Initialize  arbitrarily*

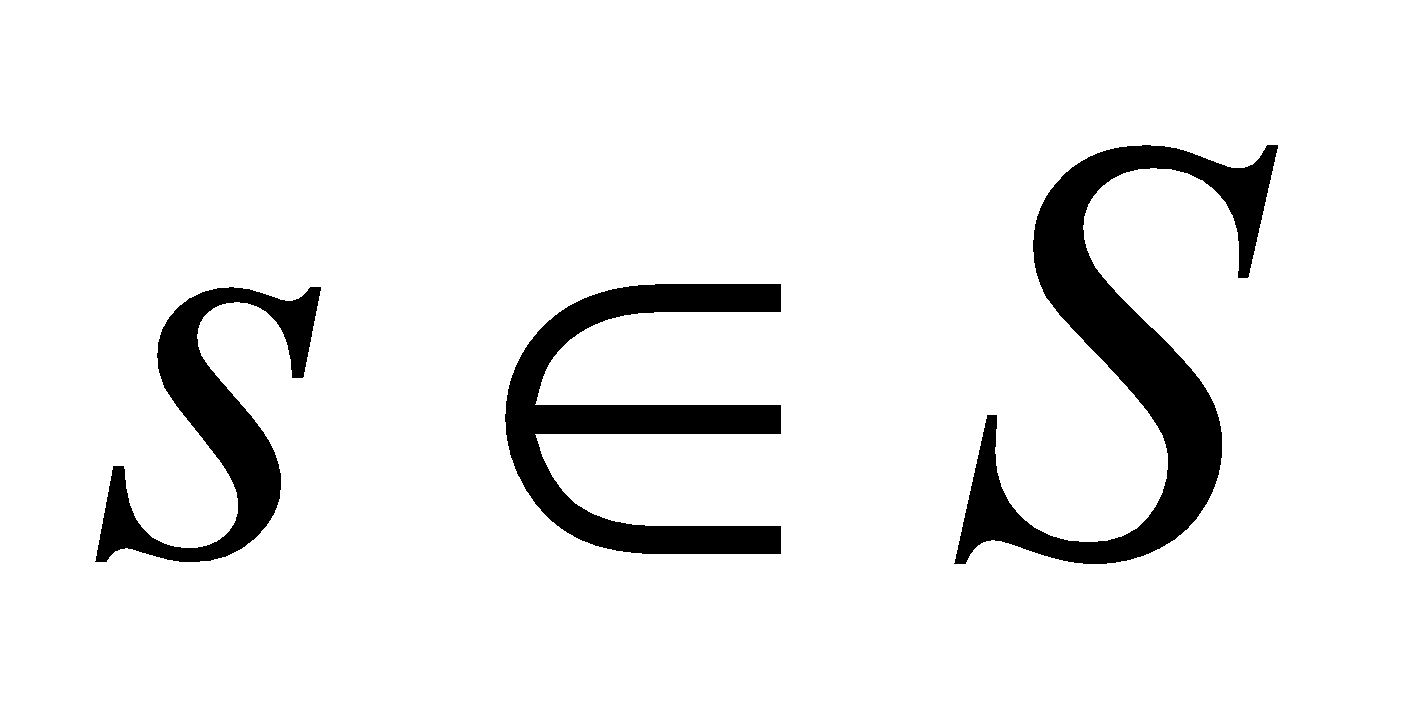
*2.* ***Policy Evaluation:***

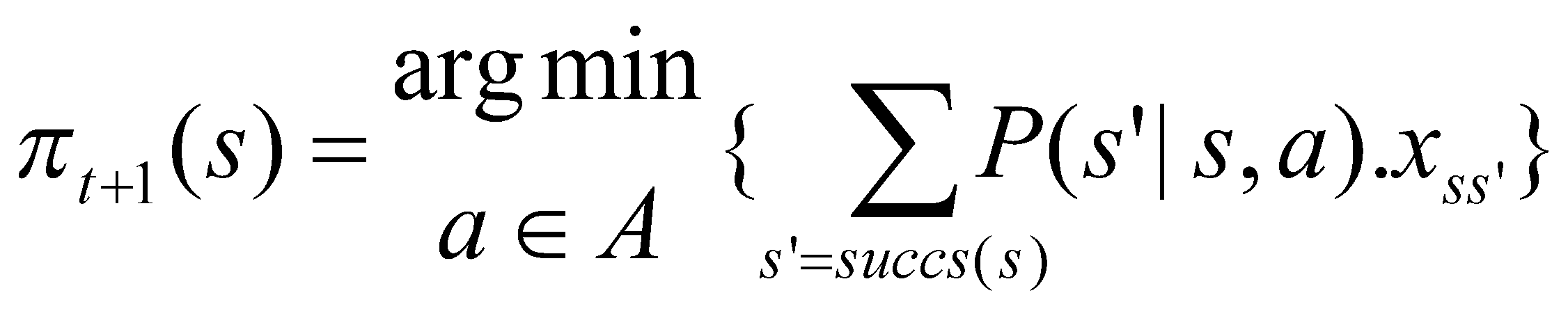
*Loop for until converged*

**

*End Loop*

*3.* ***Policy Improvement:***

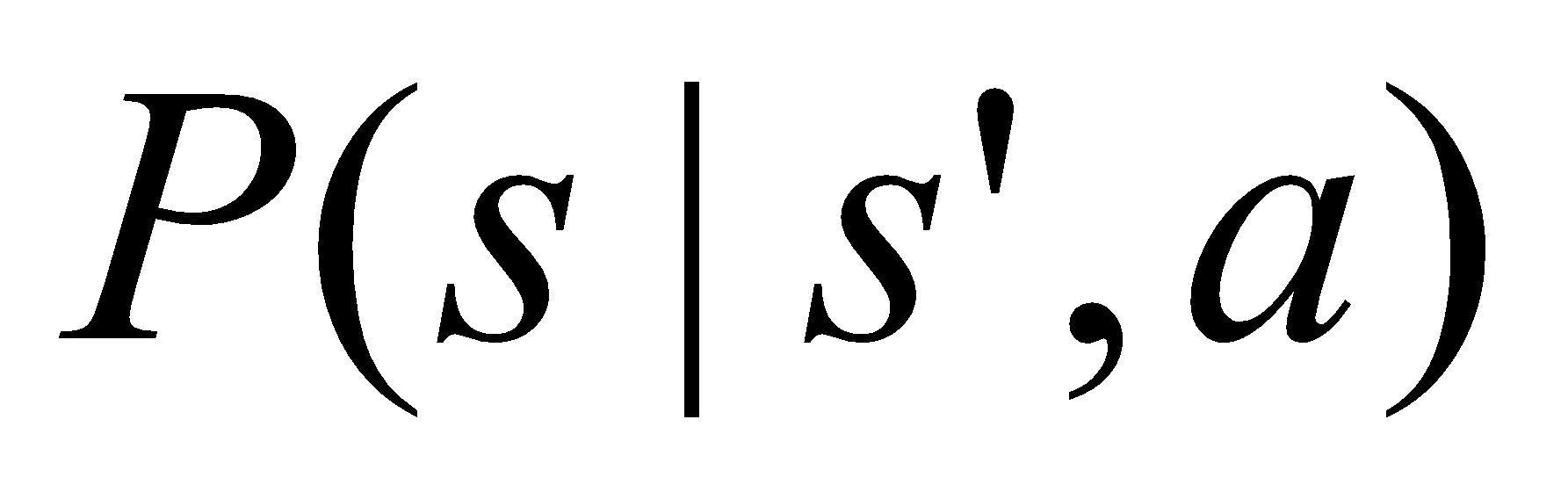
*Loop for *

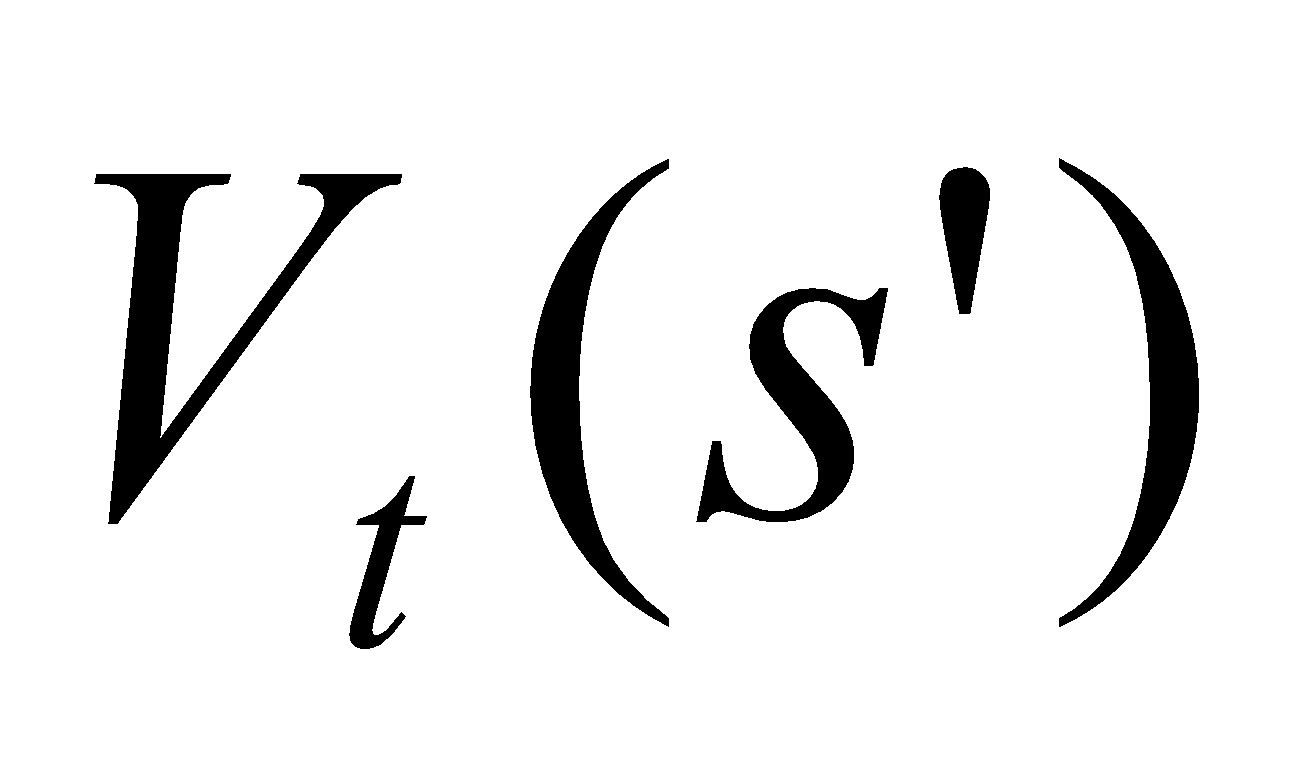
**

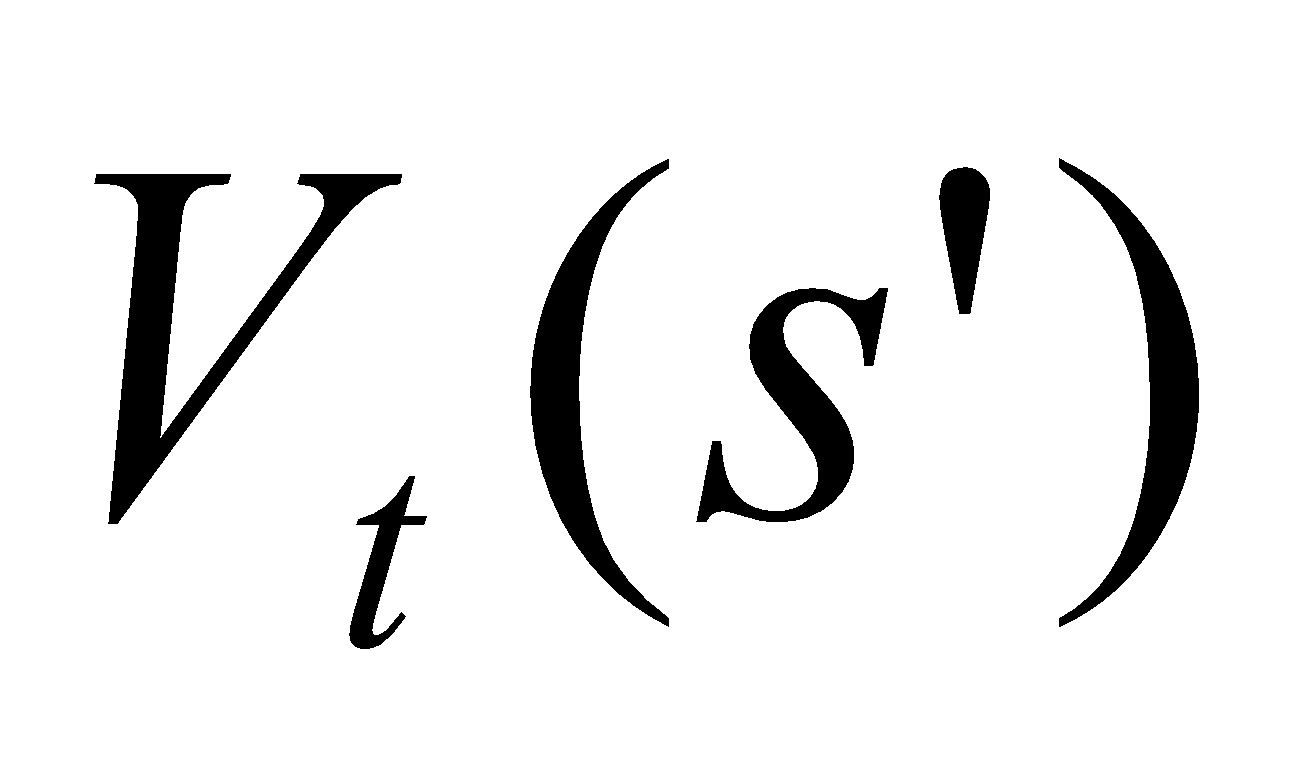
*End loop*

*If policy stable, stop else go to 2*

*Where,*

*= probability of transition from state s to state s’ after performing action a*

*xss’=, if transition from s to s’ is safe*

*= penalty+, if agent hits the wall while attempting to make the transition from s to s’*

Policy Iteration works in two steps. The first step evaluates the policy and the next step improves the policy. The policy is evaluated by finding the values associated with the policy. The values associated with the policy is the expected infinite discounted reward [2] that will be gained by each state in the state space by executing that policy. After evaluating the value of each state under the current policy, improvements are made to the policy, if possible, by changing the actions. The algorithm terminates when the policy of every state remains constant between successive iterations.

## *Comparison between value iteration and policy iteration*

Figures 3 compares policy iteration and value iteration algorithms with respect to the number of iterations required for convergence as the state space increases. Figure 4 compares the performance of value iteration and policy iteration with respect to time required for convergence as the state space increases. It can be observed that policy iteration requires lesser number of steps to converge. But policy iteration takes more time to converge as compared to value iteration. This is because policy iteration involves the policy evaluation process. The policy evaluation step computes the expected infinite discounted reward and hence consumes more time but results in reducing the action space under consideration. Thus policy iteration is ideal for problems where the action space is large because it considerably reduces the action space in lesser number of iterations. Value iteration is ideal for problems where the state space is large.

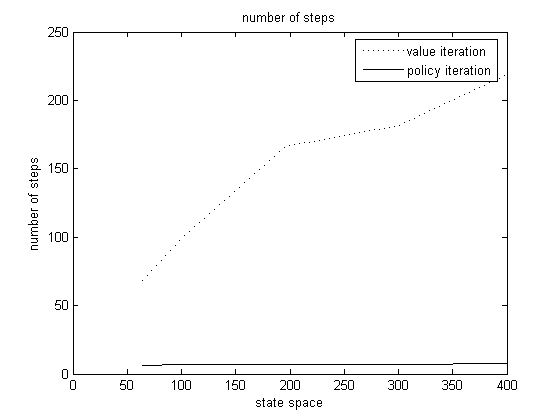


Figure 3: Comparison between the number of steps taken by value iteration and policy iteration as the state space increases.

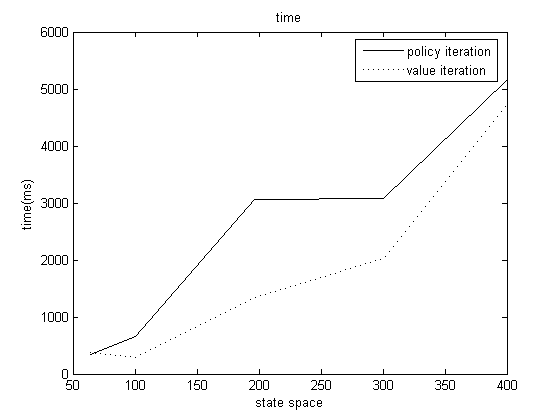
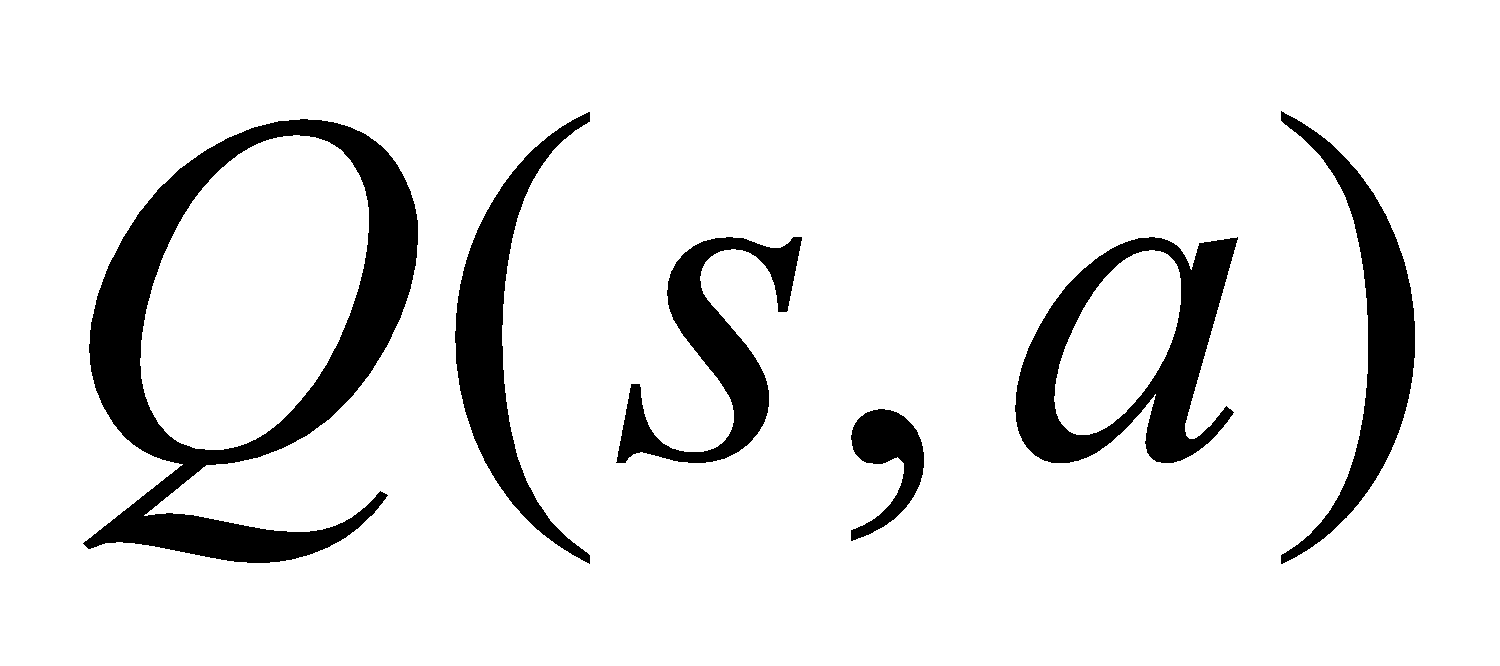
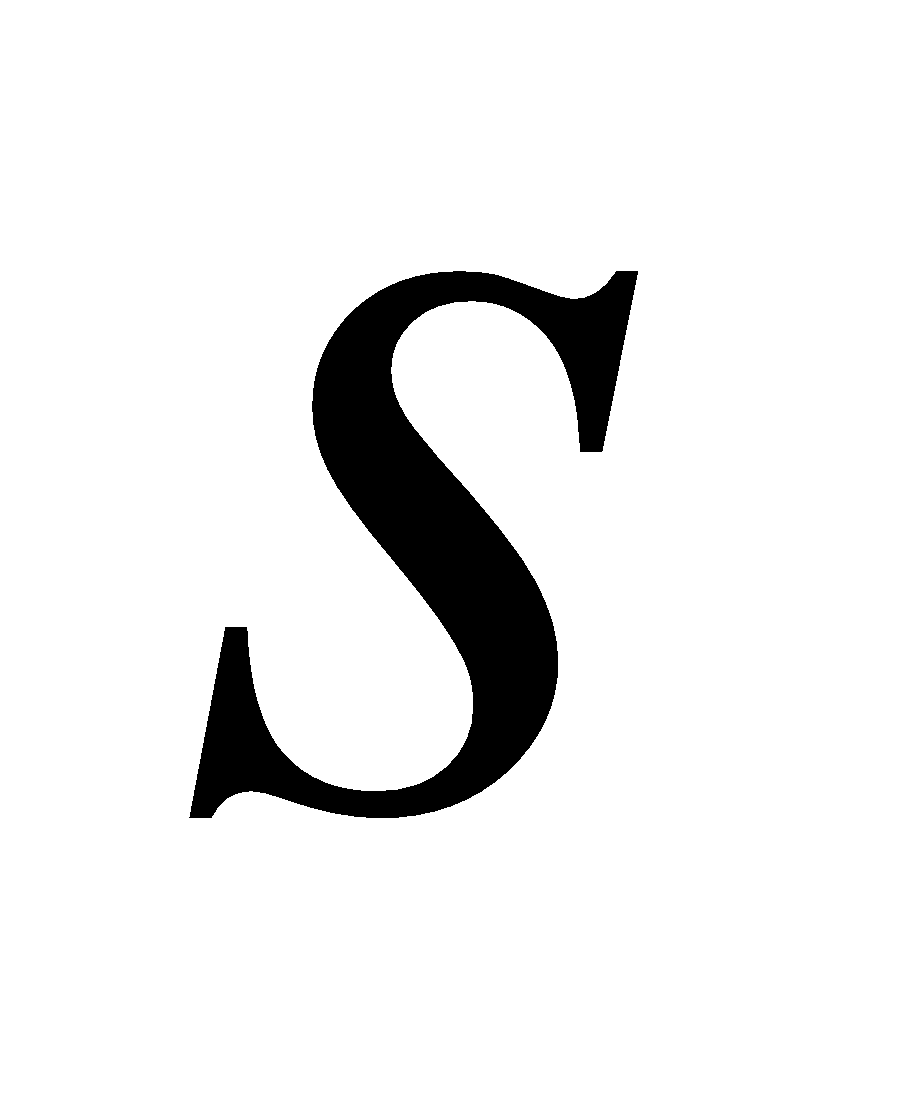


Figure 4: Comparison between the time taken by value iteration and policy iteration as the state space increases.

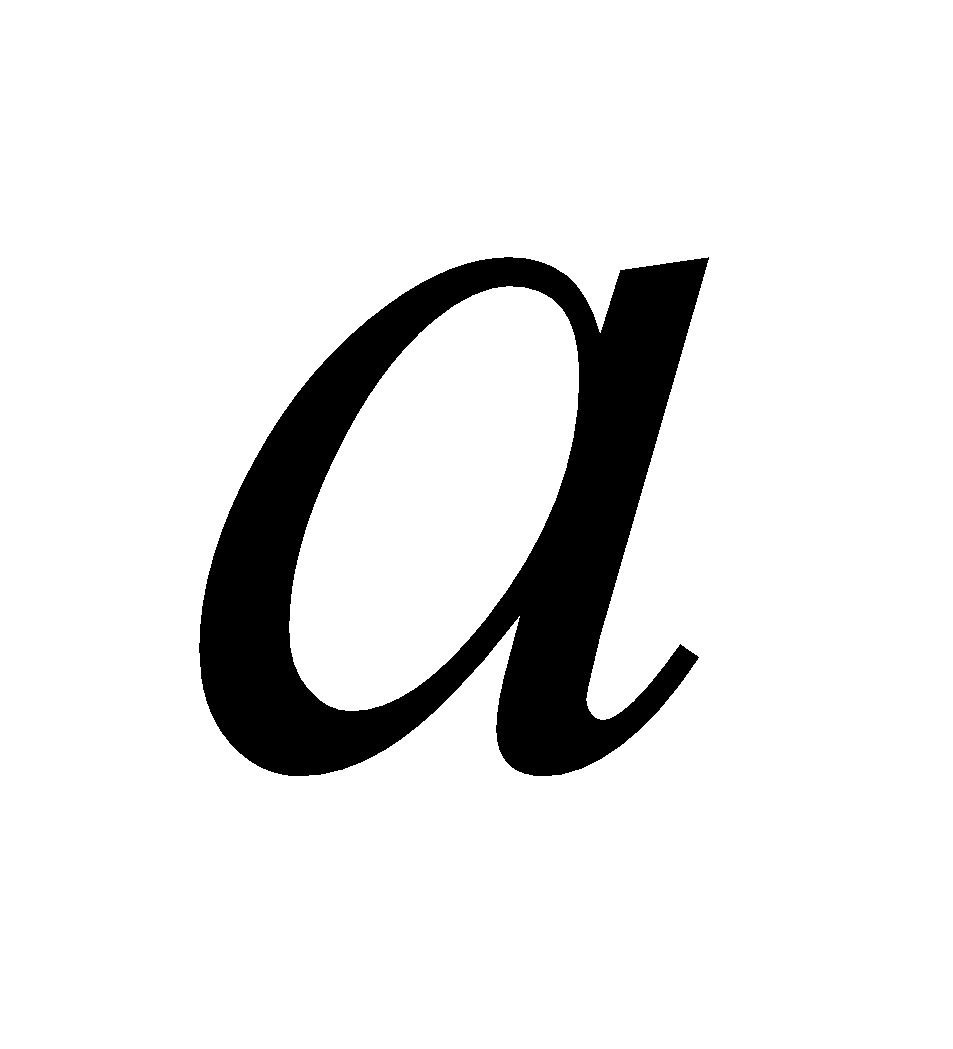
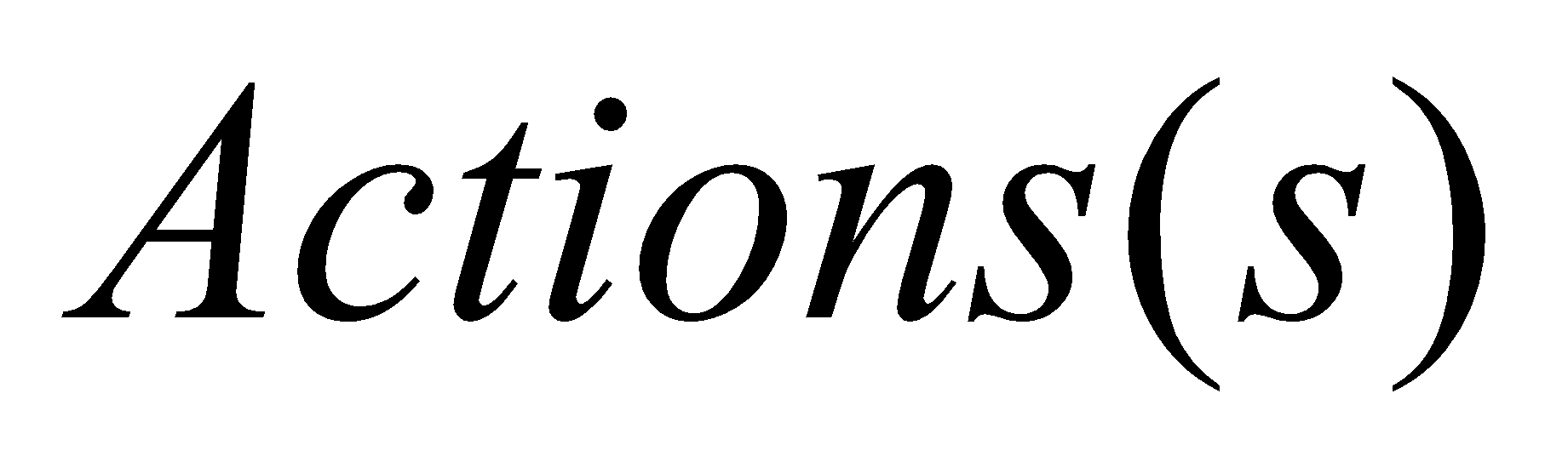
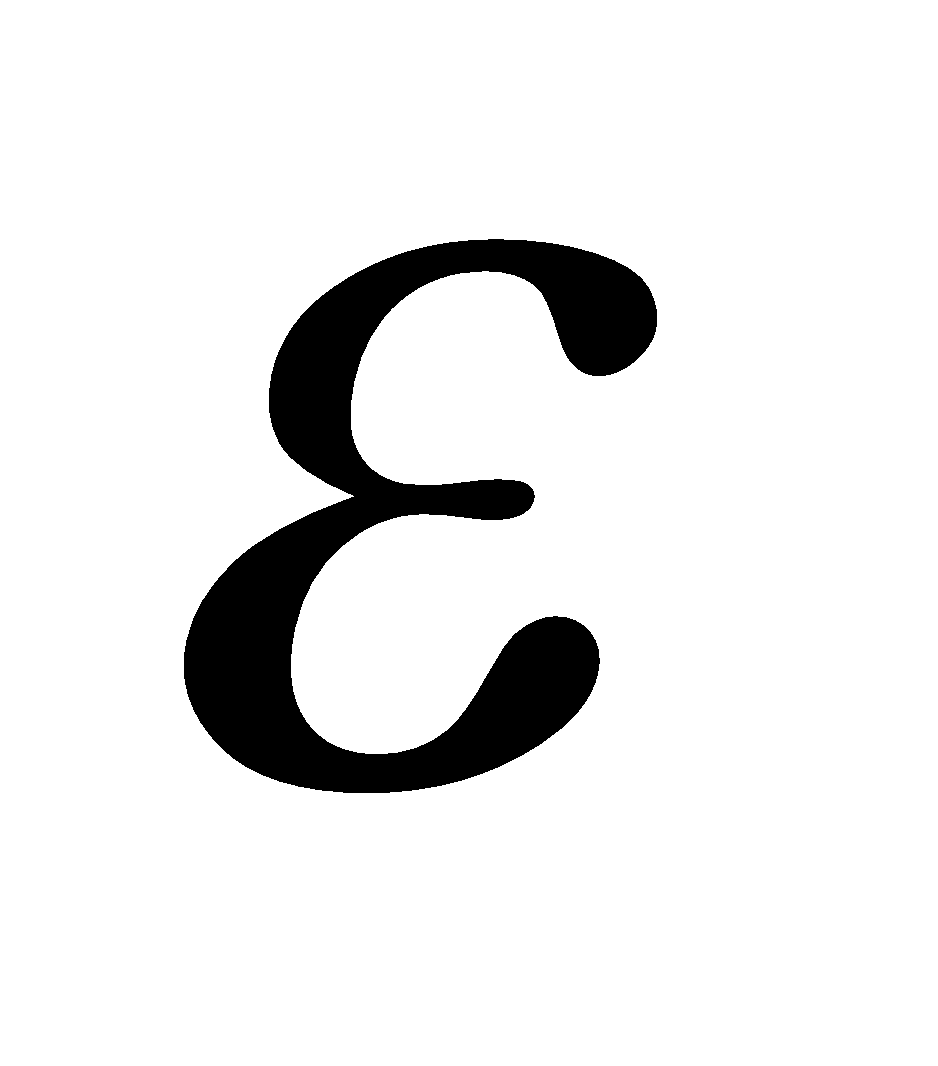
# Q Learning [2] algorithm applied to the experimental setup:

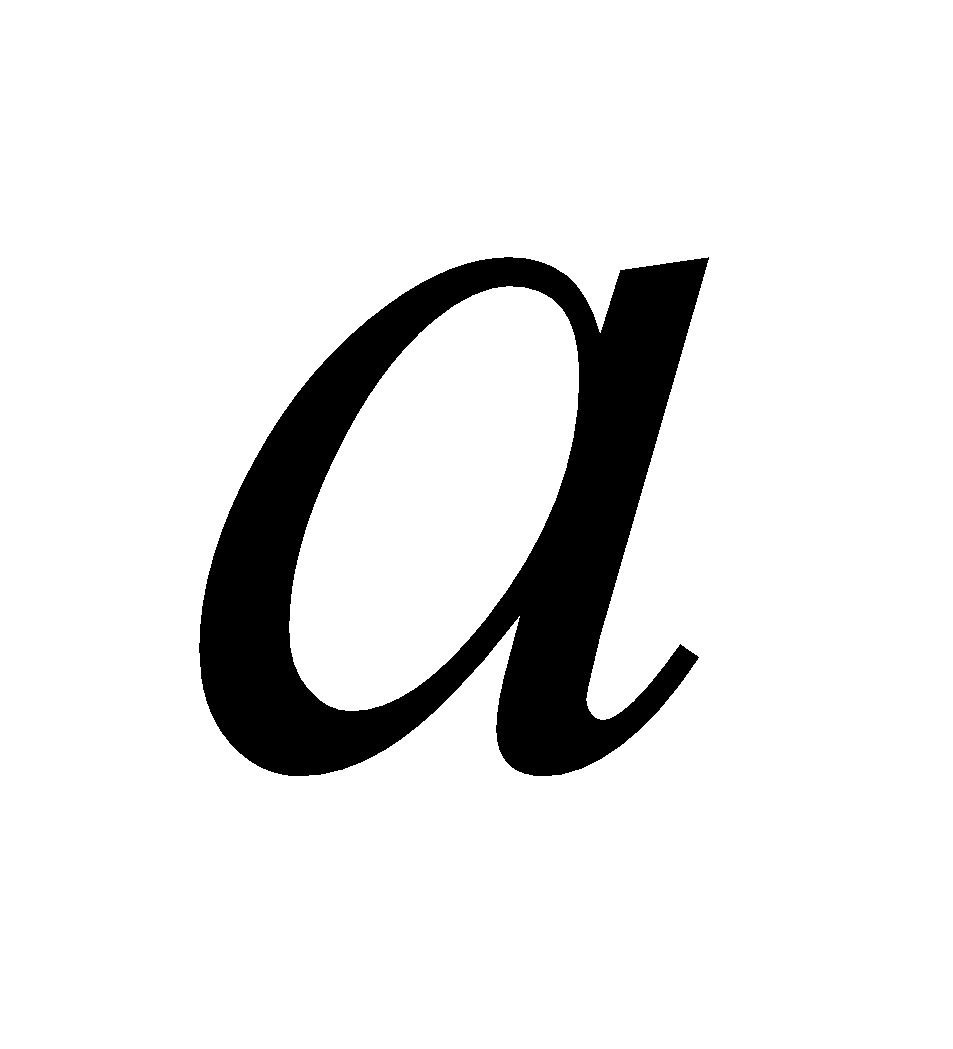
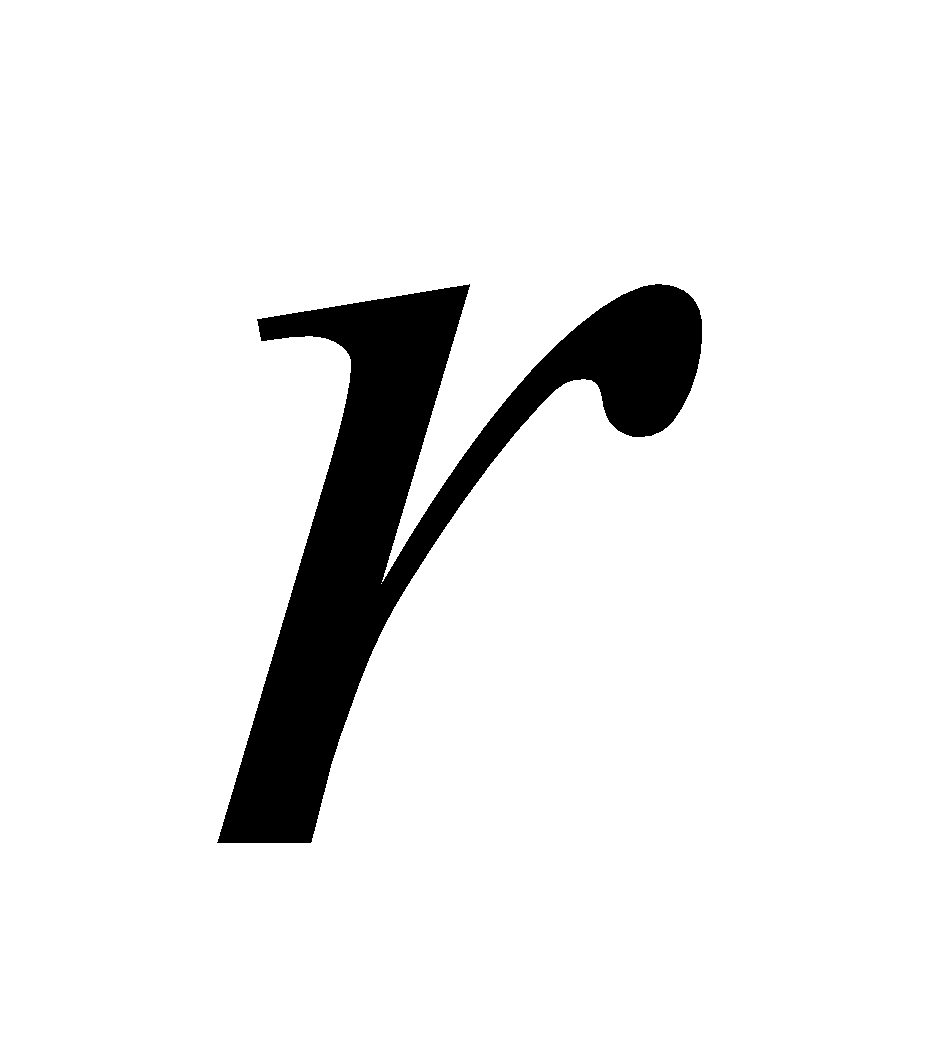
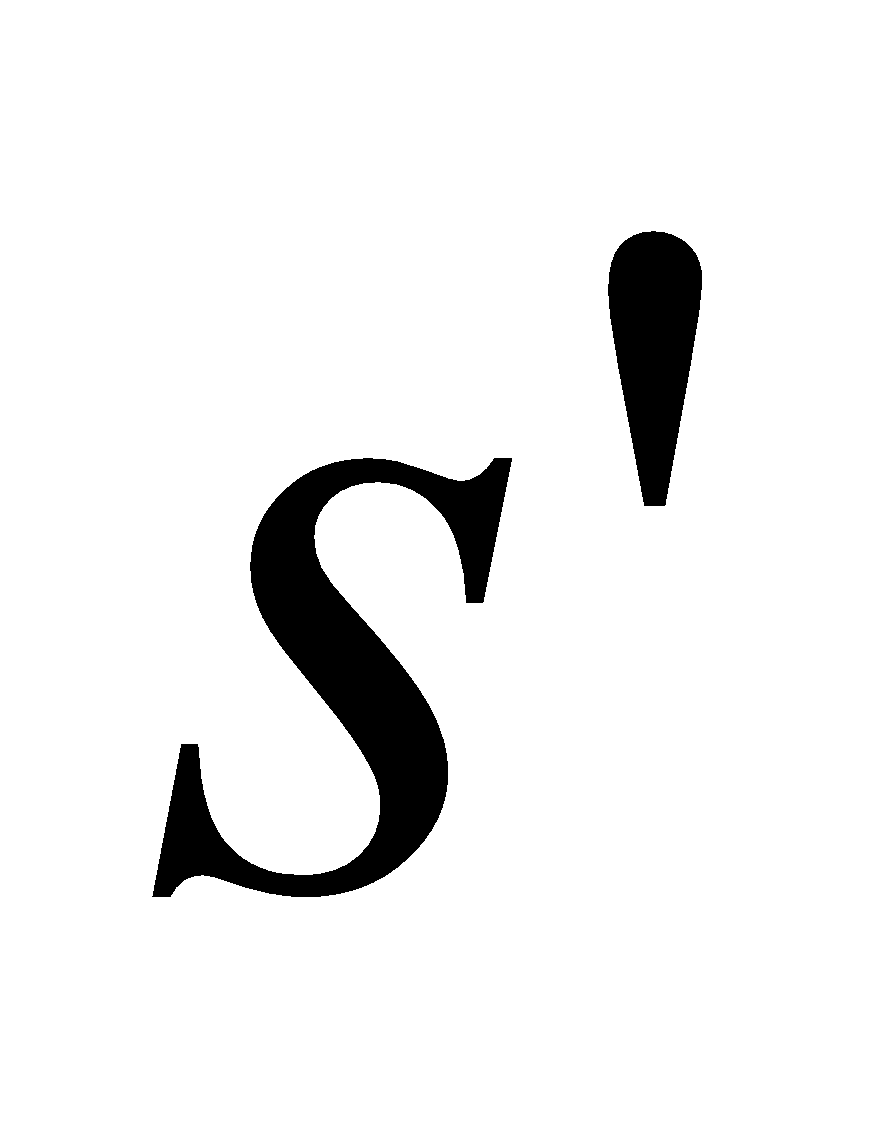
*Initialize  arbitrarily*

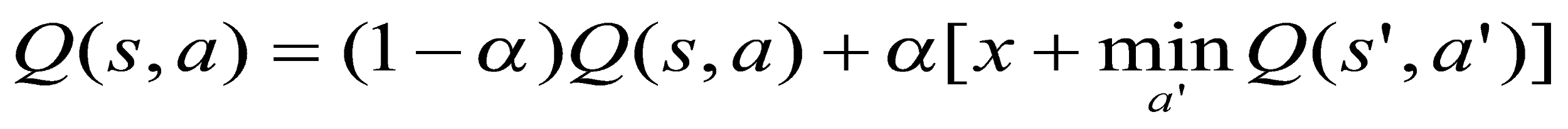
*Repeat for each episode*

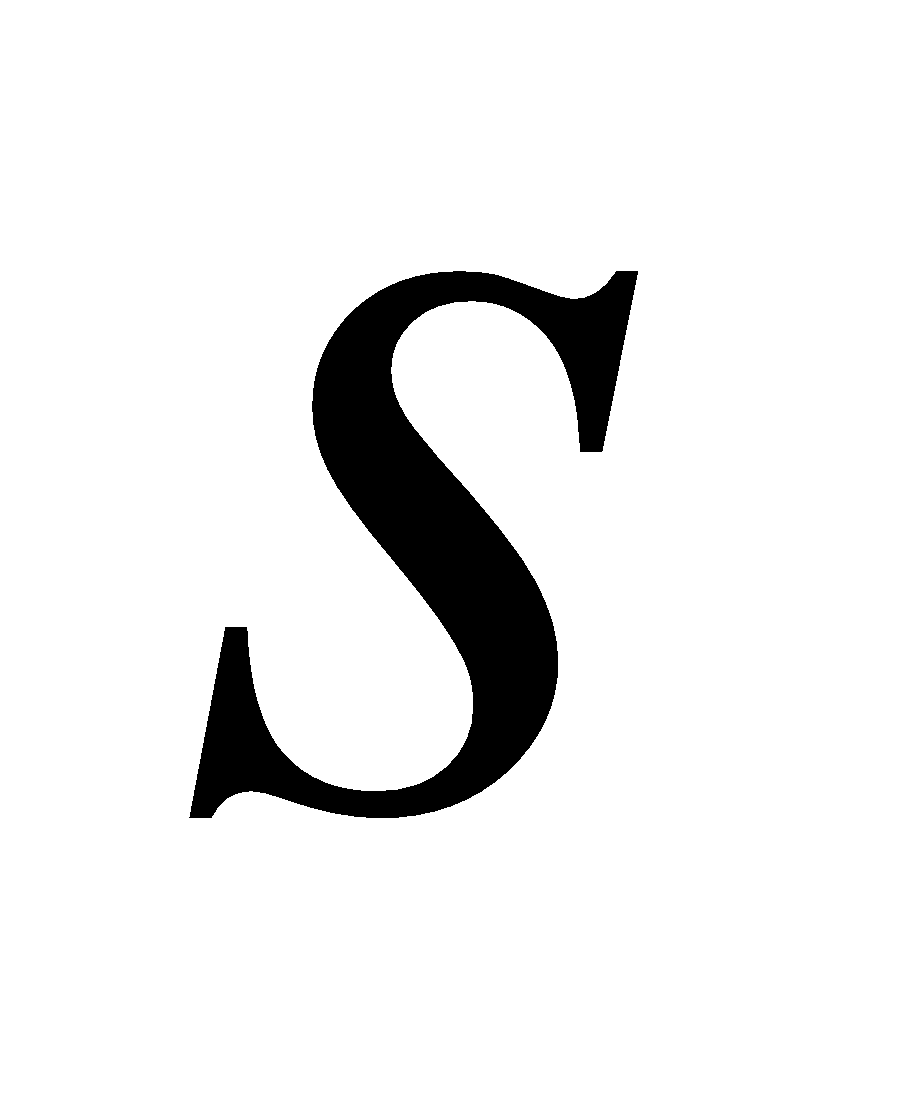
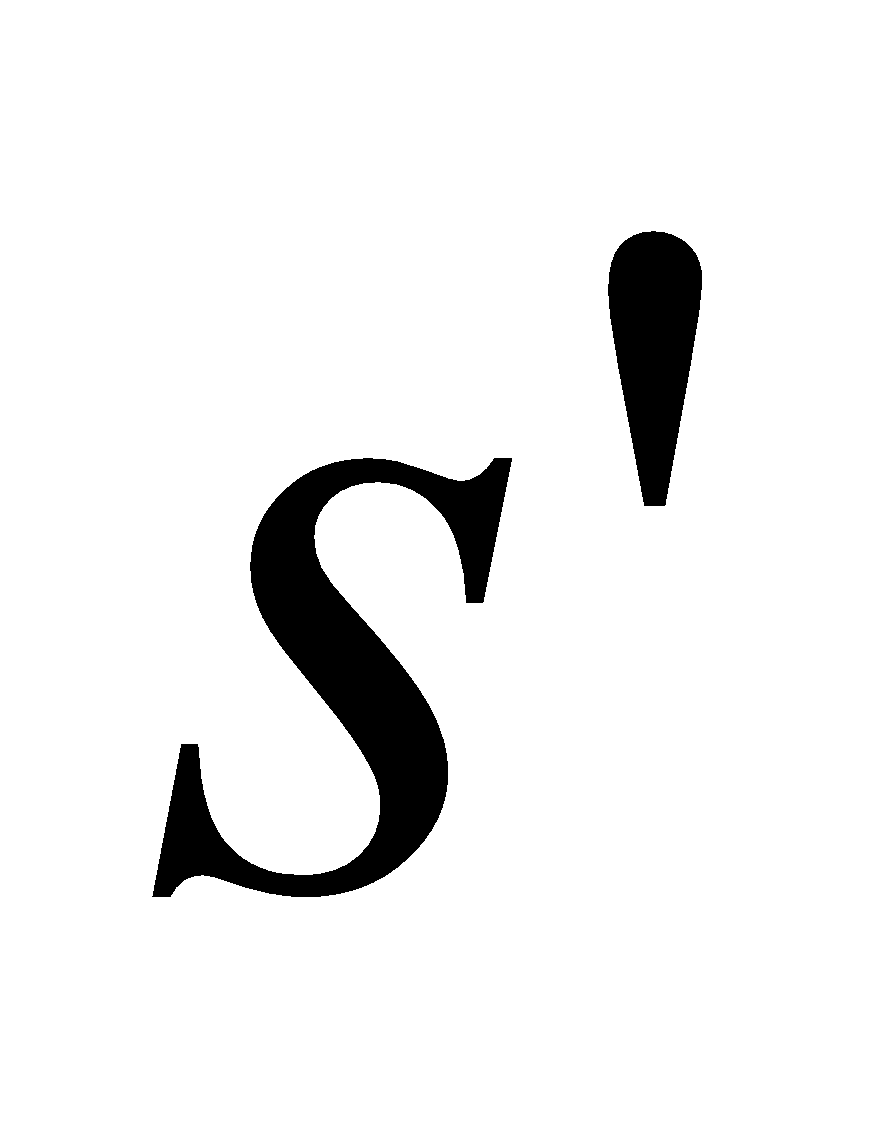
*Initialize *

*Loop until s is goal*

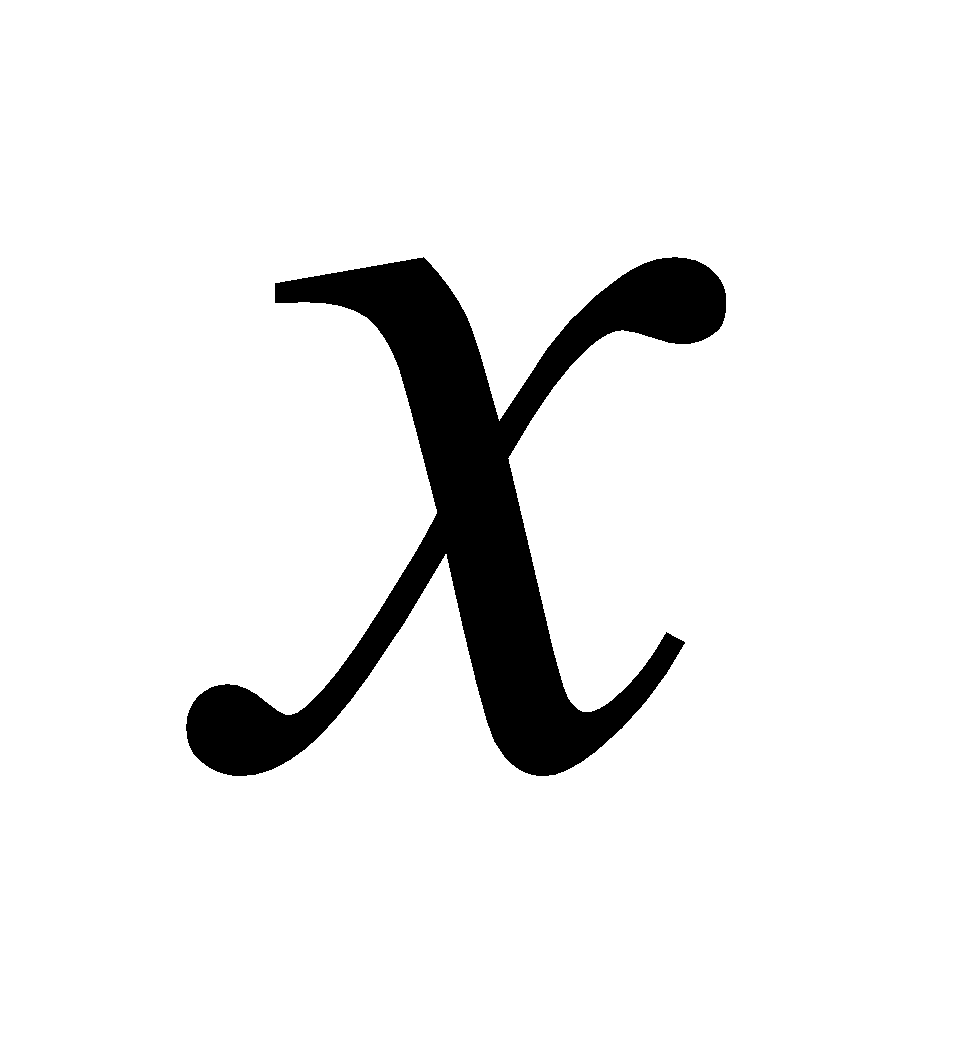
*Choose  from using -greedy policy*

*Perform and observe rewardand next state *

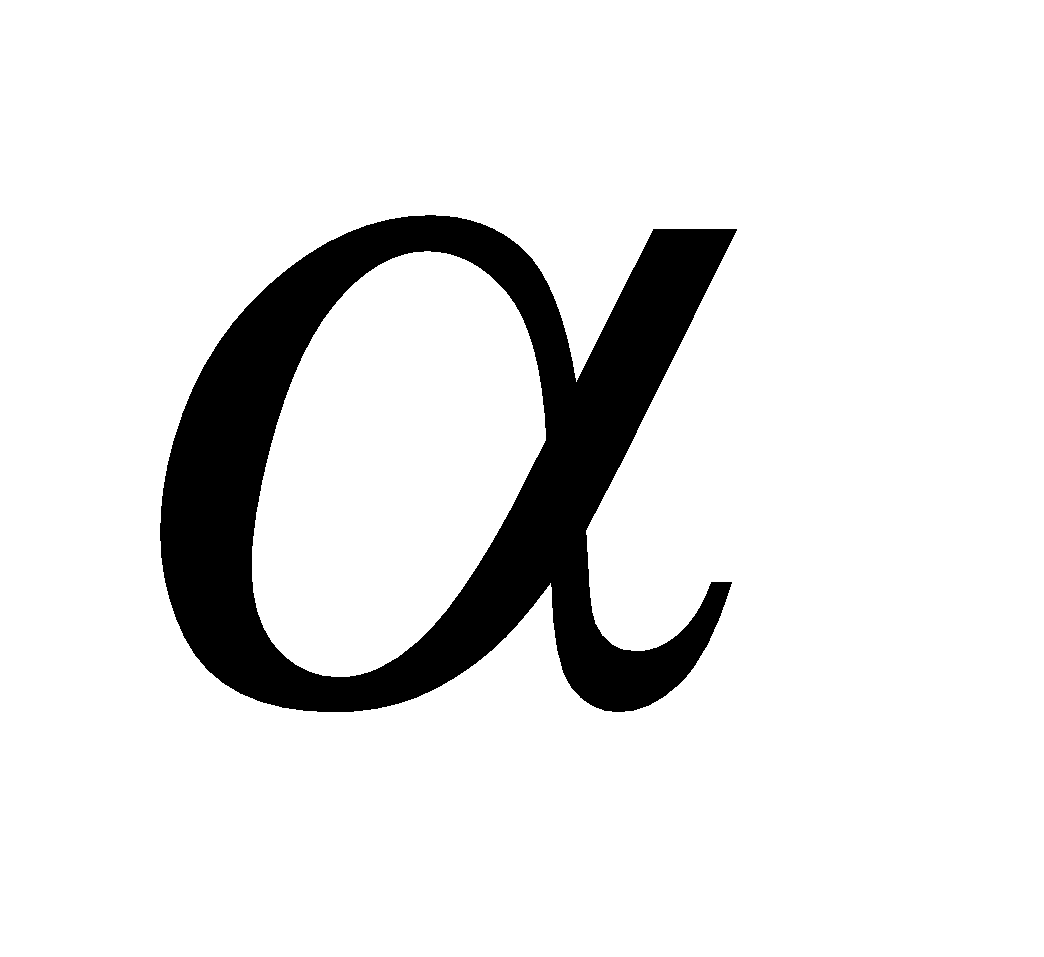
**

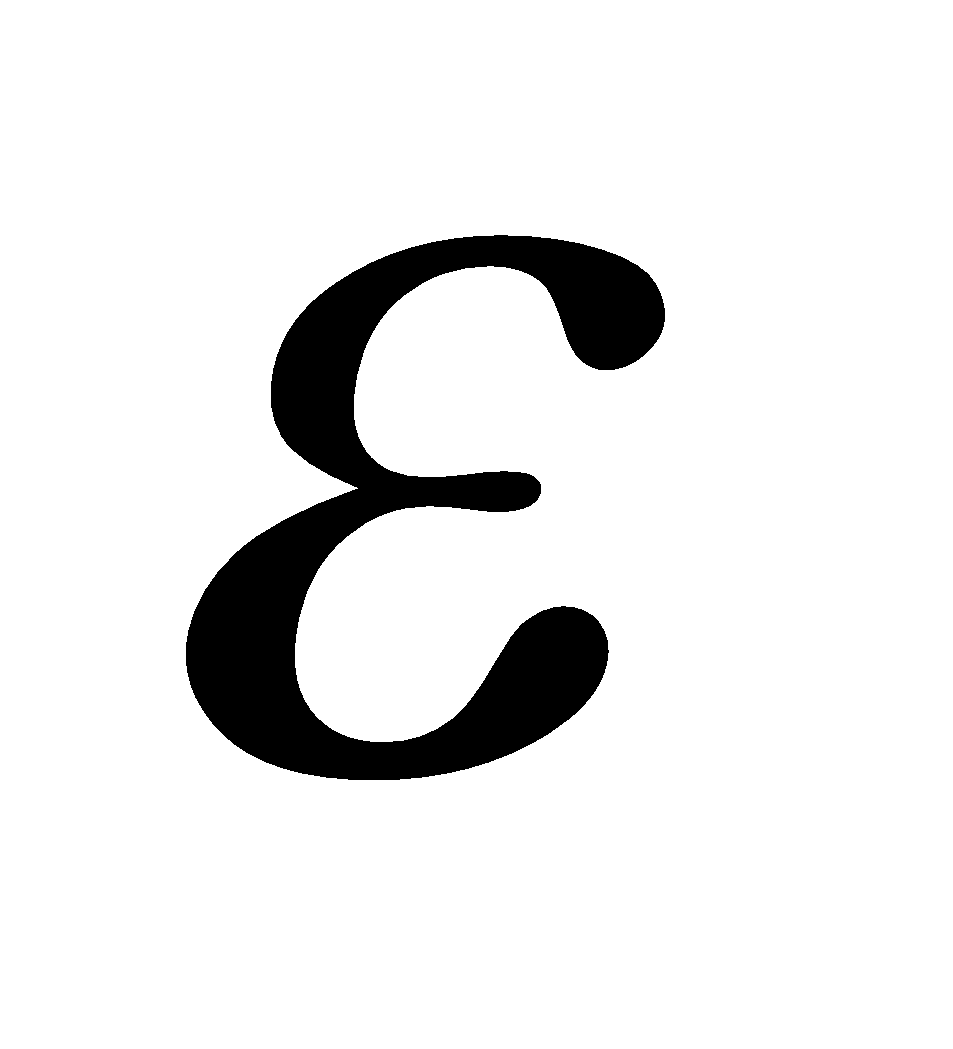
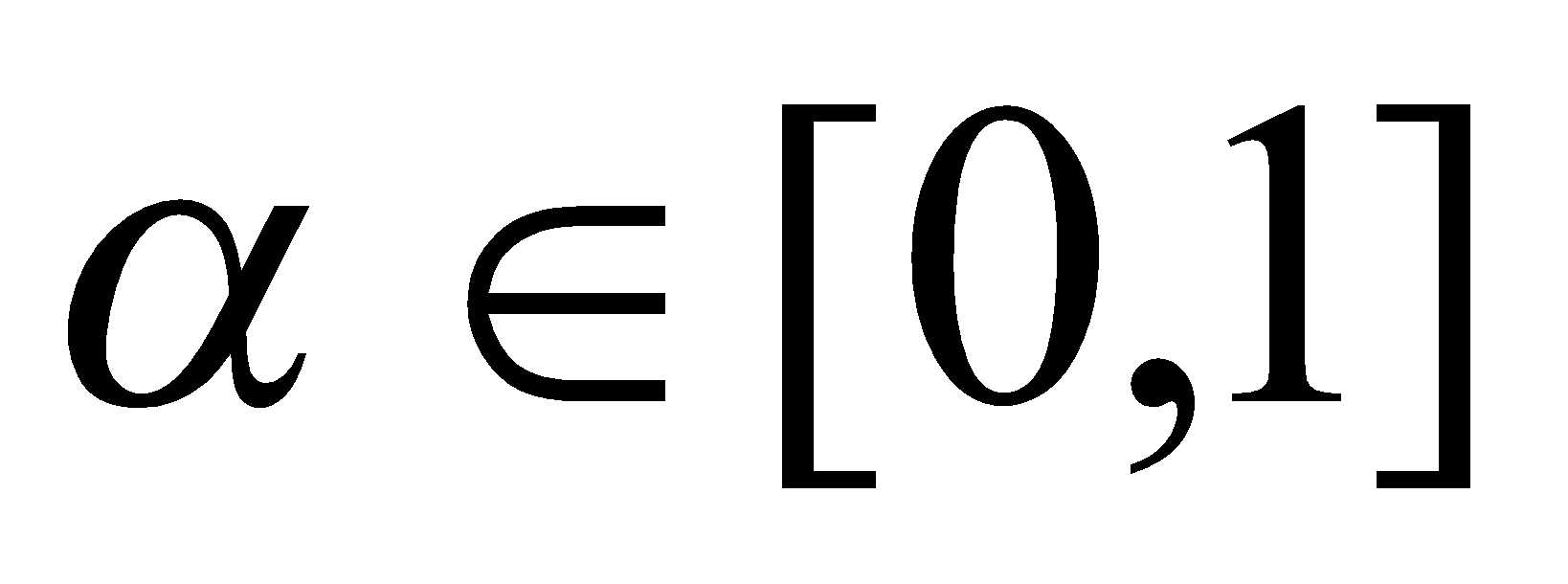
*= *

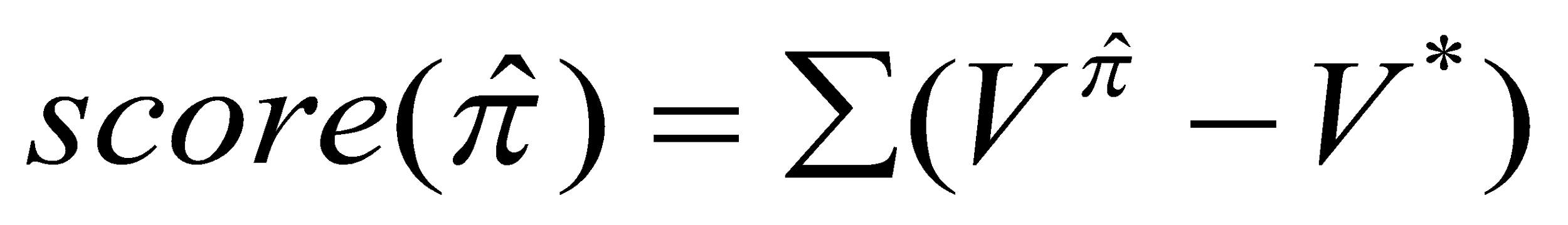
*End Loop*

*Where,  = 1, if transition was safe*

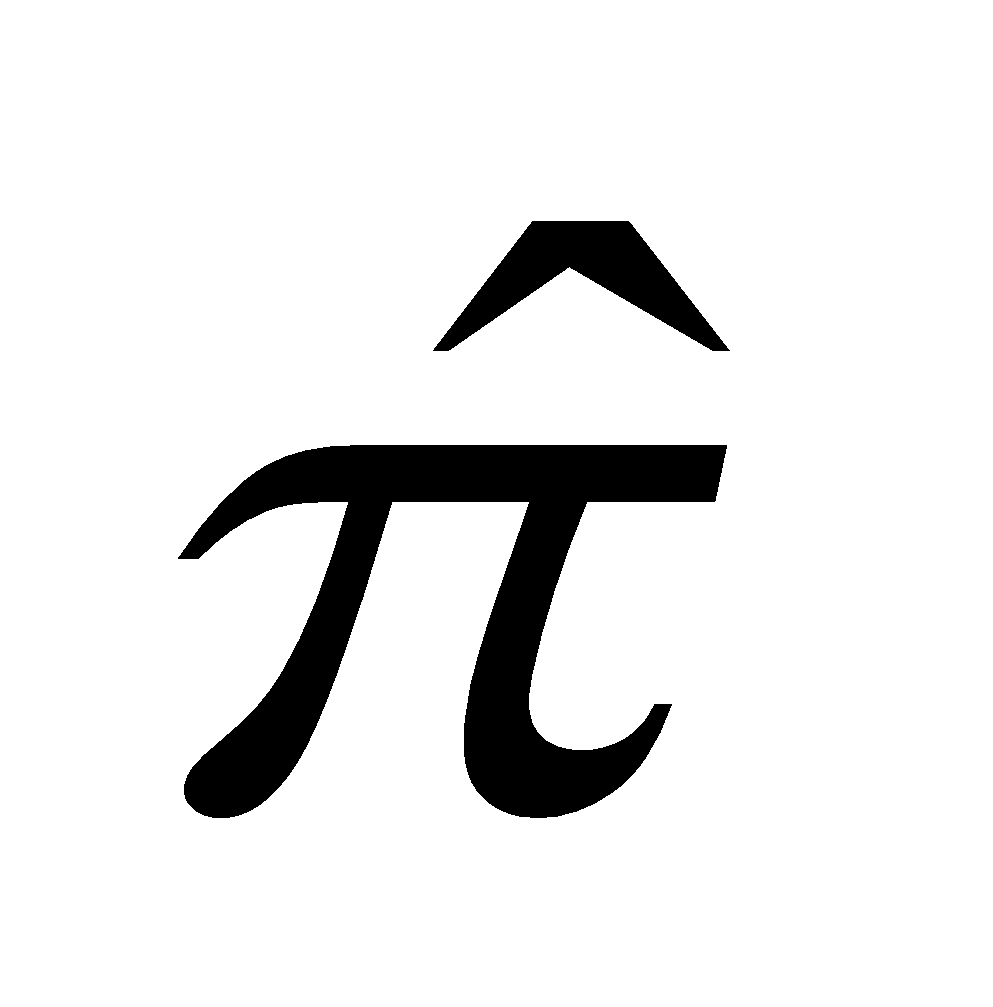
*= penalty, if transition is unsafe*

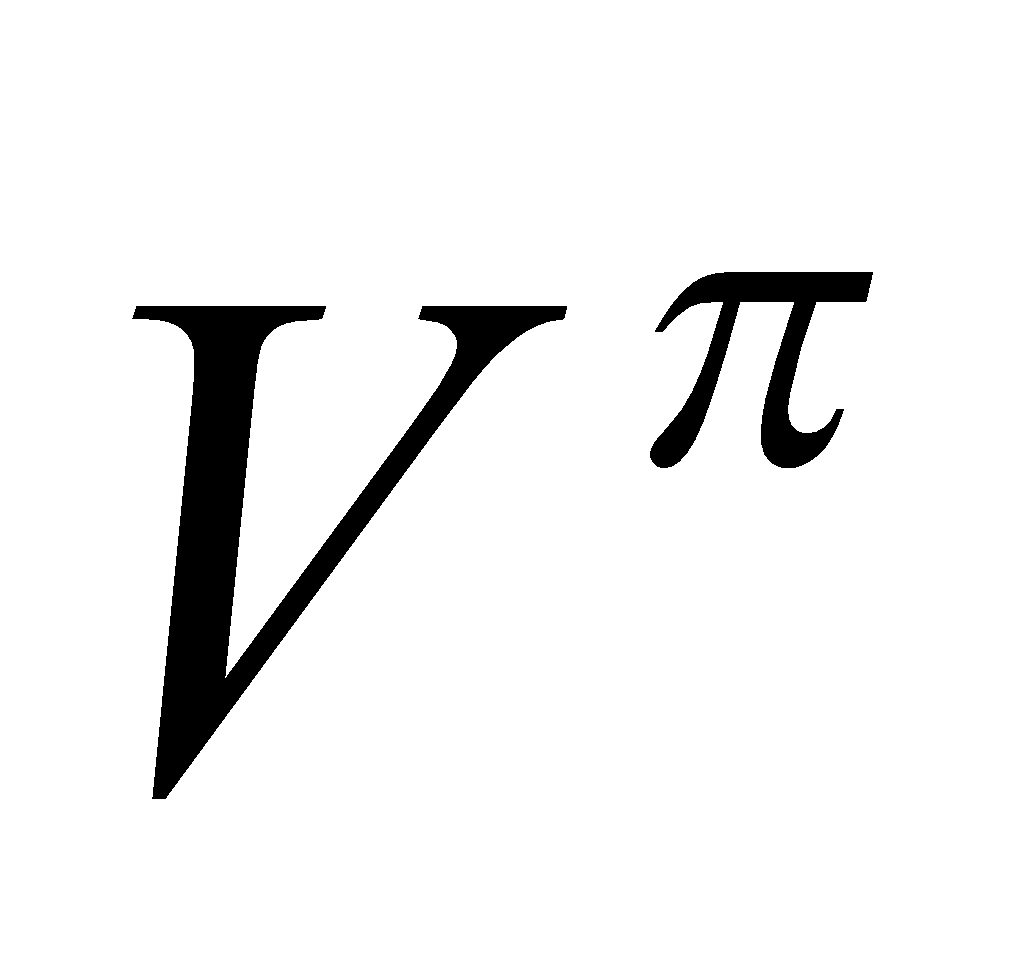
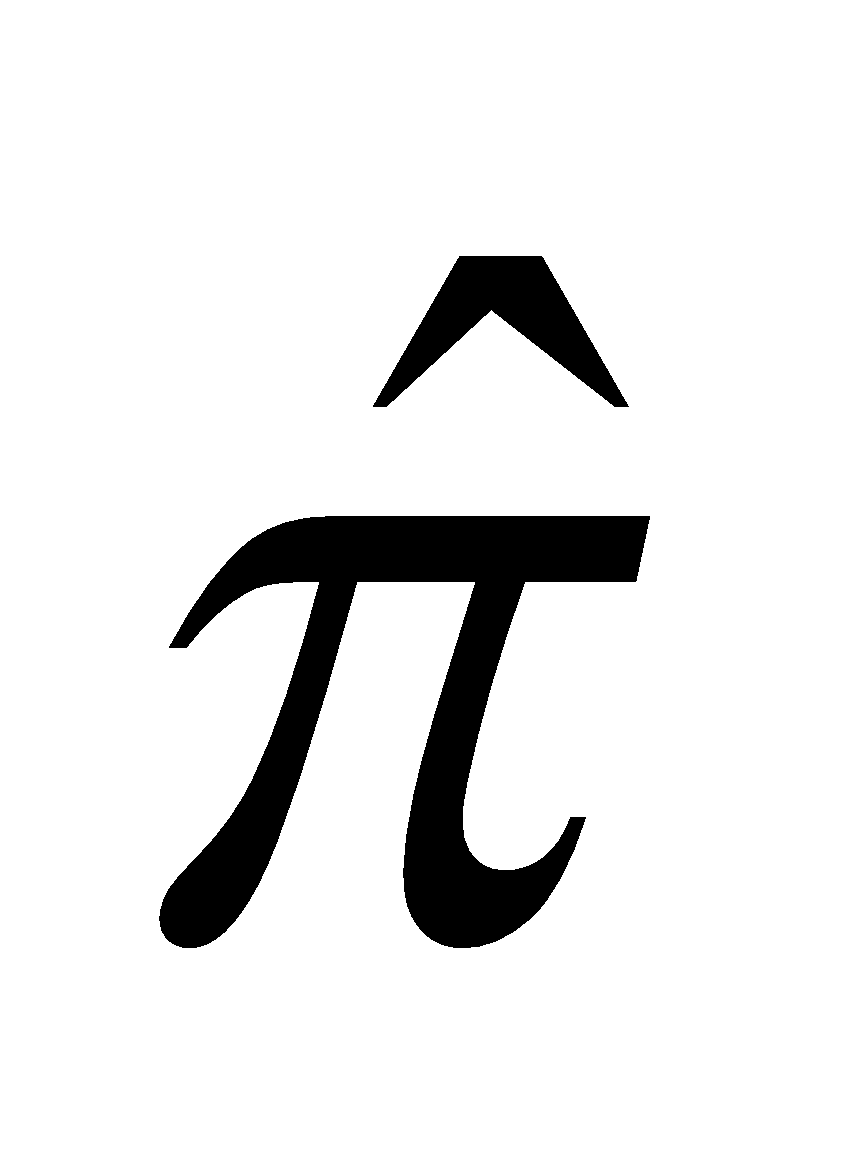
*= learning rate*

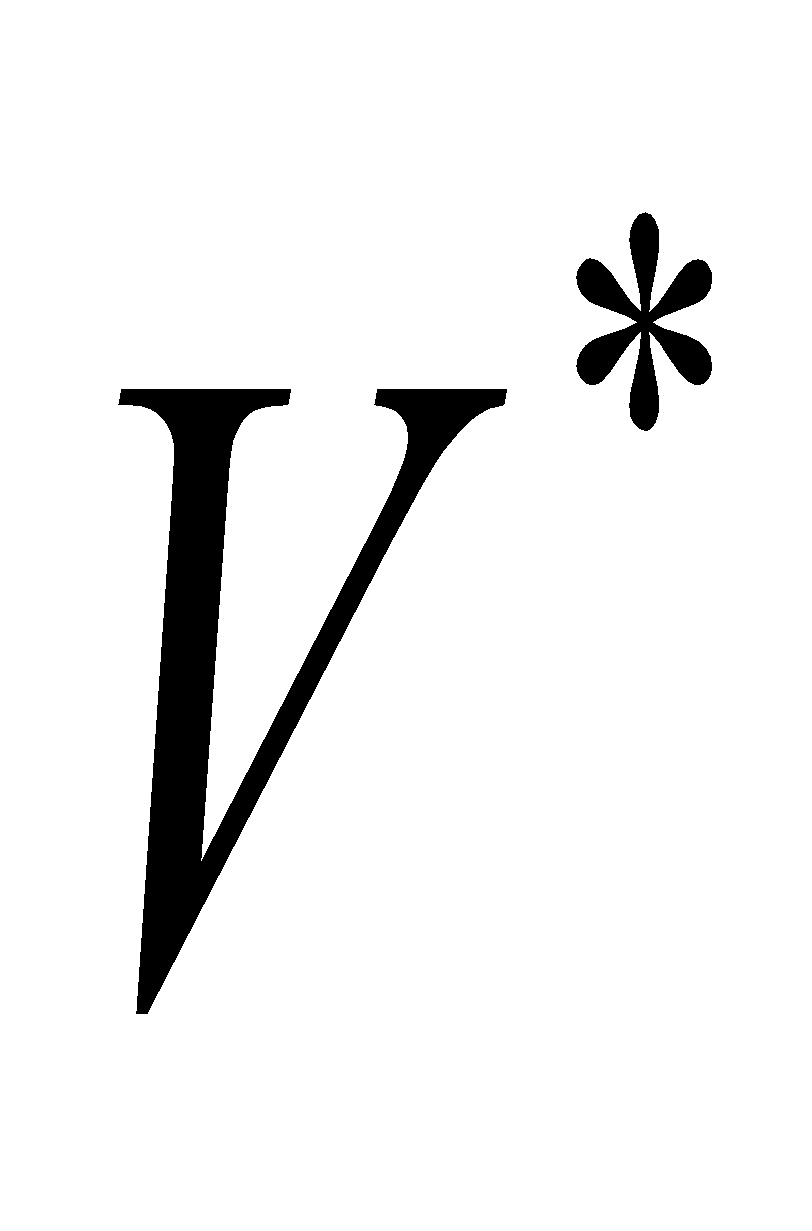
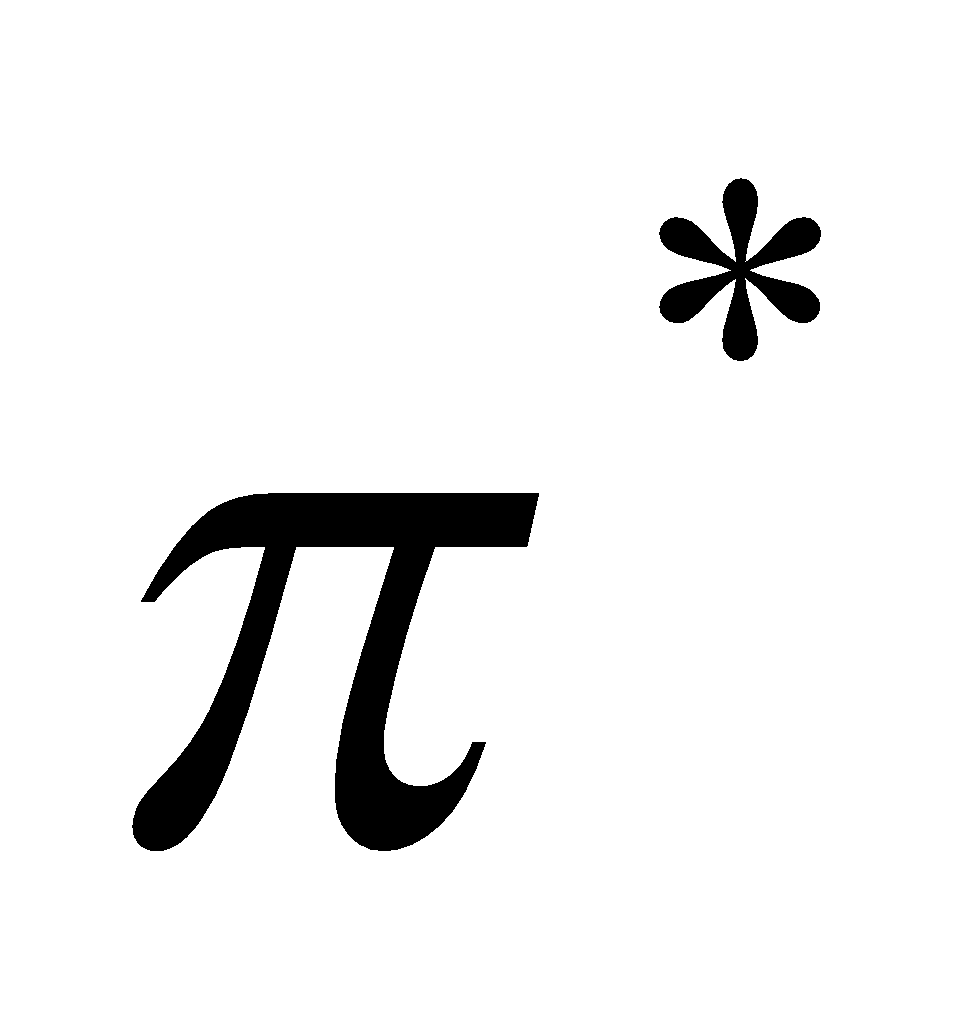
Q Learning is a model free algorithm that maintains the values for the state action pairs, called Q values, it experiences. The action in each state is chosen according to the -greedy policy. It uses a parameter called as learning rate, to update the Q values for each state it experiences. As the name signifies it determines how quickly the agent learns from the environment to discover the optimal policy. Figures 5 and 6 compare the effect of learning rate on the achievement of policy. Each figure plots the score of the policy against the number of episodes. The score of the policy is defined as follows



Where,

: Current policy whose value is to be evaluated

: Value function of each state if the agent executes the current policyuntil convergence

: Value function of each state if the agent executes the optimal policyuntil convergence

It can be observed from Figure 5 that for a learning rate of 0.1 the agent improves its policy faster but does not settle down to an optimal policy and keeps oscillating between a near optimal policy and a bad policy. Whereas, as seen in Figure 6, the agent with learning rate of 0.001 settles to the optimal policy slowly but steadily.

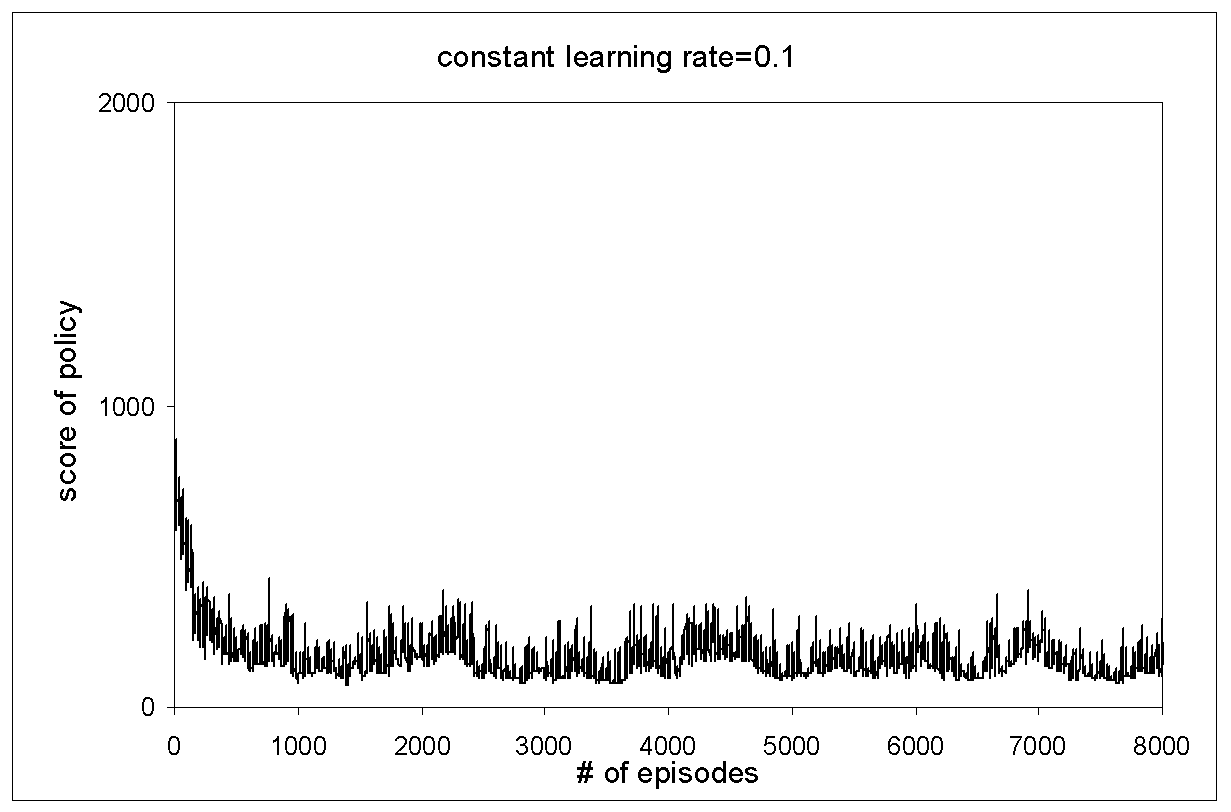


Figure 5: The score of the policy as the number of episodes increase for a relatively high constant learning rate

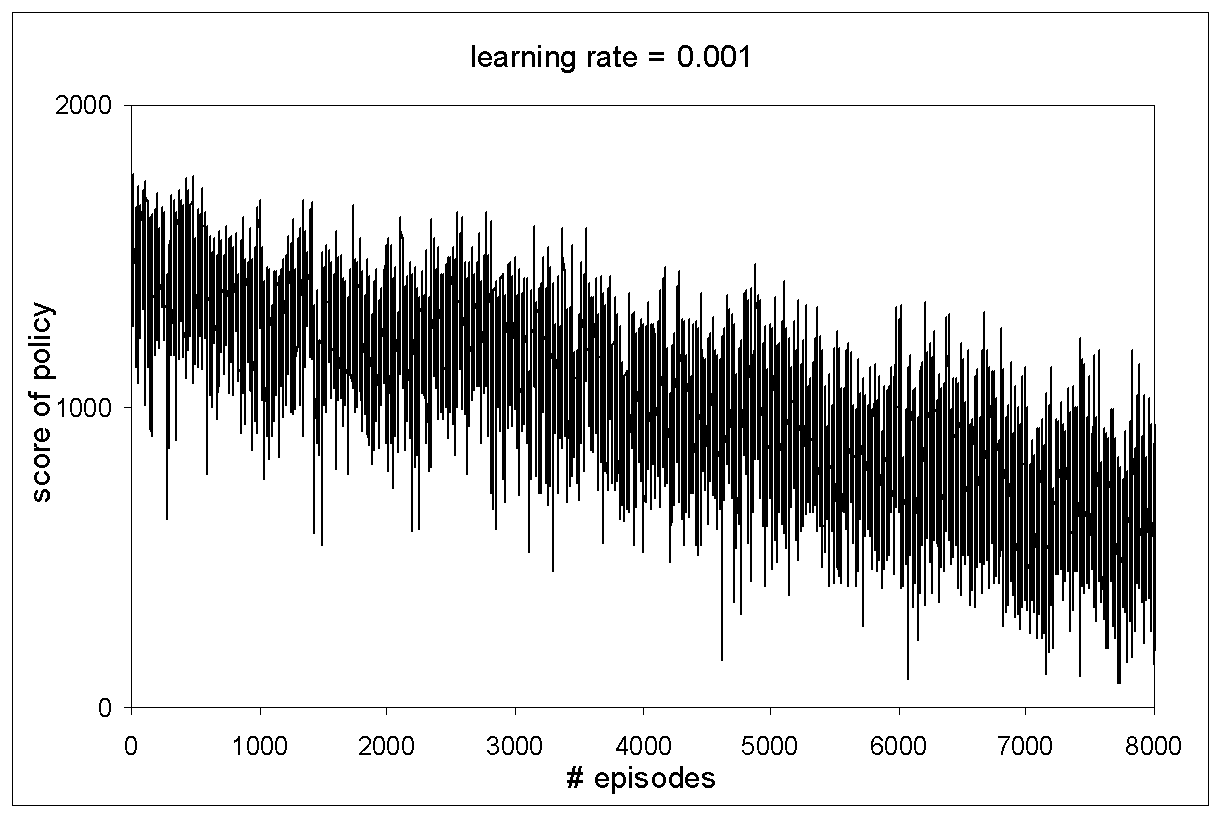


Figure 6: The score of the policy as the number of episodes increase for a relatively low constant learning rate

This can be attributed to the fact that the agent is more sensitive to the reinforcements it receives from the environment if its learning rate is higher. In a noisy environment the agent executes an action but does not reach the desired next state. Instead, the agent is pushed against its will to any of its neighboring states with certain probability. Under such a noisy environment the agent might execute the optimal action yet end up in a bad state or receive a penalty. In such a situation if the learning rate is high the value associated with that state-action pair changes significantly. This causes the agent to alter its belief in the goodness of that action and change its policy. Hence with a high learning rate the agent frequently changes its policy. Thus, with a high learning rate agent thus becomes more sensitive to noise. At the same time a high learning rate is useful in the initial stages of the agent’s interaction with the environment. A high learning rate causes the agent to imbibe the dynamics of the environment quickly. This shows that a high learning rate works better than a low learning rate in the initial stages of the agent’s interaction with the environment whereas a low learning rate works best in the later stages of the agent’s interactions with the environment.

## *Decaying learning rate* [4]

A slight modification can be done to Q learning so that it can capitalize on the advantages of both low and high learning rate. This can be achieved by decaying the learning rate as the number of iterations increase. According to this scheme the agent interacts with the environment with a very high learning rate initially so that it can quickly settle down to a near optimal policy. As the agent’s interactions with the environment increase the learning rate decays and the agent now makes only minor modifications to its near optimal policy to obtain the optimal policy. This scheme is guaranteed to asymptotically converge to the optimal policy. The learning rate under the decaying learning rate scheme is shown in Figure 7. The results of using the decaying learning rate are shown in Figure 8. It can be seen that the score of the policy decays faster due to high learning rate in the initial stages and then as the number of episodes increase the learning rate decays further and the score of the policy drops to zero.

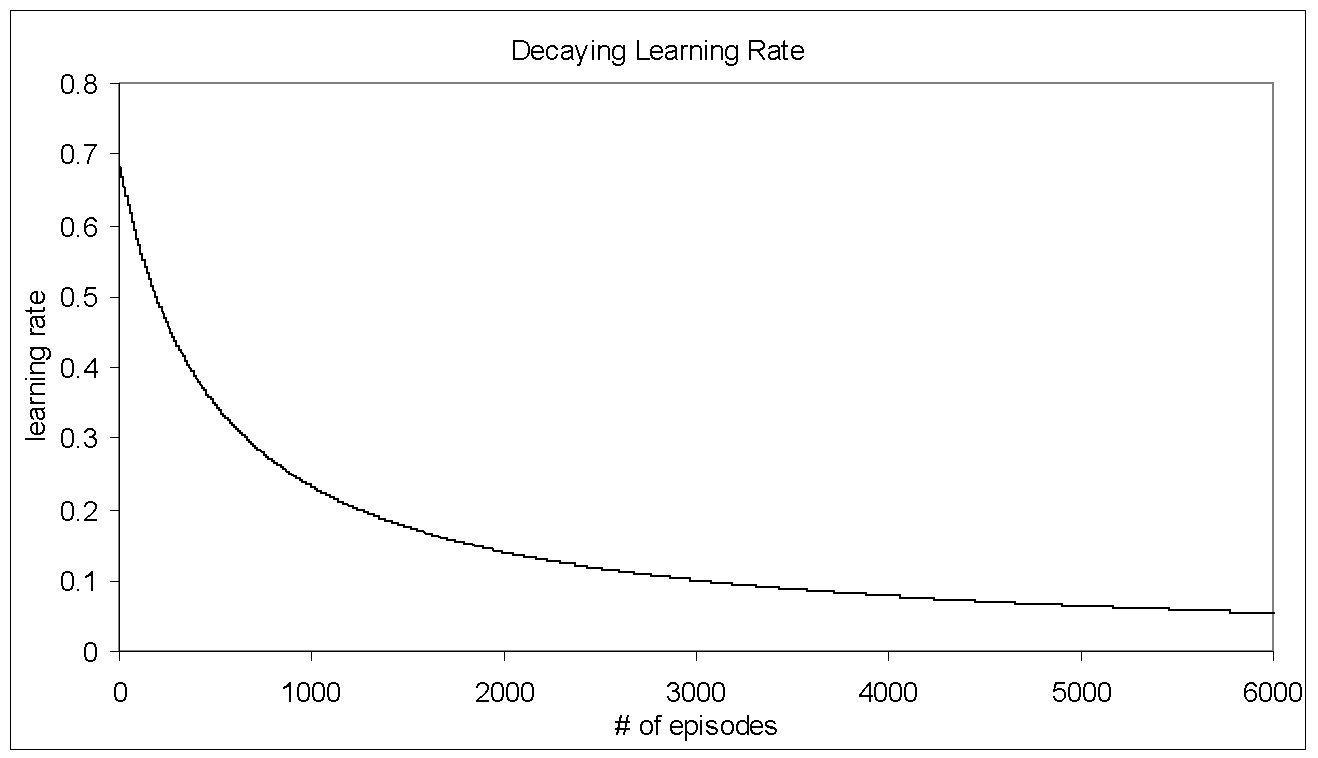


Figure 7: The learning rate as the number of episodes increase for a decaying learning rate scheme starting with a learning rate of 0.7

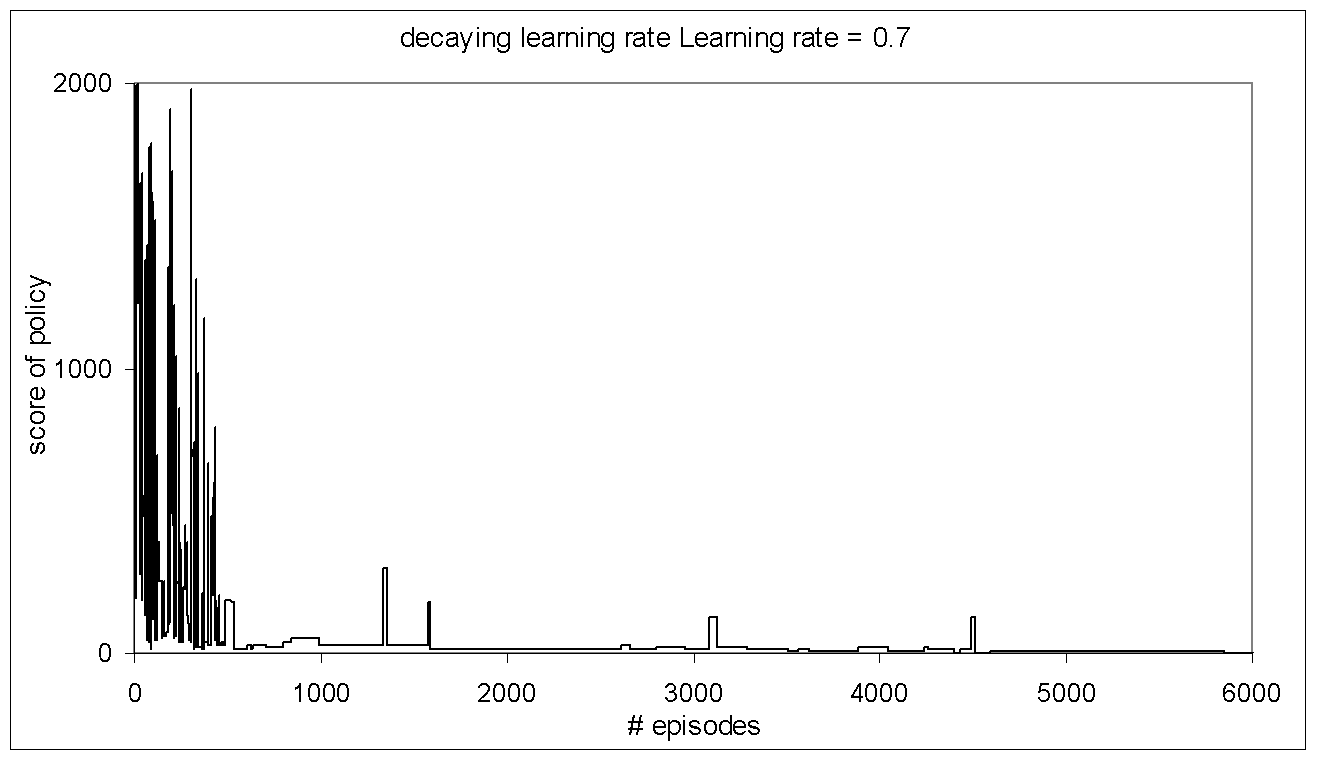


Figure 8: The score of the policy as the number of episodes increase for a decaying learning rate starting with an initial value of 0.7

## *Decaying learning rate applied to non stationary environment*

When the environment is non stationary the use of decaying learning rate causes degradation in the performance of the algorithm. As number of episodes increase the learning rate achieves a very small value and the changes in the environment make insignificant changes in the Q-values that the agent maintains. Because of this the agent keeps executing a sub-optimal policy. This phenomenon can be seen in the Figure 9. Here the environment is changed after every 2000 episodes. It can be observed at one such instance after 6000 episodes the environment changes and the score of the policy that the agent is executing takes a long time to settle to zero. This means that the agent takes longer time to discover the optimal policy. It has also been observed that as the agent’s interactions with the environment increase the learning rate further decreases and each time the environment changes the time taken by the agent to discover the optimal policy increases.

It is clear from the above discussion that the problem of increased time taken by the agent to find the optimal policy can be attributed to extremely low learning rate as a result of using the decaying learning rate scheme. At the same time the decaying learning rate scheme cannot be discarded because it undoubtedly performs better then the fixed learning rate scheme.

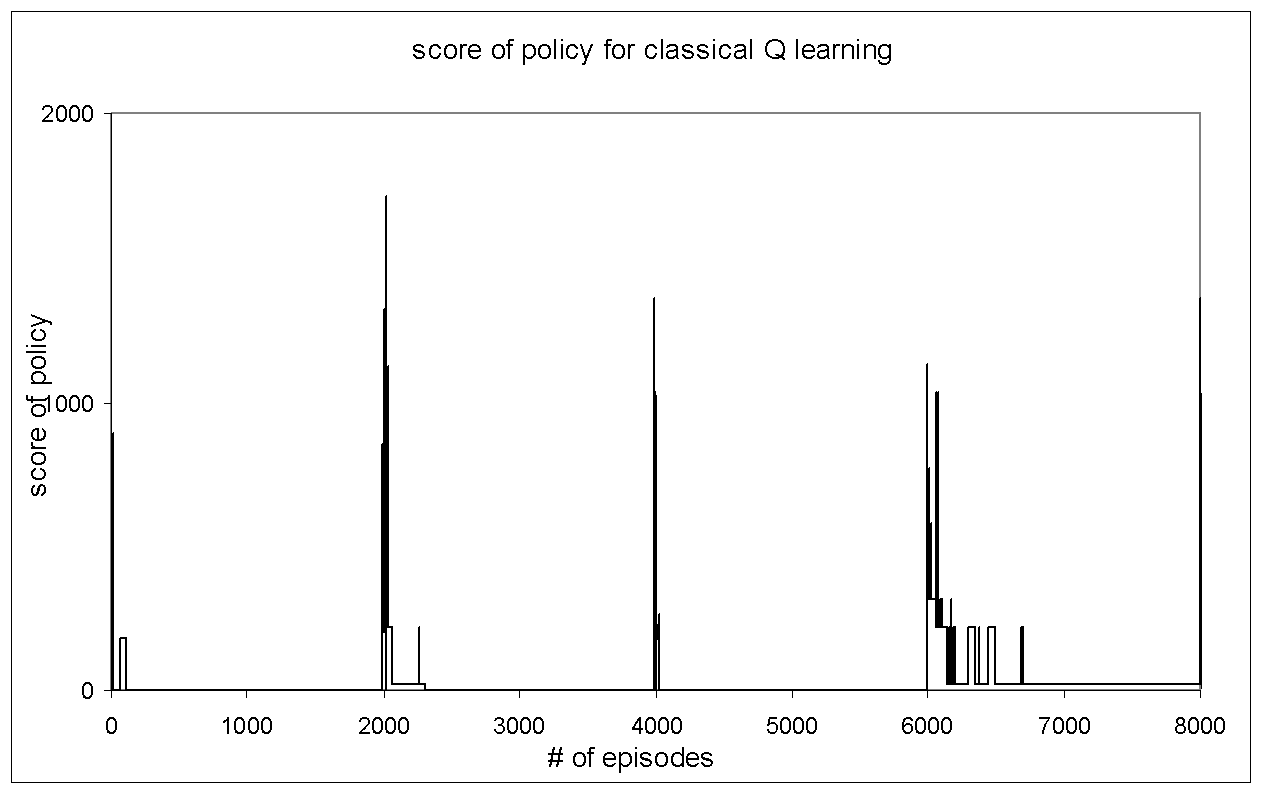
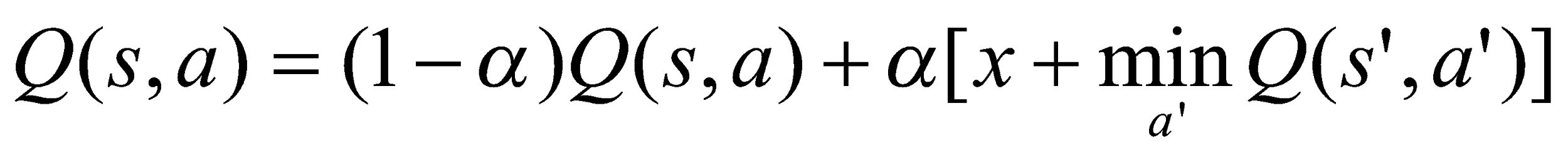


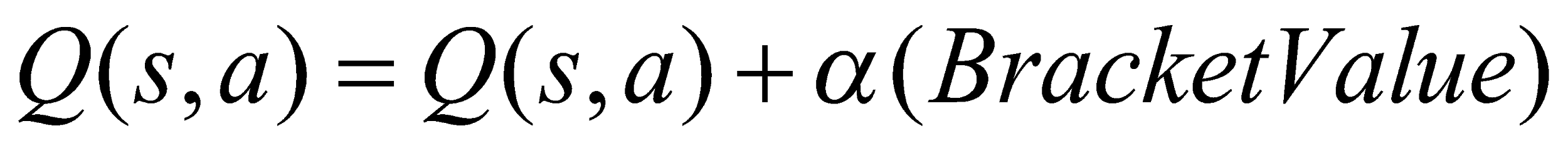
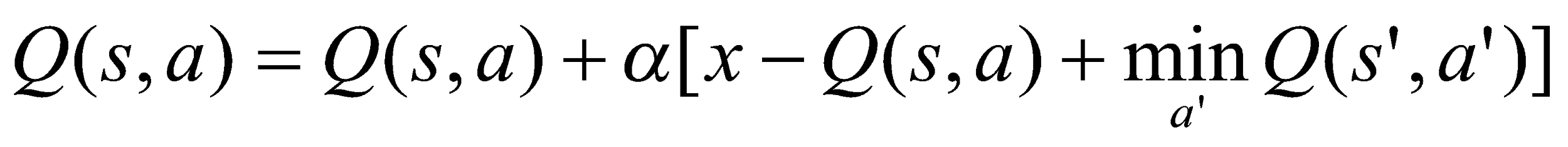
Figure 9: The score of the policy as the number of episodes increase for a decaying learning rate scheme.

## *Adaptive learning rate for Q learning in non stationary environments*

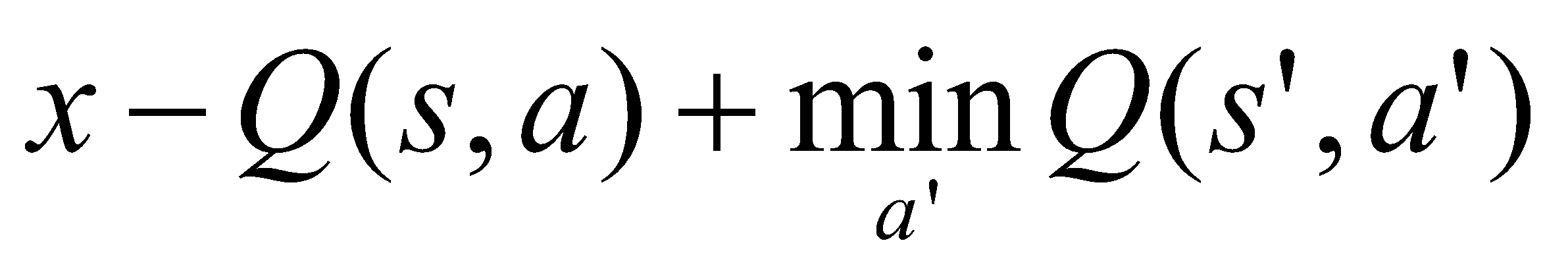
To remedy this problem with the decaying learning rate scheme as applied to non stationary environments the concept of adaptive learning rate is introduced. This involves resetting the learning rate whenever the environment changes. But this requires the agent to infer intelligently that the environment has changed based on the stimuli that it receives from the environment. One such indication that the environment has changed is the sudden rise in the values that the agent uses to update its Q values. The Q learning update rule is as follows.

**

Rearranging the terms,



Where,

*Bracket Value* = 

It is observed that whenever the environment changes there is a sudden increase in the *Bracket Values*. This illustrated in Figure 10.

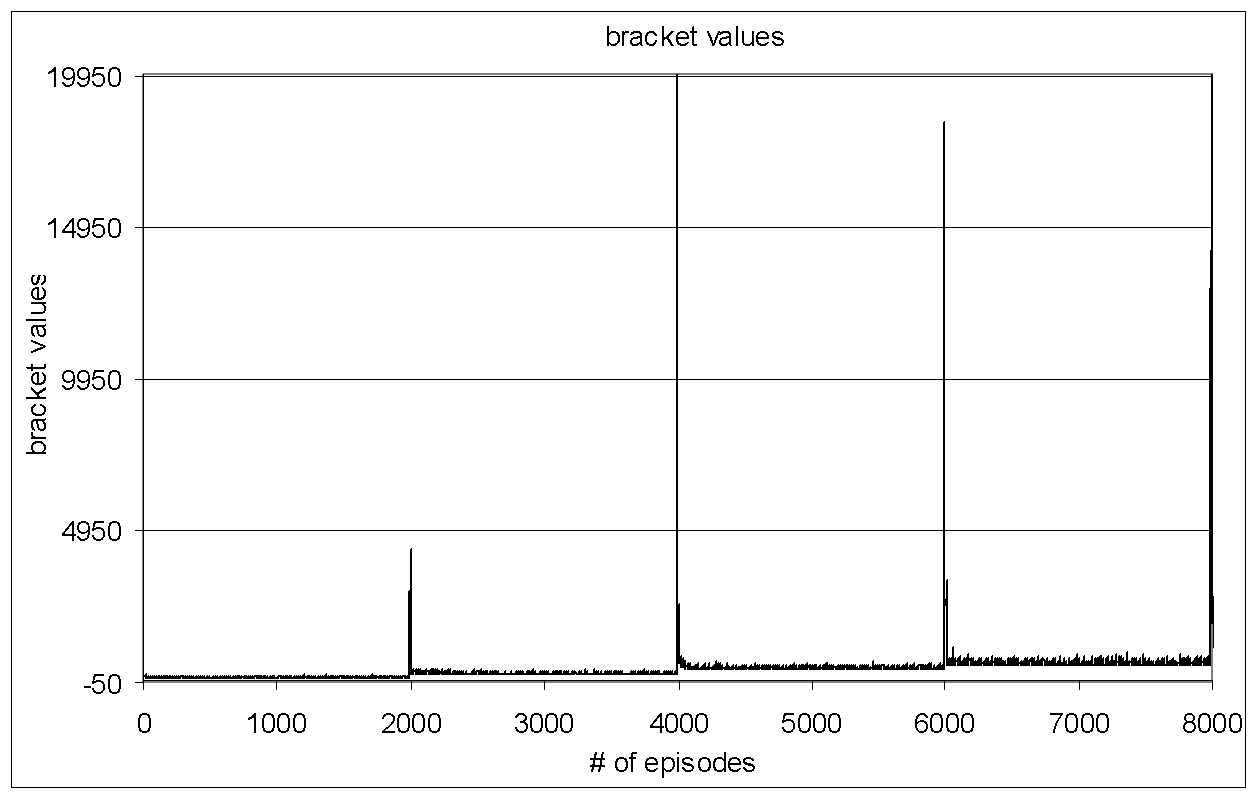
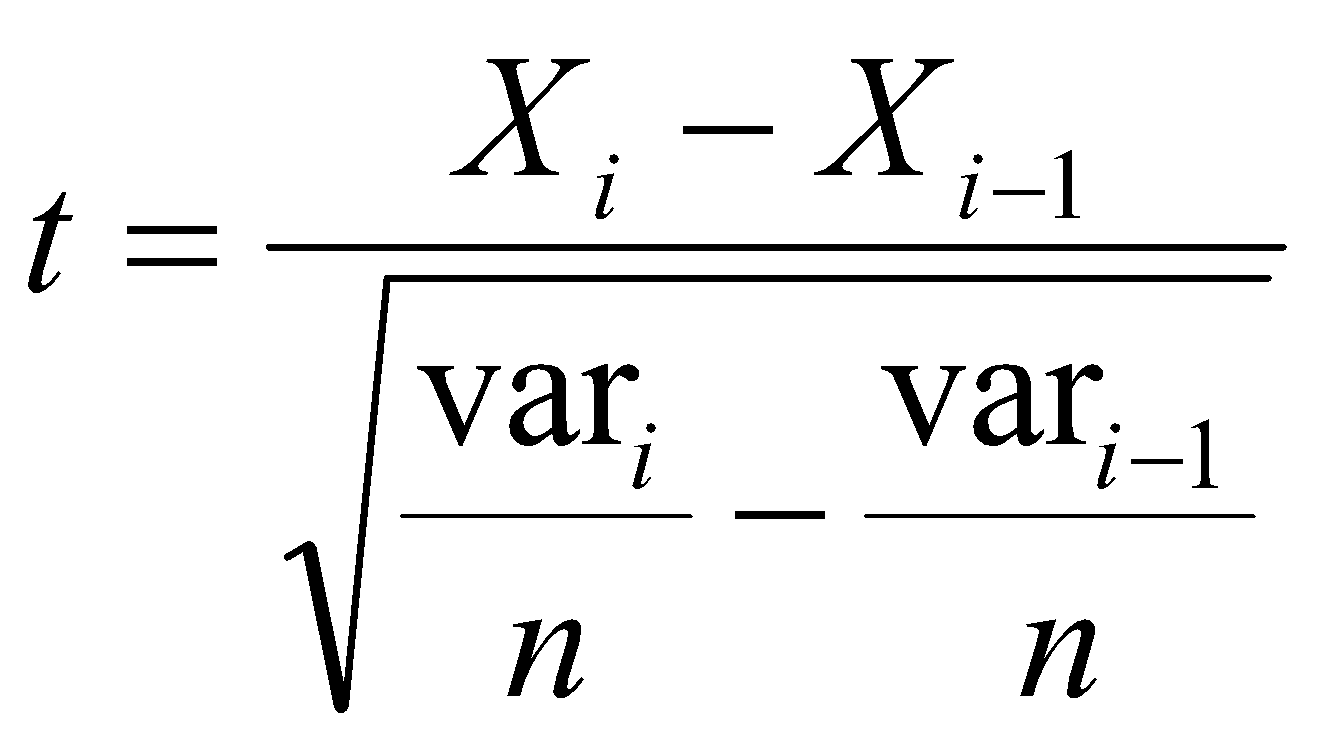


Figure 10: The bracket values as the number of episodes increase for decaying learning rate scheme.

Here the environment is changed after every 2000 episodes. It is clear from Figure 10 that the bracket values are good indicators of the changes in the environment. If the learning rate in the decaying learning rate scheme is reset each time the environment changes then such a scheme would perform with the same advantages as the decaying learning rate scheme in non stationary environments as well. So the problem now is to detect the changes in the *Bracket Values*. This can be done by maintaining a window of past *n Bracket Values* and detecting a change in the means of two successive windows. If the environment changes the change in means between two successive windows is significant. To detect the change in means the t-test is used as follows.



Where, *n*: number of episodes for which the *Bracket Values* are maintained which is same as the size of the window

Xi, Xi-1: the means of the previous *n* bracket values after the ith and the (i-1)th episodes respectively

vari, var(i-1) : the variances of the previous *n* *Bracket Values* after the ith and the (i-1)th episodes respectively.

The results of applying the adaptive learning rate scheme to Q learning in non stationary environments are shown in Figures 11 and 12. It can be clearly seen that by resetting the learning rate when the environment changes the agent is able to find the optimal policy in a shorter period of time as compared to the decaying learning rate scheme.

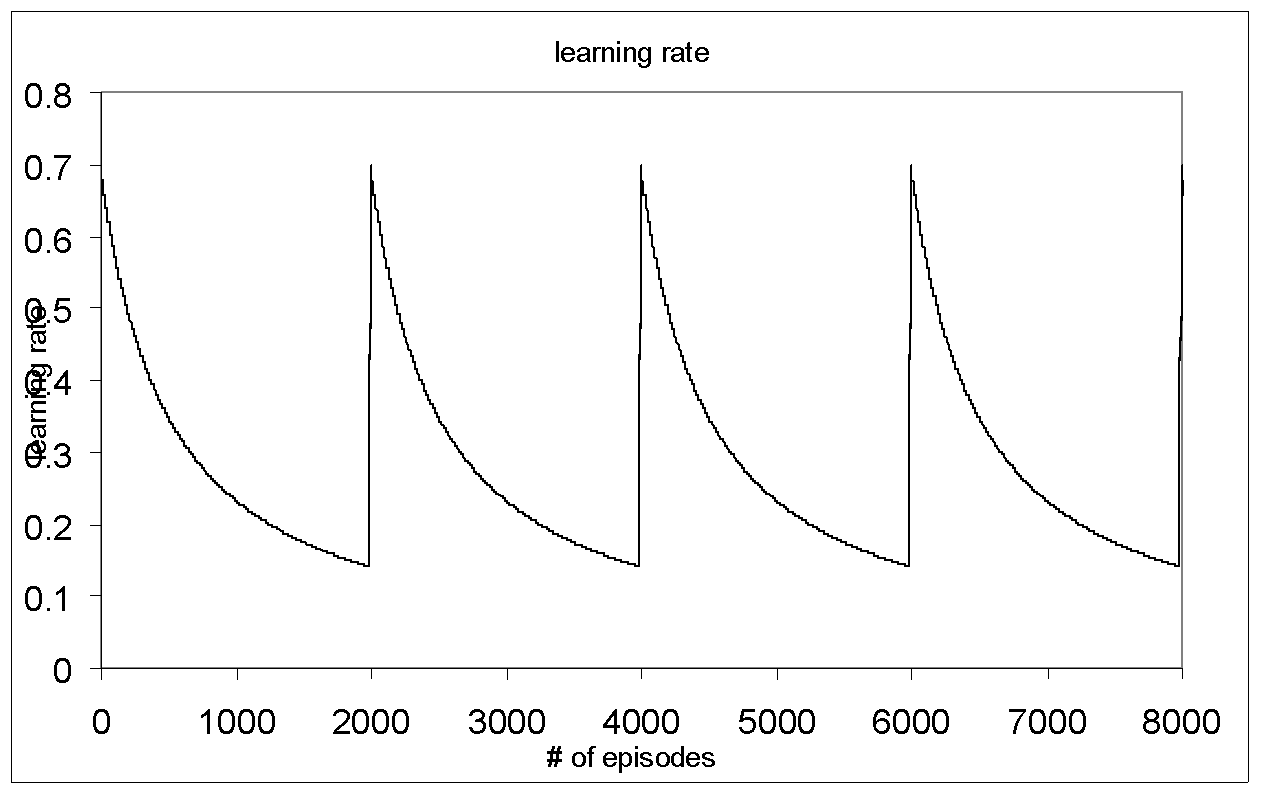


Figure 11: The learning rate for the adaptive learning rate scheme in a non stationary environment.

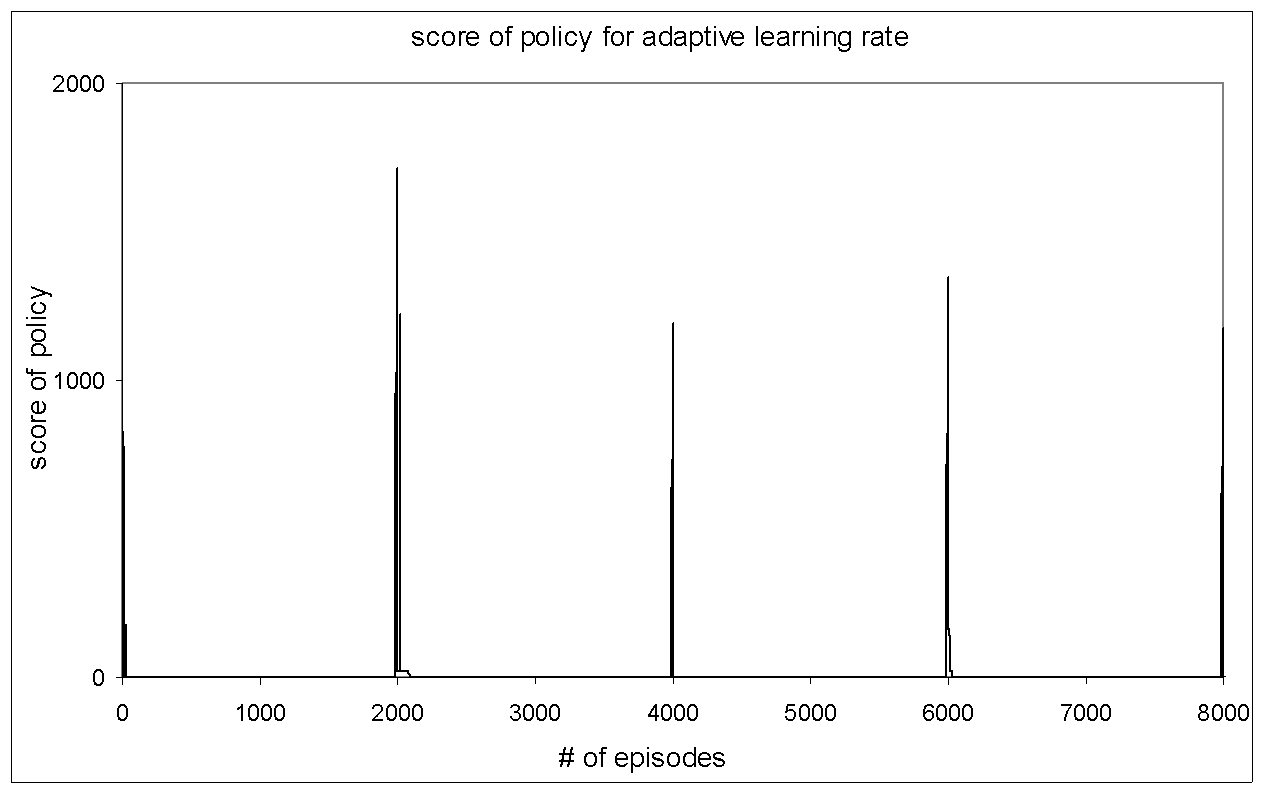
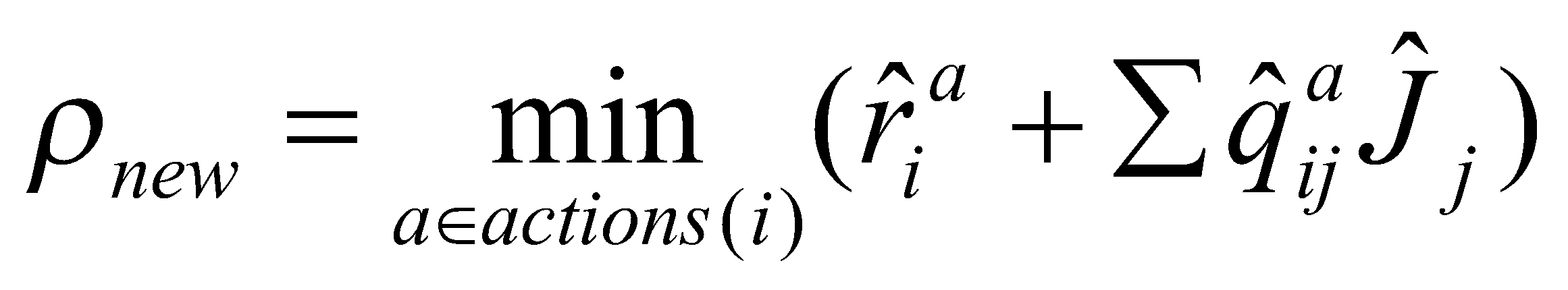
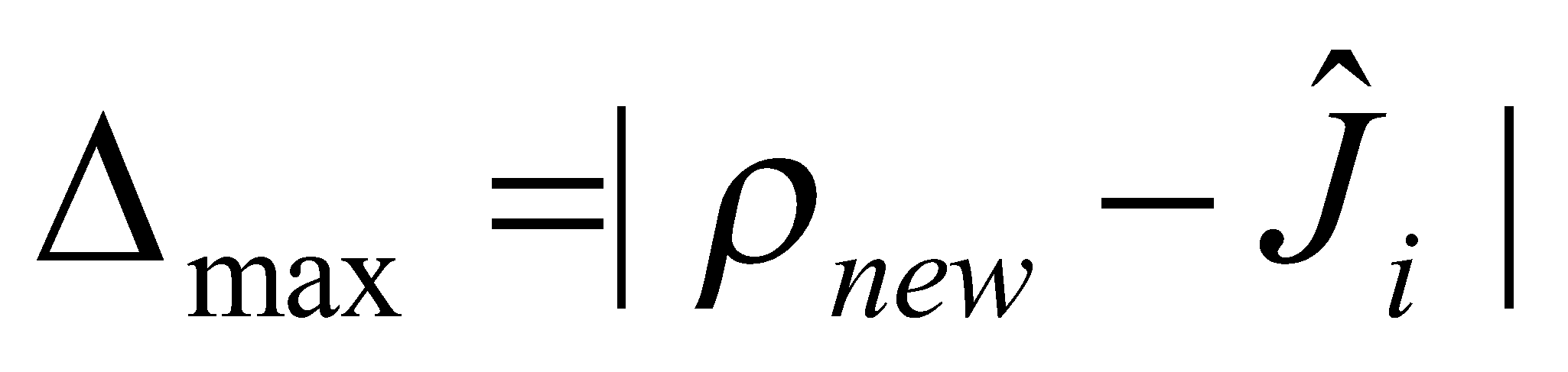
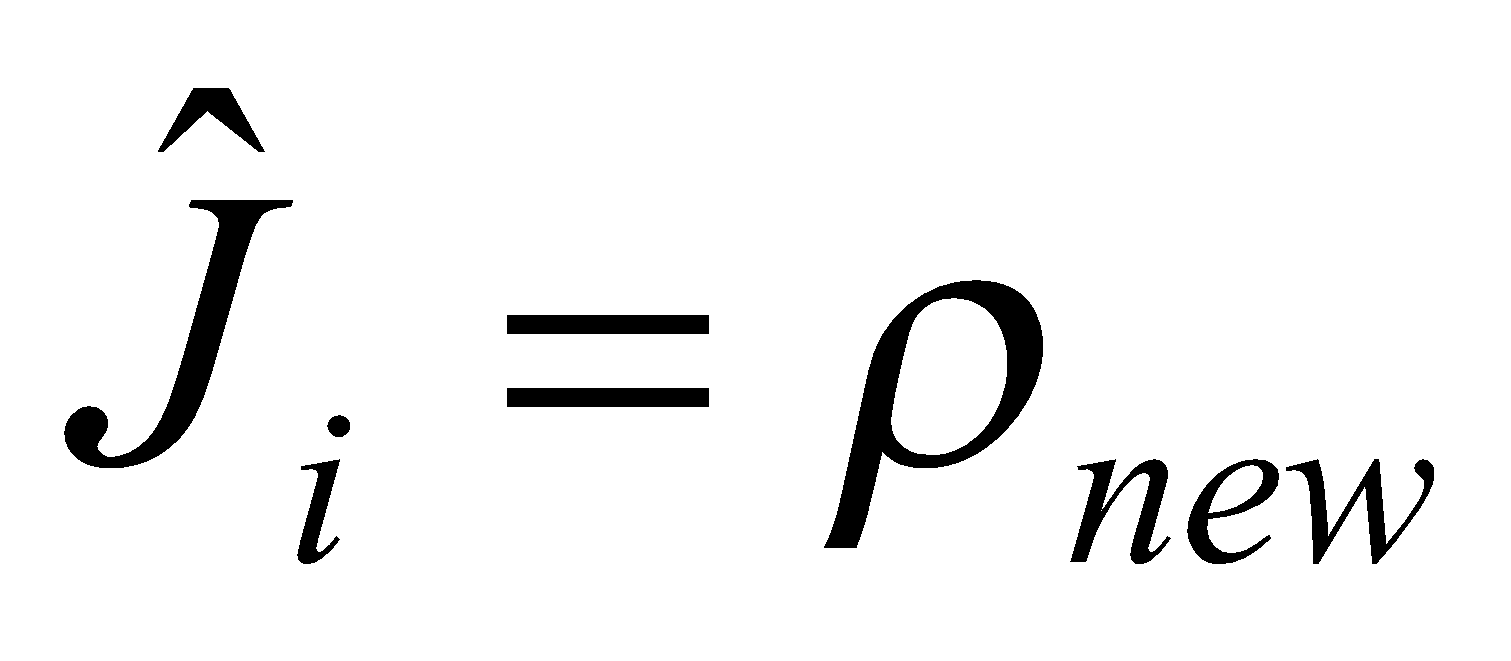
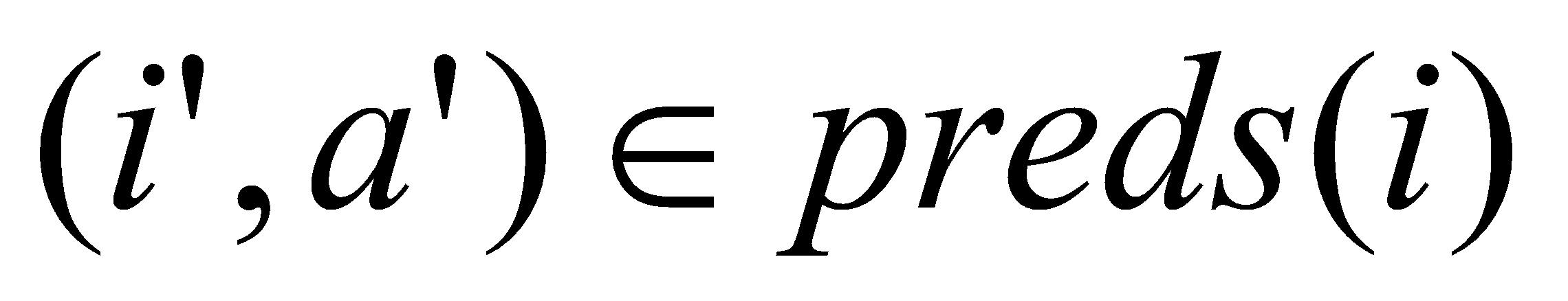
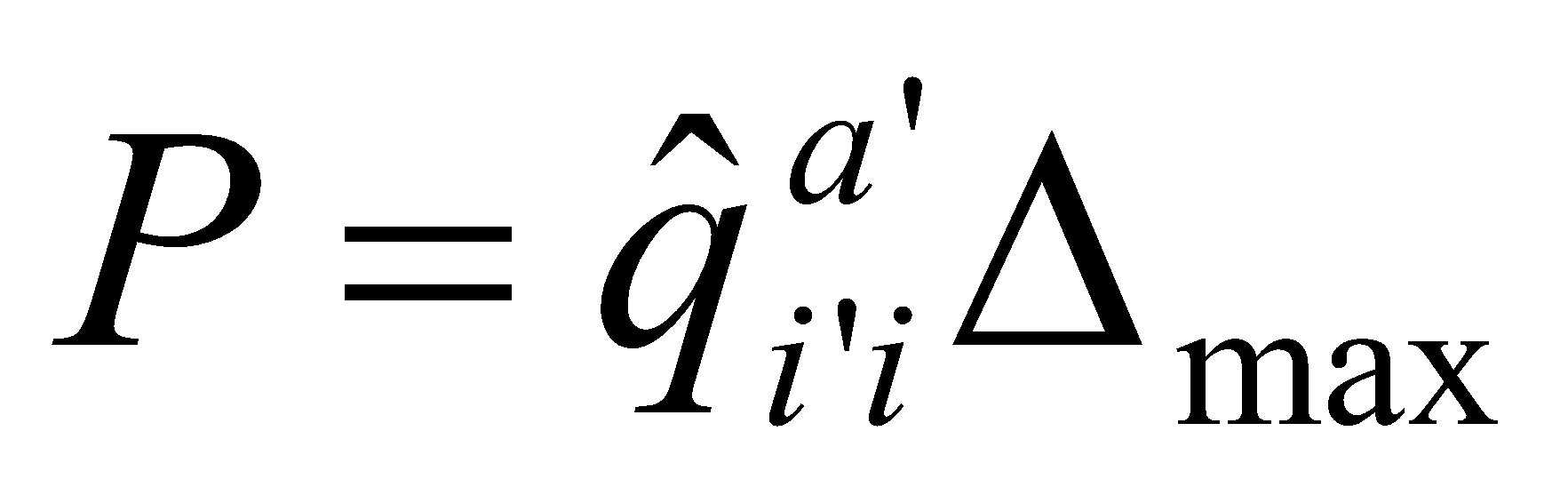
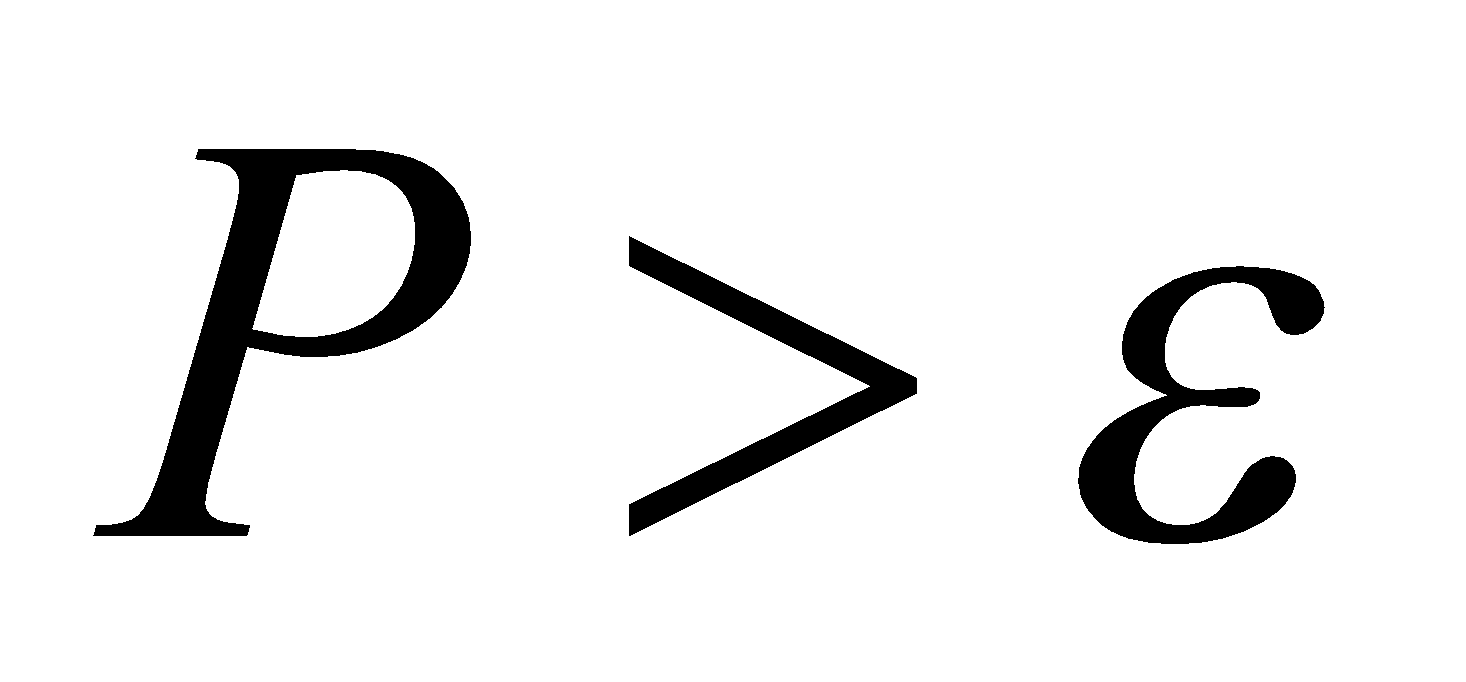
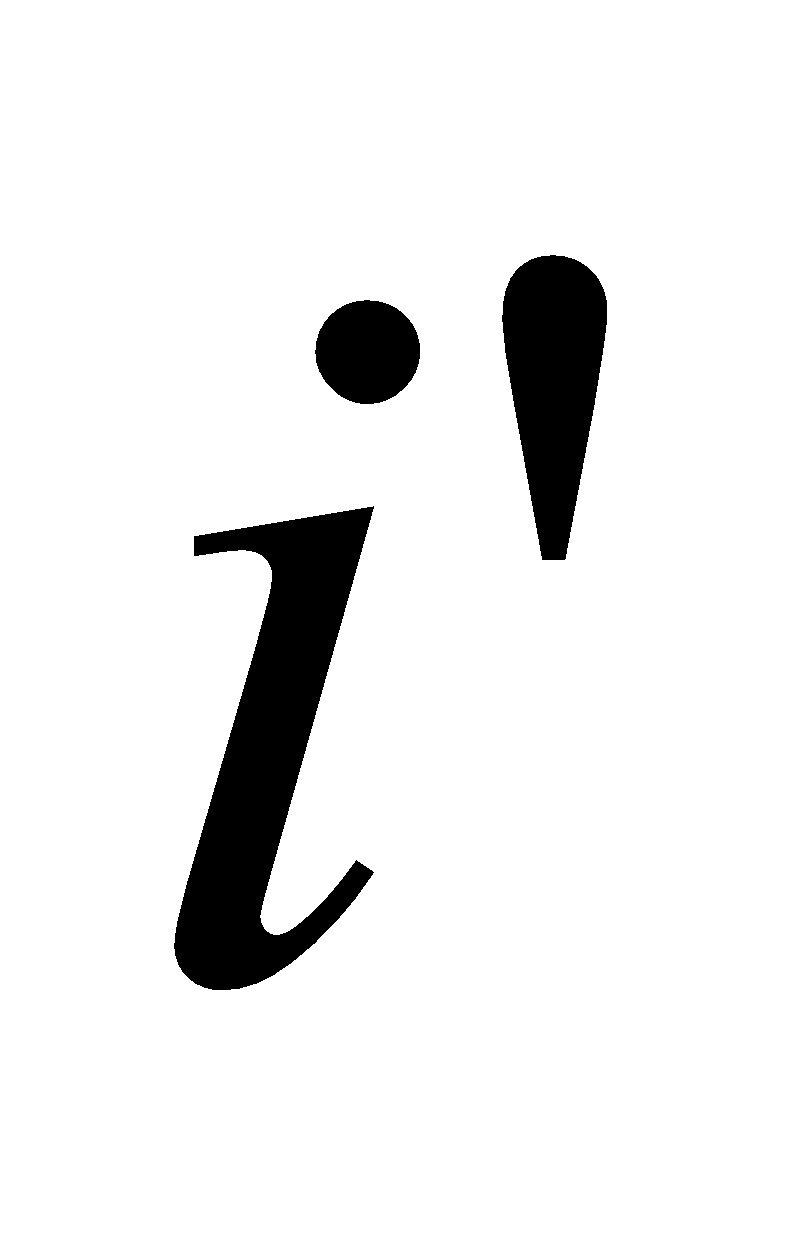
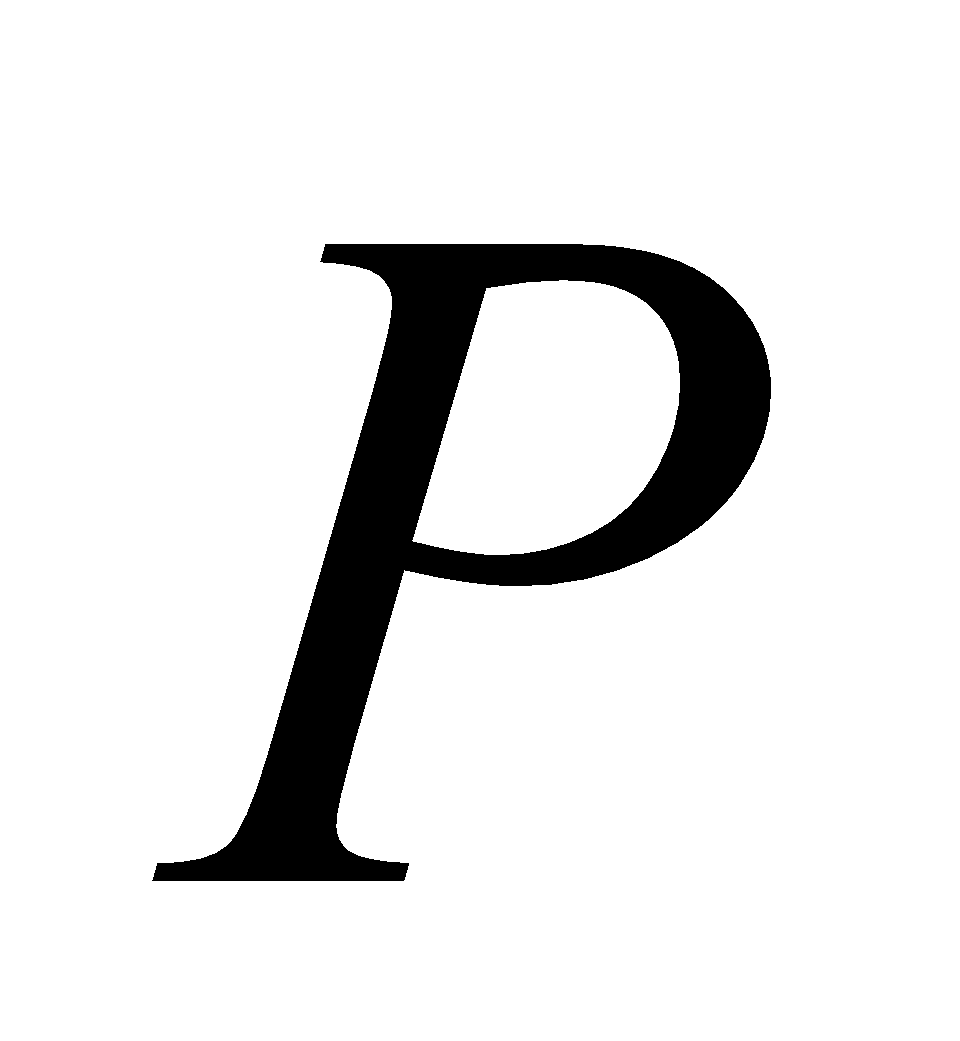
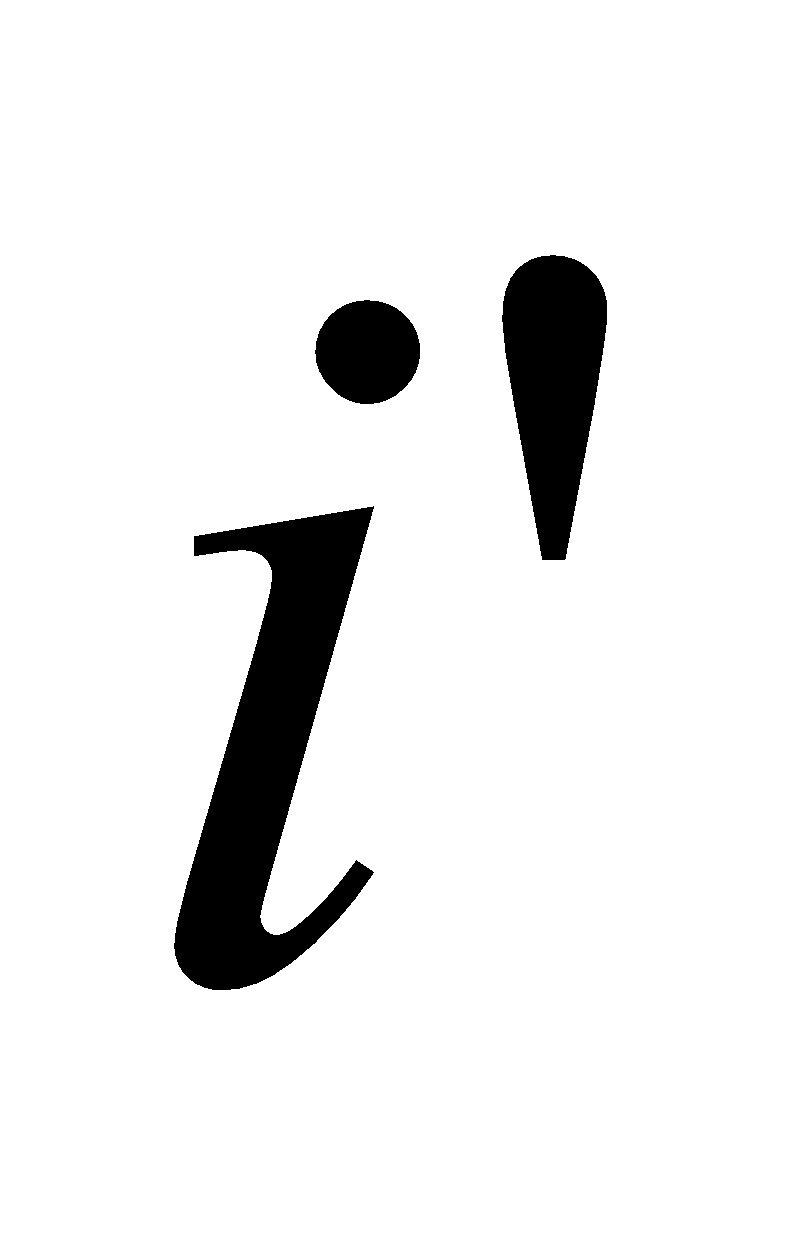
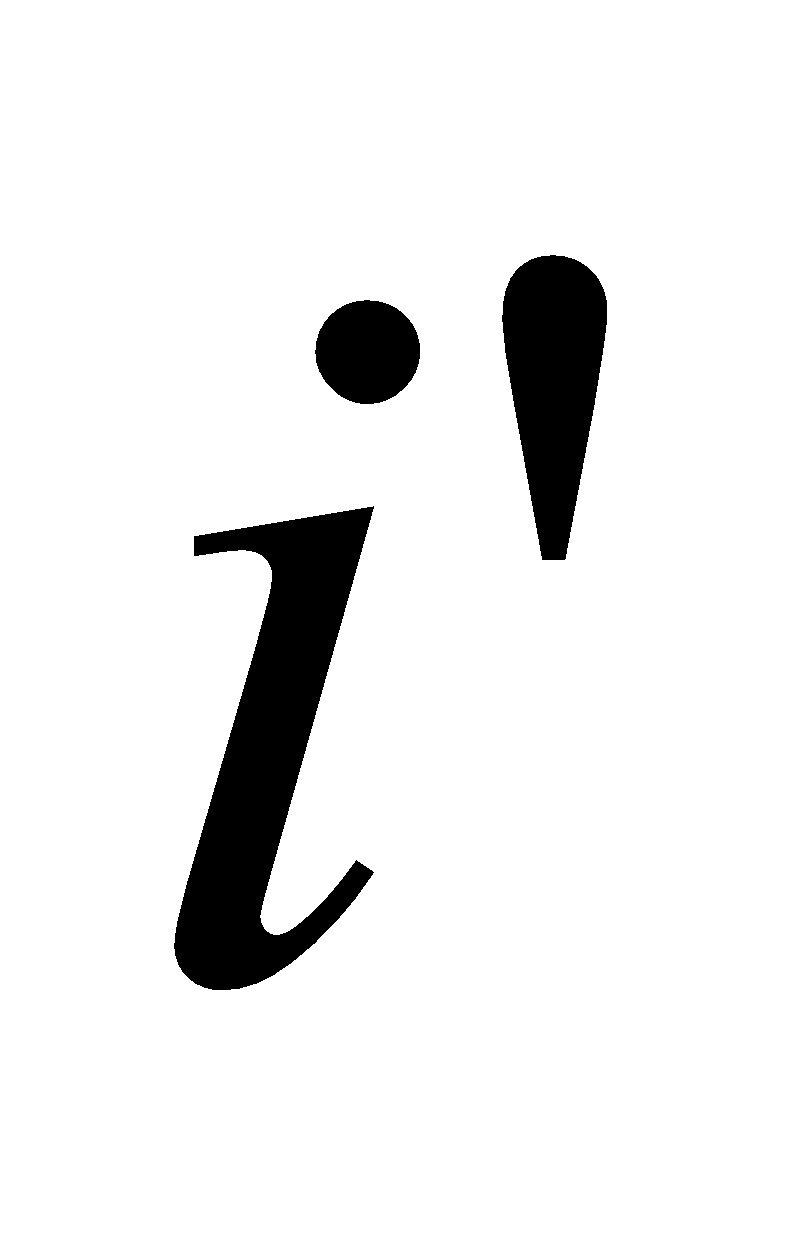
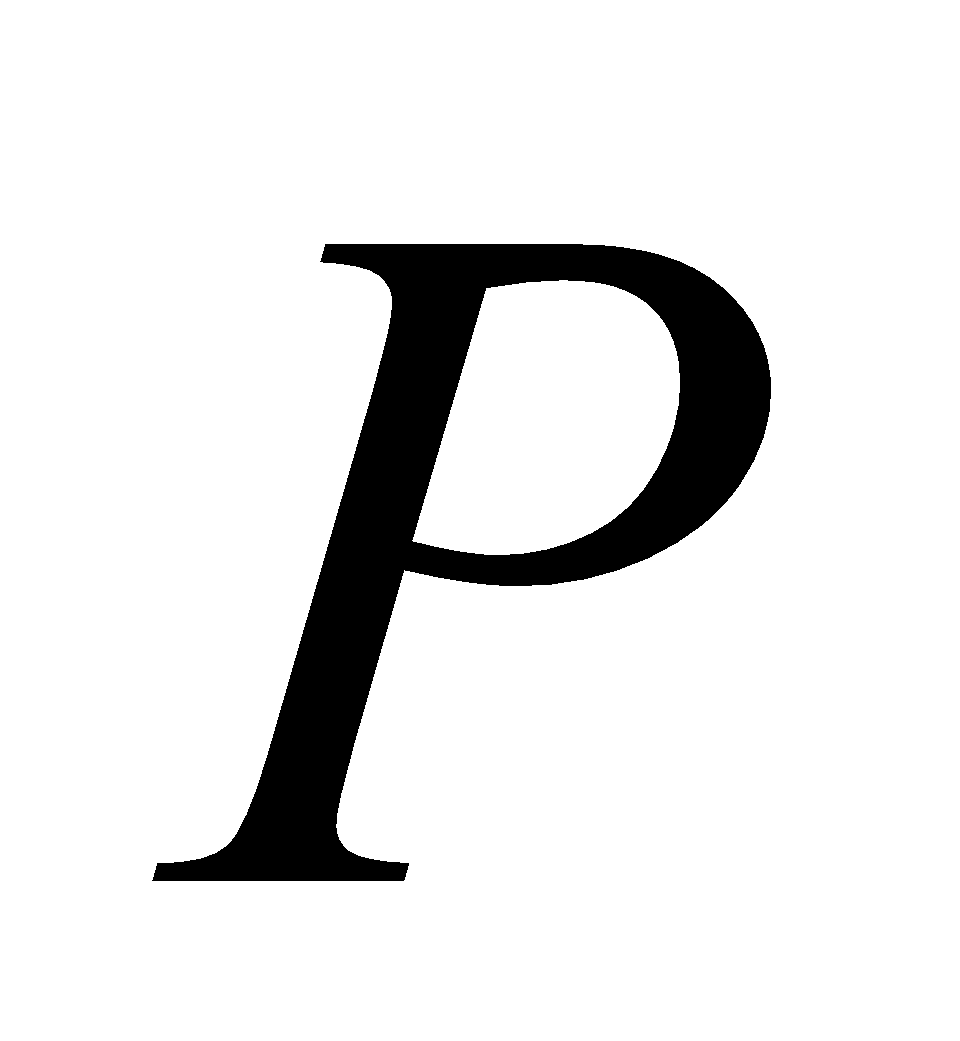


Figure 12: The score of the policy as the number of episodes increase for the adaptive learning rate scheme as applied to a non stationary environment.

# Prioritized sweeping [3] algorithm adapted to the experimental setup

1. *Promote state irecent to top of priority queue*
2. *While further processing is allowed and priority queue not empty*
   * 1. *Remove the top state from the priority queue*
     2. **
     3. **
     4. **
     5. *For each *

**

*If  (a tiny threshold) and if not on queue or exceeds the current priority of  then promote  to new priority *

Prioritized sweeping is a model based reinforcement learning algorithm. This means that the algorithm builds a model of the environment based on the agent’s interactions with the environment. As the agent’s interactions with the environment increase the model becomes more accurate. As part of the model the algorithm maintains for each state action pair in the state space the predecessor state-action pairs along with the associated transition probabilities, the estimated rewards and the successor states. Similar to the Q learning algorithm the prioritized sweeping algorithm also maintains for each state action pair a value which indicates its estimated cost to goal. Each time the agent performs an action it updates the value associated with the state action pair, the transition probability for that state action pair, the predecessors of that state and the successors associated for that action in that state. This enables the algorithm to propagate the effect of the interesting observations that the agent undergoes for a particular state action pair to the predecessors of that state. This is called as the process of making backups. Here an interesting observation occurs when a particular state action pair experiences significant change in the value associated with it. The significant change can be either because the agent transitions to a bad state or that the agent receives a very high positive reinforcement.

The process of making backups is induced whenever an interesting observation is made. But this can cause the algorithm to keep on making backups. So to avoid infinite backups the algorithm sets an upper limit on the number of backups allowed and processes only the most important states. This is achieved by prioritizing these interesting observations. The algorithm uses a priority queue and whenever an interesting observation is made the corresponding state is inserted in the priority queue. The priority of a state to be backed up is determined based on the amount of change that it experiences in its value. On occurrence of an interesting phenomenon the algorithm inserts the state in a priority queue with priority equal to the change in the expected cost to goal of that state. In the priority queue processing stage the algorithm updates the predecessors of those states which are present in the priority queue.

## *Optimal number of Backups*

An infinitely large number of backups results in large amount of computational effort to be spent in making the backups and algorithm spends very less time in actually interacting with the environment. At the same time a very small number of backups results in slower convergence of the algorithm. So there definitely exists an optimal value for the number of backups which optimizes the algorithm in two dimensions mainly the effort made to make the backups and the amount of interaction made by the agent with the environment.

To study the effect of number of backups on the performance of the algorithm experiments were conducted under the following conditions.

Maze = 45x45 (state space size = 2045)

PJOG = 0 (No noise)

Exploration policy: Epsilon-greedy with epsilon = 0.1

Values for number of backups = 1, 5, 25, 125, 625, 2045

The performance of the algorithm with respect to time is shown in Figure 13.

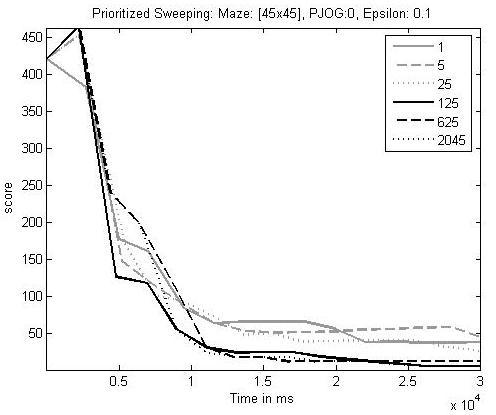


Figure 13: The score for the prioritized sweeping algorithm V/s the time taken for each episode for different values of number of backups.

From Figure 13 it can be observed that for a noiseless environment a very less value for the number of backups as compared to the size of the state space does not perform as well as a large value for the number of backups.

The above experiment was repeated for backup values 0, 405, 810, 1215, 1620 and the resulting error curve is as shown in the Figure 14.

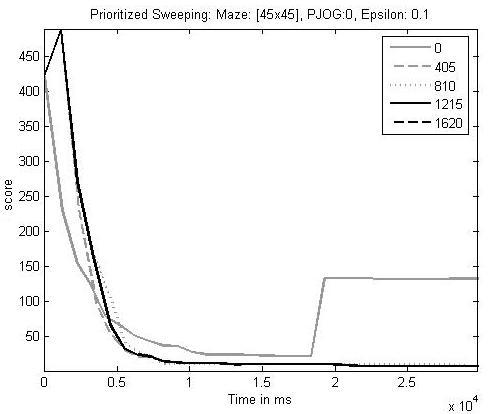


Figure 14: The score for the prioritized sweeping algorithm V/s the time taken for each episode for different values of number of backups.

It can be concluded from the results presented in Figures 13 and 14 that for a noiseless environment the performance of the algorithm improves as the value of the number of backups increases. But above a certain value there is no change in the performance of the algorithm. This can be attributed to the fact that since the environment is noiseless even though the number of backups increases the algorithm does not make full utilization of the number of backups provided.

The same experiment was performed for noisy environment under the following conditions.

Mazes = 10x10 and 45x45

PJOG = 0.3

Exploration policy: Changing start state and Epsilon-greedy with Epsilon = 0.1

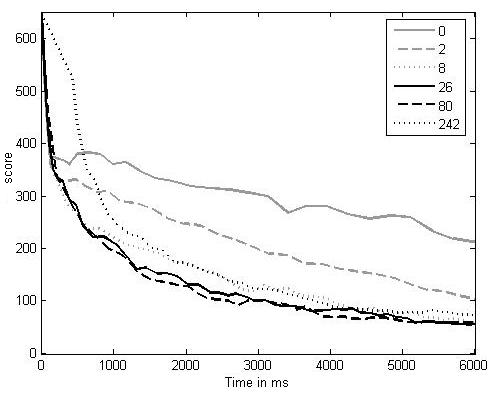


Figure 15: The score for the prioritized sweeping algorithm V/s the time taken for each episode for different values of number of backups.

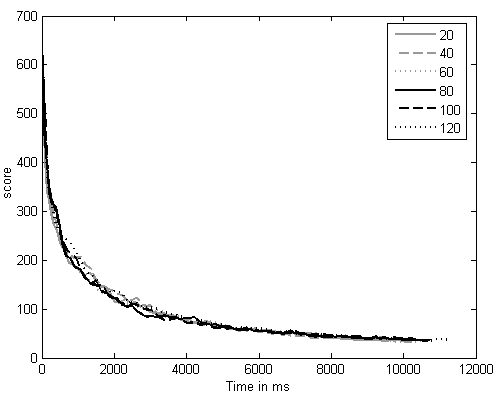


Figure 16: The score for the prioritized sweeping algorithm V/s the time taken for each episode for different values of number of backups.

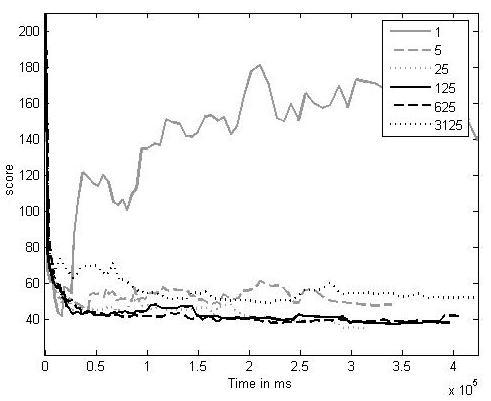


Figure 17: The score for the prioritized sweeping algorithm V/s the time taken for each episode for different values of number of backups.

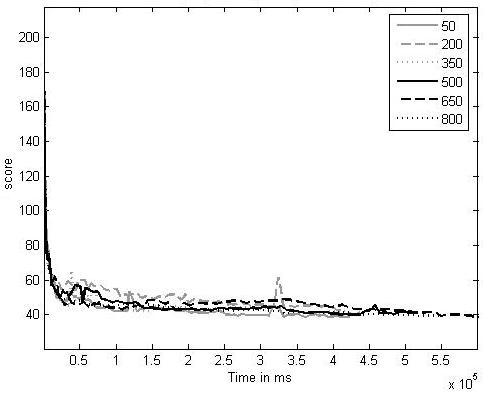


Figure 18: The score for the prioritized sweeping algorithm V/s the time taken for each episode for different values of number of backups.

Figures 15, 16, 17, 18 show the performance of the algorithm for different values of number of backups. It can be observed that performance of algorithm is not very good for very small number of backups (number of backups allowed = 0, 2 for maze size = 10x10 and number of backups allowed = 1, 5 for maze size = 45x45). Also the performance is not very good for very high number of backups (number of backups allowed = 242 for maze size = 10x10 and number of backups allowed = 3125 for maze size = 45x45). But there exists a band of intermediate values for the number of backups allowed which optimize the performance of the algorithm in both dimensions mainly the effort made to make the backups and the amount of interaction made by the agent with the environment.

# Conclusion

We have thus analyzed the reinforcement learning algorithms mainly value iteration, policy iteration, Q learning and prioritized sweeping. The learning rate certainly plays an important role in the convergence of the Q learning algorithm and that the adaptive learning rate scheme performs better than decaying learning rate scheme for Q learning in non stationary environments. For prioritized sweeping large number of backups or very small number of backups do not give optimal performance. But there exists a band of intermediate values for the number of backups allowed which gives optimal performance with respect to the time taken to converge and the computations required.

# Future work

From the preliminary results that have been obtained for prioritized sweeping there is scope of future work in the area of finding a mathematical expression for the optimal value of the number of backups for a given size of the state space.

1. Appendix

Source code and user manual for the test bed and the algorithms can be downloaded from

1. Acknowledgment

We would like to thank Professor Andrew Moore and Professor Mel Siegel for their advice and constant motivation for the project.

1. References
2. Sutton and Barto, “Reinforcement Learning”, The MIT Press, page 52-64, 122
3. Kaelbling, L.P., M.L., and Moore, A.W.(1996). Reinforcement Learning: A survey. *Journal of Artificial Intelligence Research*, 4:237-285
4. Moore, A.W., and Atkeson, C.G.(1993). Prioritized Sweeping: Reinforcement Learning with less data and less real time. *Machine Learning*, 13:103-130
5. http://www-2.cs.cmu.edu/~awm/tutorials/rl06.pdf - Tutorial Slides by Andrew Moore