Reinforcement learning for optimal control problems in continuous time and space

Anton Forsell

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6258
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2024

Abstract

This thesis considers the problem of finding an optimal control policy for control problems in continuous time and space, utilising reinforcement learning techniques. Finding analytical solutions to optimal control problems is often intractable due to involved constraints, cost-function objectives, and uncertainty. The focal point will be on reinforcement learning RL techniques to find approximate replacements for analytical solutions; or as a tool to create initial control policies, enabling findings and potential problems early in the design process. The thesis will put focus on model-free stochastic RL methods and how introduction of quasi-stochastic noise can be used as a tool for variance reduction. Specifically, we study an off-policy temporal difference TD learning method to find optimal control policies based on the quasi-stochastic approximation QSA method. In this approach, data gathering is done by sampling the system with an entirely unrelated policy to the policy to optimize.

The results show that the algorithm produces policies of increasing quality with each iteration, in many cases reaching optimality. The algorithm was tested on linear systems of varying complexity, although still relatively small, and a basic nonlinear system. We discussed how applicable the method is to optimal control problems and concluded that it is applicable to a majority of systems due to having neither harder nor softer requirements than most optimal control optimisers.

Contents

1

Introduction

1.1 Project description

Analytical solutions to optimal control problems are unavailable or difficult to obtain, due to involved constraints and cost function objectives. The focal point of the thesis will be on reinforcement learning RL techniques applied to control problems in continuous time and space to find optimal control policies. The thesis will highlight an algorithm to find optimal control as an extension of calculus of variations, and algorithms associated with machine learning ML and computer science.
The thesis will study and implement an RL approach to find a solution to continuous-time optimal control problems.

1.2 Purpose and research questions

The primary goal of the thesis is to study how RL is used to solve optimal control problems, highlighting and identifying similarities between recent RL works from an ML perspective.

1.3 Problem formulation

Optimal control attempts to solve the problem of finding the control law for a given system, such that a selected optimality criterion is achieved. Finding an optimal control sequence requires minimizing a given cost function c. The cost function is a function of the state and control variables x and u at a time t; the function returns a value indicating how good the control law is at a specific time.
The thesis will focus on minimizing the continuous-time cost functional J defined as:

$$\min_{u \in \mathbb{R}^m} J(x, u) = \min_{u \in \mathbb{R}^m} \int_0^\infty c(x(t), u(t))dt,\ x \in \mathbb{R}^n, u \in \mathbb{R}^m. \tag{1.1}$$

where the function c is a cost function based on the states x and the control signal u, according to:

$$c(x(t), u(t)) : \mathbb{R}^{n \times m} \rightarrow \mathbb{R} \tag{1.2}$$

The optimization is subject to the constraints of the general state equation $f(x, u)$:

$$\dot{x} = f(x(t), u(t)) : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^{n} \tag{1.3}$$

1.4 Methodology

In the first phase, a literature study will be conducted to build a theoretical background. Further studies include implementation and experimenting with the algorithm found within the theoretical background. Such experiments will try to find if the algorithm is feasible and find an optimal infinity-horizon solution.

1.5 Potential methods from computer science

Present-day research has had a focus on RL applications on robotics and continuous space problems, such research is recapitulated in [Laezza, 2021]. A large part of the summarised literature adopts a machine-learning perspective. An early potential RL framework for solving such problems has been suggested by [Doya, 2000]. Recent advances in RL and ML and triggered research into solving continuous-time optimal control problems using the principles of Q-learning. Characteristic examples are [Vamvoudakis, 2017a] and [Bernstein et al., 2019].

Short description of key papers [Doya, 2000] introduces a reinforcement learning framework for continuous-time dynamical systems without a priori discretization of time, state, and action. The proposed framework is model-based and derives algorithms for improving policies, estimating value functions, and function approximators.

[Vamvoudakis, 2017a] highlights that to achieve the combination of optimal and adaptive control, one could draw inspiration from RL. Q-learning is one of the first algorithms to solve this problem. It is a reinforcement learning technique developed primarily for continuous-time systems and is model-free.

In the paper [Bernstein et al., 2019] introduces the theory regarding a quasi-stochastic approximation QSA, where all processes under consideration are deterministic, rather than stochastic. A major application of QSA is a type of RL algorithm for deterministic state space models.

The previous three papers focus heavily on solving control problems, the last reference focuses more on reinforcement learning from a machine learning perspective. [Laezza, 2021] focuses on the use of machine learning for robotics. The licentiate contains thorough descriptions and explanations of how reinforcement learning works and an updated algorithm for reinforcement learning.

2

Theoretical framework

2.1 Reinforcement learning

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize some notion of cumulative reward [Sutton and Barto, 2018, pp. 1–22]. Unlike supervised learning, where the model is trained on a dataset with input-output pairs, reinforcement learning involves learning from the consequences of actions, rather than being told the correct actions. [Sutton and Barto, 2018, pp. 1–22] explains that the agent interacts with the environment, receives feedback in the form of rewards or penalties, and uses this feedback to improve its future actions. The main goals of reinforcement learning are to develop policies that maximize long-term rewards and to understand the trade-offs between exploration and exploitation.

Online learning and offline learning

Models trained from offline learning do not change their approximation of the goal function after the initial training phase has been completed. Offline learning is commonly referred to as batch learning because it needs to be re-trained to accommodate new training data. Online-trained models are capable of changing their approximation of the goal function as more information is available. The feature of being able to adapt to new training data on the fly makes online training harder.

On-policy learning and off-policy learning

According to [Sutton and Barto, 2018, pp. 100–103] both methods are ways to ensure that sufficient information is gathered to conclude that optimality is reached. On-policy methods aim to evaluate or improve the decision-making policy, which in turn generates new data to further the policy; while off-policy methods aim to evaluate or improve a policy different from the data-generating policy [Sutton and Barto, 2018, pp. 100–103].

Monte Carlo methods

Monte Carlo (MC) methods only require samples of state, action pairs, and rewards from simulations or interactions with a model or environment, unlike many other methods which require assuming complete knowledge of the environment [Sutton and Barto, 2018, pp. 91–117]. Furthermore, [Sutton and Barto, 2018, pp. 91–117] states that learning from simulated experience is powerful and does not require prior knowledge of the dynamics of the environment, but it can still achieve optimal behavior.

These methods are used to solve the reinforcement learning problem by averaging the sample returns [Sutton and Barto, 2018, pp. 91–117]. Monte Carlo methods require well-defined returns; one common way to guarantee this is to define Monte Carlo methods only in episodic tasks [Sutton and Barto, 2018, pp. 91–117]. This results in methods that can be incremented in an episode-by-episode sense, but not in a step-by-step (online) sense [Sutton and Barto, 2018, pp. 91–117].

[Sutton and Barto, 2018, pp. 91–117] clarifies that the term ”Monte Carlo” is not exclusively used for these types of methods, but rather methods that involve a significant random component.

Monte Carlo integration and quasi-Monte Carlo integration Both of these methods are ways to estimate an integral based on sampling; they differ predominately in how the exploitative noise is generated. Monte Carlo methods use random numbers to explore the domain of a function, while quasi-methods use pseudo-random numbers to fill the domain more regularly [Asmussen and Glynn, 2007, pp. 265–273].

$$\int f(x)dx = \frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{2.1}$$

(2.1) where the selection of x_i is the random or pseudo-random numbers generated by the two methods [Asmussen and Glynn, 2007, pp. 265–273].

Temporal difference learning

Temporal difference combines Monte Carlo methods with methods from dynamic programming [Sutton and Barto, 2018, pp. 119–140]. [Sutton and Barto, 2018, pp. 119–140] highlights that TD learns from raw experiences without a model of the environment, like Monte Carlo methods. Additionally, TD behaves like dynamic programming (DP) methods, TD updates estimated values partially on other learned estimates, without waiting for an outcome (bootstrapping). Bootstrapping or updating based on current estimates is the primary difference between TD methods and MC methods; the latter requires complete episodes to perform an update.

Methods such as Monte Carlo and temporal difference blend into each other, complement each other, and can be combined [Sutton and Barto, 2018, pp. 119–140]. A way of combining these two methods is by sampling using Monte Carlo and then bootstrap with dynamic programming [Sutton and Barto, 2018, pp. 119–140]. An advantage of TD learning is that it can be implemented in an online setting, where Monte Carlo is only available in an episode-by-episode offline setting [Sutton and Barto, 2018, pp. 119–140].

Neural networks as function approximators

Artificial neural networks (ANNs) are widely used for non-linear function approximation. These networks are interconnected units that resemble neurons and have some of the properties of neurons [Sutton and Barto, 2018, pp. 223–228]. [Sutton and Barto, 2018, pp. 223–228] highlights two types of ANN feedforward and recurrent networks. In feedforward, the output from a network has no path to affect the input; in recurrent networks, this is possible and is often called loops.

Networks generally consist of an input layer, an output layer, and one, several, or thousands of hidden intermediary layers. Between the units in the hidden layers are activator functions that produce the output values of the units [Sutton and Barto, 2018, pp. 223–228]. ANNs with a singular hidden layer can approximate any continuous function in a compact region of the network input space [Sutton and Barto, 2018, pp. 223–228], [Cybenko, 1989]. However, [Sutton and Barto, 2018, pp. 223–228] highlights that this is in practice not the case, and deeper networks with more layers are often needed.

2.2 Linear quadratic regulator - LQR

In the LQR case, the system function $f(x, u)$ of (1.3) is now linear and takes the form (2.2), and the cost function $c(x, u)$ is quadratic and of the form (2.3). The value function $J(x, u)$ in this case is the same as the previous and restated in (2.4).

$$\dot{x} = Ax + Bu \tag{2.2}$$

$$c(x(t), u(t)) = x^T Mx + u^T Ru \tag{2.3}$$

$$J(x, u) = \int_0^\infty c(x(t), u(t))dt. \tag{2.4}$$

It has been shown that problems of this form have optimal control feedback (2.5). The gain K is optimal if it is equivalent to (2.6), and the variable P is the solution to the continuous Algebraic Riccati equation (CARE) (2.7).

$$u = Kx \tag{2.5}$$

$$K = -R^{-1}B^T P \tag{2.6}$$

$$A^T P + PA - PBR^{-1}B^T P + M = 0 \tag{2.7}$$

From equation (2.6). We can see that if we can find an approximation R and $B^T P$ we can find an optimal control solution to the LQR problem.

A common way to find iterative improvements is to use the function Q (2.8), associated with the LQR problem [Vamvoudakis, 2017b].

$$Q(x(t), u(t)) = \begin{bmatrix} x & u \end{bmatrix} \left(\begin{bmatrix} M & 0 \\ 0 & R \end{bmatrix} + \begin{bmatrix} A^T P + PA + P & PB \\ B^T P & 0 \end{bmatrix} \right) \begin{bmatrix} x \\ u \end{bmatrix} \tag{2.8}$$

If we find the gradient of the Q function (2.8) with respect to the control signal u, set the resulting derivative to 0; we will see that the minimization will exactly correspond to the optimal solution (2.6). The Q function is improved during computation and will eventually converge towards the optimum.

3

Method

In this section, we are going to explore some of the reinforcement learning theories, to highlight a method to find the minimum to the continuous-time cost functional J (1.1), thus creating a solution to the formulation of the problem (1.3). The general approach for reinforcement learning problems is to compute or approximate the expected reward of a policy, the expected reward is generally called the reward- or value-function. With an approximation of the value function, the next step is to improve the policy; the step in which decision-making is improved. These two steps are iterated until we find a policy that maximizes the reward. However, we are not looking to maximize the reward of a policy; we are looking to find a solution to our problem formulation. That is, we are looking for a policy that minimizes the value function J (1.1). Repeating these two steps is called policy iteration and is a core concept of reinforcement learning.

Approximating the value function can be done in many ways, on-policy or off-policy, model-based or model-free, and online or offline; to name a few concepts. We will highlight a reinforcement learning method from [Bernstein et al., 2019], which is a Q-learning method with quasi-stochastic noise. Q-learning is an off-policy temporal difference (TD) learning approach [Sutton and Barto, 2018]; for model-free policy evaluation and policy improvement.

3.1 Policy iteration with quasi stochastic noise for exploration

The goal of the method is to minimize the value function J (1.1), for $u(t)$, reiterated here:

$$\min_{u\in\mathbb{R}^m} J(x,u) = \min_{u\in\mathbb{R}^m} \int_0^\infty c(x(t),u(t))dt,\ x\in\mathbb{R}^n, u\in\mathbb{R}^m. \tag{3.1}$$

$$J^\phi(x) = \int_0^\infty c(x(t),\phi(x(t)))dt \tag{3.2}$$

$$\phi^{+}(x(t)) = arg \min_{\phi} J((x(t), \phi(t)) \tag{3.3}$$

The goal is to evaluate the cost function J for a given policy $\phi(x(t))$ (3.2). Using the results from (3.2) to generate an improved policy ϕ^{+} (3.3).

Constraints

Additional constraints are:

- The state function $f(x(t), u(t))$.
- The cost function c(x(t), u(t)) is selected, or at least known.
- The value function J is convex and twice continuously differentiable

Overview of the proposed method

An overview diagram is depicted in the figure. 3.1, the method can be simplified to these four basic steps:

- Create an appropriate basis function and derivative for each system, select a starting point
- Sample the system with quasi-stochastic exploration
- Perform the steepest descent optimization to find an optimal θ^{*} for that specific policy
- Utilise θ^{*} to perform a policy iteration and compute a new better policy ϕ^{+}.

Exploration with quasi stochastic noise

A common dilemma with reinforcement learning is whether one should improve the current policy (exploitation) or attempt to find a new policy (exploration). In this method, we have selected an off-policy method for exploration, with a feedback law $u(t)$ entirely unrelated to the policy to optimize.

A feedback policy with “excitation” is chosen, of the form

$$u(t) = \kappa(x(t), \xi(t)) \tag{3.4}$$

where κ and ξ are such that the resulting state trajectories are bounded for each initial condition (x_0), and the joint process (x, u, ξ) admits an ergodic steady state.

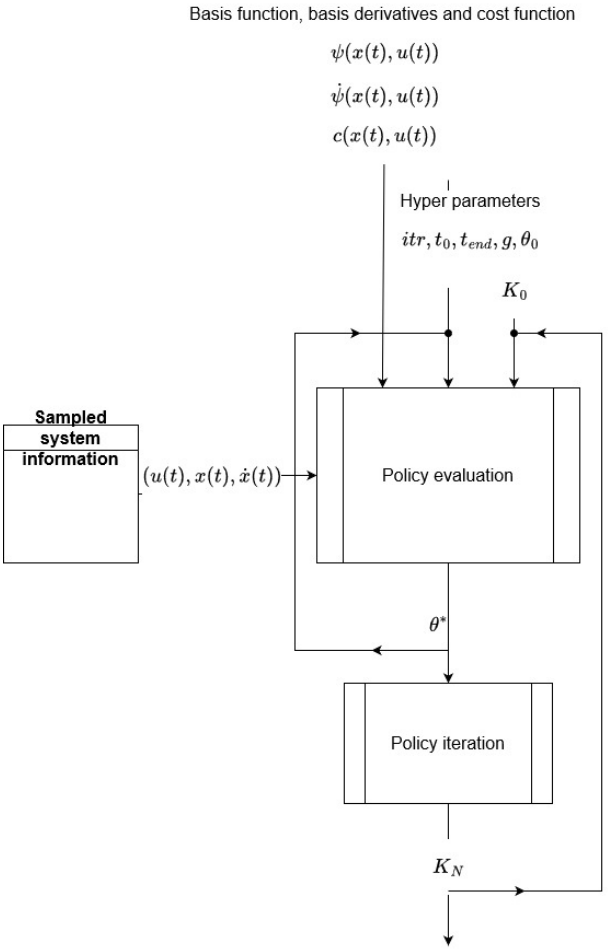


Figure 3.1 Diagram showcasing the value approximation and policy improvement for each iteration for the proposed method. $\psi(x(t), u(t))$ and $\dot{\psi}(x(t), u(t))$ is the basis function and its derivative. The cost function $c(x(t), u(t))$ in conjunction with ψ, $\dot{\psi}$ and the initial policy gain K_0 are supplied by the user and are tied to the controlled process. While the hyperparameters $(itr, t_0, t_{end}, g, \theta_0)$ e.g. (number of iterations, start time, end time, gain, and initial guess for θ) are selected by the user and are tuned separately from the system.

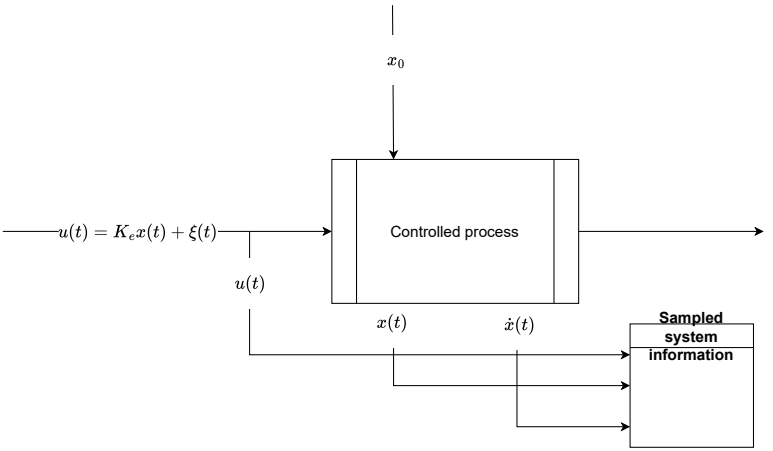


Figure 3.2 Diagram showcasing how off-policy exploration and sampling is done. K_e is the feedback gain, $\xi(t)$ is a noise function, $x(t)$ is the state variables, $\dot{x}(t)$ is the derivative of the state variables and $u(t)$ is the control signal.

Gathering of data For the algorithm we need to gather data by simulating or sampling the system, this is done by running the system in an open loop with the feedback law $u(t)$ (3.4) and sampling the variables (x, $\dot{x}$, u). An overview diagram of the process is depicted in Figure 3.2.

Approximating the value function

The first step of the algorithm is to evaluate the value function with respect to a given policy $\phi(x(t))$, we will utilize TD learning for its ability for model-free optimization and bootstrapping, that is update the estimated value function continuously in an online-sense.

Off-policy TD learning The construction begins with a Q-function defined with respect to the given policy:

$$Q^{\phi}(x(t), u(t)) = J^{\phi}(x(t)) + c(x(t), u(t)) + g(x(t), u(t)) \cdot \nabla J^{\phi}(x(t)) \quad (3.5)$$

We consider a family of functions $Q^{\phi\theta}(x, u)$ parameterised by θ, and define the Bellman error for a given parameter as

$$\epsilon^{\theta}(x, u) = -Q^{\phi\theta}(x, u) + Q^{\phi\theta}(x, \phi) + c(x, u) + f(x, u) \cdot \nabla Q^{\phi\theta}(x, \phi) \tag{3.6}$$

and the 2-norm:

$$\|\epsilon^{\theta}\|^2 = \lim_{T\to\infty} \int_0^T [\epsilon^{\theta}(x(t), u(t)]^2 \, dt \tag{3.7}$$

The Q-function (3.5) will be used for policy evaluation. The goal is to create an algorithm that minimizes the Bellman error for the given norm (3.7). This is done by computing a parameter θ^* for the parameterized Bellman error (3.6), where the error (3.7) is small.

The goal is to find θ^* that minimizes the mean square error (3.7). The first-order optimality condition is formulated as a root-finding problem. To optimize the parameter θ, we will use the steepest descendent in ODE form as:

$$\begin{aligned} \frac{d}{dt}\theta(t) &= -a(t)\epsilon^{\theta(t)}(x(t), u(t))\zeta^{\theta(t)}(t) \\ \zeta^{\theta}(t) &= \nabla_{\theta}\epsilon^{\theta}(x(t), u(t)) \end{aligned} \tag{3.8}$$

Model-free realisation From here on out we will use the notation x, u in functions for $x(t), u(t)$. From the Bellman error (3.6) we can see that the nonlinear model $f(x, u)$ needs to be known; however, recognising that for any parameter θ, and state-input pair $(x(t), u(t))$ the Bellman error (3.6) can be rewritten as:

$$\epsilon^{\theta}(x, u) = -Q^{\phi\theta}(x, u) + Q^{\phi\theta}(x, \phi) + c(x, u) + \frac{d}{dt}Q^{\phi\theta}(x, \phi) \tag{3.9}$$

Introducing a basis function We want to turn the optimization of (3.7) into a linear regression problem in θ. This is the case when the Bellman error is a linear function of θ. We introduce a basis of functions $\psi_i : 1 \leq i \leq d$ and get a linearly parameterized family of Q-functions, as:

$$Q^{\phi\theta}(x, u) = d(x, u) + \theta^T\psi(x, u), \; \theta \in \mathbb{R}^d \tag{3.10}$$

Substituting (3.10) into (3.9) and simplifying you will see that the Bellman error is now linear in θ.

Return to steepest descent With the changes we introduce a new steepest descent algorithm for optimisation:

$$\frac{d}{dt}\theta = -a(t)\left[\zeta(t)^T\theta(t) + b(t)\right]\zeta(t) \tag{3.11}$$

With variables $\zeta(t)$, $b(t)$, and $a(t)$ defined as:

$$\zeta(t) = \psi(x, \phi) - \psi(x, u) + \frac{d}{dt}\psi(x, \phi) \tag{3.12}$$

$$b(t) = c(x, u) - d(x, u) + d(x, \phi) + \frac{d}{dt}d(x, \phi) \tag{3.13}$$

$$a(t) = \frac{1}{1 + t} \tag{3.14}$$

Approximate policy improvement - PIA

From the previous section, we arrive at an approximation of the value function, with an θ for a given policy $\phi(x(t), u(t))$. Given an initial non-optimal policy, we need to go through multiple iterations of evaluating the value function and then updating the policy. This section will focus on updating the optimal policy (3.3) given an approximation of the value function with the optimal variable θ^*.

$$\phi^+(x) = arg\min_u Q^{\phi\theta^*}(x, u) \tag{3.15}$$

Taking the expression for the Q function $Q^{\phi\theta^*}(x, u)$ from (3.10) the policy improvement (approximation) becomes the following:

$$\phi^+(x) = arg\min_u d(x, u) + \theta^T\psi(x, u),\ \theta \in \mathbb{R}^d \tag{3.16}$$

From here on out we will use the notations such as θ_{uu}, this notation comes from $\theta^T\psi$. Each individual θ_i^T corresponds to a basis ψ_i for $1 \leq i \leq d$ and θ_{uu} would mean the θ_i^T for which the corresponding basis ψ_i is quadratic in u. A more formal definition:

$$\theta_{uu} = \frac{\partial}{\partial u}\frac{\partial}{\partial u}\theta^T\psi$$
$$\theta_{xx} = \frac{\partial}{\partial x}\frac{\partial}{\partial x}\theta^T\psi$$
$$\theta_{xu} = \frac{\partial}{\partial x}\frac{\partial}{\partial u}\theta^T\psi$$
$$\theta_{ux} = \frac{\partial}{\partial u}\frac{\partial}{\partial x}\theta^T\psi$$

To find the optimum to (3.16) we find a derivative of the expression and set the results equal to 0, extract all the u variables onto one side, and arrive at the new policy:

$$u = -\frac{1}{2}\theta_{uu}^{-1}\theta_{xu}x \tag{3.17}$$

The full policy improvement approximation (PIA) becomes:

$$\phi^+(x) = K_N = -\frac{1}{2}\theta_{uu}^{-1}\theta_{xu} \tag{3.18}$$

which approximates the optimal solution gained from solving the CARE function $K_N = R^{-1}B^TP$ with:

$$\theta_{uu} = R \tag{3.19}$$

$$\theta_{xu} = B^TP \tag{3.20}$$

The nonlinear case

With the highlighted method we will have trouble expressing policies that are nonlinear. Although the theory extends to nonlinear systems, the current method as written can not produce nonlinear policies, due to the PIA step. A potential solution is to utilize ANNs as these have the properties of nonlinear function approximators 2.1. An alternative is to realize that there are cases where a better informative guess of basis function for Q and u can be done to handle nonlinearities, highlighted in the next section.

An example to highlight the nonlinear case. Observe the nonlinear case:

$$\begin{aligned} f(x,u) &= \begin{bmatrix} -x_1 + x_2 \\ -0.5(x_1 + x_2) + 0.5x_1^2x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ x_1 \end{bmatrix} u \\ c(x,u) &= x^Tx + u^2 \end{aligned} \tag{3.21}$$

from [Luo et al., 2014], which has the optimal solution $u(x,u) = -x_1x_2$; the policy improvement algorithm (3.18) can not express this policy. A different way to express the policy is as:

$$u = K_N \begin{bmatrix} x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 \end{bmatrix}^T \tag{3.22}$$

The optimal policy $u = -x_1x_2$ can now be expressed as:

$$u = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 \end{bmatrix}^T \tag{3.23}$$

(3.23) highlights how one can express nonlinearities as a part of the PIA.

Replacement of theta optimization with a neural network. As mentioned in Section 2.1 ANNs have the properties of nonlinear function approximators. We want to approximate an Q^ϕ function parameterized by θ in an attempt to find local minima θ^*. The local minima are then used in a PIA step to generate our new control law ϕ, and the process is repeated. The approximation of the value function and the computation of an optimal θ^* could be replaced by an ANN, to achieve the same results using a different reinforcement learning method. Furthermore, it is possible for ANN to approximate the cost-to-go function J directly, avoiding the PIA improvement step. These methods are only mentioned to highlight different methods of finding an optimal control solution and will not be experimented with in this paper.

Continuous update of the policy

Currently, the policy update step is performed after one iteration. However, TD learning continuously updates the value function approximation, an alternative would be to train the primary policy during simulations by updating it at regular intervals.

3.2 Implementation details

Programming environment

Python was the environment selected for the implementation of the QSA algorithm. A language with preexisting packages and functions for solving different types of control systems and ordinary differential equations (ODE), it was between Python and Matlab, but Python was chosen due to recency bias. The packages utilized in the implementation were Numpy and Scipy due to their ability to solve ODEs and generally better handling of vectors and matrices. Additionally, Scipy has functions for solving the CARE and DARE functions, which are useful for verifying the solutions in the LQ cases.

Algorithm 1 QSA

procedure QSA($f, c, d, \frac{d}{dt}d, \psi, \frac{d}{dt}\psi, PIA, K_e, K_0, X_0$)
  for $i \leftarrow$ to N **do**
    $t \leftarrow t_0$
    t_points, $x(t) \leftarrow sample\_system(Ke, f)$
    $ode \leftarrow LSODA(\theta\_func(), t_0, y_0, t_{end})$ ▷ Any scipy ODE solver
    while $t \leq t_{end}$ **do**
      $ode.step()$
      $y \leftarrow ode.y$
      $t \leftarrow ode.t$
    end while
    $\theta \leftarrow$ last y
    $K_N \leftarrow PIA(\theta)$
  end for
  return K_N
end procedure

procedure $\theta\_func(t, \theta)$
  return $-a(t)\left[\zeta(t)^T\theta(t) + b(t)\right]\zeta(t)$
end procedure

procedure ζ(t)
  return $\psi(x, \phi) - \psi(x, u) + \frac{d}{dt}\psi(x, \phi)$
end procedure

procedure b(t)
  return $c(x, u) - d(x, u) + d(x, \phi) + \frac{d}{dt}d(x, \phi)$
end procedure

Algorithm 2 Continuous update of the policy

procedure QSA($f, c, d, \frac{d}{dt}d, \psi, \frac{d}{dt}\psi, PIA, K_e, K_0, X_0$)
  for $i \leftarrow$ to N **do**
    t_points, $x(t) \leftarrow sample\_system(Ke, f)$
    $ode \leftarrow LSODA(\theta\_func(), t_0, y_0, t_{end})$ ▷ Any scipy ODE solver
    for $t \in range(t_0, t_{end}), step\_size$ **do** ▷ Update policy every few second equal to step size
      while $t - 1 \leq t$ **do**
        $ode.step()$
        $y \leftarrow ode.y$
        $t \leftarrow ode.t$
      end while
      $\theta \leftarrow$ last y
      $K_N \leftarrow PIA(\theta)$
    end for
  end for
  return K_N
end procedure

4

Results and findings

4.1

Systems utilized for experimentation

Linear Quadratic systems. For these systems, the cost function follows the same shape (the quadratic part of LQ) and is defined as:

$$c(x, u) = x^T M x + u^T R u \tag{4.1}$$

where only the matrices M and R are differentiated for the four linear systems. For a linear system with a quadratic cost, we can always find an optimal feedback law u and gains K_N of the forms $u(x(t)) = K_N x$, from the CARE.

System 1 The first system under consideration is small and simple. The system is an example from [Bernstein et al., 2019], where the system is a double integrator with friction.

$$S1 = \begin{cases} A = \begin{bmatrix} 0 & 1 \\ 0 & -0.1 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 10 \end{bmatrix} \end{cases} \tag{4.2}$$

The second system complicates the control law u from being a scalar to being vector-valued, and the optimal gain K_N becomes a matrix, rather than a vector.

$$S2 = \begin{cases} A = \begin{bmatrix} 0 & 1 \\ 0 & -0.1 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\ M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \end{cases} \tag{4.3}$$

The third system increases the size of the vector x but returns to a scalar control law u.

$$S3 = \begin{cases} A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & -0.1 & -0.1 \\ 0 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \\ M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 10 \end{bmatrix} \end{cases} \tag{4.4}$$

The fourth system is a combination of the increase in complexity from the previous two systems, increasing both the dimensionality of u and the amount of states x.

$$S4 = \begin{cases} A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & -0.1 & -0.1 \\ 0 & 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \\ M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \end{cases} \tag{4.5}$$

Nonlinear systems

The theory behind the algorithm makes no assumptions on the linearity of the system. This section contains a basic example that displays this property. The system is from [Luo et al., 2014].

$$S5 = \begin{cases} f(x,u) = \begin{bmatrix} -x_1 + x_2 \\ -0.5(x_1 + x_2) + 0.5x_1^2 x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ x_1 \end{bmatrix} u \\ c(x,u) = x^T x + u^2 \end{cases} \tag{4.6}$$

4.2 Practical implementation and numerical examples

This section aims to highlight appropriate and concrete values for the variables introduced in the method, within the context of the previously introduced systems 4.1.

Values tied to the individual systems

Each of the systems has a few variables tied to individual systems and is always the same (one exception) for all of the experiments associated with the mentioned system. The first variable is our initial guess K_0 for what an optimal feedback gain might be.

Exploration and data gathering. Following the data gathering which can be done separately and before the main method, we have just a few input variables in (K_e, ξ, x_0), the starting point x_0 and the feedback gain K_e was selected such that the resulting state trajectories are bounded for each initial condition in conjunction with the noise ξ introduced for exploration. Further, the noise ξ used was constructed from the following function:

$$\xi(t) = \sum_{j=1}^{q} a_j sin(w_j t + \phi_j) \tag{4.7}$$

where a_j was sampled uniformly between $[0, amp]$, where the amp is a separate parameter that was not locked to the system and is considered the only hyperparameter of the data gathering step. w_j, was sampled uniformly between 0 and 50 rad/s and finally ϕ_j was sampled uniformly between $[0, 2\pi]$. For cases where the control signal $u(t)$ was not a scalar, the noise for each u_j was initialized and sampled separately from each other.

As an alternative for the the quasi-stochastic noise (4.7), we use stochastic noise in an experiment, in this experiment Gaussian noise was used.

Basis- and cost-functions. The cost functions were introduced with the systems in 4.1 and were all quadratic. Due to them all being quadratic, we have opted to use the same basis function for all the systems, highlighted in this equation:

$$\psi(x(t), u(t)) = vech([x, u] \otimes [x, u]^T) \tag{4.8}$$

where $vech$ is the half-vectorisation and $\otimes$ is the Kronecker product.

Initial weight guess. Additionally, the starting point θ_0 (initial guess for weights)for the linear systems was sampled from:

$$\theta_0 = |\mathcal{N}(0, 20)| \tag{4.9}$$

and for the nonlinear example, the values were the same every time and were selected to be a guess near the expected optimum. These were tied to the systems.

Hyper parameters

The hyperparameters are $(itr, t_0, t_{end}, g, amp)$ e.g. (number of iterations, start time, end time, gain, and amplitude of noise) and are varied between experiments for the same system.

Basic configuration during initial implementation and experiments with system (4.2) we found values for the hyperparameters that returned great results, these were selected as a basic configuration and used for initial trials with the other systems. These found values were:

- itr, the number of iterations was selected to be 10.
- $t_0 = 0$, starting time was selected to be 0.
- $t_{end} = 50$, we found that the optimization for a given policy converged fairly early and 50 seconds was more than enough, more time increased computation time but was unwarranted due to close to zero improvement.
- g, the gain is required to be sufficiently high, but larger values cause instability. Selected due to good convergence without any instability.
- θ, initial weight guess.
- $amp = 10$, the amplitude associated with the quasi-stochastic noise ξ in exploration, was found to give a large exploration of the system, but while still achieving the requirements on bounded state trajectories.

Table of experiments

Table 4.1 Overview of experiments and their associated hyperparameters

ID	System	t_0	t_{end}	itr	amp	g	other	nbr of runs
E1	S1	0	50	10	10	20		30
E2	S2	0	50	10	10	20		95
E3	S3	0	50	10	10	20		30
E4	S4	0	50	10	10	20		31
E5	S5	0	50	10	10	20		24
E6	S2	0	50	10	10	20	$*_1$	16
E7	S2	0	50	10	10	20	$*_2$	11
E8	S2	0	50	10	20	20		20
E9	S3	0	50	30	10	20		16
E10	S4	0	50	30	20	20		8
E11	S1	0	50	10	10	20	$*_3$	30
E12	S1	0	50	10	10	20	$*_4$	30

- $*_1$ the function $a(t)$ (3.14), was changed to decrease towards 0 at a slower pace.
- $*_2$ the initial value x_0 for simulation was changed to be randomised.
- $*_3$ the noise function $\xi(t)$ was changed from quasi stochastic to stochastic noise.
- $*_4$ the policy is updated regularly during iteration, rather than only after.

4.3 Simulation results

Overall we can observe from table 4.2 that the algorithm works, but results in varying qualities of results. The reason is that the configuration used for all systems was experimented with and found by repeated changes to the first system. The next section called Enchantments displays some more results from tweaking the configuration to the specific problem and shows what performance improvements could be gained from such a change.

Table 4.2 Table displaying the average error for all the systems tried.

ID	average error	lowest error	highest error	normalised variance
E1	0.02006	0.00004	0.0837	0.02568
E2	0.26559	0.00514	0.57861	0.05718
E3	0.11700	0.01091	0.36500	0.05996
E4	0.17226	0.05983	0.38962	0.03061
E5	0.26917	0.08767	0.47931	0.05031

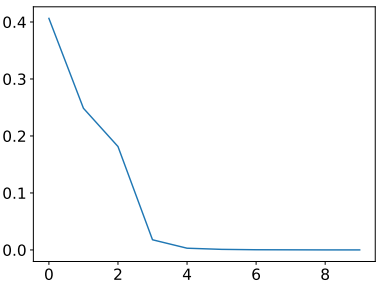
Experiment 1 - E1 for system S1 (4.2)**.** The results obtained from numerous simulations generate an average error of 0.020, showing a case where the algorithm converges to the optimum solution. Observing Figure 4.1, we can see a clear improvement in each iteration and a low variance between the simulations.

Experiment 2 - E2 for system S2 (4.3)**.** The performance of the algorithm for the second system was significantly worse, giving an average error of 0.2656 the worst of the linear systems. If we look at Figure 4.2 we can see improvement for each iteration; however, there appears to be a bit of stationary error, evident by slow convergence at the latter iterations, see subfigures 4.2a and 4.2b.

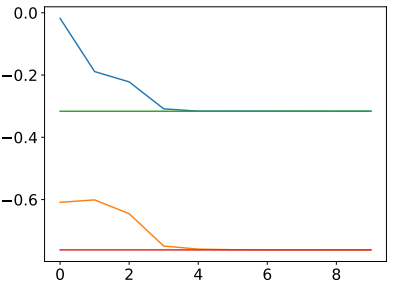
Experiment 3 - E3 for system S3 (4.4)**.** This system performed similarly to the first system, with an average error of 0.1170 a lower error compared to E2 for system 4.3. However, if we observe Figure 4.3 we can see that, unlike the second system, there appears to be no stationary error, as the error still decreases with later iterations.

Experiment 4 - E4 for system S4 (4.5)**.** The fourth system performed very similarly to system 3 with an error slightly higher at 0.1722, we can see similar slow improvement at later iterations by observing Figure 4.4.

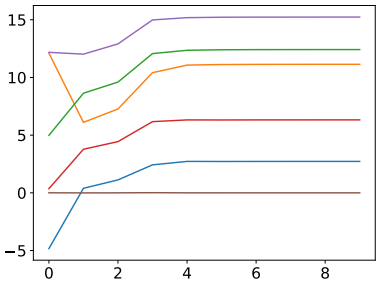
Experiment 5 - E5 for system S5 (4.6)**.** The fifth system performed the worst with an average error of 0.4020, however, unlike the other system, it is still improving the Figure 4.5, and will probably reach similar results with more iterations.



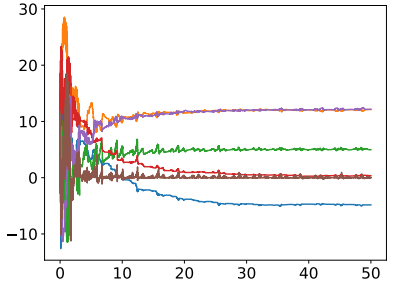
(a) Example of $\frac{||K^* - K_N||}{||K^*||}$ developing across all 10 iterations.



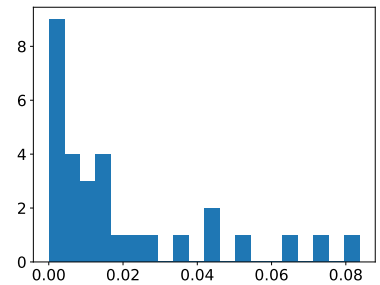
(b) Example of K_N developing across all 10 iterations.



(c) Example of θ developing across all 10 iterations.

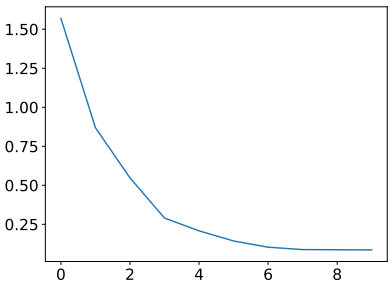


(d) Example of a singular iteration of θ developing across 50 seconds of simulated data.

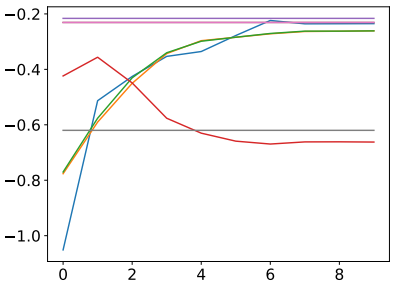


(e) An histogram showcasing the values $\frac{||K^* - K_N||}{||K^*||}$ for all simulations

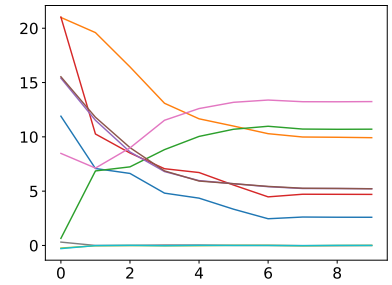
Figure 4.1 Figures showcasing some interesting information for multiple runs of experiment E1. The first two rows of sub-figures display how an average simulation looks. The last row displays the error for all simulations of this system.



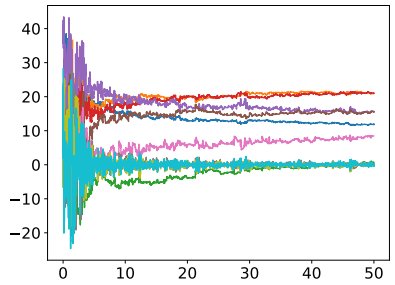
(a) Example of $\frac{||K^* - K_N||}{||K^*||}$ developing across all 10 iterations.



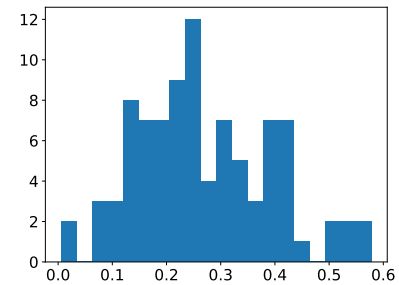
(b) Example of K_N developing across all 10 iterations.



(c) Example of θ developing across all 10 iterations.

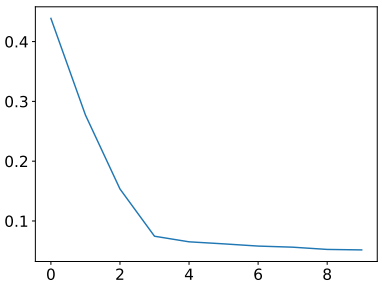


(d) Example of a singular iteration of θ developing across 50 seconds of simulated data.

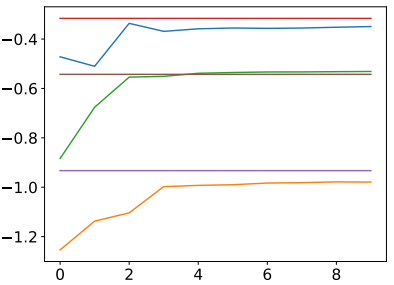


(e) An histogram showcasing the values $\frac{||K^* - K_N||}{||K^*||}$ for all simulations

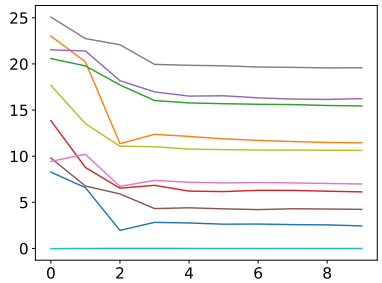
Figure 4.2 Figures showcasing some interesting information for multiple runs of experiment E2. The first two rows of sub-figures display how an average simulation looks. The last row displays the error for all simulations of this system.



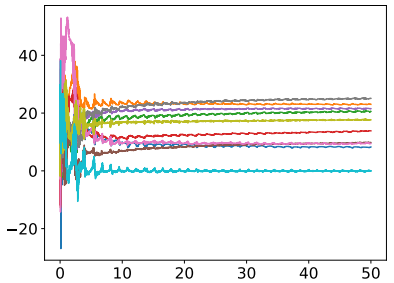
(a) Example of $\frac{||K^* - K_N||}{||K^*||}$ developing across all 10 iterations.



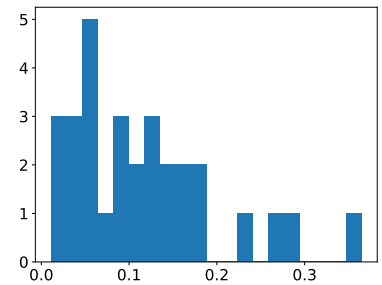
(b) Example of K_N developing across all 10 iterations.



(c) Example of θ developing across all 10 iterations.

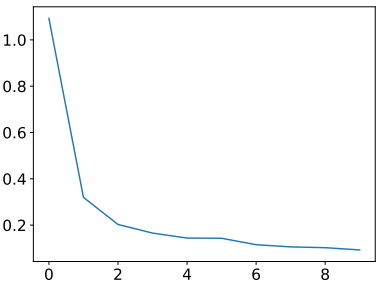


(d) Example of a singular iteration of θ developing across 50 seconds of simulated data.

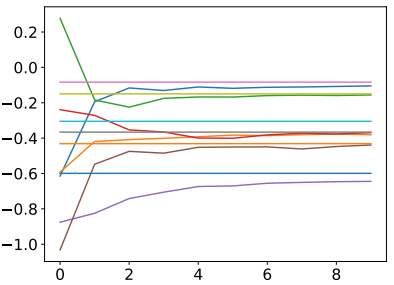


(e) An histogram showcasing the values $\frac{||K^* - K_N||}{||K^*||}$ for all simulations

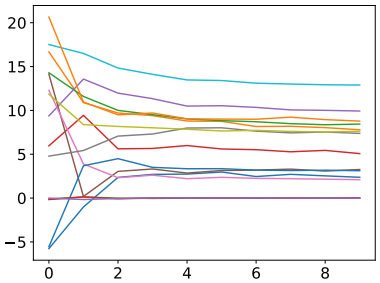
Figure 4.3 Figures showcasing some interesting information for multiple runs of experiment E3. The first two rows of sub-figures display how an average simulation looks. The last row displays the error for all simulations of this system.



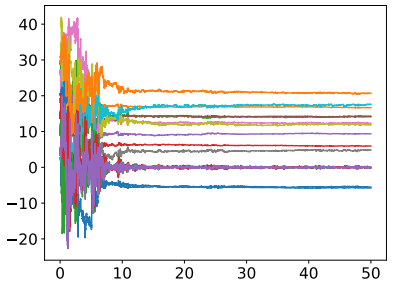
(a) Example of $\frac{||K^* - K_N||}{||K^*||}$ developing across all 10 iterations.



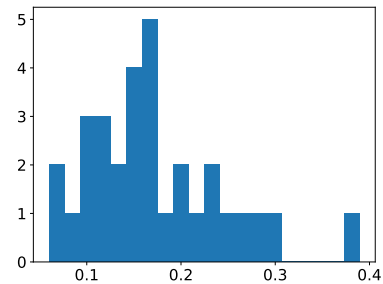
(b) Example of K_N developing across all 10 iterations.



(c) Example of θ developing across all 10 iterations.

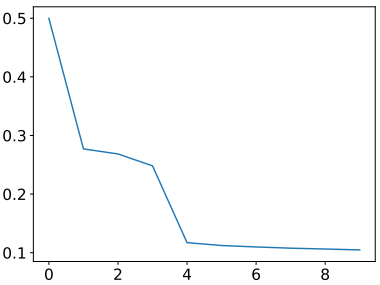


(d) Example of a singular iteration of θ developing across 50 seconds of simulated data.

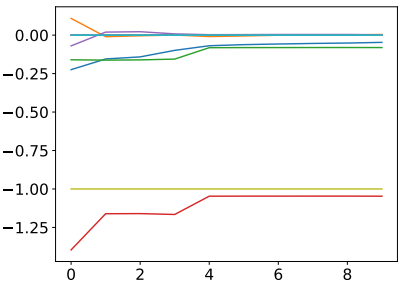


(e) An histogram showcasing the values $\frac{||K^* - K_N||}{||K^*||}$ for all simulations

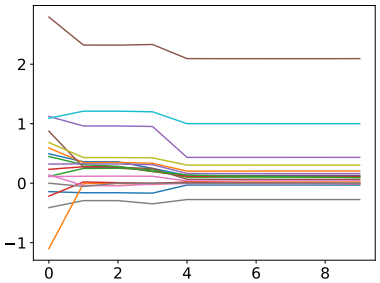
Figure 4.4 Figures showcasing some interesting information for multiple runs of experiment E4. The last row displays the error for all simulations of this system.



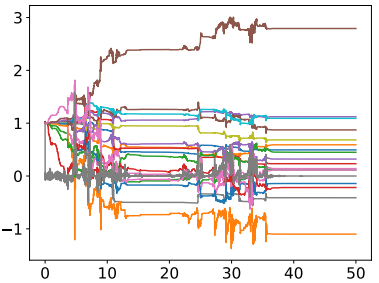
(a) Example of $\frac{||K^* - K_N||}{||K^*||}$ developing across all 10 iterations.



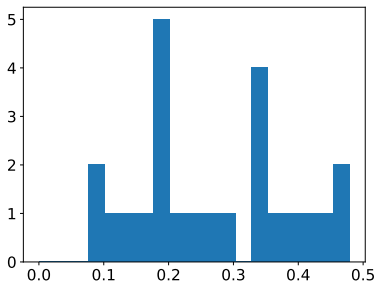
(b) Example of K_N developing across all 10 iterations.



(c) Example of θ developing across all 10 iterations.



(d) Example of a singular iteration of θ developing across 50 seconds of simulated data.



(e) An histogram showcasing the values $\frac{||K^* - K_N||}{||K^*||}$ for all simulations

Figure 4.5 Figures showcasing some interesting information for multiple runs of experiment E5. The first two rows of sub-figures display how an average simulation looks. The last row displays the error for all simulations of this system.

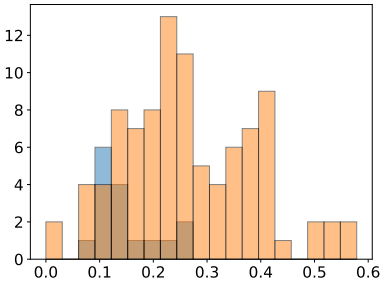
Further experiments to improve the results

In this section, we will check if changes to the configuration improve the results.

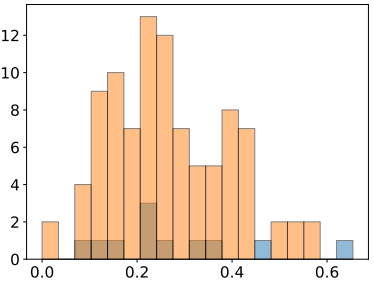
Table 4.3 Table showing the average error for all systems tested.

System	average error	lowest error	highest error	normalised variance
E2	0.26559	0.00514	0.57861	0.05718
E6	0.15292	0.08143	0.26649	0.02049
E7	0.28357	0.09930	0.65441	0.08276
E8	0.06050	0.00646	0.21231	0.05294
E9	0.01459	0.00045	0.04998	0.01364
E10	0.02065	0.00532	0.06311	0.01356

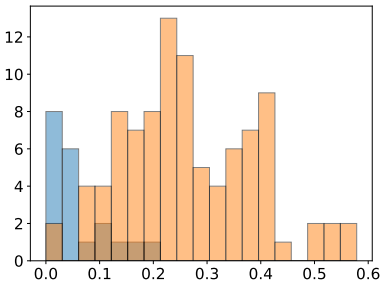
From table 4.3 and Figure 4.6 we can see that having $a(t)$ decaying slower improves the performance, while randomizing the initial value for simulations x_0 has a minor negative impact on the performance 0.2656 versus 0.2836. The most interesting results are from the third alteration, where the noise used for exploration ξ increased, causing significant improvements 0.2656 to 0.0605 putting it very similar to the results from the first system.



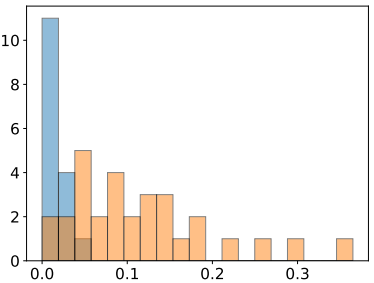
(a) E2 (orange) - E6 function (blue)



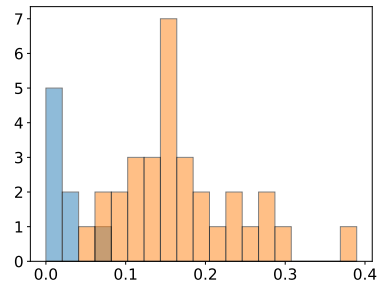
(b) E2 (orange) - E7 (blue)



(c) E2 (orange) - E8 (blue)



(d) E3 (orange) - E9 (blue)



(e) E4 (orange) - E10 (blue)

Figure 4.6 Showcasing a couple of alterations to the system configuration for system 2, and the results it brings.

Stochastic and quasi stochastic noise

Table 4.4 Table displaying values for default configuration between stochastic and quasi-stochastic noise

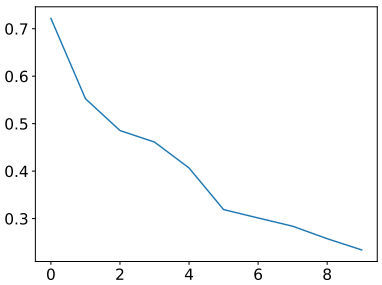
System	average error	lowest error	highest error	normalised variance
E1	0.0200616439	0.0000364981	0.0837410614	0.0256778710
E11	0.1458192853	0.0026007255	0.3962157965	0.0583222285

From table 4.5 we can see that using stochastic noise generated worse results, however, it is hard to compare the noise signals relative to each other; we will look at the normalized variance for the two experiments. Comparing the variance for the system with quasi-stochastic noise 0.02568 to the stochastic noise variance of 0.05832 we see that quasi-stochastic noise does contribute towards variance reduction which is claimed in the literature.

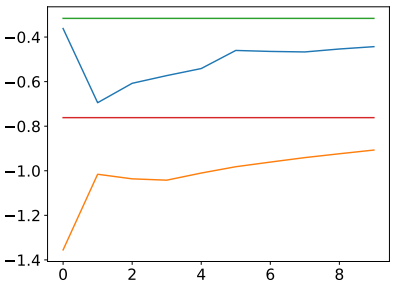
Policy update at partial completion

Table 4.5 Table displaying results from the original experiment E1 and the changed experiment E12.

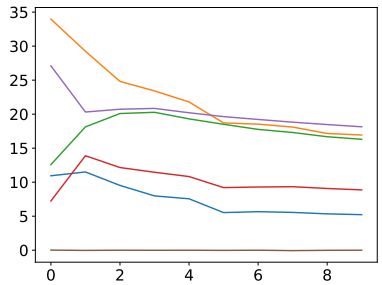
System	average error	lowest error	highest error	normalised variance
E1	0.0200616439	0.0000364981	0.0837410614	0.0256778710
E12	0.0199417041	0.0008652482	0.0685424134	0.0202465188



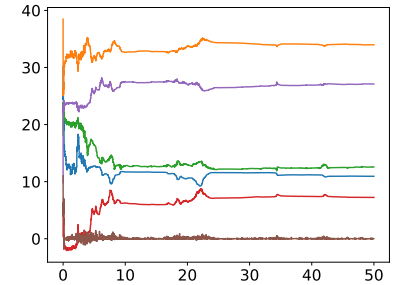
(a) Example of $\frac{||K^* - K_N||}{||K^*||}$ developing across all 10 iterations.



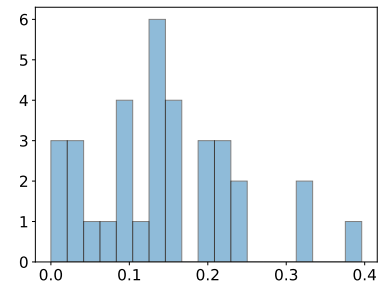
(b) Example of K_N developing across all 10 iterations.



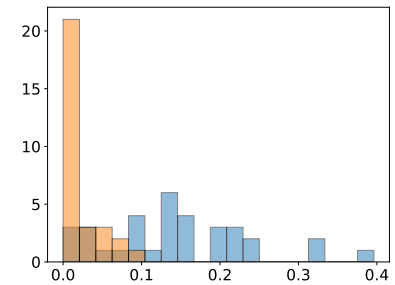
(c) Example of θ developing across all 10 iterations.



(d) Example of θ developing for the first iteration of the algorithm.

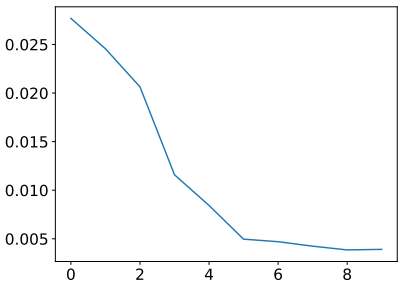


(e) An histogram showcasing the values $\frac{||K^* - K_N||}{||K^*||}$ for all simulations

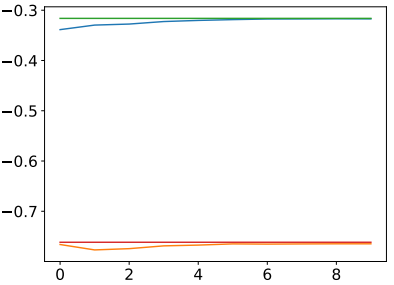


(f) K11 (blue) - K1 (orange)

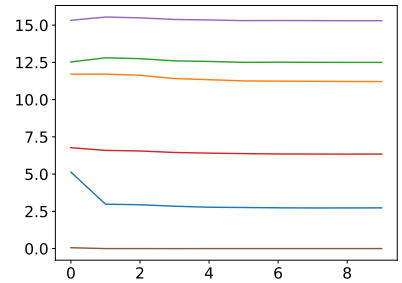
Figure 4.7 Figures showcasing some interesting information for multiple runs of experiment E11 and a comparison between it and E1. The first two rows of subfigures display how an average simulation looks. The last row displays the error for all simulations of this system.



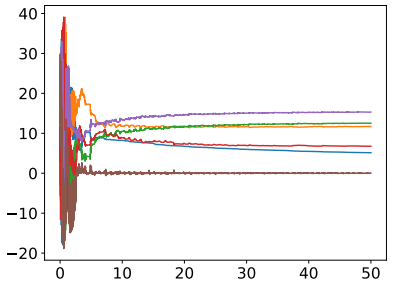
(a) Example of $\frac{||K^* - K_N||}{||K^*||}$ developing across all 10 iterations.



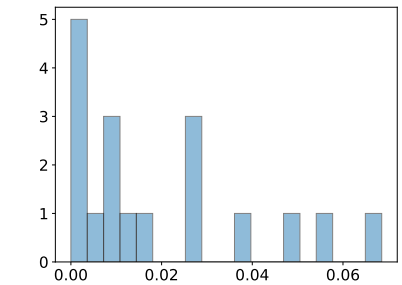
(b) Example of K_N developing across all 10 iterations.



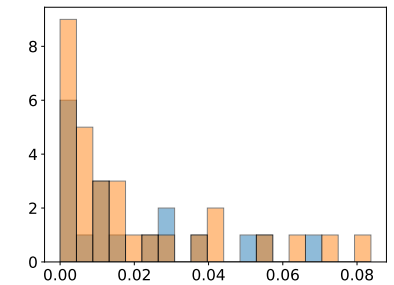
(c) Example of θ developing across all 10 iterations.



(d) Example of θ developing for the first iteration of the algorithm.



(e) An histogram showcasing the values $\frac{||K^* - K_N||}{||K^*||}$ for all simulations



(f) K12 (blue) - K1 (orange)

Figure 4.8 Figures showcasing some interesting information for multiple runs of experiment E12 and a comparison between it and E1. The first two rows of subfigures display how an average simulation looks. The last row displays the error for all simulations of this system.

5

Discussion

The goal of the policy iteration algorithm with quasi-stochastic noise was to find a method that could minimize the cost function J, with certain restrictions. These restrictions where the system constraints $f(x(t), u(t))$, the value function J is convex and twice continuously differentiable, and the cost function $c(x(t), u(t))$ was selected or at least known. The algorithm should also have continuous improvement and create a better policy with each iteration. With the design goals of the algorithm recapitulated, we can examine the results obtained in the previous section; we can now dispute if the algorithm worked.

It should be immediately noted that the first system S1 (4.2) was used during programming to asses whether or not the algorithm worked; parameters were selected to give an optimal result for this specific system. These parameters were kept and used for the other four systems (including the nonlinear one), to keep a certain quantity of parameters equal between the experiments. This was to highlight situations one might find when initial testing of systems. In fact, we cannot compare the system but how the configuration tuned for S1 (4.2) works for systems with increasing complexity.
If we inspect the initial four linear systems S1 (4.2), S2 (4.3), S3 (4.4), S4 (4.5); we see a bit of varying results. The results associated with the control system S1 (4.2) were significantly better with an average error of 0.0201 for experiment E1, compared to the second system S2 (4.3) 0.2656 as a result of E2. The algorithm performed similarly between the third S3 (4.4) and fourth system S4 (4.5) with an average result of 0.1170 and 0.1723 for experiments E3 and E4, respectively, with a lower error than S2. Continuing with S1 where the resulting control policy was a scalar, the results obtained were convergent in all cases and demonstrate that the algorithm works; the algorithm produces better policies with each iteration.
Systems S3 and S4, expanding the state vector x for both systems and the control policy u for system S4. Inspecting the plots and results, one can conclude that the algorithm does not always converge within the allotted iterations of 10. Still looking at the results, we can see that it computes an improved policy with each iteration, and given more iterations, it should converge to a near-optimal control

policy. Expanding experiments E3 and E4 to allow more iterations E9 for system S3 and E10 for system S4, we get an improved result of 0.0146, and 0.02065, respectively similar to the results from E1; convergence at around 25 iterations, confirming previous speculation.

The nonlinear system S5 (4.6) generated a similar error to S2 with an error of 0.2692 for experiment E5. However, inspecting the graphs and plots from the fifth nonlinear system, we can see that similarly to the third and fourth systems it has still not converged to an answer, and more iterations are presumably needed.

With an average error of 0.2656, it did not perform the worst overall, however, it has converged to a value that is fairly far away from the optimum; unlike the third, fourth, and fifth systems, additional iterations will presumably not improve upon the results. Does this mean that the algorithm ”fails” for some combinations of policy and system constraints? Not quite as we will see in the next section.

Configuration changes

The results are promising; however, in many cases, they do not quite reach the level of convergence of the first system and associated experiments. A subsection of the results was dedicated to see if the same results could be obtained for different systems with a configuration change. From the results obtained by increasing the noise for the second system E8, we obtained an improvement of the average error from 0.2656 E2 to 0.0605 E8. This shows that a configuration made for the specific problem is significantly better than a general one, as one would expect with optimization algorithms.

Regular update of the policy

The change was to update the policy K_N regularly during the approximation of the function, rather than only after all the time had passed. This change brought a neutral change and the average error became 0.01994 for experiment K12 an improvement over the original experiment E1 with 0.02001.

Stochastic and quasi stochastic noise

Quasi-stochastic noise can be used by Monte Carlo methods as a means of variance reduction. One of the experiments sets out to verify this claim. The results are compared on normalized variance as it can be hard to compare stochastic and quasi-stochastic noise and find if they are equivalent. The variance obtained from quasi-stochastic noise was 0.0257 and from stochastic Gaussian noise 0.05832 for the first control system, the quasi-stochastic has approximately half the variance. Quasi-stochastic noise did contribute towards variance reduction.

5.1 Does it solve the problem formulation?

The previous sections discuss how well the algorithm achieved the design goals highlighted in the method section, the method section is largely the same as the problem formulation. However, it makes some further assumptions upon the formulations; an evaluation of how realistic and how often these assumptions apply is required to make a final statement.

The additional constraint where:

- The cost function $c(x(t), u(t))$ is selected, or at least known.
- The value function J is convex and twice continuously differentiable.

The first constraint is fairly mild since the designer generally requires the system to behave within the physical limits or to behave sufficiently fast. Designing a cost function is one way to achieve these goals. The known cost function is used to compute $b(t)$ which in turn requires $d(x(t), u(t))$ and its derivative, these are used in the steepest descent algorithm and are essential in the algorithm. An alteration to the algorithm where the cost function is unknown and instead is sampled should be possible, further decreasing the constraints.

The constraints of convexity of J are common for optimization algorithms in general, as one can only guarantee global minima if the function is convex, otherwise, they find a local minima that might or might not be the global minima or very computationally intractable. Additionally, if the cost function is up to selection, one can be chosen such that the integral of the cost function over time is convex and twice continuously differentiable. Furthermore, J being twice continuously differentiable is also not a major constraint. J is an integral of the cost function over time, and the time derivative of the integral of the cost function over time is the cost function itself. The only time the cost function lacks a derivative in time is if it is independent of time, such systems are of little concern.

6

Conclusions

We have investigated the problem of finding an optimal control policy for continuous time and space by utilizing reinforcement learning. The problem formulation was described as the minimization of an integral: The integral is the cost function over an infinite time horizon, referenced as the cost-to-go in much of the literature.

To minimize the cost-to-go function we gathered a selection of reinforcement theories, methods, and ideas; e.g. Monte Carlo simulation and temporal difference learning. In conjunction with previous algorithms and optimization methods for optimal control. These ideas were used to create an off-policy policy iteration algorithm, utilizing quasi-stochastic noise for variance reduction. Furthermore, a short list of potential positive changes were brought forth including different ways of function approximation from reinforcement learning.

The quasi-stochastic policy iteration method was used to solve five different control systems, one of which was nonlinear. The results were positive but varying, the policy was improved with each iteration. The next step was an attempt to improve the results from the worst-performing systems and experiments; by changing the algorithm configuration. These changes showed that the algorithm could compute an optimal control policy for multiple cases, given a good configuration.

Reinforcement learning techniques are fitting for finding optimal control policies in large part due to the numerous methods of function approximation, capable of highlighting the existing but unknown state constraints. Either directly or through something like the Q-function. It also brings methods and solutions for when the state constraints are known, that are different from the ones found in analytical solutions; which can be hard to solve.

6.1 Research limitations and further research

Further research The major remaining questions for me are, how scaleable the algorithm is and how it compares to other optimal control solvers. The algorithm finds optimal controllers in multiple different systems however I do see it potentially outperformed in an LQ setting by more established methods. However, the algorithm also provides solutions to nonlinear examples and needs comparison to other nonlinear solvers.

A brief discussion during the method brought up using neural networks for computing an optimal θ^* for a given policy ϕ instead of utilizing the value approximation proposed in the method section. Such a method could allow for larger systems due to the optimisation of neural networks can be run in parallel and scale linearly with increased hardware. A concern I have is how scaleable the algorithm is, as it is fairly sequential in computation, and running them on devices such as GPUs and AI accelerators could allow the algorithm to be used on large systems.

References

Asmussen, S. and P. W. Glynn (2007). "Stochastic simulation: algorithms and analysis". In: URL: https://api.semanticscholar.org/CorpusID:60181893.

Bernstein, A., Y. Chen, M. Colombino, E. Dall'Anese, P. Mehta, and S. Meyn (2019). "Quasi-stochastic approximation and off-policy reinforcement learning". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 5244–5251. DOI: 10.1109/CDC40024.2019.9029247.

Cybenko, G. V. (1989). "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals and Systems* **2**, pp. 303–314. URL: https://api.semanticscholar.org/CorpusID:3958369.

Doya, K. (2000). "Reinforcement learning in continuous time and space". *Neural Computation* **12**:1, pp. 219–245. DOI: 10.1162/089976600300015961.

Laezza, R. (2021). "Robot learning for manipulation of deformable linear objects". *Licentiate of engineering; Chalmers*. URL: https://research.chalmers.se/publication/526364/file/526364_Fulltext.pdf.

Luo, B., D. Liu, and T. Huang (2014). *Q-learning for optimal control of continuous-time systems*. arXiv: 1410.2954 [cs.SY].

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA. ISBN: 0262039249.

Vamvoudakis, K. G. (2017a). "Q-learning for continuous-time linear systems: a model-free infinite horizon optimal control approach". *Systems & Control Letters* **100**, pp. 14–20. ISSN: 0167-6911. DOI: https://doi.org/10.1016/j.sysconle.2016.12.003. URL: https://www.sciencedirect.com/science/article/pii/S0167691116301967.

Vamvoudakis, K. G. (2017b). "Q-learning for continuous-time linear systems: a model-free infinite horizon optimal control approach". *Systems & Control Letters* **100**, pp. 14–20. DOI: 10.1016/j.sysconle.2016.12.003.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
September 2024

Document Number
TFRT-6258

Author(s)
Anton Forsell

Supervisor
Yiannis Karayiannidis, Dept. of Automatic Control, Lund University, Sweden
Anders Rantzer, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Reinforcement learning for optimal control problems in continuous time and space

Abstract

This thesis considers the problem of finding an optimal control policy for control problems in continuous time and space, utilising reinforcement learning techniques. Finding analytical solutions to optimal control problems is often intractable due to involved constraints, cost-function objectives, and uncertainty. The focal point will be on reinforcement learning RL techniques to find approximate replacements for analytical solutions; or as a tool to create initial control policies, enabling findings and potential problems early in the design process. The thesis will put focus on model-free stochastic RL methods and how introduction of quasi-stochastic noise can be used as a tool for variance reduction. Specifically, we study an off-policy temporal difference TD learning method to find optimal control policies based on the quasi-stochastic approximation QSA method. In this approach, data gathering is done by sampling the system with an entirely unrelated policy to the policy to optimize.

The results show that the algorithm produces policies of increasing quality with each iteration, in many cases reaching optimality. The algorithm was tested on linear systems of varying complexity, although still relatively small, and a basic nonlinear system.We discussed how applicable the method is to optimal control problems and concluded that it is applicable to a majority of systems due to having neither harder nor softer requirements than most optimal control optimisers.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-38

Recipient's notes

Security classification

http://www.control.lth.se/publications/