Fixed-final-time Optimal Tracking Control of Input-affine Nonlinear Systems

Ali Heydari¹ and S. N. Balakrishnan²

Abstract
In this study, approximate dynamics programming framework is utilized for solving the Bellman equation related to the fixed-final-time optimal tracking problem of input-affine nonlinear systems. Convergence of the weights of the neurocontroller in the proposed successive approximation based algorithm is provided and the network is trained to provide the optimal solution to the problems with a) unspecified initial conditions b) different time horizons, and c) different reference trajectories under certain general conditions. Numerical simulations illustrate the versatility of the proposed neurocontroller.

I. Introduction
Approximate dynamics programming (ADP) has shown a lot of promise in solving optimal control problems with neural networks (NN) as the enabling structure [1-10]. Mechanism for ADP is usually provided through a dual network architecture called the Adaptive Critics (AC) [3,2]. In the heuristic dynamic programming (HDP) class with ACs, one network, called the ‘critic’ network, maps the input states to output the cost and another network, called the ‘action’ network, outputs the control with states of the system as its inputs [4,5]. In the dual heuristic programming (DHP) formulation, while the action network remains the same as the HDP, the critic network outputs the costates with the current states as inputs [2,6,7]. The convergence proof of DHP for linear systems is presented in [8] and that of HDP for general case is presented in [4]. The Single Network Adaptive Critics (SNAC) architecture developed in [9] is shown to be able to eliminate the need for the second network and perform DHP using only one network. This process results in a considerable decrease in the offline training effort and the resulting simplicity makes it attractive for online implementation requiring less computational resources and storage memory. Similarly, the cost function based SNAC (J-SNAC) eliminates the need for the action network in an HDP scheme [10]. While [2-10] deal with discrete-time systems, some researchers have recently focused on continuous time problems [11-13].

Note that these developments in the neural network (NN) literature have mainly addressed only the infinite horizon or regulator type problems. Finite-horizon optimal control is relatively more difficult due to the time varying Hamilton-Jacobi-Bellman (HJB) equation resulting in a time-to-go dependent optimal cost function and costates. Using numerical methods [14] a two-point boundary value problem (TPBVP) needs to be solved for each set of initial condition for a given final time and it will provide only an open loop solution. The control loop can be closed using techniques like Model Predictive Control (MPC) as done in [15], however, the result will be valid only for one set of initial conditions and final time. This limitation holds for the method developed in [16] also. Ref. [17] develops a dynamics optimization scheme which gives an open-loop solution, then, optimal tracking is used for rejecting the online perturbation and deviations from the optimal trajectory. Authors of [18] used power series to solve the problem with small nonlinearities, and in [19] an approximated solution is given through the so-called Finite-horizon State Dependent Riccati Equation (Finite-SDRE) method.

Neural networks are used for solving finite-horizon optimal control problem in [20-25]. Authors of [20] developed a neurocontroller for a scalar problem with terminal constraint using AC. Continuous-time problems are considered in [21] and [22] where the time-dependent weights are calculated through a backward integration. The finite-horizon problem with unspecified terminal time and a fixed terminal state is considered in [23,24]. The neurocontroller developed in [23] can work only with one set of initial conditions and if the initial state is changed, the network needs to be re-trained to give the optimal solution for the new state. This limitation holds for [24] as well.

This work was partially support by a grant from the National Science Foundation.
¹South Dakota School of Mines and Technology, email: ali.heydari@sdsmt.edu.
²Missouri University of Science and Technology.
In many practical systems one is interested in tracking a desired signal. Examples of such systems are contour tracking in machining processes [34-35] and control of robotic manipulators [36]. In some systems the tracking is required to be carried out in a given time, see [37] for an example of such a case in an autopilot design. The constraint of final time being fixed makes the problem very difficult to solve. Missile guidance problems and launch vehicle problems are some other applications in this class of problems. Solving optimal tracking problems for nonlinear systems using adaptive critics has been investigated by researchers in [26-32]. In [26] the authors have developed a tracking controller for the system whose input gain matrix is invertible. In [27] the reference signal is limited to those which satisfy the dynamics of the system. Developments in [28-31] solve the tracking problem for the systems of nonlinear Brunovsky canonical form. Finally, the finite-horizon tracking neurocontroller developed in [32] can control only one set of initial conditions and requires the input gain matrix of the dynamics to be invertible.

In this paper, a single neural network based solution, called Finite-horizon Single Network Adaptive Critics (Finite-SNAC), is developed which embeds an approximate solution to the discrete-time HJB equation for fixed-final-time optimal tracking problems. The approximation can be made as accurate as desired using rich enough basis functions. Consequently, the offline trained network can be used to generate online feedback control in real-time. The neurocontroller is able to solve optimal tracking problem of general nonlinear control-affine dynamics for tracking either a given arbitrary trajectory or a family of trajectories which share the same, possibly nonlinear, dynamics. Once the network is trained, it will give optimal solution for every initial condition as long as the resulting trajectory lies on the domain for which the network is trained, hence, Finite-SNAC does not have the restrictions of some of the cited references in the field. Furthermore, a major advantage of the proposed technique is that this network provides optimal feedback solutions to any different final time as long as it is less than the final time for which the network is synthesized. An innovative proof is developed which shows the successive approximation based training algorithm is a contraction mapping [33].

Comparing the developed controller in this paper with the available intelligent controllers in the literature, the closest ones are [20,25]. As compared to [20], in this study, only one network is needed for computing the control, and this idea has been generalized to tracking with free final state. Moreover, convergence proofs are provided. The differences between this study and [25] are a) solving tracking problem versus the problem of bringing the states to zero in [25] b) using time varying weights for the neural networks as opposed to the time invariant weights in that reference, c) developing a ‘backward in time’ training algorithm versus the pure ‘iterative’ algorithm in [25], and d) providing a completely different convergence proof in this study. The advantages of the development in this study versus the available finite-horizon optimal tracking methods in the literature [32,37] are providing solutions for different initial conditions and different final-time, without needing to retrain the network as in [32], or needing to recalculate the series of differential Riccati equation till they converge as in [37]. Moreover, the restriction of requiring an invertible input-gain matrix in [32] does not exist here. Finally the advantage of this study versus the MPC approach utilized in [15] for optimal tracking is having a negligible computational load in here versus the huge real-time computational load in MPC for online numerical calculation of the optimal solution at each instant, as detailed in [15]. In here, however, once the networks are trained offline, the online calculation of the control is as simple as a feeding the states to the network to get the costate vector and hence, the control.

The rest of the paper is organized as follows: Finite-SNAC is developed in section II. Relevant convergence theorems are presented in section III. A modified version of the controller for higher versatility is proposed in section IV, and the numerical results and analyses are presented in Section V. Finally, the conclusions are given in Section VI.

II. Theory of Finite-SNAC

Consider the nonlinear continuous-time input-affine system
\[ \dot{x}(t) = f_c(x(t)) + g_c(x(t))u(t), \]
where \( x(t) \in \mathbb{R}^n \) and \( u(t) \in \mathbb{R}^l \) denote the state and the control vectors at time \( t \), respectively, and parameters \( n \) and \( l \) are the dimension of the state and control vectors. Smooth functions \( f_c: \mathbb{R}^n \to \mathbb{R}^n \) and \( g_c: \mathbb{R}^n \to \mathbb{R}^{nxl} \) are the system dynamics and the initial states are given by \( x(0) \). Given reference signal \( r(t) \in \mathbb{R}^n \) for \( t \in [0 \ t_f] \), where the initial time is selected as zero and the final time is denoted by \( t_f \), the objective is selecting a control history \( u(t) \), \( t \in [0 \ t_f] \), such that the cost function given below is minimized.

\[ J = \frac{1}{2}(x(t_f) - r(t_f))^T S(x(t_f) - r(t_f)) + \frac{1}{2} \int_0^{t_f} \left( (x(t) - r(t))^T Q_c(x(t) - r(t)) + u(t)^T R_c u(t) \right) dt. \]
Symmetric matrices $S \in \mathbb{R}^{n \times n}$, $Q_k \in \mathbb{R}^{n \times n}$ and $R_c \in \mathbb{R}^{k \times k}$ are the penalizing matrices for the final states, states, and control vectors, respectively. Matrices $S$ and $Q_k$ are positive semi-definite and matrix $R_c$ is a positive definite matrix. Superscript $T$ denotes transpose operation.

In many practical applications, discretization is used since the states are estimated and the control is calculated at discrete times and not calculated continuously, though the description of dynamics is continuous. The approach selected in this paper is discretizing the problem using a small sampling time $\Delta t$ to have

$$ x_{k+1} = f(x_k) + g(x_k)u_k, \ k \in \{0,1, ..., N-1\}, \tag{2} $$

$$ J = \frac{1}{2} (x_N - r_N)^T S (x_N - r_N) + \frac{1}{2} \sum_{\Delta}^N (x_k - r_k)^T Q (x_k - r_k) + u_k^T R u_k, \tag{3} $$

where integer $k$ denotes the time index, $N \equiv t_f / \Delta t$, $x_k \equiv x(k \Delta t)$, $u_k \equiv u(k \Delta t)$, and $r_k \equiv r(k \Delta t)$. Using Euler integration scheme one has $f(x_k) \equiv x_k + \Delta t f_c(x_k)$, $g(x_k) \equiv \Delta t g_c(x_k)$, $Q \equiv \Delta t Q_c$, and $R \equiv \Delta t R_c$. \textbf{Remark 1}: The assumption that discrete-time system (2) is obtained through ‘discretizing’ a continuous dynamics is utilized in convergence analysis of the developed algorithms in this paper. Excluding systems with inherent discrete evolution, including hybrid systems, all the physical systems are continuous; therefore, this assumption does not impose a limitation on the results obtained here for such systems.

The cost-to-go at each time step $k$ and state vector $x_k$, denoted by $J(x_k, k)$, is equal to

$$ J(x_k, k) = \frac{1}{2} (x_k - r_k)^T Q (x_k - r_k) + u_k^T R u_k + J(x_{k+1}, k+1), \ k \in \{0,1, ..., N-1\}, \tag{4} $$

which leads to the recursive equation

$$ J(x_k, k) = \frac{1}{2} (x_k - r_k)^T Q (x_k - r_k) + u_k^T R u_k + J(x_{k+1}, k+1), \ k \in \{0,1, ..., N-1\}. $$

The discrete-time HJB equation [14], or Bellman equation, can be derived from the abovementioned recursive equation as

$$ J^*(x_k, k) = \min_{u_k} \left\{ \frac{1}{2} (x_k - r_k)^T Q (x_k - r_k) + u_k^T R u_k + J(x_{k+1}, k+1) \right\}, \ k \in \{0,1, ..., N-1\}, $$

where $J^*(x_k, k)$ denotes the optimal cost-to-go. Defining the costate vector at time step $k$ as $\lambda_k \equiv \partial J(x_k, k) / \partial x_k$ and taking the derivative of (4) leads to

$$ \lambda_k = Q (x_k - r_k) + A_k^T \lambda_{k+1}, \ k \in \{0,1, ..., N-1\}, \tag{5} $$

or equivalently

$$ \lambda_{k+1} = Q (x_{k+1} - r_{k+1}) + A_{k+1}^T \lambda_{k+2}, \ k \in \{0,1, ..., N-2\}, \tag{6} $$

where $A_{k+1} \equiv \frac{\partial f(x_{k+1})}{\partial x_{k+1}}$. Note that

$$ J(x_N, N) = \frac{1}{2} (x_N - r_N)^T S (x_N - r_N), $$

hence,

$$ \lambda_N = S (x_N - r_N). \tag{7} $$

The optimal control $u_k$ is such that it satisfies the optimality condition $\partial J(x_k, k) / \partial u_k = 0$. Considering $J(x_k, k)$ given in (4), the optimal control which minimizes $J(x_k, k)$ is calculated as

$$ u_k^* = -R^{-1} g(x_k)^T \lambda_{k+1}, \ k \in \{0,1, ..., N-1\}, \tag{8} $$

where the costate vector satisfies recursive equation (6) subject to final condition (7). Therefore, having optimal $\lambda_{k+1}$ in a closed form, i.e., as a function of the current state and current time, the closed-form optimal control can be calculated. A single NN called Finite-SNAC in the form given below is suggested to output the desired costate vector based on the state vector and the time step, which are fed to the network.

$$ \lambda_{k+1} = W_k^\phi (x_k), \ k \in \{0,1, ..., N-1\}, \tag{9} $$

where $W_k$ denotes the network time step dependent weight matrix. The selected neural network is linear in the weights [38] where $\phi: \mathbb{R}^m \rightarrow \mathbb{R}^m$ is composed of $m$ smooth linearly independent scalar basis functions and $W_k \in \mathbb{R}^{m \times n}$, $k \in \{0,1, ..., N-1\}$. 
Remark 2: As seen in (8), the optimal control at time step \( k \) depends on the costate vector at the next time step, i.e., \( u_k \) is a function of \( \lambda_{k+1} \). Moreover, \( \lambda_{k+1} \) is a function of \( \lambda_k \) and hence \( x_k \). Therefore, there is a mapping between \( x_k \) and \( \lambda_{k+1} \). In this study, Finite-SNAC is used to capture this relation, i.e., the mapping between \( x_k \) and \( \lambda_{k+1} \).

Using costate equation (6) and final condition (7), the network training target, denoted by \( \lambda^T \), can be calculated using the following two equations

\[
\lambda^T_N = S(x_N - r_N),
\]

\[
\lambda^T_{k+1} = Q(x_{k+1} - r_{k+1}) + A^T_{k+1}\lambda_{k+2}, \quad k \in \{0,1, ..., N-2\}
\]

which in the training process, \( \lambda_{k+2} \) on the right hand side of (11) will be substituted by \( W^T_{k+1} \phi(x_{k+1}) \). Noting that the closed loop dynamics, using (2), (8), and (9), is given by

\[
x_{k+1} = f(x_k) - g(x_k)R^{-1}g(x_k)^TW^T_k \phi(x_k),
\]

hence, equations (10) and (11), respectively, change to

\[
\lambda^T_k = S(f(x_{k-1}) - g(x_{k-1})R^{-1}g(x_{k-1})^TW^T_{k-1} \phi(x_{k-1}) - r_k)
\]

\[
\lambda^T_{k+1} = Q(f(x_k) - g(x_k)R^{-1}g(x_k)^TW^T_k \phi(x_k) - r_{k+1}) + A^T_{k+1}W^T_{k+1} \phi(f(x_k) - g(x_k)R^{-1}g(x_k)^TW^T_k \phi(x_k)), \quad k \in \{0,1, ..., N-2\}
\]

Note that, for example in (13) which is supposed to generate \( \lambda^T_{k+1} \) to be used for learning \( W_k \), the right hand side of the equation is dependent on \( x_k \). Hence, the calculated target \( \lambda^T_{k+1} \) is itself a function of the weights and the optimal target needs to be obtained through a successive approximation scheme. Denoting the weights at \( k \)th iteration with \( W_k \), the successive approximation is given by

\[
W^{i+1} \phi(x_{k-1}) = S \left( f(x_{k-1}) - g(x_{k-1})R^{-1}g(x_{k-1})^TW^i_{k-1} \phi(x_{k-1}) - r_k \right)
\]

\[
W^{i+1} \phi(x_k) = Q \left( f(x_k) - g(x_k)R^{-1}g(x_k)^TW^i_k \phi(x_k) - r_{k+1} \right) + A^T_{k+1}W^{i+1} \phi(f(x_k) - g(x_k)R^{-1}g(x_k)^TW^i_k \phi(x_k)), \quad k \in \{0,1, ..., N-2\}
\]

It should be noted that the training error is a function of the selected state \( x_k \), because, the network may learn different points of the domain of interest with different accuracies. The training can be summarized as given in Algorithm 1.

Algorithm 1:
1- Select an initial guess on \( W_k^0, k \in \{0,1, ..., N-1\} \).
2- Randomly select \( x_{N-1} \in \Omega \), where \( \Omega \subset \mathbb{R}^n \) represents the domain of interest.
3- Through the process showed in Fig. 1 or given by Eq. (12), calculate \( \lambda^T_k \).
4- Train network weight \( W_{k-1} \) using input-target pair \( \{x_{N-1}, \lambda^T_N\} \).
5- Calculate training error \( e_k(x_k) \equiv \lambda^T_k - \lambda^T_{k+1} = W_k^T \phi(x_k) - \lambda^T_{k+1} \).
6- Repeat steps 2 to 5 until weights \( W_{k-1} \) converge with a preselected tolerance and \( \|e_{N-1}(x_{N-1})\| < \beta \) for different randomly selected \( x_{N-1} \)s, where \( \beta \) is a small positive real number and \( \|.\| \) denotes vector norm.
7- For \( k = N-2 \) to \( k = 0 \) repeat
8- Randomly select state vector \( x_k \in \Omega \)
9- Through the process showed in Fig. 2 or given by Eq. (13), calculate \( \lambda^f_{k+1} \).
10- Train network weight \( W_k \) using input-target pair \( \{x_k, \lambda^f_{k+1}\} \).
11- Calculate the training error \( e_k(x_k) \) using (16).
12- Repeat steps 8 to 11 until weights \( W_k \) converge with a preselected tolerance and \( \|e_k(x_k)\| < \beta \) for different randomly selected \( x_k \) s.

Fig. 1. Finite-SNAC training diagram for learning \( W_{N-1} \), using Algorithm 1.

Fig. 2. Finite-SNAC training diagram for learning \( W_k, k \in \{0, 1, ..., N - 2\} \), using Algorithm 1.

**Remark 3:** The capability of uniform approximation of neural networks [39,40] indicates that once the network is trained for a large enough number of samples distributed evenly throughout the domain of interest, the network is able to approximate the output for any new sample of the domain with a bounded approximation error. This error bound can be made arbitrarily small once the network is rich enough and the number of training samples is large enough. For the linear in weight neural network selected in this study and the polynomial basis function utilized in the numerical examples, Weierstrass approximation theorem [41] proves a similar uniform approximation capability.

**Remark 4:** The stopping criteria for the iterations given in Algorithm 1 are the convergence of the weights with a preselected tolerance ‘and’ the convergence of the training error to a small ball around the origin with arbitrarily selected radius \( \beta > 0 \), namely \( \|e_k(x_k)\| < \beta \). Theorem 1 given in Section III provides sufficient condition for satisfaction of the first criterion. The satisfaction of the second criterion, i.e., \( \|e_k(x_k)\| < \beta \), however, is guaranteed once the basis functions of the network are rich enough. This issue will be investigated in Section III. In that section it is also shown that for linear systems which the perfect basis functions are available, the approximation error vanishes, i.e., the trained weights converge to the optimal solution.

Having the input-target pair \( \{x_k, \lambda^f_{k+1}\} \) calculated, the network can be trained using any training method. The selected training law in this study is the least square method. Assume that in each iteration of Algorithm 1, instead of one random state, \( p \) random states denoted with \( x^{(i)} \), \( 1 \leq i \leq p \), are selected. Denoting the training target \( \lambda^f \) calculated using random state \( x^{(i)} \) by \( \lambda^f(x^{(i)}) \), the objective is finding \( W_k \) such that it solves...
\[ \begin{align*}
W_k^T \phi(x^{(1)}) &= \lambda^T(x^{(1)}) \\
W_k^T \phi(x^{(2)}) &= \lambda^T(x^{(2)}) \\
&\vdots \\
W_k^T \phi(x^{(p)}) &= \lambda^T(x^{(p)})
\end{align*} \] (17)

Define
\[ \begin{align*}
\phi &\equiv [\phi(x^{(1)}) \ \phi(x^{(2)}) \ \ldots \ \phi(x^{(p)})] \\
\lambda^T &\equiv [\lambda^T(x^{(1)}) \ \lambda^T(x^{(2)}) \ \ldots \ \lambda^T(x^{(p)})]
\end{align*} \]

Using the least squares, the solution to system of linear equations (17) is given by
\[ W_k = \left( \phi \phi^T \right)^{-1} \phi \lambda^T, \quad k \in \{0, 1, \ldots, N - 1\} \] (18)

Note that for the inverse of matrix \( \phi \phi^T \) to exist, one needs the basis functions \( \phi(\cdot) \) to be linearly independent and the number of random states \( p \) to be greater than or equal to the number of neurons, \( m \).

Though (18) looks like a one shot solution for the ideal NN weights at step \( k \), but, as mentioned earlier, the training is an iterative process which needs selecting different random states and updating the weights through solving (18) successively. To make it clearer, one should note that \( \lambda^T \) used in the weight update (18) is not the true optimal costate and is a function of current estimation of the ideal unknown weight, i.e. \( \lambda^T = \lambda^T(W_k) \). Denoting the weights at the \( t \)th epoch of the weight update by \( W_k^t \), the iterative procedure is given by
\[ W_{k+1}^{t+1} = \left( \phi \phi^T \right)^{-1} \phi \lambda^T(W_k^t)^T \]

Once the network is trained, it can be used for online optimal feedback control in the sense that in the online implementation, the states will be fed into the network to generate the optimal costate vector and the optimal control will be calculated through (8).

### III. Convergence Theorems

The proposed algorithm for Finite-SNAC training is based on DHP, that is, it learns the optimal costate vector through a successive approximation scheme. Starting with an initial value for the costate vector one iterates to converge to the optimal costate. Here, the proof of convergence of the weights using the training scheme given in Algorithm 1 to the optimal weights is given.

**Theorem 1**: Selecting any finite initial guess on \( W_k^0 \) for \( k \in \{0, 1, \ldots, N - 1\} \), there exists some sampling time \( \Delta t \) for discretization of the continuous dynamics (1), which for any sampling time smaller than that, using Algorithm 1 the weight matrix \( W_k^i \) converges as \( i \to \infty \), providing that the dynamics \( f(\cdot), g(\cdot) \), and the selected basis function \( \phi(\cdot) \) are smooth.

**Proof**: See Appendix A.

In Theorem 1 the role of sampling time in discretization of a continuous system is emphasized. It is worthwhile to discuss this issue in detail. Substituting (9) in costate equation (6), leads to
\[ W_k^T \phi(x_k) = Q(f(x_k) - g(x_k)R^{-1}g(x_k)^T W_k^T \phi(x_k) - r_{k+1}) + A_{k+1}^T W_{k+1}^T \phi \left( f(x_k) - g(x_k)R^{-1}g(x_k)^T W_k^T \phi(x_k) \right), \quad k \in \{0, 1, \ldots, N - 2\}. \] (19)

Optimal weights \( W_k, k \in \{0, 1, \ldots, N - 2\} \) at each time instant \( k \) can be calculated by solving the nonlinear equation (19), which is the same as (15) except that \( W_k^{i+1} \) and \( W_k^i \) are replaced with \( W_k^i \) in both sides. However, there is no analytical solution to the set of nonlinear equations (19) in general. Therefore, one needs to resort to numerical methods for solving the set of equations. Theorem 1 proves that for any given smooth dynamics, if the sampling time is small enough, the iterations given in (15) converge to the solution to the nonlinear equation (19). However, if the sampling time is fixed, certain conditions on \( R, Q \), and \( g(x) \) need to hold in order for the iterations to converge. These conditions can be easily derived from the proof of Theorem 1.

In solution to linear problems, a similar issue is observed. To see this fact, one may consider the optimal control equation (8). Consider the case of \( r_k = 0, \forall k \), for simplicity. For this case, the costate vector is assumed to be of form \( \lambda_k = P_k x_k \) for some \( P_k \in \mathbb{R}^{n \times n} \). Considering this assumption, optimal control equation (8) reads
\[ u_k = -R^{-1}B^TP_{k+1}x_{k+1} = -R^{-1}B^TP_{k+1}(Ax_k + Bu_k), \] (20)

which, similar to (19), the unknown, \( u_k \) in here, exists in both sides of the equation. However, equation (20) is linear and the analytical solution can be calculated, that is

\[ u_k = -(R + B^TP_{k+1}B)^{-1}B^TP_{k+1}Ax_k. \] (21)

If solution (21) was not available, one could use the following iterations, starting with any initial guess \( u_k^0 \), to find \( u_k \)

\[ u_k^{l+1} = -R^{-1}B^TP_{k+1}(Ax_k + Bu_k^l). \] (22)

Following the idea presented in proof of Theorem 1, it is straightforward to show that (22) is a contraction mapping, and hence \( u_k^l \) converges to the solution of (20), providing the sampling time is small enough. Therefore, as long as one can solve the set of equations (20) in the linear problem, no iterations and hence, no condition on the sampling time is required. In practical implementation however, since the training is being done offline, one can always adjust the sampling time such that the convergence is achieved.

As seen in Theorem 1, the converged weight matrices \( W_k \) are the fixed points of equations (14) and (15), hence, they satisfy

\[ W_N T \phi(x_{N-1}) = S(x_N - r_N) + \epsilon_{N-1}(x_{N-1}) \]
\[ W_k \phi(x_k) = Q(x_{k+1} - r_{k+1}) + A_k T \lambda_{k+2} + \epsilon_k(x_k), \quad k \in \{0,1,...,N-2\} \]

where \( \epsilon_k: \mathbb{R}^n \rightarrow \mathbb{R}^n, \forall k \), are the network reconstruction errors at different time steps. In other words, \( \epsilon_k(.) \)s are the approximation errors due to using neural networks with finite number of basis function for approximating a function. It should be noted that each \( \epsilon_k(x_k) \) is a function of \( x_k \), because the network may learn the functional mapping with different degrees of accuracy for different points in the domain of interest. Moreover, it can be observed that \( \epsilon_k(.) \) is the converged value of \( \epsilon_k(.) \) defined in (16). In other words, once the iterations given in Algorithm 1 converges, one has \( \epsilon_k(x_k) = \epsilon_k(x_k) \), where \( \epsilon_k(x_k) \) is the error corresponding to the last run of the iterations. By using the Galerkin method of approximation [42] which simplifies to least squares for this problem [21,25] it has been shown that the reconstruction errors can be made arbitrarily small once the number of basis functions becomes large [42]. In the ideal case where \( m \rightarrow \infty \) which results in \( \epsilon_k(x) = 0, \forall x \in \Omega \), and \( \forall k \), the generated \( \lambda_k \) through equations (9) satisfy the optimal costate equation

\[ \lambda_{k+1} = Q(x_{k+1} - r_{k+1}) + A_k T \lambda_{k+2}, \quad k \in \{0,1,...,N-1\} \]

with the final condition

\[ \lambda_N = S(x_N - r_N) \]

hence, the costate vector \( \lambda_k \) generated using the Finite-SNAC is the optimal costate vector at time \( k \). In implementation, however, the number of basis functions is finite, hence, \( \epsilon_k(.) \)s are not zero and the solution is an approximation of the optimal solution.

**Remark 5:** It should be noted that as the sampling time is made smaller for the iterative relation (15) to converge, the number of time steps, denoted with \( N \), increases. Increase in \( N \) leads to increase in the number of weight matrices, i.e., \( W_k \)s, to be trained. Since the training phase is done offline, the computational load is not an issue. However, for online implementation, one needs enough storage space for storing the trained weights.

For linear systems, which optimal finite-time tracking solution is known, it is interesting to see that Algorithm 1 provides the same solution that the discrete-time Algebraic Riccati Equation (ARE) and the respective difference equation for the tracking term provide. In other words, consider continuous-time linear system

\[ \dot{x}(t) = A_c x(t) + B_c u(t), \] (23)

which is discretized to

\[ x_{k+1} = Ax_k + Bu_k \] (24)

where \( A \in \mathbb{R}^{n \times n} \) and \( B \in \mathbb{R}^{n \times l} \) are constant matrices. The analytical optimal solution to the problem of minimizing cost function (3) subject to dynamics (24) is given by

\[ \lambda_{k+1} = P_{k+1}x_{k+1} + q_{k+1}, \] (25)

with closed-loop dynamics.
where matrix $P_k$ and vector $q_k$ are, respectively, the solutions to discrete-time ARE
\begin{equation}
\begin{align*}
P_k &= Q + A^T P_{k+1} A - A^T P_{k+1} B M B^T P_{k+1} A, \quad k \in \{0, 1, ..., N - 1\}, \\
q_k &= (A - B M B^T P_{k+1} A)^T q_{k+1} - Q r_k, \quad k \in \{0, 1, ..., N - 1\}, \quad q_N = -S r_N
\end{align*}
\end{equation}

and $M \equiv (R + B^T P_{k+1} B)^{-1}$. The next theorem proves that the optimal control solution obtained through Algorithm 1 converges to the abovementioned analytical solution.

**Theorem 2:** Let the continuous time linear system (23) be discretized to (24) using a small enough sampling time. The solution to the problem of minimizing cost function (3) subject to dynamics (24) obtained through Algorithm 1 converges to the solution of the Riccati equation with the basis functions of $\phi(x) = [x^T, 1]^T$.

**Proof:** See Appendix A.

### IV. A modification on the Finite-SNAC for optimal tracking of a family of reference signals

The finite-time optimal tracking control at each time step $k$ is a function of the system states, the reference signal, and the time-to-go. As seen in (9), the network structure is selected such that only the system state is fed to the network, however, dependency of the control on the reference signal and the time-to-go are being learned by the time-step dependent weight matrix. This synthesis is suitable for cases where the reference signal is fixed and, hence, the network can be trained offline based on that and then, in online, it will generate the optimal tracking control for different initial states $x_0$ in a closed form. However, there are cases that the reference signal is not fixed a priori, but, the dynamics of the signal is known. Let the reference signal have dynamics
\begin{equation}
\dot{r}(t) = F_c(r(t))
\end{equation}

where $F_c: \mathbb{R}^n \rightarrow \mathbb{R}^n$ and the initial condition is given by $r(0)$. It is desired to train a network to be able to optimally track the family of trajectories generated using (29) for different initial conditions $r(0)$. Discretizing (29) leads to
\begin{equation}
r_{k+1} = F(r_k), \quad k \in \{0, 1, ..., N - 1\}
\end{equation}

where $F(r_k) \equiv r_k + \Delta t F_c(r_k)$, using Euler integration, and $r_0 = r(0)$. In such cases, one may modify the Finite-SNAC in this form
\begin{equation}
\lambda_{k+1} = W_k^T \phi(x_k, r_k), \quad k \in \{0, 1, ..., N - 1\}
\end{equation}

where $\phi: \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^m$, i.e., the reference signal is directly being fed to the network, hence, it does not need to be fixed and different trajectories can be learned during offline training to be tracked online. Using the network structure given in (31), the training target equations (12) and (13) need to change to
\begin{align*}
\lambda_{N-1}^t &= S (f(x_{N-1}) - g(x_{N-1}) R^{-1} g(x_{N-1})^T W_{N-1}^T \phi(x_{N-1}, r_{N-1}) - F(r_{N-1})) \\
\lambda_{k+1}^t &= Q (f(x_k) - g(x_k) R^{-1} g(x_k)^T W_k^T \phi(x_k, r_k) - F(r_k)) \\
+ &A_{k+1}^T W_{k+1}^T \phi(f(x_k) - g(x_k) R^{-1} g(x_k)^T W_k^T \phi(x_k, r_k), F(r_k)), \quad k \in \{0, 1, ..., N - 2\}
\end{align*}

The successive approximations (14) and (15) change to
\begin{align*}
W_{N-1}^{i+1} \phi(x_{N-1}, r_{N-1}) &= S (f(x_{N-1}) - g(x_{N-1}) R^{-1} g(x_{N-1})^T W_{N-1}^T \phi(x_{N-1}, r_{N-1}) - F(r_{N-1})) \\
W_{k}^{i+1} \phi(x_k, r_k) &= Q (f(x_k) - g(x_k) R^{-1} g(x_k)^T W_k^T \phi(x_k, r_k) - F(r_k)) \\
+ &A_{k+1}^T W_{k+1}^T \phi(f(x_k) - g(x_k) R^{-1} g(x_k)^T W_k^T \phi(x_k, r_k), F(r_k)), \quad k \in \{0, 1, ..., N - 2\}
\end{align*}
The changes are the network inputs and using terms $F(r_{N-1})$ and $F(r_k)$ in the right hand sides of equations (32)-(35), instead of $r_N$ and $r_{k+1}$ which were used in (12)-(15).

It should be noted that, even though the reference signal is being fed to the network in online implementation, but, the network can generate (approximate) optimal tracking control only for the reference signal whose dynamics, i.e., $F(.)$, is learned offline. Because, as seen in (32) and (33) the training target used for offline training are dependent on function $F(.)$. However, the benefit of using network (31) versus (9) is this issue that the latter gives optimal tracking control for a single reference trajectory, while, the former gives the optimal tracking control for a family of reference signals whose dynamics is given by $F(.)$, i.e., different reference signals created using different initial conditions $r_0$ and propagated using $F(.)$.

The training algorithm for the network form (31) is given by Algorithm 2. It can easily be seen that the convergence results given in Theorems 1 hold for the network structure (31) under Algorithm 2, as well.

**Algorithm 2:**

1- Select an initial guess on $W_k^0$, $k \in \{0, 1, ..., N - k\}$.
2- Randomly select $x_{N-1} \in \Omega$ and $r_{N-1} \in \Omega$, where $\Omega$ denotes the domain of interest.
3- Through the process showed in Fig. 3 or given by Eq. (32), calculate $\lambda^k_N$.
4- Train network weight $W_{N-1}$ using input-target pair $\{(x_{N-1}, r_{N-1}), \lambda^k_N\}$.
5- Calculate the training error $e_{N-1}$ using (16).
6- Repeat step 2 to 5 until weights $W_{N-1}$ converge with a preselected tolerance and $\|e_{N-1}(x_{N-1})\| < \beta$ for different randomly selected $x_{N-1}$ and $r_{N-1}$, where $\beta$ is a small positive real number.
7- For $k = N - 2$ to $k = 0$ repeat
   
8- Randomly select state vector $x_k \in \Omega$ and reference signal $r_k \in \Omega$
9- Through the process showed in Fig. 4 or given by Eq. (33), calculate $\lambda^k_{k+1}$.
10- Train network weight $W_k$ using input-target pair $\{(x_k, r_k), \lambda^k_{k+1}\}$.
11- Calculate the training error $e_k$ using (16).
12- Repeat step 8 to 11 until weights $W_k$ converge with a preselected tolerance and $\|e_k(x_k)\| < \beta$ for different randomly selected $x_k$ and $r_k$.

---

**Fig. 3. Finite-SNAC training diagram for learning $W_{N-1}$ using Algorithm 2.**
V. Numerical Analysis

Example 1:

As the first example, the NN structure given in (9) is simulated based on Algorithm 1. A nonlinear benchmark system, namely Van der Pol’s oscillator is selected with the dynamics of

\[
\begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= (1 - x_1^2)x_2 - x_1 + u
\end{align*}
\]

where \(x_i, i = 1,2\), denotes the state vector elements. The reference trajectory to be tracked is selected as \(r(t) = [r_1(t) \ r_2(t)]^T\) which is generated using the dynamics of

\[
\dot{r}(t) = \begin{bmatrix}
\sin(\pi t) \\
\pi \cos(\pi t)
\end{bmatrix}
\]

and the initial condition of \(r(0) = [0 \ 0]^T\). The fixed horizon is selected as 3 sec. and the sampling time of \(\Delta t = 0.005\) sec. is used for the discretization of the problem. The cost function (3) with the weight matrices

\[
Q = \text{diag}(10^4 \ 10^3), \ R = 1, \ S = \text{diag}(0 \ 0)
\]

is used for the simulation.

An important step in the neurocontroller design is the selection of the basis functions. Weierstrass approximation theorem [41] proves that any continuous function on a closed and bounded interval can be uniformly approximated on that interval by polynomials to any degree of accuracy. Assuming the control, and hence the costate vector, are continuous functions of states, the basis functions \(\phi(.)\) are selected as polynomials \(x_i^a x_j^b\), where \(i, j \in \{1, 2\}, a, b \in \{0,1,2,3\}\) and \(a + b \leq 3\), which leads to 10 neurons. The network is trained for the domain of \(\Omega = \{x \in \mathbb{R}^2, -3 \leq x_1 \leq 3, -3 \leq x_2 \leq 3\}\). In performing the least squares, 20 random states \(\text{at each iteration}\) are selected and the weights have converged after 4 iterations as seen in Fig. 5 which shows the evolution of some of the weight elements versus the training iterations. The resulting (optimal) weight history is given in Fig. 6 which its variation versus time corresponds to the dependency of the costate and hence, the control on the time and the reference signal.

Once trained, the network is used for simulation of initial condition \(x_0 = [-1 \ 1]^T\) and the resulting trajectories are given in Fig. 7. As seen in this figure, Finite-SNAC has been able to force the states to track the reference signal using some finite control. In order to evaluate the optimality of the result, the open loop optimal solution to the problem is calculated using the direct method of optimization and the results are superimposed in Fig. 7 using dash plots. The open loop optimal solution is almost exactly lying on the top of the Finite-SNAC result, which confirms the optimality of the Finite-SNAC solution. However, the open loop solution is optimal only for the given initial condition and time-to-go, while, Finite-SNAC gives optimal solution to a variety of initial conditions and horizons. To see this, using the same trained network, another initial condition, i.e., \(x_0 = [1 \ 1]^T\) is simulated and the results
are depicted in Fig. 8. Since the initial condition has changed, the already calculated open loop solution is no longer valid. Another open loop solution is calculated using the numerical method and the results are shown in Fig. 8 using dash plots. Again, the Finite-SNAC results are almost exactly on the optimal trajectory which demonstrates the versatility of the Finite-SNAC controller in optimally controlling different initial conditions.

**Example 2:**

As the second example, the neurocontroller given in (31), i.e., feeding the reference signal along with the states to the network, is selected. The training is to be performed using Algorithm 2. The system is selected the same as Example 1, i.e., the Van der Pol’s oscillator, but the reference signal $r(t) = [r_1(t) \ r_2(t)]^T$ is changed to

$$\dot{r}(t) = \begin{bmatrix} -r_1^3 \\ -r_2^3 \end{bmatrix}$$

with the initial condition of $r(0) = [1 \ 1]^T$. The weight matrices are selected as

$$Q = diag(0 \ 0), \ R = 1, \ S = diag(500 \ 0)$$

which corresponds to bringing the first state of the system to close to the first element of the reference signal at the end of the horizon.

The elements of the basis functions $\phi(.)$ are selected as polynomials $x_i^a x_j^b r_1^c r_2^d$, where $i, j, u, w \in \{1, 2\}$, $a, b, c, d \in \{0,1,2,3\}$ and $a + b + c + d \leq 3$, which leads to 35 neurons. For performing the least squares, 80 random states at each iteration are selected from the domain of $\Omega = \{x \in \mathbb{R}^2, -2 \leq x_1 \leq 2, -2 \leq x_2 \leq 2\}$ and the weights converged after 5 iterations. In Fig. 9, the results with initial $x_0 = [-1 \ 0]^T$ are plotted for three different horizons of 1, 2, and 3 sec. Note that, based on the Bellman principle of optimality [14] the optimal solution for the horizon 0 to $t_f$ will be optimal for any horizon $t_1$ to $t_f$ as well, providing $0 \leq t_1 < t_f$. Hence, to have the optimal result for $t \in [t_1 \ t_f]$ using the same trained Finite-SNAC, one needs to utilize $W_k$ for $k = k_1$ to $N$, in the implementation, where $k_1 = t_1/\Delta t$ and $N = t_f/\Delta t$. The open loop optimal solution using direct method of optimization for the three different horizons are numerically calculated through three different iterative processes and the results are plotted in Fig. 9 using dash plots. The proximity of the Finite-SNAC and open loop trajectories demonstrate the potential of the Finite-SNAC in generating optimal solution for different horizons.

Finally, to analyze the performance of the controller in learning different reference trajectories which share the same dynamics, another simulation is performed using the same network with the initial reference signal of $r(0) = [-1 \ 1]^T$ and the results are shown in Fig. 10. In this figure, the initial condition $x_0$ is varied for five different values, where $x_1$ changes from -1 to 1, and $x_2$ is equal to 1. Also the horizon is changed for three different values of 1, 2, and 3 sec., and as can be seen, the controller has been successful in bringing the state $x_1$ to the trajectory at the end of the horizon, as desired. These results confirm that feeding the reference signal to the network and using its dynamics in the training provides the network with the ability of learning different trajectories with the common dynamics.

![Fig. 5. Weights evolution versus training iterations](image1)

![Fig. 6. Optimal weights evolution versus time](image2)
Fig. 7. Simulation results of Example 1 with $x_0 = [-1\ 1]^T$.

Fig. 8. Simulation results of Example 1 with $x_0 = [1\ 1]^T$.

Fig. 9. Simulation results of Example 2 with $x_0 = [-1\ 0]^T$ with different time-to-gos.
VI. Conclusions

An approximate dynamics programing based neurocontroller is developed for the optimal tracking control of nonlinear systems and the proofs are given for the convergence of the weights and the optimality of the results. The controller does not have the limitations of the cited optimal tracking controllers and can learn to track either a given reference trajectory or a family of trajectories which share the same dynamics, e.g., different trajectories resulted from different initial conditions. The numeric simulations indicate that the developed method is very versatile through approximating the optimal solution to different initial conditions, different final times, and different trajectories.

Appendix A

Proofs of the theorems are given in this appendix.

Proof of Theorem 1: The approach used in this proof is to show that the fixed point iterations given in (14) and (15) is are Contraction Mappings [33], hence, starting from any finite initial guess on $W^0_k$, $k \in \{0,1,\ldots,N-1\}$, it converges. Using the right hand side of (14) to form $\mathbf{A}^t$, equation (18) reads

$$W^{t+1}_{N-1} = \mathcal{F}(W^t_{N-1})$$

where
\( \mathcal{F}(W_{N-1}^i) \equiv (\mathbf{\Phi} \mathbf{\Phi}^T)^{-1} \phi \left[ \begin{array}{c}
abla (x_{N-1}^{(1)}) - g(x_{N-1}^{(1)}) R^{-1} g(x_{N-1}^{(1)})^T W_{N-1}^i \phi(x_{N-1}^{(1)}) - r_N \\
abla (x_{N-1}^{(p)}) - g(x_{N-1}^{(p)}) R^{-1} g(x_{N-1}^{(p)})^T W_{N-1}^i \phi(x_{N-1}^{(p)}) - r_N \end{array} \right] S \) \quad (37)

Since \( \mathbb{R}^{m \times n} \) with 2-norm denoted by \( \| \cdot \| \) is a Banach space, iterations given by (36) converges to some \( W_k^* = \mathcal{F}(W_k^*) \) if there is a \( 0 \leq \rho < 1 \) such that for every \( U_1 \) and \( U_2 \) in \( \mathbb{R}^{m \times n} \), the following inequality holds [33]

\[ \| \mathcal{F}(U_1) - \mathcal{F}(U_2) \| \leq \rho \| U_1 - U_2 \|. \] \quad (38)

Using (37)

\[ \| \mathcal{F}(U_1) - \mathcal{F}(U_2) \| = \| (\mathbf{\Phi} \mathbf{\Phi}^T)^{-1} \phi \left[ \begin{array}{c}
abla (x_{N-1}^{(1)})^T (U_1 - U_2) g(x_{N-1}^{(1)}) R^{-1} g(x_{N-1}^{(1)})^T S \\
abla (x_{N-1}^{(p)})^T (U_1 - U_2) g(x_{N-1}^{(p)}) R^{-1} g(x_{N-1}^{(p)})^T S \end{array} \right] \| \]

hence,

\[ \| \mathcal{F}(U_1) - \mathcal{F}(U_2) \| \leq \sqrt{\rho} \| (\mathbf{\Phi} \mathbf{\Phi}^T)^{-1} \phi \| \| \phi(x_{N-1}^{(1)})^T (U_1 - U_2) g(x_{N-1}^{(1)}) R^{-1} g(x_{N-1}^{(1)})^T S \| = \max_{1 \leq s \leq \rho} \| \phi(x_{N-1}^{(1)})^T (U_1 - U_2) g(x_{N-1}^{(1)}) R^{-1} g(x_{N-1}^{(1)})^T S \| \] \quad (39)

where \( s \in \{1,2,\ldots,\rho\} \) is such that

\[ \| \phi(x_{N-1}^{(1)})^T (U_1 - U_2) g(x_{N-1}^{(1)}) R^{-1} g(x_{N-1}^{(1)})^T S \| = \max_{1 \leq s \leq \rho} \| \phi(x_{N-1}^{(1)})^T (U_1 - U_2) g(x_{N-1}^{(1)}) R^{-1} g(x_{N-1}^{(1)})^T S \| \]

In (39), the following norm inequality is used

\[ \| y^{(1)} \| \leq \sqrt{\rho} \| y^{(i)} \| \] \quad (40)

where \( y^{(i)} \) are real valued row-vectors and \( \| y^{(i)} \| = \max_{1 \leq s \leq \rho} \| y^{(i)} \| \). Moreover, from (39) one has

\[ \| \mathcal{F}(U_1) - \mathcal{F}(U_2) \| \leq \sqrt{\rho} \| (\mathbf{\Phi} \mathbf{\Phi}^T)^{-1} \phi \| \| \phi(x_{N-1}^{(1)})^T R^{-1} g(x_{N-1}^{(1)})^T S \| \| U_1 - U_2 \| \] \quad (41)

Defining

\[ \rho_T \equiv \sqrt{\rho} \| (\mathbf{\Phi} \mathbf{\Phi}^T)^{-1} \phi \| \| \phi(x_{N-1}^{(1)})^T R^{-1} g(x_{N-1}^{(1)})^T S \| \] \quad (42)

inequality (41) converts to (38). What remains to show is \( 0 \leq \rho_T < 1 \). The condition of time step being small enough is used here. Note that the norms of \( g(\cdot) \) and \( R \) are proportional to \( \Delta t \), regardless of the scheme used for the discretization, for example in Euler integration one has \( g(\cdot) = \Delta t g_c(\cdot) \) and \( R = \Delta t R_c \). Moreover, the continuity of \( g(\cdot) \) and \( \phi(\cdot) \) puts some bound on the value of the state dependent terms in \( \rho_T \) in the compact set \( \Omega \) [43], hence, because \( \rho_T \) is proportional to \( \Delta t \), there always exists some sampling time \( \Delta t \) which for any sampling time smaller than that one has \( 0 \leq \rho_T < 1 \). This completes the convergence proof of \( W_{N-1}^* \).

The next step is showing the convergence of the iterations on \( W_k^*, k \in \{0,1,\ldots,N-2\} \), using (15). Utilizing the right hand side of (15) to form \( X^k \), the least squares given in (18) leads to

\[ W_{k+1}^i = G(W_k^i), \quad k \in \{0,1,\ldots,N-2\} \]

where

\[ G(W_k^i) \equiv (\mathbf{\Phi} \mathbf{\Phi}^T)^{-1} \phi \left[ \begin{array}{c}
abla (x_k^{(1)}) - g(x_k^{(1)}) R^{-1} g(x_k^{(1)})^T W_k^i \phi(x_k^{(1)}) - r_{k+1} \\
abla (x_k^{(p)}) - g(x_k^{(p)}) R^{-1} g(x_k^{(p)})^T W_k^i \phi(x_k^{(p)}) - r_{k+1} \end{array} \right] Q \]
In equation (43), to emphasize the dependency of \( A_{k+1} \) on \( W_k^{(i)} \) and \( x_k^{(j)} \), it has been denoted by \( A_{k+1}(W_k^{(i)}, x_k^{(j)}) \). Here, one needs to show that for every two matrices \( U_1 \) and \( U_2 \) in \( \mathbb{R}^{m \times n} \), function \( G \) satisfies
\[
\| G(U_1) - G(U_2) \| \leq \rho_G \| U_1 - U_2 \|
\]
where \( 0 \leq \rho_G < 1 \). Using (43), one has
\[
\| G(U_1) - G(U_2) \| \leq \|(\Phi \Phi^T)^{-1}\Phi\| \left\| \Phi(x_k^{(1)})^T (U_1 - U_2)g(x_k^{(1)})R^{-1}g(x_k^{(1)})^T Q \right\| + \|(\Phi \Phi^T)^{-1}\Phi\| \left\| \Phi(x_k^{(2)})^T (U_1 - U_2)g(x_k^{(2)})R^{-1}g(x_k^{(2)})^T Q \right\|
\]
where \( A_i^{(j)} = A_{k+1}(U_i, x_k^{(j)}), j \in \{1,2,\ldots,p\}, i \in \{1,2\} \), and
\[
\phi_i^{(j)} \equiv \phi(f(x_k^{(j)}) - g(x_k^{(j)})R^{-1}g(x_k^{(j)})^T U_i^T \Phi(x_k^{(j)})), j \in \{1,2,\ldots,p\}, i \in \{1,2\}
\]
Moreover, using the norm inequality (40), the inequality (45) reads
\[
\| G(U_1) - G(U_2) \| \leq \sqrt{\rho} \|(\Phi \Phi^T)^{-1}\Phi\| \left\| \phi_i^{(j)}^T (U_1 - U_2)g(x_k^{(j)})R^{-1}g(x_k^{(j)})^T Q \right\| + \sqrt{\rho} \|(\Phi \Phi^T)^{-1}\Phi\| \left\| \phi_m^{(m)}^T W_{k+1} A_1^{(m)} - \phi_m^{(m)}^T W_{k+1} A_2^{(m)} \right\|
\]
where \( \ell, m \in \{1,2,\ldots,p\} \) are such that
\[
\left\| \phi_i^{(j)}^T (U_1 - U_2)g(x_k^{(j)})R^{-1}g(x_k^{(j)})^T Q \right\| = \max_{1 \leq \sigma \leq p} \left\| \phi_i^{(j)}^T (U_1 - U_2)g(x_k^{(j)})R^{-1}g(x_k^{(j)})^T Q \right\|
\]
and
\[
\left\| \phi_m^{(m)}^T W_{k+1} A_1^{(m)} - \phi_m^{(m)}^T W_{k+1} A_2^{(m)} \right\| = \max_{1 \leq \sigma \leq p} \left\| \phi_m^{(m)}^T W_{k+1} A_1^{(m)} - \phi_m^{(m)}^T W_{k+1} A_2^{(m)} \right\|
\]
Once has
\[
A_i^{(j)} = f_i^{(j)} + g_i^{(j)} u_i^{(j)}, j \in \{1,2,\ldots,p\}, i \in \{1,2\}
\]
where
\[
f_i^{(j)} \equiv \frac{\partial f(x)}{\partial x} \bigg|_{x = f(x^{(j)}) - g(x^{(j)})R^{-1}g(x^{(j)})^T U_i^T \Phi(x^{(j)})}
\]
\[
g_i^{(j)} \equiv \frac{\partial g(x)}{\partial x} \bigg|_{x = f(x^{(j)}) - g(x^{(j)})R^{-1}g(x^{(j)})^T U_i^T \Phi(x^{(j)})}
\]
\[
g_i^{(j)} \equiv g(f(x^{(j)}) - g(x^{(j)})R^{-1}g(x^{(j)})^T U_i^T \Phi(x^{(j)}))
\]
\[
u_i^{(j)} \equiv -R^{-1} g_i^{(j)^T} W_{k+1} \Phi_i^{(j)}
\]
Note that \( g' \) is defined as a row-vector of matrices \([45]\), that is, \( g' \in \mathbb{R}^{n \times m \times n} \) and the \( i \)th element of this ‘vector’ is \( \partial g(x)/\partial x_i \) which is an \( n \times l \) matrix. The multiplication of \( g' \) by a matrix \( Y \in \mathbb{R}^{l \times d} \), for every positive integer \( q \), is defined as
Also \((g_1 + g_2)Y = g_1Y + g_2Y, \forall g_i \in \mathbb{R}^{n \times n}, i = 1, 2\). Moreover, we define the norm 
\[ \|g'\|_{VM} \equiv \sqrt{n} \max_i \|\partial g(x)/\partial x_i\| \]
for the vector of matrices \(g'\). Such a definition satisfies the requirements of a norm and the inequality \(\|g'Y\| \leq \|g'\|_{VM}\|Y\|\) holds.

One has 
\[ \phi_1^{(j)} W_{k+1} A_1^{(j)} - \phi_2^{(j)} W_{k+1} A_2^{(j)} = (\phi_1^{(j)} - \phi_2^{(j)})^T W_{k+1} A_1^{(j)} + \phi_2^{(j)} W_{k+1} (A_1^{(j)} - A_2^{(j)}) \]
and 
\[ A_1^{(j)} - A_2^{(j)} = \left[ f_1^{(j)} - g_1^{(j)} R^{-1} g_1^{(j)^T} W_{k+1} \phi_1^{(j)} \right] - \left[ f_2^{(j)} - g_2^{(j)} R^{-1} g_2^{(j)^T} W_{k+1} \phi_2^{(j)} \right] = \left( f_1^{(j)} - f_2^{(j)} \right) + \left( -g_1^{(j)} + g_2^{(j)} \right) R^{-1} g_1^{(j)^T} W_{k+1} \phi_1^{(j)} + g_2^{(j)} R^{-1} g_2^{(j)^T} W_{k+1} (\phi_1^{(j)} - \phi_2^{(j)}) \]
hence, 
\[ \|A_1^{(j)} - A_2^{(j)}\| \leq \|f_1^{(j)} - f_2^{(j)}\| + \|g_1^{(j)} - g_2^{(j)}\|_{VM} \left\| R^{-1} g_1^{(j)^T} W_{k+1} \phi_1^{(j)} \right\| + \|g_2^{(j)} R^{-1} g_2^{(j)^T} W_{k+1} \phi_2^{(j)}\| \] 
\[ + \|g_2^{(j)} R^{-1} - g_2^{(j)^T} W_{k+1} \phi_1^{(j)}\| + \|g_2^{(j)} R^{-1} g_2^{(j)^T} W_{k+1} \phi_2^{(j)}\| \] 

Smoothness of \(\phi(.)\), \(f(.)\), and \(g(.)\) leads to the Lipschitz continuity of \(\phi(.)\), \(g(.)\), \(\partial f/\partial x\), and \(\partial g/\partial x\) on compact set \(\Omega\) [44]. Therefore, for every \(x_1\) and \(x_2\) in \(\Omega\) one has 
\[ \|\phi(x_1) - \phi(x_2)\| \leq \rho_\phi \|x_1 - x_2\| \]
\[ \|g(x_1) - g(x_2)\| \leq \rho_g \|x_1 - x_2\| \]
\[ \|f'(x_1) - f'(x_2)\| \leq \rho_f \|x_1 - x_2\| \]
\[ \|g'(x_1) - g'(x_2)\| \leq \rho_g \|x_1 - x_2\| \]
for some positive real numbers \(\rho_\phi, \rho_g, \rho_f,\) and \(\rho_g\), where functions \(\partial f/\partial x\) and \(\partial g/\partial x\) are denoted with \(f'\) and \(g'\), respectively. Using (53) along with (46)
\[ \|\phi_1^{(j)} - \phi_2^{(j)}\| \leq \rho_\phi \|g\left(\phi_1^{(j)}\right) - g\left(\phi_2^{(j)}\right)\| U_1^T \phi\left(\phi_1^{(j)}\right) - g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T U_2^T \phi\left(\phi_1^{(j)}\right) \]
hence, 
\[ \|\phi_1^{(j)} - \phi_2^{(j)}\| \leq \rho_\phi \left\| g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T \right\| U_1 - U_2 \]

Similarly, using (53) along with (48) to (50) results in
\[ \|f_1^{(j)} - f_2^{(j)}\| \leq \rho_f \left\| g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T \right\| U_1^T \phi\left(\phi_1^{(j)}\right) - g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T U_2^T \phi\left(\phi_1^{(j)}\right) \]
\[ \|g_1^{(j)} - g_2^{(j)}\|_{VM} \leq \rho_g \left\| g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T \right\| U_1^T \phi\left(\phi_1^{(j)}\right) - g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T U_2^T \phi\left(\phi_1^{(j)}\right) \]
\[ \|g_1^{(j)} - g_2^{(j)}\| \leq \rho_g \left\| g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T \right\| U_1^T \phi\left(\phi_1^{(j)}\right) - g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T U_2^T \phi\left(\phi_1^{(j)}\right) \]
hence, 
\[ \|f_1^{(j)} - f_2^{(j)}\| \leq \rho_f \left\| \phi\left(\phi_1^{(j)}\right) \right\| \left\| g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T \right\| U_1 - U_2 \]
\[ \|g_1^{(j)} - g_2^{(j)}\|_{VM} \leq \rho_g \left\| \phi\left(\phi_1^{(j)}\right) \right\| \left\| g\left(\phi_1^{(j)}\right) R^{-1} g\left(\phi_1^{(j)}\right)^T \right\| U_1 - U_2 \]
\[ \| g_1^{(j)} - g_2^{(j)} \| \leq \rho_g \| \phi(x_k^{(j)}) \| \| g(x_k^{(j)}) R^{-1} g(x_k^{(j)})^T \| \| U_1 - U_2 \| \] \tag{57}

Using (54) and (55)-(57) in (52) leads to
\[
\left\| A_1^{(j)} - A_2^{(j)} \right\| \leq \left\{ \rho_{\phi} + \rho_{g} \| R^{-1} g_1^{(j)T} W_{k+1} T \phi_1^{(j)} \| + \rho_g \| g_2^{(j)T} R^{-1} W_{k+1} T \phi_1^{(j)} \| \right. \\
+ \rho_{\phi} \rho_{g} \| g_2^{(j)T} R^{-1} g_2^{(j)T} W_{k+1} T \phi_1^{(j)} \| \left. \right\} \phi(x_k^{(j)}) \| g(x_k^{(j)}) R^{-1} g(x_k^{(j)})^T \| \| U_1 - U_2 \| \] \tag{58}

Taking the norm of (51) leads to
\[
\left\| \phi_1^{(j)T} W_{k+1} A_1^{(j)} - \phi_2^{(j)T} W_{k+1} A_2^{(j)} \right\| \leq \left\| \phi_1^{(m)} - \phi_2^{(m)} \right\| \left\| W_{k+1} A_1^{(j)} \right\| + \left\| \phi_2^{(j)T} W_{k+1} \right\| \left\| A_1^{(j)} - A_2^{(j)} \right\| \] \tag{59}

Using (54) and (58) in (59) and setting \( j = m \) gives
\[
\left\| \phi_1^{(m)T} W_{k+1} A_1^{(m)} - \phi_2^{(m)T} W_{k+1} A_2^{(m)} \right\| \leq \rho_1 \| U_1 - U_2 \| \] \tag{60}

where
\[
\rho_1 \equiv \left\{ \rho_{\phi} \left\| W_{k+1} A_1^{(m)} \right\| + \| \phi_2^{(m)T} W_{k+1} \| \left\{ \rho_{\phi} + \rho_{g} \right. \| R^{-1} g_1^{(m)T} W_{k+1} T \phi_1^{(m)} \| + \rho_g \| g_2^{(m)T} R^{-1} W_{k+1} T \phi_1^{(m)} \| \right. \\
+ \rho_{\phi} \rho_{g} \left. \| g_2^{(m)T} R^{-1} g_2^{(m)T} W_{k+1} T \phi_1^{(m)} \| \right\} \left\| \phi(x_k^{(m)}) \| g(x_k^{(m)}) R^{-1} g(x_k^{(m)})^T \| \right. \] \tag{61}

Moreover, defining
\[
\rho_2 \equiv \sqrt{\rho} \left\| (\phi \phi^T)^{-1} \phi \right\| \left\| \phi(x_{N-1}^{(j)}) \| g(x_{N-1}^{(j)}) R^{-1} g(x_{N-1}^{(j)})^T Q \right\| \tag{62}

and using (60) and (61) in (47) leads to (44) where
\[
\rho_g = \rho_2 + \sqrt{\rho} \left\| (\phi \phi^T)^{-1} \phi \| \rho_1. \tag{63}
\]

Because of existence of term \( g\left(x_{k}^{(j)}\right) R^{-1} g\left(x_{k}^{(j)}\right)^T \) in both \( \rho_1 \) and \( \rho_2 \) as a factor, the condition of \( 0 \leq \rho_g < 1 \) can always be obtained using small enough time step \( \Delta t \), as discussed earlier in this proof for the convergence of \( W_{k-1}^{i} \). Note that, despite \( \rho_{g} \) given in (42), the coefficient \( \rho_{g} \) is a function of the parameter subject to approximation, i.e., \( W_{k}^{i} \). Hence, one may not be able to analytically calculate the proper \( \Delta t \) beforehand. But, its proved that starting with a finite \( W_{k}^{0} \) there exists some small time step, which every time step smaller than that leads to the convergence of this algorithm. This completes the proof of convergence of \( W_{k}^{i} \) for \( k \in \{0,1, \ldots, N - 2\} \). \[\blacksquare\]

**Proof of Theorem 2:** Partitioning the NN weights in the form of \( W_k = [w_k, v_k]^T \) for \( w_k \in \mathbb{R}^{n \times n} \) and \( v_k \in \mathbb{R}^n \), so that \( \lambda_{k+1} = W_k^T \phi(x_k) = w_k x_k + v_k \), and comparing this form for \( \lambda_{k+1} \) with (25) along with considering the closed loop dynamics (26) one has
\[
w_k = (p_{k+1} - p_{k+1} B M B^T p_{k+1}) A, \ k \in \{0,1, \ldots, N-2\}, \tag{63}
\]
\[
v_k = (I - p_{k+1} B M B^T) q_{k+1}, \ k \in \{0,1, \ldots, N-2\}. \tag{64}
\]

where \( I \) denotes the identity matrix. The problem is whether or not using the given representation in (63) and (64) in the weight update law of Algorithm 1 results in the difference equations (27) and (28).

Once the iterative weight update law of Algorithm 1 converges for small enough sampling time based on Theorem 1, the resulting weights satisfy
\[
w_k x_k + v_k = Q(x_{k+1} - r_{k+1}) + A^T w_{k+1} x_{k+1} + v_{k+1}. \tag{65}
\]

The closed loop dynamics based on \( w_k \) and \( v_k \) is given by
\[
x_{k+1} = (A - B R^{-1} B^T w_k) x_k - B R^{-1} B^T v_k. \tag{66}
\]

Note that as \( \Delta t \to 0 \), \( x_{k+1} \to x_k \), hence, \( (A - B R^{-1} B^T w_k) \to I \). Therefore, selecting small enough sampling time, \( (A - B R^{-1} B^T w_k) \) is invertible and the foregoing equation leads to
Substituting $x_k$ in (65) by (66) and separating terms with dependency on $x_{k+1}$ gives

$$w_k(A - BR^{-1}B^Tw_k)^{-1}x_{k+1} = Qx_{k+1} + A^Tw_{k+1}x_{k+1},$$

which to be valid for all $x_{k+1}$s in the domain of interest leads to

$$w_k(A - BR^{-1}B^Tw_k)^{-1} = Q + A^Tw_{k+1}. \quad (67)$$

Using Matrix Inversion Lemma, from (63) one has

$$w_k = (P_{k+1} - P_{k+1}BB^TP_{k+1})A = (P_{k+1}^{-1} + BR^{-1}B^T)^{-1}A.$$  

Using the representation given in the foregoing equation for $w_k$ in the left hand side of (67) gives

$$w_k(A - BR^{-1}B^Tw_k)^{-1} = (P_{k+1}^{-1} + BR^{-1}B^T)^{-1}A(A - BR^{-1}B^T(P_{k+1}^{-1} + BR^{-1}B^T)^{-1}A)^{-1}$$

$$= (P_{k+1}^{-1} + BR^{-1}B^T - BR^{-1}B^T(P_{k+1}^{-1} + BR^{-1}B^T)^{-1}(P_{k+1}^{-1} + BR^{-1}B^T))^{-1} = P_{k+1}$$

which shows that the left hand side of (67) is equivalent to the left hand side of (27) evaluated at $k + 1$. Note that in (68) invertibility of $A$ is used, and we know that as $\Delta t \to 0$, we have $A \to I$, hence, for small enough sampling time, $A$ is invertible. As for the right hand sides, using (63) for substituting $w_{k+1}$ in the right hand side of (67) leads to the right hand side of (27) evaluated at $k + 2$. Hence, (67) is equivalent to (27) considering (63). Regarding the final condition, based on the weight update law, $w_{N-1}$ satisfies

$$w_{N-1}x_{N-1} + v_N = S(x_N - r_N). \quad (69)$$

Using (66) and separating term with $x_N$ dependency gives

$$w_{N-1}(A - BR^{-1}B^Tw_k)^{-1} = S \quad (70)$$

Considering (68), equation (70) is equivalent to $P_N = S$.

Once done with $w_k$, the next step is to show that the weight update for $v_k$ is equivalent to the recursive equation (28). To this end, substituting $x_k$ in (65) by (66) and separating terms without dependency on $x_{k+1}$ gives

$$v_k(A - BR^{-1}B^Tw_k)^{-1}BR^{-1}B^Tv_k + v_k = -Qr_{k+1} + A^Tv_{k+1}. \quad (71)$$

Using (68) in (71) gives

$$P_{k+1}BR^{-1}B^Tv_k + v_k = -Qr_{k+1} + A^Tv_{k+1}. \quad (72)$$

Note that using (64)

$$-Qr_{k+1} + A^Tv_{k+1} = -Qr_{k+1} + (A - BMB^TP_{k+2}A)^Tq_{k+2}$$

Hence, the right hand side of (72) is equivalent to (28), and what remains to show is the left hand side of (72) being equal to $q_{k+1}$. Note that using (64)

$$P_{k+1}BR^{-1}B^Tv_k + v_k = (P_{k+1}BR^{-1}B^T + I)v_k = (P_{k+1}BR^{-1}B^T + I)(I - P_{k+1}BB^T)q_{k+1}$$

$$= (P_{k+1}BR^{-1}B^T - P_{k+1}BR^{-1}B^TP_{k+1}B(R + B^TP_{k+1}B)^{-1}B^T + I - P_{k+1}B(R + B^TP_{k+1}B)^{-1}B^T) q_{k+1}$$

$$= (P_{k+1}BR^{-1}B^T - P_{k+1}BR^{-1}(B^TP_{k+1}B + R)(R + B^TP_{k+1}B)^{-1}B^T + I)q_{k+1} = q_{k+1}.$$  

Finally, regarding the final condition $q_N = -Sr_N$, using (66) in (69) and separating the terms which do not depend on $x_N$ leads to

$$w_{N-1}(A - BR^{-1}B^Tw_{N-1})^{-1}BR^{-1}B^Tv_{N-1} + v_{N-1} = -Sr_N$$

Using (68) in the foregoing equation one has

$$P_NBR^{-1}B^Tv_{N-1} + v_{N-1} = -Sr_N$$

Considering (73), the foregoing equation simplifies to $q_N = -Sr_N$. This completes the proof that the recursive equation of the weights given by (65) with the final condition (69) simplifies to the recursive equations (27) and (28).\]
References
