Empowering Neural Networks with Control and Planning Abilities

Anonymous Author(s)
Affiliation
Address
email

Abstract

Differentiable planning promises end-to-end differentiability and adaptability, effectively combining the advantages of both model-free and model-based control approaches. However, many powerful nonlinear controllers still lack differentiable capabilities. For numerical controllers, the scalability of differentiating through extended horizons remains a challenge. This paper introduces a framework that facilitates differentiation through the iterative Linear Quadratic Regulator (iLQR) algorithm. A novel aspect of this framework is the analytical solution that it provides for the gradient of an iLQR controller through implicit differentiation, which ensures a constant backward cost regardless of iteration, while producing an accurate gradient. This enables a constant backward cost regardless of the planning horizon. Importantly, our method contains a correction for the gradient solution previously established in a well-regarded paper. We evaluate our framework through imitation tasks within cartpole and inverted pendulum domains. Our approach demonstrates superior sample efficiency and generalization compared to traditional neural network policies, and it provides more accurate estimations than the aforementioned method.

1 Introduction

Differentiable control has emerged as a powerful approach in the fields of reinforcement learning (RL) and imitation learning, enabling significant improvements in sample efficiency and performance. By integrating control policies into a differentiable framework, researchers can leverage gradient-based optimization techniques to directly optimize policy parameters. This integration allows for end-to-end training, where both the control strategy and the underlying model can be learned simultaneously, enhancing the adaptability and precision of control systems.

However, many existing nonlinear controllers lack differentiable capabilities, which limits their applicability in end-to-end learning frameworks. Naively differentiating through a numerical controller is a reasonable choice, but the scalability of differentiating through extended planning horizons poses a significant challenge.

As a numerical controller, iterative Linear Quadratic Regulator (iLQR) [1] have been widely used for trajectory optimization. The primary challenge in differentiable control using iLQR lies in the coupling of the forward and backward passes during training. The forward pass involves iteratively solving the iLQR optimization problem, which requires significant computational resources and memory, especially for long-horizon tasks. The backward pass, which computes gradients through backpropagation, becomes increasingly complex as it needs to traverse through all the layers of the forward pass. This coupling not only increases memory usage but also significantly slows down the training process, making it difficult to scale to larger problems.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

In this paper, we tackle these issues by introducing an innovative framework that leverages implicit differentiation to handle iLQR at its fixed point. This approach effectively separates the forward and backward computations, maintaining a constant computational load during the backward pass, irrespective of the planning horizon length. By doing so, our method significantly reduces the memory usage and computational time needed for training, thereby enhancing scalability and efficiency in handling complex control problems. Additionally, our analytical solution makes improvement from a previously established gradient computation method [2], increasing its precision and the sample efficiency of the resulting differentiable modular.

This paper makes the following contributions.

1. We develop an efficient method for analytical differentiation. We derive analytical trajectory derivatives for optimal control problems with tunable additive cost functions and constrained dynamics described by first-order difference equations, focusing on iLQR as the controller. Our analytical solution is exact, considering the entire iLQR graph. The method guarantees $O(1)$ computational complexity with respect to the number of iteration steps.
2. We demonstrate the effectiveness of our framework in imitation and system identification tasks using the inverted double pendulum and cartpole examples, showcasing superior sample efficiency and generalization compared to traditional neural network policies, as well as more accurate parameter estimations than the reference method.

Notation For a scalar function f with a vector input, ∇f denotes the gradient. Subscripts on ∇ indicate partial derivatives with respect to a subvector or the variable of interest. For tensor operations, we use $\frac{\partial(\cdot)}{\partial(\cdot)}$ for linearization (e.g., eq. (9) for Jacobians). To simplify some equations, we use $D_\theta X$ to represent $\frac{\partial X}{\partial \theta}$. Careful tracking of dependencies is crucial throughout.

2 Related Work

2.1 Differentiable Planning

Model-free policy search methods have achieved notable results in various domains by mapping observations directly to actions [3, 4, 5, 6]. However, these methods often lack interpretability, generalization, and sample efficiency [7, 8, 9, 10]. Differentiable planning combines classical planning with deep learning, enabling end-to-end training of models and policies, leveraging the strengths of both model-free and model-based approaches. Value Iteration Networks (VIN) [11] and its extensions [12, 13, 14, 15] have shown significant improvements, particularly in discrete spaces.

In continuous control, differentiable LQR approaches are prevalent, focusing on finite and infinite horizon LQR [2, 16, 17], constrained LQR [18], and PMP-based methods [19, 20, 21]. While PMP-based methods have slower convergence compared to iLQR [19], iLQR remains a robust numerical control technique [1, 22, 23], with approaches like [24] differentiating through iLQR to learn cost-shaping terms.

Many of these methods, however, involve unrolling optimization procedures, which increases memory, computation costs, and reduces stability [25, 26]. A key advancement is [2], which avoids full unrolling by differentiating through the last iLQR layer, though their analytical gradients are limited due to treating the fixed point as a constant, rather than a function of learning parameters.

2.2 Relations with Neural Networks

A neural network is essentially a function. By formulating iLQR as a differentiable module, iLQR can be embedded into auto-differentiation tools such as PyTorch for training. Thus, the iLQR module can be treated as a trainable network or embedded into a larger network structure for complex tasks, such as interacting with image processing networks for vision-based control (end-to-end learning and control). Previous works have demonstrated the advantages of such methodologies in terms of sample efficiency over black-box neural networks [11, 27, 28]. Our work serves as a foundational study on differentiable iLQR, and we believe our technique can be broadly utilized as a fundamental tool for complex tasks as illustrated by the references. Detailed applications of the technique are beyond the scope of this research.

3 Differentiable iLQR

3.1 End-to-end learning framework

In the differentiable control pipeline, the cost function g and system dynamics f are parameterized by θ, which is unknown and learnable through a performance metric L. We followed standard practice by leaving the θ-dependence implicit in the previous section, but at this point we must bring this into view by changing the notation to $f = f(x, u, \theta)$ and $g = g(x, u, \theta)$. Of course the derivatives shown in (9) and (10) are also functions of θ. For a given reference trajectory τ, the dynamics will generate three θ-dependent matrices we must consider:

$$A_t(\theta) = \frac{\partial f}{\partial x}, \quad B_t(\theta) = \frac{\partial f}{\partial u}, \quad \text{and} \quad \frac{\partial f}{\partial \theta}.$$

Careful accounting for the θ-dependence at every level is required for accurate gradients. If the loss function L can be presented as an explicit function of the trajectory τ, the influence of θ on the observed L-values will be indirect. For the composite function $\theta \mapsto L(\tau(\theta))$, the chain rule gives

$$\nabla_\theta (L \circ \tau)(\theta) = \nabla_\tau L(\tau(\theta)) \frac{\partial \tau}{\partial \theta}. \tag{1}$$

The partial derivatives required to form $\nabla_\tau L$ are provided during the backward pass by automatic differentiation tools [29, 30]. The main challenge, however, is to solve $\frac{\partial \tau}{\partial \theta}$, i.e., *the derivative of the optimal trajectory with respect to the learnable parameters*. Next, we will analytically solve $\frac{\partial \tau}{\partial \theta}$ through implicit differentiating on the fixed point.

3.2 Fixed point differentiation

The sequence of trajectories produced by iLQR can be understood as an iterative rollout of LQR layers:

$$\tau^0 \to LQR_1 \xrightarrow{\tau^1} LQR_2 \xrightarrow{\tau^2} \cdots \xrightarrow{\tau^\star} LQR_N \xrightarrow{\tau^\star}. \tag{2}$$

Each LQR layer includes the three steps noted above: linearizing the system, conducting the backward pass, and performing the forward pass. After a sufficient number of iterations (shown as N in above), the output $\tau^\star$ to an LQR step will be the same as the input, indicating that the LQR can no longer improve the trajectory it linearizes around. This trajectory $\tau^\star$ is called a fixed point for the iLQR. The fixed point of iLQR is usually reached because the line search operation in the forward pass ensures that the accumulated cost decreases with each iteration.

A fixed-point problem can be calculated by various methods, typically iterative in nature. As pointed out in [26], naively differentiating through such a scheme would require intensive memory usage [11, 13] and computational effort [25]. Instead, we propose use implicit differentiation directly on the defining identity. This gives direct access to the derivatives required by decoupling the forward (fixed-point iteration as the solver) and backward passes (differentiating through the solver).

Let us write $X = (x_1, \ldots, x_T)$ and $U = (u_1, \ldots, u_T)$ for the components of a trajectory $\tau = (x_1, u_1, x_2, u_2, \ldots, x_T, u_T)$, and abuse notation somewhat by identify τ with (X, U). At a fixed point (X^*, U^*) of the iLQR, we have the following:

$$X^* = F(X^*, U^*, \theta), \quad U^* = G(X^*, U^*, \theta) \tag{3}$$

where F and G summarize the operations that make up a single LQR layer.

In eq. (3), the solutions X^* and U^* depend on the parameter θ. By treating both X^* and U^* explicitly as functions of θ, we can interpret eq. (3) as an identity valid for all θ. Differentiating through this identity yields a new one:

$$\begin{aligned} D_\theta X^* &= \frac{\partial F}{\partial X} D_\theta X^* + \frac{\partial F}{\partial U} D_\theta U^* + \frac{\partial F}{\partial \theta} \\ D_\theta U^* &= \frac{\partial G}{\partial X} D_\theta X^* + \frac{\partial G}{\partial U} D_\theta U^* + \frac{\partial G}{\partial \theta}. \end{aligned} \tag{4}$$

Here $D_\theta X^*$ and $D_\theta U^*$ are the sensitivity matrices (Jacobians) that quantify the θ-dependence of the optimal trajectory; both depend on θ. The matrix-valued partial derivatives of F and G above

are evaluated at $(X^*(\theta), U^*(\theta), \theta)$. Rearranging the identities above produces a system of linear equations in which these matrices provide the unknowns:

$$\begin{aligned} \left(I - \frac{\partial F}{\partial X}\right) D_\theta X^* - \frac{\partial F}{\partial U} D_\theta U^* &= \frac{\partial F}{\partial \theta} \\ -\frac{\partial G}{\partial X} D_\theta X^* + \left(I - \frac{\partial G}{\partial U}\right) D_\theta U^* &= \frac{\partial G}{\partial \theta}. \end{aligned} \tag{5}$$

The analytical solution for this system is given below.

Proposition 1. *The sensitivity matrices in eq.* (5) *are given by*

$$\begin{aligned} D_\theta X^* &= M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta - G_\theta)) \\ D_\theta U^* &= (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta), \end{aligned} \tag{6}$$

where we denote $M = (I - F_X)^{-1}$ *and* $K = I - G_U$*, and use the condensed notation*

$$F_X = \frac{\partial F}{\partial X}, \quad F_U = \frac{\partial F}{\partial U}, \quad F_\theta = \frac{\partial F}{\partial \theta}, \quad G_X = \frac{\partial G}{\partial X}, \quad G_U = \frac{\partial G}{\partial U}, \quad G_\theta = \frac{\partial G}{\partial \theta}. \tag{7}$$

Proof. Please check Appendix for detailed proof. □

To be completely explicit, suppose a parameter θ is given. Then eq. (3) defines a fixed point τ^* in terms of this particular θ, and this τ^* provides the evaluation point $(X^*(\theta), U^*(\theta), \theta)$ for all the Jacobian matrices involving F and G in Equations (4) to (6).

4 Brief Experiments

We conduct experiments on two well-known control benchmarks: CartPole and Inverted Double Pendulum (IDP). The experiments demonstrate our method's runtime performance, superior learning speed and generalization. To benchmark against [2] (MPC.dx, MPC.cost, MPC.cost.dx), we evaluate three variations of our method: iLQR.dx, iLQR.cost, and iLQR.cost.dx, where 'dx' indicates learning dynamics, 'cost' learning the cost, and 'dx.cost' learning both. All experiments were conducted on a platform with an AMD 3700X 3.6GHz CPU, 16GB RAM, and an RTX3080 GPU with 10GB VRAM. The experiments are implemented with Pytorch [29].

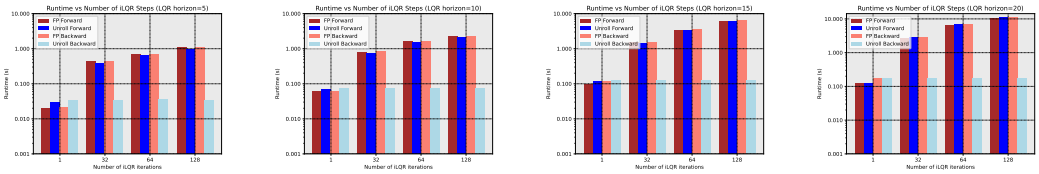

Figure 1: Runtime comparison of fixed point differentiation (FP) to unrolling the iLQR solver (Unroll) under different LQR horizons (cartpole).

References

[1] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[2] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable MPC for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[6] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596, 2018.

[7] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering Atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

[8] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.

[9] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), 2017.

[10] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[11] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[12] Sufeng Niu, Siheng Chen, Hanyu Guo, Colin Targonski, Melissa Smith, and Jelena Kovačević. Generalized value iteration networks: Life beyond lattices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[13] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. In *International Conference on Machine Learning*, pages 2947–2955. PMLR, 2018.

[14] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. Differentiable spatial planning using transformers. In *International conference on machine learning*, pages 1484–1495. PMLR, 2021.

[15] Daniel Schleich, Tobias Klamt, and Sven Behnke. Value iteration networks on multiple levels of abstraction. *arXiv preprint arXiv:1905.11068*, 2019.

[16] Jatan Shrestha, Simon Idoko, Basant Sharma, and Arun Kumar Singh. End-to-end learning of behavioural inputs for autonomous driving in dense traffic. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10020–10027. IEEE, 2023.

[17] Sebastian East, Marco Gallieri, Jonathan Masci, Jan Koutnik, and Mark Cannon. Infinite-horizon differentiable model predictive control. *Proceedings of ICLR 2020*, 2020.

[18] Ming Xu, Timothy L Molloy, and Stephen Gould. Revisiting implicit differentiation for learning problems in optimal control. *Advances in Neural Information Processing Systems*, 36, 2024.

[19] Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. *Advances in Neural Information Processing Systems*, 33:7979–7992, 2020.

[20] Wanxin Jin, Shaoshuai Mou, and George J Pappas. Safe pontryagin differentiable programming. *Advances in Neural Information Processing Systems*, 34:16034–16050, 2021.

[21] Lucas Böttcher, Nino Antulov-Fantulin, and Thomas Asikis. AI Pontryagin or how artificial neural networks learn to control dynamical systems. *Nature communications*, 13(1):333, 2022.

[22] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *First International Conference on Informatics in Control, Automation and Robotics*, volume 2, pages 222–229. SciTePress, 2004.

[23] James Zhu, J Joe Payne, and Aaron M Johnson. Convergent ilqr for safe trajectory planning and control of legged robots. *arXiv preprint arXiv:2304.00346*, 2023.

[24] Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic MPC improvement. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 336–343. IEEE, 2017.

[25] Linfeng Zhao, Huazhe Xu, and Lawson LS Wong. Scaling up and stabilizing differentiable planning with implicit differentiation. *arXiv preprint arXiv:2210.13542*, 2022.

[26] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

[27] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

[28] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*, 2019.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[30] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[31] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[32] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[33] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

A Appendix / supplemental material

A.1 ILQR Background

The Iterative Linear Quadratic Regulator (iLQR) is a trajectory optimization algorithm. At each iteration step, it linearizes the dynamics and makes quadratic approximation of the cost function, and then solves the problem using a finite-time Linear Quadratic Regulator (LQR). For a non-linear control problem defined by

$$J^\star = \min_{x_{1:T}, u_{1:T}} \sum_{t=1}^{T} g(x_t, u_t) \text{ s.t. } x_{t+1} = f(x_t, u_t), \quad \underline{u} \leq u \leq \bar{u}, \quad x_1 = x_{init}, \tag{8}$$

the iLQR algorithm optimizes the trajectory by repeatedly proceeding the following steps:

A.1.1 Linearizing the System

In the ith iteration, given trajectory $\tau^i = \{\tau_1^i, \ldots, \tau_T^i\}$ with $\tau_t^i = \{x_t^i, u_t^i\}$, linearize the dynamics

$$D_t = [A_t, B_t] = \left[\left.\frac{\partial f}{\partial x}\right|_{\tau_t^i}, \left.\frac{\partial f}{\partial u}\right|_{\tau_t^i} \right], \quad d_t = f(x, u)\Big|_{\tau_t^i} \tag{9}$$

and approximate a quadratic cost function

$$c_t = [c_{t,x}, c_{t,u}] = \left[\left.\frac{\partial g}{\partial x}\right|_{\tau_t^i}, \left.\frac{\partial g}{\partial u}\right|_{\tau_t^i} \right], \quad C_t = \begin{bmatrix} C_{t,xx} & C_{t,xu} \\ C_{t,ux} & C_{t,uu} \end{bmatrix} \tag{10}$$

with

$$C_{t,xx} = \left.\frac{\partial^2 g}{\partial x^2}\right|_{\tau_t^i}, \quad C_{t,uu} = \left.\frac{\partial^2 g}{\partial u^2}\right|_{\tau_t^i}, \quad C_{t,xu} = C_{t,ux}^\top = \left.\frac{\partial^2 g}{\partial u \partial x}\right|_{\tau_t^i}.$$

Define $\delta\tau_t = \tau_t - \tau_t^i$, where $\tau_t = (x_t, u_t)$ are the variables to be optimized. We will have an locally approximated finite-time LQR (FT-LQR)

$$\delta\tau^\star = \underset{\delta\tau_{1:T}}{\operatorname{argmin}} \sum_{t=1}^{T} \frac{1}{2} \delta\tau_t{}^\top C_t \delta\tau_t + c_t^\top \delta\tau_t \quad \text{s.t. } \delta x_{t+1} = D_t \delta\tau_t, \quad \delta x_1 = 0, \quad \underline{u} \leq u \leq \bar{u}. \tag{11}$$

A.1.2 Solving FT-LQR

The solving of FT-LQR involves of backward pass and forward pass. The backward pass is conducted by using Riccati-Mayne method [31] to obtain quadratic value function and projected-Newton method [32] to optimize actions with box constraints. The forward pass rollouts a new trajectory τ^{i+1} with u_t, that is obtained using the linear control gains K_t, k_t obtained in backward pass. Details of FT-LQR algorithm can be found in [32]. The new trajectory is prepared for the next iteration of linearization and solving of FT-LQR. Unlike the backward pass that works on an approximated linear system, the forward pass works on the true dynamics $x_{t+1}^{i+1} = f(x_t^{i+1}, u_t^{i+1})$.

A.2 Proof of proposition 1

Proposition 2. *Define $F_\theta := \frac{\partial F}{\partial \theta}$, $F_U := \frac{\partial F}{\partial U}$, $F_X := \frac{\partial F}{\partial X}$, $G_\theta := \frac{\partial G}{\partial \theta}$, $G_U := \frac{\partial G}{\partial U}$, $G_X := \frac{\partial G}{\partial X}$. Define $M := (I - F_X)^{-1}$, and $K := I - G_U$. The analytical form of the gradients $\frac{dX}{d\theta}$ and $\frac{dU}{d\theta}$ are given as follows:*

$$\begin{aligned} \frac{dX}{d\theta} &= M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta - G_\theta)) \\ \frac{dU}{d\theta} &= (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta) \end{aligned} \tag{12}$$

Proof. With the new notations, equations can be rewritten as:

$$\begin{aligned} (I - F_X)\frac{dX^*}{d\theta} - F_U \frac{dU^*}{d\theta} &= F_\theta \\ -G_X \frac{dX^*}{d\theta} + (I - G_U)\frac{dU^*}{d\theta} &= G_\theta \end{aligned} \tag{13}$$

Focusing on the first equation, $\frac{dX}{d\theta}$ can be represented with $\frac{dU}{d\theta}$:

$$\begin{aligned}\frac{dX}{d\theta} &= (I - F_X)^{-1}(F_\theta + F_U \frac{dU}{d\theta}) \\ &= M(F_\theta + F_U \frac{dU}{d\theta})\end{aligned} \tag{14}$$

Then, substituting 14 into the second equation of 13 to obtain an equation with respect to only $\frac{dU}{d\theta}$:

$$-G_X(M(F_\theta + F_U \frac{dU}{d\theta})) + (I - G_U)\frac{dU^*}{d\theta} = G_\theta \tag{15}$$

Solving equation 15 will give the solution to $\frac{dU^*}{d\theta}$:

$$\frac{dU}{d\theta} = (K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta) \tag{16}$$

Substituting 16 into 14, the solution to $\frac{dX}{d\theta}$ can be obtained:

$$\frac{dX}{d\theta} = M(F_\theta + F_U(K - G_X M F_U)^{-1}(G_X M F_\theta + G_\theta)) \tag{17}$$

This completes the proof. □

A.3 Obtaining each term in (13)

The functions F and G whose sensitivity matrices appear in eq. (7) are defined by rather complicated arg min operations. The Chain-Rule pattern below, which we can apply to either $H = F$ or $H = G$, suggests that

$$\begin{aligned}H_X &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial X} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial X} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial X} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial X}, \\ H_U &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial U} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial U} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial U} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial U}, \\ H_\theta &= \frac{\partial H}{\partial D}\frac{\partial D}{\partial \theta} + \frac{\partial H}{\partial d}\frac{\partial d}{\partial \theta} + \frac{\partial H}{\partial C}\frac{\partial C}{\partial \theta} + \frac{\partial H}{\partial c}\frac{\partial c}{\partial \theta}.\end{aligned} \tag{18}$$

In each term on the right, the first matrix factor (e.g., $\partial H/\partial D$) expresses the sensitivity of the optimal LQR trajectory to the corresponding named ingredient of the formulation in eq. (11). Efficient methods for calculating these terms are known: see [2, 33]. The second factor in each term of (18) can be computed using automatic differentiation.

A.4 Forward algorithm

If we use auto differentiation tools such as `torch.autograd.jacobian` in Pytorch to calculate $\frac{\partial D}{\partial \theta}$, the time consuming is high. Because of the backpropagation features of these tools, the gradients information from the previous time step wasn't reused for the current time step. For example, $\frac{\partial D_t}{\partial \theta}$ can indeed be simply induced once $\frac{\partial D_{t-1}}{\partial \theta}$ is ready.

To this end, we developed an efficient **forward approach** so that previous information can be reused in the consequent steps. We refer to $\frac{\partial D_t}{\partial \theta}$ in (**??**) as $\nabla_\theta D_t$ here for clarity and to distinguish it from other gradient notations, a convention we apply similarly to other gradients such as $\frac{\partial d_t}{\partial \theta}$.

Given a trajectory satisfying $x_{t+1} = f(x_t, u_t, \theta)$, the linearized dynamics D_t and d_t are functions of x_t, u_t and θ. For time step t, do

$$\nabla_\theta D_t = \frac{\partial D_t}{\partial \theta} + \left[\frac{\partial D_t}{\partial x_t} + \frac{\partial D_t}{\partial u_t}\frac{\partial u_t}{\partial x_t}\right]\nabla_\theta x_t \tag{19}$$

with

$$\nabla_\theta x_t = \frac{\partial x_t}{\partial \theta} + \left[\frac{\partial x_t}{\partial x_{t-1}} + \frac{\partial x_t}{\partial u_{t-1}}\frac{\partial u_{t-1}}{\partial x_{t-1}}\right]\nabla_\theta x_{t-1}, \tag{20}$$

where $\frac{\partial D_t}{\partial \theta}$, $\frac{\partial D_t}{\partial x_t}$, $\frac{\partial D_t}{\partial u_t}$ and $\frac{\partial x_t}{\partial \theta}$, $\frac{\partial x_t}{\partial x_{t-1}}$, $\frac{\partial x_t}{\partial u_{t-1}}$ are analytically calculated in first so that on each time step we only need to instantly plugin the corresponding parameter values to obtain the numerical gradients. $\frac{\partial u_t}{\partial x_t}$ and $\frac{\partial u_{t-1}}{\partial x_{t-1}}$ are the linear control gain solved from FT-LQR. $\nabla_\theta x_{t-1}$ is the stored information from time step $t-1$ and reused here, and $\nabla_\theta x_t$ is prepared for the next time step $t+1$. Finally

$$\nabla_\theta d_t = \nabla_\theta x_{t+1}. \tag{21}$$

The calculation of $\nabla_\theta C_t$ and $\nabla_\theta c_t$ keeps similar.