Policy Derivation Methods for Critic-Only Reinforcement Learning in Continuous Spaces

Eduard Alibekov[a], Jiří Kubalík[a], Robert Babuška[b,a]

[a] *Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic,* {*eduard.alibekov, jiri.kubalik*}*@cvut.cz*
[b] *Cognitive Robotics, Faculty 3mE, Delft University of Technology, Delft, The Netherlands, r.babuska@tudelft.nl*

Abstract

This paper addresses the problem of deriving a policy from the value function in the context of critic-only reinforcement learning (RL) in continuous state and action spaces. With continuous-valued states, RL algorithms have to rely on a numerical approximator to represent the value function. Numerical approximation due to its nature virtually always exhibits artifacts which damage the overall performance of the controlled system. In addition, when continuous-valued action is used, the most common approach is to discretize the action space and exhaustively search for the action that maximizes the right-hand side of the Bellman equation. Such a policy derivation procedure is computationally involved and results in steady-state error due to the lack of continuity. In this work, we propose policy derivation methods which alleviate the above problems, by means of action space refinement, continuous approximation, and post-processing of the V-function by using symbolic regression. The proposed methods are tested on non-linear control problems: 1-DOF and 2-DOF pendulum swing-up problems, and on magnetic manipulation. The results show significantly improved performance in terms of cumulative return and computational complexity.

Keywords: reinforcement learning, continuous actions, multi-variable systems, optimal control, policy derivation, optimization

1. Introduction

Reinforcement Learning (RL) algorithms provide a way to solve dynamic decision-making and control problems [1, 2, 3]. An RL agent interacts with the system to be controlled by measuring its states and applying actions according to a certain policy. After applying an action, the agent receives a scalar reward related to the immediate performance. The goal is to find an optimal policy, which maximizes the cumulative reward.

The available RL algorithms can be broadly classified into critic-only, actor-only, and actor-critic methods [4]. Critic-only methods first find the value function (V-function) and then derive an optimal policy from this value function. In contrast, actor-only methods search directly in the policy space. The two approaches can be combined into actor-critic architectures, where the actor and critic are both represented explicitly and trained simultaneously. Each class can be further divided into model-based and model-free algorithms. In the model-based scenario, a system model is used during learning or policy derivation. The system model may be stochastic or deterministic. In this paper, we consider the critic-only, model-based and deterministic variant of RL in continuous state and action spaces. For the methods developed, it is irrelevant whether the system model is available a priori or learnt on-line.

We address the policy derivation step, assuming that an approximation of the true unknown V-function has already been computed. Policy derivation can be understood as a hill climbing process: at each time step, the agent searches for the control input that leads to a state with a highest value given by the right-hand side (RHS) of the Bellman equation. An advantage of this control law is its inherent stability – the value function is analogous to the control Lyapunov function [5, 6]. However, direct policy derivation from the V-function suffers from several problems:

- **Computational inefficiency**. The most common approach to dealing with a continuous action space is to discretize it into a small number of actions, compute the value of the Bellman equation RHS for all of them,

and select the one that corresponds to the largest value [1, 7]. The number of possible discrete actions grows exponentially with the dimension of the action space and so does the computational complexity of this method.

- **Insufficient smoothness of the V-function**. The above hill-climbing process is adversely affected by the approximate nature of the V-function, which has been observed e.g. in [8]. A typical approximation by means of basis functions exhibits artifacts which can lead to oscillations, as illustrated in the left column of Figure 1. We refer to this problem by the term "insufficient smoothness", without relying on the exact mathematical definition of smoothness.

- **Discrete-valued control input**. The use of discrete actions leads in combination with insufficient smoothness to steady-state errors, as shown in the right column of Figure 1. In the long run, the steady-state error can induce big losses in terms of the overall performance.

The aim of this paper is to alleviate the above problems. We extend our earlier work on policy derivation methods [9]. In addition to the methods originally proposed in this paper we introduce symbolic regression to address the V-function smoothness problem. Additionally, to enable the use of continuous actions, we propose a method based on a computationally efficient optimization technique.

The paper is organized as follows. Related work is discussed in Section 2 and Section 3 reviews the necessary background of reinforcement learning. The proposed policy derivation methods are introduced in Section 4. The results obtained on several benchmark problems are presented in Section 5 and discussed in detail in Section 6. Finally, Section 7 concludes the paper.

2. Related work

The problem of deriving policies for continuous state and action spaces in critic-only methods has not been sufficiently addressed in the literature. The

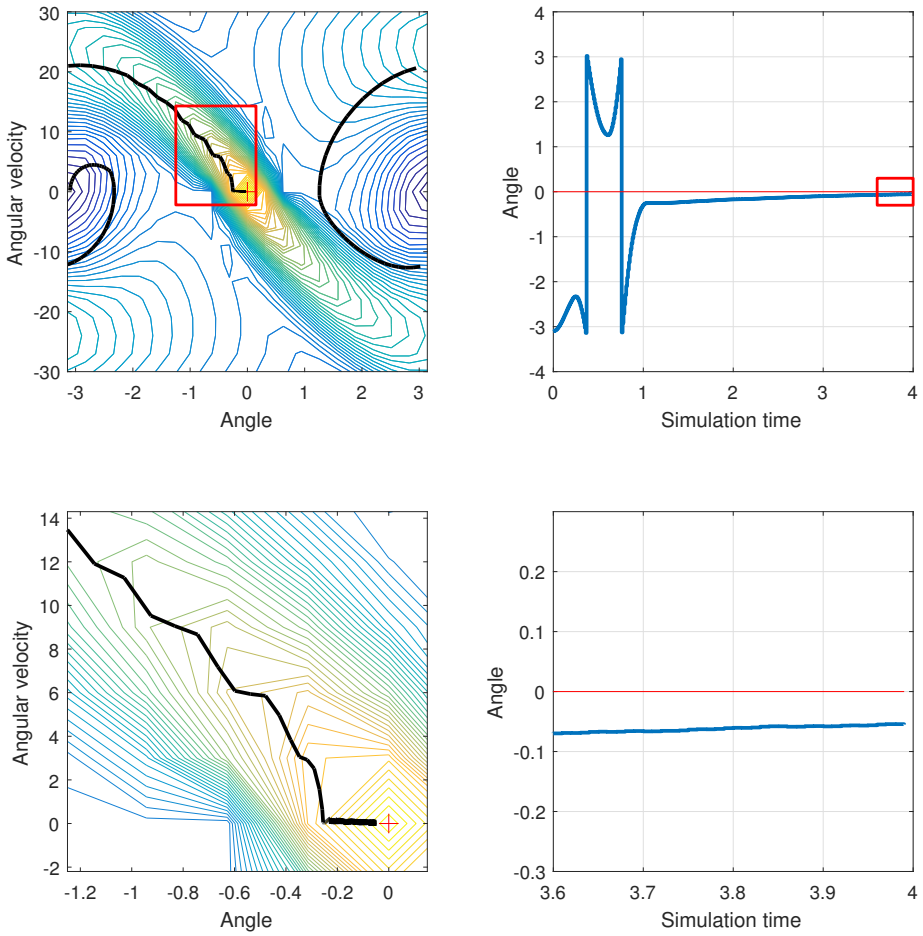

Figure 1: A sample state trajectory obtained by simulating the pendulum swing-up task (see Section 5 for details). The bottom plots show an enlarged view of the areas indicated in the upper plots. The wiggly state trajectory (superimposed on the contours of the Bellman equation RHS) and the extremely slow convergence to the desired position result from the insufficient smoothness of the V-function approximation in combination with the use of discrete actions.

most common approach is to discretize the action space, compute the RHS of the Bellman equation for all the discrete actions, and select the action that corresponds to the largest value. One of the earliest references to this approach is [10]. The drawbacks of this method were discussed in the previous section.

Another similar approach is based on sampling [11, 12]. Using Monte-Carlo estimation, this approach can find a near-optimal action without resorting to exhaustive search over the discretized action space. However, for a good performance, this method requires a large number of samples and therefore it is computationally inefficient.

An alternative method would be *policy interpolation* [13], which is based on computing the control actions off-line for a pre-selected set of states and then interpolating these actions on-line. While computationally less involved, this method does not give any closed-loop stability guarantees and can suffer from severe interpolation errors, especially in constrained problems. Therefore, we do not consider policy interpolation in this paper.

A different approach relies on translating the continuous action selection step into a sequence of binary decisions [14, 15]. Each decision whether to decrease or increase the control action eliminates a part of the action space. This process stops once a predefined precision is reached. There are two main drawbacks of this approach: it requires a binary code representation of each action, which is difficult to design properly and it does not resolve the insufficient smoothness problem of the V-function.

There are several approaches to smoothing data in general, e.g., elastic mapping [16], smoothing splines [17], or approximation by neural networks [18]. However, none of them is directly applicable to RL. To the best of our knowledge, the only approach that has been developed to address the insufficient smoothness problem in the RL context is [8]. It is based on the concept of a smooth *proxy function* which encodes the preference for the optimal action, while it does not have to satisfy the Bellman equation. The proxy function is derived through symbolic regression and can be used for policy derivation. This method has two limitations: it does not allow for penalizing the control action

in the reward function and it is restricted to discrete-valued actions.

In this work, we use symbolic regression [19], which is based on genetic programming, and its goal here is to find an analytic approximation of the sampled V-function. The main benefit is the inherent smoothness of such a V-function approximation, and the possibility to effectively compute its partial derivatives, which makes the search for the optimal control action more effective.

3. Preliminaries

3.1. Reinforcement learning

Define an n-dimensional state space $\mathcal{X} \subset \mathbb{R}^n$, and m-dimensional action space $\mathcal{U} \subset \mathbb{R}^m$. The deterministic model of the process to be controlled is described by the state transition function

$$x_{k+1} \quad = \quad f(x_k, u_k) \tag{1}$$

with $x_k, x_{k+1} \in \mathcal{X}$ and $u_k \in \mathcal{U}$. A user-defined reward function assigns a scalar reward $r_{k+1} \in \mathbb{R}$ to each state transition from x_k to x_{k+1}, as a consequence of action u_k:

$$r_{k+1} \quad = \quad \rho\big(x_k, u_k, x_{k+1}\big) \tag{2}$$

The value function can be computed by solving the Bellman equation:

$$V(x) = \max_{u \in \mathcal{U}} \big[\rho\big(x, u, f(x,u)\big) + \gamma V\big(f(x,u)\big)\big] \tag{3}$$

where γ is a user-defined discount factor. There are several methods to calculate an approximation of the V-function for continuous state spaces. We denote the V-function approximation by $\hat{V}$. In this paper, we use the fuzzy V-iteration algorithm [13] as it is guaranteed to converge and the parameters of the fuzzy approximator are the V-function values at the fuzzy set cores. We briefly review the algorithm in Appendix A.

The most straightforward way [20] to derive the policy $\hat{h}\colon \mathcal{X} \to \mathcal{U}$ corresponding to the approximate value function $\hat{V}(\cdot)$ is:

$$\hat{h}(x) \in \arg\max_{u \in U} \ \Big[\rho\big(x, u, f(x,u)\big) + \gamma \hat{V}\big(f(x,u)\big)\Big] \tag{4}$$

where $U = \left\{u^1, u^2, \ldots, u^M\right\}$ is a finite set of actions drawn from $\mathcal{U}$. However, this policy will be discrete-valued and generally will not perform well on control problems with continuous actions. Therefore, in Section 4 we propose alternative methods, whose aim is to provide a better policy than (4). Note that the transition model f must be available for all the methods considered.

3.2. Symbolic regression

To obtain a smooth approximation of the V-function, we build its analytic approximation using symbolic regression. This general technique is based on genetic programming and its goal is to find an equation describing given training data. Our specific objective is to find an analytic formula that fits well data sampled from the approximated V-function $\hat{V}$. Symbolic regression is a suitable technique for this task, as we cannot assume any kind of a priori knowledge on the shape of the V-function sought. We use a variant of Single Node Genetic Programming, which is described in detail in Appendix B.

4. Policy derivation methods

The first part of this section briefly reviews the available solutions for policy derivation. Then, we introduce a novel method, based on symbolic regression, to address the V-function smoothness problem, described in the Introduction. Furthermore, we propose a computationally efficient method to enable the use of continuous actions.

4.1. Baseline solution

The common solution from the literature is to evaluate (4) at every sampling instant, where the maximization is computed over the same discrete action set U on which $\hat{V}$ has been learned (see Appendix A). We will use this method as the baseline approach.

4.2. Evaluation over a fine grid of actions

It can be beneficial to refine the action space with respect to the space used for computing the V-function. Define $A \subset \mathcal{U}$ as

$$A = A_1 \times A_2 \times \cdots A_m, \tag{5}$$

where each set A_i contains points distributed in a suitable way (e.g., equidistantly) along the ith dimension of the action space. Set A therefore contains all combinations of the control inputs for which the right-hand side of the Bellman equation (4) is evaluated for the current state x:

$$G_x(u) = \rho\big(x, u, f(x,u)\big) + \gamma \hat{V}\big(f(x,u)\big) \tag{6}$$

We refer to G_x as the *control surface*. An example of such as surface for a two-dimensional action space is shown in Fig. 2.

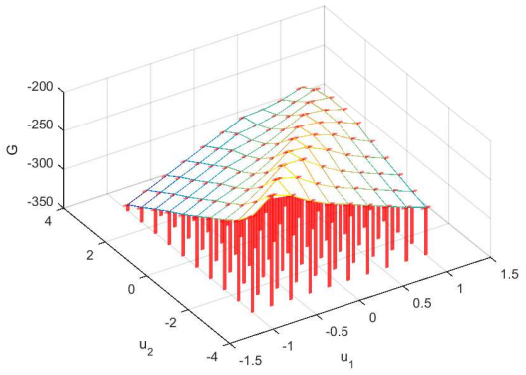

Figure 2: An illustrative example of grid evaluation over the action space for a given x. Each dimension of the action space is discretized in 10 points, i.e., at each time step, a total of 100 points are evaluated according to (6).

This fine discretization allows to control the system by applying actions more precisely, while it does not increase the computational complexity during learning.

4.3. Chebyshev polynomial approximation

The main idea of this method is to find a smooth approximation of the control surface, such as the one shown in Fig. 2. This smooth approximation facilitates more accurate control and helps avoid chattering in the vicinity of unstable system equilibria. We use Chebyshev polynomials of the first kind, which are defined by the following recurrent relation:

$$\begin{aligned} T_0(\bar{u}) &= 0 \\ T_1(\bar{u}) &= \bar{u} \\ T_{n+1}(\bar{u}) &= 2\bar{u}T_n(\bar{u}) - T_{n-1}(\bar{u}) \end{aligned}$$

They are orthogonal to each other on the interval $[-1, 1]$ with respect to the weighting function $1/\sqrt{1-\bar{u}^2}$. In order to take advantage of this property, the domain of each control variable must be mapped onto the interval $[-1, 1]$. This is accomplished using the affine transformation:

$$\bar{u} = -1 + 2\frac{u - u_L}{u_H - u_L}$$

where u_L and u_H are respectively the lower and upper bound of u in each input dimension. The approximation by means of Chebyshev polynomials can be extended to higher dimensions by using the Cartesian product of univariate polynomials. The details of this procedure are beyond the scope of this paper and can be found in [21, 22, 23]. In this paper we use *Chebfun* open-source package [24] to construct a polynomial function $P(u)$, which receives an action $u \in \mathcal{U}$ and returns the value of the approximated right-hand side of the Bellman equation. Note that this function includes the above affine transformation.

The polynomial structure of the policy approximator allows us to efficiently find the maxima in (4) by numerically solving the set of algebraic equations obtained by equating the first partial derivatives to zero, see [25]. In some cases, the polynomial attains its maximum inside the domain $\mathcal{U}$, but in other cases, the maximum lies outside of this domain. Therefore, the boundaries must also be tested. In our experimental study we use *Chebfun* open-source package

[24], which effectively searches for the extrema inside the domain and on its boundaries. The argument of the global maximum of $P(\cdot)$ on the $\mathcal{U}$ domain is then the control action sought. This policy derivation step is formalized in Algorithm 1.

Algorithm 1: Maximization using Chebyshev polynomials (in the sequel denoted as *Cheby*)

Input: $f, \rho, \gamma, A, \hat{V}, x_0$

$k \leftarrow 0$

while *control experiment not terminated* **do**

  foreach $u' \in A$ **do**

    $G[u'] = \rho\big(x_k, u', f(x_k, u')\big) + \gamma \hat{V}\big(f(x_k, u')\big)$;

  end

  build Chebyshev polynomial approximation $P(\cdot)$ using data (A, G)

  $u_k \leftarrow \arg\max_{u' \in \mathcal{U}} P(u')$

  $x_{k+1} \leftarrow f(x_k, u_k)$

  $k \leftarrow k + 1$

end

Output: trajectory $[x_0, x_1, ...]$, $[u_0, u_1, ...]$

4.4. Symbolic regression smoothing

To mitigate the insufficient smoothness problem, we smooth the numerical approximation of the V-function by applying symbolic regression. This results in an analytical expression which accurately describes the V-function, while eliminating artifacts caused by the numerical approximator. In this work, we use the following basic operators and functions to build the analytical expressions: $F = \{*, +, -, square, cube, tanh\}$. The complete symbolic regression procedure is described in Appendix B.

The *Baseline*, *Grid* and *Cheby* algorithms can all be enhanced by using a symbolic approximation $\hat{V}^s$ of the V-function instead of the numerical approximation $\hat{V}$. In the sequel the superscript s denotes symbolic functions or

operations. The modified algorithms are denoted as *SR-Baseline*, *SR-Grid* and *SR-Cheby*, respectively.

4.5. Quasi-symbolic policy derivation

To reduce the computational complexity of policy derivation, the Quasi-Symbolic Policy Derivation algorithm (in the sequel denoted as *QSPD*) exploits the symbolic nature of $\hat{V}^s$. Rather than exhaustively enumerating all possible actions in a discrete action set, we apply a standard numerical optimization algorithm to the following problem:

$$u_k = \arg\max_{u' \in \mathcal{U}} \; R^s(x_k, u') \tag{7}$$

where R^s stands for the right-hand side of the Bellman equation:

$$R^s(x,u) = \Big[\rho^s\big(x,u,f^s(x,u)\big) + \gamma \hat{V}^s(f^s(x,u))\Big] \tag{8}$$

with ρ^s and f^s the symbolic representations of the reward and state-transition function, respectively. In model-based RL, the reward function is always designed by the experimenter and therefore can easily be defined as an analytic function. However, this is often not the case with the state transition function f. The system dynamics are typically described in continuous time and the state transitions are generated by means of numerical integration, using e.g. Runge-Kutta methods. In such as case, the symbolic approximation f^s can be obtained by means of the forward Euler method.

In this paper, we use the trust region reflective (TRR) algorithm [26] and enhance its convergence speed by providing it with the symbolic partial derivatives $\nabla R^s(x,u)$:

$$\nabla R^s(x,u) = \left[\frac{\partial R^s(x,u_1)}{\partial u_1}, \frac{\partial R^s(x,u_2)}{\partial u_2}, \ldots \frac{\partial R^s(x,u_m)}{\partial u_m}\right] \tag{9}$$

The *QSPD* algorithm is presented in Algorithm 2.

5. Simulation experiments

The proposed methods are evaluated on three non-linear control problems: 1-DOF and 2-DOF swing-up, and magnetic manipulation in two variants. The

Algorithm 2: Quasi-symbolic policy derivation (*QSPD*)

Input: $f, f^s, \rho^s, \gamma, \hat{V}^s, x_0$

$k \leftarrow 0$

while *control experiment not terminated* **do**

$u_k = \arg\max_{u' \in \mathcal{U}} \left[\rho^s\big(x, u, f^s(x,u)\big) + \gamma \hat{V}^s\big(f^s(x,u)\big)\right]$; TRR optimization

$x_{k+1} \leftarrow f(x_k, u_k)$;

$k \leftarrow k+1$

end

Output: trajectory $[x_0, x_1, ...]$, $[u_0, u_1, ...]$

characteristics and the parameter values for each problem are listed in Table 1.

Table 1: Experiment parameters

	1-DOF	2-DOF	Magman2	Magman5
Number of state dimensions	2	4	2	2
Number of action dimensions	1	2	2	5
State space, $\mathcal{X}$	$[-\pi, \pi]\times$ $[-30, 30]$	$[-\pi, \pi] \times [-2\pi, 2\pi]\times$ $[-\pi, \pi] \times [-2\pi, 2\pi]$	$[0, 0.05]\times$ $[-0.39, 0.39]$	$[0, 0.1]\times$ $[-0.39, 0.39]$
Input space, $\mathcal{U}$	$[-2, 2]$	$[-3, 3] \times [-1, 1]$	$[0, 0.6]\times$ $[0, 0.6]$	$[0, 0.6]\times$ $\ldots \times [0, 0.6]$
Samples per input dimension, B	$[21, 21]^T$	$[11, 11, 11, 11]^T$	$[27, 27]^T$	$[27, 27]^T$
Discount factor, γ	0.9999	0.99	0.999	0.999
Convergence threshold, ϵ	10^{-5}	10^{-5}	10^{-8}	10^{-8}
Refined action space size	27	$[11, 11]^T$	$[11, 11]^T$	$[11, \cdots, 11]^T$
Simulation time, T_{sim} [s]	10	20	10	10
Sampling period, T_s [s]	0.01	0.01	0.005	0.005

The performance of the algorithms is measured by the following criteria:

- Performance percentage ratio

$$P_{alg} = \frac{1}{N} \sum_{j=1}^{N} \left[p^j_{baseline} / p^j_{alg}\right] \cdot 100\%$$

with $p_{alg} = \sum_{T_{sim}/T_s} \rho\big(x_k, u_k, f(x_k, u_k)\big)$, where the subscript alg refers to

the algorithm tested, N is the number of runs, T_{sim} and T_s are the simulation time and the sampling period, respectively. The reward functions are defined such that they are negative except in the goal state, where they equal to zero. The performance measure P_{alg} is therefore 100% for the baseline and larger than 100% for the algorithms that outperform the baseline.

- Runtime percentage ratio

$$T_{alg} = \frac{1}{N}\sum_{j=1}^{N}\left[t^j_{alg}/t^j_{baseline}\right]\cdot 100\%$$

where t_{alg} stands for the runtime of the algorithm. Analogously to the performance, T_{alg} is 100% for the baseline and smaller than 100% for the algorithms that run faster than the baseline.

- Mean distance of the final state (at the end of simulation) to the goal state

$$D_{alg} = \frac{1}{N}\|x_{des} - x_{end}\|$$

with x_{des} the desired (goal) state, x_{end} the final state of the simulation run, and $\|\cdot\|$ the Mahalanobis norm.

- Mean return

$$R_{alg} = \frac{1}{N}\sum_{j=1}^{N} p_{alg}$$

Each algorithm was tested $N = 50$ times on each benchmark with randomly chosen initial states.

To construct Chebyshev polynomials, we used the *Chebfun* open-source package [24]. The Matlab 2015a `fmincon` implementation of the trust region reflective algorithm is used in *QSPD*.

The genetic programming method evolving the symbolic approximations of the V-function was run with the following control parameters setting:

- The total population size was 300. The ratio of the head partition size to the size of the tail partition was 1:2.

- The maximum number of features the evolved LASSO model can be composed of was set to 16. The maximum depth of individual features was 7.
- The number of parallel threads run in a single epoch was 3. The number of epochs was set to 50.

Multiple independent runs were carried out with symbolic regression to find the symbolic value functions V^s. The symbolic regression parameters did not change between the runs (except for the random seed). The value function which performed best with respect to P_{alg} is presented in the results.

5.1. 1-DOF swing-up

The inverted pendulum consists of a mass m attached to an actuated link that rotates in the vertical plane (see Fig. 3). The available torque is insufficient to push the pendulum up in a single rotation from the majority of initial states. Instead, from a certain state (e.g., pointing down), the pendulum needs to be swung back and forth to gather energy, before it can be pushed up and stabilized.

The continuous-time model of the pendulum dynamics is:

$$\ddot{\alpha} = \frac{1}{J} \cdot \left[mgl \sin(\alpha) - b\dot{\alpha} - \frac{K^2}{R}\dot{\alpha} + \frac{K}{R}u \right] \tag{10}$$

where $J = 1.91 \cdot 10^{-4}\,\mathrm{kgm}^2$, $m = 0.055\,\mathrm{kg}$, $g = 9.81\,\mathrm{ms}^{-2}$, $l = 0.042\,\mathrm{m}$, $b = 3 \cdot 10^{-6}\,\mathrm{Nms/rad}$, $K = 0.0536\,\mathrm{Nm/A}$, $R = 9.5\,\Omega$. The angle α varies in the interval $[-\pi, \pi]$, with $\alpha = 0$ pointing up, and 'wraps around' so that e.g. $\alpha = 3\pi/2$ corresponds to $\alpha = -\pi/2$. The state vector is $x = [\alpha, \dot{\alpha}]^T$. The sampling period

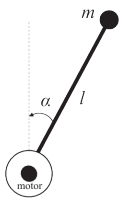

Figure 3: Inverted pendulum schematic.

is $T_s = 0.01\,\mathrm{s}$, and the discrete-time transitions are obtained by numerically integrating the continuous-time dynamics using the fourth-order Runge-Kutta method. The control action u is limited to $[-2, 2]\,\mathrm{V}$, which is insufficient to push up the pendulum in one go. The set of discrete control inputs is $U = \{-2, -1, 0, 1, 2\}$.

The control goal is to stabilize the pendulum in the unstable equilibrium $\alpha = \dot{\alpha} = 0 = x_{des}$ using desired control input $u_{des} = 0$. This goal is expressed by the following reward function:

$$\rho\big(x, u, f(x, u)\big) = -\mathrm{abs}(x_{des} - x)^T Q - \mathrm{abs}(u_{des} - u) \tag{11}$$

with $Q = [5, 1]^T$ a weighting vector to adjust the relative penalty between the angle, angular velocity and control input. Function $abs(\cdot)$ works element-wise.

The simulation results are presented in Table 4, with the best performance printed in bold.

The most important conclusion is that the use of symbolic regression improves the overall performance, with *SR-Cheby* yielding the best results. All the proposed approaches result in longer runtime, except for *SR-Baseline*. *Grid* and *Cheby* algorithms perform worse than the baseline solution with respect to P_{alg}. This is caused by the combination of insufficient smoothness of the value function and discrete actions, as illustrated in Figure 4. The approximation by means of basis functions produces a "ridge" in the vicinity of the goal state. The state trajectory then moves back and forth from one side of the ridge to the other, slowly converging toward the goal state. The *Baseline* algorithm employs fewer actions and is therefore forced to use actions with a larger magnitude. The *Grid* algorithm, on the contrary, employs more actions with a smaller magnitude. Therefore, it can follow the ridge more precisely, which results in a larger value of the Bellman equation RHS, but a slower overall convergence to the goal. The *Cheby* algorithm uses data provided by the *Grid* algorithm and therefore suffers from the same problem.

The *QSPD* algorithm alleviates this problem thanks to the use of smooth symbolic approximation of the V-function in combination with continuous ac-

tions. The state trajectory is virtually ideal, as depicted at the right-hand side of Figure 4.

5.2. 2-DOF swing-up

The double pendulum is described by the continuous-time fourth-order nonlinear model:

$$
\begin{aligned}
u &= M(\alpha)\ddot{\alpha} + C(\alpha,\dot{\alpha})\alpha + G(\alpha) \\
M(\alpha) &= \begin{bmatrix} P_1 + P_2 + 2P_3\ \cos(\alpha_2) & P_2 + P_3\ \cos(\alpha_2) \\ P_2 + P_3\ \cos(\alpha_2) & P_2 \end{bmatrix} \\
C(\alpha,\dot{\alpha}) &= \begin{bmatrix} b_1 - P_3\dot{\alpha_2}\sin(\alpha_2) & -P_3(\dot{\alpha_1} + \dot{\alpha_2})\sin(\alpha_2) \\ P_3\dot{\alpha_1}\sin(\alpha 2) & b_2 \end{bmatrix} \\
G(\alpha) &= \begin{bmatrix} -F_1\sin(\alpha_1) - F_2\sin(\alpha_1+\alpha_2) \\ -F_2\sin(\alpha_1+\alpha_2) \end{bmatrix}
\end{aligned}
\tag{12}
$$

where $\alpha = [\alpha_1, \alpha_2]^T$ contains the angular positions of the two links, $u = [u_1, u_2]^T$ is the control input which contains the torques of the two motors, $M(\alpha)$ is the mass matrix, $C(\alpha,\dot{\alpha})$ is the Coriolis and centrifugal forces matrix and $G(\alpha)$ is the gravitational forces vector. The state x contains the angles and angular velocities and is defined as $x = [\alpha_1, \dot{\alpha_1}, \alpha_2, \dot{\alpha_2}]^T$. The angles $[\alpha_1, \alpha_2]$ vary in the interval $[-\pi, \pi)$ rad and wrap around. The auxiliary variables are defined as $P_1 = m_1c_1^2 + m_2l_1^2 + I_1$, $P_2 = m_2c_2^2 + I_2$, $P_3 = m_2l_1c_2$, $F_1 = (m_1c_1 + m_2l_2)g$ and $F_2 = m_2c_2g$. The control action u is limited to $[-3, 3]$ for the first link and $[-1, 1]$ for the second link. The meaning and the values of the system parameters are given in Table 2.

The discrete set of control inputs U is defined as the Cartesian product of the sets $\{-3, 0, 3\}$ and $\{-1, 0, 1\}$ for each link. The transition function $f(x, u)$ is obtained by numerically integrating (12) between discrete time samples using the fourth-order Runge-Kutta method with the sampling period T_s.

The control goal is expressed by the following quadratic reward function:

$$
\rho\big(x, u, f(x,u)\big) = -\text{abs}(x_{des} - x)^T Q, \text{ where } Q = [1,\ \ 0.05,\ \ 1.2,\ \ 0.05]^T \tag{13}
$$

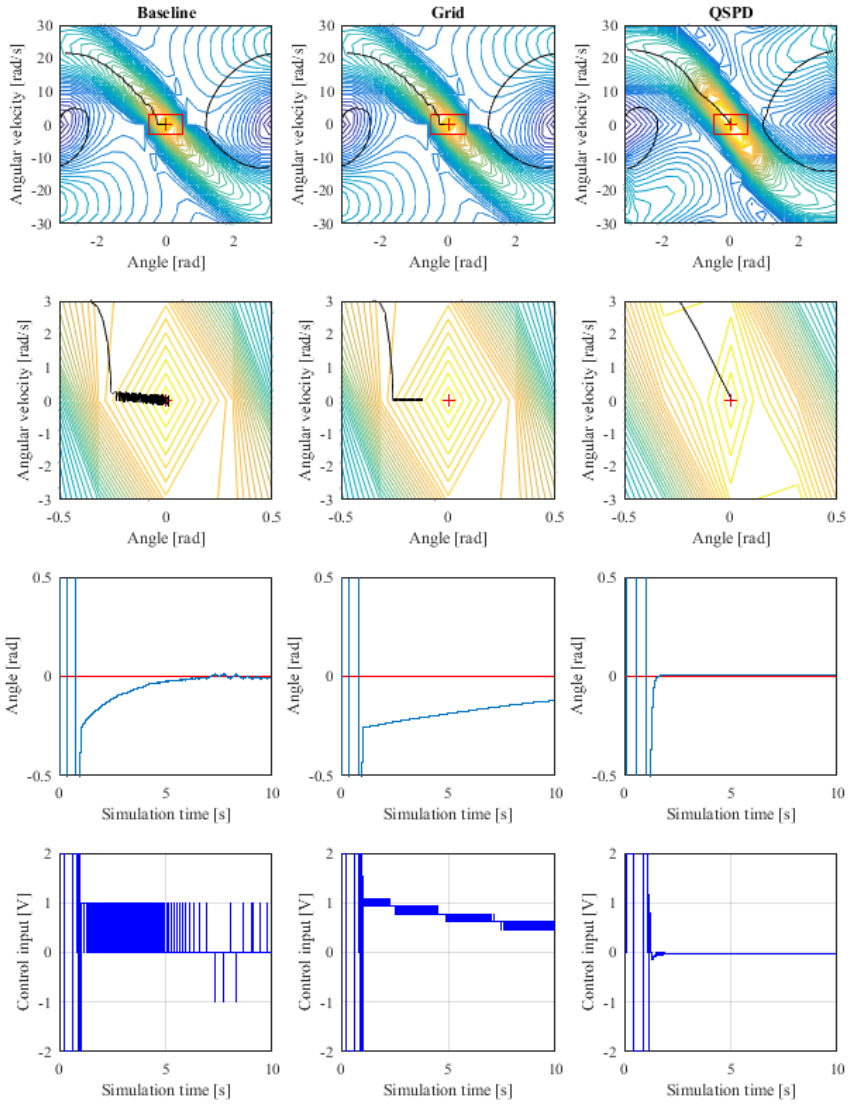

Figure 4: Trajectories from the initial point $[-\pi, 0]$ to the goal state, obtained using Baseline (left column), Grid (middle column) and QSPD (right column) on the 1-DOF pendulum swing-up task. The first row shows the whole state trajectory superimposed on the contours of the Bellman equation RHS, the second row the zoomed area around the goal state, the third row the angle, and the forth row the applied control inputs during simulation.

Table 2: Double pendulum parameters

Model parameter	Symbol	Value	Unit
Link lengths	l_1, l_2	0.4, 0.4	m
Link masses	m_1, m_2	1.25, 0.8	kg
Link inertias	I_1, I_2	0.0667, 0.0427	$\mathrm{kg\,m^2}$
Center of mass coordinates	c_1, c_2	0.2, 0.2	m
Damping in the joints	b_1, b_2	0.08, 0.02	kg/s
Gravitational acceleration	g	-9.8	$\mathrm{m/s^2}$
Sampling period	T_s	0.01	s
Desired state	x_{des}	$[0, 0, 0, 0]^T$	

The simulation results presented in Table 4 show the same pattern as with the 1-DOF pendulum swing-up. Also here the symbolically approximated value function leads to a significant improvement in performance.

5.3. Magnetic manipulation

Magnetic manipulation (abbreviated as Magman) is an challenging nonlinear control problem. In our setup (see Fig. 5), the current through the electromagnets is controlled to dynamically shape the magnetic field above the magnets and so to accurately and quickly position a steel ball to the desired set point. We consider two variants, one with two electromagnets and one with five.

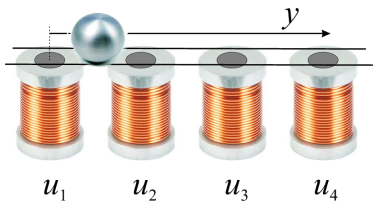

Figure 5: Magman schematic.

The horizontal acceleration of the ball is given by:

$$\ddot{y} = -\frac{b}{m}\dot{y} + \frac{1}{m}\sum_{i=0}^{1} g(y,i)\, u_i \tag{14}$$

with

$$g(y,i) = \frac{-c_1\ (y - 0.025i)}{\Big((y - 0.025i)^2 + c_2\Big)^3}. \tag{15}$$

Here, y denotes the position of the ball, $\dot{y}$ its velocity and $\ddot{y}$ the acceleration. With u_i the current through coil i, $g(y,i)$ is the nonlinear magnetic force equation, m (kg) the ball mass, and b $(\frac{Ns}{m})$ the viscous friction of the ball on the rail. Magnetic manipulation task with two coils uses $i = 0, 1$, and in the sequel is named Magman2. The five-coil variant uses $i = 0, 1, 2, 3, 4$ and is named Magman5. The model parameters are listed in Table 3.

Table 3: Magnetic manipulation system parameters

Model parameter	Symbol	Value	Unit
Ball mass	m	$3.200 \cdot 10^{-2}$	kg
Viscous damping	b	$1.613 \cdot 10^{-2}$	Nms
Empirical parameter	c_1	$5.520 \cdot 10^{-10}$	$\mathrm{Nm^5A^{-1}}$
Empirical parameter	c_2	$1.750 \cdot 10^{-4}$	$\mathrm{m^2}$
Sampling period	T_s	0.005	s

State x is given by the position and velocity of the ball. The set of control inputs U is defined as the Cartesian product of the vectors $[0, 0.3, 0.6]$, containing the discrete values of the control input to the ith coil. The reward function is defined as:

$$\rho\big(x, u, f(x,u)\big) = -\mathrm{abs}(x_{des} - x)^T Q, \text{ where } Q = [5,\ \ 1]^T \tag{16}$$

where the desired position x_{des} is set to 0.01 (m).

The simulation results for both variants of the system are presented in Table 4.

The *QSPD* algorithm shows the best result in terms of the overall return. Also here, *SR-Baseline* tends to be the fastest algorithm. However, when the

dimensionality of the action space grows, *QSPD* starts dominating in both measures. The runtime of Baseline, Grid and Cheby algorithms (and their symbolic regression variants) grows exponentially with the input dimension. This prevents us from applying *Grid* and *Cheby* algorithms (and their symbolic variants) to the Magman5 benchmark, so their results are not reported in the table. The runtime of the *QSPD* algorithm, according to our experiments, grows approximately linearly. This results in the outstanding runtime performance $T_{alg} = 2.85\%$ for the Magman 5 benchmark.

6. Discussion

Table 4 summarizes the simulation results. As can be seen, *Grid* and *Cheby* algorithms perform worse than the *Baseline* solution with respect to P_{alg}. This is caused by the insufficient smoothness problem, illustrated in Figure 4. However, when combined with the smooth approximation of the V-function surface by means of symbolic regression, these algorithms lead to a significant performance improvement. Moreover, the use of the symbolically approximated surface alone is beneficial as well, as illustrated by the *SR-Baseline* results. It can be seen, that the difference between *SR-Grid* and *SR-Cheby* is quite unpredictable and depends on the quality of approximation of the local features of the V-function surface.

The *QSPD* algorithm performs better when the dimensionality of the action space grows. The reason why *QSPD* performs worse compared to the SR algorithms is that it uses a rather poor Euler approximation $f^s(\cdot)$ of the system dynamics. Instead of the Euler method, another more sophisticated approximation could be used, e.g., the Adams-Bashforth method.

As can be seen in Table 4, the exponential growth of computational complexity prevents the use of *Grid* and *Cheby* algorithms (and their symbolic variants) on the Magman5 benchmark. Interestingly, *SR-Baseline* runs faster than the *Baseline* solution. This is thanks to the simpler form of the V-function approximation, which leads to faster computations.

Table 4: Summary of the simulation results

		1DOF swing-up	2DOF swing-up	Magman2	Magman5
Baseline	P_{alg}	100%	100%	100%	100%
	T_{alg}	100%	100%	100%	100%
	D_{alg}	0.0493	0.0825	0.0007	0.0019
	R_{alg}	-1288.28	-869.81	-14.42	-35.85
Grid	P_{alg}	47.45%	91.25%	75.09%	-
	T_{alg}	634.19%	1159.30%	965.26%	-
	D_{alg}	0.1175	0.1132	0.0010	-
	R_{alg}	-2582.65	-969.374	-19.34	-
Cheby	P_{alg}	48.03%	95.69%	130.77%	-
	T_{alg}	2281.80%	1937.80%	3641.10%	-
	D_{alg}	0.1147	0.0946	0.0003	-
	R_{alg}	-2551.43	-925.85	-11.76	-
SR-Baseline	P_{alg}	129.33%	247.80%	163.09%	242.31%
	T_{alg}	**86.01**%	**77.59**%	**84.81**%	87.26%
	D_{alg}	0.0901	0.1050	0.0002	0.0004
	R_{alg}	-1040.77	-573.49	-9.46	-16.02
SR-Grid	P_{alg}	166.54%	287.11%	178.80%	-
	T_{alg}	589.74%	909.47%	885.85%	-
	D_{alg}	0.0065	0.0408	**0.0001**	-
	R_{alg}	-887.48	-547.08	-8.89	-
SR-Cheby	P_{alg}	**188.89**%	**293.25**%	176.96%	-
	T_{alg}	2200.80%	1626.90%	4092.20%	-
	D_{alg}	**0.0000**	**0.0233**	0.0003	-
	R_{alg}	**-837.32**	**-545.38**	-9.26	-
QSPD	P_{alg}	149.67%	193.61%	**254.08**%	**332.98**%
	T_{alg}	586.94%	611.23%	208.24%	**2.85**%
	D_{alg}	0.0076	0.1401	**0.0001**	**0.0001**
	R_{alg}	-955.11	-667.50	**-7.39**	**-12.55**

7. Conclusion

Several alternative policy derivation methods for continuous action spaces were proposed in this paper. The simulation results showed that the proposed symbolic approximation significantly outperforms the standard policy derivation approach. Moreover, in combination with the *QSPD* algorithm, it allows to curb the exponentially growing computational complexity.

In terms of control performance, *SR-Cheby* outperformed the other algorithms in most of the problems tested (with the exception of the Magman 2 example). However, when the dimensionality of action space is larger than three, the computational complexity of this algorithm becomes prohibitive. In applications where computational resources are limited (e.g., in robotics) we recommend to use the *SR-Baseline* algorithm. The *QSPD* algorithm is suitable for problems with input dimension larger than three.

The *Grid* and *Cheby* algorithms have not performed equally well across the all problems tested; they perform worse when the V-function approximation is not smooth. Providing the right criteria for measuring this aspect is difficult and it may be a part of future research. In conclusion, it is not recommended to use these algorithms out of the box.

The proposed methods can be used with any kind of value function approximators. Incorporating symbolic regression into the learning process is a part of our future work.

8. Acknowledgements

This research was supported by the Grant Agency of the Czech Republic (GAČR) with the grant no. 15-22731S titled "Symbolic Regression for Reinforcement Learning in Continuous Spaces" and by the Grant Agency of the Czech Technical University in Prague, grant no. SGS16/228/OHK3/3T/13 titled "Knowledge extraction from reinforcement learners in continuous spaces".

Appendix A. Fuzzy value iteration

Define a set of samples $S = \{s_1, s_2, \ldots, s_N\}$ placed on an equidistant rectangular grid in $\mathcal{X}$. The number of grid points per dimension is described by vector $B = [b_1, b_2, \ldots, b_n]^T$ with the total number of samples $N = \prod_{i=1}^{n} b_i$. Further define a vector of fixed triangular membership functions $\phi = [\phi_1(x), \phi_2(x), \ldots, \phi_N(x)]^T$ where each $\phi_i(x)$ is centered in s_i, i.e., $\phi_i(s_i) = 1$ and $\phi_j(s_i) = 0$, $\forall j \neq i$. The basis functions are normalized so that $\sum_{j=1}^{N} \phi_j(x) = 1$, $\forall x \in \mathcal{X}$. The V-function approximation is defined as:

$$\hat{V}(x) = \theta^T \phi(x) \tag{A.1}$$

where $\theta = [\theta_1, \ldots, \theta_N]^T \in \mathbb{R}^N$ is a parameter vector found through the following value iteration:

$$\theta_i \leftarrow \max_{u \in U} \Big[\rho\big(s_i, u, f(s_i, u)\big) + \gamma \theta^T \phi\big(f(s_i, u)\big) \Big] \tag{A.2}$$

for $i = 1, 2, \ldots, N$, with $U = \{u^1, u^2, \ldots, u^M\}$ a finite set of actions drawn from $\mathcal{U}$. This value iteration is guaranteed to converge [13] and terminates when the following condition is satisfied:

$$||\theta - \theta^-||_\infty \leq \epsilon \tag{A.3}$$

with θ^- the parameter vector calculated in the previous iteration and ϵ a user-defined convergence threshold. Fuzzy value iteration is very effective for second-order systems – computing the value function is a matter of seconds. However, the computational and memory requirements grow exponentially and the method is not practical for systems above order four. Other methods [27] can be used for higher-order systems.

Appendix B. Single node genetic programming

In this work, we used a Single Node Genetic Programming (SNGP) [28, 29] to implement the symbolic regression. SNGP is a graph-based GP method that evolves a population of individuals, each consisting of a single program node.

The node can be either terminal, i.e., a constant or a variable in case of the symbolic regression problem, or a function chosen from a set of functions defined for the problem at hand. The individuals are not entirely distinct, instead they are interlinked in a graph structure so some individuals act as input operands of other individuals.

Formally, a SNGP population is a set of L individuals $M = \{m_0, m_1, \ldots, m_{L-1}\}$, with each individual m_i being a single node represented by the tuple $m_i = \langle e_i, f_i, Succ_i, Pred_i, O_i \rangle$, where

- $e_i \in T \cup F$ is either an element chosen from a function set F or a terminal set T defined for the problem;
- f_i is the fitness of the individual;
- $Succ_i$ is a set of successors of this node, i.e. the nodes whose output serves as the input to the node;
- $Pred_i$ is a set of predecessors of this node, i.e. the nodes that use this node as an operand;
- O_i is a vector of outputs produced by this node.

In this work, the following basic operators and functions were used to build the symbolic expressions $F = \{*, +, -, square, cube, tanh\}$. The terminal set T consisted of the state variables x_i and basic constants 0, 1, 2, and 3.

Typically, the population is partitioned so that the first L_{term} nodes are terminals, basic constants and variables, followed by function nodes. Links between nodes in the population must satisfy a condition that any function node can use as its successor (i.e. the operand) only nodes that are positioned lower down in the population. This means that for each $m_j \in Succ_i$ it holds $0 \leq j < i$. Similarly, predecessors of individual i must occupy higher positions in the population, i.e. for each $m_j \in Pred_i$ we have $i < j < L$. Note that each function node is in fact a root of a program tree that can be constructed by recursively traversing its successors towards the leaf terminal nodes.

An operator called *successor mutation* (*smut*), proposed in [28], is used to modify the population. It takes an individual and replaces one of its successors by a reference to another randomly chosen individual of the population making sure that the constraint imposed on successors is satisfied. Output values of the mutated node and all nodes higher up in the population affected by the mutation operation are recalculated. Also the predecessor lists of all affected nodes are updated accordingly.

Finally, the population is evolved using a local search-like procedure. In each iteration, a new population is produced by the *smut* operator which is then accepted for the next iteration if it is no worse than the original population.

The SNGP implementation used in this work to solve the symbolic regression problem differs from the one described in [28, 29] in the following aspects

- Organization of nodes in the population. We use a population that divides the function nodes into head and tail partitions as introduced in [8]. Any node in the head partition can use other head function nodes and constant terminal nodes as its input. Tail nodes can use all preceding head and tail nodes and both constants and variables as their input. The head nodes therefore represent only constant output expressions, while the tail nodes can represent functions with complex coefficients produced by the head nodes.

- Form of the final model. A hybrid SNGP proposed in [30] and denoted there as the Single-Run SNGP with LASSO (s-SNGPL) is used in this work. It produces a generalized linear regression model composed of possibly nonlinear features, represented by expressions rooted in the tail partition nodes. The generalized linear regression models are built using the Least Absolute Shrinkage and Selection (LASSO) regression technique [31]. In this way, a precise, linear-in-parameters nonlinear regression models can be produced. The complexity of the LASSO model is controlled by (i) the maximal depth of features evolved in the population and (ii) the maximum number of features the LASSO model can be composed of.

- Fitness function. The generalized regression models are optimized with respect to the mean squared error calculated over the set of training samples.

- Selection strategy used to choose the nodes to be mutated. The original SNGP selects the nodes to be mutated purely at random. Here we use the so-called depth-wise selection introduced in [30]. This selection method is biased toward deeper nodes of well-performing expressions. The idea behind such a strategy is that changes made to the nodes at deeper levels are more likely to bring an improvement than changes applied to the nodes close to the root of the expression. The quality of the individual nodes is assessed as the mean squared error produced by the expression rooted in the node.

- Processing mode to evolve the population. The process of evolving the population is carried out in epochs. In each epoch, multiple independent parallel threads are run for a predefined number of generations, all of them starting from the same population – the best final population out of the previous epoch threads. This reduces the chance of getting stuck in a local optimum.

References

[1] R. Sutton, A. Barto, Reinforcement learning: An introduction, Vol. 116, Cambridge Univ Press, 1998.

[2] A. S. Polydoros, L. Nalpantidis, Survey of model-based reinforcement learning: Applications on robotics., Journal of Intelligent and Robotic Systems 86 (2) (2017) 153–173.

[3] L. Kuvayev, R. S. Sutton, Model-based reinforcement learning with an approximate, learned model, in: Proc. Yale Workshop Adapt. Learn. Syst, 1996, pp. 101–105.

[4] V. Konda, J. Tsitsiklis, Actor-critic algorithms, in: SIAM Journal on Control and Optimization, MIT Press, 2000, pp. 1008–1014.

[5] F. Lewis, D. Vrabie, K. Vamvoudakis, Reinforcement Learning and Feedback Control, IEEE Control Systems Magazine 32 (6) (2012) 76–105.

[6] J. A. Primbs, V. Nevistić, J. C. Doyle, Nonlinear optimal control: A control lyapunov function and receding horizon perspective, Asian Journal of Control 1 (1) (1999) 14–24.

[7] D. P. Bertsekas, Dynamic programming and optimal control 3rd edition, volume ii, Belmont, MA: Athena Scientific.

[8] E. Alibekov, J. Kubalík, R. Babuška, Symbolic method for deriving policy in reinforcement learning, in: Decision and Control (CDC), 2016 IEEE 55th Conference on, IEEE, 2016, pp. 2789–2795.

[9] E. Alibekov, J. Kubalík, R. Babuška, Policy derivation methods for critic-only reinforcement learning in continuous action spaces, IFAC-PapersOnLine 49 (5) (2016) 285–290.

[10] J. C. Santamaria, R. S. Sutton, A. Ram, Experiments with reinforcement learning in problems with continuous state and action spaces (1996).

[11] B. Sallans, G. E. Hinton, Reinforcement learning with factored states and actions, J. Mach. Learn. Res. 5 (2004) 1063–1088.

[12] H. Kimura, Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and gibbs sampling, in: Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on, 2007, pp. 88–95.

[13] L. Busoniu, R. Babuska, B. D. Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, 1st Edition, CRC Press, Inc., Boca Raton, FL, USA, 2010.

[14] J. Pazis, M. G. Lagoudakis, Binary action search for learning continuous-action control policies, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, ACM, New York, NY, USA, 2009, pp. 793–800.

[15] J. Pazis, M. Lagoudakis, Reinforcement learning in multidimensional continuous action spaces, in: Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on, 2011, pp. 97–104.

[16] J. Yang, S. Huang, K. Lin, J. Czernin, P. Wolfenden, M. Dahlbom, C. Hoh, M. Phelps, A new axial smoothing method based on elastic mapping, IEEE Transactions on Nuclear Science 43 (6) (1996) 3355–3360.

[17] N. Graham, Smoothing with periodic cubic splines, Bell System Technical Journal 62 (1) (1983) 101–110.

[18] S. Ferrari, R. F. Stengel, Smooth function approximation using neural networks, IEEE Transactions on Neural Networks 16 (1) (2005) 24–38.

[19] J. Kubalik, E. Alibekov, J. Zegklitz, R. Babuska, Hybrid single node genetic programming for symbolic regressionAccepted as a regular paper in LNCS Transactions on Computational Collective Intelligence.

[20] D. P. Bertsekas, J. N. Tsitsiklis, Neuro-Dynamic Programming, 1st Edition, Athena Scientific, 1996.

[21] R. O. Hays, Multi-dimensional extension of the Chebyshev polynomials, Mathematics of computation 27 (123) (1973) 621–624.

[22] L. N. Townsend, Alex; Trefethen, An extension of Chebfun to two dimensions, SIAM Journal on Scientific Computing 35.

[23] B. Hashemi, L. N. Trefethen, Chebfun in three dimensions, Under review by the SIAM Journal on Scientific Computing.

[24] T. A. Driscoll, N. Hale, L. N. Trefethen, Chebfun Guide, Pafnuty Publications, 2014.

[25] J. P. Boyd, Computing zeros on a real interval through Chebyshev expansion and polynomial rootfinding, SIAM Journal on Numerical Analysis 40 (5) (2002) 1666–1682.

[26] M. A. Branch, T. F. Coleman, Y. Li, A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems, SIAM Journal on Scientific Computing 21 (1) (1999) 1–23.

[27] D. Ernst, P. Geurts, L. Wehenkel, Tree-based batch mode reinforcement learning, Journal of Machine Learning Research 6 (2005) 503–556.

[28] D. Jackson, A New, Node-Focused Model for Genetic Programming, Springer, Berlin, Heidelberg, 2012, pp. 49–60.

[29] D. Jackson, Single Node Genetic Programming on Problems with Side Effects, Springer, Berlin, Heidelberg, 2012, pp. 327–336.

[30] J. Kubalík, E. Alibekov, J. Žegklitz, R. Babuška, Hybrid Single Node Genetic Programming for Symbolic Regression, Springer, Berlin, Heidelberg, 2016, pp. 61–82.

[31] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society, Series B 58 (1994) 267–288.