Synthesis of Optimal Feedback Controllers from Data via Stochastic Gradient Descent

Laura Ferrarotti* and Alberto Bemporad*

Abstract— **We propose a policy search method for synthesizing optimal feedback control laws for reference tracking directly from data. During the learning phase, the control law is optimized by using stochastic gradient descent iterations and (optionally) applied to the plant while collecting data. Differently from model-based methods, in which a full model of the open-loop plant is first identified from data, here a simple linear model is recursively identified with forgetting factor for the only reason of computing approximately the gradients required for the descent. We report examples showing that the method recovers the optimal feedback law in case the underlying plant is linear, and outperforms the best control law that is achieved by first identifying an open-loop linear model in case the underlying plant is nonlinear.**

I. INTRODUCTION

Data-driven synthesis of control systems has recently gained increasing popularity within the control community. The idea is to construct control laws directly from experimental data without going first through the identification of an open-loop model of the process. In [1] the authors combine direct data-driven controller synthesis with model predictive control (MPC) to synthesize optimized control laws that can handle constraints without the need of a model of the open-loop process, leaving the nominal model of the unconstrained closed-loop system as a degree of freedom in the design. Nonlinear optimization over such a nominal closed-loop model is proposed in [2] to design optimal control laws from data. In [3], a model-free approach is taken by considering a database of past input/state trajectories and optimizing their linear combination in real-time with respect to a given performance index.

Reinforcement Learning (RL) also exploits experience gathered from process/environment interactions to design optimal data-driven control laws. The close relation between the design of control laws for continuous state and input spaces and RL is underlined in [4], using the linear quadratic regulator (LQR) as a benchmark. In the RL framework, the key concepts of action and cost (or reward) are mathematically expressed through the concepts of *policy* and *value function*, respectively. Based on how they treat the policy and value functions, RL methods are usually categorized in *actor-only* methods, *critic-only* methods, and *actor-critic* methods [5]. Critic-only and actor-critic methods calculate control policies by using an approximation of the value function, while this is avoided in actor-only methods. A remarkable example of critic-only method is *Q-learning* [6], that requires no model of the system dynamics. When applied to LQR, Q-learning was proved to converge in [7], provided the persistence of excitation by adding exploration noise on the control action. Actor-only methods, instead, avoid giving an overall estimate of the cost-function, as they work with a parameterized family of policies and search the optimal policy directly in the policy parameter space; for this reason they are also called *policy search* methods. Using parametrized policies gives one the advantage of effectively dealing with a continuous set of actions and high dimensional, including continuous, state spaces [8], [9].

Policy search algorithms can be divided into *model-free* and *model-based* methods. Model-based algorithms use data collected from the process to learn a model of its open-loop dynamics, then used for control design. Model-free methods attempt instead at learning the control policy directly from collected data; as intermediate policies might be quite off, additional care must be put during the learning phase to avoid risky operations on the plant [10], [11].

Solid theoretical foundation, starting from the guarantee of convergence when applied to LQR, is achieved for algorithms belonging to the subclass of policy search methods denoted as *policy gradient* (see [12], [13]). These methods rely upon optimizing parametrized policies following the direction indicated by the gradient of the cost-function. In [14] conditions on the principal policy gradient updates are given to guarantee convergence to a globally optimal solution, efficiently with respect to the number of samples and computational complexity.

In this paper we propose a control design approach that belongs to the policy gradient family. Our method is not model-based, in the sense that it does not first identify a complete model of the open-loop system to then design a control strategy. Nonetheless, it is not completely model-free, as it needs to compute simple local linear models that are estimated on line from the input/output data stream. The purpose of such local linear models is not to provide information on the overall dynamics of the plant, that could be possibly nonlinear, but rather merely to approximate the gradient of the performance index driving the synthesis of the control law. Focusing on solving output tracking problems for arbitrary reference signals, we synthesize the corresponding control law by learning it iteratively on line by Stochastic Gradient Descent (SGD) [15], so to optimize a given performance index that weights tracking errors and input increments.

The paper is organized as follows. The problem of synthesising optimal policies is formulated in Section II. A detailed description of the proposed algorithm follows in Section III.

*IMT School for Advanced Studies Lucca, Lucca, Piazza S.Francesco 19, 55100 Italy. Email: {laura.ferrarotti,alberto.bemporad}@imtlucca.it

Numerical results related to the synthesis of linear feedback laws for linear and nonlinear systems are shown in Section IV. Some final conclusions are drawn in Section V.

Notation. Let $\mathbb{R}^n$ be the set of real vectors of dimension n. Given a matrix $A \in \mathbb{R}^{n\times m}$ we denote its transpose by A'. If $x \in \mathbb{R}^n$ is a vector, then x_i is its ith element. Given a matrix $Q \in \mathbb{R}^{n\times n}$, $\|x\|_Q^2 = x' Q x$. We denote by I the identity matrix.

II. PROBLEM FORMULATION

We describe the dynamics of a plant interacting with its environment by a Markovian signal $s_t \in \mathbb{R}^{n_s}$ that evolves in time according to the following (unknown) model

$$s_{t+1} = h(s_t, p_t, u_t, d_t) \tag{1}$$

where $p_t \in \mathbb{R}^{n_s}$ is a vector of measured exogenous signals, $u_t \in \mathbb{R}^{n_u}$ a vector of decision variables, and $d_t \in \mathbb{R}^{n_d}$ a vector of unmeasured disturbances. The components of s_t may include the state $x_t \in \mathbb{R}^{n_x}$ of an (unknown) plant model

$$x_{t+1} = f(x_t, u_t, d_t) \tag{2a}$$
$$y_t = g(x_t, d_t) \tag{2b}$$

driven by the command input u_t and the disturbance d_t, with output $y_t \in \mathbb{R}^{n_y}$ that we want to track a reference signal $r_t \in \mathbb{R}^{n_y}$. In case the plant is described in input/output form, s_t may include a finite set x_t of n_i past input and n_o past output values, $x_t \in \mathbb{R}^{n_o n_y + n_i n_u}$, that is

$$x_t = [y_t' \;\dots\; y_{t-n_o+1}' \; u_{t-1}' \;\dots\; u_{t-n_i}']' \tag{3}$$

along with (2b) and the state-update equation

$$x_{t+1} = [g(f(x_t, u_t, d_t), d_{t+1})' \dots \; y_{t-n_o+2}' \; u_t \dots \; u_{t-n_i+1}']'$$

Vector s_t may also include additional states that one wants to include in the control policy, such as the integrals of output tracking errors, as we will show later. The non-Markovian signal $p_t \in \mathbb{R}^{n_p}$, that we will model as a random vector, can represent instead exogenous variables, such as measured disturbances, time-varying parameters, and/or the reference signal r_t itself.

We define a *deterministic control policy* π as a function $\pi : \mathbb{R}^{n_s+n_p} \to \mathbb{R}^{n_u}$ that associates to each s_t and p_t an action $u_t = \pi(s_t, p_t)$. In order to define the concept of optimal policy, we introduce the *stage cost* $\rho : \mathbb{R}^{n_s+n_p+n_u} \to \mathbb{R}$, a real-valued function such that $\rho(s_t, p_t, u_t)$ represents the cost of applying the input u_t while in (s_t, p_t). Then, for a given initial condition s_0 and values $p_0, d_0, p_1, d_1, \dots$ we set

$$J_\infty(\pi, s_0, \{p_l, d_l\}_{l=0}^{\infty}) = \sum_{l=0}^{\infty} \rho(s_l, p_l, \pi(s_l, p_l)) \tag{4}$$

as the cost of applying the deterministic policy π to (1) over an infinite time execution, with $(s_{l+1}, s_l, p_l, \pi(s_l, p_l), d_l)$ satisfying (1) for all l. Finally, we characterize the overall cost of a deterministic policy π by introducing the *performance index*

$$J(\pi) = \mathbb{E}_{S_0, \{P_l, D_l\}_{l=0}^{\infty}}[J_\infty(\pi, S_0, \{P_l, D_l\})]$$

where the expectation of J_∞ is taken with respect to the random variables S_0 and P_l, D_l, $l = 0, 1, \dots$, representing the initial point of the trajectory and the value of signals p_l, d_l at step l, respectively.

Our aim is to find a policy π^* that optimizes

$$\pi^* = \arg \min_{\pi \in \mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})} J(\pi) \tag{5}$$

where $\mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})$ is the set of functions of $n_s + n_p$ real variables taking values in $\mathbb{R}^{n_u}$.

A. Approximation of policy optimization problem

Problem (5) is a general abstract problem of optimal policy search. In order to find a computable solution to it, in this paper we make the following approximations:

- We parametrize the policy π by a matrix K of parameters, $K \in \mathbb{R}^{n_u \times n_k}$, and denote by $\pi_K(s_t, p_t)$ the corresponding policy.
- We consider a finite trajectory of length L for evaluating the cost of the policy, i.e., we use

$$J_L(K, s_0, \{p_l, d_l\}_{l=0}^{L-1}) = \sum_{l=0}^{L-1} \rho(s_l, p_l, \pi_K(s_l, p_l)) \tag{6}$$

instead of (4).

Under the above simplifications, problem (5) is approximated as the following optimization problem

$$K^* = \arg \min_K \; \mathbb{E}_{\substack{S_0,\\ \{P_l, D_l\}_{l=0}^{\infty}}} \left[\sum_{l=0}^{L-1} \rho(S_l, P_l, \pi_K(S_l, P_l)) \right] \tag{7a}$$

with

$$S_{l+1} = h(S_l, P_l, D_l) \tag{7b}$$

We solve problem (7) using the mini-batch SGD algorithm [15]. The algorithm works as follows: given a function $F(x, y)$, it attempts computing $x^* = \arg\min_x \mathbb{E}_Y[F(x, Y)]$ by updating the candidate solution x_t recursively as

$$x_t = x_{t-1} - \frac{\alpha_t}{M} \sum_{i=1}^{M} \nabla_x F(x_{t-1}, y_i) \tag{8}$$

by starting from an initial guess x_0. In (8), M is a fixed number of samples y_i of the random variable Y and α_t a positive learning rate. If the learning rate is suitably chosen and F is convex, x_t converges to a neighborhood of a global minimum x^* as $t \to \infty$. Convergence to the global minimum is not guaranteed when F is not convex; however, SGD is widely applied in practice, even in non-convex minimization problems, especially in machine learning algorithms.

B. Optimal policy search for output tracking

In this paper we focus on an output-tracking task, i.e., we want to learn a policy that makes the output y_t of system (2) track a reference signal r_t, using input/output data only. We look for an optimal policy minimizing the quadratic stage cost

$$\rho(x_t, r_t, \Delta u_t) = \|C x_{t+1} - r_t\|_{Q_y}^2 + \|\Delta u_t\|_R^2 \tag{9}$$

where x_t is defined in (3) and $\Delta u_t = u_t - u_{t-1}$ input increment, that we treat as the new control variable. Matrix C is such that $y_t = Cx_t$. Matrix $Q_y = Q_y' \succeq 0$ weights the tracking error while $R = R' \succ 0$ tracks the control effort. For offset-free tracking of constant set-points in steady-state, we take into account the integral term $q_{t+1} = q_t + (y_{t+1} - r_t)$ and we add a penalty $\|q_{t+1}\|^2_{Q_q}$ to the stage cost (9), weighted by a tuning parameter $Q_q = Q_q' \succeq 0$[1]. Accordingly, we set

$$s_t = \begin{bmatrix} x_t \\ q_t \end{bmatrix}, \quad p_t = r_t. \tag{10}$$

In this paper we focus on the simplest case of a linear policy parametrization π_K

$$\pi_K(s_t, r_t) = -K^s s_t - K^r r_t \tag{11}$$

although the approach could be immediately generalized to other parameterizations.

The application of SGD to the output-tracking problem formulated in (7) requires at each step t a number N_s of sample values s_0^z of the initial state, $z = 1, \ldots, N_s$, N_r samples of the reference trajectory $\{r_l^k\}_{l=0}^{L-1}$, $k = 1, \ldots N_r$, and N_d samples of the disturbance trajectory $\{d_l^h\}_{l=0}^{L-1}$, $h = 1, \ldots N_d$. In particular, we choose the samples x_0^i of the initial state from past states recorded during the evolution of the plant, that we store in the dataset

$$X_t = \{x_0, x_1, \ldots, x_t\} \tag{12}$$

of states collected until time t[2]. At each time t we select N_0 vectors randomly from X_t. The reason for this choice is that we want to pick states for which we know how to estimate the gradient, using the local model to approximate (1). Different strategies might be applied here as well.
Then, we combine the initial states x_0^i with N_q randomly sampled values q_0^j for the initial integral action value, to construct $N_s = N_0 \times N_q$ samples $s_0^{i,j}$. The full sampling procedure is described in detail in Section III-B.

For each sample $(s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})$ the SGD update (8) requires calculating $\nabla_K J_L(K_{t-1}, s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})$, which in turn requires knowing the dynamic relation (1). For the sole purpose of estimating the gradient $\nabla_K J_L$ we approximate (1) via the linear model

$$\begin{aligned} s_{l+1} = \mathcal{A}(s_0^{i,j}, p_l^k, d_l^h) s_l \;&+\; \mathcal{B}(s_0^{i,j}, p_l^k, d_l^h) u_l \\ &+\; \mathcal{H}(s_0^{i,j}, p_l^k, d_l^h) \end{aligned} \tag{13}$$

In (13) $\mathcal{A}$, $\mathcal{B}$, $\mathcal{H}$ are matrices defining a local linear model that is valid around $s_0^{i,j}$. According to our data-driven approach, the linearized model in (13) is estimated by recursive systems identification, as we will detail in Section III-A. Finally, the direction followed by SGD is

$$\mathcal{D}(K) = \frac{1}{N} \sum_{i=1}^{N_s} \sum_{k=1}^{N_r} \sum_{h=1}^{N_d} \nabla_K \hat{J}_L(K, s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1}) \tag{14}$$

where $\hat{J}_L$ approximates J_L in accordance with the local model (13) associated with $s_0^{i,j}$ and $N = N_s \cdot N_r \cdot N_d$.

III. POLICY SEARCH ALGORITHM

Starting from an initial policy parameterization K_0, we use SGD steps as in (8) to attempt finding a solution K^* of (7). While searching for the optimal policy, we update the local linear model (13) recursively to estimate the gradient $\nabla_K J_L$, as described in Section III-A. In addition, we consider two possible settings: an *offline* setting, in which π_{K_t} is not applied to the process because new experimental data cannot be collected, and an *online* setting, in which new data can be collected. In the latter case, after updating the policy from K_{t-1} to K_t as in (19) we will check if it stabilizes the local linear plant (16) at time t before applying it to the process, as described in Section III-C. A summary of the overall method is given in Algorithm 1.

A. Update of the local linear model

We get a local linear approximation (13)

$$y_t = \Theta_t \cdot [y_{t-1}' \;\ldots\; y_{t-n_o}' \; u_{t-1}' \;\ldots\; u_{t-n_i-1}']' + d_t \tag{15}$$

of the (unknown) system dynamics (2), with $\Theta_t \in \mathbb{R}^{n_y \times n_x}$, by Kalman filtering (KF) [3], assuming that $\Theta_{t+1} = \Theta_t + \xi_t$ where ξ_t is a Gaussian white noise with covariance Q_k and d_t in (15) is a a Gaussian white noise with covariance matrix R_k.

Considering the input/output form with x_t as in (3), (15) can be rewritten in terms of the input increments $\Delta u_{t-1} = u_{t-1} - u_{t-2}$ as $y_t = \Theta_t^x \cdot x_{t-1} + \Theta_t^{\delta u} \cdot \Delta u_{t-1} + d_t$, where Θ_t^x and $\Theta_t^{\delta u}$ are easily obtained from Θ_t.

By taking into account also the integral action, we finally set s_t, p_t as in (10) and obtain the linear model approximation

$$\begin{cases} s_{l+1} &= A_t s_l + B_t \Delta u_l + E p_l + D d_l \\ y_l &= C s_l \end{cases} \tag{16}$$

where $A_t = \begin{bmatrix} A & 0 \\ CA & I \end{bmatrix}$, $B_t = \begin{bmatrix} B \\ CB \end{bmatrix}$, $E = \begin{bmatrix} 0 \\ -I \end{bmatrix}$, $D = \begin{bmatrix} I \\ 0 \end{bmatrix}$ and $A = \begin{bmatrix} \Theta_t^x \\ \bar{A} \end{bmatrix}$, $B = \begin{bmatrix} \Theta_t^{\delta u} \\ 0 \end{bmatrix}$, $C = [\, I \; 0 \; 0 \,]$, being $\bar{A}$ a fixed matrix containing entries equal to 0 or 1 where appropriate.

B. Gradient estimation and policy update

For each sample $(s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})$, $i = 1, \ldots, N_0$, $j = 1, \ldots, N_q$, $k = 1, \ldots N_r$, $h = 1, \ldots N_d$, the SGD update (8) requires calculating $\nabla_K J_L(K_{t-1}, s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})$.
In order to construct the $N_s = N_0 \times N_q$ samples $s_0^{i,j}$, we consider the dataset X_t defined in (12) of past and current states collected from the plant during the learning phase and denote by $x_{\sigma_t(i)}$ the ith sample chosen from X_t, where $\sigma_t(1), \ldots, \sigma_t(N_0)$ are integers randomly selected between 0 and t. In order to explore the neighborhood of the trajectories generated by the plant, each state $x_{\sigma_t(i)}$ is perturbed by a (small) zero-mean noise v_i, whose components are bounded in $[-v_{\max}, v_{\max}]$. Regarding the components of the integrator q_t of the output tracking error, we sample N_q samples

[1]A penalty on u_t might be easily introduced as well in (9), such as $\|u_t - u_t^r\|^2_{Q_u}$.

[2]To limit the memory occupancy of X_t, in a practical application we can maintain in X_t only a subset of representative states.

[3]Recursive least squares with forgetting factor could be used here too.

q_0^j from a normally distributed random variable with mean zero and variance σ_q^2. Hence, we set vector $s_0^{i,j}$ as

$$s_0^{i,j} = \begin{bmatrix} x_{\sigma_t(i)} + v_i \\ q_0^j \end{bmatrix} \quad i = 1, \dots, N_0,\ j = 1, \dots, N_q \quad (17)$$

Regarding the reference and disturbance trajectories, we assume that each reference trajectory is constant between 0 and $L-1$, that is

$$r_l^k \equiv r^k,\ l = 0, \dots, L-1, \quad k = 1, \dots, N_r \quad (18)$$

where the constant reference values r^k are chosen randomly from the interval $[r_{\min}, r_{\max}]$ of references of interest. The samples d_l^h are randomly generated for $l = 0, \dots, L-1$, $h = 1, \dots, N_d$ from a given box $[-d_{\max}, d_{\max}]$).

Note that when $(s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})$ are drawn in accordance with given (possibly non-uniform) probability distributions, (14) can be easily extended to include probabilities of samples.

The analytic gradient of $\hat{J}_L(K_{t-1}, s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})$ for each sample is then evaluated using the local linear model $(A_{\sigma_t(i)}, B_{\sigma_t(i)})$ estimated at the time instant $\sigma_t(i)$ in which the state $x_{\sigma_t(i)}$ was visited by the plant. The idea is that such a model represents the local behavior of system (1) in a neighborhood of $x_{\sigma_t(i)}$. The model is then kept constant along the L steps of the trajectory represented in (6). Then, for each given initial condition $s_0^{i,j}$, the corresponding subsequent states $s_{l+1}^{i,j}$ under the local closed-loop linear dynamics (16), (11) are given by $s_{l+1}^{i,j} = A_{ti}^{l+1} s_0^{i,j} + \sum_{n=0}^{l} A_{ti}^h (E_{ti} r_{l-n}^k + D d_{l-n}^h)$, where $A_{ti} = A_{\sigma_t(i)} - B_{\sigma_t(i)} K_{t-1}^s$, $E_{ti} = E - B_{\sigma_t(i)} K_{t-1}^r$ which are combined with (6) and (9) to obtain $\hat{J}_L$.

The average of $\{\nabla_K \hat{J}_L(K_{t-1}, s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})\}_{i,j,k,h}$ provide $\mathcal{D}(K_{t-1})$ as in (14). The policy update step performed by the SGD algorithm with decreasing learning rate α_t is

$$K_t = K_{t-1} - \alpha_t \mathcal{D}(K_{t-1}) \quad (19)$$

SGD has a solid theoretical background and a consistent amount of examples of successful application. However it is known to be characterized by a slow convergence rate. Thus, in the numerical examples reported in Section IV we will use the AMSGrad method [16], a faster variant of SGD[4].

C. Stability check and policy implementation

In an *online* setting, after updating the policy K_t as in (19) we want to check if it stabilizes the local linear plant (16) at time t before applying it to the process. This is done by computing the dominant eigenvalue λ_t of the matrix A characterizing the current linear closed-loop model, obtained by applying K_t to the local linear model (16). Although there is no guarantee that K_t will also stabilize the underlying plant, we decide to apply $K_t^{s\star} = K_t$ to the plant only if $|\lambda_t| < 1$.

In case K_t is not stabilizing (16), we solve the following semidefinite program

$$\begin{aligned} \min_{Y,P,W} \quad & \|Y - K_t^s P\|_2^2 \\ \text{s.t.} \quad & \begin{bmatrix} P & P & M' \\ P & W & 0 \\ M & 0 & P \end{bmatrix} \succeq 0 \\ & \text{with } M = A_t P - B_t Y \\ & W \succ 0, \quad P \succ 0 \end{aligned} \quad (20)$$

The feedback gain $K_t^{s\star} = Y^\star (P^\star)^{-1}$ obtained from the solution $Y^\star, P^\star, W^\star$ of (20) is the gain closest to K_t^s that is asymptotically stabilizing (A_t, B_t), as proved for instance in [18]. The gain $K_t^{s\star}$ is applied to the plant instead of the gain K_t^s obtained from the SGD update (8). In conclusion, the input u_t applied to the plant at time t is $u_t = u_{t-1} - K_t^{s\star} s_t - K_t^r r_t$.

Algorithm 1 Optimal policy search

Input: Initial policy K_0 and model Θ_0, number N_{learn} of steps, training reference $\{r_t \mid t = 0, \dots, N_{\text{learn}}\}$.
Output: Final policy K_{SGD}.

```
1:  for $t = 1, 2, ..., N_{\text{learn}}$ do
2:      Recursively update linear model $\Theta_t$;
3:      for $i = 1, 2, ..., N_0$ do
4:          sample $x_{\sigma_t(i)}$ from state history $X_t$;
5:          for $j = 1, 2, ..., N_q$ do
6:              sample $q_0^j$ and build $s_0^{i,j}$ as in (17);
7:              retrieve model coefficients $\Theta_{\sigma_t(i)}$;
8:              for $k = 1, 2, ..., N_r$ do
9:                  get random state trajectory $\{r_l^k\}_{l=0}^{L-1}$;
10:                 for $h = 1, 2, ..., N_d$ do
11:                     get random disturbance trajectory $\{d_l^h\}_{l=0}^{L-1}$;
12:                     calculate $\nabla_K \hat{J}_L(K_{t-1}, s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})$;
13:                 end for
14:             end for
15:         end for
16:         $\mathcal{D}(K_{t-1}) \leftarrow \frac{\sum_{i,j,k,h} \nabla_K \hat{J}_L(K_{t-1}, s_0^{i,j}, \{r_l^k, d_l^h\}_{l=0}^{L-1})}{N_s N_r N_d}$;
17:         Policy update: $K_t \leftarrow K_{t-1} - \alpha_t \mathcal{D}(K_{t-1})$;
18:         online setting: apply $u_t \leftarrow u_{t-1} - K_t^{s\star} s_t - K_t^r r_t$;
                acquire $x_{t+1}$;
            offline setting: retrive $u_t, x_{t+1}$ from dataset;
19:     end for
20: end for
21: $K_{\text{SGD}} \leftarrow K_{N_{\text{learn}}}$;
22: end.
```

IV. NUMERICAL RESULTS

We test the proposed policy search algorithm when applied to a linear time-invariant (LTI) system and to a nonlinear system. In the nonlinear case, we show the results of applying the algorithm in an online setting. The policy update is performed in both examples using the AMSGrad algorithm [16] tuned using the parameters $\alpha = 0.1$ for the LTI example and $\alpha = 1$ for the nonlinear example, weight on the 1st moment $\beta_1 = 0.9$, and weight on the 2nd moment $\beta = 0.999$.

[4]The adaptive moment estimation (Adam) algorithm [17] could be also used in alternative.

TABLE I

POLICY SEARCH PARAMETERS-LTI

n_y	n_u	n_i	n_o	Q_k	R_k	L
1	1	2	3	$10^{-3}I$	0.1	20
N_0	N_r	N_q	$r_{\min}$	$r_{\max}$	σ_q^2	$v_{\max}$
50	1	10	-10^3	10^3	100	10^6

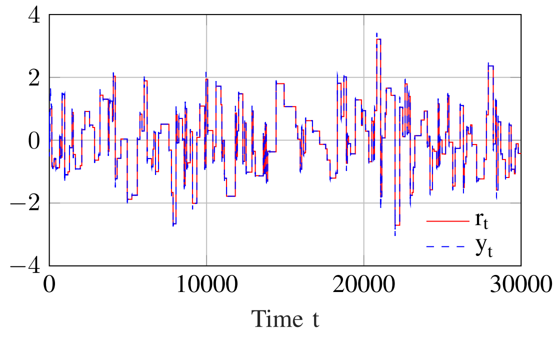

Fig. 1. LTI example: tracking task performed in learning. Output y_t (dashed blue line), training reference r_t (red line).

The initial value Θ_0 of the model is $\Theta_0 = 0$.

A. Example 1: LTI case

Let the system generating the data be the (unknown) single-input single-output (SISO) LTI system

$$\begin{cases} x_{t+1} &= \begin{bmatrix} -0.669 & 0.378 & 0.233 \\ -0.288 & -0.147 & -0.638 \\ -0.337 & 0.589 & 0.043 \end{bmatrix} x_t + \begin{bmatrix} -0.295 \\ -0.325 \\ -0.258 \end{bmatrix} u_t \\ y_t &= \begin{bmatrix} -1.139 & 0.319 & -0.571 \end{bmatrix} x_t \end{cases} \tag{21}$$

We assume that there is no disturbance d_t affecting the model and choose weights $Q_y = 1$, $R = 0.1$, and $Q_q = 1$ in (9). The parameters of the model, of the KF, and of the cost function approximation are reported in Table I.

Starting from an initial policy $K_0 = [1 \dots 1]$, the learning procedure is executed for $N_{\text{learn}} = 30000$ iterations.

The reference signal r_t used for policy learning is piecewise constant, see Figure 1 (red line). The figure also reports the output y_t of the plant, to show the tracking performance during training. After N_{learn} steps the obtained policy is

$$K_{\text{SGD}} = [-1.255, 0.218, 0.652, 0.895, 0.050, 1.115, -2.186]$$

The optimal policy computed using the real system (21) as in (16) for a constant reference r_t, coherently with (18), is

$$K_{\text{opt}} = [-1.257, 0.219, 0.653, 0.898, 0.050, 1.141, -2.196]$$

The convergence of K_t to K_{opt} during the learning phase is shown in Figure 2 in terms of the Euclidean norm of the difference between K_t and K_{opt}.

B. Example 2: Nonlinear case

We want to learn an optimal linear policy for the classical Continuous Stirred Tank Reactor (CSTR) benchmark problem [19], which has nonlinear dynamics. All the simulations are performed by sampling the continuous-time model of the

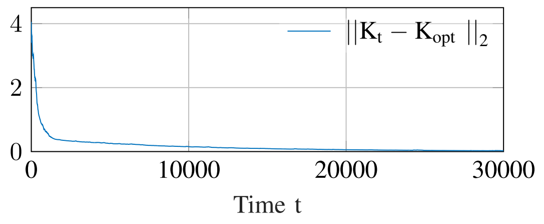

Fig. 2. LTI example: evolution of the error $\|K_t - K_{opt}\|_2$ at every instant of the learning phase.

TABLE II

POLICY SEARCH PARAMETERS-CSTR

n_y	n_u	n_i	n_o	Q_k	R_k	L
2	1	1	3	I	0.1	10
N_0	N_r	N_q	$r_{\min}$	$r_{\max}$	σ_q^2	$v_{\max}$
50	20	20	8.6	9.5	100	1

CSTR system reported in the MPC Toolbox for MATLAB [20], with sampling time $T_s = 6$ s.

The state $\bar{x}_t$ of the nonlinear system is composed by the concentration $\bar{x}_t^1$ of the reagent inside the tank and the temperature $\bar{x}_t^2$. Both signals are measured and their measurements are subject to additive Gaussian white noise with standard deviation $n_c = 0.01$ units, respectively. The concentration and the temperature of the inlet feed stream are kept constant and equal to $v_1 = 10$ kg mol/m^3 and $v_2 = 298.15$ K, respectively.

The control objective is to optimally make $\bar{y}_t = \bar{x}_t^1$ track a desired set point $\bar{r}_t$ by manipulating the temperature $\bar{u}_t$ of the jacket coolant. We take $\bar{u}_{\text{off}} = 300$ K, $\bar{x}_{\text{off}}^1 = 8.45$ kg mol/m^3 and $\bar{x}_{\text{off}}^2 = 312.74$ K as the operating point. Let $y_t = [\,(\bar{x}_t^1 - \bar{x}_{\text{off}}^1)\ (\bar{x}_t^2 - \bar{x}_{\text{off}}^2)\,]'$, $u_t = \bar{u}_t - \bar{u}_{\text{off}}$, and $r_t = \bar{r}_t - \bar{y}_{\text{off}}$. The stage-cost weights are

$$Q_y = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right], \quad R = 0.1, \quad Q_q = \left[\begin{smallmatrix} 0.01 & 0 \\ 0 & 0 \end{smallmatrix}\right] \tag{22}$$

The linear model (15) is taken as a local model, without taking into account the disturbance ($d_t = 0$). Table II contains the policy-search parameters used in this example.

The learning phase is performed in $N_{\text{learn}} = 10000$ steps, starting from an initial policy K_0 having all components equal to 0.0001 and using the piecewise constant reference r_t shown in Figure 3 (red line), together with the concentration $\bar{y}_t$ inside the CSTR and the commanded jacket temperature u_t. The performance of the policy K_{SGD} obtained after N_{learn} iterations is compared against an unconstrained MPC controller K_{ID} with the same weights as in (22) and a large prediction and control horizon (300 steps), based on a linear model identified from the same data used to learn K_{SGD}. As for K_{SGD}, also in the synthesis of K_{ID} we assume (18).

The results of applying the two policies to the CSTR plant, performing a tracking task with a new time-varying reference signal r_t that is different from the reference used in training, for 20000 steps are shown in Figure 4. The sum of stage costs (9) accumulated by K_{SGD} during the validation task is $4.3 \cdot 10^3$ while the one related to K_{ID} is $2.4 \cdot 10^4$.

It can be seen that our controller K_{SGD} outperforms the optimal controller K_{ID}. This is not surprising, as K_{ID} is

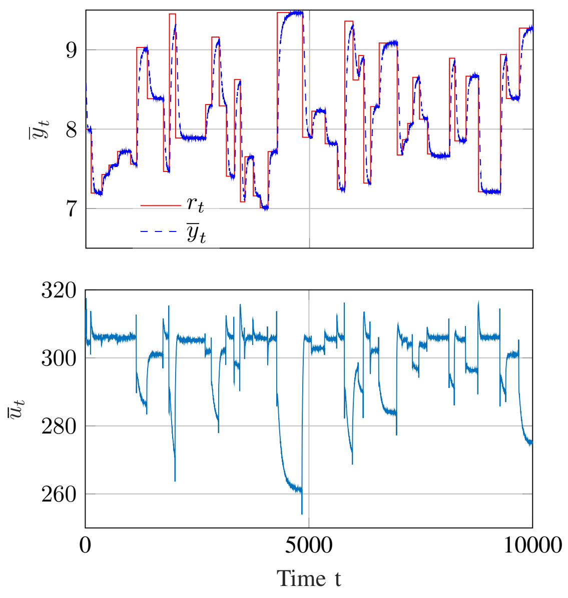

Fig. 3. CSTR example: tracking task performed in learning. Upper plot: output $\bar{y}_t$ (dashed blue line), reference $\bar{r}_t$ used in learning (red-line). Bottom plot: temperature of the jacket coolant $\bar{u}_t$.

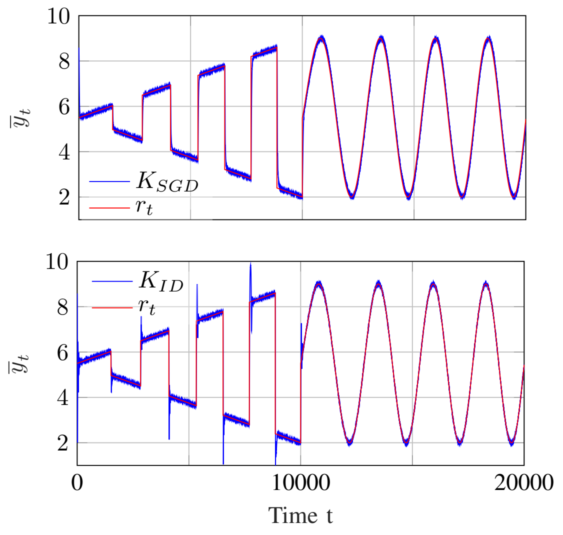

Fig. 4. CSTR online example: tracking task performed in validation.

optimal with respect to the best linear model that can be identified from the dataset, not to the actual underlying nonlinear CSTR model.

V. CONCLUSIONS

We have presented a policy-search method to synthesize optimal control laws directly from data by using stochastic gradient descent iterations. A linear process model is recursively identified with a forgetting factor for the only purpose of getting a local linear model to compute gradients.

The reported examples show that the method is able to converge to the optimal feedback law when the system generating the data belongs to the class of models chosen for the recursive identification. Moreover, if the latter assumption is not satisfied, the method seems to outperform what can be achieved by open-loop identification cascaded by model-based optimal control design.

Compared to reinforcement learning and other policy search methods, an advantage of our approach is that the design of the experiment required to collect data is relatively easy, due to the fact that the exploration of the state space is simply delegated to the reference signal used during the learning phase, and in particular no artificial noise must be added on input or output signals.

Current research is devoted to extending the approach in several directions, including more general parameterizations on the control policy and the investigation of conditions for guaranteeing convergence of the stochastic gradient descent method to a fixed (possibly not globally optimal) policy.

REFERENCES

[1] D. Piga, S. Formentin, and A. Bemporad, “Direct data-driven control of constrained systems,” *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1422–1429, Jul 2018.

[2] D. Selvi, D. Piga, and A. Bemporad, “Towards direct data-driven control design of optimal controllers,” in *Proc. European Control Conference*, Limassol, Cyprus, 2018.

[3] J. R. Salvador, D. Muñoz de la Peña, T. Alamo, and A. Bemporad, “Data-based predictive control via direct weight optimization,” in *6th IFAC Conference on Nonlinear Model Predictive Control, pp. 437-442*, Madison, WI, USA, 2018.

[4] B. Recht, “A Tour of Reinforcement Learning: The View from Continuous Control,” 2018.

[5] V. R. Konda and J. N. Tsitsiklis, “On actor-critic algorithms,” *SIAM Journal of Optimal Control*, vol. 42, pp. 1143–1166, 2003.

[6] C. J. C. H. Watkins and P. Dyan, “Q-learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[7] S. J. Bradtke, B. E. Ydstie, and A. G. Barto, “Adaptive linear quadratic control using policy iteration,” in *Proc. American Control Conference*, 1994.

[8] I. Grondman, “Online model learning algorithms for actor-critic control,” PhD dissertation, Delft Center for Systems and Control, 2015.

[9] M. Zanon, S. Gros, and A. Bemporad, “Practical reinforcement learning of stabilizing economic MPC,” in *Proc. European Control Conference*, Naples, Italy, 2019.

[10] M. P. Desienroth, G. Neumann, and J. Peters, “A survey on policy search for robotics,” *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2011.

[11] T. P. Lillicrap, J. H. Jonathan, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” 2015.

[12] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist Reinforcement Learning,” *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992.

[13] J. Peters and S. Schaal, “Reinforcement Learning of motor skills with policy gradients,” *Neural Networks*, vol. 21, no. 4, pp. 682–697, May 2008.

[14] M. Fazel, R. Ge, S. M. Kakade, and M. Mesbahi, “Global convergence of policy gradient methods for linearized control problems,” 2018.

[15] H. Robbins and S. Monoro, “A stochastic approximation method,” *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[16] S. Reddi and S. Kumar, “On the convergence of Adam and beyond,” in *Proc. International Conference on Learning Representation*, Vancouver, CA, USA, April 30th-May 3rd 2018.

[17] D. P. Kingma and J. L. Ba, “Adam: a method for stochastic optimization,” in *Proc. International Conference on Learning Representation*, San Diego, CA, USA, May 7-9 2015.

[18] D. Bernardini and A. Bemporad, “Stabilizing model predictive control of stochastic constrained linear systems,” *IEEE Transactions on Automatic Control*, vol. 57, no. 6, Jun 2012.

[19] D. Seborg, T. F. Edgar, and D. A. Mellichamp, *Process Dynamics and Control, 2nd ed.* Wiley, 2004.

[20] A. Bemporad, M. Morari, and N. Ricker, *Model Predictive Control Toolbox for MATLAB*. The Mathworks, Inc., 2016, http://www.mathworks.com/access/helpdesk/help/toolbox/mpc/.