VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELGAVI

Project Report on

APPLICATION OF REINFORCEMENT LEARNING ALGORITHMS TO OPTIMIZATION AND CONTROL PROBLEMS

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF ENGINEERING

In

ELECTRICAL AND ELECTRONICS ENGINEERING

By

SAVINAY NAGENDRA	**NIKHIL P V S**
1PI13EE082	**1PI13EE058**

Under the guidance of

Dr. Koshy George

PROFESSOR
DEPARTMENT OF TELECOMMUNICATION ENGINEERING
PESIT, BANGALORE – 560085

Carried out at

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
P E S INSTITUTE OF TECHNOLOGY
(Autonomous Institute under VTU, Belgavi)
100 Feet Ring Road, BSK III stage, Bangalore-560085

2017

VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELGAVI

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

PES INSTITUTE OF TECHNOLOGY

(Autonomous Institute under VTU, Belgaum), Bangalore - 560085

CERTIFICATE

*Certified that the final year project work entitled "**APPLICATIONS OF REINFORCEMENT LEARNING ALGORITHMS TO OPTIMIZATION AND CONTROL PROBLEMS**" is a bonafide work carried out by the team of students **SAVINAY NAGENDRA** and **NIKHIL P V S** in partial fulfillment for the award of degree of **Bachelor of Engineering in Electrical and Electronics** of P.E.S Institute of Technology, Bangalore, Autonomous Institute under Visvesvaraya Technological University, Belgavi during the academic year **2016-2017**. It is certified that all corrections/suggestions indicated during internal assessment have been incorporated in the report submitted in the departmental library. The project report has been approved as it satisfies the academic requirements with respect to the project work prescribed for the said degree.*

(SIGNATURE OF THE GUIDE)	(SIGNATURE OF HOD)	(SIGNATURE OF PRINCIPAL)
DR. KOSHY GEORGE	**DR. B K KESHAVAN**	**DR. K S SRIDHAR**
PROFESSOR	PROFESSOR AND HOD	PRINCIPAL, PESIT,
Dept. Of Telecommunication	Dept. Of Electrical and	BANGALORE
Engineering, PESIT,	Electronics Engineering,	
BANGALORE	PESIT, BANGALORE	

EXAMINER	**SIGNATURE**
1.	

DECLARATION

We, *Nikhil P V S (1PI13EE058), and Savinay Nagendra (1PI13EE082),* students of final semester B.E., Department of Electrical and Electronics Engineering at PES Institute of Technology, Bangalore-85, hereby declare that the project titled "Applications of Reinforcement Learning algorithms to Optimization and Control Problems" has been done and submitted in partial fulfillment of requirement of the award of the degree in Bachelor of Engineering in Electrical and Electronics Engineering of VISVESVARAYA TECHNOLOGICAL UNIVERSTIY, BELGAVI during the academic year 2016-17.

Place: Bangalore
Date:

TEAM MEMBERS:

Savinay Nagendra	Nikhil P V S
(1PI13EE082)	(1PI13EE058)

Acknowledgement

We would like to express our sincere gratitude to the Principal of PESIT, **Dr. K S Sridhar** for providing us with great facilities and opportunities, which has developed in us, a zest to learn.

We would like to thank the Head of the Department, Electrical and Electronics Engineering, **Dr. B K Keshavan** for supporting us throughout the course of this project.

We would like to express our gratitude to our project guide **Dr. Koshy George**, for the guidance and support throughout the course of this project.

We would also want to thank all our **teachers** for their help, guidance and the knowledge they have imparted, which has been used in various stages of this project

Finally, we would like to thank our **parents** and **friends** for their support.

Contents

List of Figures

List of Tables

Abstract

Reinforcement learning is a branch of machine learning inspired by behaviorist psychology, which is a systematic approach to study of human and animal behavior. It deals with how agents should behave and ought to take actions in an environment so as to maximize some notion of cumulative reward. Learning can be based on several forms of evaluative feedback. Standard Supervised learning methods require a training set of data consisting of input vectors and corresponding desired output vectors, i.e., in other words, they require a supervisory system. Such methods cannot be applied to tasks for which the desired output of the system is not known. These tasks can be solved using various Reinforcement learning algorithms. If the desired output is not available, the performance of the system must be evaluated indirectly by considering the effect of its output on the environment with which the system interacts. Reinforcement learning methods can be applied when this effect is measured by changes in an evaluation signal, or ***reinforcement***. Further, these algorithms focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

In machine learning, environment is typically formulated as a Markov Decision Process (MDP). An MDP can be either a discrete time or a continuous time stochastic process which constitutes the current state, policy (action to be taken), next state, reward and the transition probabilities. MDPs provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of the decision maker. The core problem of MDPs is to find a policy for the agent (decision maker) based on its current state. MDPs are useful for studying a wide range of optimization and control problems solved via reinforcement learning. Dynamic programming (DP) is a branch of reinforcement learning, which refers to a collection of algorithms that can be used to compute optimal policies, given a perfect model of the environment as an MDP. Dynamic programming algorithms are best suited for optimization problems, since these algorithms are designed such that they will examine the previously solved sub-problems and will combine their solutions to give the best solution for the given problem. But, classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense. Hence, dynamic programming algorithms are not preferred for control problems.

In complex non-linear systems, an accurate plant model is difficult to obtain. In such cases, model free reinforcement learning algorithms can be applied to achieve optimal control. Monte Carlo learning, Temporal Difference learning, Actor-Critic Policy Gradient methods are some of the examples for model free RL algorithms. While these algorithms deal with discrete state spaces, Value function approximation is applied to extend the RL algorithms to continuous state spaces.

In this report, we have explored various reinforcement learning algorithms by solving an optimization and a control problem. ***Jack's Car rental problem*** is chosen as the optimization problem which has been solved using Dynamic programming, and ***Cart Pole balancing problem*** is chosen as the non-linear control problem, with inherent instability, which has been solved using all the above mentioned discrete time and continuous time model free RL algorithms. The results and performance of various RL algorithms on the control problem have been compared with one another and with that of classical control techniques.

Chapter 1: Reinforcement Learning, an Introduction

Reinforcement learning problems involve learning how to map situations to actions so as to maximize a numerical reward signal. These problems are closed-loop because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. In most complex problems, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Thus, these are the three most important distinguishing features of a reinforcement learning problem:

1) Being closed-loop
2) Not having instructions as to what actions to take, and the consequences of these actions including reward signals.
3) Operate over extended time periods

Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. The objective of this kind of learning is for the system to extrapolate or generalize its responses so that it acts correctly in situations not present in the training set. Unsupervised learning deals with finding structure hidden in collections of unlabeled data. Reinforcement learning is therefore, a third machine learning paradigm that deals with a goal-directed agent interacting with an uncertain environment.

One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between ***exploration*** and ***exploitation*** [1]. To obtain maximum reward, the agent has to exploit what it already knows, but it also has to explore in order to make better action selections in the future. Neither exploration nor exploitation can be pursued exclusively without failing at the task. In case of a deterministic environment, the agent must explore by trying various actions in each state, and progressively learn to select the best action at each state. On the other hand, a stochastic environment requires repeated trials of the same action at each state in order to obtain an estimate of the amount of reward expected from that state. This dilemma between exploration and exploitation is one of the key features of reinforcement learning.

1.1 Elements of Reinforcement Learning

1.1.1 **Policy:** A ***policy*** defines the learning agent's way of behaving at a given time, i.e., a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to the stimulus - response rules of psychology

1.1.2 **Reward signal:** A ***reward signal*** defines the goal in a reinforcement learning problem. The sole objective of any Reinforcement Learning algorithm is to maximize the total reward it receives over the long run. The reward signal thus defines what the good and bad events are for the agent. In a biological system, rewards are analogous to the experiences of pleasure or pain.

1.1.3 **Value function:** A ***value function*** of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. The value function is estimated as a function of rewards by the RL algorithm. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow, and rewards available in those states.

1.1.4 **Model:** A ***model*** mimics the behavior of the environment, or more generally, allows inferences to be made about how the environment will behave. Models are used for planning, i.e., way of deciding on a course of action by considering possible future situations before they are actually experienced.

1.2 Finite Markov Decision Processes

Markov Decision Processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of the decision maker. MDPs are used to solve a wide range of optimization and control problems, solved using various reinforcement learning algorithms.

1.3 The Agent – Environment Interface

The reinforcement learning problem is a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision maker is called the ***agent***. The system it interacts with, comprising everything outside the agent, is called the ***environment***. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to ***rewards***, special numerical values that the agent tries to maximize over time. A complete specification of an environment, including how rewards are determined, defines a ***task***, one instance of the reinforcement learning problem.

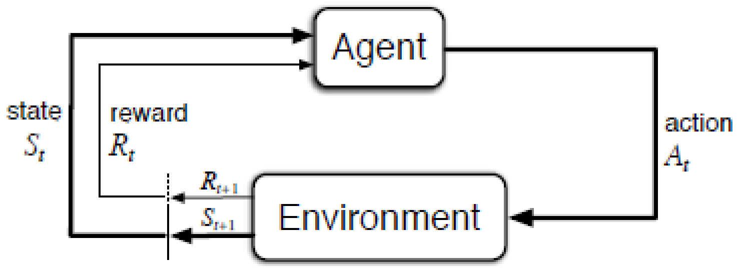


Fig. 1.1: The agent – environment interaction in reinforcement learning

The agent and environment interact at discrete time steps, t = 0, 1, 2, 3… At each time step t, the agent receives some representation of the environment's ***state***, $S_t \in S,$ where S is the set of possible states, and on that basis selects an ***action***, $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions available in state S_t. One time step later, as a consequence of its action, the agent receives a numerical ***reward***, $R_{t+1} \in R$ and finds itself in a new state S_{t+1}. Figure 1 shows the agent – environment interaction.

At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's ***policy*** denoted as π_t, where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal is to maximize the total amount of reward it receives over the long run.

This framework is abstract and flexible and can be applied to many different problems in many different ways. The same frame work of MDP has been implemented in the optimization and control problems that have been solved in the later stages of this report.

1.4 Returns

The agent's goal is to maximize the cumulative reward it receives in the long run. If the sequence of rewards received after time step t is denoted as R_{t+1}, R_{t+2}, R_{t+3} ,…, then what precise aspect of this sequence do we maximize? In general, the ***expected return*** is maximized over time. If G_t is defined as some specific function of the reward sequence, in the simplest case, the return is the sum of rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \tag{1.1}$$

where T is a final time step. Each final time step marks the end of an ***episode***, and each episode ends in a special state called the ***terminal state***. The next episode begins from a standard pre-defined starting state, or randomly. These tasks are called ***episodic tasks***. On the other hand, in many cases the agent-environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. These tasks are called ***continuing tasks***. Various reinforcement learning algorithms have been used to solve the control problem as an episodic task in this report, and their results have been compared.

The additional concept that we need is ***discounting***. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses A_t to maximize the expected ***discounted return***:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,, \tag{1.2}$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the ***discount rate***.

The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately.

1.5 The Markov Property

Ideally, a state signal that summarizes past sensations compactly, yet in such a way that all relevant information is retained is preferred in a reinforcement learning problem. Such a signal that succeeds in retaining all relevant information is said to be ***Markov***, or said to have ***Markov property***.

Consider how a general environment might respond at time *t+1* to the action taken at time *t*. In the most general, causal case, this response may depend on every -thing that has happened earlier. In this case, the dynamics can be defined only by specifying the complete joint probability distribution:

$$\Pr\{S_{t+1} = s', R_{t+1} = r \mid S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}, \tag{1.3}$$

for all r, s', and all possible values of the past events: $S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t$. A state signal is said to possess the Markov property if the next state and reward obtained through the environment dynamics received by the agent at time t+1, depend only on the state of the system and the action taken by the RL agent at time t. The agent can maintain an estimate of the inherent environment dynamics through State Transition Probabilities, which can be defined by:

$$p(s',r|s,a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}, \tag{1.4}$$

For all $r, s', s\ and\ a$.

If an environment has the Markov property, then its one-step dynamics (1.4) enable us to predict the next state and expected next reward given the current state and action. One can show that, by iterating this equation, one can predict all future states and expected rewards from knowledge only of the current state as well as would be possible given the complete history up to the current time. It also follows that Markov states provide the best possible basis for choosing actions. That is, the best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete histories.

1.6 Value Functions

Almost all reinforcement learning algorithms involve estimating ***value functions***, which are functions of states or of state-action pairs that estimate ***how good*** it is for the agent to be in a given state. The notion of 'how good' is defined in terms of future rewards that can be expected, i.e., in terms of expected return. The ***value*** of a state s under a policy π, denoted as $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s] \tag{1.5}$$

v_π is called the ***state-value*** for policy π.

Similarly, $q_\pi(s,a)$ is defined as the value of taking action a in state s under a policy π.

$$q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a] = E_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \tag{1.6}$$

q_π is called the ***action-value*** function for policy π.

1.7 Bellman Equation

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy particular recursive relationships. For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

$$= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s']\Big]$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma v_\pi(s')], \quad \forall s \in S \qquad (1.7)$$

Equation (1.7) is the ***Bellman equation*** for v_π. It expresses a relationship between the value of a state and the values of its successor states. The value function v_π is the unique solution to its Bellman equation. Bellman equation forms the basis of a number of ways to compute, approximate, and learn v_π using various reinforcement learning algorithms.

1.8 Organization of the Report

This report is organized as follows. The basic terminology and the fundamental equations are described in sections 1.1 to 1.7. Chapter 2 explains Multi-armed Bandits, a Reinforcement Learning problem to choose actions without any explicit states. In Chapter 3, Dynamic Programming as a planning method is described and an optimization problem known as the Jack's car rental problem is defined and solved using DP method. Chapter 4 introduces the Cart-Pole balancing problem and defines the environment of the problem. Chapter 5 explores various Model-Free Reinforcement Learning algorithms in reference to the Cart-Pole balancing problem and describes the results of RL algorithms applied on the problem. In Chapter 6, the Cart-Pole balancing problem is solved by Swing-up and stabilization using classical control techniques. Chapter 7 integrates the Swing-up method with Reinforcement Learning algorithms, and compares results with that obtained in Chapter 6. Finally, Chapter 8 and Chapter 9 cover the conclusion of the report and future work to be performed.

Chapter 2: Reinforcement Learning - Multi Armed Bandits

The Multi armed Bandit is an optimization problem which defines the fundamental problem types solved by Reinforcement Learning algorithms.

The Multi Armed Bandit problem or the K-armed Bandit problem is defined as: Consider k different choices or options available. When any of the k options is selected, certain amount of reward is obtained which depends on a probability distribution whose parameters vary among the different options. The aim of the problem is to choose among the k options repeatedly in order to collect the maximum amount of rewards.

Designing an algorithm to solve the Multi Armed Bandits problem forms the basis for all algorithms in Reinforcement Learning as all these algorithms rely on certain rewards for taking actions or selecting options.

Here, we define multiple terms relating to the entire system of Multi Armed Bandits.

The Bandit, on which the selection is made and the stochastic reward is generated, is a part of the Environment. All the K-bandits together make up the entire environment.

The entity which strives to achieve the aim, by taking actions or selecting the bandits is the Agent.

In Multi Armed Bandits, and in all Reinforcement Learning algorithms, there exist an Agent and an Environment. Both the Agent and the Environment interact and communicate using various stimuli and signals in order for the Agent to achieve its goal in the environment. In the case of the Multi Armed Bandits, two such signals are used for the interaction between the Agent and the Environment:

1. Action (denoted by a): The action is a signal in the Multi Armed Bandits problem for the Agent to stimulate the Environment and select an option or choice.
2. Reinforcement (denoted by R): The reinforcement is a signal in the Multi Armed Bandits problem for the Environment to communicate to the agent regarding the amount of rewards obtained by selecting the option.

As the aim of the Agent is to maximize its rewards over repeated selection of the k different options, it must use the Reinforcement signal obtained and remember which action has resulted in the largest sum of Reinforcement values. From this, it can be derived that taking one particular action will result in Reinforcement approximately defined by the average of all the rewards that the Agent has received when it took the action:

$$Approximate\ Reinforcement = \frac{\sum_{i=1}^{n} R_i}{n} \quad (2.1)$$

where R_i is the Reinforcement received by the Agent in selecting that particular option for the ith time. n is the total number of times the Agent has selected the option.

Thus, the Agent must have memory to store the sum or average Reinforcement received by taking each action. This memory element, which contains all the sums in the context of the k-Armed Bandits problem, is known as the Action Value function. The action value function is defined as follows:

$$Q_t(a) = \mathbf{E}[R_t |\, A_t = a] = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}(A_i = a)}{\sum_{i=1}^{t-1} \mathbf{1}(A_i = a)} \quad (2.2)$$

where $\mathbf{E}[R_t |\, A_t = a]$ denotes the Expectation of all the reinforcements obtained until time t given that action ‘a’ was taken when that reinforcement was obtained. R_i is the Reinforcement obtained at time instant i $\mathbf{1}(A_i = a)$ is the qualifier that action a was taken at time instant i

Now, that the Action Value Function has been defined, the Agent will use this function to determine the next action that it should take, in order to maximize its expected reward at every repetition of the experiment. Thus, the Agent must choose an action that resulted in maximum average reward in the Action value function. This method of choosing the action that corresponds to the maximum action value is known as a Greedy policy, and is given by:

$$A_t = \text{argmax}_a\, Q_t(a) \quad (2.3)$$

This is known as a greedy policy as the Agent greedily chooses the best action, without considering any other action. Following the greedy policy can result in a large error both in choosing the actions as well as updating the Action value function in certain situations. Consider a case when the entire action value estimate is unknown at the beginning of an experiment, and has been initialized to 0 for all k options. In the first trial, when an action “m” is ambiguously selected using the greedy policy, the Agent receives a positive Reinforcement to update its value estimate for the action “m”, and all other action value estimates remain 0. From the second trial, the action “m” will always be selected by the greedy policy as only the action value estimate of the action “m” is greater than 0. This prevents exploration of all other actions, even though there might exist some action “p” which always provides a Reinforcement signal higher than that provided by action “m”.

To prevent this and many such situations resulting from acting greedily (also known as Exploiting) on the Action value functions, other policies for selecting actions must be considered. The significant features of these policies which do not resort to just Exploitation, is that they must select actions other than the best actions at least sometimes. This is known as Exploration. On the other hand, pure exploration can prevent the Agent from achieving its goal as the best actions would not be selected and in turn, the maximum reward would not be obtained. One policy that deals with the trade-off between Exploration and Exploitation is known as the ε-greedy policy. The ε-greedy policy chooses an action among the set of available actions with a probability of ε and acts greedily with a probability of (1-ε). Thus, using this policy, the Multi Armed Bandits problem can be solved in an efficient manner, with the agent receiving maximum rewards.

Chapter 3: Dynamic Programming

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP). Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically. DP provides an essential foundation for the understanding of various reinforcement learning algorithms. In fact, all of these algorithms can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment. The key idea of DP is to use value functions to organize and structure the search for good policies. DP algorithms are obtained by turning Bellman equations into assignments, i.e., into update rules for improving approximations of the desired value functions.

3.1 Jack’s Car Rental Optimization problem

In the course of exploring and understanding the algorithms of dynamic programming, Jack’s car rental problem is chosen as the optimization problem as defined in [1], and is solved using the ***Policy Iteration*** algorithm.

3.1.1 Problem Statement

Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved.

It is assumed that cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is n is $\frac{\lambda^n}{n!}e^{-\lambda}$, where λ is the expected number. It is also assumed that there can be no more than 20 cars at each location and additional cars are returned to the nationwide company, and thus disappear from the problem. A maximum of five cars can be moved from one location to the other in one night.

3.1.2 Formulation of MDP

State: Number of cars at each location at the end of the day.

Actions: Net number of cars moved between the two locations overnight.

Rewards: +10$ for every rental, and -2$ for every transfer

Time steps: Number of days

Discount rate (γ): 0.9

λ, which is the expected number, is defined as the mode (highest frequency of occurrence) of the number of requests or number of returns taken over a fixed number of days. Since there are 2 locations, there are four λ values:

$\lambda_{rental}^{first\ location} = 3$

$\lambda_{rental}^{second\ location} = 4$

$\lambda_{return}^{first\ location} = 3$

$\lambda_{return}^{second\ location} = 2$

The goal of the solution is to optimize the number of transfers between the two locations to maximize the total money Jack receives at the end of each day.

Since the expected number of returns at the second location is considerably lesser than the number of rental requests, the optimized solution of the problem must include more number of transfers from first location to second location.

3.2 Policy Iteration applied to Optimization problem

This algorithm has two parts which operate alternatively:

3.2.1 Policy Evaluation

Policy evaluation is used to compute the state-value function v_π for an arbitrary policy π. Consider the Bellman's equation (1.7). If the dynamics of the environment are completely known, then (1.7) is a system of $|S|$ simultaneous linear equations in $|S|$ unknowns. In principle, its solution is a straightforward, tedious computation. Hence, iterative solution methods are most suitable, and have been used to solve the optimization problem.

Consider a sequence of approximate value functions $v_0, v_1, v_2, \ldots$ The initial approximation v_0 is chosen arbitrarily, and each successive approximation is obtained by using the Bellman equation as an update rule:

$$v_{k+1}(s) = \sum_a \pi\,(a|s) \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_k(s')] \qquad (3.1)$$

This algorithm is called ***iterative policy evaluation***. To produce each successive approximation, v_{k+1} from v_k, iterative policy evaluation applies the same operation to each state s: it replaces the old value of s with a new value obtained from the old values of the successor states of s, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated. This kind of operation is called ***full backup***. This means that all the next states from a state are explored, which is the reason for computational expense.

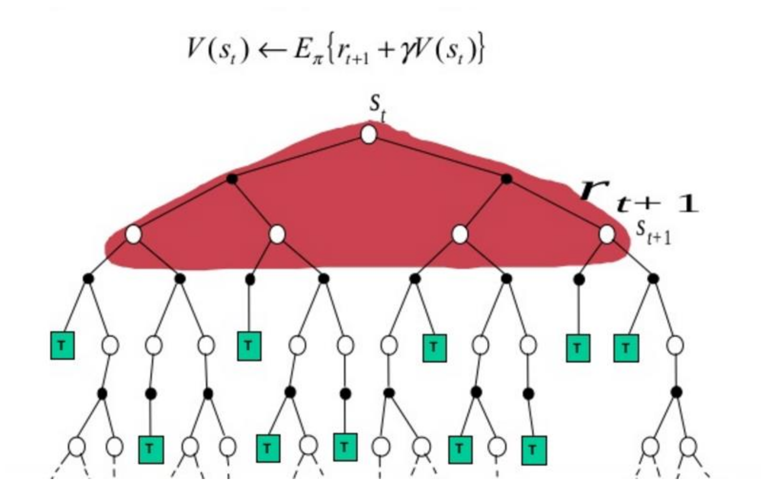


Fig. 3.1: Iterative policy evaluation full backup diagram

Iterative policy evaluation

```
Input $\pi$, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$
```

Fig. 3.2: Algorithm for Iterative policy iteration

3.2.2 Policy Improvement

The reason for computing the value function for a policy is to help find better policies. Suppose a value function v_π has been determined for an arbitrary deterministic policy π, for some state s, it is necessary to know whether or not a change of policy is required to deterministically choose an action $a \neq \pi(s)$. One method to change the policy will be to select a in s and thereafter follow the existing policy π. The value of this way of behaving is

$$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a \quad (3.2)$$

$$= \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_\pi(s')]$$

The key criterion is whether this value is greater than or less than $v_\pi(s)$. If it is greater, i.e., if it is better to select a once in s and thereafter follow π than it would be to follow π all the time, then it is better to select a every time s is encountered, and the new policy will in fact be a better one overall. That this is true is a special case of a general result called the ***policy improvement theorem***.

Let π and π' be any pair of deterministic policies during the course of a policy improvement, such that

$$q_\pi(s, \pi'(\mathrm{s})\,) \geq v_\pi(s) \quad (3.3)$$

Then, the policy π' must be as good as, or better than π, i.e., it must obtain greater or equal expected return from all states.

$$v_{\pi'}(s) \geq v_\pi(s) \quad (3.4)$$

Instead of considering a change in the policy at a single state to a particular action, it is computationally less expensive to consider changes at all states to all possible actions, selecting at each state the action that appears best according to $q_\pi(s,a)$. In other words, a new ***greedy*** policy is obtained, which is given by

$$\pi'(\mathrm{s}) = \underset{\mathrm{a}}{\mathrm{argmax}} \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_\pi(s')]\,, \quad (3.5)$$

where $argmax_a$ denotes the value of a at which the expression is maximized. The greedy policy takes the action that looks best in the short term, after one step look -ahead. The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called ***policy improvement.***

Then, $v_{\pi'} = v_\pi$, and from equation (2.12),

$$v_{\pi'}(s) = \max_{\mathrm{a}} \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_\pi(s')] \quad (3.6)$$

This is called Bellman optimality equation when $v_{\pi'} = v_*$ and both π and π' are optimal policies. Thus, Policy improvement strictly gives better policy in each iteration, except when the original policy is already optimal.

3.2.3 Policy Iteration

Once a policy π has been improved using v_π to yield a better policy π', we can then compute $v_{\pi'}$ and improve it again to yield an even better π'' [1]. We can thus obtain a sequence of monotonically improving policies and value functions.

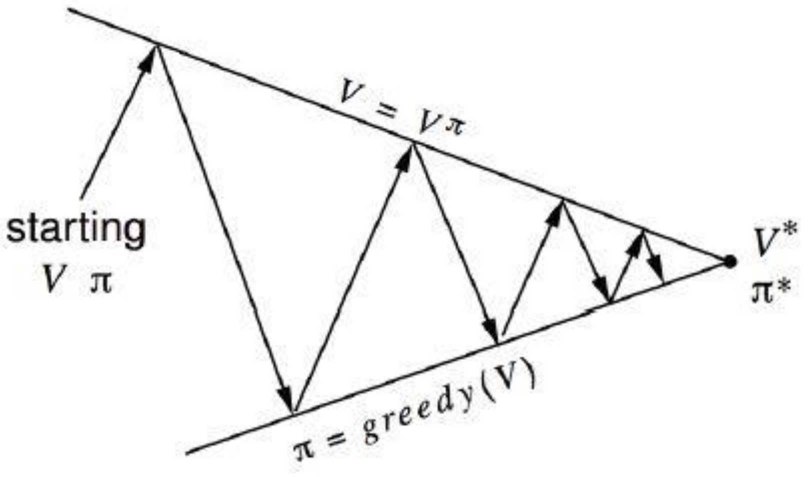
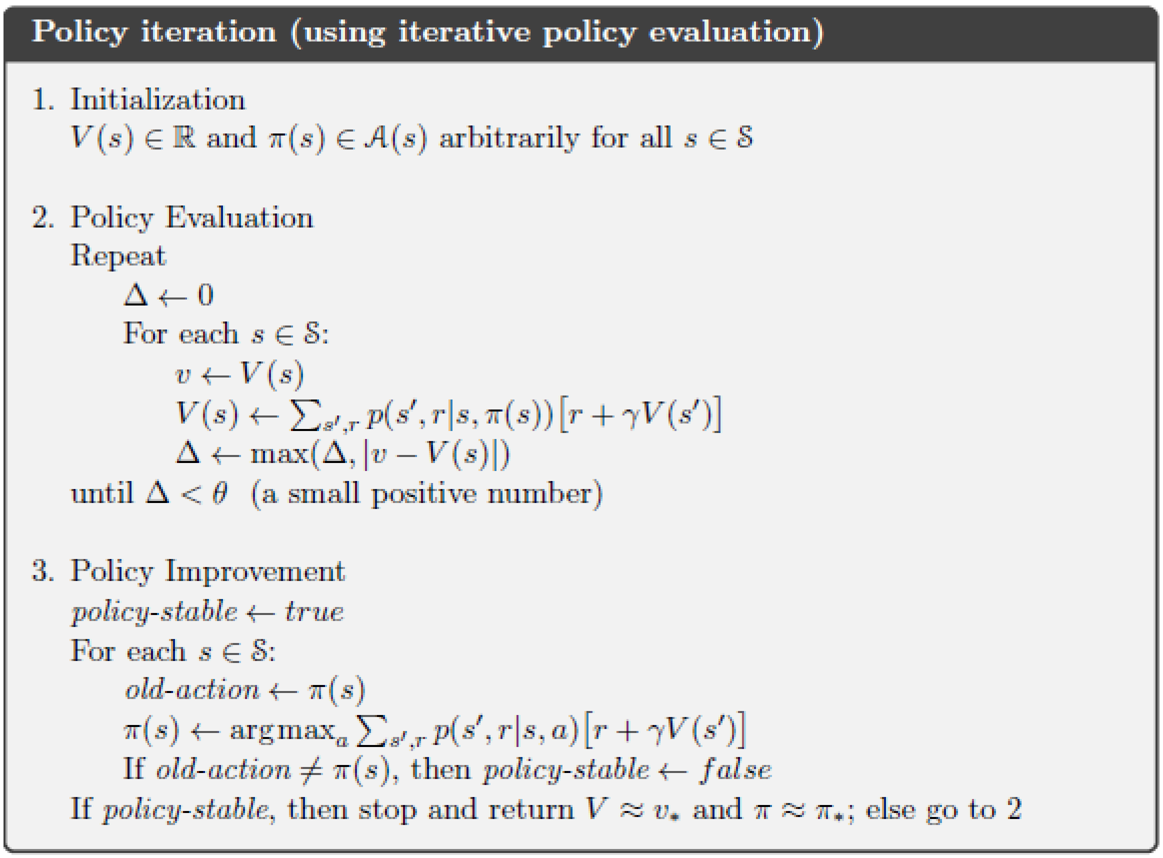


Fig. 3.3: Policy Iteration: Policy evaluation and policy improvement occurs alternatively



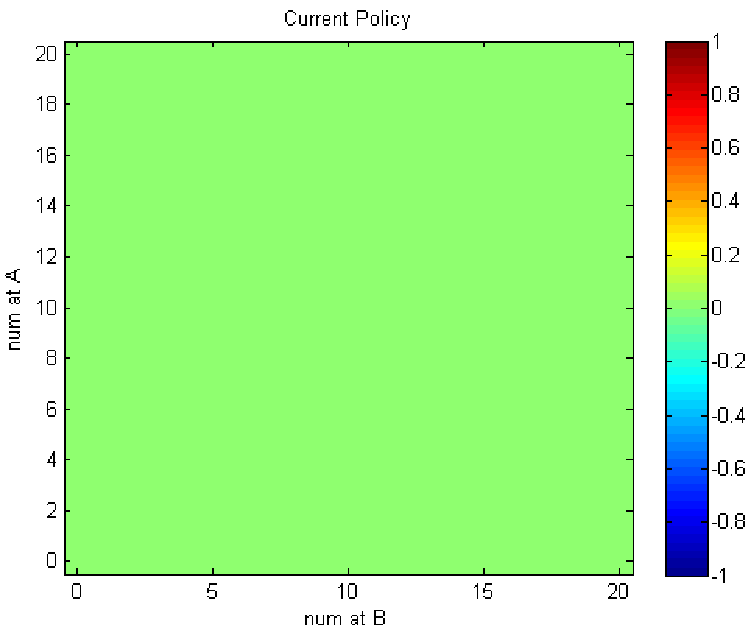
```
Policy iteration (using iterative policy evaluation)

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
       $\Delta \leftarrow 0$
       For each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

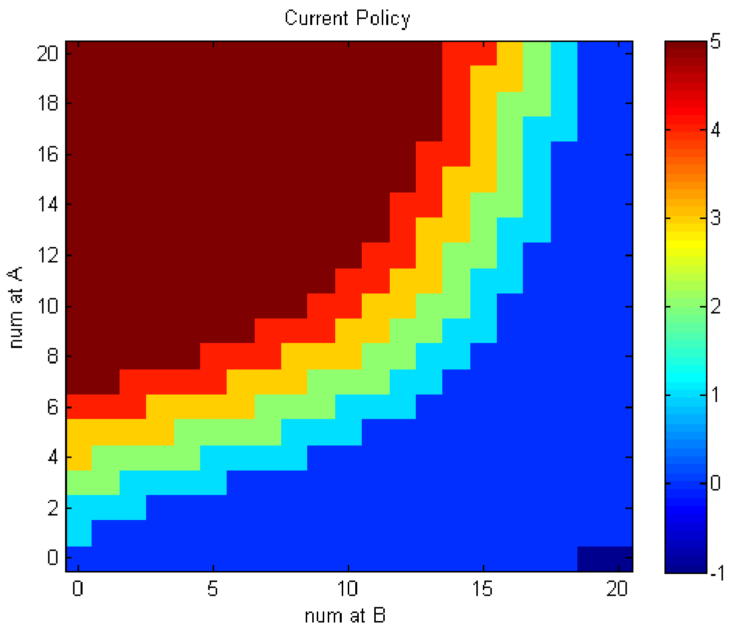
Fig. 3.4: Policy iteration algorithm

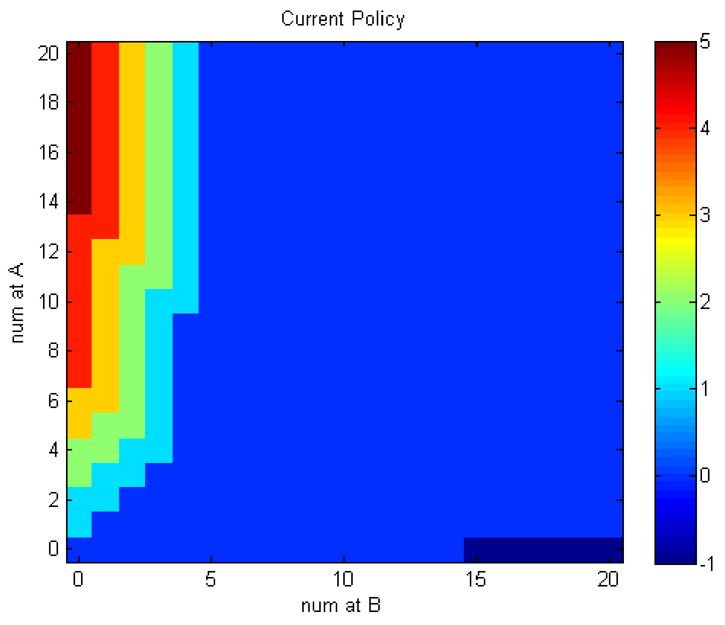
3.3 Results

Policy iteration algorithm is applied to solve the MDP of Jack's Car Rental optimization problem to maximize the value function, i.e., maximize the reward Jack gets at the end of each day till an optimal policy and optimal value function have been calculated. This is achieved by optimizing the number of car transfers between the two locations.

Policy Improvement surface plots:







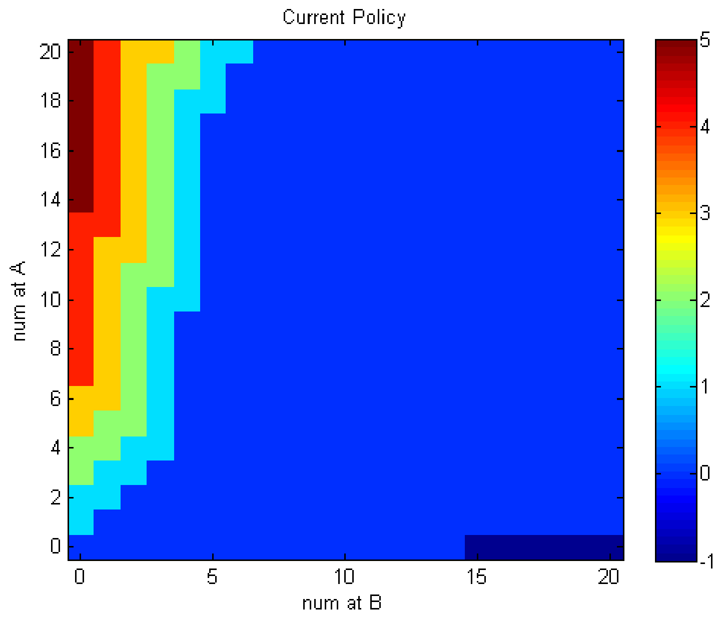
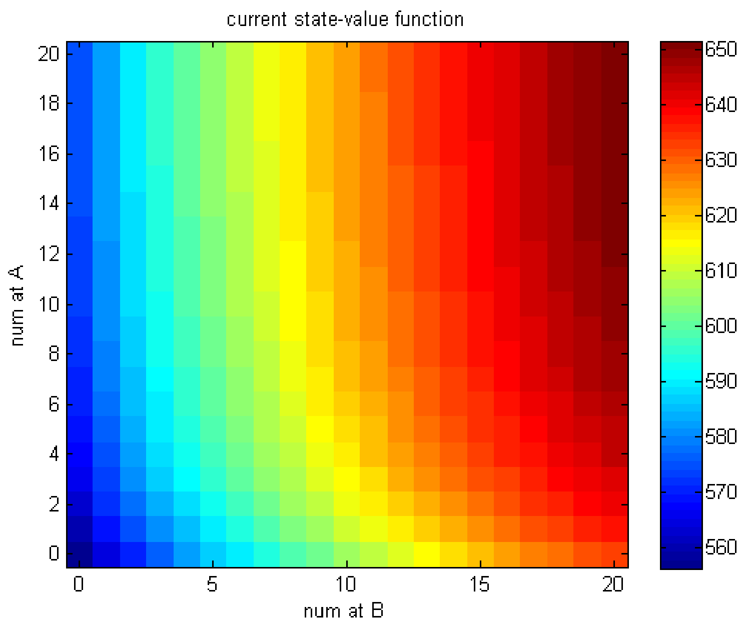
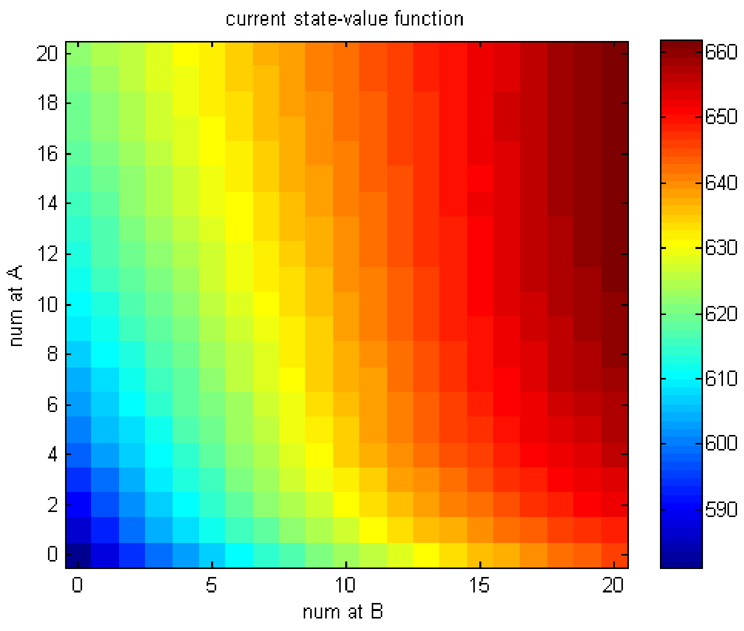
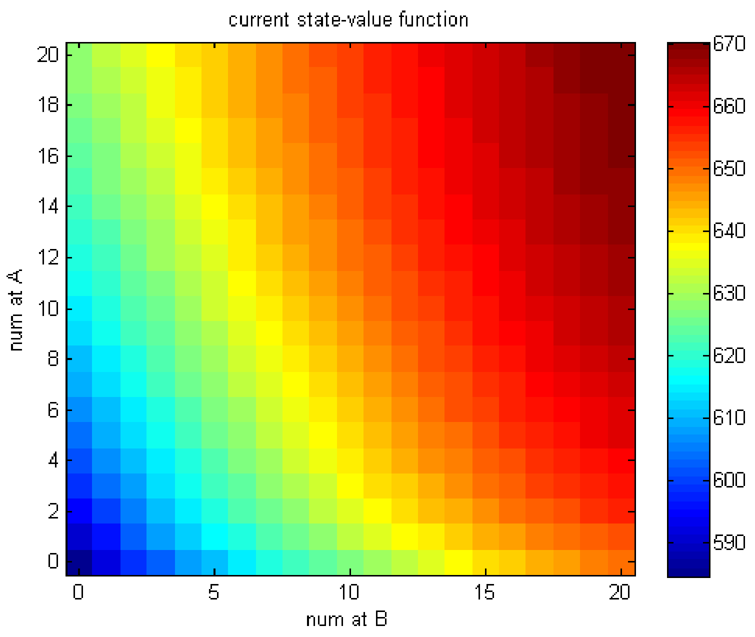


Fig. 3.5: Policy Improvement (4 figures)


current state-value function
num at A
num at B
0
2
4
6
8
10
12
14
16
18
20
0
5
10
15
20
650
640
630
620
610
600
590
580
570
560


current state-value function
num at A
num at B
0
2
4
6
8
10
12
14
16
18
20
0
5
10
15
20
660
650
640
630
620
610
600
590



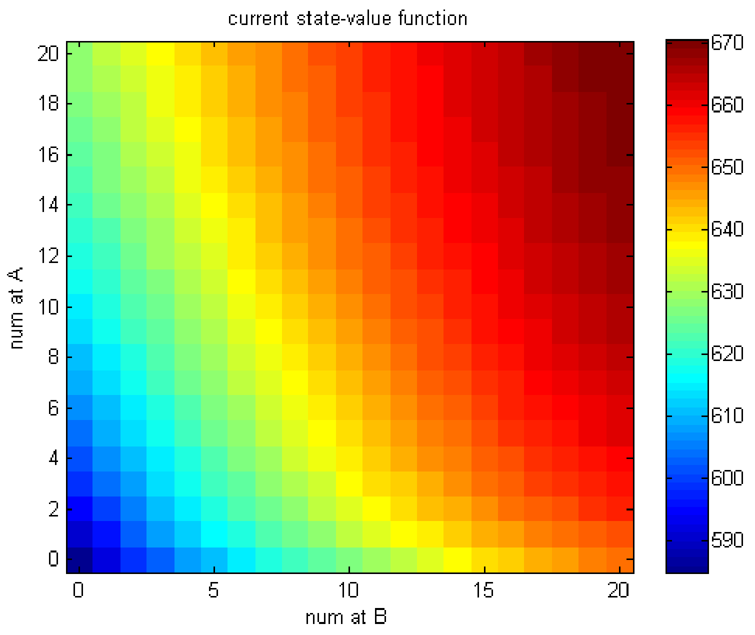


Fig. 3.6: Value Improvement (4 figures)

3.4 Conclusion

Fig. 3.5 depicts the course of policy improvement, the final figure being the optimal policy. The number of cars transferred between location A and location B have been minimized as the policy as improved, which is shown by the increasing area at the center of the figures.

Fig. 3.6 depicts the value function improvement due to the improved policies. The peak of the surface plot above has increased from the first to the fourth figure. This depicts that the amount Jack receives at the end of each day has been maximized by using the dynamic programming optimization algorithm, with Policy evaluation and Policy improvement happening simultaneously until the optimal policy and the corresponding optimal value function is obtained.

Chapter 4: Cart-Pole Control Problem

4.1 Reinforcement Learning on Control Problems

With increasing demand for complex nonlinear control systems, design of optimal controllers through classical methods is challenging. The fact that Reinforcement Learning enables a system to learn through exploration, and adapt to a dynamic environment makes it optimal to be applied to control a nonlinear complex system. An inverted pendulum is simulated as a control task with the goal of learning to balance the pendulum with no a priori knowledge of the dynamics, using different Reinforcement learning algorithms such as Temporal Difference, Actor-Critic Networks and Value Function Approximation. The results of these algorithms are compared with one another and with the results of classical control techniques.

4.2 Cart Pole Benchmarks:

The Cart Pole balancing problem has been used as a benchmark for Reinforcement Learning algorithms for many decades now such as [2]. While the newer definitions of the Cart Pole balancing problem focus on achieving control in complex state and action spaces, the older definitions of the problem are fundamental, first arising in [3], [4]. The fundamental problem statement has been derived from an Adaptive Control technique known as the BOXES [5], and thus has been explored by both the Reinforcement Learning and Control research communities in the past. In addition to its prominence in these literatures, this problem is a challenge for the RL agent as it has to select and take actions in a very limited and discrete action space.

4.3 Cart Pole Dynamics

A pendulum is pivoted to a cart, which has one degree of freedom in the horizontal axis. The goal of the problem is to swing up and balance the pendulum at the upright position, also known as the unstable equilibrium position by using only the bi-directional force that is imparted on the cart. The state of the pendulum at any time is defined by four state variables:

1) Angular Position θ
2) Angular Velocity $\dot{\theta}$
3) Linear Position of the cart x
4) Linear Velocity of the cart $\dot{x}$

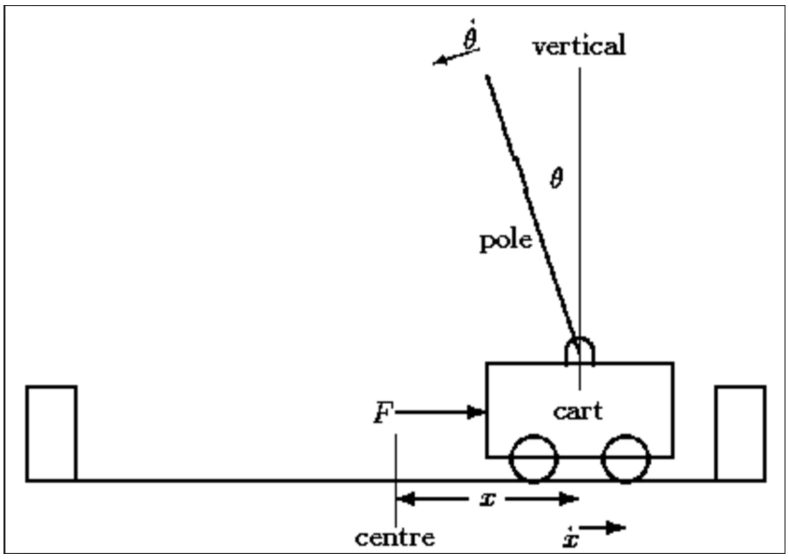


Fig 4.1: Cart Pole Inverted Pendulum

4.3.1 Cart Pole Equations of motion

The Cart Pole system is simulated using the following equations of motion:

$$\ddot{\theta} = \frac{(M+m)g sin\theta - cos\theta[F + ml\dot{\theta}^2 sin\theta]}{\left(\frac{4}{3}\right)(M+m)l - mlcos^2\theta} \tag{4.1}$$

$$\ddot{x} = \frac{\{F + ml[\dot{\theta}^2 sin\theta - \ddot{\theta} cos\theta]\}}{(M+m)} \tag{4.2}$$

where,

M: Mass of cart

m: Mass of pendulum

g: Acceleration due to gravity

F: Force applied on the cart

$\ddot{\theta}$: Angular Acceleration of pendulum

$\ddot{x}$: Linear Acceleration of the cart

l: Length from the centre of mass to the pivot

The system is simulated by numerically approximating the equations of motion using Euler's method with a time step $\tau = 0.02$ seconds, i.e., the sampling rate of the cart pole system and the rate at which control forces are applied are at 50Hz frequency, using discrete time state equations of the form

$$\theta[t+1] = \theta[t] + \tau\dot{\theta}[t] \tag{4.3}$$

4.4 Cart Pole Plant Model for Reinforcement Learning

When Reinforcement Learning is applied to the Cart-Pole problem, the dynamics of the Cart-Pole subsystem remain the same as in Eqn. 4.1 and Eqn. 4.2. However, a few changes are made in order to incorporate features of Reinforcement Learning and ensure that the Cart-Pole Plant can be completely described by a Markov Decision Process, as described in the next chapter.

Reinforcement Learning requires that the Environment be in the form that can be described as a Markov Decision Process, where the Action space must be finite. Thus, to limit the number of actions that can be taken on the pendulum, a simplified form is considered, as defined in [3]. Here, a constant magnitude of Force F is considered. Thus, the action that can be taken by the RL agent on the Environment can either be +F or –F.

4.4.1 State Space Quantization

An entire class of Reinforcement Learning algorithms explored in the next chapter are known as Tabular Lookup Methods. In Tabular Lookup Methods, unlike Continuous methods, the states of the Cart-Pole system are quantized, that is, the values of pendulum angle, its velocity, cart position and velocity spanning each of their range have been separated into multiple bins. A box is defined here, as a tuple comprising of one bin from each of the four state variables.

These quantized states are the states with which the RL Agent builds a Value function, similar to that of Dynamic Programming. The Cart-Pole plant model has been quantized in using different quantization parameters, to explore the effect of quantization on the performance of the algorithm.

4.4.1.1State Space Quantization Type 1: getBox

In the first type of State Space Quantization, the states of the Cart position and velocity as well as the Pole angle and angular velocity have been quantized into 15 bins, and permuted to 162 different boxes representing a tuple of θ, $\dot{\theta}$, x and $\dot{x}$, using the following rule:

θ: $[-12,-6),[-6,-1),[-1,0),[0,1),[1,6),[6,12]$ radians

$\dot{\theta}$: $(-inf,-50),[-50,50],[50,inf)$ radians/second

x: $[-2.4,-0.8),[-0.8,0.8],(0.8,2.4]$ meters

$\dot{x}$: $(-inf,-0.5)$, $[-0.5,0.5]$, $(0.5,inf)$ meters/second

4.4.1.2State Space Quantization Type 2: getBox2

In the second type of State Space Quantization, the states of the Cart position and velocity as well as the Pole angle and angular velocity have been quantized into 21 bins, and permuted to 324 different boxes representing a tuple of θ, $\dot{\theta}$, x and $\dot{x}$, using the following rule:

θ: $[-12,-6], (-6,-4], (-4,-3], (-3,-2], (-2,-1], (-1,0], (0,1], (1,2], (2,3], (3,4], (4,6]$

$, (6,12]$ radians

$\dot{\theta}$: $(-inf,-50), [-50,50], [50,inf)$ radians/second

x: $[-2.4,-0.8), [-0.8,0.8], (0.8,2.4]$ meters

$\dot{x}$: $(-inf,-0.5)$, $[-0.5,0.5]$, $(0.5,inf)$ meters/second

4.4.2 Initial State after failure

If the state of the Cart pole system at any time, does not belong to the discrete bins of the quantized state space, then a failure is said to have occurred and a reinforcement of -1 is generated, which marks the end of an episode. Each time a failure occurs, the pendulum is reset to the upright unstable equilibrium position, which is the initial position for the next episode.

Chapter 5: Reinforcement Learning - Model Free Learning

Dynamic Programming can be useful in reducing computation time for problems with long episodic or continuing environments. But the caveat is that they require the full knowledge of the MDP. The State value function is updated using the Reward function, $\boldsymbol{R_s^a}$ and the State Transition Probability Matrix, $\boldsymbol{P_{ss'}^a}$ which are the MDP model parameters of the environment. When either or both of these parameters are not available in the MDP, Dynamic Programming cannot be applied and thus, a large amount of research has been carried out to develop algorithms to solve problems with such MDPs. The entire class of such algorithms are termed as Model-Free methods [1].

Model-Free methods rely on MDPs to sample various sizes of experiences from the environment depending on the algorithm used. These experiences are then used by an agent to either estimate a model or directly use them to make decisions and take actions in the environment. In Model Free Learning, an agent usually uses the experience to directly take actions, while the process of estimating a model by an agent using experiences from the environment is known as Model Learning.

It is perceived that Note: Model Learning is an alternative approach to solving a problem by Model Free Learning approach. By learning a model through sampled experiences, Dynamic Programming methods can be used to take optimal actions using the learned model. The problem with this approach, however, is that it assumes the environment to be deterministic and can result in high variance in certain states depending on the experience of the entire system. Thus, it cannot be used in problems where reward matrix or transition matrix is a probability distribution rather than a set of fixed values.

Model-Free methods are of different types such that they can be used to take actions in the environment after an entire episode of experience, after just one step of experience or n-steps of experience, where n is less than the number of steps in an episode. Algorithms for Model-Free Learning can be categorized into various types based on the length of backups, for example:

1. One Step Backup: Temporal Difference or TD(0) Learning
2. Infinite Step or Full length Backup: Monte Carlo Learning
3. N-step Backup
4. Multi-step backup: TD(λ)

Like Dynamic Programming, Model Free methods largely use the Generalized Policy Iteration to evaluate the environment and select actions. The first part of Generalized Policy Iteration, also known as Policy Evaluation is performed in a similar manner to Dynamic Programming, but the update equations vary depending on the type of Model Free algorithm used.

The second part of Generalized Policy Iteration is Policy Improvement. Once a policy is evaluated, it must be changed so that the actions taken by the agent result in better rewards, and in the Cart pole problem, extended period of the cart and pole within the limits defined. Policy improvement is performed by choosing the actions from the current space using the Value function developed during the Policy Evaluation. Intuitively, we select an action a, that results in the next state having the maximum state value, i.e.

$$\pi_{new}(s) = \text{argmax}_{a\in A}(V(s')) \tag{5.1}$$

Though the quality of the next state has been taken into account while choosing the new policy, the transition from the current state to the next state has not been considered. As defined by the MDP, a reinforcement signal is not received when the agent is in the state, but after some delay. Thus, the reinforcement of the current state would be received during the transition from current state to the next state, and must be added to the next state value while updating the policy for the current state. Also, the path to be taken to reach state s' has not been defined. For example, there might be states unreachable from the current state, or the agent's action might be affected by environment's stochastic properties and not lead to the expected state. In the cart pole problem, the latter is not possible as the environment of the Cart Pole system has been assumed to be deterministic. As an example, when a small constant force is applied on the cart for a small duration, it might not be possible for the agent to move the Pole from -12° to +12° by taking a single action. Thus, incorporating the Reinforcement signal and the path (or probability) from current state to the next state, the Policy improvement using the State Value function is given by:

$$\pi_{new}(s) = \text{argmax}_{a\in A}(R_s^a + P_{ss'}^a V(s')) \tag{5.2}$$

Now, since we assume Model Free approach, the Reward function R_s^a and the State Probability Transition Matrix $P_{ss'}^a$ are unknown to the RL agent. Thus, the policy improvement cannot be performed using the Eqn. (5.2). The reinforcement signal and path from current state to the next state must be included in the selection of policy π_{new} . This calls for using another type of value function known as the State-Action value function, or Action value function: $Q(S, A)$. Since this type of value function incorporates the consequence of the current action, the Policy Improvement using the Action Value function is given by:

$$\pi_{new}(s) = \operatorname{argmax}_{a \in A}(Q(s', a')) \tag{5.3}$$

When using Model Free Learning for Control, there are two ways to join Policy evaluation and Policy improvement into Generalized Policy Iteration. Exploration of the environment is necessary to ensure that the agent is not trapped inside the local maxima of expected return while selecting actions based on the quality of the state. This exploration can be performed in many ways, one being to choose actions randomly in the given state. On the other hand, Exploitation or choosing the action which results in the maximum reward is required to achieve the objective. This trade-off is handled by two methods of Model Free control:

1. On-Policy Control:
   The agent uses the same policy algorithm to select actions from the action value estimates as well as generate action value estimates
2. Off-Policy Control:
   The agent uses one policy algorithm, usually a greedy policy, to select actions from action value estimates and another policy algorithm, usually an exploratory policy, to generate action value estimates.

5.1 Monte Carlo Learning

The Monte Carlo method is a mathematical approach to solve various optimization problems by the process of random sampling. Monte Carlo Learning is the application of the Monte Carlo method on an MDP by randomly sampling the environment for an entire episode, before using the experience to select the best actions to maximize rewards from the environment.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

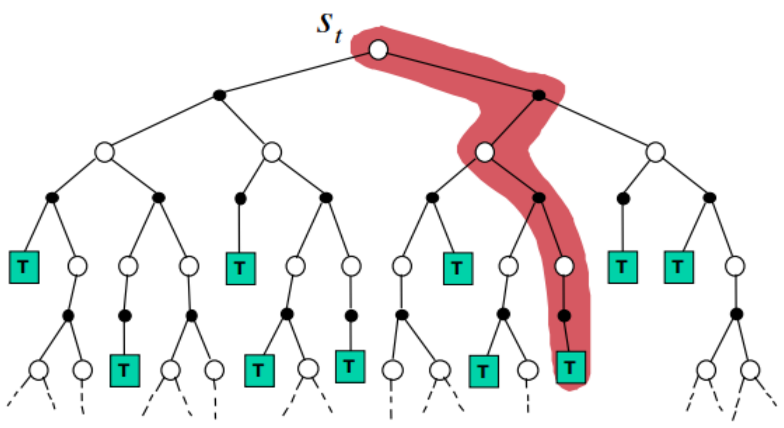


Fig. 5.1: Monte Carlo Learning Backup Diagram

Monte Carlo Learning applied to Control Problem uses the same algorithmic structure as Dynamic Programming, namely the Generalized Policy Iteration. (Figure 5)

First, the Monte Carlo for the Prediction problem is considered, and can be further classified into two types:

1. First Visit Monte Carlo Policy Evaluation

2. Every Visit Monte Carlo Policy Evaluation

Selection of the Monte Carlo Prediction type is based on the environment and the Objective. In this structure of Generalized Policy Iteration, the update equation for the value function can be replaced with the following Monte Carlo update:

Number of first (or all) visits to state s, $N_t(s)$

Total returns from first (or every) visit to state, $S_t(s) = S_t(s) + G_t$

Average Return observed from first (or every) visit to the state s, $V_t(s) = \frac{S_t(s)}{N_t(s)}$

As the algorithm requires a Terminal State, Monte Carlo Methods can be applied only on episodic tasks. This is ideal for the Cart Pole Balancing task as the terminal state is considered as the state where the agent receives a Failure signal, as it has exceeded the limitations defined by the Objective.

Reinforcement of -1 is received only at the end of an episode in the Cart Pole Balancing task, the update equation for $S_t(s)$ is reduced to:

$$S_t(s) = S_t(s) + R_T \tag{5.4}$$

In the above Monte Carlo update, the state value function V(s) is calculated by dividing the total return by the number of first (or all) visits, which is an overhead as this computation has to be performed for every state at every episode. The update rule can be reduced by following the approach of Incremental means, where

$$Mean_t(s) = Mean_{t-1}(s) + \frac{1}{N(s)}\left(R_T - Mean_{t-1}(s)\right) \tag{5.5}$$

Thus, the algorithm for Monte Carlo Method with incremental updates applied on Cart Pole Balancing problem, at the end of every episode is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_T - Q(S_t, A_t)] \tag{5.6}$$

In Monte Carlo Learning, the value function is updated at the end of an episode using the total return obtained from that episode. In the Cart Pole problem, the length of the episode grows as the agent performs better in selecting actions. In such a situation, when Monte Carlo Learning is used to update the agent's value function, a state (say, s_1) explored at the beginning of the episode will have to wait until the end of the entire episode to improve the estimate of its quality. Meanwhile, the old estimate of the state s_1 would be used to update the quality of the other states throughout the episode, whenever the agent's next state space S' contains s_1. While Monte Carlo Method can reach optimal policy, the long interval between the exploration of a state and update of its estimate can result in a slow learning process. Speeding up of the Learning process has been addressed in the past [6] and another class of Reinforcement Learning algorithms have been introduced to overcome this disadvantage.

5.2 Temporal Difference Learning:

Temporal Difference Learning (TD) is a class of Reinforcement Learning algorithms which involve one step updates of the value function and bootstrapping to estimate the quality of a state.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

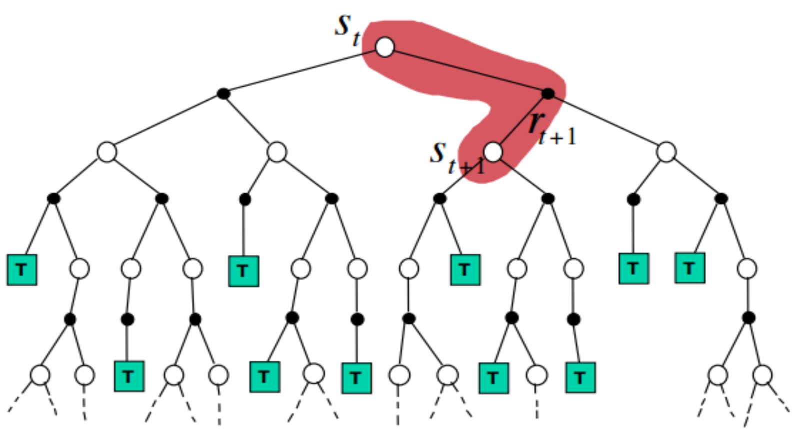


Fig. 5.2: Temporal Difference Learning Backup Diagram

Unlike Monte Carlo Methods, the TD methods are step-by-step algorithms with online updates of value estimates. At every step of an episode, the quality of the state is updated using the reward obtained at that step and the old estimate of the quality of the next state. In other words, a guess of the state's quality is updated towards a better guess.

The TD Learning update equation for:

1. Prediction:

$$V(S_t) \leftarrow V(S_t) + \ \alpha[R_t + \ \gamma V(S_{t+1}) - V(S_t)] \qquad (5.7)$$

   Where(S_t) corresponds to the current state,

   (S_{t+1}) corresponds to the next state,

   $V(S_T)$ is the quality of the agent being in state S_T

   R_t is the reward obtained by from state S_t

2. Control:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \ \alpha[R_t + \ \gamma Q(S_{t+1}, a_{t+1}) - Q(S_t, A_t)] \qquad (5.8)$$

   Where(S_t, A_t) corresponds to the current state and action,

   (S_{t+1}, a_{t+1}) corresponds to the next state and action,

   $Q(S_T, A_T)$ is the quality of the agent being in state S_T and taking action A_T

   R_t is the reward obtained by taking an action A_t from state S_t

5.3 On-Policy Temporal Difference Control Algorithm (SARSA)

When used for control, the On-policy TD algorithm, also known as SARSA (representing State-action, Reinforcement, next State-action), involves update of an action value function, $Q(S, A)$ at every step.

The SARSA update equation is given by:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (5.9)$$

Here, $R_t + \gamma Q(S_{t+1}, A_{t+1})$ is the TD target and the expression in rectangular brackets is the TD error.
We consider each step in the algorithm to represent a State-action pair:

1. The current state, S_t or the internal state of the Cart Pole dynamics represented in a form that the RL agent can interpret.
2. The action A_t to be taken from that state. In the case of Cart Pole balancing, action can be either a constant acceleration of the cart towards LEFT or RIGHT in the track.

The reinforcement signal, R_t is the reward or punishment that the agent receives after the time step t. As the Objective requires the Pole and the Cart to meet the restrictions, we punish the agent with a Reinforcement of -1 if either the Pole or the Cart does not meet its restrictions. When maintaining the restrictions, no reinforcement is rewarded to the agent.

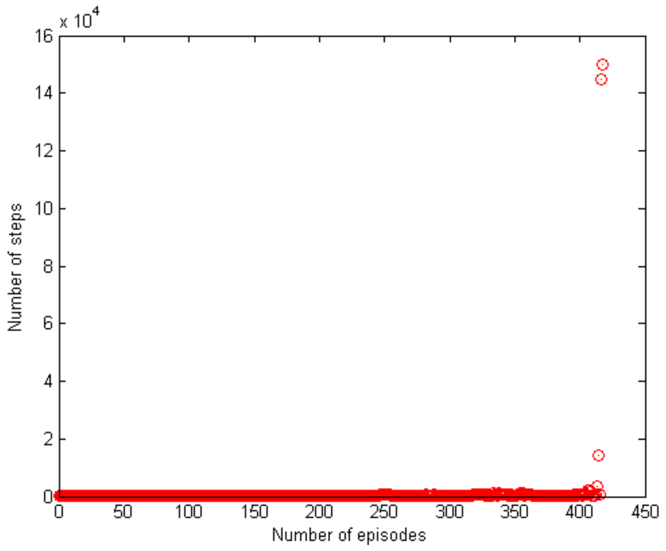
Using the update equation, SARSA can be used to solve the Cart Pole balancing problem with the following algorithm:

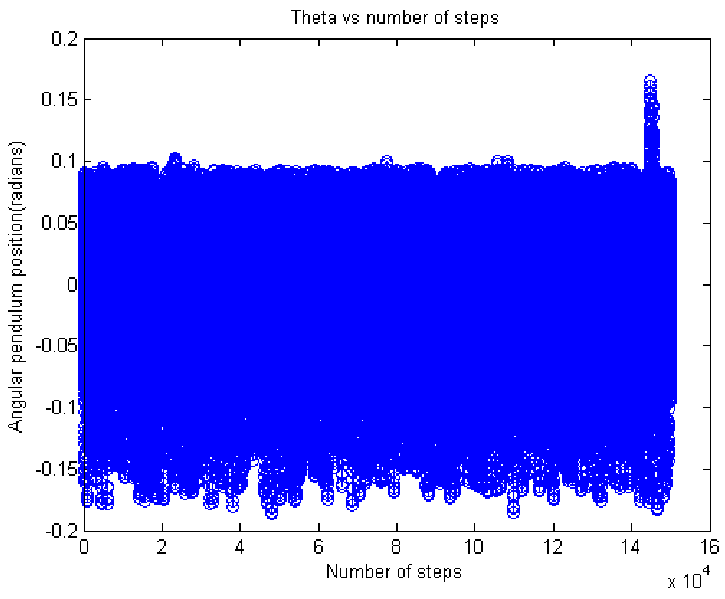
- Initialize all $Q(S, A)\ \forall\ S \in \mathbf{S},\ A \in \mathbf{A}$
- For each episode:
    - For each step in an episode:
        - Given current state S_t, choose A_t using $A_t = \text{argmax}_A\ Q(S_t, A)$
        - Take the action A_t
        - Observe R_t and S_{t+1} from the environment
        - Update the Action value function, $Q(S_t, A_t)$ towards the TD target using the SARSA update equation
        - Until the terminal state, where the state S_t exceeds the limitations defined by the Objective.

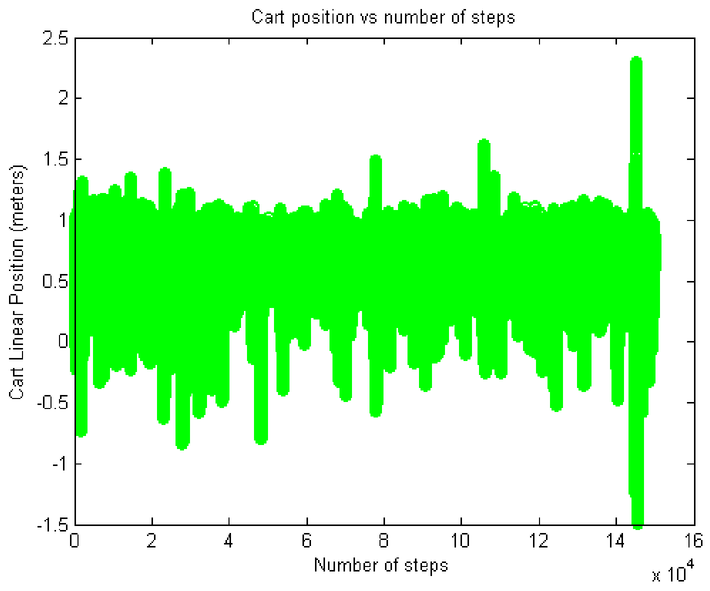
To create a balance between Exploration and Exploitation, stochastic policies are usually considered to select the action in the given state. But the actions defined by the MDP ensure that the region of the state space corresponding to optimal policy is explored even when a greedy policy is used. Thus, in the implementation of SARSA considered for Cart-Pole balancing, a pure greedy policy is used.

5.3.1 Results of SARSA(0):

5.3.1.1 $FORCE = \{+10, -10\}N,\ \alpha = 0.5,\ \gamma = 0.99$







Linear Position and Theta samples for Optimal Policy

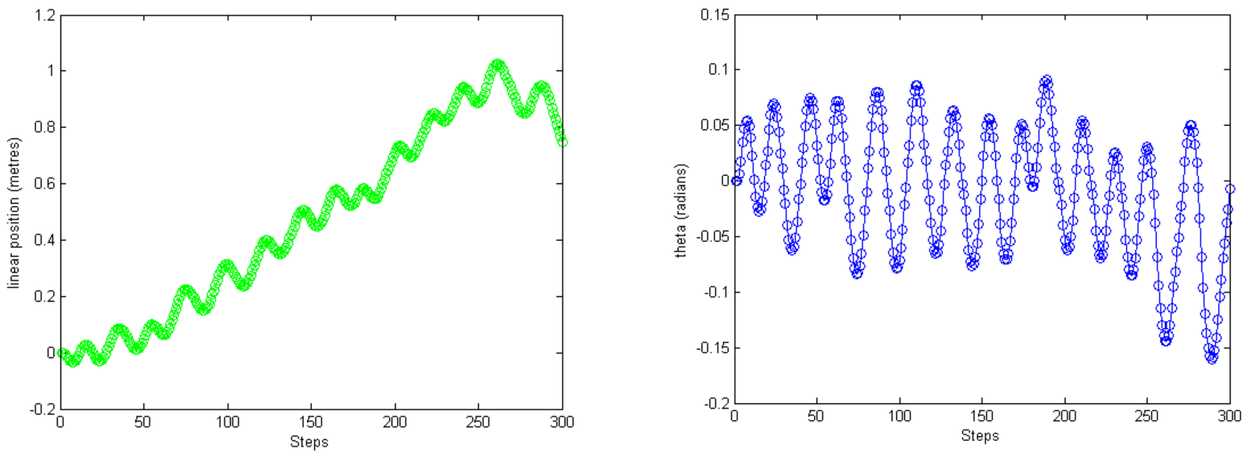
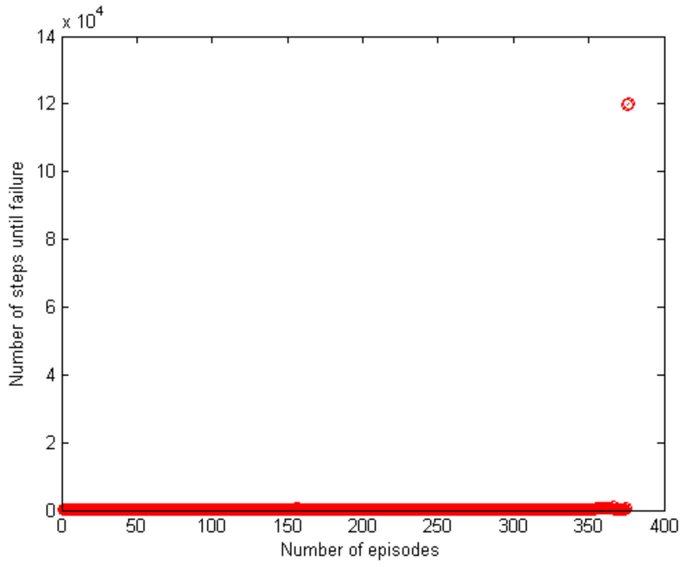
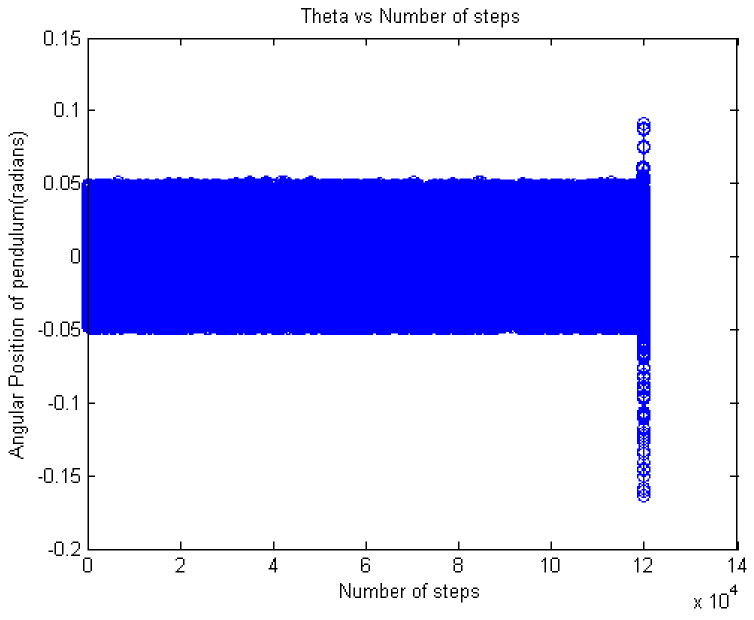
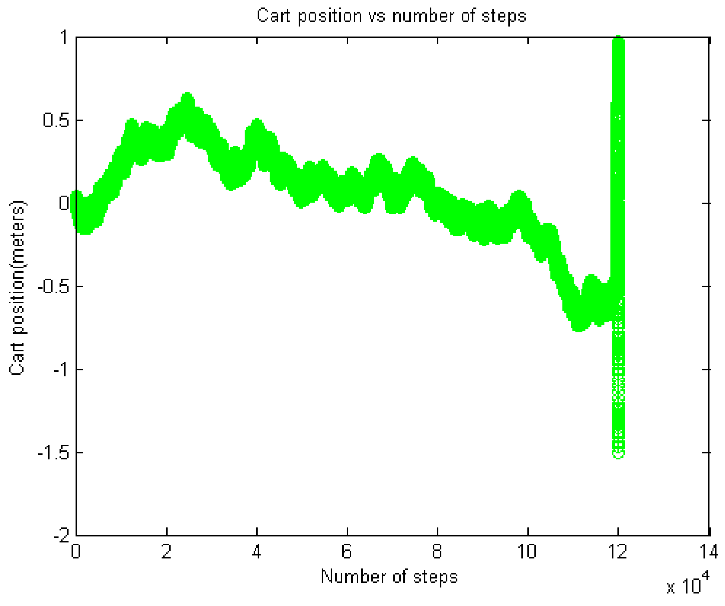


Fig. 5.3: Sarsa(0) for [10,-10] N force

5.3.1.2 $FORCE = \{+15, -15\}N,\ \alpha = 0.6,\ \gamma = 0.99$


x 10^4
14
12
10
8
6
4
2
0
Number of steps until failure
0
50
100
150
200
250
300
350
400
Number of episodes


Theta vs Number of steps
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
Angular Position of pendulum(radians)
0
2
4
6
8
10
12
14
Number of steps
x 10^4


Cart position vs number of steps
1
0.5
0
-0.5
-1
-1.5
-2
Cart position(meters)
0
2
4
6
8
10
12
14
Number of steps
x 10^4

Linear Position and Theta samples for Optimal Policy

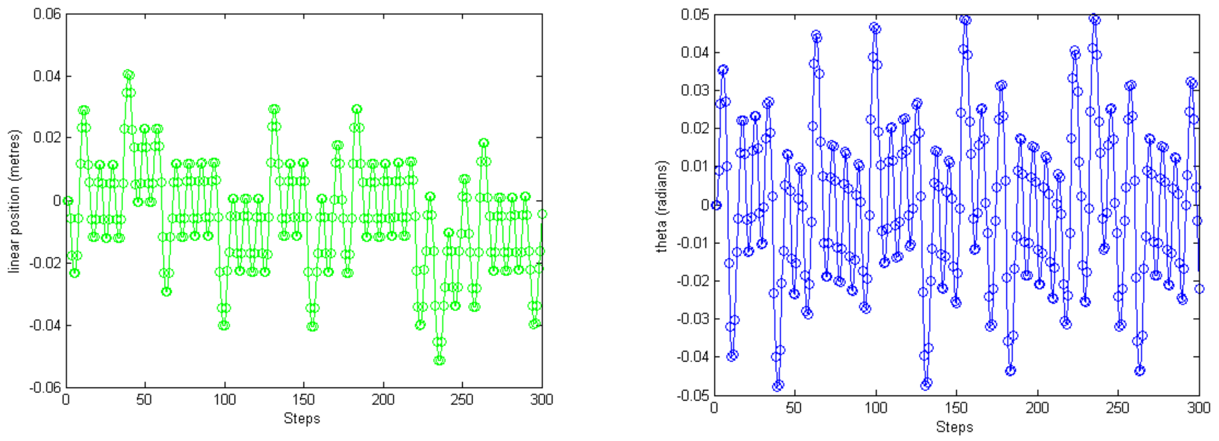
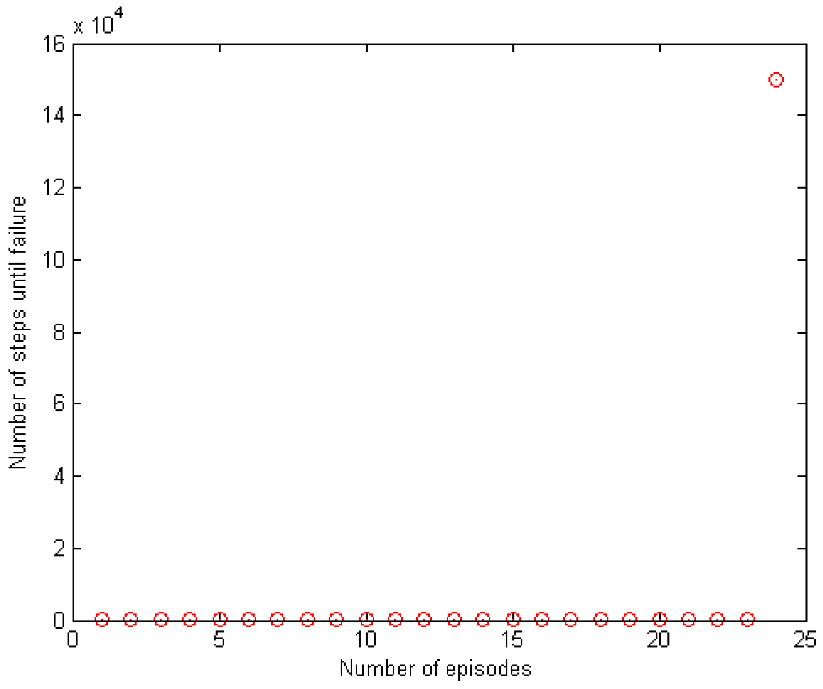
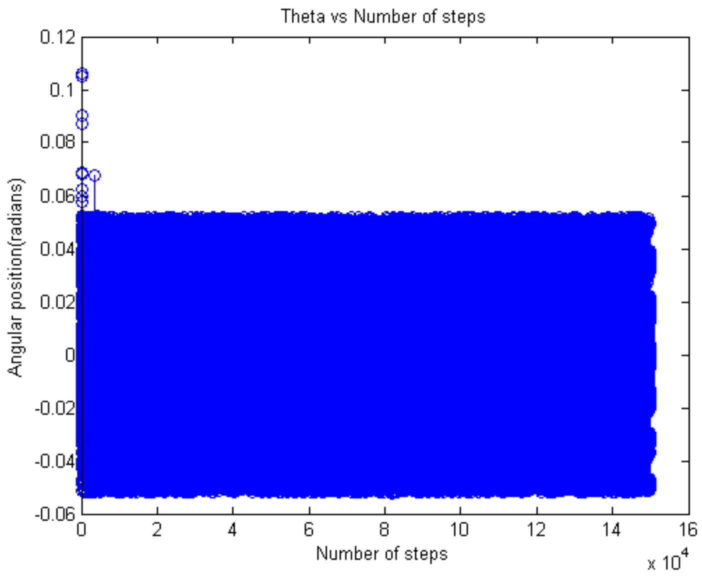
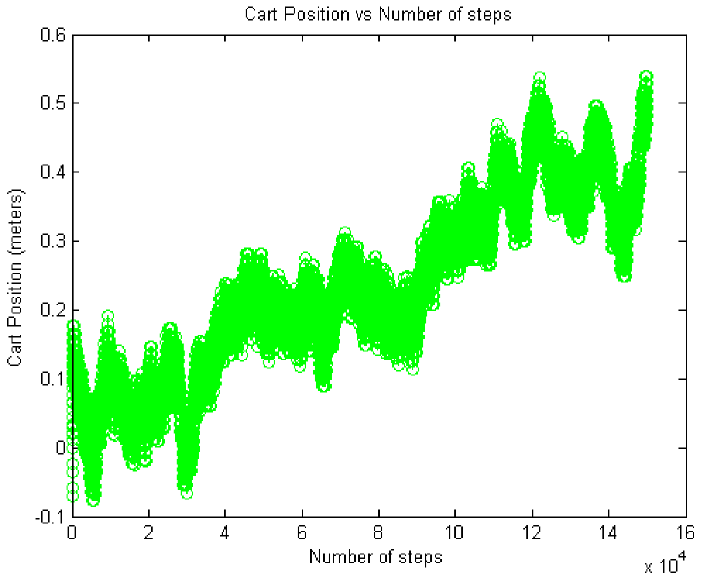


Fig. 5.4: Sarsa(0) for [15,-15] N

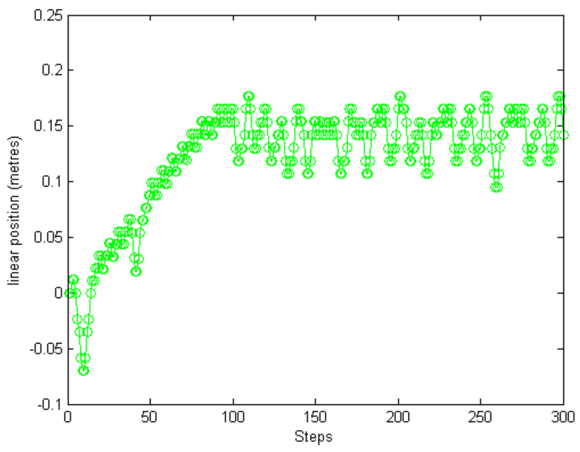
5.3.1.3 $FORCE = \{+30, -30\}N,\ \alpha = 0.4,\ \gamma = 0.99$







Linear Position and Theta samples for Optimal Policy



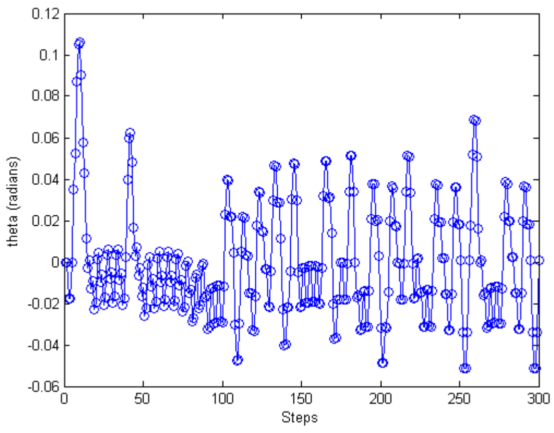


Fig. 5.5: Sarsa(0) for [30,-30] N

5.4 Off-Policy Temporal Difference Control Algorithm (Q-Learning)

Q-Learning is an Off-Policy algorithm since two different policies are utilized by the agent. The policy used to select actions using the state-action values is the greedy policy, given by $\max_{a'} Q(S_{t+1}, a')$. On the other hand, usually an exploratory policy is used to generate the action-value estimates. The Q-learning update equation is given by:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \qquad (5.10)$$

Here, $R_t + \gamma \max_{a'} Q(S_{t+1}, a')$is the Q-target and the expression in rectangular brackets is the Q-error. The reinforcement given to the Q-learning agent is the same as the reinforcement given to the SARSA agent.

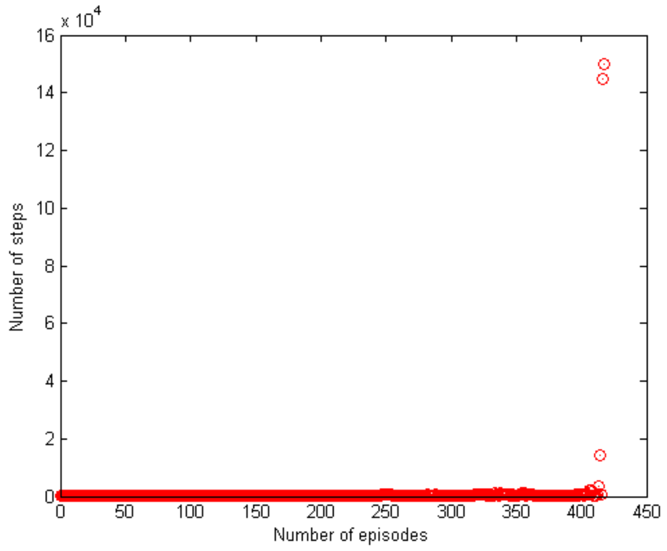
Using the update equation, Q-learning can be used to solve the Cart Pole balancing problem with the following algorithm:

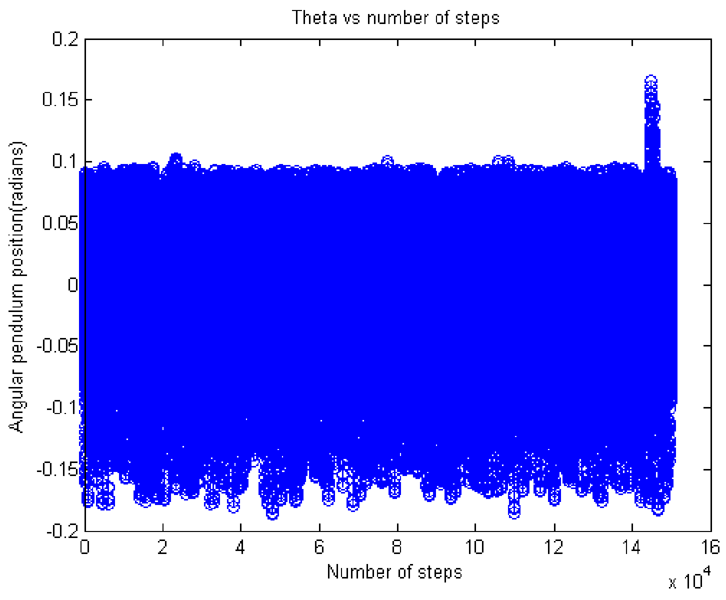
- Initialize all $Q(S, A)\ \forall\ S \in \mathbf{S},\ A \in \mathbf{A}$
- For each episode:
    - For each step in an episode:
        - Given current state S_t, choose A_t using $A_t = \mathrm{argmax}_A\, Q(S_t, A)$
        - Take the action A_t
        - Observe R_t and S_{t+1} from the environment
        - Update the Action value function, $Q(S_t, A_t)$ towards the Q-target $R_t + \gamma \max_{a'} Q(S_{t+1}, a')$ using the update equation
        - Until the terminal state, where the state S_t exceeds the limits set by the objective.

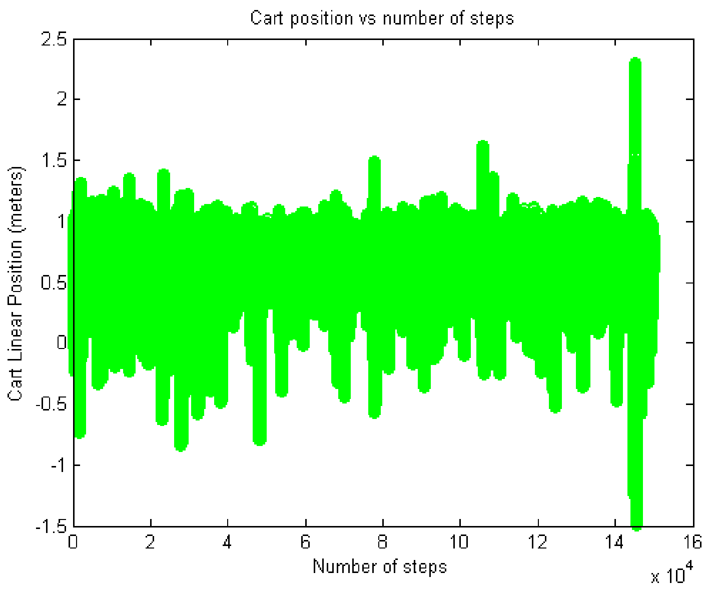
Since the state space of the Cart-Pole MDP is explored even with a greedy policy, the two policy algorithms followed by Off-Policy Control are chosen as greedy policies. As a result of this selection, this special case of Off-Policy method converges to an On-Policy approach.

5.4.1 Results of Q Learning

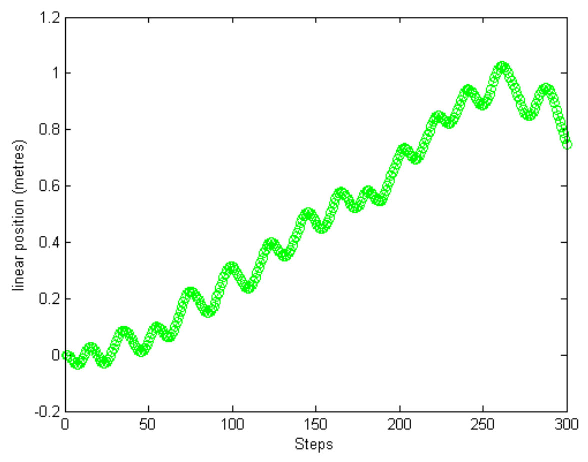
5.4.1.1 $FORCE = \{+10, -10\}N,\ \alpha = 0.5,\ \gamma = 0.99$







Linear Position and Theta samples for Optimal Policy



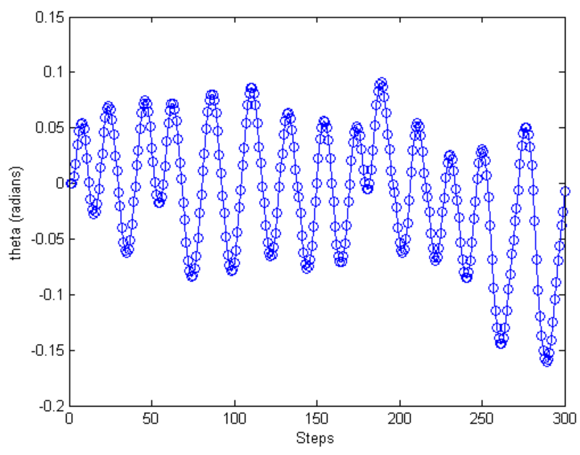
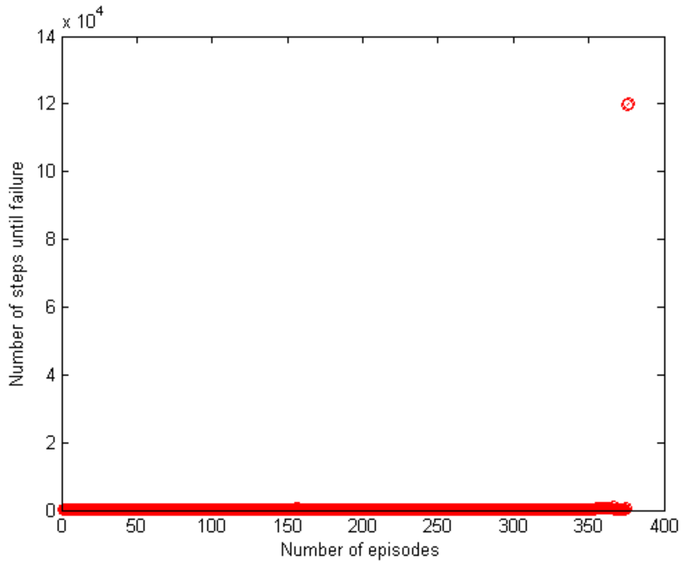
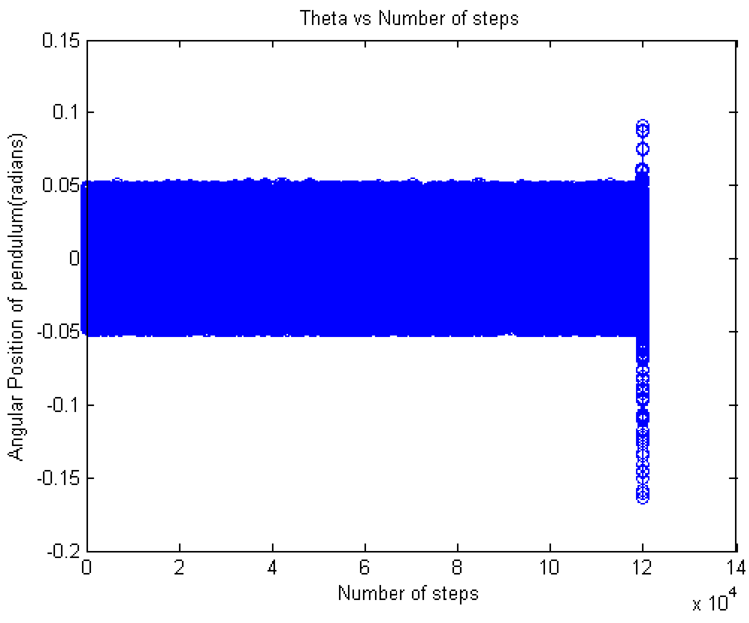
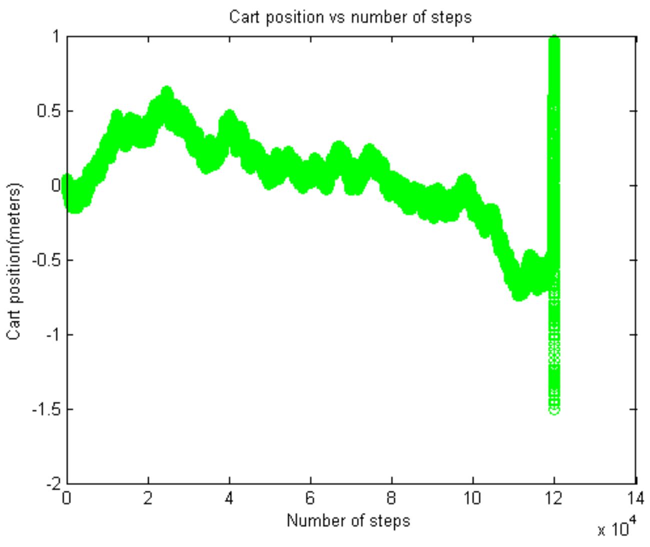


Fig. 5.6: Q learning for [10,-10] N force

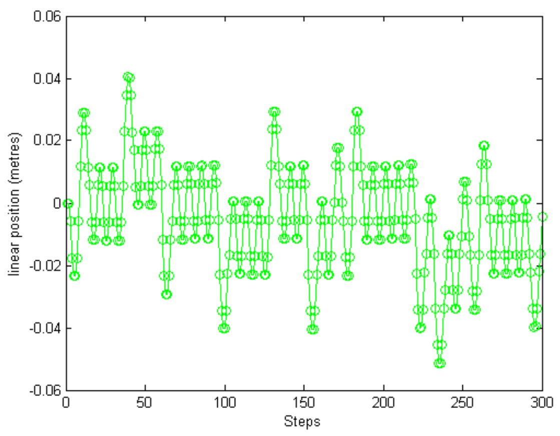
5.4.1.2 $FORCE = \{+15, -15\}N,\ \alpha = 0.6,\ \gamma = 0.99$







Linear Position and Theta samples for Optimal Policy



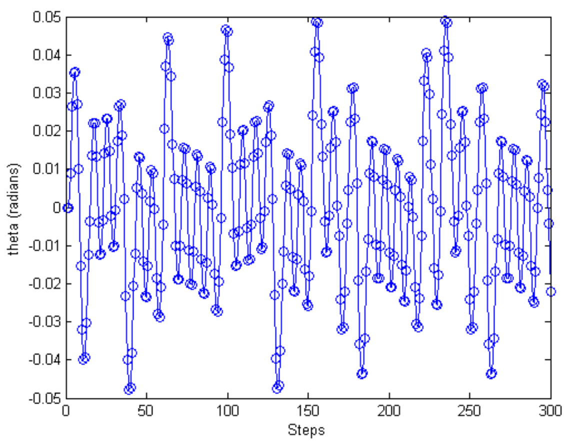
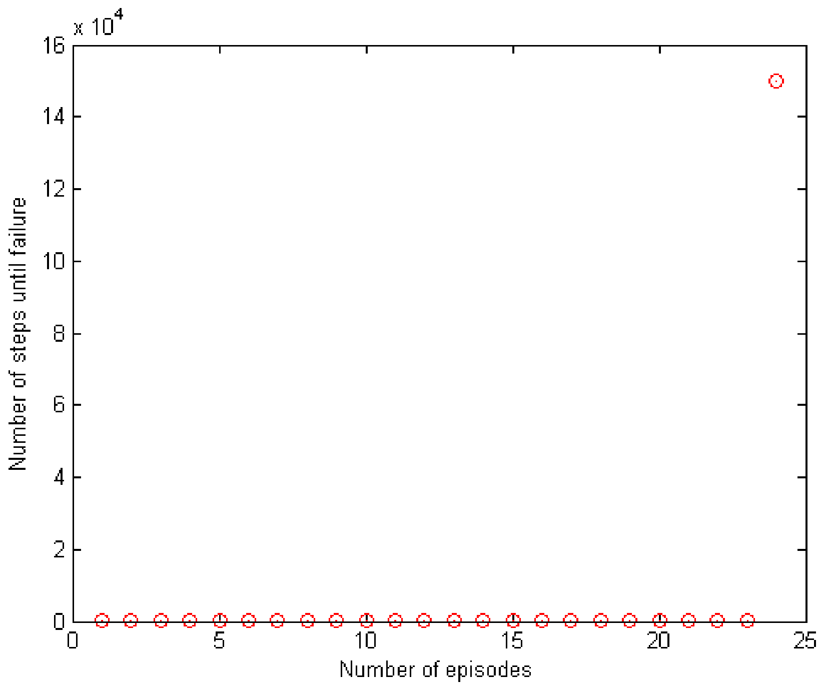
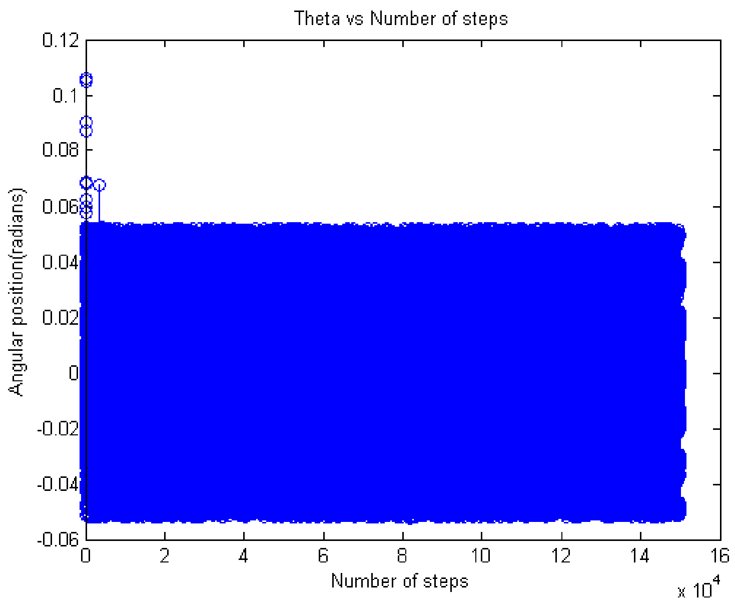
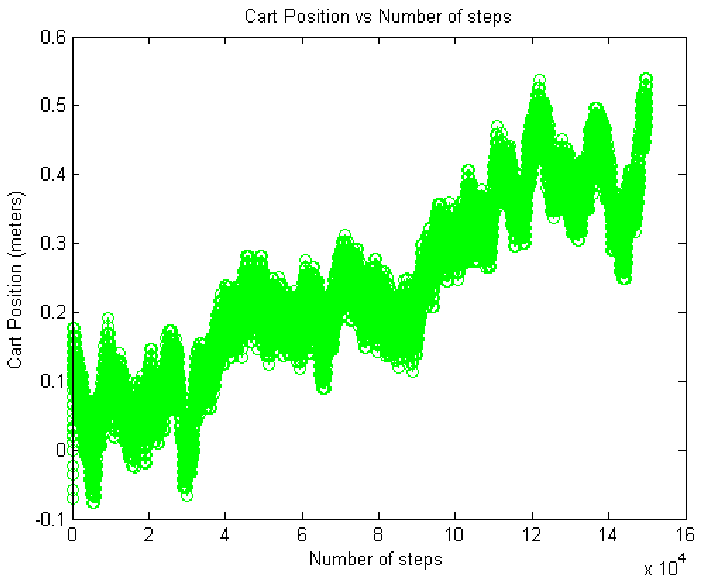


Fig. 5.7: Q learning for [15,-15] N

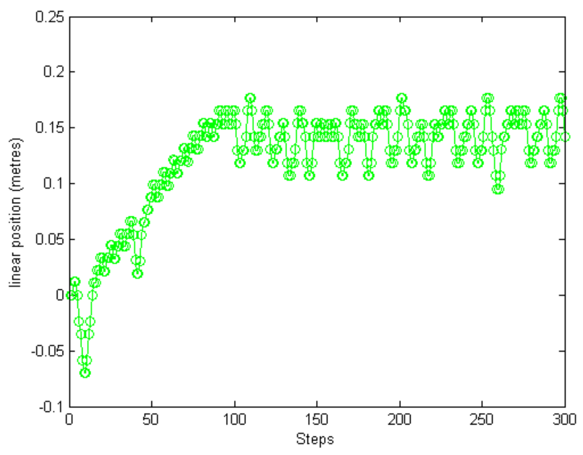
5.4.1.3 $FORCE = \{+30, -30\}N,\ \alpha = 0.4,\ \gamma = 0.99$







Linear Position and Theta samples for Optimal Policy



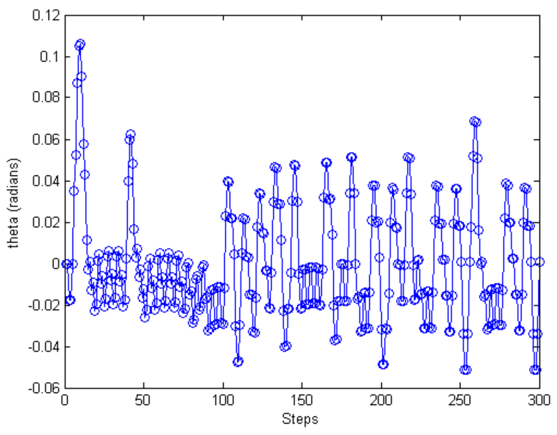


Fig. 5.8: Q learning for [30,-30] N

5.5 Value Function Approximation

5.5.1 Continuous State MDP

The results obtained by using the above Reinforcement Learning methods on the Cart-Pole problem have a major drawback. Their assumption of a discretized state space increases the sensitivity of choosing the value of constant force to be applied by the Agent on the cart, and does not accurately represent a real world scenario of a Cart-Pole system. Also, in the discrete state space, Constant force must be chosen in such a way as to ensure that the system crosses one box and reaches another box at the end of that time step. In other words, the constant force should cause $P_{ss}^{a} = 0$ and this increases the difficulty in selection of the constant force value. Thus, a Continuous state space must be considered in order to overcome these drawbacks. As the state representation in any

Reinforcement Learning problem is defined by the Markov Decision Process, the MDP must be modified. This modified MDP is known as the Continuous State MDP.

Continuous State MDPs represent all the states in terms of a continuous set of values. But the values that are represented by the Continuous state MDP cannot be intermediate values of the Discrete MDP in the case of the Cart-Pole problem, as the assignment of the state indices in the Discrete MDP was arbitrary. Thus, values that are assigned to states must be relative to some standard base value. These values relative to the standard base are known as features. In fact, the continuous state of the cart-pole system can be represented by four features. The same features have been used in Plant models, while attempting to stabilize the Cart-Pole system using conventional controllers:

1. Distance of the cart from the center of the track (x):
   As the cart is placed on a track system of finite length, and that the objective requires that the cart be positioned within the limits of this track at all times during the experiment, the distance of the cart must be measured and examined by the RL Agent.
2. Angle of the pole with respect to the upright position (θ):
   The pole angle must be measured at all times, as the RL Agent must ensure that the pole remains upright and does not fall towards its stable equilibrium.
3. Velocity of the cart ($\dot{x}$): Derivative of x
4. Angular velocity of the pole ($\dot{\theta}$): Derivative of θ

These features form the continuous state $x(s)$ in the Continuous State MDP. Thus, the state $x(s)$ can be represented by a 4-element vector. This state value is then said to have a domain of $\mathbf{R}^4$, which contains 4 real numbers. As a result, the continuous state $x(s)$ contains infinite number of states. $x(s)$ can be written as:

$$x(s) = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \in \mathbf{R}^4$$

5.5.2 Value Function Approximation

With continuous state MDP, it is not possible to update the value of every state individually, as each state $x(s) \in \mathbf{R}^4$. Also, storing a separate value to represent the quality of each state result in very large MDP, which cannot be stored in the memory efficiently. These drawbacks call for a new value function with the following features:

1. Generalises values from states visited by the agent, towards states that:

    a. Have not been visited by the agent
    b. Belong to the neighbourhood of the visited states
2. A set of parameters $\boldsymbol{w} \in \mathbf{R}^4$, which can represent the quality of all states without taking up large memory space.

The new value function, which contains these features is known as an Approximate value function, and is defined by:

$$\hat{v}(s, \boldsymbol{w}) \rightarrow v(s)$$

$$\hat{q}(s, a, \boldsymbol{w}) \rightarrow q(s, a)$$

where

$\hat{v}(s, \boldsymbol{w})$ is the approximate form of the state value function $v(s)$ used in the prediction problem, with the approximation parameter $\boldsymbol{w}$

$\hat{q}(s, a, \boldsymbol{w})$ is the approximate form of the state-action value function $q(s, a)$ used in the control problem, with the approximation parameter $\boldsymbol{w}$

Based on the type of problem that the approximate value function is applied to, it can be classified into three types:

1. State Value Function Approximation:
    a. Input: State of the Cart-pole system
    b. Output: estimated value of state
2. Action-in State-Action Value Function Approximation:
    a. Input: state of the Cart-pole system and action taken by the RL Agent
    b. Output: estimated value of state-action pair
3. Action-out State-Action Value Function Approximation:
    a. Input: state of the Cart-pole system
    b. Output: estimated values of state-action for all actions, $a \in A$

5.5.3 Stochastic Gradient Descent applied on Value Function Approximation

The goal of the RL Agent defined in terms of Value function approximation is to update the approximate value function towards the TD-target in case of the Prediction problem and the Q-target in case of the Control problem. This update can be performed by first, finding a cost function $J(\boldsymbol{w})$ which represents the error between the TD-target or the Q-target and the approximate value obtained from the approximate value function. The error used to calculate the cost is considered to be the Mean squared error:

$$J(\boldsymbol{w}) = \mathbf{E}\left[\left(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w})\right)^2\right] \tag{5.11}$$

The next step is to minimize this cost, by finding a gradient value towards an updated approximate value function that results in a very low cost, thus, bringing the approximate value closer to the TD-target or the Q-target. The gradient update using stochastic gradient descent, samples the gradient to find the local minimum:

$$\Delta \boldsymbol{w} = -\frac{1}{2}\alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w})$$

$$= \alpha\big(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w})\big)\nabla_{\boldsymbol{w}} \hat{q}(S_t, A_t, \boldsymbol{w}) \qquad (5.12)$$

5.5.4 Linear Value Function Approximation with Stochastic Gradient Descent

So far, the approximate value function considered was generalised. Thus, any function approximator can be used. A few examples of function approximators are:

1. Linear combination
2. Artificial Neural Networks
3. Nearest Neighbour
4. Decision Trees
5. Fourier basis
6. Wavelet Basis

As Stochastic Gradient Descent requires a gradient of the approximate value function, only the differentiable function approximators, that is, Linear combination and Artificial Neural Networks can be utilized.

Also, the Cart-Pole problem definition is a fundamental concept and the Linear combination of features can be easily applied to this problem to derive at a near-perfect Reinforcement Learning controller, beyond which, only a change in the problem definition can result in better RL controllers. Thus, adhering to the fundamental problem definition throughout the report, only Linear Combination of Features is utilized as an approximator.

Representing the state-action value function using the linear combination of features, in the context of the Cart-pole problem:

$$\hat{q}(S, A, \boldsymbol{w}) = x(S, A)^T \boldsymbol{w} = \sum_{j=1}^{4} \big(x_j(S, A) \cdot \boldsymbol{w}_j\big) \qquad (5.13)$$

We can now find the gradient of the state-action value function $\hat{q}(S, A, \boldsymbol{w})$. This function can now be represented by $x(S, A)^T \boldsymbol{w}$. As the gradient taken in Stochastic Gradient Descent is with respect to the parameter $\boldsymbol{w}$ and because $x(S, A)$ does not depend on the parameter $\boldsymbol{w}$, the gradient can be reduced to:

$$\nabla_{\boldsymbol{w}}\hat{q}(S_t, A_t, \boldsymbol{w}) = x(S_t, A_t) \tag{5.14}$$

Thus, the parameter update, in order to minimize the error between the Q-target and the linear value function approximate, is reduced to:

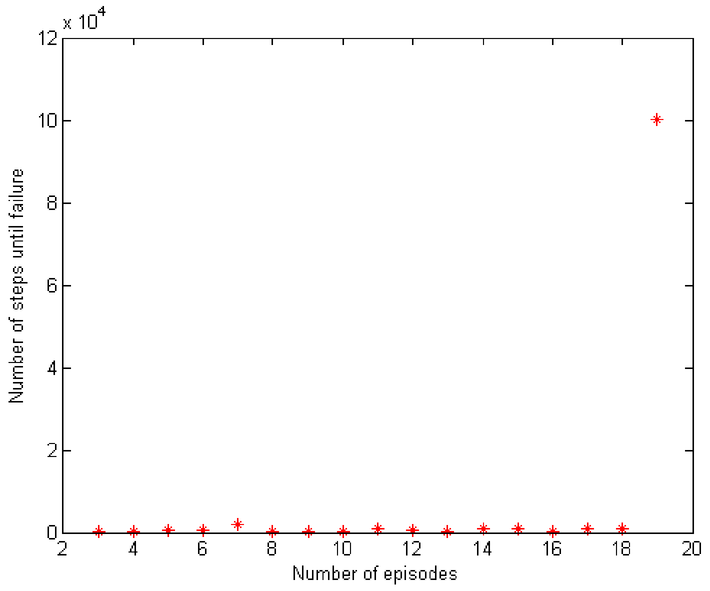
$$\begin{aligned}\Delta\boldsymbol{w} &= -\frac{1}{2}\alpha\nabla_{\boldsymbol{w}}J(\boldsymbol{w}) \\ &= \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w})\big)x(S_t, A_t)\end{aligned} \tag{5.15}$$

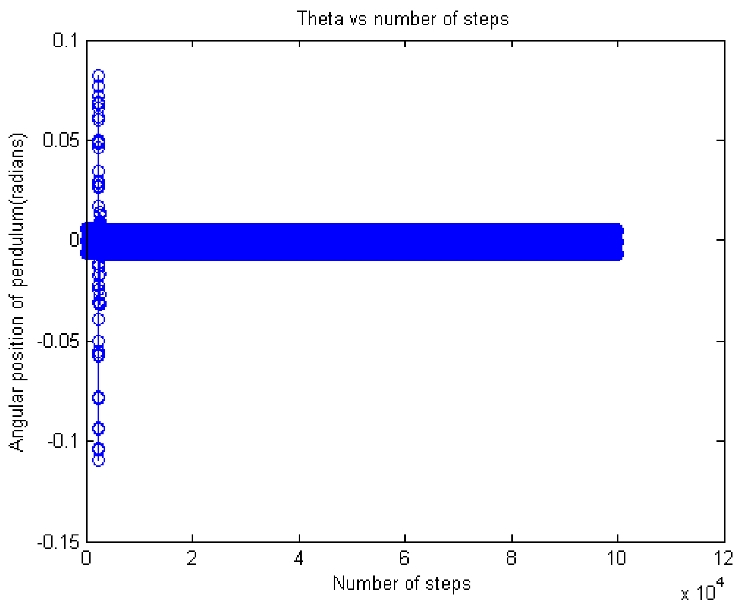
This update equation is replaced by the update equation in the SARSA tabular methods, along with replacing the state-action value function by the linear approximate state-action value function, and implemented in the Cart-Pole system to achieve the Objective. The updated algorithm is:

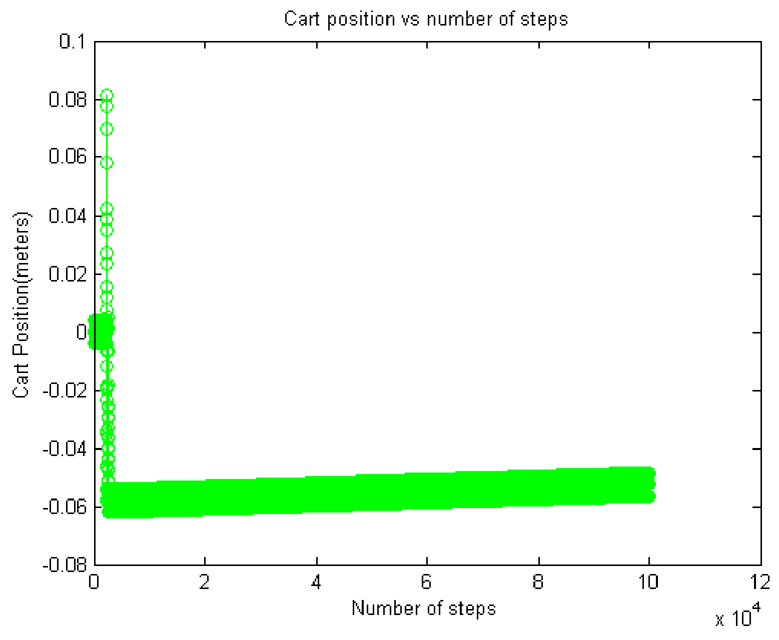
- Initialize all $\hat{q}(S, A, \boldsymbol{w})\ \forall\ S \in \mathbf{S},\ A \in \mathbf{A}, \boldsymbol{w}$
- For each episode:
  - For each step in an episode:
    - Given current state S_t, choose A_t using $A_t = \text{argmax}_A\ \hat{q}(S, A, \boldsymbol{w})$
    - Take the action A_t
    - Observe R_{t+1} and S_{t+1} from the environment
    - Update the Action value function, $\hat{q}(S, A, \boldsymbol{w})$ towards the TD target using the update equation
      $$\Delta\boldsymbol{w} = \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{w}) - \hat{q}(S_t, A_t, \boldsymbol{w})\big)x(S_t, A_t)$$
    - Until the terminal state, where the state S_t exceeds the limitations defined by the Objective.

5.5.5 Results

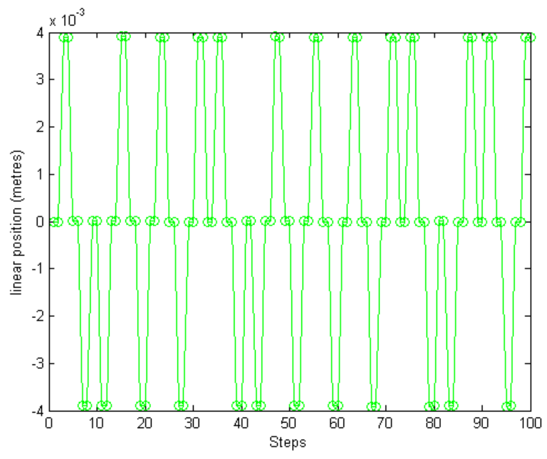
5.5.5.1 $FORCE = \{+10, -10\}N,\ \alpha = 0.07,\ \gamma = 0.992$







Linear Position and theta samples for optimal policy



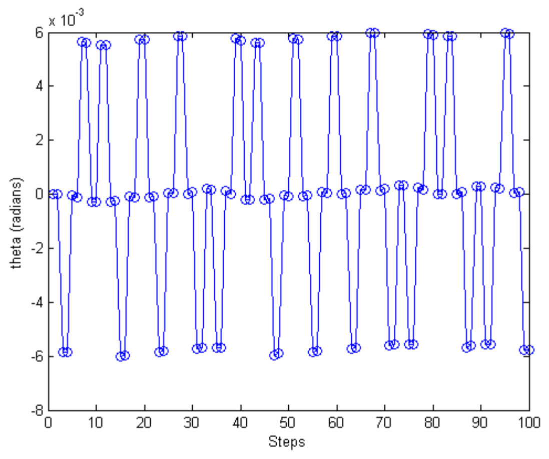


Fig. 5.9: Value Function Approximation

5.6 Actor-Critic Policy Gradient Method

5.6.1 Policy Gradient Methods

The RL algorithms considered so far involve a value function which quantifies the significance of the system being the current state and taking the specified action. Action selection using these value function based methods can be performed either deterministically or with some stochasticity that can be reduced to a deterministic form based on the overall performance of the agent. Unlike value function based methods, Policy gradient methods allow the extent of exploration and ambiguous action selection to be altered based on the quality of each state [1].

Policy gradient methods can lead to optimal stochastic policy, which can be useful in many applications, where determining the accurate value function is complex. In case of Cart-Pole balancing problem, one such example is the upright state, where the pole is in the upright position but the agent must take either action defined by the Objective. In this context, the agent may not prefer a deterministic action as it may limit the exploration across the state space.

The policy gradient method determines the probability of picking an action a, given that the system is in state s and the parameters are $\boldsymbol{\theta}$. Representing in mathematical form:

$$\pi_{\boldsymbol{\theta}}(s,a) = \mathbf{P}[a \mid s, \boldsymbol{\theta}] \tag{5.16}$$

In Policy based methods, we select a cost function to find the policy $\pi_{\boldsymbol{\theta}}(s,a)$ using the expression:

$$J_R(\boldsymbol{\theta}) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_{\boldsymbol{\theta}}(s,a) R_s^a \tag{5.17}$$

where $d^{\pi_\theta}(s)$ is the distribution of the Markov Decision Process for $\pi_{\boldsymbol{\theta}}$

Using stochastic gradient descent to minimize this cost function with respect to $\boldsymbol{\theta}$, and manipulating the equation:

$$\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s,a) = \pi_{\boldsymbol{\theta}}(s,a) \cdot \frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s,a)}{\pi_{\boldsymbol{\theta}}(s,a)}$$

$$\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(s,a) = \pi_{\boldsymbol{\theta}}(s,a) \cdot \nabla_{\boldsymbol{\theta}}\log(\pi_{\boldsymbol{\theta}}(s,a)) \tag{5.18}$$

The manipulation is performed using Likelihood ratios. The expression $\nabla_{\boldsymbol{\theta}}\log(\pi_{\boldsymbol{\theta}}(s,a))$ is known as the score function.

The score function represents a method to select the probability of selection of an action based on the state s and the parameters. This probability can be represented by selecting a probability distribution over the parameters and the set of features representing the quality of the state-action pairs. Using Softmax policy on the score function,

$$\nabla_{\boldsymbol{\theta}}\log(\pi_{\boldsymbol{\theta}}(s,a)) = x(s,a) - \mathbf{E}[x(s,\cdot)] \tag{5.19}$$

where $\varphi(s,a)$ is the feature vector.

5.6.2 Policy Gradient Actor-Critic Methods

A drawback of the Policy Gradient methods is the high variance in estimating the feature vector, and thus, the probabilities of choosing the action based on the state and parameter values. By estimating the action value function using a critic, the variance of the method can be reduced. Thus, in Policy-Gradient Actor-Critic methods [3], [7],

1. Critic: Update action-value function $Q(S_t, A_t)$ or their parameters
2. Actor: Update the policy parameters $\boldsymbol{\theta}$ in the direction of the action-value function as estimated by the critic

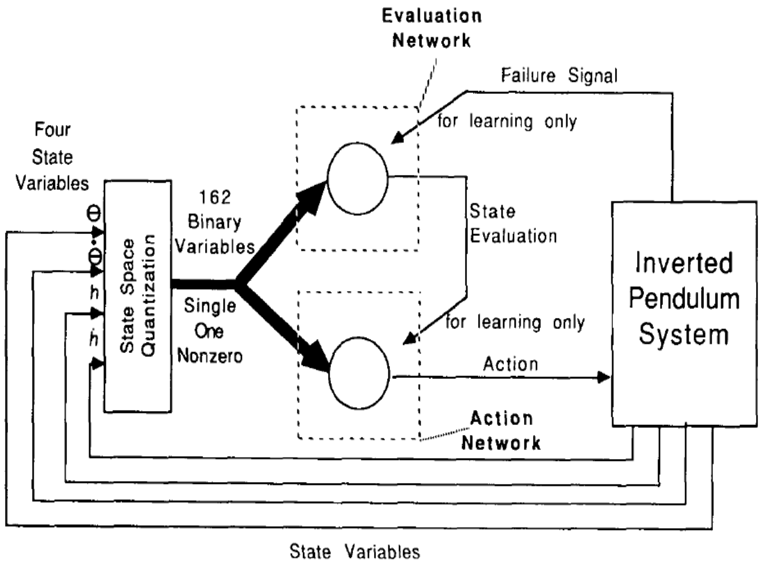


Fig. 5.10: Actor Critic Neuron Like elements

The update equations for the Actor network parameters are given by:

$$\theta_i(t+1) = \theta_i(t) + \alpha\big(r(t+1) + \gamma p^s(t+1) - p^s(t)\big)\bar{e}_i(t) \qquad (5.20)$$

where $e_i(t) = \left(y(t) - \frac{1}{2}\right)x_i(t)$

, $\bar{e}_i(t) = (1-\delta)\bar{e}_i(t-1) + \delta e_i(t)$, with $\bar{e}_i(0) = 0\ for\ 0 < \delta < 1$

, $y(t) = \begin{cases} 1, if\ (s(t) + \eta(t) > 0) \\ 0, otherwise \end{cases}$

, $\eta(t)$ is a random variable chosen from a normal distribution with mean=0 and (standard deviation)=0.1

, $s(t) = \sum_{i=1}^{n} \theta_i(t) \cdot x_i(t)$

, $p^s(t) = \sum_{i=1}^{n} \theta_i(s) \cdot x_i(t), p(t) = r(1)$

The update equation for the Critic network parameters are given by:

$$w(t+1) = w_i(t) + \beta\big(r(t+1) + \gamma p^s(t+1) - p^s(t)\big)\bar{x}_i(t) \qquad (5.21)$$

where

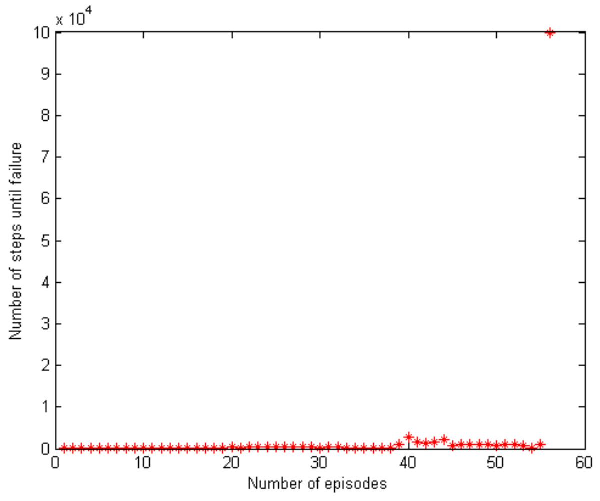
, $\bar{x}_i(t) = (1-\delta)\bar{x}_i(t-1) + \delta x_i(t)$, with $x_i(0) = 0\ for\ 0 < \delta < 1$

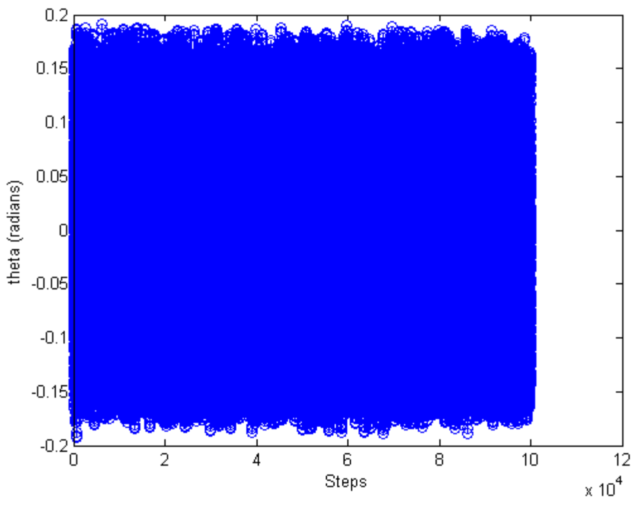
, $p^s(t) = \sum_{i=1}^{n} \theta_i(s) \cdot x_i(t), p(t) = r(1)$

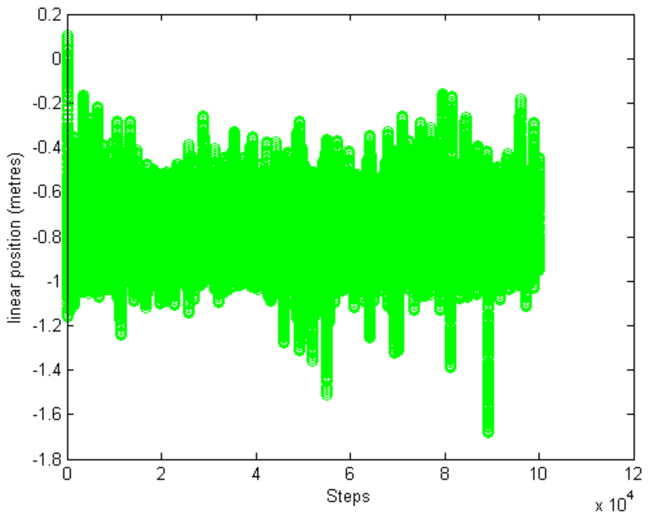
By varying the parameters, $\alpha, \beta, \delta, \gamma, \eta$, implementing the update equation in the Generalised Policy Iteration algorithm, and testing on the Cart-Pole balancing problem, different results are obtained and compared.

5.6.3 Results:

5.6.3.1 $FORCE = \{+10, -10\}N,\ \alpha = 1000,\ \gamma = 0.95,\ \lambda_w = 0.9,\ \lambda_v = 0.8$







Linear positions and Theta samples for optimal policy

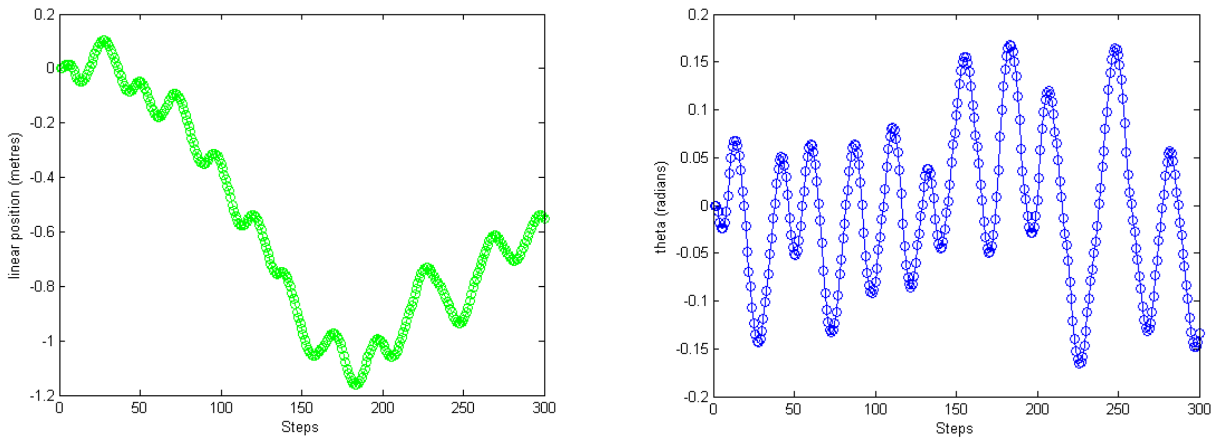


Fig. 5.11: Actor Critic Policy Gradient

5.7 Table of Comparison

Algorithm	**Force (in N)**	α	γ	**State-space Quantizer**	λ_w **and** λ_v	θ **Range (in °)**	x **Range (in m)**	**Episodes until optimal policy**
SARSA(0)	{+10, -10}	0.5	0.99	getBox	-	[-11, +10]	[-1.5, +2.4]	420
SARSA(0)	{+15, -15}	0.6	0.99	getBox2	-	[-2.9, +2.9]	[-1. +1]	380
SARSA(0)	{+30, -30}	0.4	0.99	getBox	-	[-3, +6.3]	[-0.1, +0.6]	24
Q-Learning	{+10, -10}	0.5	0.99	getBox	-	[-11, +10]	[-1.5, +2.4]	420
Q-Learning	{+15, -15}	0.6	0.99	getBox2	-	[-2.9, +2.9]	[-1, +1]	380
Q-Learning	{+30, -30}	0.4	0.99	getBox	-	[-3, +6.3]	[-0.1, +0.6]	24
Value Function Approximation	{+10, -10}	0.07	0.992	-	-	[-0.4, +0.3]	[-0.062, +0.054]	19
Policy Gradient Actor-Critic	{+10, -10}	1000	0.95	-	0.9, 0.8	[-12, +12]	[-1.7, +0.2]	62

Table 5.1: Comparison of Model-Free RL Algorithms

5.8 Conclusion

SARSA(0) is a basic TD control algorithm. While the result of SARSA(0) in Fig. 5.3 achieves optimal policy, a policy for which it achieves the Objective of the Cart-Pole problem, it does so after 420 trials. The cart almost hits one end of the track during the experiment and the pendulum angle also reaches the edge of its restriction limits. Specifically, the range of the angles covered by the Pole, measured from the upright position of the Pole during the experiment is [-11°,+10°], and the range of the Linear Cart Position covered from the centre of the track during the experiment is [-1.5m, +2.4m] Thus, adjusting parameters and choosing more complex Reinforcement Learning algorithms should improve the performance.

Consider Fig. 5.4. This implementation of the SARSA(0) algorithm with specific parameters on the Cart-Pole problem results in optimal policy in 380 trials. Although the parameter selection leads to longer learning time, the agent achieves optimal policy within a very small range of Pendulum and cart position values. Thus, the pendulum angle is within ±2.9° range, very close to the upright position. The cart remains within a position of ±1m from the track's centre. This ensures lower power required to drive the cart, since there is not much wastage of power due to large accelerations. This also causes the system to be more robust to noise as small noisy forces cannot cause the pendulum angle or the cart position to go beyond the limitations and fail.

Consider Fig. 5.5. The SARSA(0) implementation on the Cart-Pole problem using improved parameters caused the Pole to achieve the Cart-Pole objective of balancing for a minimum specified number of steps, in just 24 trials. Selection of parameters suitable for this application, leads to improved performance. Comparing with the previous parameter set, it is evident that parameter selection is crucial for the algorithm's performance especially in Table lookup methods. The pendulum angle oscillates between ±3° but initially shoots up to +6.3°. The cart position is between +0.6m and -0.1m. This shows the increased performance of the algorithm, although, the Cart still drifts slowly towards the right the track, which can cause the Cart to reach the track limit if the optimal policy is implemented for a prolonged period.

Consider Fig. 5.6 – 5.8. Due to the convergence of the Off-Policy approach towards the On-Policy SARSA(0) approach, the results of Q-Learning applied to the Cart-Pole problem are the same as the results of the SARSA(0) algorithm applied on the Cart-Pole problem. The inferences of the Q-learning algorithms with different parameters remain the same and thus, only the Q-Learning results are presented here.

Consider Fig. 5.9. Linear Value Function Approximation applied on the Cart-Pole problem using Stochastic Gradient Descent update, give almost accurate results. By using a fixed Force value of +10N or -10N, decided by a policy that maximizes the approximate value function, the Agent achieves Optimal Control in just 19 episodes. In addition to the best rate of convergence to optimal policy, this algorithm ensures that all oscillations are suppressed to a minimum that are determined by the constant Force values. Lower force values with adjusted parameters can further suppress the oscillations. In the implementation, the Pendulum angle remains within a small range of [-0.4°, +0.3°] during steady state. Along with this, the Cart position with respect to the centre of the track is in the range [-0.062, +0.054]m in steady state. Although there are a few oscillations at the beginning of the episode of Optimal control, that exist beyond this range, these oscillations still lie much within the limits defined by the Cart Pole problem. Similarly, the oscillations in the Cart Position in the beginning of the episode also lie within the limits of the Cart-Pole problem, but in this case, a characteristic of the Cart position is observed, where the Cart permanently deviates from the centre of the track, by a distance of 0.06m. This represents accurate control and is highly favourable over many conventional control approaches.

Consider Fig. 5.11. The policy gradient actor-critic method involves the calculation of a value function by the Critic Network as well as the selection of the best action for the given state and the state value function, through a probabilistic approach. With the parameters adjusted in order to achieve optimal policy, the Policy Gradient Actor-Critic method performs poorly with this set of parameters although it manages to remain within the Cart-Pole state limitations. With a large learning rate, $\alpha = 1000$, it would be expected that the agent would never learn the optimal policy. On the other hand, through the results, it is observed that optimal policy is achieved but through large oscillations in the steady state, which causes large power loss for the entire system. The results show that the range of the pendulum angle during the episode corresponding to the optimal policy is very high, and covers the entire allowed state space from -12° to +12°. The linear position of the cart with respect to the centre of the pendulum also varies widely during the optimal episode, and ranges from -1.7m to +0.2 m. This large variation affects the motion of the Cart and the swinging of the pendulum, and leads to large wastage of power in swinging up the pole. It is noticed that, although there are large errors in the stabilization of the pendulum, the trajectories of the Cart position and the Pendulum angle are nearly periodic, which suggests that this is a characteristic of Actor-Critic methods and improving the algorithm by selecting better parameters, increases the chance of improved performance.

Chapter 6: Swing up and Stabilization of an Inverted Pendulum using Classical Control Techniques

6.1 The Cart Pole Problem and the Inverted Pendulum

The Cart Pole problem has been defined in Control Theory literature, more generally, as the Inverted Pendulum problem, for example [8]. With the apt title of Inverted Pendulum, the problem statement reveals the internal dynamics of the system and provides a hint while designing a controller for this problem.

The Inverted Pendulum Plant model consists of a suspended rod, also called the pendulum, which is hinged freely at one end to a mobile fulcrum. The Inverted Pendulum is considered to be in a non-inertial frame of reference with respect to a fixed point on the Earth's or planet's surface under ambient conditions. This fixed point is the origin of the Mobile fulcrum, from where the control experiment is initialized. The pendulum in the upright position is the position in which the direction of the free end of the rod from the hinged end of the rod is opposite to the direction of gravitational acceleration. Since the rod can rotate around the hinge, the upright position is arbitrarily chosen to represent the rotation angle of 0°.

When the fulcrum is not accelerated, there exist two angles in which the rod can remain in the same state (of angular position and velocity) when no external noise affects the pendulum. The states at these two angles are known as equilibrium states. But these two points represent two different types of equilibrium:

1. Stable Equilibrium:
   This type of Equilibrium can be defined using the Inverted Pendulum concept as the point of equilibrium such that, if any external force acts on the pendulum, it undergoes transients and returns to the same equilibrium as the steady state. This stable equilibrium exists in the state where the pendulum is in the downward direction, or rotation angle of 180°.
2. Unstable Equilibrium:
   This type of Equilibrium defined using the Inverted Pendulum concept is the point of equilibrium such that, if any external force acts on the pendulum for a short duration, the pendulum undergoes transient state and the following steady state is different from the initial state. Hence, the force shifts the system from unstable equilibrium to stable equilibrium. This unstable equilibrium is in the upright position of the pendulum, where the angle is 0°.

Thus, the Inverted pendulum problem is a control problem where the pendulum must be shifted from a state of Stable Equilibrium to the state of Unstable Equilibrium and maintained at the Unstable Equilibrium even in the presence of noise or external forces acting on the pendulum.

The Cart Pole problem is a special case of the Inverted Pendulum, which defines the actuation subsystem. Here, the Inverted Pendulum is specifically hinged on to a cart, which is mounted on a track of finite length. The cart has 1 degree of freedom, and the position of the cart is measured from the centre of the track. In the Cart-Pole problem considered here, the length of the track is 2.4m.

6.2 Swing up strategies

The first task in the Inverted Pendulum or the Cart Pole problem is to bring the system from a state of Stable Equilibrium to a state of Unstable Equilibrium. This task is known as the Swing up strategy. Many types of controllers have been designed in the past to tackle this problem, for example [9]. Most of the controllers designed have linearized the Inverted Pendulum plant and applied methods based on Linear Time Invariant systems. But this approach is an approximate method and does not fully consider the exact plant dynamics. On the other hand, the linearization of plant is more accurate near the Upright position as the Linearization of trigonometric functions of the rotation angle around 0° is accurate. But this does not apply to the Swing up, as rotation angles are must higher. Thus, the Swing up task must be achieved through Non-linear control strategies. Many swing up strategies have been defined in the past. In [10], a Non-linear control strategy known as Energy Control has been explained, which is a robust approach. The [11] research article considers an approach specific to the Cart-Pole problem considering the restrictions due to limited track length. These two approaches have been defined in reference to the defined Cart-Pole problem in the next two sections.

6.2.1 Energy Control Method

The Energy control method considers only the angle of the pendulum and its angular velocity, and neglects the position and velocity of the Cart on the track.

The equation of motion for an inverted pendulum in non-linear form is given by:

$$\bar{I}_p\ddot{\theta} - mgl\sin\theta + mul\cos\theta = 0 \tag{6.1}$$

where θ is the angular displacement of the pendulum from the upright position
$\dot{\theta}$ is the angular velocity of the pendulum
$\ddot{\theta}$ is the angular acceleration of the pendulum

m is the mass of the pendulum

g is the gravitational acceleration

l is the half-length of the pendulum

$\bar{I}_p = I_p + ml^2$ where I_p is the Moment of Inertia of the Pendulum with respect to the centre of gravity

The state variables in the system are defined by θ and $\dot{\theta}$. The Stable Equilibrium corresponds to the state where $\theta = \pi$ and $\dot{\theta} = 0$, while the Unstable Equilibrium corresponds to the State where $\theta = 0$ and $\dot{\theta} = 0$.

Designing the control strategy involves determining the control input u that must be given to the Cart-Pole Plant Model in order to swing up the Pendulum. This control input must incorporate the limitations to the velocity of the pendulum, and consider the weight and mass of the pendulum. An Energy method is designed by considering the total energy of the pendulum at any state, given by:

$$E = \frac{1}{2}\bar{I}_p\dot{\theta}^2 + mgl(\cos\theta - 1) \qquad (6.2)$$

Here, $\frac{1}{2}\bar{I}_p\dot{\theta}^2$ represents the Kinetic Energy of the Pendulum in angular form. $mgl(\cos\theta - 1)$ represents the Gravitational Potential Energy of the pendulum w.r.t the upright position. $E = 0$ in the upright position, that is, when $\theta = 0$ and $\dot{\theta} = 0$. Since the Energy must also include the acceleration of Cart, which is the main Actuation signal, the derivative of Energy is considered:

$$\frac{dE}{dt} = \bar{I}_p\ddot{\theta}\dot{\theta} - mgl\dot{\theta}\sin\theta \qquad (6.3)$$

Using Eqn(6.1) from above…

$$\frac{dE}{dt} = -mul\dot{\theta}\cos\theta \qquad (6.4)$$

From this Eqn(6.4), the control signal can be determined. The system works as variable gain integrator, with non-Uniform controllability over the state space. The Controllability of the system is lost when the angular velocity $\dot{\theta} = 0$, which implies that the control signal must be a function of $\dot{\theta}$, so that $u = 0$, when $\dot{\theta} = 0$. Similarly, the Controllability of the system is also lost when the angle of the pendulum is $\theta = \pm\frac{\pi}{2}$, and the control signal has the most effect on the Plant when $\theta = 0 \; or \; \pi$. This is possible only when a trigonometric function of the pendulum angle is considered, $\cos\theta$. Thus, the Control signal depends on $\dot{\theta}\cos\theta$. Since the goal is to reach the upright position, where the Energy of the pendulum is given by $E_0 = 0$, the Control law used is:

$u = k(E - E_0)\dot{\theta}cos\,\theta.$

Here the Lyapunov function used to derive the control law is: $V = (E - E_0)^2/2$

In order to ensure reliability in the control, simple design strategy using On-Off system, and high dynamic property, a Bang-bang control approach of the derived control law is utilized. Further, the maximum value of the control signal is limited as per required application. Thus, the new control law is:

$$u = sat_{ng}(k(E - E_0)sign(\dot{\theta}\cos\theta) \tag{6.5}$$

where sat_{ng} is the saturation function, which saturates at the value of ng

n is the maximum rate of Energy change, given by $n = \frac{u_{max}}{g}$

$sign$ is the signum function, which achieves Bang-Bang control with respect to $\dot{\theta}\cos\theta$

The parameters n and k can be varied for a fixed Cart-Pole setup, in order to achieve different types of Swing-up procedures:

1. Single-swing Double switch behaviour: $n > 2$ and $k\ is\ large$
2. Multi-swing Behaviour: $n < \frac{4}{3}$ and $variable\ k$

6.2.2 Swing-up Control using a defined Lyapunov function with Restricted Cart Length

This method of Control, defined by [11] is based on a fixed Lyapunov function defined by the sum of mechanical energies squared and weighted cart velocities squared. Here, the Plant Model or the Inverted Pendulum dynamics model is considered as:

$$\overline{I}_p\ddot{\theta} + ml(\cos\theta)\ddot{x} - mgl\sin\theta = 0 \tag{6.6}$$

$$(M + m)\ddot{x} - ml(\sin\theta)\dot{\theta}^2 + ml(\cos\theta)\ddot{\theta} + b\,\dot{x} = F \tag{6.7}$$

where, $\bar{I}_p = I_p + ml^2$

$I_p =$ Moment of Inertia of the Pendulum w.r.t Center of gravity

$M =$ Mass of Cart

$m =$ Mass of Pendulum

$l =$ length of the pendulum from pivot to center of gravity

$g =$ acceleration due to gravity

$F =$ force exerted on the cart

$\theta =$ angular displacement of the pendulum from the upright position

$x =$ linear displacement of the Cart from the center of the track

From the Energy Control Method, the Energy of the pendulum is defined by Eqn (6.2). Since the pendulum must be made upright, the Energy $E_p = 0$ must be considered. Thus,

$$\frac{1}{2}\bar{I}_p\dot{\theta}^2 + mgl(\cos\theta - 1) = 0 \tag{6.8}$$

The swing up strategy must thus ensure that this Energy E_p , must converge to zero, in order to bring the angle of the pendulum to zero, or bring the pendulum to the upright position. Along with this requirement, since any cart track is only of a finite length, the cart's position must not cross these limits. The velocity of the cart must be restricted in order to prevent the Cart from crossing the limits. This restriction is imposed by converging the cart velocity towards zero. The convergence of E_p and the cart velocity $\dot{x}$ is performed by using the following Lyapunov function:

$$V = \frac{1}{2}({E_p}^2 + ml\lambda\dot{x}^2) \tag{6.9}$$

where $\lambda > 0$ is a parameter defined as per application. In addition to the Lyapunov function introduced in Energy control method, this Lyapunov function also considers the velocity of the Cart for limitation. Thus, the Control strategy includes the velocity limitation, and is given by:

$$u = u_a(E_p\dot{\theta}\, cos\,\theta - \lambda\dot{x}) \tag{6.10}$$

From this, the derivative of the Lyapunov function is:

$$\frac{dV}{dt} = -mlu_a(E_p\dot{\theta}\, cos\,\theta - \lambda\dot{x})^2 \tag{6.11}$$

The parameter λ is used to control the linear movement of the cart, so that it never hits the boundaries of the rails.

Implementation of the Swing up strategy on a real system requires the use of an Actuator that induces acceleration u on the Cart. As the Cart must undergo linear motion, a motor is chosen as the actuator. In [10] paper, a DC motor is considered. As the voltage input to motor, changes the flow of current and causes the motor to run at different speeds, the relation between the acceleration of the Cart by the motor and the Voltage to be supplied to the motor, must be defined. The DC motor equation is given by:

$$I_m\ddot{\theta}_m + B_m\dot{\theta}_m = T_m - T \tag{6.12}$$

where I_m is the Inertia of the motor
θ_m is the Angular displacement of the Rotor
B_m is the viscous-friction coefficient of the rotor on the brushes
T_m is the Torque that the motor generates
T is the Load Torque of the Cart to be moved

Since the Force exerted is $F = \frac{T}{r}$ and using the relations between the Torque and the Voltage to be applied, the Force exerted on the cart is given by:

$$F = -\frac{I_m}{r^2}\ddot{x} - \left(\frac{B_m}{r^2} + \frac{K_m K_b}{R_m r^2}\right)\dot{x} + \frac{K_m}{R_m r}V_c \qquad (6.13)$$

Thus, substituting F in the Cart-Pole dynamics considered, the following expression can be derived:

$$\ddot{x} = \frac{\Lambda_1 - \Lambda_2(\Lambda_3 + \Lambda_4 \mathrm{V}_c)}{\Lambda_5} \qquad (6.14)$$

where,

$$\Lambda_1 = m^2 l^2 g \sin\theta \cos\theta$$

$$\Lambda_2 = \bar{I}_p$$

$$\Lambda_3 = -\left(b + \frac{B_m}{r^2} + \frac{K_m K_b}{R_m r^2}\right)\dot{x} + ml\dot{\theta}^2 \sin\theta$$

$$\Lambda_4 = \frac{K_m}{R_m r}$$

$$\Lambda_5 = (ml\cos\theta)^2 - \bar{I}_p(M + m + \frac{I_m}{r^2})$$

The transformation from u to V_c is given by:

$$\mathrm{V}_c = \frac{\Lambda_5 u - \Lambda_1 + \Lambda_2\Lambda_3}{-\Lambda_2\Lambda_4} \qquad (6.15)$$

The parameters are considered as:

Parameter	**Value**
$\bar{I}_p$	0.0267 kgm^2
M	0.711 kg
m	0.209 kg
l	0.326 m
B	0
I_m	0 kgm^2
B_m	0
r	0.0194 m
K_m	0.257 kgm^2
R_m	2.32 Ω

Table 6.1: Cart-Pole model parameters

6.3 Stabilization

After Swing-up of the Pendulum, the Pendulum reaches a value around the upright position, but due to the nature of Energy control, it cannot maintain a constant upright position, and results in oscillations and continuous swings, in order to try to achieve the upright position of $\theta = 0$ and $\dot{\theta} = 0$.

Thus, the Pendulum must switch over to another Control strategy in order to maintain the upright position, without falling over and swinging up continuously. Many methods have been explored in the past, in this domain of the Stabilization of an inverted pendulum in the upright position. The main advantage of this task is that the value $\theta \approx 0$ throughout the task, because of which, the system can be linearized using $\sin\theta \approx \theta$ and $\cos\theta \approx 1$. Linear control strategies can now be applied on the Cart-Pole plant model, in order to maintain the upright position. One such method considered here, is the Linear Quadratic Regulator, also known as the LQR controller, detailed in the following section.

6.3.1 Linear Quadratic Regulator

The LQR method of control on the Cart-Pole system is also known as a method of Optimal Control as it causes the best possible behaviour for the defined Plant model, and can lead to maximum performance. The goals of the Optimal Control problem in the Cart-Pole system are two-fold:

1. To determine the control signal that causes the Cart-Pole dynamics to reach a target state or follow a target trajectory.
2. Maximize performance by minimizing a Cost function defined

Thus, the LQR is a simple and Robust Optimal control strategy for the application to the stabilization task of the Cart-Pole balancing problem. After the Cart-Pole dynamics are linearized about the upright position of the Pendulum, which is the Unstable Equilibrium position, the linear state space equation is given by:

$$\dot{X} = AX + Bu \tag{6.16}$$

where X is the state space vector, given by $X = [\theta, \dot{\theta}, x, \dot{x}]^T$

A Feedback must be supplied to the system in order to determine the error between the current state and the target state. This feedback is known as the State Feedback. The control of the State feedback to the system is given by:

$$u = -KX \tag{6.17}$$

Thus, Eqn(6.17) above, changes to

$$\dot{X} = (A - BK)X \tag{6.18}$$

A Cost function is defined in order to maximize the performance of the Controller by minimizing the Quadratic Performance Index, and drive the state X of the Pendulum towards the target state, assumed as the Origin, $X_0 = [0,0,0,0]^T$. This Cost function is given by:

$$J = \int (X^T Q X + u^T R u) dt \tag{6.19}$$

where J is the cost function,

Q and R are positive semi-definite and positive definite constant symmetric matrices

X is the state vector

u is the control input

A constant matrix of positive definite nature must be derived in order to define the LQR gain. This matrix is obtained by solving the Algebraic Riccatti Equation (ARE). The ARE is defined by:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{6.20}$$

where P is the constant positive definite matrix

A and B are matrices derived from the State space equation

Q is the positive semi-definite constant symmetric matrix defined in the cost function

R is the positive definite constant symmetric matrix defined in the cost function

Thus, the LQR gain vector K is given by:

$$K = R^{-1}B^T P \tag{6.21}$$

The LQR gain vector K corresponding to the stabilization task of the Cart-Pole system considered is designed in Matlab:

Step 1: Calculation of state space equation constants A, B, C, D

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{I_p}{(M+m+m^2l^2)} & \frac{m^2l^2g/I_p}{(M+m+\frac{m^2l^2}{I_p})} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{b}{\left(\frac{(M+m)I_p}{ml}\right)+ml} & \frac{(M+m)g}{\left(\frac{(M+m)I_p}{ml}\right)+ml} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{I_p}{(M+m+m^2l^2)} \\ 0 \\ \frac{-1}{\left(\frac{(M+m)I_p}{ml}\right)+ml} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Step 2:

$Q = C^T C$

$R = 1$

Step 3: Calculate K

$K = [-1.0000 \quad -1.7788 \quad -26.3106 \quad -3.8440]$

6.4 Integration of Swing-Up and LQR

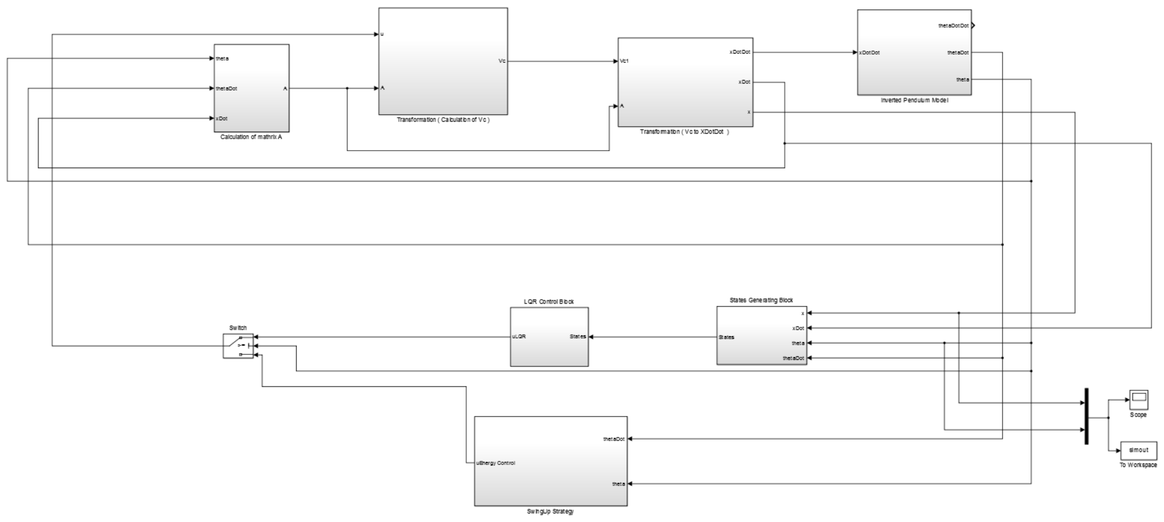


Fig. 6.1: Swing Up and LQR with switching

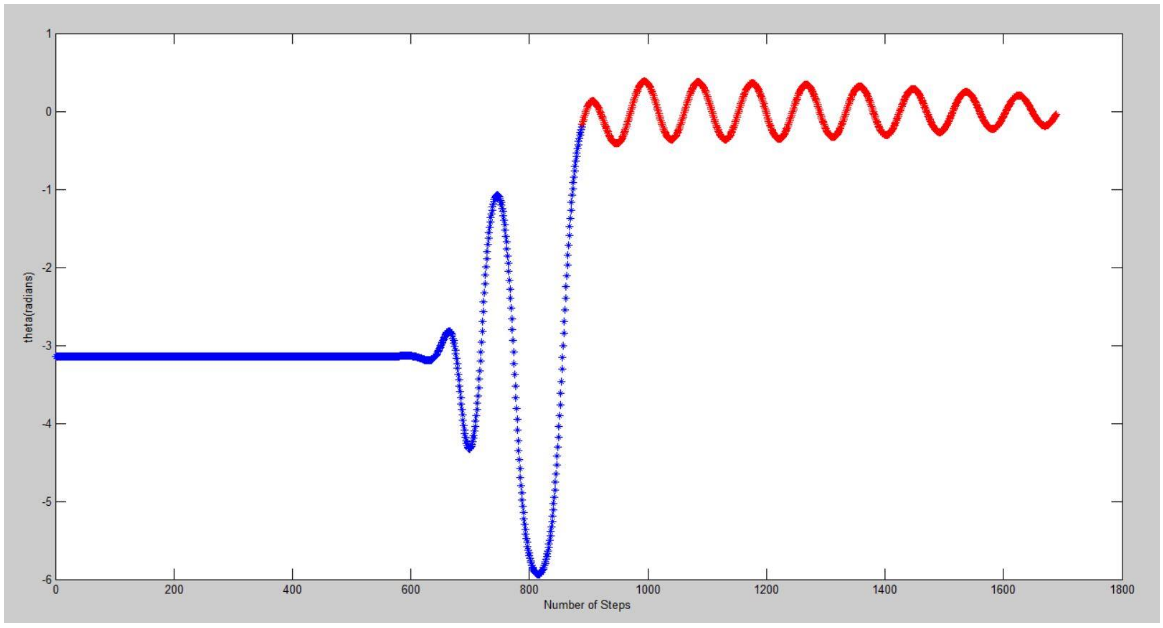


Fig. 6.2: Theta – Swing Up and LQR stabilization, switching at 8 degrees

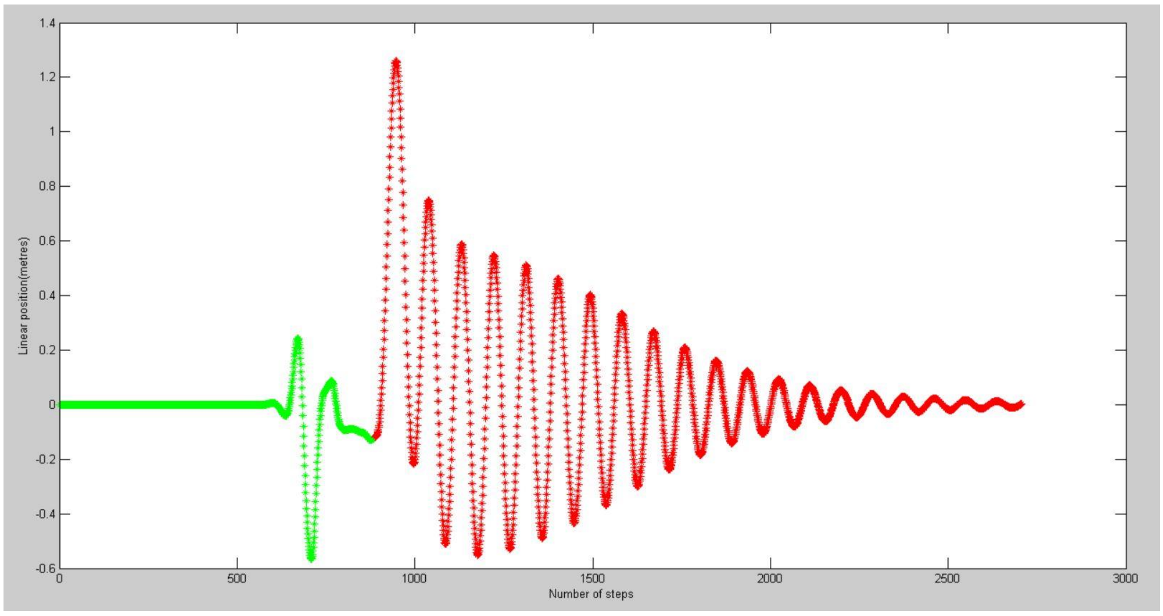


Fig. 6.3: Linear Position - Swing Up and LQR stabilization, switching at 8 degrees

6.5 Conclusions

The pendulum is initially at the stable equilibrium position. The Swing Up controller takes into account the non-linear dynamics of the cart pole system, and induces energy into the system by imparting force on the cart in the form of control signals. The length of the cart is taken into consideration while designing the Lyapunov function for the Swing Up controller, so that the cart never hits the boundaries of the rails. The parameter λ in the design determines the cart length limitations. The Swing Up is achieved in iterations, until the pendulum arrives within the range of pre-defined angular position values. Once the pendulum is within this range, the control is shifted to the Linear Quadratic Regulator, which works with the linearized system. The LQR controller provides an adaptive force to the cart, which decreases in magnitude as the states of the cart pole system comes closer to the upright equilibrium position. Finally, the pendulum is stabilized at the upright unstable equilibrium position. The Swing up and balance and the transition between these controllers is shown in Fig. 6.2 and 6.3.

Chapter 7: Integration of Swing Up Control with Reinforcement Learning Algorithms and Comparison

7.1 Comparison of LQR stabilization with RL stabilization

In chapter 6, it has been observed that a Linear Quadratic Regulator (LQR) control system is designed to generate an adaptive control signal, which is the force that is imparted on the cart based on the state of the inverted pendulum system, to achieve stabilization of the pendulum at the upright position. The adaptive force is generated based on the error between the equilibrium state and current state. Hence, the magnitude of the control signal (force) decreases as the pendulum gets closer to the equilibrium state. This type of adaptive control demands the knowledge of the complete dynamics of the system. The desired state of the system should be known, which is used to generate the adaptive control signal.

Chapter 5 constitutes of various Model free Reinforcement Learning algorithms applied on the Cart Pole problem to achieve balance at the upright position. The problem is formulated such that the dynamics of the cart pole system is not known a priori and only failure signal, which is the reinforcement, is given back to the system if the state of the inverted pendulum system goes out of a pre-defined range of state variable values. Also, unlike the LQR controller, where a force is generated by the controller, a constant force in either direction is given to the RL controller as the action to be taken. Starting with a random policy, the RL controller arrives at the optimal policy based on different reinforcement learning algorithms, and learns to maintain its state within the pre-defined state values.

The control achieved using an RL controller is classified as ‘Bang Bang’ control, due to the constant force given to the system. Even after arriving at the optimal policy and the RL controller knows the action it has to take for a particular state, the force imparted on the cart cannot be reduced. Hence, it is not an adaptive controller and in a physical design, the energy consumption is higher. Also, as the results of the various RL algorithms depict in chapter 5, even after arriving at the optimal policy, the angular position of the pendulum, and the linear position of the cart have a wider range of values when compared to LQR. The range decreases as we move from discrete to continuous state space, and the Linear Value Function Approximation algorithm, which uses continuous state spaces gives the results closest to that of the LQR controller.

But, an RL controller can adapt to an unknown environment and achieve optimal control, unlike the classical control techniques, where the entire controller must be replaced. Hence, an LQR controller, or any classical controller can be replaced with a controller using Reinforcement Learning algorithms to achieve optimal control.

7.2 Why Integration of Swing Up with RL controller?

A novel method has been introduced in this report, where a Swing Up controller has been integrated with an RL controller. The reasons are as follows:

1) The existing research on application of Reinforcement learning algorithms to Cart Pole problem specifies that the pendulum must be reset to the unstable equilibrium position after each failure. But, this method, when applied to a physical system seems redundant and cumbersome. Instead, the pendulum is reset to the stable equilibrium position each time failure occurs, and then swung up to a pre-defined value of angular position from where the control is switched to the RL controller.
2) Once the failure occurs, the pendulum reaches the stable equilibrium due to natural damping, from where Swing Up is initialized. So, the whole process can be automated.
3) When the pendulum is Swung Up to a window of pre-defined range of angular position values, the pendulum can end up in a different state each time. Hence, the scope for exploration increases

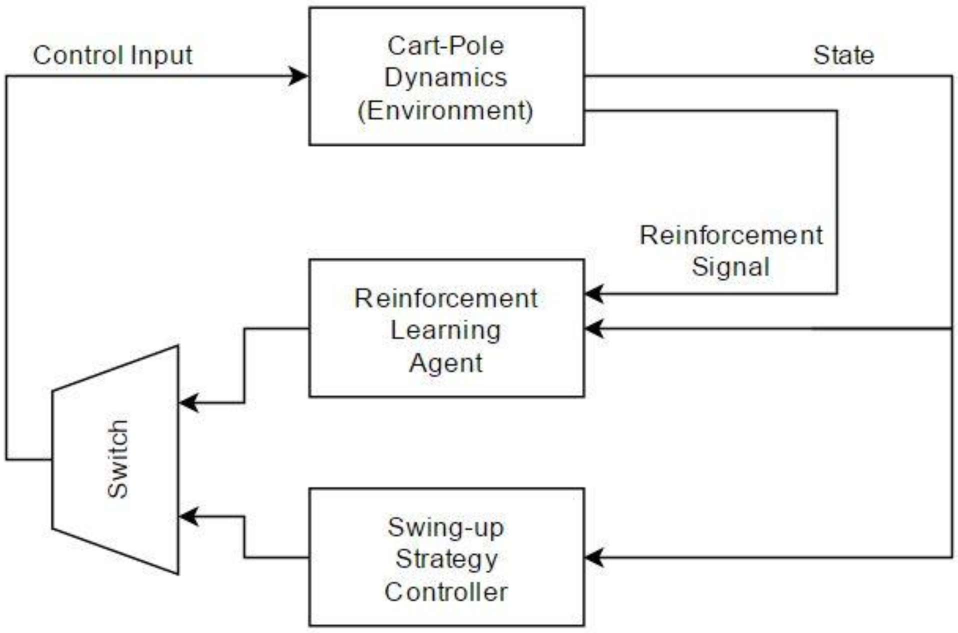
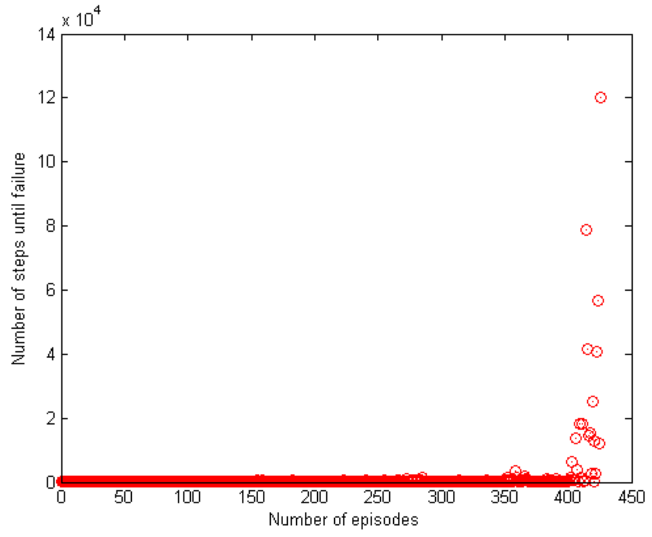

Control Input
Cart-Pole Dynamics (Environment)
State
Reinforcement Signal
Reinforcement Learning Agent
Switch
Swing-up Strategy Controller

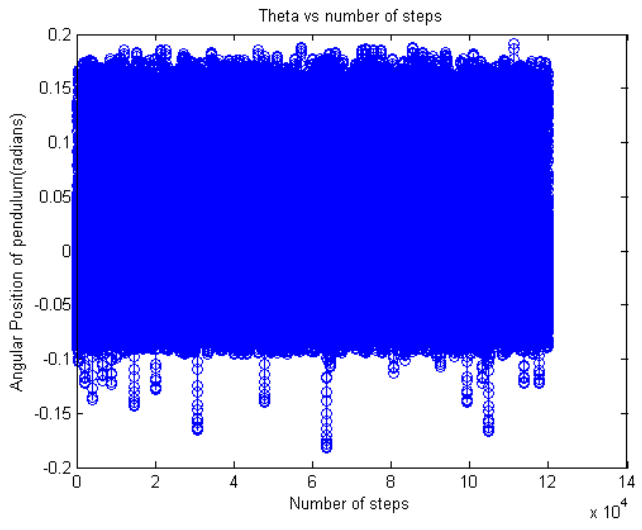
Fig. 7.1: Block diagram of Swing Up with RL

7.3 Results

The control is switched to RL controller from Swing Up controller after the angular position of the pendulum is greater than -12 degrees and less than 12 degrees. The initial state of the pendulum after each swing up is random. Each result is tested for a force of [10, -10] Newtons.

7.3.1 SARSA/Q





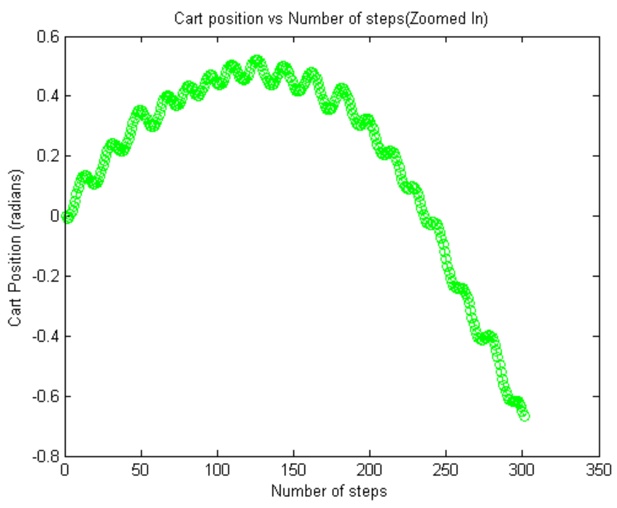
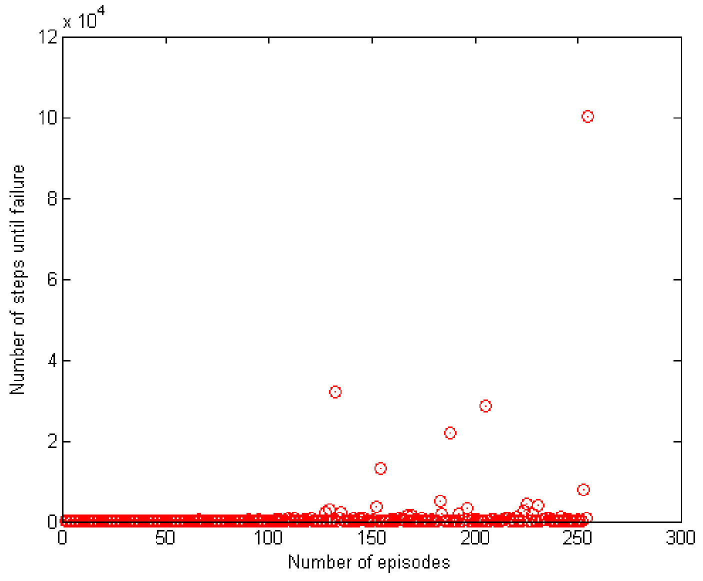
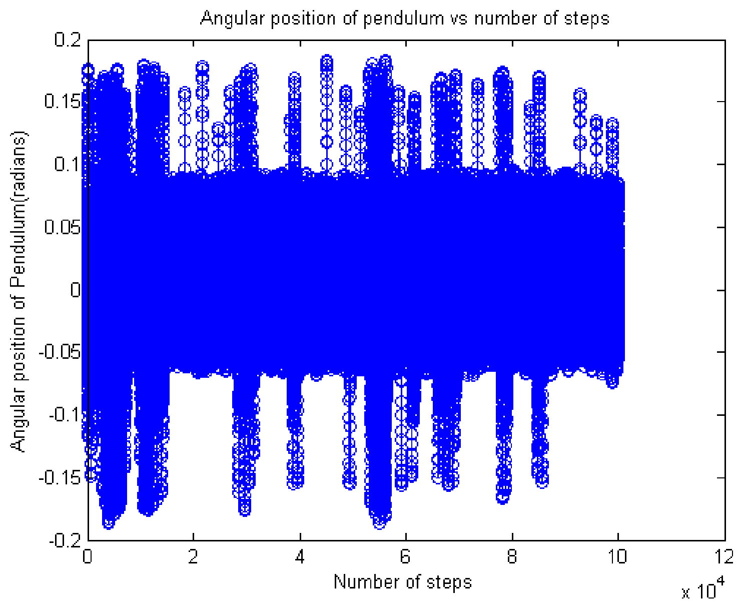


Fig. 7.2: Swing-Up with RL Sarsa

7.3.2 Actor Critic





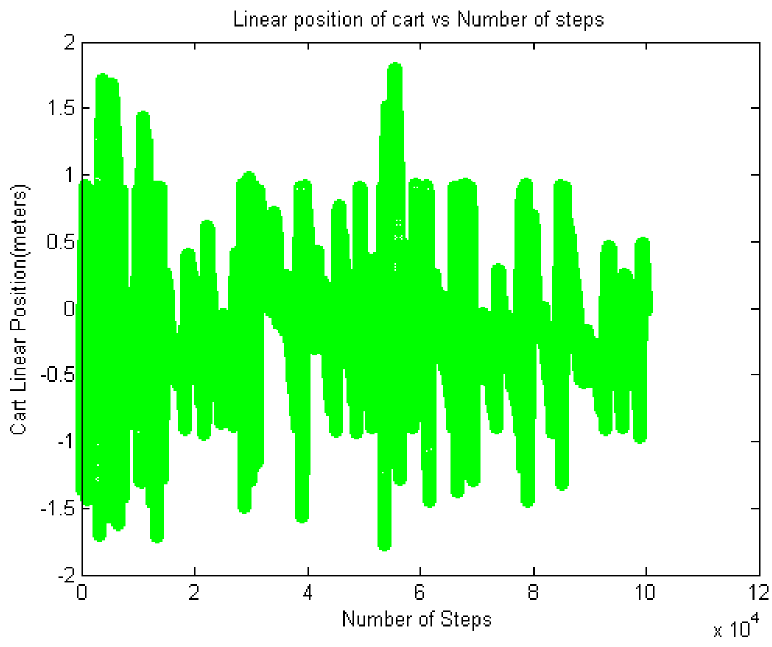
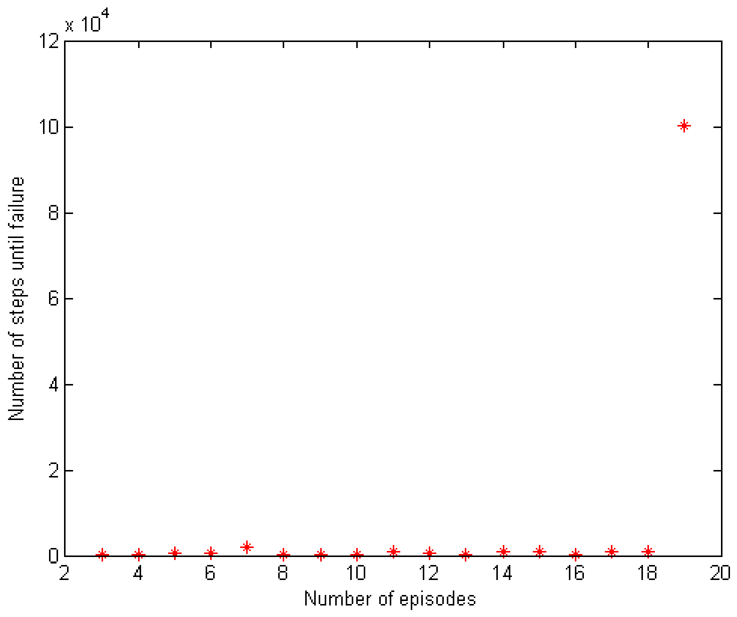
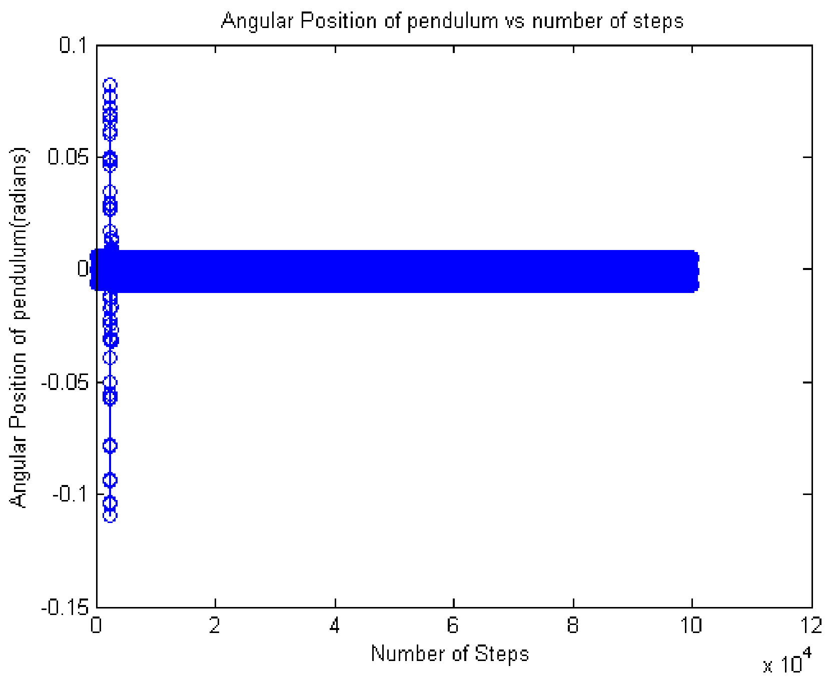


Fig. 7.3: Swing Up with RL Actor Critic

7.3.3 Value Function Approximation





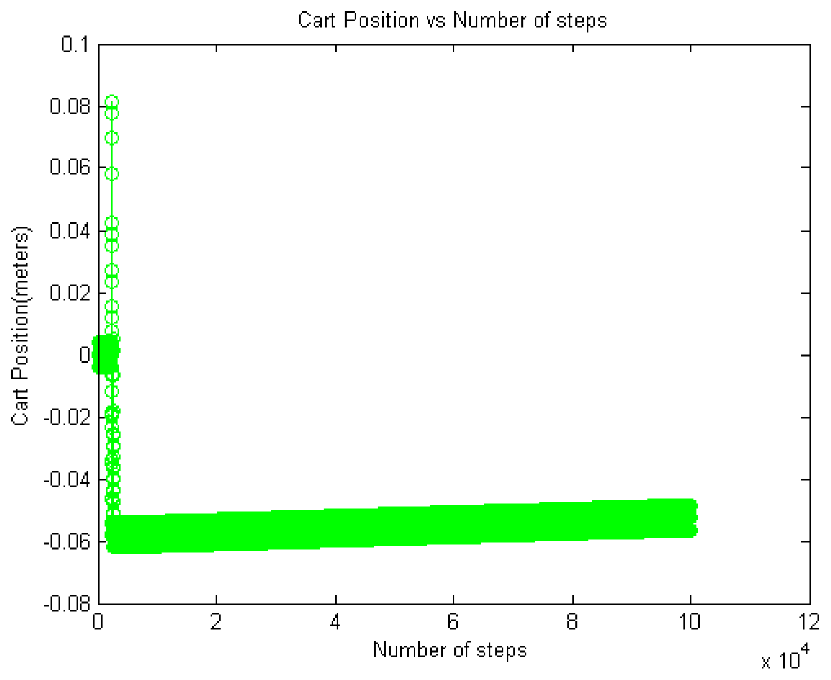


Figure 7.4: Swing Up with RL Linear Function Approximation

7.4 Comparison Table

Algorithm	**Force (in N)**	α	γ	**State-space Quantizer**	λ_w **and** λ_v	θ **Range (in °)**	x **Range (in m)**	**Episodes until optimal policy**
SARSA(0)	{+10, -10}	0.5	0.99	getBox	-	[-11,+10]	[-1.5, +2.4]	420
SARSA(0)	{+15, -15}	0.6	0.99	getBox2	-	[-2.9, +2.9]	[-1. +1]	380
SARSA(0)	{+30, -30}	0.4	0.99	getBox	-	[-3, +6.3]	[-0.1, +0.6]	24
Q-Learning	{+10, -10}	0.5	0.99	getBox	-	[-11,+10]	[-1.5, +2.4]	420
Q-Learning	{+15, -15}	0.6	0.99	getBox2	-	[-2.9, +2.9]	[-1, +1]	380
Q-Learning	{+30, -30}	0.4	0.99	getBox	-	[-3, +6.3]	[-0.1, +0.6]	24
Value Function Approximation	{+10, -10}	0.07	0.99	-	-	[-0.4, +0.3]	[-0.062, +0.054]	19
Policy Gradient Actor-Critic	{+10, -10}	1000	0.95	0.9	0.8	[-12, +12]	[-1.7, +0.2]	62
SARSA(0)/Q with Swing Up	{+10, -10}	0.5	0.99	getBox	-	[-8.6, +11.5]	[-0.6,+0.5]	430
Policy gradient Actor-Critic with Swing Up	{+10,-10}	1000	0.95	getBox	0.8, 0.9	[-8,11]	[-1.7,1.7]	255
Value function approx with Swing Up	{+10,-10}	0.07	0.99	-	-	[-0.4, +0.3]	[-0.062, +0.054]	19

Table 7.1: Comparison of Model-Free RL Algorithms with Swing-up

7.5 Conclusion

From the above table, it is clear that with the addition of Swing Up, the performance of the Reinforcement Learning Controller has not been affected. But, the system is now completely automated, without any external assistance. The scope for exploration of the state space has been increased as the swing up ensures the randomized initialization of states for the Reinforcement Controller.

Chapter 8: Conclusions and Future Work

8.1 Conclusions

Reinforcement Learning has been explored in the context of optimization and control problems. Various Reinforcement Learning algorithms have been used to solve the Jack's Car rental problem and the Cart Pole Inverted pendulum problem, which are the benchmark problems in optimization and control respectively. These problems have been chosen so that various Reinforcement learning algorithms can be understood in perspective of these problem statements. In the course of the project, various aspects of application of reinforcement learning on a given problem have been studied:

1) Separating the problem into agent and environment.
2) Obtaining states, rewards, transition probabilities and actions from a given problem and model the MDP.
3) Applying RL algorithms on the modelled MDP to solve the given problem.
4) If the complete knowledge of an MDP is not known, model free reinforcement learning algorithms are applied to solve the given problem.

The observations and inferences from the project are:

1) Multi Arm bandit algorithm works with the trade-off between exploration and exploitation very effectively, and is applicable to a problem with a single state and multiple actions.
2) A complete knowledge of MDP is necessary to solve a problem using Dynamic programming. In Dynamic Programming, the agent goes through all the next states from a given state. So, it is computationally expensive and is redundant to apply on a control problem.
3) The Policy iteration algorithm, which is a DP algorithm, was used to solve Jack's Car rental problem effectively, where Policy evaluation and Policy improvement were carried out alternatively to optimize the number of transfers between the two locations and maximize the reward. For the given environment, the problem was solved within four policy iterations.
4) The range of angular positions of the pendulum and linear positions of the cart for the optimal policy decreases with the transition from discrete to continuous state space.
5) The Policy gradient Actor Critic algorithm takes significantly lesser time to achieve optimal policy than Q learning and Sarsa.

6) After learning the optimal policy, even though the pendulum is within the defined state limitations, the states of the cart pole system never settles down at the upright position, for Actor Critic, Sarsa and Q learning controllers. The angular position of the pendulum and the linear position of the cart have a wide range of oscillations as a result of Bang-bang control with a constant force pair of [10,-10] Newtons.
7) Tuning the learning rate, discount factor, increasing the magnitude of the force, and/ or increasing the number of state bins will narrow the range of oscillations.
8) The linear value function approximation controller has a better performance when compared to the other discrete state space model free controllers, and the system has the narrowest range of oscillations for the same force pair of [10,-10] Newtons. The performance of this controller is similar and comparable to that of the LQR controller.
9) The performance of control is not affected by the integration of an RL controller with a Swing Up controller. The initial state of every episode is now the stable equilibrium position that is reached through natural damping. After each Swing Up, the pendulum reaches a different state bin, which randomizes the initial state for the RL controller. Hence, this increases the scope for exploration.

8.2 Future Work

Reinforcement Learning has a large variety of applications, and can be used to solve complex problems that are persistent in the world today. While significant amount of research is being pursued in this field, the algorithms used have become more complex and are specific to the applications for which the algorithms are designed. The advantage of this approach is that a specific problem can be solved quickly, without the knowledge of the dynamics and internal working of the problem itself. However, one of the long term goals and challenges of Artificial Intelligence researchers is to approach a point of General AI, where one algorithm or a set of algorithms can be used to solve multiple, if not all the problems that could otherwise be solved by separate AI algorithms. In the view of General AI, Reinforcement algorithms must also be unified and generalized. Extending the algorithms applied in the previous sections, from the problems explored, to large classes of problems in Control and Optimization domains can be really helpful to achieve General AI. For this,

1. A generalized Markov Decision Process which can describe all the problems in the domain must be designed
2. Reinforcement Learning Algorithm must be chosen in order to satisfy the Bias-Variance trade-offs in all the problems of the domain
3. Hyper-parameters of the RL algorithm should either be general to all problems or mathematically derived from the MDP description and the Objective
4. The RL Agents must be able to handle all types of Input and Output data, and handle all types of delayed Reinforcement signals

The Jack's Car Rental problem, is an optimization problem and has been optimally solved through the Dynamic Programming approach described in Section 3.4. Exploring a large and complex version of the problem might reveal further advantages and disadvantages of solving the problem through this approach. Methods to extend the problem may include:
1. Introducing a larger number of Rental stores owned by Jack
2. A decayed Reinforcement based on the depreciation of the car's value or adjustment to inflation in the real world
3. Larger and varying number of cars allowed in a Rental store on each day
4. Removing the assumption that the Rental and Return of cars is defined by a Poission distribution

The Cart-Pole Balancing problem is a fundamental problem in Non-linear control systems. As Non-Linear Control has a large number of applications such as Process control, Robotics, Defense and Transportation technology, the Reinforcement Learning algorithm should be further improved to perform complex tasks in all Non-Linear Control applications optimally. To approach complex Non-Linear controls from the Cart-Pole balancing problem, the following changes should be made:

1. Replacement of the Cart-Pole MDP with complex application specific MDPs.
2. Improvement of the RL algorithm in order to deal with Non-linear, Stochastic and Non-stationary environments
3. Selection of hyper-parameters of the RL Agent in order to ensure Optimal policy is reached.

From the perspective of the Reinforcement Learning algorithms explored so far, Tabular methods have been the focus and have been compared with a Value function Approximation method which considers a Linear Combination of features. The Value function approximation method has resulted in an optimal policy that converges faster and minimizes the requirement of the actions to be taken multiple times, based on the current states, which is a major disadvantage of the Tabular Methods. With respect to the Cart-Pole balancing problem, the following changes in the RL algorithm can be explored further:

1. Usage of complex Differentiable Function approximators such as Neural Networks on the problem
2. Check for the suitability of complex non-differentiable function approximators such as Wavelet basis, Decision trees, and Fourier basis.
3. Exploration of Loss functions other than the Mean squared error
4. Exploration of Gradient Descent Methods other than the Stochastic Gradient Descent, such as Batch Gradient Descent
5. Exploration of Optimization techniques other than Gradient Descent
6. Application of function approximation to Actor-Critic methods
7. Application of function approximation to Policy Gradient methods
8. Optimization and Learning algorithms applied on the Hyper-parameters of the RL Agent used to solve the Cart-Pole problem, in order to achieve the best Hyper-parameter set
9. End-to-End Reinforcement Learning algorithms for the Cart-Pole problem for various applications that involve Inverted Pendulum.

In this report, we have explored the root level Reinforcement Learning algorithms on optimization and control problems. The understanding of these algorithms will be the motivation to explore new, complex and problem specific algorithms. The fact that we have solved the Cart Pole problem, which is the benchmark non-linear control problem, using Reinforcement Learning algorithms proves that Reinforcement Learning algorithms can be applied to more complex, non-linear, dynamic systems.

References

1. R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Vol. 1. No. 1. Cambridge: MIT press, 1998.
2. A. Dutech, et al. "Reinforcement learning benchmarks and bake-offs II." *Advances in Neural Information Processing Systems (NIPS)* 17 (2005).
3. C. W. Anderson "Learning to control an inverted pendulum using neural networks." *IEEE Control Systems Magazine* 9.3 (1989): 31-37.
4. A. G. Barto, R. S. Sutton, and C. W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems." *IEEE transactions on systems, man, and cybernetics* 5 (1983): 834-846.
5. D. Michie and R. A. Chambers. "BOXES: An experiment in adaptive control." *Machine intelligence* 2.2 (1968): 137-152.
6. R.S. Sutton "Learning to predict by the methods of temporal differences." *Machine learning* 3.1 (1988): 9-44.
7. R. S. Sutton “Temporal credit assignment in reinforcement learning” (1984)
8. N. Muskinja and B. Tovornik. "Swinging up and stabilization of a real inverted pendulum." *IEEE Transactions on Industrial Electronics* 53.2 (2006): 631-639.
9. K. Furuta, M. Yamakita, and S. Kobayashi. "Swing up control of inverted pendulum." *Industrial Electronics, Control and Instrumentation, 1991. Proceedings. IECON'91., 1991 International Conference on*. IEEE, 1991.
10. M. Iwashiro, K. Furuta, and K. J. Astrom. "Energy based control of pendulum." *Control Applications, 1996., Proceedings of the 1996 IEEE International Conference on*. IEEE, 1996.
11. J. Yang, et al. "Swing-up control for an inverted pendulum with restricted cart rail length." *International Journal of Control, Automation and Systems* 7.4 (2009): 674-680.