Bridging the gap between QP-based and MPC-based Reinforcement Learning

Shambhuraj Sawant * **Sebastien Gros** *

* *Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway (e-mail: shambhuraj.sawant@ntnu.no, sebastien.gros@ntnu.no).*

Abstract: Reinforcement learning methods typically use Deep Neural Networks to approximate the value functions and policies underlying a Markov Decision Process. Unfortunately, DNN-based RL suffers from a lack of explainability of the resulting policy. In this paper, we instead approximate the policy and value functions using an optimization problem, taking the form of Quadratic Programs (QPs). We propose simple tools to promote structures in the QP, pushing it to resemble a linear MPC scheme. A generic unstructured QP offers high flexibility for learning, while a QP having the structure of an MPC scheme promotes the explainability of the resulting policy, additionally provides ways for its analysis. The tools we propose allow for continuously adjusting the trade-off between the former and the latter during learning. We illustrate the workings of our proposed method with the resulting structure using a point-mass task.

Keywords: Quadratic Programming, Reinforcement Learning, Model Predictive Control

1. INTRODUCTION

Reinforcement Learning (RL) focuses on teaching an agent how to act in a given environment by rewarding desired behaviours and punishing negative behaviours. It is a powerful tool to find solutions to Markov Decision Processes (MDPs) without depending on a priori knowledge of the underlying system dynamics. Most RL methods depend purely on observed state transitions and stage cost realizations, to improve the current policy. On the other hand, model-based optimal control methods attempt to solve the same task using a priori knowledge of system dynamics, and, give performance and stability guarantees through rigorous analysis. Model Predictive Control (MPC) is one such well-known method for the optimal control of complex dynamical systems. Many approaches have been proposed recently that combine optimal control methods with model-based RL in various ways (Kamthe and Deisenroth, 2018; Pinneri et al., 2020, 2021).

Recently, RL has gained popularity due to striking accomplishments ranging from the success in Atari games (Mnih et al., 2013) and in robotics (Heess et al., 2017), to mastering the game of *Go* (Silver et al., 2016). In most RL methods, the optimal control policy required for solving the task at hand is learned either directly or indirectly. Indirect methods of finding the policy rely on learning an approximation of the optimal value function underlying the MDP, typically based on Temporal Difference methods (Kober et al., 2013). Direct RL methods seek to learn the policy directly with either stochastic or deterministic policy gradient methods (Sutton et al., 2000; Silver et al., 2014). However, such direct methods typically require finding an approximation of the value functions to compute the updates of the policy parameters. In both cases, as the value function is not known a priori, it is approximated with a generic function approximator, typically a deep neural network (DNN). DNNs are additionally often used to approximate the policy. Unfortunately, the closed-loop behaviour of a DNN-based policy can be difficult to explain and formally analyze.

Recently, (Gros and Zanon, 2019) proposed an MPC-based function approximator for generic MDPs. (Gros and Zanon, 2019, Theorem 1) states that, under some conditions, the optimal policy π_* and associated value functions can be generated using a single MPC scheme, even if based on an inaccurate model of the dynamics, provided that adequate modifications of the MPC stage cost and constraints are carried out. An MPC-based policy and value functions approximation offer a high explainability about the policy behaviour and is equipped with a broad set of theoretical tools for formal verification of the policy in terms of safety and stability. Furthermore, RL methods can be used to learn the adequate cost and constraints modifications in the MPC-based policy. Hence, the MPC parameters in such an approach are learned from data similar to how DNN activation weights are learned in a typical RL pipeline. Because they are convex and can be solved in very short computational times, linear MPC schemes are arguably the most popular in control. In terms of optimization, linear MPC schemes take the form of a Quadratic Program (QP). The use of RL in accelerating QPs has been investigated in (Choi et al., 2020; Ichnowski et al., 2021). In this work, we use linear MPC schemes and generic QPs to carry approximations of the policy and value functions. We believe that the piece-wise quadratic nature of QPs would provide sufficient richness of features to approximate such functions for continuous control tasks.

A generic QP and a linear MPC scheme chiefly differ in the structure of their cost and constraints. Indeed, the constraint and cost matrices in a generic QP do not have any specific structure. In contrast, the matrix underlying

the cost function of a linear MPC scheme is block-diagonal, and the matrix underlying its equality constraint has a specific banded structure matching the associated system dynamics. Due to its structure, the QP underlying a linear MPC scheme has less freedom, and therefore, less flexibility to perform in the learning task. A generic, unstructured QP has, in contrast, more flexibility and can arguably perform better as an approximation of the policy and value functions.

In this paper, we explore the difference between linear MPC and generic QP formulations, and propose tools to smoothly transition between them by more or less aggressively promoting MPC-like structures during RL-based learning. In our formulation, constraints are fully parameterized and freed up for learning without assigning any specific structure. However, we introduce simple penalties in the learning step for promoting the emergence of an MPC-like structure in the QP. We evaluate how such penalties can help in transitioning from a generic QP to structured MPC-like constraints on a point-mass task.

The paper is structured as follows. Section 2 gives an overview of the MPC-based function approximators from (Gros and Zanon, 2019) along with Q-learning method and Quadratic Programming. Section 3 details the proposed QP-based function approximator with the proposed heuristic penalties. Section 4 discusses the used experimental setup for a point-mass task and obtained results, followed by conclusion in section 5.

2. METHODS

In this section, we summarize the linear MPC scheme and QP formulation used for approximating value functions in an MDP and apply Q-learning method for updating parameters in these formulations.

2.1 MPC-based function approximation for MDPs

A Markov Decision Process (MDP) is a mathematical framework used for modelling decision-making in a discrete-time stochastic control process. An MDP is characterized by a tuple $(\mathcal{S}, \mathcal{A}, P_a, L, \gamma)$ where $\mathcal{S}, \mathcal{A}$ represent state and action spaces respectively, $P_a(s, s_+) = P[s_+|s, a]$ is the probability that action a and state s leads to state s_+, $L(s, a) = l$ describes the stage cost l for taking action a in state s, and γ is a discount factor. Additionally, a transition is defined to be a tuple (s, a, s_+, l). Solving an MDP consists of finding an optimal *policy*: a function π that maps a state s into an action a, i.e. $\pi : \mathcal{S} \to \mathcal{A}$, for minimizing the cumulative reward J given as:

$$J(\pi) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k L(s_k, a_k) \,\middle|\, a_k = \pi(s_k)\right]$$

where the stage cost $L(s, a)$ reads as:

$$L(s, a) = l(s, a) + \mathcal{I}_\infty(h(s, a)) + \mathcal{I}_\infty(g(a)) \tag{1}$$

In (1), function l captures cost associated to different state-action pairs, while the constraints $h(s, a)$ and $g(a)$ capture the undesirable state-action pairs. The indicator function $\mathcal{I}_\infty(x)$ penalizes constraint violations:

$$\mathcal{I}_\infty(x) = \begin{cases} \infty & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The underlying MDP M with the associated stage cost L and discount factor $0 < \gamma < 1$ is assumed to yield a well-posed problem, i.e. the policy π exists and is well defined over some regions in $\mathcal{S}$ and the associated value functions are well posed and finite over some regions in $\mathcal{S} \times \mathcal{A}$.

We label $\pi_*(s)$ the optimal policy, $Q_*(s, a)$ the action-value function, and $V_*(s)$ the value function. The policy and value functions are solution of the Bellman equations (Sutton and Barto, 2018):

$$Q_*(s, a) = L(s, a) + \gamma \mathbb{E}[V_*(s_+|s, a)]$$
$$V_*(s) = Q_*(s, \pi_*(s)) = \min_a Q_*(s, a)$$

We briefly recall next the theory behind MPC-based approximations of these functions.

We consider a (possibly deterministic) model of system dynamics as $P[\hat{s}_+|s, a]$. (Gros and Zanon, 2019) consider a modified stage cost:

$$\hat{L}(s, a) = \begin{cases} Q_*(s, a) - \gamma V^+(s, a) & \text{if } V^+(s, a) < \infty \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

where $V^+(s, a) = \mathbb{E}[V_*(\hat{s}_+)|s, a]$ uses the expected value stemming from the model of system dynamics. (Gros and Zanon, 2019, Theorem 1) states that, under some conditions, an MPC scheme using the model $P[\hat{s}_+|s, a]$ and the modified stage cost (2) yields the optimal policy π_* for the true MDP, together with the associated optimal value functions. Simply stated, an optimal policy can be generated using an MPC scheme based on simplified or inaccurate model dynamics provided that the MPC stage cost is adequately modified. In particular, an MPC scheme based on a deterministic model can yield the optimal policy and value functions of a stochastic MDP.

However, computing $\hat{L}$ can be difficult and requires the knowledge of the true system dynamics. Fortunately, (Gros and Zanon, 2019) showed that RL techniques are well suited to build this cost modification from data.

We consider the following parameterization of a linear MPC scheme for approximating the value function V of a generic MDP:

$$\begin{aligned} V_\theta(s) = \min_{u,x,\sigma} \quad & \sum_{k=0}^{N-1} \gamma^k ([x_k; u_k]^T W_\theta [x_k; u_k] + w^T \sigma_k) \\ & + \gamma^N (x_N W_\theta^f x_N + w_f^T \sigma_N) && \text{(3a)} \\ \text{s.t.} \quad & x_{k+1} = A_\theta x_k + B_\theta u_k,\ x_0 = s && \text{(3b)} \\ & G u_k \leq 0 && \text{(3c)} \\ & D_\theta [x_k; u_k]^T \leq \sigma_k,\ D_\theta^f x_N \leq \sigma_N && \text{(3d)} \\ & \sigma_k \geq 0,\ \sigma_N \geq 0 && \text{(3e)} \end{aligned}$$

(3) holds a stage cost parameterization W_θ, a constraint parameterization D_θ, a terminal cost parameterization W_θ^f, a terminal constraint parameterization D_θ^f and a model parameterization A_θ, B_θ. A point to note in (1) and (3) is that, the constraints are separated into pure action constraints (3c) and mixed constraints (3d). Although the pure action constraints are arguably fixed, the mixed constraints need to be parameterized and learned to capture the domain in which $\hat{L}$ in (1) is finite. Additionally, an l_1 relaxation of the mixed constraints (3d) is carried out using slack variables σ_k to avoid infinite penalties in case of constraint violations.

Similarly, the associated action-value function Q approximation is given as:

$$Q_\theta(s,a) = \min_{u,x,\sigma} \quad \text{(3a)} \tag{4a}$$
$$\text{s.t.} \quad \text{(3b)} - \text{(3e)} \tag{4b}$$
$$u_0 = a \tag{4c}$$

For these value function approximations, the Bellman equations still hold:

$$\pi_\theta(s) = \arg\min_a Q_\theta(s,a), \quad V_\theta(s) = \min_a Q_\theta(s,a)$$

where $\pi_\theta(s)$ is the optimal policy for (3) and (4).

2.2 Quadratic Programming

Quadratic Programming (QP) is a process of solving mathematical optimization problems involving a quadratic cost function subject to linear constraints. A generic quadratic program with n variables takes the form:

$$\min_z \quad \frac{1}{2} z^T H z + q^T z + c \tag{5a}$$
$$\text{s.t.} \quad Cz = b \tag{5b}$$
$$lb \leq Gz \leq ub \tag{5c}$$

where $z \in \mathbb{R}^n$ is the optimization variable, H is an $n \times n$ symmetric and often positive semi-definite matrix that defines the quadratic cost, $q \in \mathbb{R}^n$ defines the linear cost, c is a constant, C is an $m \times n$ matrix that defines m linear equality constraints, and, G is an $l \times n$ matrix defining l linear inequality constraints with lb, ub as their lower and upper bounds, respectively.

QP as a function approximator for MDPs

We formulate next a generic QP for approximating the policy and value functions underlying an MDP. We consider the optimization variables as:

$$z = [\ x_0;\ u_0;\ x_1;\ \dots\ x_N]^T \tag{6}$$

where $x_{0,\dots,N}, u_{0,\dots,N-1}$ have the dimension of the state and action spaces of the MDP, respectively.

We then introduce parameterizations of the matrices and vectors in (5), labelled as $H_\theta, q_\theta, c_\theta, C_\theta, G_\theta$. We refer to C_θ as the constraint matrix in later discussions. Additionally, l_1 relaxation is carried out for inequality constraints (5c) for previously specified reasons. The resulting QP provides a value function approximation V_θ as:

$$V_\theta(s) = \min_{z,\sigma} \quad \frac{1}{2} z^T H_\theta z + q_\theta^T z + c_\theta + w^T \sigma \tag{7a}$$
$$\text{s.t.} \quad C_\theta z = b \tag{7b}$$
$$lb - \sigma \leq G_\theta z \leq ub + \sigma \tag{7c}$$
$$\sigma \geq 0 \tag{7d}$$
$$x_0 = s \tag{7e}$$

The action-value function approximation Q_θ is:

$$Q_\theta(s,a) = \min_{z,\sigma} \quad \text{(7a)} \tag{8a}$$
$$\text{s.t.} \quad \text{(7b)} - \text{(7e)} \tag{8b}$$
$$u_0 = a \tag{8c}$$

An observation regarding (7) is that the N state-action pairs in the optimization variable z do not necessarily form a Markov chain or respect any system dynamics. These pairs simply satisfy the QP constraints and help in approximating the value functions. Hence, in general, (7) and (8) do not necessarily correspond to an MPC scheme and offer no explainability as such. However, due to the high number of degrees of freedom in selecting the entries of the QP matrices in (7) and (8), such a QP offers a high flexibility in approximating the value functions.

2.3 Q-learning for QP

Q-learning method offers a simple yet powerful tool to adjust the parameters in MPC (4) and a QP (8) such that the resulting Q_θ are close to the true optimal action-value function $Q_\star$. In basic Q-learning, the action-value function parameters θ are updated to minimize the differences in approximation for instantaneous transitions:

$$\theta = \arg\min_\theta \mathbb{E}\big[Q_*(s) - Q_\theta(s)\big]$$

A version of Q-learning with batch updates (sampled from stored transition dataset, i.e. replay buffer $\mathcal{D}$) reads as:

$$\delta = L(s,a) + \gamma V_\theta(s_+) - Q_\theta(s,a) \tag{9}$$
$$\theta \leftarrow \theta + \alpha \mathbb{E}\big[\delta \nabla_\theta Q_\theta(s,a)\big] \tag{10}$$

where δ is the temporal difference error for the sampled transition, $(s,a) \in \mathcal{B}$, a batch of transitions sampled from replay buffer $\mathcal{D}$, and α is the learning rate.

The batch update version of Q-learning can be applied to learn the parametric Q functions in (4) and (8). The gradient of Q_θ in (8) can be obtained at a very low computational cost using the associated Lagrange function:

$$\begin{aligned}\mathcal{L}_\theta(s,y) = \frac{1}{2} z^T H_\theta z + q_\theta^T z + c_\theta + w^T \sigma + \chi^T (C_\theta z - b) \\ + \upsilon^T (G_\theta z - lb + \sigma) + \eta^T (ub + \sigma - G_\theta z) \\ + \mu^T \sigma + \xi^T (x_0 - s) + \zeta^T (u_0 - a)\end{aligned} \tag{11}$$

where $\chi, \upsilon, \eta, \mu, \xi, \zeta$ are the multipliers associated with constraints (8b), i.e. (7b)-(7e), and (8c), respectively, and, $y = (x, u, \chi, \upsilon, \eta, \mu, \xi, \zeta)$ compiles the associated primal-dual variables. From (Büskens and Maurer, 2001), the gradient of Q_θ is:

$$\nabla_\theta Q_\theta(s,a) = \nabla_\theta \mathcal{L}(s, y^*) \tag{12}$$

wherein y^* is the primal-dual solution of (8).

We assume H_θ in (8) to be a positive semi-definite matrix, for ensuring a convex QP. Hence, a semi-definite program (SDP) is used for updating θ in (8):

$$\Delta\theta_s = \arg\min_{\Delta\theta} \quad \Delta\theta^2 - \alpha \mathbb{E}\big[\delta \nabla_\theta Q_\theta\big] \Delta\theta \tag{13a}$$
$$\text{s.t.} \quad H_{\theta + \Delta\theta} \geq 0 \tag{13b}$$

In case, H_θ is positive semi-definite, (13) yields the same update as (10). However, if H_θ does not remain positive semi-definite, SDP carries out a constrained optimization step for updating θ. Additionally, it is beneficial to use different learning rates α for updating θ in the optimization cost and the constraints.

In this paper, we make use of Q-learning method for the sake of simplicity. However, most typical RL methods can also be used to learn the parameters of (4) and (8).

3. QP-BASED RL WITH PENALTIES

In this section, we discuss the proposed formulation to smoothly transition between value function approximations using QPs and MPC schemes by promoting dynamic system-like constraints during learning.

3.1 Proposed formulation

We consider the QP-based approximation of Q function given in (8) wherein the optimization variable is $z = [x_0;\ u_0;\ x_1;\ \dots\ x_N]^T$. Consider the optimization cost as:

$$
\begin{aligned}
J &= \gamma^N \Big(\frac{1}{2} x_N^T H_\theta^f x_N + (q^f)_\theta^T x_N\Big) + c_\theta \\
&\quad + \sum_{k=0}^{N-1} \gamma^k \Big(\frac{1}{2}[x_k; u_k]^T (H_k)_\theta [x_k; u_k] + (q_k)_\theta^T [x_k; u_k]\Big) \\
&= \frac{1}{2} z^T H_\theta z + q_\theta^T z + c_\theta \qquad (14)
\end{aligned}
$$

Hence, H_θ from (7) becomes a block-diagonal matrix, introducing a structure to the QP objective similar to that of an MPC scheme (3). In order for (8) to resemble an MPC-based approximation of the action-value function, the linear constraints in (8b) need to take the form corresponding to a linear model of the dynamics. More specifically, (7b) should take the form of model constraints (3b), which can be combined to be written as:

$$
\underbrace{\begin{bmatrix} A & B & -I & 0 & \dots & \dots & 0 \\ 0 & 0 & A & B & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & A & B & -I \end{bmatrix}}_{C_\theta} \underbrace{\begin{bmatrix} x_0 \\ u_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}}_{z} = \underbrace{0}_{b} \qquad (15)
$$

The linear dynamical constraints in an MPC scheme result in a banded constraint matrix having a very specific pattern. In order to make sense of the entries in z, C_θ should emerge with a similar structure after the learning routine and we consider $b = 0$. With these considerations, the proposed value function $V_\theta(s)$ is given as:

$$
\begin{aligned}
V_\theta(s) = \min_{z,\sigma} \quad & (7a) && (16a)\\
\text{s.t.} \quad & C_\theta z = 0 && (16b)\\
& (7c) - (7e) && (16c)
\end{aligned}
$$

Similarly, the proposed action-value function $Q_\theta(s, a)$ is:

$$
\begin{aligned}
Q_\theta(s,a) = \min_{z,\sigma} \quad & (8a) && (17a)\\
\text{s.t.} \quad & (16b) && (17b)\\
& (7c) - (7e), (8c) && (17c)
\end{aligned}
$$

(16) and (17) closely resemble the QP-based approximations in (7) and (8) with no specific structure for C_θ. We make use of Q-learning for updating θ in (17).

The parameterizations introduced in the QP-based approximation of action-value function in (17), $H_\theta, q_\theta, c_\theta$, C_θ, G_θ, are fully unconfined, i.e. each element is, in principle, free to be updated. However, we would like to see the emergence of MPC-like structure in these matrices and vectors to afford some explainability, without forcing the same, in the interest of flexibility for function approximation. However, in this paper, we focus on promoting resembling structure in the fully parameterized constraint matrix C_θ, and assume a block diagonal structure for the optimization cost and in G. However, we believe that similar simple penalties would promote a structure in the remaining parameterizations. In case of C_θ, we would like to see a structure like (15) emerge out of learning. However, as constraint matrix C_θ is not structured, i.e. it is fully parameterized and free for tuning, the resulting constraint matrix C_θ would be updated to minimize the temporal difference error and would likely be a dense matrix. Hence, we next explain how simple penalties can help in promoting such a structure in C_θ.

3.2 Motivating dynamical system-like constraints

The constraint matrix C_θ is parameterized to give the learning algorithm complete freedom to have maximal flexibility for better approximation. However, the learned constraint matrix C_θ would likely become dense, having no particular structure. To tackle this, we propose a simple penalty in the SDP scheme (13) to promote learning a banded structure like in (15). It introduces a trade-off between performance and a penalty for deviation from the banded constraint matrix in (15).

To achieve a smooth transition between an MPC-based and a QP-based formulation by more or less aggressively promoting the desired structure, the penalty for deviation from the banded structure can be scaled as required. A high value of the scaling constant makes the SDP routine with the penalty stick to the banded structure, while a smaller value provide the RL tool with the freedom to update C_θ. This scaling is achieved using the scaling matrix C_{mask}. We consider all parameters in (8) constitute θ, i.e. θ is a vector representing all parameters learned using RL method.

The proposed SDP scheme with the deviation penalty is:

$$
\begin{aligned}
\Delta\theta_s = \arg\min_{\Delta\theta} \quad & \Delta\theta^2 - \alpha \mathbb{E}[\delta \nabla_\theta Q_\theta] \Delta\theta \\
& + |C_{mask} \odot (C_{\theta+\Delta\theta} - C_0)| && (18a)\\
\text{s.t.} \quad & W_{\theta+\Delta\theta} \geq 0 && (18b)
\end{aligned}
$$

where $(X \odot Y)$ is the Hadamard product, C_{mask} is the scaling matrix for 1-*norm* penalty and C_0 is the banded constraint matrix for the model of state dynamics (A, B):

$$
C_0 = \begin{bmatrix} A & B & -I & 0 & \dots & \dots & 0 \\ 0 & 0 & A & B & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & A & B & -I \end{bmatrix}
$$

$$
C_{mask} = \begin{bmatrix} c_2 & c_2 & c_2 & c_1 & \dots & \dots & c_1 \\ c_3 & c_3 & c_2 & c_2 & \dots & \dots & c_1 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ c_3 & c_3 & \dots & \dots & c_2 & c_2 & c_2 \end{bmatrix}
$$

We considered 1-*norm* penalty so as to force the deviation to zero. Penalizing with 1-*norm* also helps with ease of compute as dense C_θ results in complex constraints over many state action pairs, leading to difficulty in finding a solution to (17) and (18).

Different C_{mask} i.e. different values of $\{c_1, c_2, c_3\}$ result in promoting varied structures in constraint matrix C_θ, and hence, giving different meaning to entries in the optimization variable z. For example, a configuration of $[0, 0, 0]$, i.e. essentially freeing C_θ to be updated as required, results in dense C_θ, but z no longer forms a Markov chain. A configuration of $[1, 1, 1]$ results in C_θ closely resembling C_0, leading to further ability to analyse entries in z. 1-*norm* penalty would force the diagonal (on-band) entries in C_θ close to the model dynamics and the off-diagonal (off-band) entries to zero. C_{mask} can be further configured such that its entries progressively increase away from the diagonal, resulting in C_θ with non-zero entries close to the diagonal and resembling non-Markovian model dynamics with less correlations for distant state-action pairs in time.

3.3 System Identification along with constraint learning

We propose a second penalty to take into account the true system dynamics $P[s_+|s,a]$. The SDP scheme with this penalty takes the form:

$$\Delta\theta_s = \arg\min_{\Delta\theta} \quad \Delta\theta^2 - \alpha\mathbb{E}[\delta\nabla_\theta Q_\theta]\Delta\theta + \sum_{i=0}^{M} \beta\|C_{\theta+\Delta\theta}\tau_i\|^2 \tag{19a}$$

$$\text{s.t.} \quad W_{\theta+\Delta\theta} \geq 0 \tag{19b}$$

where $\tau_i = [s_j, a_j, s_{j+1}, \ldots, s_{j+N}]$ is the i-th sampled sequence of consecutive state transitions from the true system dynamics of length N (sampled from the replay buffer $\mathcal{D}$), and β is a scaling constant. (19) can be interpreted as trading off performance against fitting the constraint matrix to the true system dynamics and hence carrying out system identification (SI).

In Partially Observable MDPs or problems with temporally correlated noise, this SI penalty would help in better fitting the constraint matrix C_θ with the true system dynamics by allowing for helpful changes and limiting adverse ones. The SI penalty should also come in handy for correcting the error in the model of system dynamics.

4. EXPERIMENTS AND RESULTS

We discuss the experiment setup used to illustrate the workings of QP-based function approximator with proposed penalties in this section, and summarize the results.

4.1 Point-Mass

We consider a *Point-Mass* task wherein the objective is to push a point mass to the origin. Its state space $\mathcal{S} \in \mathbb{R}^4$ consists of the location (x, y) and the corresponding velocities $(\dot{x}, \dot{y})$ of point mass, i.e. $s = [x, y, \dot{x}, \dot{y}]^T$ and is bounded by $lb_s = [-2, -2, -10, -10]^T$ and $ub_s = [2, 2, 10, 10]^T$. The action space $\mathcal{A} \in \mathbb{R}^2$ consists of forces applied in x and y directions, i.e. $a = [F_x, F_y]^T$ and is bounded by $lb_a = [-1, -1]^T$ and $ub_a = [1, 1]^T$. The true system dynamics is defined as:

$$s_+ = As + Ba + \nu$$

$$A = \begin{bmatrix} 1 & 0 & 0.1 & 0 \\ 0 & 1 & 0 & 0.1 \\ 0 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 0.9 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

where ν is the noise present in the system. The reward function is set to be $L(s,a) = s^T W s$ with $W = diag(3, 3, 0.25, 0.25)$ and the discount factor γ is 0.9.

To evaluate the proposed penalties and emergence of structure, we consider following cases: a) ν is Gaussian in nature, and b) ν is Brownian Noise. Additionally, we use a corrupted model of state dynamics in the optimization formulation to further showcase the importance of constraint tuning. The corrupted model is obtained by adding a random matrix, whose entries are sample uniformly between $[-0.05, 0.05]$, to the true system dynamics (A, B).

4.2 Experimental Setup

We consider following parameterization for (17): $(H_k)_\theta = diag(\theta_1, \theta_2, \theta_3, \theta_4, 0, 0)$, $(H^f)_\theta = \mathbb{I}(4)$, $(q_k)_\theta = (q^f)_\theta = 0$, and $c_\theta = \theta_5$ with θ_i initialized randomly $\forall i$. $G_\theta = \mathbb{I}(6N+4)$ is used for bounding states and actions over optimization horizon $N = 10$. The constraint matrix C_θ is:

$$C_\theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \ldots \\ \theta_{21} & \theta_{22} & \ldots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

C_θ is a dense $4N \times (6N + 4)$ matrix, initialized to C_0. The parameter vector is $\theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_{11}, \theta_{12}, \ldots]$. For the proposed penalties in (18) and (19), the scaling constants are $c_1 = 1$, $c_2 = 1e-4$, $c_3 = 0$, $\beta = 1e-6$.

4.3 Results

We present the performance results of QP-based RL with proposed penalties in this section. Fig. 1 shows the performance of QPs with different penalties for Gaussian noise case. Constraint learning is important for improving the task performance, as seen in fig. 1, as QP with fixed constraints fails to improve due to model mismatch while other configurations outperform. However, the resulting constraint matrix, shown in fig. 2, show emergence different structures, especially, QP without any penalty and with SI penalty result in denser C_θ while the deviation penalty promotes C_θ to still hold MPC-like structure.

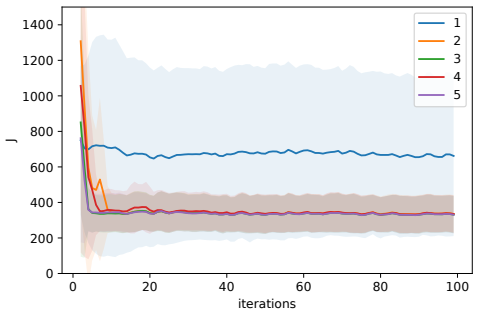

Fig. 1. Cumulative reward J for QP-based RL with different penalties in point-mass task with Gaussian noise. (1: fixed constraints, 2: fully parameterized C_θ, 3: C_θ with deviation penalty (18), 4: C_θ with SI penalty (19), 5: C_θ with combined penalty from (18) and (19))

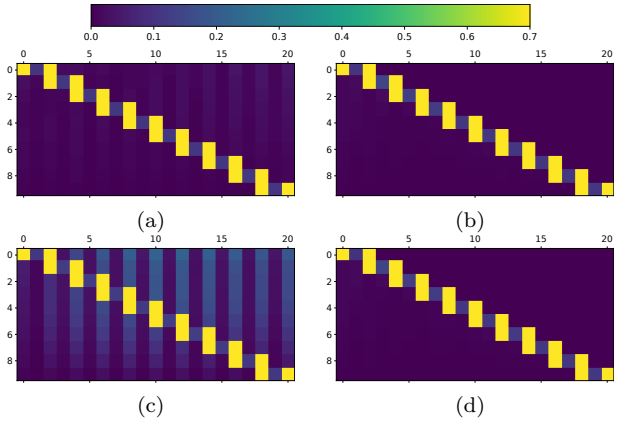

Fig. 2. Learned constraint matrix C_θ in Gaussian noise case for: a) no penalty, b) deviation penalty (18), c) SI penalty (19), d) both penalty from (18) and (19)

Similar results are observed in case of the point-mass task with Brownian noise, as shown in fig. 3. Constraint learning helps improve the performance while QP with fix

constraints fails. From fig. 4, with temporally correlated noise, we observe emergence of significantly denser C_θ for QP without penalties and QP with SI penalty while the deviation penalty pushes the learning routine to hold MPC-like structure while still improving the task performance.

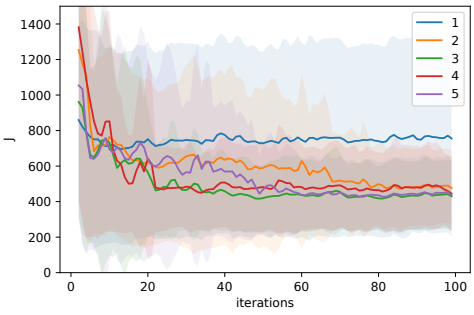

Fig. 3. Cumulative reward J for QP-based RL with different penalties in point-mass task with Brownian noise. (1: fixed constraint, 2: fully parameterized C_θ, 3: C_θ with deviation penalty (18), 4: C_θ with SI penalty (19), 5: C_θ with combined penalty from (18) and (19))

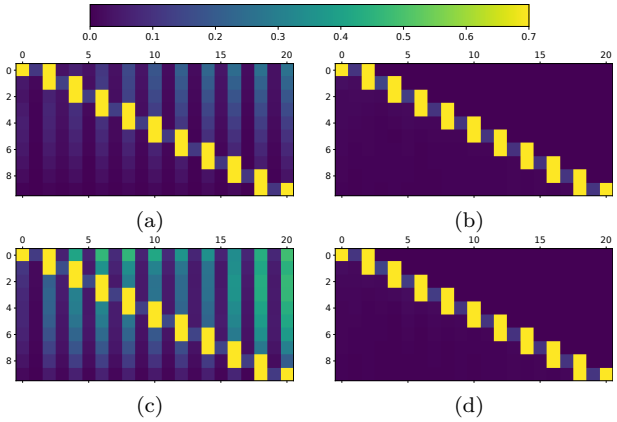

Fig. 4. Learned constraint matrix C_θ in Brownian noise case for: a) no penalty, b) deviation penalty (18), c) SI penalty (19), d) both penalty from (18) and (19)

5. CONCLUSION

In this work, we approximate the policy and value functions of an MDP using linear MPC-based and QP-based function approximators. We propose simple heuristic penalties to smoothly transition between an MPC-based and a QP-based approximation scheme by promoting structure in the optimization formulation. A generic QP-based formulation offers high flexibility to approximate the value functions, however, it lacks explainability, while a QP having the structure of an MPC scheme promotes the explainability of the resulting policy and provides tools for its analysis. With the proposed penalties, we can smoothly transition between QP-based and MPC-based approximations and continuously adjust the trade-off between the former and the latter during learning. These penalties enable maintaining an MPC-like structure while still tuning the constraints. We show that in a point-mass task with stochastic transitions, it is possible to promote structure along with improving performance. Building on this, these tools need to be investigated in complex tasks and partial observable systems.

ACKNOWLEDGEMENTS

We thank the generous funding given by the Research Council of Norway (RCN) through *Safe Reinforcement Learning using MPC* (SARLEM) project.

REFERENCES

Büskens, C. and Maurer, H. (2001). Sensitivity analysis and real-time optimization of parametric nonlinear programming problems. In *Online Optimization of Large Scale Systems*, 3–16. Springer.

Choi, J., Castaneda, F., Tomlin, C.J., and Sreenath, K. (2020). Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions. *arXiv preprint arXiv:2004.07584*.

Gros, S. and Zanon, M. (2019). Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2), 636–648.

Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

Ichnowski, J., Jain, P., Stellato, B., Banjac, G., Luo, M., Borrelli, F., Gonzalez, J.E., Stoica, I., and Goldberg, K. (2021). Accelerating quadratic optimization with reinforcement learning. *Advances in Neural Information Processing Systems*, 34.

Kamthe, S. and Deisenroth, M. (2018). Data-efficient reinforcement learning with probabilistic model predictive control. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, 1701–1710. PMLR.

Kober, J., Bagnell, J.A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11), 1238–1274.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Pinneri, C., Sawant, S., Blaes, S., Achterhold, J., Stueckler, J., Rolinek, M., and Martius, G. (2020). Sample-efficient cross-entropy method for real-time planning. *arXiv preprint arXiv:2008.06389*.

Pinneri, C., Sawant, S., Blaes, S., and Martius, G. (2021). Extracting strong policies for robotics tasks from zero-order trajectory optimizers. In *International Conference on Learning Representations*.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587), 484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, 387–395. PMLR.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R.S., McAllester, D.A., Singh, S.P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.