Iterative Linear Quadratic Optimization for Nonlinear Control: Differentiable Programming Algorithmic Templates

Vincent Roulet†, Siddhartha Srinivasa‡, Maryam Fazel◇, Zaid Harchaoui†

† Department of Statistics University of Washington, Seattle, USA

‡ Paul G. Allen School of Computer Science and Engineering, University of Washington, Seattle

◇ Department of Electrical and Computer Engineering, University of Washington, Seattle

March 10, 2022

Abstract

We review the implementation of nonlinear control algorithms based on linear and quadratic approximations of the objective from a functional viewpoint. We present a gradient descent, a Gauss-Newton method, a Newton method, differential dynamic programming approaches with linear quadratic or quadratic approximations, various line-search strategies and regularized variants of these algorithms. We derive the computational complexities of all algorithms in a differentiable programming framework and present sufficient optimality conditions. We compare the algorithms on several benchmarks, such as autonomous car racing using a bicycle model of a car. The algorithms are coded in a differentiable programming language in a publicly available package.

Contents

1 Introduction

We consider nonlinear control problems in discrete time with finite horizon, i.e., problems of the form

$$\min_{\substack{x_0,\ldots,x_\tau \in \mathbb{R}^{n_x} \\ u_0\ldots,u_{\tau-1} \in \mathbb{R}^{n_u}}} \sum_{t=0}^{\tau-1} h_t(x_t, u_t) + h_\tau(x_\tau) \tag{1}$$
$$\text{subject to} \quad x_{t+1} = f_t(x_t, u_t), \quad \text{for } t \in \{0, \ldots, \tau-1\}, \quad x_0 = \bar{x}_0,$$

where at time t, $x_t \in \mathbb{R}^{n_x}$ is the state of the system, $u_t \in \mathbb{R}^{n_u}$ is the control applied to the system, $f_t : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is the discrete dynamic, $h_t : \mathbb{R}^{n_x} \to \mathbb{R}$ is the cost on the state and control variables and $\bar{x}_0 \in \mathbb{R}^{n_x}$ is a given fixed initial state. Problem (1) is entirely determined by the choice of the initial state and the sequence of controllers as illustrated in Fig. 1.

Problems of the form (1) have been tackled in various ways, from direct approaches using nonlinear optimization (Betts, 2010; Wright, 1990, 1991a; Pantoja, 1988; Dunn and Bertsekas, 1989; Rao et al., 1998) to convex relaxations using semidefinite optimization (Boyd and Vandenberghe, 1997). A popular approach of the former category proceeds by computing at each iteration the linear quadratic regulator associated to a linear quadratic approximation of the problem around the current candidate solutions (Jacobson and Mayne, 1970; Li and Todorov, 2007; Sideris and Bobrow, 2005; Tassa et al., 2012). The computed feedback policies are then applied either on the linearized dynamics or on the original dynamics to output a new candidate solution.

The present technical report reviews and details the algorithmic implementation of such approaches, from the computational complexities of the optimization oracles to various implementations of line-search procedures. The objective of this work is to delineate the discrepancies between the different algorithms and identify the common subroutines. We review the implementation of (i) a Gauss-Newton method (Sideris and Bobrow, 2005), (ii) a Newton method (Pantoja, 1988; Liao and Shoemaker, 1991; Dunn and Bertsekas, 1989), (iii) a Differential Dynamic Programming (DDP) approach based on linear approximations of the dynamics and quadratic approximations of the costs (Tassa et al., 2012), (iv) a DDP approach based on quadratic approximations of both dynamics and costs (Jacobson and Mayne, 1970). We also consider regularized variants of the aforementioned algorithms with their corresponding line-searches. In addition, we present simple formulations of the gradient and the Hessian of the overall objective w.r.t. the control variables that can be used to estimate the smoothness properties of the objective. We also recall necessary optimality conditions for problem (1) and present sufficient optimality conditions derived from the continuous counterpart of the problem.

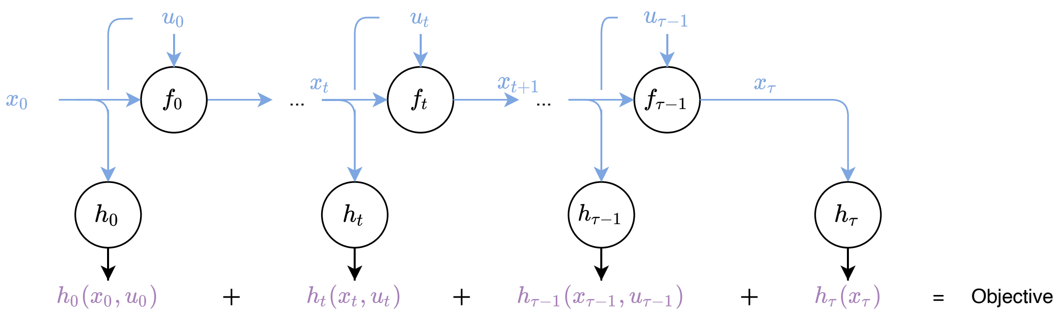

Figure 1: Computational scheme of the discrete time control problem (1).

Related work. The idea of tackling nonlinear control problems of the form (1) by minimizing linear quadratic approximations of the problem is at least 50 years old (Jacobson and Mayne, 1970). One of the first approaches consisted in a Differential Dynamic Programming (DDP) approach using quadratic approximations as presented by Jacobson and Mayne (1970) and further explored by Mayne and Polak (1975); Murray and Yakowitz (1984); Liao and Shoemaker (1991). An implementation of a Newton method for nonlinear control problems of the form (1) was developed after the DDP approach by Pantoja (1988); Dunn and Bertsekas (1989). A parallel implementation of a Newton step and sequential quadratic programming methods were developed by Wright (1990, 1991a), which led to efficient implementations of interior point methods for linear quadratic control problems (Wright, 1991b) under constraints by using the block band diagonal structure of the system of KKT equations solved at each step. A detailed comparison of the DDP approach and the Newton method was conducted by Liao and Shoemaker (1992), who observed that the original DDP approach generally outperforms its Newton counterpart. We extend this analysis by comparing regularized variants of the algorithms.

The inconvenient of the original DDP approach and a Newton method is a priori the requirement to compute and store the second order information of the dynamics. Simpler approaches consisting in taking linear approximations of the dynamics and quadratic approximations of the cost were then implemented as part of public software (Todorov et al., 2012). The resulting Iterative Linear Quadratic Regulator algorithm as formulated by Li and Todorov (2007) amounts naturally to a Gauss-Newton method (Sideris and Bobrow, 2005). A variant that mixes linear quadratic approximations of the problem with a DDP approach was further analyzed empirically by Tassa et al. (2012). Here, we detail the line-searches for both approaches and present their regularized variants. Note that a careful implementation of the original DDP approach or a Newton step does not require the computation nor the storage of second order information of the dynamics as noted by (Nganga and Wensing, 2021). We provide detailed computational complexities of all aforementioned algorithms that illustrate the trade-offs between the approaches.

Our derivations are based on the decomposition of the first and second derivatives of the problem in a compact formulation that can be used to, e.g., estimate the smoothness properties of the problem in a straightforward way. We also present sufficient optimality conditions of a candidate solution for problem (1) by translating sufficient conditions developed in continuous time by Arrow (1968); Mangasarian (1966); Kamien and Schwartz (1971).

For our experiments, we adapted the bicycle model of a miniature car developed by Liniger et al. (2015) with slight variations. We provide a simple implementation in Python[1] to focus on optimization considerations.

Outline. In Sec. 2 we recall how linear quadratic control problems are solved by dynamic programming and how the linear quadratic case serves as a building block for nonlinear control algorithms. Sec. 3 presents then how first and second order information of the objective can be expressed in terms of the first and second order information of the dynamics. The implementation of classical optimization oracles such as a gradient step, a Gauss-Newton step or a Newton step is presented in Sec. 4. Sec 5 details the rationale and the implementation of differential dynamic programming approaches. Sec. 6 details the line-search procedures. Sec. 7 presents the computational complexities of each oracle in terms of space and time complexities in a differentiable programming framework. We recall necessary optimality conditions for problem (1) and present sufficient optimality conditions in Sec. 8. A summary of all algorithms with detailed pseudo-code and computational schemes is given in Sec. 9. All algorithms

[1] The code is available at https://github.com/vroulet/ilqc.

are then tested on several synthetic problems in Sec. 10: the control of a simple pendulum, the control of a car with simple dynamics, the control of a car with more realistic dynamics using a model contouring cost.

Notations. For a sequence of vectors $x_1, \ldots, x_\tau \in \mathbb{R}^{n_x}$, we denote by semi-colons their concatenation s.t. $\boldsymbol{x} = (x_1; \ldots; x_\tau) \in \mathbb{R}^{\tau n_x}$. For a function $f : \mathbb{R}^d \to \mathbb{R}^n$, we denote by $\nabla f(x) = (\partial_{x_i} f_j(x))_{i \in \{1;\ldots d\} j \in \{1,\ldots,n\}} \in \mathbb{R}^{d \times n}$ the gradient of f, i.e., the transpose of the Jacobian of f on x. For a function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}^n$, we denote for $x \in \mathbb{R}^d$, $y \in \mathbb{R}^p$, $\nabla_x f(x,y) = (\partial_{x_i} f_j(x,y))_{i \in \{1;\ldots d\} j \in \{1,\ldots,n\}} \in \mathbb{R}^{d \times n}$ the partial gradient of f w.r.t. x on (x,y). For $f : \mathbb{R}^d \to \mathbb{R}^n$, we denote the Lipschitz-continuity constant of f as $l_f = \sup_{x,y \in \mathbb{R}^d, x \neq y} \|f(x) - f(y)\|_2 / \|x - y\|_2$.

A tensor $\mathcal{A} = (a_{i,j,k})_{1 \leq i \leq d, 1 \leq j \leq p, 1 \leq k \leq n} \in \mathbb{R}^{d \times p \times n}$ is represented as a list of matrices $\mathcal{A} = (A_1, \ldots, A_n)$ where $A_k = (a_{i,j,k})_{1 \leq i \leq d, 1 \leq j \leq p} \in \mathbb{R}^{d \times p}$ for $k \in \{1, \ldots n\}$. Given $\mathcal{A} \in \mathbb{R}^{d \times p \times n}$ and $P \in \mathbb{R}^{d \times d'}, Q \in \mathbb{R}^{p \times p'}, R \in \mathbb{R}^{n \times n'}$, we denote

$$\mathcal{A}[P, Q, R] = \left(\sum_{k=1}^n R_{k,1} P^\top A_k Q, \ldots, \sum_{k=1}^n R_{k,n'} P^\top A_k Q \right) \in \mathbb{R}^{d' \times p' \times n'}.$$

For $\mathcal{A}_0 \in \mathbb{R}^{d_0 \times p_0 \times n_0}$, $P \in \mathbb{R}^{d_0 \times d_1}, Q \in \mathbb{R}^{p_0 \times p_1}, R \in \mathbb{R}^{n_0 \times n_1}$ and $S \in \mathbb{R}^{d_1 \times d_2}, T \in \mathbb{R}^{p_1 \times p_2}, U \in \mathbb{R}^{n_1 \times n_2}$, denote $\mathcal{A}_1 = \mathcal{A}_0[P, Q, R] \in \mathbb{R}^{d_1 \times p_1 \times n_1}$. Then, we have $\mathcal{A}_1[S, T, U] = \mathcal{A}_0[PS, QT, RU] \in \mathbb{R}^{d_2 \times p_2 \times n_2}$. If P, Q or R are identity matrices, we use the symbol " $\cdot$ " in place of the identity matrix. For example, we denote $\mathcal{A}[P, Q, \mathrm{I}_n] = \mathcal{A}[P, Q, \cdot] = \left(P^\top A_1 Q, \ldots, P^\top A_n Q\right)$. If P, Q or R are vectors we consider the flatten object. In particular, for $x \in \mathbb{R}^d, y \in \mathbb{R}^p$, we denote $\mathcal{A}[x, y, \cdot] = \left(x^\top A_1 y, \ldots, x^\top A_n y\right)^\top \in \mathbb{R}^n$, rather than having $\mathcal{A}[x, y, \cdot] \in \mathbb{R}^{1 \times 1 \times n}$. Similarly, for $z \in \mathbb{R}^n$, we denote $\mathcal{A}[\cdot, \cdot, z] = \sum_{k=1}^n z_k A_k \in \mathbb{R}^{d \times p}$. We denote $\|a\|_2$ the Euclidean norm for $a \in \mathbb{R}^d$, $\|A\|_{2,2}$ the spectral norm of a matrix $A \in \mathbb{R}^{d \times p}$ and we define the norm of a tensor $\mathcal{A}$ induced by the Euclidean norm as $\|\mathcal{A}\|_{2,2,2} = \sup_{x \neq 0, y \neq 0, z \neq 0} \mathcal{A}[x, y, z] / (\|x\|_2 \|y\|_2 \|z\|_2)$.

For a multivariate function $f : \mathbb{R}^d \to \mathbb{R}^n$ composed of coordinates $f_j : \mathbb{R}^d \to \mathbb{R}$ for $j \in \{1, \ldots, n\}$, we denote its Hessian $x \in \mathbb{R}^d$ as a tensor $\nabla^2 f(x) = (\nabla^2 f_1(x), \ldots, \nabla^2 f_n(x)) \in \mathbb{R}^{d \times d \times n}$. For a multivariate function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}^n$ composed of coordinates $f_j : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$ for $j \in \{1, \ldots, n\}$, we decompose its Hessian on $x \in \mathbb{R}^d$, $y \in \mathbb{R}^p$ by defining, e.g., $\nabla^2_{xx} f(x,y) = (\nabla^2_{xx} f_1(x,y), \ldots, \nabla^2_{xx} f_n(x,y)) \in \mathbb{R}^{d \times d \times n}$. The quantities $\nabla^2_{yy} f(x,y) \in \mathbb{R}^{p \times p \times n}, \nabla^2_{xy} f(x,y) \in \mathbb{R}^{d \times p \times n}, \nabla^2_{yx} f(x,y) \in \mathbb{R}^{p \times d \times n}$ are defined similarly.

For a function $f : \mathbb{R}^d \to \mathbb{R}^n$, and $x \in \mathbb{R}^d$, we define the finite difference expansion of f around x, the linear expansion of f around x and the quadratic expansion of f around x as, respectively,

$$\delta_f^x(y) = f(x + y) - f(x), \qquad \ell_f^x(y) = \nabla f(x)^\top y, \qquad q_f^x(y) = \nabla f(x)^\top y + \frac{1}{2} \nabla^2 f(x)[y, y, \cdot]. \tag{2}$$

The linear and quadratic approximations of f around x are then $f(x + y) \approx f(x) + \ell_f^x(y)$ and $f(x + y) \approx f(x) + q_f^x(y)$ respectively.

2 From Linear Control Problems to Nonlinear Control Algorithms

Algorithms for nonlinear control problems revolve around the resolution of linear quadratic control problems by dynamic programming. We therefore start by recalling the rationale of dynamic programming and how discrete time control problems with linear dynamics and quadratic costs can be solved by dynamic programming.

2.1 Dynamic Programming

The idea of dynamic programming is to decompose dynamical problems such as (1) into a sequence of nested sub-problems defined by the cost-to-go from x_t at time $t \in \{0, \ldots, \tau - 1\}$ as

$$\begin{aligned} c_t(x_t) = \min_{\substack{u_t, \ldots, u_{\tau-1} \in \mathbb{R}^{n_u} \\ y_t, \ldots, y_\tau \in \mathbb{R}^{n_x}}} \quad & \sum_{s=t}^{\tau-1} h_s(y_s, u_s) + h_\tau(y_\tau) \\ \text{subject to} \quad & y_{s+1} = f_s(y_s, u_s) \quad \text{for } s \in \{t, \ldots, \tau - 1\}, \quad y_t = x_t. \end{aligned}$$

Algorithm 1 Dynamic programming procedure
$\left[\mathrm{DynProg} : (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \mathrm{BP} \to (u_0^*; \ldots ; u_{\tau-1}^*)\right]$.

```
Inputs: Dynamics (f_t)_{t=0}^{τ-1}, costs (h_t)_{t=0}^{τ}, initial state x̄_0, procedure BP
Initialize c_τ = h_τ
for t = τ-1, ..., 0 do
    Compute c_t, π_t = BP(f_t, h_t, c_{t+1}), store π_t
end for
Initialize x_0^* = x̄_0
for t = 0, ..., τ-1 do
    Compute u_t^* = π_t(x_t^*),   x_{t+1}^* = f_t(x_t^*, u_t^*)
end for
Output: Optimal command u = (u_0^*; ...; u_{τ-1}^*) for problem (1)
```

The cost-to-go from x_τ at time τ is simply the last cost, namely, $c_\tau(x_\tau) = h_\tau(x_\tau)$, and the original problem (1) amounts to compute $c_0(\bar{x}_0)$. The cost-to-go functions define nested sub-problems that are linked for $t \in \{0, \ldots, \tau-1\}$ by Bellman's equation (Bellman, 1971)

$$c_t(x_t) = \min_{u_t \in \mathbb{R}^{n_u}} \{h_t(x_t, u_t) + c_{t+1}(f_t(x_t, u_t))\} . \tag{3}$$

The optimal control at time t from state x_t is given by the policy

$$\pi_t(x_t) = \operatorname*{arg\,min}_{u_t \in \mathbb{R}^{n_u}} \{h_t(x_t, u_t) + c_{t+1}(f_t(x_t, u_t))\} .$$

Define the procedure that back-propagates the cost-to-go functions as

$$\mathrm{BP} : f_t, h_t, c_{t+1} \to \begin{pmatrix} c_t : x \to \min_{u \in \mathbb{R}^{n_u}} \{h_t(x, u) + c_{t+1}(f_t(x, u))\} , \\ \pi_t : x \to \operatorname{arg\,min}_{u \in \mathbb{R}^{n_u}} \{h_t(x, u) + c_{t+1}(f_t(x, u))\} \end{pmatrix} .$$

A dynamic programming approach, formally described in Algo. 1, solves problems of the form (1) as follows.

1. Compute recursively the cost-to-go functions c_t for $t = \tau, \ldots, 0$ using Bellman's equation (3), i.e., compute from $c_\tau = h_\tau$,
$$c_t, \pi_t = \mathrm{BP}(f_t, h_t, c_{t+1}) \quad \text{for } t \in \{\tau-1, \ldots, 0\},$$
and record at each step the policies π_t.

2. Unroll the optimal trajectory that starts from time 0 at $\bar{x}_0$, follows the dynamics f_t, and uses at each step the optimal control given by the computed policies, i.e., starting from $x_0^* = \bar{x}_0$, compute
$$u_t^* = \pi_t(x_t^*), \qquad x_{t+1}^* = f_t(x_t^*, u_t^*) \qquad \text{for } t = 0, \ldots, \tau-1. \tag{4}$$

The resulting command $\boldsymbol{u}^* = (u_0^*; \ldots ; u_{\tau-1}^*)$ and trajectory $\boldsymbol{x}^* = (x_1^*; \ldots ; x_\tau^*)$ are then optimal for problem (1). In the following, we consider Algo. 1 as a procedure

$$\mathrm{DynProg} : (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \mathrm{BP} \to u_0^*, \ldots, u_{\tau-1}^*.$$

The bottleneck of the approach is the ability to solve Bellman's equation (3), i.e., having access to the procedure BP defined above.

2.2 Linear Dynamics, Quadratic Costs

For linear dynamics and quadratic costs, problem (1) takes the form

$$\begin{aligned} \min_{\substack{x_0, \ldots, x_\tau \in \mathbb{R}^{n_x} \\ u_0 \ldots, u_{\tau-1} \in \mathbb{R}^{n_u}}} \quad & \sum_{t=0}^{\tau-1} \left(\frac{1}{2} x_t^\top P_t x_t + \frac{1}{2} u_t^\top Q_t u_t + x_t^\top R_t u_t + p_t^\top x_t + q_t^\top u_t \right) + \frac{1}{2} x_\tau^\top P_\tau x_\tau + p_\tau^\top x_\tau \\ \text{subject to} \quad & x_{t+1} = A_t x_t + B_t u_t, \quad \text{for } t \in \{0, \ldots, \tau-1\}, \quad x_0 = \bar{x}_0. \end{aligned}$$

Namely, we have $h_t(x_t, u_t) = \frac{1}{2}x_t^\top P_t x_t + \frac{1}{2}u_t^\top Q_t u_t + x_t^\top R_t u_t + p_t^\top x_t + q_t^\top u_t$ and $f_t(x_t, u_t) = A_t x_t + B_t u_t$. In that case, under appropriate conditions on the quadratic functions, Bellman's equation (3) can be solved analytically as recalled in Lemma 2.1. Note that the operation LQBP defined in (5) amounts to compute the Schur complement of a block of the Hessian of the quadratic $x, u \to q_t(x, u) + c_{t+1}(\ell_t(x, u))$, namely, the block corresponding to the Hessian w.r.t. the control variables.

Lemma 2.1. *The back-propagation of cost-to-go functions for linear dynamics and quadratic costs is implemented in Algo.*[2] *2 which computes*

$$\mathrm{LQBP} : (\ell_t, q_t, c_{t+1}) \to \begin{pmatrix} c_t : x \to \min_{u\in\mathbb{R}^{n_u}} \{q_t(x, u) + c_{t+1}(\ell_t(x, u))\} \\ \pi_t : x \to \arg\min_{u\in\mathbb{R}^{n_u}} \{q_t(x, u) + c_{t+1}(\ell_t(x, u))\} \end{pmatrix}, \tag{5}$$

for linear functions ℓ_t and quadratic functions q_t, c_{t+1}, s.t. $q_t(x, \cdot) + c_{t+1}(\ell_t(x, \cdot))$ is strongly convex for any x.

Proof. Consider ℓ_t, q_t, c_{t+1} to be parameterized as $\ell_t(x, u) = Ax + Bu$, $q_t(x, u) = \frac{1}{2}x^\top Px + \frac{1}{2}u^\top Qu + x^\top Ru + p^\top x + q^\top u$, $c_{t+1}(x) = \frac{1}{2}x^\top J_{t+1}x + j_{t+1}^\top x + j_{t+1}^0$. The cost-to-go function at time t is

$$\begin{aligned} c_t(x) = {} & \frac{1}{2}x^\top Px + p^\top x + j_{t+1}^0 \\ & + \min_{u\in\mathbb{R}} \left\{ \frac{1}{2}(Ax + Bu)^\top J_{t+1}(Ax + Bu) + j_{t+1}^\top (Ax + Bu) + \frac{1}{2}u^\top Qu + x^\top Ru + q^\top u \right\}. \end{aligned}$$

Since $h(x, \cdot) + c_{t+1}(\ell(x, \cdot))$ is strongly convex, we have that $Q + B^\top J_{t+1}B \succ 0$. Therefore, the policy at time t is

$$\pi_t(x) = -(Q + B^\top J_{t+1}B)^{-1}[(R^\top + B^\top J_{t+1}A)x + q + B^\top j_{t+1}],$$

such that the cost-to-go function at time t is given by

$$\begin{aligned} c_t(x) = {} & \frac{1}{2}x^\top \left(P + A^\top J_{t+1}A - (R + A^\top J_{t+1}B)(Q + B^\top J_{t+1}B)^{-1}(R^\top + B^\top J_{t+1}A)\right) x \\ & + \left(p + A^\top j_{t+1} - (R + A^\top J_{t+1}B)(Q + B^\top J_{t+1}B)^{-1}(q + B^\top j_{t+1})\right)^\top x \\ & - \frac{1}{2}(q + B^\top j_{t+1})^\top (Q + B^\top J_{t+1}B)^{-1}(q + B^\top j_{t+1}) + j_{t+1}^0. \end{aligned}$$

□

If problem (1) consists of linear dynamics and quadratic costs that are strongly convex w.r.t. the control variable, the procedure LQBP can be applied iteratively in a dynamic programming approach to give the solution of the problem, as formally stated in Corollary 2.2.

Corollary 2.2. *Consider problem (1) such that for all $t \in \{0, \ldots, \tau - 1\}$, f_t is linear, h_t is convex quadratic with $h_t(x, \cdot)$ strongly convex for any x and h_τ is convex quadratic. Then, the solution of problem (1) is given by*

$$\boldsymbol{u}^* = \mathrm{DynProg}((f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \mathrm{LQBP}),$$

with DynProg *implemented in Algo. 1 and* LQBP *implemented in Algo. 2*

Proof. Note that at time $t \in \{0, \ldots, \tau - 1\}$ for a given $x \in \mathbb{R}^{n_x}$, if c_{t+1} is convex, then $c_{t+1}(f_t(x, \cdot))$ is convex as the composition of a convex function and a linear function and $c_{t+1}(f_t(x, \cdot)) + h_t(x, \cdot)$ is then strongly convex as the sum of a convex and a strongly convex function. Moreover, $x, u \to c_{t+1}(f_t(x, u)) + h_t(x, u)$ is jointly convex since $x, u \to c_{t+1}(f_t(x, u))$ is the composition of a convex function with a linear function and h_t is convex by assumption. Therefore $c_t : x \to \min_{u\in\mathbb{R}^{n_u}} c_{t+1}(f_t(x, u)) + h_t(x, u)$ is convex as the partial infimum of jointly convex function.

In summary, at time $t \in \{0, \ldots, \tau - 1\}$, if c_{t+1} is convex, then (i) $c_{t+1}(f_t(x, \cdot)) + h_t(x, \cdot)$ is strongly convex, and (ii) c_t is convex as the partial infimum of a jointly convex function. This ensures that the assumptions of Lemma 2.1 are satisfied at each iteration of Algo. 1 (line 4) since $c_\tau = h_\tau$ is convex. □

[2]For ease of reference and comparisons, we grouped all following procedures, algorithms and computational schemes in Sec. 9.

2.3 Nonlinear Control Algorithm Example

Nonlinear control algorithms use linear or quadratic approximations of the dynamics and the costs at a current candidate sequence of controllers to apply a dynamic programming procedure on the resulting problem. For example, the Iterative Linear Quadratic Regulator (ILQR) algorithm uses linear approximations of the dynamics and quadratic approximations of the costs (Li and Todorov, 2007). Each iteration of the ILQR algorithm is composed of three steps illustrated in Fig. 4.

1. Forward pass: Given a set of control variables $u_0, \ldots, u_{\tau-1}$, compute the trajectory $x_1, \ldots, x_\tau$ as $x_{t+1} = f_t(x_t, u_t)$ starting from $x_0 = \bar{x}_0$, and the associated costs $h_t(x_t, u_t)$, $h_\tau(x_\tau)$, for $t \in \{0, \ldots, \tau-1\}$. Record along the computations, i.e., for $t \in \{0, \ldots, \tau-1\}$, the gradients of the dynamics and the gradients and Hessians of the costs.

2. Backward pass: Compute the optimal policies associated to the linear quadratic control problem

$$
\begin{aligned}
\min_{\substack{y_0,\ldots y_\tau \in \mathbb{R}^{n_x} \\ v_0,\ldots,v_{\tau-1} \in \mathbb{R}^{n_u}}} \quad & \sum_{t=0}^{\tau-1} \frac{1}{2} y_t^\top P_t y_t + \frac{1}{2} v_t^\top Q_t v_t + y_t^\top R_t v_t + p_t^\top y_t + q_t^\top v_t + \frac{1}{2} y_\tau^\top P_\tau y_\tau + p_\tau^\top y_\tau \\
\text{subject to} \quad & y_{t+1} = A_t y_t + B_t v_t, \quad \text{for } t \in \{0, \ldots, \tau-1\}, \quad y_0 = 0, \\
\text{where} \quad & P_t = \nabla^2_{x_t x_t} h_t(x_t, u_t) \;\; Q_t = \nabla^2_{u_t u_t} h_t(x_t, u_t) \;\; R_t = \nabla^2_{x_t u_t} h_t(x_t, u_t) \\
& p_t = \nabla_{x_t} h_t(x_t, u_t) \qquad q_t = \nabla_{u_t} h_t(x_t, u_t) \\
& A_t = \nabla_{x_t} f_t(x_t, u_t)^\top \quad B_t = \nabla_{u_t} f_t(x_t, u_t)^\top,
\end{aligned}
$$

which can compactly be written as

$$
\begin{aligned}
\min_{\substack{y_0,\ldots y_\tau \in \mathbb{R}^{n_x} \\ v_0,\ldots,v_{\tau-1} \in \mathbb{R}^{n_u}}} \quad & \sum_{t=0}^{\tau-1} q_{h_t}^{x_t,u_t}(y_t, v_t) + q_{h_\tau}^{x_\tau}(y_\tau) \\
\text{subject to} \quad & y_{t+1} = \ell_{f_t}^{x_t,u_t}(y_t, v_t), \quad \text{for } t \in \{0, \ldots, \tau-1\}, \quad y_0 = 0,
\end{aligned}
\tag{6}
$$

where $q_{h_\tau}^{x_\tau}(y_\tau) = \frac{1}{2} y_\tau^\top P_\tau y_\tau + p_\tau^\top y_\tau$ and $q_{h_t}^{x_t,u_t}(y_t, v_t) = \frac{1}{2} y_t^\top P_t y_t + \frac{1}{2} v_t^\top Q_t v_t + y_t^\top R_t v_t + p_t^\top y_t + q_t^\top v_t$ are the quadratic expansions of the costs and $\ell_{f_t}^{x_t,u_t}(y_t, v_t) = A_t y_t + B_t v_t$ is the linear expansion of the dynamics, both expansions being defined around the current sequence of controls and associated trajectory. The optimal policies associated to this problem are obtained by computing recursively, starting from $c_\tau = q_{h_\tau}^{x_\tau}$,

$$
c_t, \pi_t = \mathrm{LQBP}(\ell_{f_t}^{x_t,u_t}, q_{h_t}^{x_t,u_t}, c_{t+1}) \quad \text{for } t \in \{\tau-1, \ldots, 0\}.
$$

3. Roll-out pass: From the computed policies $\pi_t : y_t \to K_t y_t + k_t$, define the set of candidate policies as $\{\pi_t^\gamma : y \to \gamma k_t + K_t y \;\; \text{for } \gamma \geq 0\}$. The next sequence of controllers is then given as $u_t^{\text{next}} = u_t + v_t^\gamma$, where v_t^γ is given by rolling out the policies π_t^γ from $y_0 = 0$ along the linearized dynamics as

$$
v_t^\gamma = \pi_t^\gamma(y_t), \quad y_{t+1} = \ell_{f_t}^{x_t,u_t}(y_t, v_t^\gamma),
$$

for γ found by a line-search such that

$$
\sum_{t=0}^{\tau-1} (h_t(x_t + y_t, u_t + v_t^\gamma) - h_t(x_t, u_t)) + h_\tau(x_\tau + y_\tau) - h_\tau(x_\tau) \leq \gamma c_0(0),
$$

with $c_0(0)$ the solution of the linear quadratic control problem (6).

The procedure is then repeated on the next sequence of control variables. Ignoring the line-search phase (namely, taking $\gamma = 1$), each iteration can be summarized as computing $\boldsymbol{u}^{\text{next}} = \boldsymbol{u} + \boldsymbol{v}$ where

$$
\boldsymbol{v} = \mathrm{DynProg}((\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau}, y_0, \mathrm{LQBP})
$$

for $y_0 = 0$, where DynProg is presented in Algo 1. Note that for convex costs h_t such that $h_t(x, \cdot)$ is strongly convex, the sub-problems (6) satisfy the assumptions of Cor. 2.2.

The iterations of the following nonlinear control algorithms can always be decomposed into the three passes described above for the ILQR algorithm. They vary by (i) what approximations of the dynamics and the costs are computed in the forward pass, (ii) how the policies are computed in the backward pass, (iii) how the policies are rolled-out.

3 Optimization Objective

Problem (1) is entirely determined by the choice of the initial state and a sequence of control variables, such that the objective in (1) in terms of the control variables can be written as, for $\boldsymbol{u} = (u_0; \ldots; u_{\tau-1})$,

$$\begin{aligned} \mathcal{J}(\boldsymbol{u}) = &\sum_{t=0}^{\tau-1} h_t(x_t, u_t) + h_\tau(x_\tau) \\ &\text{s.t.} \quad x_{t+1} = f_t(x_t, u_t) \quad \text{for } t \in \{0, \ldots, \tau-1\}, \quad x_0 = \bar{x}_0. \end{aligned}$$

The objective can be decomposed into the costs and the control of τ steps of a sequence of dynamics defined as follows.

Definition 3.1. *We define the control of τ discrete time dynamics $(f_t : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x})_{t=0}^{\tau-1}$ as the function $f^{[\tau]} : \mathbb{R}^{n_x} \times \mathbb{R}^{\tau n_u} \to \mathbb{R}^{\tau n_x}$, which, given an initial point $x_0 \in \mathbb{R}^{n_x}$ and a sequence of controls $\boldsymbol{u} = (u_0; \ldots; u_{\tau-1}) \in \mathbb{R}^{\tau n_u}$, outputs the corresponding trajectory $x_1, \ldots, x_\tau$, i.e.,*

$$\begin{aligned} &f^{[\tau]}(x_0, \boldsymbol{u}) = (x_1; \ldots; x_\tau) \\ &\textit{s.t.} \quad x_{t+1} = f_t(x_t, u_t) \quad \textit{for } t \in \{0, \ldots, \tau-1\}. \end{aligned} \tag{7}$$

Overall, problem (1) can then be written as the minimization of a composition

$$\min_{\boldsymbol{u} \in \mathbb{R}^{\tau n_u}} \left\{ \mathcal{J}(\boldsymbol{u}) = h \circ g(\boldsymbol{u}) \right\}, \quad \text{where} \quad h(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{\tau-1} h_t(x_t, u_t) + h_\tau(x_\tau), \quad g(\boldsymbol{u}) = (f^{[\tau]}(\bar{x}_0, \boldsymbol{u}), \boldsymbol{u}). \tag{8}$$

The implementation of classical oracles for problem (8) simplifies thanks to the dynamical structure of the problem encapsulated in the control $f^{[\tau]}$ of the discrete time dynamics $(f_t)_{t=0}^{\tau-1}$. The following lemma presents a compact formulation of the first and second order information of $f^{[\tau]}$ with respect to the first and second order information of the dynamics $(f_t)_{t=0}^{\tau-1}$.

Lemma 3.2. *Consider the control $f^{[\tau]}$ of τ dynamics $(f_t)_{t=0}^{\tau-1}$ as defined in Def. 3.1 and an initial point $x_0 \in \mathbb{R}^{n_x}$. For $\boldsymbol{x} = (x_1; \ldots; x_\tau)$ and $\boldsymbol{u} = (u_0; \ldots; u_{\tau-1})$, define*

$$F(\boldsymbol{x}, \boldsymbol{u}) = (f_0(x_0, u_0); \ldots; f_{\tau-1}(x_{\tau-1}, u_{\tau-1})).$$

The gradient of the control $f^{[\tau]}$ of the dynamics $(f_t)_{t=0}^{\tau-1}$ on $\boldsymbol{u} \in \mathbb{R}^{\tau n_u}$ can be written

$$\nabla_{\boldsymbol{u}} f^{[\tau]}(x_0, \boldsymbol{u}) = \nabla_{\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})(\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1}.$$

The Hessian of the control $f^{[\tau]}$ of the dynamics $(f_t)_{t=0}^{\tau-1}$ on $\boldsymbol{u} \in \mathbb{R}^{\tau n_u}$ can be written

$$\nabla^2_{\boldsymbol{u}\boldsymbol{u}} f^{[\tau]}(x_0, \boldsymbol{u}) = \nabla^2_{\boldsymbol{x}\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})[N, N, M] + \nabla^2_{\boldsymbol{u}\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})[\cdot, \cdot, M] + \nabla^2_{\boldsymbol{x}\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})[N, \cdot, M] + \nabla^2_{\boldsymbol{u}\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})[\cdot, N, M].$$

where $M = (\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1}$ and $N = \nabla_{\boldsymbol{u}} f^{[\tau]}(x_0, \boldsymbol{u})^\top$.

Proof. Denote simply, for $\boldsymbol{u} \in \mathbb{R}^{\tau n_u}$, $\phi(\boldsymbol{u}) = f^{[\tau]}(x_0, \boldsymbol{u})$ with x_0 a fixed initial state. By definition, the function ϕ can be decomposed, for $\boldsymbol{u} \in \mathbb{R}^{\tau n_u}$, as $\phi(\boldsymbol{u}) = (\phi_1(\boldsymbol{u}); \ldots; \phi_\tau(\boldsymbol{u}))$, such that

$$\phi_{t+1}(\boldsymbol{u}) = f_t(\phi_t(\boldsymbol{u}), E_t^\top \boldsymbol{u}) \quad \text{for } t \in \{0, \ldots, \tau-1\}, \tag{9}$$

with $\phi_0(\boldsymbol{u}) = x_0$ and for $t \in \{0, \ldots, \tau-1\}$, $E_t = e_t \otimes \mathrm{I}_{n_u}$ is such that $E_t^\top \boldsymbol{u} = u_t$, with e_t the $t+1^{\text{th}}$ canonical vector in $\mathbb{R}^\tau$, $\otimes$ the Kronecker product and $\mathrm{I}_{n_u} \in \mathbb{R}^{n_u \times n_u}$ the identity matrix. By derivating (9), we get, denoting $x_t = \phi_t(\boldsymbol{u})$ for $t \in \{0, \ldots, \tau\}$ and using that $E_t^\top \boldsymbol{u} = u_t$,

$$\nabla \phi_{t+1}(\boldsymbol{u}) = \nabla \phi_t(\boldsymbol{u}) \nabla_{x_t} f_t(x_t, u_t) + E_t^\top \nabla_{u_t} f_t(x_t, u_t) \quad \text{for } t \in \{0, \ldots, \tau-1\}.$$

So, for $\boldsymbol{v} = (v_0; \ldots; v_{\tau-1}) \in \mathbb{R}^{\tau n_u}$, denoting $\nabla \phi(\boldsymbol{u})^\top \boldsymbol{v} = (y_1; \ldots; y_\tau)$ s.t. $\nabla \phi_t(\boldsymbol{u})^\top \boldsymbol{v} = y_t$ for $t \in \{1, \ldots, \tau\}$, we have

$$y_{t+1} = \nabla_{x_t} f_t(x_t, u_t)^\top y_t + \nabla_{u_t} f_t(x_t, u_t)^\top v_t \quad \text{for } t \in \{0, \ldots, \tau-1\}, \tag{10}$$

with $y_0 = 0$. Denoting $\boldsymbol{y} = (y_1; \dots; y_\tau)$, we have then

$$(\mathrm{I} - A)\boldsymbol{y} = B\boldsymbol{v}, \quad \text{i.e.,} \quad \nabla\phi(\boldsymbol{u})^\top \boldsymbol{v} = (\mathrm{I} - A)^{-1} B\boldsymbol{v},$$

where $A = \sum_{t=1}^{\tau-1} e_t e_{t+1}^\top \otimes A_t$ with $A_t = \nabla_{x_t} f_t(x_t, u_t)^\top$ for $t \in \{1, \dots, \tau-1\}$ and $B = \sum_{t=1}^{\tau} e_t e_t^\top \otimes B_{t-1}$ with $B_t = \nabla_{u_t} f_t(x_t, u_t)^\top$ for $t \in \{0, \dots, \tau-1\}$. , i.e.

$$A = \begin{pmatrix} 0 & A_1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ & & & \ddots & 0 \\ \vdots & & & \ddots & A_{\tau-1} \\ 0 & \dots & & \dots & 0 \end{pmatrix}, \quad B = \begin{pmatrix} B_0 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & B_{\tau-1} \end{pmatrix}.$$

By definition of F in the claim, one easily check that $A = \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})^\top$ and $B = \nabla_{\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})^\top$. Therefore we get

$$\nabla_{\boldsymbol{u}} f^{[\tau]}(x_0, \boldsymbol{u}) = \nabla\phi(\boldsymbol{u}) = \nabla_{\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})(\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1}.$$

For the Hessian, note that for $g : \mathbb{R}^d \to \mathbb{R}^p$, $f : \mathbb{R}^p \to \mathbb{R}$, we have for $x \in \mathbb{R}^d$, $\nabla^2(f \circ g)(x) = \nabla g(x) \nabla^2 f(x) \nabla g(x)^\top + \nabla^2 g(x)[\cdot, \cdot, \nabla f(x)] \in \mathbb{R}^{d \times d}$. If $f : \mathbb{R}^p \to \mathbb{R}^n$, we have $\nabla^2(f \circ g)(x) = \nabla^2 f(x)[\nabla g(x)^\top, \nabla g(x)^\top, \cdot] + \nabla^2 g(x)[\cdot, \cdot, \nabla f(x)] \in \mathbb{R}^{d \times d \times n}$. Applying this on $f_t \circ g_t$ for $g_t(\boldsymbol{u}) = (\phi_t(\boldsymbol{u}), E_t^\top \boldsymbol{u})$, we get from Eq. (9), using that $\nabla g_t(\boldsymbol{u}) = (\nabla\phi_t(\boldsymbol{u}), E_t)$,

$$\begin{aligned} \nabla^2 \phi_{t+1}(\boldsymbol{u}) = \nabla^2 \phi_t(\boldsymbol{u})[\cdot, \cdot, \nabla_{x_t} f_t(x_t, u_t)] \\ &+ \nabla^2_{x_t x_t} f_t(x_t, u_t)[\nabla\phi_t(\boldsymbol{u})^\top, \nabla\phi_t(\boldsymbol{u})^\top, \cdot] + \nabla^2_{u_t u_t} f_t(x_t, u_t)[E_t^\top, E_t^\top, \cdot] \\ &+ \nabla^2_{x_t u_t} f_t(x_t, u_t)[\nabla\phi_t(\boldsymbol{u})^\top, E_t^\top, \cdot] + \nabla^2_{u_t x_t} f_t(x_t, u_t)[E_t^\top, \nabla\phi_t(\boldsymbol{u})^\top, \cdot], \end{aligned}$$

for $t \in \{0, \dots, \tau-1\}$, with $\nabla^2 \phi_0(\boldsymbol{u}) = 0$. Therefore for $\boldsymbol{v} = (v_0; \dots; v_{\tau-1}), \boldsymbol{w} = (w_0; \dots; w_{\tau-1}) \in \mathbb{R}^{\tau n_u}$, $\boldsymbol{\mu} = (\mu_1; \dots; \mu_\tau) \in \mathbb{R}^{\tau n_x}$, we get

$$\begin{aligned} \nabla^2 \phi(\boldsymbol{u})[\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{\mu}] &= \sum_{t=0}^{\tau-1} \nabla^2 \phi_{t+1}(\boldsymbol{u})[\boldsymbol{v}, \boldsymbol{w}, \mu_{t+1}] \\ &= \sum_{t=0}^{\tau-1} \Big(\nabla^2_{x_t x_t} f_t(x_t, u_t)[y_t, z_t, \lambda_{t+1}] + \nabla^2_{u_t u_t} f_t(x_t, u_t)[v_t, w_t, \lambda_{t+1}] \\ &\qquad + \nabla^2_{x_t u_t} f_t(x_t, u_t)[y_t, w_t, \lambda_{t+1}] + \nabla^2_{u_t x_t} f_t(x_t, u_t)[v_t, z_t, \lambda_{t+1}] \Big), \end{aligned} \tag{11}$$

where $\boldsymbol{y} = (y_1; \dots; y_\tau) = \nabla\phi(\boldsymbol{u})^\top \boldsymbol{v}$, $\boldsymbol{z} = (z_1; \dots; z_\tau) = \nabla\phi(\boldsymbol{u})^\top \boldsymbol{w}$, with $y_0 = z_0 = 0$ and $\boldsymbol{\lambda} = (\lambda_1; \dots; \lambda_\tau) \in \mathbb{R}^{\tau n_x}$ is defined by

$$\lambda_t = \nabla_{x_t} f_t(x_t, u_t) \lambda_{t+1} + \mu_t \qquad \text{for } t \in \{1, \dots, \tau-1\}, \quad \lambda_\tau = \mu_\tau.$$

On the other hand, denoting $F_t(\boldsymbol{x}, \boldsymbol{u}) = f_t(x_t, u_t)$ for $t \in \{0, \dots, \tau-1\}$, the Hessian of F with respect to the variables $\boldsymbol{u}$ can be decomposed as

$$\nabla^2_{\boldsymbol{u}\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{\lambda}] = \sum_{t=0}^{\tau-1} \nabla^2_{\boldsymbol{u}\boldsymbol{u}} F_t(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{v}, \boldsymbol{w}, \lambda_{t+1}] = \sum_{t=0}^{\tau-1} \nabla^2_{u_t u_t} f_t(x_t, u_t)[v_t, w_t, \lambda_{t+1}].$$

The Hessian of F with respect to the variable $\boldsymbol{x}$ can be decomposed as

$$\nabla^2_{\boldsymbol{x}\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\lambda}] = \sum_{t=0}^{\tau-1} \nabla^2_{\boldsymbol{x}\boldsymbol{x}} F_t(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{y}, \boldsymbol{z}, \lambda_{t+1}] = \sum_{t=1}^{\tau-1} \nabla^2_{x_t x_t} f_t(x_t, u_t)[y_t, z_t, \lambda_{t+1}].$$

A similar decomposition can be done for $\nabla^2_{\boldsymbol{x}\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})$. From Eq. (11), we then get

$$\nabla^2 \phi(\boldsymbol{u})[\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{\mu}] = \nabla^2_{\boldsymbol{x}\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\lambda}] + \nabla^2_{\boldsymbol{u}\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{\lambda}] + \nabla^2_{\boldsymbol{x}\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{\lambda}] + \nabla^2_{\boldsymbol{u}\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})[\boldsymbol{v}, \boldsymbol{z}, \boldsymbol{\lambda}].$$

Finally, by noting that $\boldsymbol{y} = (\nabla_{\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})(\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1})^\top \boldsymbol{v}$, $\boldsymbol{z} = (\nabla_{\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})(\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1})^\top \boldsymbol{w}$, and $\boldsymbol{\lambda} = (\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1} \boldsymbol{\mu}$, the claim is shown. $\square$

Lemma 3.2 can be used to have simple estimates on the smoothness properties of the control of τ dynamics given the smoothness properties of each individual dynamics.

Lemma 3.3. *If τ dynamics $(f_t)_{t=0}^{\tau-1}$ are Lipschitz-continuous with Lipschitz-continuous gradients, then the function $\boldsymbol{u} \to f^{[\tau]}(x_0, \boldsymbol{u})$, with $f^{[\tau]}$ the control of the τ dynamics $(f_t)_{t=0}^{\tau-1}$, is $l_{f^{[\tau]}}$-Lipschitz continuous and has $L_{f^{[\tau]}}$-Lipschitz-continuous gradients with*

$$l_{f^{[\tau]}} \leq l_f^u S, \qquad L_{f^{[\tau]}} \leq S(L_f^{xx} l_{f^{[\tau]}}^2 + 2L_f^{xu} l_{f^{[\tau]}} + L_f^{uu}), \tag{12}$$

where $S = \sum_{t=0}^{\tau-1} (l_f^x)^t$, $l_{f_t}^u = \sup_{x,u} \|\nabla_u f(x,u)\|_{2,2}$, $l_{f_t}^x = \sup_{x,u} \|\nabla_x f(x,u)\|_{2,2}$, $L_{f_t}^{xx} = \sup_{x,u} \|\nabla_{xx}^2 f(x,u)\|_{2,2,2}$, $L_{f_t}^{uu} = \sup_{x,u} \|\nabla_{uu}^2 f(x,u)\|_{2,2,2}$, $L_{f_t}^{xu} = \sup_{x,u} \|\nabla_{xu}^2 f(x,u)\|_{2,2,2}$ and we drop the index to denote the maximum over all dynamics such as $l_f^x = \max_{t \in \{0,\ldots,\tau-1\}} l_{f_t}^x$.

Proof. The Lipschitz-continuity of $\boldsymbol{u} \to f^{[\tau]}(x_0, \boldsymbol{u})$ and its gradients can be estimated by upper bounding the norm of the gradients and the Hessians. With the notations of Lemma 3.2, $\nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})$ is nilpotent of degree τ since it can be written $\nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=1}^{\tau-1} e_{t+1} e_t^\top \otimes \nabla_{x_t} f_t(x_t, u_t)$ and $(A \otimes B)(C \otimes D) = (AC \otimes BD)$. Hence, we have $(\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1} = \sum_{t=0}^{\tau-1} \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u})^t$. The Lipschitz-continuity of $f^{[\tau]}$ is then estimated by $\|\nabla_{\boldsymbol{u}} f^{[\tau]}(x_0, \boldsymbol{u})\|_{2,2} \leq \|\nabla_{\boldsymbol{u}} F(\boldsymbol{x}, \boldsymbol{u})\|_{2,2} \|(\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1}\|_{2,2} \leq l_f^u \sum_{t=0}^{\tau-1} (l_f^x)^t$. As shown in Lemma 3.2, the Hessian of $\boldsymbol{u} \to f^{[\tau]}(x_0, \boldsymbol{u})$ can be decomposed as

$$\nabla_{\boldsymbol{uu}}^2 f^{[\tau]}(x_0, \boldsymbol{u}) = \nabla_{\boldsymbol{xx}}^2 F(\boldsymbol{x}, \boldsymbol{u})[N, N, M] + \nabla_{\boldsymbol{uu}}^2 F(\boldsymbol{x}, \boldsymbol{u})[\cdot, \cdot, M] + \nabla_{\boldsymbol{xu}}^2 F(\boldsymbol{x}, \boldsymbol{u})[N, \cdot, M] + \nabla_{\boldsymbol{ux}}^2 F(\boldsymbol{x}, \boldsymbol{u})[\cdot, N, M],$$

where $M = (\mathrm{I} - \nabla_{\boldsymbol{x}} F(\boldsymbol{x}, \boldsymbol{u}))^{-1}$ and $N = \nabla_{\boldsymbol{u}} f^{[\tau]}(\bar{x}_0, \boldsymbol{u})^\top$. Given the structure of F, bounds on the Hessians are $\|\nabla_{ab}^2 F(\boldsymbol{x}, \boldsymbol{u})\|_{2,2,2} \leq L_f^{ab}$ for $a, b \in \{\boldsymbol{x}, \boldsymbol{u}\}$, where $\|\mathcal{A}\|_{2,2,2}$ is the norm of a tensor $\mathcal{A}$ w.r.t. the Euclidean norm as defined in the notations. Note that for a given tensor $\mathcal{A} \in \mathbb{R}^{d \times p \times n}$ and P, Q, R of appropriate sizes, we have $\|\mathcal{A}[P, Q, R]\|_{2,2,2} \leq \|\mathcal{A}\|_{2,2,2} \|P\|_{2,2} \|Q\|_{2,2} \|R\|_{2,2}$. We then get

$$||\nabla_{\boldsymbol{uu}}^2 f^{[\tau]}(x_0, \boldsymbol{u})||_{2,2,2} \leq L_f^{xx} \|N\|_{2,2}^2 \|M\|_{2,2} + L_f^{uu} \|M\|_{2,2} + 2L_f^{xu} \|M\|_{2,2} \|N\|_{2,2},$$

where for twice differentiable functions we used that $L_f^{xu} = L_f^{ux}$. □

4 Classical Optimization Algorithms

4.1 Oracles Formulation

Classical optimization algorithms rely on the availability to some oracles on the objective. Here, we consider these oracles to compute the minimizer of an approximation of the objective around the current point with an optional regularization term. Formally, on a point $\boldsymbol{u} \in \mathbb{R}^{\tau n_u}$, given a regularization $\nu \geq 0$, for an objective of the form

$$\min_{\boldsymbol{u} \in \mathbb{R}^{\tau n_u}} \quad h \circ g(\boldsymbol{u}),$$

as in (8), we consider

(i) a *gradient* oracle to use a linear approximation of the objective, and to output, for $\nu > 0$,

$$\operatorname*{arg\,min}_{\boldsymbol{v} \in \mathbb{R}^{\tau n_u}} \left\{ \ell_{h \circ g}^{\boldsymbol{u}}(\boldsymbol{v}) + \frac{\nu}{2} \|\boldsymbol{v}\|_2^2 \right\} = -\nu^{-1} \nabla (h \circ g)(\boldsymbol{u}), \tag{13}$$

(ii) a *Gauss-Newton* oracle to use a linear quadratic approximation of the objective, and to output

$$\operatorname*{arg\,min}_{\boldsymbol{v} \in \mathbb{R}^{\tau n_u}} \left\{ q_h^{g(\boldsymbol{u})}(\ell_g^{\boldsymbol{u}}(\boldsymbol{v})) + \frac{\nu}{2} \|\boldsymbol{v}\|_2^2 \right\} = -(\nabla g(\boldsymbol{u}) \nabla^2 h(g(\boldsymbol{u})) \nabla g(\boldsymbol{u}) + \nu \, \mathrm{I})^{-1} \nabla (h \circ g)(\boldsymbol{u}), \tag{14}$$

(iii) a *Newton* oracle to use a quadratic approximation of the objective, and to output

$$\operatorname*{arg\,min}_{\boldsymbol{v} \in \mathbb{R}^{\tau n_u}} \left\{ q_{h \circ g}^{\boldsymbol{u}}(\boldsymbol{v}) + \frac{\nu}{2} \|\boldsymbol{v}\|_2^2 \right\} = -(\nabla^2 (h \circ g)(\boldsymbol{u}) + \nu \, \mathrm{I})^{-1} \nabla (h \circ g)(\boldsymbol{u}), \tag{15}$$

where ℓ_f^x, q_f^x are the linear and quadratic expansions of a function f around x as defined in the notations in Eq. (2).

Gauss-Newton and Newton oracles are generally defined without a regularization, i.e., for $\nu = 0$. However, in practice, a regularization may be necessary to ensure that Gauss-Newton and Newton oracles provide a descent direction. Moreover, the reciprocal of the regularization, $1/\nu$, can play the role of a stepsize as detailed in Sec. 6. Lemma 4.1 presents how the computation of the above oracles can be decomposed into the dynamical structure of the problem.

Lemma 4.1. *Consider a nonlinear dynamic problem summarized as*

$$\min_{\boldsymbol{u}\in\mathbb{R}^{\tau n_u}} h\circ g(\boldsymbol{u}), \quad \textit{where} \quad h(\boldsymbol{x},\boldsymbol{u}) = \sum_{t=0}^{\tau-1} h_t(x_t,u_t) + h_\tau(x_\tau), \quad g(\boldsymbol{u}) = (f^{[\tau]}(\bar{x}_0,\boldsymbol{u}),\boldsymbol{u}),$$

with $f^{[\tau]}$ the control of τ dynamics $(f_t)_{t=0}^{\tau-1}$ as defined in Def. 3.1.

Let $\boldsymbol{u} = (u_0;\ldots;u_{\tau-1})$ and $f^{[\tau]}(\bar{x}_0,\boldsymbol{u}) = (x_1;\ldots;x_\tau)$. Gradient (13), Gauss-Newton (14) and Newton (15) oracles for $h\circ g$ amount to solving for $\boldsymbol{v}^* = (v_0^*;\ldots;v_{\tau-1}^*)$ linear quadratic control problems of the form

$$\begin{aligned}
\min_{\substack{v_0,\ldots,v_{\tau-1}\in\mathbb{R}^{n_u}\\ y_0,\ldots,y_\tau\in\mathbb{R}^{n_x}}} \quad & \sum_{t=0}^{\tau-1} q_t(y_t,v_t) + q_\tau(y_\tau) \\
\textit{subject to} \quad & y_{t+1} = \ell_{f_t}^{x_t,u_t}(y_t,v_t) \quad \textit{for } t\in\{0,\ldots,\tau-1\}, \quad y_0 = 0,
\end{aligned} \tag{16}$$

where for

(i) the gradient oracle (13), $q_\tau(y_\tau) = \ell_{h_\tau}^{x_\tau}(y_\tau)$ and, for $0\le t\le \tau-1$,

$$q_t(y_t,v_t) = \ell_{h_t}^{x_t,u_t}(y_t,v_t) + \frac{\nu}{2}\|v_t\|_2^2,$$

(ii) the Gauss-Newton oracle (14), $q_\tau(y_\tau) = q_{h_\tau}^{x_\tau}(y_\tau)$ and, for $0\le t\le\tau-1$,

$$q_t(y_t,v_t) = q_{h_t}^{x_t,u_t}(y_t,v_t) + \frac{\nu}{2}\|v_t\|_2^2,$$

(iii) for the Newton oracle (15), $q_\tau(y_\tau) = q_{h_\tau}^{x_\tau}(y_\tau)$ and, defining

$$\lambda_\tau = \nabla h_\tau(x_\tau), \quad \lambda_t = \nabla_{x_t} h_t(x_t,u_t) + \nabla_{x_t} f_t(x_t,u_t)\lambda_{t+1} \quad \textit{for } t\in\{\tau-1,\ldots,1\}, \tag{17}$$

we have, for $0\le t\le\tau-1$,

$$q_t(y_t,v_t) = q_{h_t}^{x_t,u_t}(y_t,v_t) + \frac{1}{2}\nabla^2 f_t(x_t,u_t)[\cdot,\cdot,\lambda_{t+1}](y_t,v_t) + \frac{\nu}{2}\|v_t\|_2^2,$$

where for $f:\mathbb{R}^{n_x}\times\mathbb{R}^{n_u}\to\mathbb{R}^{n_x}$, $x\in\mathbb{R}^{n_x}$, $u\in\mathbb{R}^{n_u}$, $\lambda\in\mathbb{R}^{n_x}$, we define

$$\nabla^2 f(x,u)[\cdot,\cdot,\lambda] : (y,v) \to \nabla_{xx}^2 f(x,u)[y,y,\lambda] + 2\nabla_{xu}^2 f(x,u)[y,v,\lambda] + \nabla_{uu}^2 f(x,u)[v,v,\lambda]. \tag{18}$$

Proof. In the following, we denote for simplicity $\phi(\boldsymbol{u}) = f^{[\tau]}(\bar{x}_0,\boldsymbol{u})$. The optimization oracles can be rewritten as follows.

1. The gradient oracle (13) is given by

$$v^* = \operatorname*{arg\,min}_{\boldsymbol{v}\in\mathbb{R}^{\tau n_u}} \left\{ \nabla h(g(\boldsymbol{u}))^\top \nabla g(\boldsymbol{u})^\top \boldsymbol{v} + \frac{\nu}{2}\|\boldsymbol{v}\|_2^2 \right\}. \tag{19}$$

2. The Gauss-Newton oracle (14) is given by

$$v^* = \operatorname*{arg\,min}_{\boldsymbol{v}\in\mathbb{R}^{\tau n_u}} \left\{ \frac{1}{2}\boldsymbol{v}^\top \nabla g(\boldsymbol{u})\nabla^2 h(g(\boldsymbol{u}))\nabla g(\boldsymbol{u})^\top \boldsymbol{v} + \nabla h(g(\boldsymbol{u}))^\top \nabla g(\boldsymbol{u})^\top \boldsymbol{v} + \frac{\nu}{2}\|\boldsymbol{v}\|_2^2 \right\}. \tag{20}$$

3. The Newton oracle (15) is given by

$$v^* = \operatorname*{arg\,min}_{\boldsymbol{v}\in\mathbb{R}^{\tau n_u}} \left\{ \frac{1}{2}\boldsymbol{v}^\top \nabla g(\boldsymbol{u})\nabla^2 h(g(\boldsymbol{u}))\nabla g(\boldsymbol{u})^\top \boldsymbol{v} + \frac{1}{2}\nabla^2 g(\boldsymbol{u})[\boldsymbol{v},\boldsymbol{v},\nabla h(g(\boldsymbol{u}))] + \nabla h(g(\boldsymbol{u}))^\top \nabla g(\boldsymbol{u})^\top \boldsymbol{v} + \frac{\nu}{2}\|\boldsymbol{v}\|_2^2 \right\}. \tag{21}$$

We have, denoting $\boldsymbol{x} = \phi(\boldsymbol{u})$,

$$
\begin{aligned}
\nabla h(g(\boldsymbol{u}))^\top \nabla g(\boldsymbol{u})^\top \boldsymbol{v} &= \nabla_{\boldsymbol{x}} h(\boldsymbol{x}, \boldsymbol{u})^\top \nabla \phi(\boldsymbol{u})^\top \boldsymbol{v} + \nabla_{\boldsymbol{u}} h(\boldsymbol{x}, \boldsymbol{u})^\top \boldsymbol{v} \\
\boldsymbol{v}^\top \nabla g(\boldsymbol{u}) \nabla^2 h(g(\boldsymbol{u})) \nabla g(\boldsymbol{u})^\top \boldsymbol{v} &= \boldsymbol{v}^\top \nabla \phi(\boldsymbol{u}) \nabla^2_{\boldsymbol{xx}} h(\boldsymbol{x}, \boldsymbol{u}) \nabla \phi(\boldsymbol{u})^\top \boldsymbol{v} + \boldsymbol{v}^\top \nabla^2_{\boldsymbol{uu}} h(\boldsymbol{x}, \boldsymbol{u}) \boldsymbol{v} + 2\boldsymbol{v}^\top \nabla \phi(\boldsymbol{u}) \nabla^2_{\boldsymbol{xu}} h(\boldsymbol{x}, \boldsymbol{u}) \boldsymbol{v} \\
\nabla^2 g(\boldsymbol{u})[\boldsymbol{v}, \boldsymbol{v}, \nabla h(g(\boldsymbol{u}))] &= \nabla^2 \phi(\boldsymbol{u})[\boldsymbol{v}, \boldsymbol{v}, \nabla_{\boldsymbol{x}} h(\boldsymbol{x}, \boldsymbol{u})].
\end{aligned}
$$

For $\boldsymbol{v} = (v_0; \ldots; v_{\tau-1}) \in \mathbb{R}^{\tau n_u}$, denoting $\boldsymbol{y} = \nabla \phi(\boldsymbol{u})^\top \boldsymbol{v} = (y_1; \ldots; y_\tau)$, with $y_0 = 0$, we have

$$
\nabla h(g(\boldsymbol{u}))^\top \nabla g(\boldsymbol{u})^\top \boldsymbol{v} = \sum_{t=0}^{\tau-1} \left[\nabla_{x_t} h_t(x_t, u_t)^\top y_t + \nabla_{u_t} h_t(x_t, u_t)^\top v_t\right] + \nabla h_\tau(x_\tau)^\top y_\tau = \sum_{t=0}^{\tau-1} \ell_{h_t}^{x_t, u_t}(y_t, v_t) + \ell_{h_\tau}^{x_\tau}(y_\tau). \tag{22}
$$

Following the proof of Lemma 3.2, we have that $\boldsymbol{y} = \nabla \phi(\boldsymbol{u})^\top \boldsymbol{v} = (y_1; \ldots; y_\tau)$ satisfies

$$
y_{t+1} = \nabla_{x_t} f_t(x_t, u_t)^\top y_t + \nabla_{u_t} f_t(x_t, u_t)^\top v_t = \ell_{f_t}^{x_t, u_t}(y_t, v_t), \tag{23}
$$

with $y_0 = 0$. Hence, plugging Eq. (22) and Eq. (23) into Eq. (19) we get the claim for the gradient oracle.

The Hessians of the total cost are block diagonal with, e.g., $\nabla^2_{\boldsymbol{uu}} h(\boldsymbol{x}, \boldsymbol{u})$ being composed of τ diagonal blocks of the form $\nabla^2_{u_t u_t} h_t(x_t, u_t)$ for $t \in \{0, \ldots, \tau-1\}$. Therefore, we have

$$
\begin{aligned}
&\frac{1}{2} \boldsymbol{v}^\top \nabla g(\boldsymbol{u}) \nabla^2 h(g(\boldsymbol{u})) \nabla g(\boldsymbol{u})^\top \boldsymbol{v} \\
&= \sum_{t=0}^{\tau-1} \left[\frac{1}{2} y_t^\top \nabla^2_{x_t x_t} h_t(x_t, u_t) y_t + \frac{1}{2} v_t^\top \nabla^2_{u_t u_t} h_t(x_t, u_t) u_t + y_t^\top \nabla^2_{x_t u_t} h_t(x_t, u_t) v_t\right] + \frac{1}{2} y_\tau^\top \nabla^2 h_\tau(x_\tau) y_\tau.
\end{aligned}
$$

The linear quadratic approximation in (20) amounts then to

$$
\frac{1}{2} \boldsymbol{v}^\top \nabla g(\boldsymbol{u}) \nabla^2 h(g(\boldsymbol{u})) \nabla g(\boldsymbol{u})^\top \boldsymbol{v} + \nabla h(g(\boldsymbol{u}))^\top \nabla g(\boldsymbol{u})^\top \boldsymbol{v} = \sum_{t=0}^{\tau-1} q_{h_t}^{x_t, u_t}(y_t, v_t) + q_{h_\tau}^{x_\tau}(y_\tau). \tag{24}
$$

Hence, plugging Eq. (24) and Eq. (23) into Eq. (20) we get the claim for the Gauss-Newton oracle.

For the Newton oracle, denoting $\boldsymbol{\mu} = \nabla_{\boldsymbol{x}} h(\boldsymbol{x}, \boldsymbol{u}) = (\nabla_{x_1} h_1(x_1, u_1); \ldots; \nabla_{x_{\tau-1}} h_{\tau-1}(x_{\tau-1}, u_{\tau-1}); \nabla h_\tau(x_\tau))$, and defining adjoint variables λ_t as

$$
\lambda_\tau = \nabla h_\tau(x_\tau) \qquad \lambda_t = \nabla_{x_t} h_t(x_t, u_t) + \nabla_{x_t} f_t(x_t, u_t) \lambda_{t+1} \qquad \text{for } t \in \{1, \ldots, \tau-1\}.
$$

we have, as in the proof of Lemma 3.2,

$$
\begin{aligned}
\nabla^2 \phi(\boldsymbol{u})[\boldsymbol{v}, \boldsymbol{v}, \nabla_{\boldsymbol{x}} h(\boldsymbol{x}, \boldsymbol{u})] &= \sum_{t=0}^{\tau-1} \nabla^2 \phi_{t+1}(\boldsymbol{u})[\boldsymbol{v}, \boldsymbol{v}, \mu_{t+1}] \\
&= \sum_{t=0}^{\tau-1} \Big(\nabla^2_{x_t x_t} f_t(x_t, u_t)[y_t, y_t, \lambda_{t+1}] + \nabla^2_{u_t u_t} f_t(x_t, u_t)[v_t, v_t, \lambda_{t+1}] \\
&\qquad + 2\nabla^2_{x_t u_t} f_t(x_t, u_t)[y_t, v_t, \lambda_{t+1}]\Big).
\end{aligned} \tag{25}
$$

Hence, plugging Eq. (24), Eq. (25) and Eq. (23) into Eq. (21) we get the claim for the Newton oracle. □

From an optimization viewpoint, gradient, Gauss-Newton or Newton oracles are considered as black-boxes. Second order methods such as Gauss-Newton or Newton methods are generally considered to be too computationally expensive for optimizing problems in high dimension because they a priori reuire to solve a linear system at a cubic cost in the dimension of the problem. Here, the dimension of the problem in the control variables is τn_u, with n_u, the dimension of the control variables, usually small (see the numerical examples in Sec. 10), but τ, the number of time steps, potentially large if, e.g., the discretization time step used to define (1) from a continuous time control problem is small while the original time length of the continuous time control problem is large. A cubic cost w.r.t. the number of time steps τ is then a priori prohibitive.

A closer look at the implementation of all above oracles (13), (14), (15), shows that they all amount to solving linear quadratic control problems as presented in Lemma 4.1. Hence they can be solved by a dynamic programming approach detailed in Sec. 4.2 at a cost linear w.r.t. the number of time steps τ. In other words, the dynamical structure of the problem renders second order optimization oracles such as Gauss-Newton or Newton almost as computationally cheap as gradient oracles as long as n_u is small. This observation was done by Pantoja (1988); Dunn and Bertsekas (1989) for a Newton step and Sideris and Bobrow (2005) for a Gauss-Newton step. Wright (1990) also presented how sequential quadratic programming methods can naturally be cast in a similar way. Lemma 4.1 casts all classical optimization oracles in a same formulation.

4.2 Implementation

Given Lemma 4.1, classical optimization oracles for objectives of the form

$$\mathcal{J}(\boldsymbol{u}) = h \circ g(\boldsymbol{u}), \quad \text{where} \quad h(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{\tau-1} h_t(x_t, u_t) + h_\tau(x_\tau), \quad g(\boldsymbol{u}) = (f^{[\tau]}(\bar{x}_0, \boldsymbol{u}), \boldsymbol{u}).$$

with $f^{[\tau]}(\bar{x}_0, \boldsymbol{u})$ the control of τ dynamics $(f_t)_{t=0}^{\tau-1}$ defined in Def. 3.1, can be implemented by (i) instantiating the linear-quadratic control problem (16) with the chosen approximations, (ii) solving the linear-quadratic control problem (16) by a dynamic procedure as detailed in Sec. 2. Precisely, their implementation can be split in the three following phases.

1. Forward pass: All oracles start by gathering the information necessary for the step in a forward pass that takes the generic form of Algo. 5 and can be summarized as

$$\mathcal{J}(\boldsymbol{u}), (m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau} = \mathrm{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f, o_h)$$

   that compute the objective $\mathcal{J}(\boldsymbol{u})$ associated to the given sequence of controls $\boldsymbol{u}$ and record approximations $(m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}$ of the dynamics and the costs up to the orders o_f and o_h, respectively, provided as inputs.

2. Backward pass: Once approximations of the dynamics have been computed, a backward pass on the corresponding linear quadratic control problem (16) can be done as in the linear quadratic case presented in Sec. 2. The backward passes of the gradient oracle in Algo. 6, the Gauss-Newton oracle in Algo. 7 and the Newton oracle in Algo. 8 take generally the form

$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \mathrm{Backward}((m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}, \nu).$$

   Namely, they take as input a regularization $\nu \geq 0$ and some approximations of the dynamics and the costs $(m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}$ computed in a forward pass, and return a set of policies and the final cost-to-go corresponding to the subproblem (16).

3. Roll-out pass: Given the output of a backward pass defined above, the oracle is computed by rolling-out the policies along the linear trajectories defined in the subproblem (16). Formally, given a sequence of policies $(\pi_t)_{t=0}^{\tau-1}$, the oracles are then given as $\boldsymbol{v} = (v_0; \ldots; v_{\tau-1})$ computed, for $y_0 = 0$, by Algo. 11 as

$$\boldsymbol{v} = \mathrm{Roll}(y_0, (\pi_t)_{t=0}^{\tau-1}, (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}),$$

   with $(\pi_t)_{t=0}^{\tau-1}$ output by one of the backward passes in Algo. 6, Algo. 7 or Algo. 8.

Gradient, Gauss-Newton and Newton oracles are implemented by, respectively, Algo. 12, Algo. 13, Algo. 14. Additional line-searches are presented in Sec. 6. The computational scheme of a gradient oracle, i.e., gradient back-propagation, is illustrated in Fig. 3. The computational scheme of the Gauss-Newton oracle is presented in Fig. 4. Finally, the computational scheme of a Newton oracle is illustrated in Fig. 6.

Gradient back-propagation. For a gradient oracle (13), the procedure LQBP normally used to solve linear quadratic control problems simplifies to the procedure LBP presented in Algo. 3 that implements

$$\mathrm{LBP} : (\ell_t^f, \ell_t^h, c_{t+1}, \nu) \to \begin{pmatrix} c_t : x \to \min_{u \in \mathbb{R}^{n_u}} \left\{ \ell_t^h(x,u) + c_{t+1}(\ell_t^f(x,u)) + \frac{\nu}{2}\|u\|_2^2 \right\} \\ \pi_t : x \to \arg\min_{u \in \mathbb{R}^{n_u}} \left\{ \ell_t^h(x,u) + c_{t+1}(\ell_t^f(x,u)) + \frac{\nu}{2}\|u\|_2^2 \right\} \end{pmatrix}, \tag{26}$$

for linear functions $\ell_t^f, \ell_t^h, c_{t+1}$. Plugging in Algo. 3 the linearizations of the dynamics and the costs, we get that the gradient oracle in Algo. 6 computes affine cost-to-go functions of the form $c_t(y_t) = j_t^\top y_t + j_t^0$ with

$$j_\tau = \nabla h_\tau(x_\tau), \quad j_t = \nabla_{x_t} h_t(x_t, u_t) + \nabla_{x_t} f_t(x_t, u_t) j_{t+1} \quad \text{for } t \in \{0, \ldots, \tau-1\}.$$

Moreover, the policies are constant, i.e., $\pi_t(y_t) = k_t$, with

$$k_t = -\nu^{-1}(\nabla_{u_t} h_t(x_t, u_t) + \nabla_{u_t} f_t(x_t, u_t) j_{t+1}) = -\nu^{-1} \nabla_{u_t}(h \circ g)(\boldsymbol{u}).$$

The roll-out of these policies is independent of the dynamics and output directly the gradient up to a factor $-\nu^{-1}$. Note that we naturally retrieve the gradient back-propagation algorithm.

Simplifications. Some simplifications can be done in the implementations of the oracles. The gradient oracle can directly return the values of the gradient without the need of a roll-out phase. For the Gauss-Newton oracle, if there is no intermediate cost ($h_t = 0$ for $t \in \{0, \ldots, \tau-1\}$), the oracle can be computed by solving the dual subproblem by making calls to an automatic differentiation procedure (see, e.g., (Roulet et al., 2019)). For the Newton oracle, the quadratic approximations of the dynamics do not need to be stored and can simply be computed in the backward pass by computing the second derivative of $f_t^\top \lambda_{t+1}$ on x_t, u_t as explained in Sec. 7.

5 Differential Dynamic Approaches

The original Differential Dynamic Programming algorithm was developed by Jacobson and Mayne (1970) and revisited by, e.g., Mayne and Polak (1975); Murray and Yakowitz (1984); Liao and Shoemaker (1992); Tassa et al. (2014). The reader can verify from the aforementioned citations that our presentation matches the original formulation in, e.g., the quadratic case, while offering a larger perspective on the method that incorporate, e.g., linear quadratic approximations.

5.1 Principle

Denoting h the total cost as in (8) and $f^{[\tau]}$ the control in τ dynamics $(f_t)_{t=0}^{\tau-1}$, Differential Dynamic Programming (DDP) oracles consist in solving approximately

$$\min_{\boldsymbol{v} \in \mathbb{R}^{\tau n_u}} h(f^{[\tau]}(\bar{x}_0, \boldsymbol{u} + \boldsymbol{v}), \boldsymbol{u} + \boldsymbol{v}),$$

by means of a dynamic programming procedure and use the resulting policies to update the current sequence of controllers. For a consistent presentation with the classical optimization oracles presented in Sec. 4, we consider the regularized version of the DDP oracles, that is,

$$\min_{\boldsymbol{v} \in \mathbb{R}^{\tau n_u}} h(f^{[\tau]}(\bar{x}_0, \boldsymbol{u} + \boldsymbol{v}), \boldsymbol{u} + \boldsymbol{v}) + \frac{\nu}{2}\|\boldsymbol{v}\|_2^2, \tag{27}$$

for some regularization $\nu \geq 0$.

The objective in problem (27) can be rewritten as

$$h(f^{[\tau]}(\bar{x}_0, \boldsymbol{u} + \boldsymbol{v}), \boldsymbol{u} + \boldsymbol{v}) = h(f^{[\tau]}(\bar{x}_0, \boldsymbol{u})) + \delta_h^{f^{[\tau]}(\bar{x}_0, \boldsymbol{u})}(\delta_{f^{[\tau]}}^{\bar{x}_0, \boldsymbol{u}}(0, \boldsymbol{v}), \boldsymbol{v}), \tag{28}$$

where for a function f, δ_f^x is the finite difference expression of f around x as defined in the notations in Eq. (2). In particular, $\delta_{f^{[\tau]}}^{\bar{x}_0, \boldsymbol{u}}(0, \boldsymbol{v})$ is the trajectory defined by the finite differences of the dynamics given as

$$\delta_{f_t}^{x_t, u_t}(y_t, v_t) = f_t(x_t + y_t, u_t + v_t) - f_t(x_t, u_t).$$

The dynamic programming approach is then applied on the above dynamics. Namely, the goal is to solve

$$\min_{\substack{v_0,\ldots,v_{\tau-1}\in\mathbb{R}^{n_u}\\ y_0,\ldots,y_\tau\in\mathbb{R}^{n_x}}} \sum_{t=0}^{\tau-1} \delta_{h_t}^{x_t,u_t}(y_t,v_t) + \frac{\nu}{2}\|v_t\|_2^2 + \delta_{h_\tau}^{x_\tau}(y_\tau) \tag{29}$$
$$\text{s.t.} \quad y_{t+1} = \delta_{f_t}^{x_t,u_t}(y_t,v_t) \quad \text{for } t\in\{0,\ldots,\tau-1\}, \quad y_0 = 0,$$

by a dynamic programming approach. Denote then c_t^* the cost-to-go functions associated to problem (29) for $t\in\{0,\ldots\tau\}$. These cost-to-go functions satisfy the recursive equation

$$c_t^*(y_t) = \min_{v_t\in\mathbb{R}^{n_u}} \left\{\delta_{h_t}^{x_t,u_t}(y_t,v_t) + \frac{\nu}{2}\|v_t\|_2^2 + c_{t+1}^*(\delta_{f_t}^{x_t,u_t}(y_t,v_t))\right\}, \tag{30}$$

starting from $c_\tau^* = \delta_{h_\tau}^{x_\tau}$ and such that our objective is to compute $c_0^*(0)$. Since the dynamics $\delta_{f_t}^{x_t,u_t}$ are not linear and the costs $\delta_{h_t}^{x_t,u_t}$ are not quadratic, there is no analytical solution for the subproblem (30). To circumvent this issue, the cost-to-go functions are approximated as $c_t^*(y_t) \approx c_t(y_t)$, where c_t is computed from approximations of the dynamics and the costs. The approximation is done around the nominal value of the subproblem (29) which is $\boldsymbol{v}=0$ and corresponds to $\boldsymbol{y}=0$ and no change of the original objective in (28).

Denoting m_f an expansion of a function f around the origin such that $f(x)\approx f(0)+m_f(x)$, the cost-to-go functions are computed with a procedure

$$\widehat{\mathrm{BP}} : \delta_t^f, \delta_t^h, c_{t+1} \to \begin{pmatrix} c_t : y \to \min_{v\in\mathbb{R}^{n_u}} \left\{(\delta_t^h + c_{t+1}\circ\delta_t^f)(0,0) + m_{\delta_t^h}(y,v) + m_{c_{t+1}\circ\delta_t^f}(y,v) + \frac{\nu}{2}\|v\|_2^2\right\}, \\ \pi_t : y \to \arg\min_{v\in\mathbb{R}^{n_u}} \left\{m_{\delta_t^h}(y,v) + m_{c_{t+1}\circ\delta_t^f}(y,v) + \frac{\nu}{2}\|v\|_2^2\right\} \end{pmatrix}, \tag{31}$$

applied to the finite differences $\delta_{f_t}^{x_t,u_t} \to \delta_t^f$ and $\delta_{h_t}^{x_t,u_t} \to \delta_t^h$. A differential dynamic procedure computes then a sequence of policies by iterating in a backward pass, starting from $c_\tau = m_{\delta_{h_\tau}^{x_\tau}}$,

$$c_t, \pi_t = \widehat{\mathrm{BP}}(\delta_{f_t}^{x_t,u_t}, \delta_{h_t}^{x_t,u_t}, c_{t+1}) \quad \text{for } t\in\{\tau-1,\ldots,0\}. \tag{32}$$

Given a set of policies, an approximate solution is given by rolling-out the policies along the dynamics defining problem (29), i.e., by computing $v_0,\ldots,v_{\tau-1}$ as

$$v_t = \pi_t(y_t), \qquad y_{t+1} = \delta_{f_t}^{x_t,u_t}(y_t,v_t) = f_t(x_t+y_t, u_t+v_t) - f_t(x_t,u_t) \qquad \text{for } t=0,\ldots,\tau-1. \tag{33}$$

The main difference with the classical optimization oracles relies a priori in the computation of the policies in (32) detailed below and in the roll-out pass that uses the finite differences of the dynamics. Note that, while only the non-constant parts of the cost-to-go functions are useful to compute the policies, the overall procedure computes also the constant part of the cost-to-go functions. The latter is used in line-search procedures as detailed in Sec. 6.

5.2 Detailed Derivations of the Backward Passes

5.2.1 Linear Approximation

If we consider a linear approximation for the composition of the cost-to-go function and the dynamics, we have

$$m_{c_{t+1}\circ\delta_{f_t}^{x,u}} = \ell^{(0,0)}_{c_{t+1}\circ\delta_f^{x,u}} = \ell^{\delta_f^{x,u}(0,0)}_{c_{t+1}} \circ \ell^{(0,0)}_{\delta_f^{x,u}} = \ell_{c_{t+1}} \circ \ell_f^{x,u}.$$

Plugging this model into (31) and using linear approximations of the costs, the recursion (32) amounts to computing, starting from $c_\tau = \ell_{\delta_{h_\tau}^{x_\tau,u_\tau}} = \ell_{h_\tau}^{x_\tau,u_\tau}$,

$$\begin{aligned} c_t(y) &= \min_{v\in\mathbb{R}^{n_u}} \delta_{h_t}^{x_t,u_t}(0,0) + \ell_{\delta_{h_t}^{x_t,u_t}}(y,v) + c_{t+1}(\delta_{f_t}^{x_t,u_t}(0,0)) + \ell_{c_{t+1}}(\ell_{f_t}^{x_t,u_t}(y,v)) + \frac{\nu}{2}\|v\|_2^2, \\ &= \min_{v\in\mathbb{R}^{n_u}} \ell_{h_t}^{x_t,u_t}(y,v) + c_{t+1}(\ell_{f_t}^{x_t,u_t}(y,v)) + \frac{\nu}{2}\|v\|_2^2, \end{aligned}$$

where in the last line we used that the cost-to-go functions c_t are necessarily affine, s.t. $c_{t+1}(y) = c_{t+1}(0) + \ell_{c_{t+1}}(y)$. We retrieve then the same recursion as the one used for a gradient oracle and the output policies are then the same. Since the computed policies are constant, they are not affected by the dynamics along which a roll-out phase is performed. In other words, the oracle returned by using linear approximations in a differential dynamic programming approach is just a gradient oracle.

5.2.2 Linear Quadratic Approximation

If we consider a linear quadratic approximation for the composition of the cost-to-go function and the dynamics, we have

$$m_{c_{t+1}\circ\delta_{f^{x,u}}} = q_{c_{t+1}}^{\delta_f^{x,u}(0,0)} \circ \ell^{(0,0)}_{\delta_f^{x,u}} = q_{c_{t+1}} \circ \ell_f^{x,u}.$$

Plugging this model into (31) and using quadratic approximations of the costs, the recursion (32) amounts to computing, starting from $c_\tau = q_{\delta_{h_\tau}^{x_\tau,u_\tau}} = q_{h_\tau}^{x_\tau,u_\tau}$,

$$\begin{aligned} c_t(y) &= \min_{v\in\mathbb{R}^{n_u}} \delta_{h_t}^{x_t,u_t}(0,0) + q_{\delta_{h_t}^{x_t,u_t}}(y,v) + c_{t+1}(\delta_{f_t}^{x_t,u_t}(0,0)) + q_{c_{t+1}}^{\delta_f^{x,u}(0,0)} \circ \ell^{(0,0)}_{\delta_f^{x,u}}(y,v) + \frac{\nu}{2}\|v\|_2^2 \\ &= \min_{v\in\mathbb{R}^{n_u}} q_{h_t}^{x_t,u_t}(y,v) + c_{t+1}(0) + q_{c_{t+1}}(\ell_{f_t}^{x_t,u_t}(y,v)) + \frac{\nu}{2}\|v\|_2^2. \end{aligned} \tag{34}$$

If the costs h_t are convex for all t and $q_{h_t}^{x_t,u_t}(y,\cdot) + \frac{\nu}{2}\|\cdot\|_2^2$ is strongly convex for all t and all y, then the cost-to-go functions c_t are convex quadratics for all t, i.e., $c_{t+1}(y) = c_{t+1}(0) + q_{c_{t+1}}(y)$. In that case, the recursion (34) simplifies as

$$c_t(y) = \min_{v\in\mathbb{R}^{n_u}} q_{h_t}^{x_t,u_t}(y,v) + c_{t+1}(\ell_{f_t}^{x_t,u_t}(y,v)) + \frac{\nu}{2}\|v\|_2^2, \tag{35}$$

and the policies are given by the minimizer of Eq. (35). The recursion (35) is then the same as the recursion done when computing a Gauss-Newton oracle. Namely, the backward pass in this case is the backward pass of a Gauss-Newton oracle. Though the output policies are the same, the output of the oracle will differ since the roll-out phase does not follow the linearized trajectories in the differential dynamic programming approach. The computational scheme of a DDP approach with linear quadratic approximations presented in Fig. 5 is then almost the same as the one of a Gauss-Newton oracle presented in Fig. 4, except that in the roll-out phase the linear approximations of the function are replaced by the finite differences of the objective.

5.2.3 Quadratic Approximation

If we consider a quadratic approximation for the composition of the cost-to-go function and the dynamics, we get

$$m_{c_{t+1}\circ\delta_f^{x,u}} = q_{c_{t+1}\circ\delta_f^{x,u}} = \frac{1}{2}\nabla^2 f(x,u)[\cdot,\cdot,\nabla c_{t+1}(0)] + q_{c_{t+1}} \circ \ell_f^{x,u},$$

where $\nabla^2 f(x,u)[\cdot,\cdot,\lambda]$ is defined in (18). Plugging this model into (31) and using quadratic approximations of the costs, the recursion (32) amounts to, starting from $c_\tau = q_{\delta_{h_\tau}^{x_\tau,u_\tau}} = q_{h_\tau}^{x_\tau,u_\tau}$,

$$\begin{aligned} c_t(y) &= \min_{v\in\mathbb{R}^{n_u}} \delta_{h_t}^{x_t,u_t}(0,0) + q_{\delta_{h_t}^{x_t,u_t}}(y,v) + c_{t+1}(\delta_{f_t}^{x_t,u_t}(0,0)) + q_{c_{t+1}\circ\delta_{f_t}^{x,u}}(y,v) + \frac{\nu}{2}\|v\|_2^2 \\ &= \min_{v\in\mathbb{R}^{n_u}} q_{h_t}^{x_t,u_t}(y,v) + c_{t+1}(0) + q_{c_{t+1}} \circ \ell_{f_t}^{x_t,u_t}(y,v) + \frac{1}{2}\nabla^2 f_t(x_t,u_t)[\cdot,\cdot,\nabla c_{t+1}(0)](y,v) + \frac{\nu}{2}\|v\|_2^2. \end{aligned} \tag{36}$$

Provided that the costs are convex and that $q_{h_t}^{x_t,u_t}(y,\cdot) + \frac{1}{2}\nabla^2 f_t(x_t,u_t)[\cdot,\cdot,\nabla c_{t+1}(0)](y,\cdot) + \frac{\nu}{2}\|\cdot\|_2^2$ is strongly convex for all t and all y, the cost-to-go functions c_t are convex quadratics for all t. In that case, the recursion (36) simplifies as

$$c_t(y) = \min_{v\in\mathbb{R}^{n_u}} q_{h_t}^{x_t,u_t}(y,v) + c_{t+1}(\ell_{f_t}^{x_t,u_t}(y,v)) + \frac{1}{2}\nabla^2 f_t(x_t,u_t)[\cdot,\cdot,\nabla c_{t+1}(0)](y,v) + \frac{\nu}{2}\|v\|_2^2, \tag{37}$$

and the policies are given by the minimizer of Eq. (37). The overall backward pass is detailed in Algo. 9.

Compared to the backward pass of the Newton step in Algo. 8, we note that the additional cost derived from the curvatures of the dynamics are not computed the same way. Namely, the Newton oracle computes this additional cost by using back-propagated adjoint variables in Eq. (17), while in the differential dynamic programming approach the additional cost is directly defined through the previously computed cost-to-go function. Fig. 7 illustrates the computational scheme of the implementation of DDP with quadratic approximations and can be compared to the computational scheme of the Newton oracle in Fig. 6.

Note that, while we used second order Taylor expansions for the compositions and the costs, the approximate cost-to-go-functions c_t are *not* second-order Taylor expansion of the true cost-to-go functions c_t^*, except for c_τ. Indeed, c_t is computed as an approximate solution of the Bellman equation. The true Taylor expansion of the cost-to-go function requires the gradient and the Hessian of the cost and the dynamic in Eq. (36) computed at the minimizer of the subproblem. Here, since we only use an approximation of the minimizer, we do not have access to the true gradient and Hessian of the cost-to-go function.

5.3 Implementation

The implementation of the differential dynamic programming approaches follow the same steps as the ones given for classical optimization oracles as detailed below. The implementation of a DDP oracle with linear quadratic approximations is given in Algo. 15 and illustrated in Fig. 5. The implementation of a DDP oracle with quadratic approximations is given in Algo. 16 and illustrated in Fig. 7.

1. Forward pass: As for the classical optimization methods, the forward pass is provided in Algo. 5 that gathers the information necessary for the backward pass. Namely, the oracle starts with Algo. 5 and computes

$$\mathcal{J}(\boldsymbol{u}), (m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau} = \mathrm{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f, o_h),$$

   where o_f and o_h define the order of the approximations used for the dynamics and the costs respectively.

2. Backward pass: As for the classical optimization oracles, the backward pass can generally be written

$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \mathrm{Backward}((m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}, \nu),$$

   If linear approximations are used, the backward pass is given in Algo. 6, if linear-quadratic approximations are used, the backward pass is given in Algo. 7 and if quadratic approximations are used, the backward pass is given in Algo. 9.

3. Roll-out pass: The roll-out phase differs by using the dynamics of the problem (29) rather than the linearized dynamics. Formally, given a sequence of policies $(\pi_t)_{t=0}^{\tau-1}$, the oracles are then given as $\boldsymbol{v} = (v_0; \ldots; v_{\tau-1})$ computed, for $y_0 = 0$, by Algo. 11 as

$$\boldsymbol{v} = \mathrm{Roll}(y_0, (\pi_t)_{t=0}^{\tau-1}, (\delta_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}),$$

   where $\delta_{f_t}^{x_t,u_t}(y_t, v_t) = f_t(x_t + y_t, u_t + v_t) - f_t(x_t, u_t)$.

6 Line-searches

So far, we defined procedures that, given a command and some regularization parameter, outputs a direction that minimizes an approximation of the objective or approximately minimizes a shifted objective. Given access to such procedures, the next command can be computed in several ways. The main criterion is to ensure that the next command decreases the value of the objective, which is generally done by a line-search.

In the following, we only consider oracles based on linear quadratic or quadratic approximations of the objective such as Gauss-Newton and Newton, and refer the reader to (Nocedal and Wright, 2006) for classical line-searches on gradient descent.

6.1 Principle for Classical Optimization Oracles

We start by considering the implementation of line-searches for classical optimization oracles which can again exploit the dynamical structure of the problem and are mimicked by differential dynamic programming approaches. We consider, as in Sec. 4, that we have access to an oracle for an objective $\mathcal{J}$, that, given a command $\boldsymbol{u} \in \mathbb{R}^{\tau n_u}$ and any regularization $\nu \geq 0$, outputs

$$\mathrm{Oracle}_\nu(\mathcal{J})(\boldsymbol{u}) = \operatorname*{arg\,min}_{\boldsymbol{v} \in \mathbb{R}^{\tau n_u}} m_{\mathcal{J}}^{\boldsymbol{u}}(\boldsymbol{v}) + \frac{\nu}{2}\|\boldsymbol{v}\|_2^2, \tag{38}$$

where $m_{\mathcal{J}}^{\boldsymbol{u}}$ is a linear quadratic or quadratic expansion of the objective $\mathcal{J}$ around $\boldsymbol{u}$ s.t. $\mathcal{J}(\boldsymbol{u}+\boldsymbol{v}) \approx \mathcal{J}(\boldsymbol{u}) + m_{\mathcal{J}}^{\boldsymbol{u}}(\boldsymbol{v})$. Given such oracle, we can define a new candidate command that decreases the value of the objective in several ways.

6.1.1 Descent Direction.

The next iterate can be computed along the direction provided by the oracle, as long as this direction is a descent direction. Namely, the next iterate can be computed as

$$\boldsymbol{u}^{\text{next}} = \boldsymbol{u} + \gamma \boldsymbol{v}, \quad \text{with } \boldsymbol{v} = \text{Oracle}_{\nu}(\mathcal{J})(\boldsymbol{u}) \text{ for } \nu \geq 0 \text{ s.t. } \nabla \mathcal{J}(\boldsymbol{u})^\top \boldsymbol{v} < 0, \tag{39}$$

where the stepsize γ is chosen to satisfy, e.g., an Armijo condition, that is,

$$\mathcal{J}(\boldsymbol{u} + \gamma \boldsymbol{v}) \leq \mathcal{J}(\boldsymbol{u}) + \frac{\gamma}{2} \nabla \mathcal{J}(\boldsymbol{u})^\top \boldsymbol{v}. \tag{40}$$

In this case, the search is usually initialized at each step with $\gamma = 1$. If condition (40) is not satisfied for $\gamma = 1$, the stepsize is decreased by a factor $\rho_{\text{dec}} < 1$ until condition (40) is satisfied. If a stepsize $\gamma = 1$ is accepted, then the linear quadratic or quadratic algorithms may exhibit a quadratic local convergence (Nocedal and Wright, 2006). Alternative line-search criterions such as Wolfe's condition or trust-region methods can also be implemented (Nocedal and Wright, 2006). We rather consider regularized steps since second-order oracles in nonlinear control have a reasonable computational complexity w.r.t. the leading dimension of the problem as explained in Sec. 7.

6.1.2 Regularized Steps

Given a current iterate $\boldsymbol{u} \in \mathbb{R}^{\tau n_u}$, we can find a regularization such that the current command plus the command output by the oracle decreases the objective. Namely, the next command can be computed as

$$\boldsymbol{u}^{\text{next}} = \boldsymbol{u} + \boldsymbol{v}^\gamma, \quad \text{where} \quad \boldsymbol{v}^\gamma = \text{Oracle}_{1/\gamma}(\mathcal{J})(\boldsymbol{u}) = \operatorname*{arg\,min}_{\boldsymbol{v} \in \mathbb{R}^{\tau n_u}} m_{\mathcal{J}}^{\boldsymbol{u}}(\boldsymbol{v}) + \frac{1}{2\gamma} \|\boldsymbol{v}\|_2^2, \tag{41}$$

where the parameter $\gamma > 0$ acts as a stepsize that controls how large should be the step (the smaller the γ, the smaller the step $\boldsymbol{v}^\gamma$). The stepsize γ can then be chosen to satisfy

$$\mathcal{J}(\boldsymbol{u} + \boldsymbol{v}^\gamma) \leq \mathcal{J}(\boldsymbol{u}) + m_{\mathcal{J}}^{\boldsymbol{u}}(\boldsymbol{v}^\gamma) + \frac{1}{2\gamma} \|\boldsymbol{v}^\gamma\|_2^2, \tag{42}$$

which ensures a sufficient decrease of the objective to, e.g., prove convergence to stationary points (Roulet et al., 2019). In practice, as for the line-search on the descent direction, given an initial stepsize for the iteration, the stepsize is either selected or reduced by a factor ρ_{dec} until condition (42) is satisfied. However, here, we initialize the stepsize at each iteration as $\rho_{\text{inc}} \gamma_{prev}$ where γ_{prev} is the stepsize selected at the previous iteration and $\rho_{\text{inc}} > 1$ is an increasing factor. By trying a larger stepsize at each iteration, we may benefit from larger steps in some regions of the optimization path. Note that such approach is akin to trust region methods which increase the radius of the trust region at each iteration depending on the success of each iteration (Nocedal and Wright, 2006).

In practice, we observed that, when using regularized steps, acceptable stepsizes for condition (42) tend to be arbitrarily large as the iterations increase. Namely, we tried choosing $\rho_{\text{inc}} = 10$ and observed that the acceptable stepsizes tended to plus infinity with such procedure. To better capture this tendency, we consider regularizations that may depend on the current state and of the form $\nu(\boldsymbol{u}) = \bar{\nu} \|\nabla h(\boldsymbol{x}, \boldsymbol{u})\|_2$, i.e., stepsizes of the form $\gamma(\boldsymbol{u}) = \bar{\gamma} / \|\nabla h(\boldsymbol{x}, \boldsymbol{u})\|_2$. The line-search is then performed on $\bar{\gamma}$ only. Intuitively, as we are getting closer to a stationary point, quadratic models are getting more accurate to describe the objective. By scaling the regularization with respect to $\|\nabla h(\boldsymbol{x}, \boldsymbol{u})\|_2$, which is a measure of stationarity, we may better capture such behavior. Note that for $\nu = 0$, we retrieve the iteration with a descent direction of stepsize $\gamma = 1$ described above.

6.2 Implementation

6.2.1 Descent Direction

Condition (40) can be computed directly from the knowledge of a gradient oracle and the chosen oracle (such as Gauss-Newton or Newton). We present here the implementation of the line-search in terms of the dynamical structure of the problem. Denote

$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \text{Backward}((m_{f_t}^{x_t, u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t, u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}, \nu)$$

the policies and the value of the cost-to-go function output by the backward pass of the considered oracle, i.e., Gauss-Newton or Newton.

By definition, $c_0(0)$ is the minimum of the corresponding linear quadratic control problem (16). Moreover, the linear quadratic control problem can be summarized as a quadratic problem of the form $\min_{\boldsymbol{v}} m_{\mathcal{J}}(\boldsymbol{v}) + \frac{\nu}{2}\|\boldsymbol{v}\|_2^2 = \min_{\boldsymbol{v}} \frac{1}{2}\boldsymbol{v}^\top(Q+\nu\,\mathrm{I})\boldsymbol{v} + \nabla\mathcal{J}(\boldsymbol{u})^\top\boldsymbol{v}$ with Q a quadratic that is either the Hessian of $\mathcal{J}$ for a Newton oracle or an approximation of it for a Gauss-Newton oracle. Therefore, we have that, for $\boldsymbol{v} \in \mathbb{R}^{\tau n_u}$ a Newton or a Gauss-Newton oracle,

$$\frac{1}{2}\nabla\mathcal{J}(\boldsymbol{u})^\top\boldsymbol{v} = -\frac{1}{2}\nabla\mathcal{J}(\boldsymbol{u})^\top(Q+\nu\,\mathrm{I})^{-1}\nabla\mathcal{J}(\boldsymbol{u}) = \min_{\boldsymbol{v}\in\mathbb{R}^{\tau n_u}} m_{\mathcal{J}}(\boldsymbol{v}) + \frac{\nu}{2}\|\boldsymbol{v}\|_2^2 = c_0(0).$$

Therefore the right-hand part of condition (40) can be given by the value of the cost-to-go function $c_0(0)$. On the other hand, sequence of controllers of the form $\gamma\boldsymbol{v}$ can be defined by modifying the policies output in the backward pass as shown in the following lemma adapted from Liao and Shoemaker (1992, Theorem 1).

Lemma 6.1. *Given a sequence of affine policies $(\pi_t)_{t=0}^{\tau-1}$, linear dynamics $(\ell_t)_{t=0}^{\tau-1}$ and an initial state $y_0 = 0$, denote $\boldsymbol{v}^* = \mathrm{Roll}(y_0, (\pi_t)_{t=0}^{\tau-1}, (\ell_t)_{t=0}^{\tau-1})$ and $\pi_t^\gamma : y \to \gamma\pi_t(0) + \nabla\pi_t(0)^\top y$ for $t = 0, \ldots, \tau-1$. We have that*

$$\gamma\boldsymbol{v}^* = \boldsymbol{v}^\gamma, \quad \textit{where} \quad \boldsymbol{v}^\gamma = \mathrm{Roll}(y_0, (\pi_t^\gamma)_{t=0}^{\tau-1}, (\ell_t)_{t=0}^{\tau-1}).$$

Proof. Define $(y_t^\gamma)_{t=0}^{\tau-1}$ as $y_{t+1}^\gamma = \ell_t(y_t^\gamma, \pi_t(y_t^\gamma))$ for $t \in \{0,\ldots,\tau-1\}$ with $y_0^\gamma = 0$. We have that y_1^γ is linear w.r.t. γ. Proceeding by induction, we have that y_t^γ is linear w.r.t. γ using the form of π_t^γ and the fact that ℓ_t is linear. Therefore $v_t^\gamma = \pi_t^\gamma(y_t^\gamma)$ is linear w.r.t. γ which gives the claim. □

Therefore, computing the next sequence of controllers by moving along a descent direction as in (39) according to an Armijo condition (40) amounts to compute, with Algo. 17,

$$\begin{aligned}
&\boldsymbol{u}^{\mathrm{next}} = \mathrm{LineSearch}(\boldsymbol{u}, (h_t)_{t=0}^{\tau}, (f_t)_{t=0}^{\tau-1}, (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, \mathrm{Pol}),\\
\text{where}\quad &\mathrm{Pol}:\gamma \to \begin{pmatrix} (\pi_t^\gamma: & y \to \gamma\pi_t(0) + \nabla\pi_t(0)^\top y)_{t=0}^{\tau-1} \\ c_0^\gamma: & y \to \gamma c_0(y) \end{pmatrix}\\
&(\pi_t)_{t=0}^{\tau-1}, c_0 = \mathrm{Backward}((m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}, \nu), \text{ for } \nu \geq 0 \text{ s.t. } c_0(0) < 0,
\end{aligned}$$

where $\mathrm{Backward} \in \{\mathrm{Backward}_{\mathrm{GN}}, \mathrm{Backward}_{\mathrm{NE}}\}$ is given in Algo. 7 or Algo. 8.

In practice, in our implementation of the backward passes in Algo. 7, Algo. 8, the returned initial cost-to-go function is either negative if the step is well defined or infinite if it is not. To find a regularization that ensures a descent direction, i.e., $c_0(0) < 0$, it suffices thus to find a feasible step. In our implementation, we first try to compute a descent direction without regularization ($\nu = 0$), then try a small regularization $\nu = 10^{-6}$, which we increase by 10 until a finite negative cost-to-go function $c_0(0)$ is returned. See Algo. 18 for an instance of such implementation.

From the above discussion it is clear that one iteration of the Iterative Linear Quadratic Regulator algorithm described in Sec. 2.3 uses a Gauss-Newton oracle without regularization to move along the direction of the oracle by using an Armijo condition. The overall iteration is given in Algo. 18, where we added a procedure to ensure moving along a descent direction. All other algorithms, with or without regularization can be written in a similar way using a forward, a backward pass and multiple roll-out phases until a next sequence of controllers is found.

6.2.2 Regularized Steps

For regularized steps, the line-search (42) requires to compute $m_{\mathcal{J}}^{\boldsymbol{u}}(v^\gamma) + \frac{1}{2\gamma}\|v^\gamma\|_2^2$. This is by definition the minimum of the sub-problem that is computed by dynamic programming. This minimum can therefore be accessed as $m_{\mathcal{J}}^{\boldsymbol{u}}(\boldsymbol{v}^\gamma) + \frac{1}{2\gamma}\|\boldsymbol{v}^\gamma\|_2^2 = c_0(0)$ for c_0 output by the backward pass with a regularization $\nu = 1/\gamma$. Overall, the next sequence of controls is then provided through the line-search procedure given in Algo. 17 as

$$\begin{aligned}
&\boldsymbol{u}^{\mathrm{next}} = \mathrm{LineSearch}(\boldsymbol{u}, (h_t)_{t=0}^{\tau}, (f_t)_{t=0}^{\tau-1}, (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, \mathrm{Pol}),\\
\text{where}\quad &\mathrm{Pol}:\gamma \to \mathrm{Backward}((m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}, 1/\gamma),
\end{aligned}$$

where $\mathrm{Backward} \in \{\mathrm{Backward}_{\mathrm{GN}}, \mathrm{Backward}_{\mathrm{NE}}\}$ is given in Algo. 7 or Algo. 8.

6.3 Line-searches for Differential Dynamic Programming Approaches

The line-search for DDP approaches as presented by, e.g., Liao and Shoemaker (1992, Sec. 2.2) based on (Jacobson and Mayne, 1970), mimic the ones done for the classical optimization oracles except that the policies are rolled-out on the original dynamics. Namely, the usual line-search consists in applying Algo. 17 as follows

$$\begin{aligned}
\boldsymbol{u}^{\text{next}} &= \text{LineSearch}(\boldsymbol{u}, (h_t)_{t=0}^{\tau}, (f_t)_{t=0}^{\tau-1}, (\delta_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, \text{Pol}) \\
\text{where} \quad &\text{Pol} : \gamma \to \begin{pmatrix} (\pi_t^\gamma : & y \to \gamma\pi_t(0) + \nabla\pi_t(0)^\top y)_{t=0}^{\tau-1}, \\ c_0^\gamma : & y \to \gamma c_0(y) \end{pmatrix} \\
&(\pi_t)_{t=0}^{\tau-1}, c_0 = \text{Backward}((m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}, \nu) \text{ for } \nu \geq 0 \text{ s.t. } c_0(0) < 0,
\end{aligned}$$

where $\text{Backward} \in \{\text{Backward}_{\text{GN}}, \text{Backward}_{\text{DDP}}\}$ is given by Algo. 7 or Algo. 9. As for the classical optimization oracles, a direction is first computed without regularization and if the resulting direction is not a descent direction a small regularization is added to ensure that $c_0(0) < 0$.

We also consider line-searches based on selecting an appropriate regularization. Namely, we consider line-searches of the form

$$\begin{aligned}
u_{\text{new}} &= \text{LineSearch}(\boldsymbol{u}, (h_t)_{t=0}^{\tau}, (f_t)_{t=0}^{\tau-1}, (\delta_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, \text{Pol}), \\
\text{where} \quad &\text{Pol} : \gamma \to \text{Backward}((m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}, 1/\gamma),
\end{aligned}$$

where $\text{Backward} \in \{\text{Backward}_{\text{GN}}, \text{Backward}_{\text{DDP}}\}$ is given by Algo. 7 or Algo. 9.

7 Computational Complexities

7.1 Basic Computational Complexities

We present in Table 1 the computational complexities of the algorithms following the implementations described in Sec. 4 and Sec. 5 and detailed in Sec. 9. We ignore the additional cost of the line-searches which require a theoretical analysis of the admissible stepsizes depending on the smoothness properties of the dynamics and the costs. We consider for simplicity that the cost of evaluating a function $f : \mathbb{R}^d \to \mathbb{R}^n$ is of the order of $O(nd)$, as it is the case if f is linear. For the computational complexities of the core operation of the backward pass, i.e, LQBP in Algo. 2 or LBP in Algo. 3, we simply give the leading computational complexities, which, in the case of LQBP, are the matrix multiplications and inversions.

The time complexities differ between using linear or quadratic approximations of the costs. In the latter case, matrices of size $n_u \times n_u$ need to be inverted and matrices of size $n_x \times n_x$ need to be multiplied. However, all oracles have a linear time complexity with respect to the horizon τ.

We note that the space complexities of the gradient descent and the Gauss-Newton method or the DDP approach with linear quadratic approximations are essentially the same. On the other hand, the space complexity of the Newton pass is a priori larger.

7.2 Improved Computational Complexities

The decomposition of each oracles between forward, backward and roll-out passes has the advantage to clarify the discrepancies between each approach. However, storing the linear or quadratic approximations of the costs or the dynamics may come at a prohibitive cost in terms of memory. A careful implementation of these oracles only requires to store in memory the function and the inputs given at each time-step. Namely, the forward pass can simply keep in memory h_t, f_t, x_t, u_t for $t \in \{0, \ldots, \tau\}$. The backward pass computes then, on the fly, the information necessary to compute the policies.

The time complexities of the forward pass corresponding to the computations of the gradients of the dynamics or the costs and Hessians of the costs remain the same a priori except that now they are incurred during the backward pass. A major difference lies in the computation of the quadratic information required in quadratic oracles such as a Newton oracle or a DDP oracle with quadratic approximations. Indeed, a closer look at Algo. 8 and Algo. 9 show that only the Hessians of scalar functions of the form $x, u \to f(x,u)^\top\lambda$ need to be computed, which come at a cost $(n_x + n_u)^2$. In comparison, the cost of computing the second order information of f is $O((n_x + n_u)^2 n_x)$. As an example, Algo. 10 presents an implementation of a Newton step using stored functions and inputs.

Time complexities of the forward pass in Algo. 5	
Function eval. ($o_f = o_h = 0$)	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{f_t}+\underbrace{n_x+n_u}_{h_t}\Big)=O(\tau({n_x}^2+n_xn_u))$
Linearization ($o_f = o_h = 1$)	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{f_t,\nabla f_t}+\underbrace{n_x+n_u}_{h_t,\nabla h_t}\Big)=O(\tau({n_x}^2+n_xn_u))$
Lin.-quad. ($o_f = 1, o_h = 2$)	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{f_t,\nabla f_t}+\underbrace{n_x+n_u}_{h_t,\nabla h_t}+\underbrace{{n_x}^2+{n_u}^2+n_xn_u}_{\nabla^2 h_t}\Big)=O(\tau(n_x+n_u)^2)$
Quad. ($o_f = o_h = 2$)	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{f_t,\nabla f_t}+\underbrace{({n_x}^2+{n_u}^2+n_xn_u)n_x}_{\nabla^2 f_t}+\underbrace{n_x+n_u}_{h_t,\nabla h_t}+\underbrace{{n_x}^2+{n_u}^2+n_xn_u}_{\nabla^2 h_t}\Big)=O(\tau n_x(n_x+n_u)^2)$

Space complexities of the forward pass in Algo. 5	
Function eval. ($o_f = o_h = 0$)	0
Linearization ($o_f = o_h = 1$)	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{\nabla f_t}+\underbrace{n_x+n_u}_{\nabla h_t}\Big)=O(\tau({n_x}^2+n_xn_u))$
Lin.-quad. ($o_f = 1, o_h = 2$)	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{\nabla f_t}+\underbrace{n_x+n_u}_{\nabla h_t}+\underbrace{{n_x}^2+{n_u}^2+n_xn_u}_{\nabla^2 h_t}\Big)=O(\tau(n_x+n_u)^2)$
Quad. ($o_f = o_h = 2$)	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{\nabla f_t}+\underbrace{({n_x}^2+{n_u}^2+n_xn_u)n_x}_{\nabla^2 f_t}+\underbrace{n_x+n_u}_{\nabla h_t}+\underbrace{{n_x}^2+{n_u}^2+n_xn_u}_{\nabla^2 h_t}\Big)=O(\tau n_x(n_x+n_u)^2)$

Time complexities of the backward passes in Algo. 6, 7, 8, 9 and the roll-out in Algo. 11	
GD	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{\text{Roll}}+\underbrace{{n_x}^2+n_xn_u}_{\text{LBP}}\Big)=O(\tau({n_x}^2+n_xn_u))$
GN/DDP-LQ	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{\text{Roll}}+\underbrace{{n_x}^3+{n_u}^3+{n_u}^2n_x}_{\text{LQBP}}\Big)=O(\tau(n_x+n_u)^3)$
NE/DDP-Q	$\tau\Big(\underbrace{{n_x}^2+n_xn_u}_{\text{Roll}}+\underbrace{{n_x}^3+{n_u}^3+{n_u}^2n_x}_{\text{LQBP}}+\underbrace{({n_x}^2+{n_u}^2+n_xn_u)n_x}_{\nabla f_t^2[\cdot,\cdot,\lambda]}\Big)=O(\tau(n_x+n_u)^3)$

Table 1: Space and time complexities of the oracles of Sec. 4 and 5. GD stands for gradient Descent, GN for Gauss-Newton, NE for Newton, DDP-LQ and DDP-Q stand for DDP with linear quadratic or quadratic approx.

The computational complexities of the oracles when the dynamics and the costs themselves are stored in memory are presented in Table 2. We consider for simplicity that the memory cost of storing the information necessary to evaluate a function $f : \mathbb{R}^d \to \mathbb{R}^n$ is nd as it is the case for a linear function f.

In summary, by considering an implementation that simply stores in memory the inputs and the programs that implement the functions, a Newton oracle or an oracle based on a DDP approach with quadratic approximation have the same time and space complexities as their linear quadratic counterparts up to constant factors. This remark was done by (Nganga and Wensing, 2021) for implementing a DDP algorithm with quadratic approximations.

	Time complexities of the forward pass
All cases	$\tau\Big(\underbrace{{n_x}^2+n_x n_u}_{f_t}+\underbrace{n_x+n_u}_{h_t}\Big)=O(\tau({n_x}^2+n_x n_u))$

	Space complexities of the forward pass
Function eval.	0
All other cases	$\tau\Big(\underbrace{{n_x}^2+n_x n_u}_{f_t}+\underbrace{n_x+n_u}_{h_t}\Big)=O(\tau({n_x}^2+n_x n_u))$

	Time complexities of the backward passes
GD	$\tau\Big(\underbrace{{n_x}^2+n_x n_u}_{\nabla f_t}+\underbrace{n_x+n_u}_{\nabla h_t}+\underbrace{{n_x}^2+n_x n_u}_{\text{Roll}}+\underbrace{{n_x}^2+n_x n_u}_{\text{LBP}}\Big)=O(\tau({n_x}^2+n_x n_u))$
GN/DDP-LQ	$\tau\Big(\underbrace{{n_x}^2+n_x n_u}_{\nabla f_t}+\underbrace{n_x+n_u}_{\nabla h_t}+\underbrace{{n_x}^2+{n_u}^2+n_x n_u}_{\nabla^2 h_t}\Big)$
	$+\,\tau\Big(\underbrace{{n_x}^2+n_x n_u}_{\text{Roll}}+\underbrace{{n_x}^3+{n_u}^3+{n_u}^2 n_x}_{\text{LQBP}}\Big)=O(\tau(n_x+n_u)^3)$
NE/DDP-Q	$\tau\Big(\underbrace{{n_x}^2+n_x n_u}_{\nabla f_t}+\underbrace{n_x+n_u}_{\nabla h_t}+\underbrace{{n_x}^2+{n_u}^2+n_x n_u}_{\nabla^2 h_t}+\underbrace{{n_x}^2+{n_u}^2+n_x n_u}_{\nabla^2(f_t^\top\lambda)}\Big)$
	$+\,\tau\Big(\underbrace{{n_x}^2+n_x n_u}_{\text{Roll}}+\underbrace{{n_x}^3+{n_u}^3+{n_u}^2 n_x}_{\text{LQBP}}\Big)=O(\tau(n_x+n_u)^3)$

Table 2: Space and time complexities of the oracles when storing functions as in, e.g., Algo. 10.

8 Necessary and Sufficient Optimality Conditions

We briefly recall the optimality conditions for nonlinear control problems in continuous and discrete time. The problem we consider in continuous time is

$$\min_{\substack{x \in \mathcal{C}^1([0,1]) \\ u \in \mathcal{C}([0,1])}} \int_0^1 h(x(t), u(t), t)dt + h(x(1), 1) \tag{43}$$
$$\text{s.t.} \quad \dot{x}(t) = \mathrm{f}(x(t), u(t), t), \quad x(0) = \bar{x}_0,$$

where $\mathcal{C}([0,1])$ and $\mathcal{C}^1([0,1])$ demote the set of continuous and continuously differentiable functions on $[0,1]$ respectively. By using an Euler discretization scheme with discretization stepsize $\Delta = 1/\tau$, we get the discrete time control problem

$$\min_{\substack{x_0,\ldots,x_\tau \in \mathbb{R}^{n_x} \\ u_0\ldots,u_{\tau-1} \in \mathbb{R}^{n_u}}} \sum_{t=0}^{\tau-1} h_t(x_t, u_t) + h_\tau(x_\tau) \tag{44}$$
$$\text{subject to} \quad x_{t+1} = x_t + \mathrm{f}_t(x_t, u_t), \quad \text{for } t \in \{0, \ldots, \tau-1\}, \quad x_0 = \bar{x}_0,$$

where $x_t = x(\Delta t)$, $u_t = u(\Delta t)$, $h_t = \Delta h(\cdot, \cdot, \Delta t)$, $h_\tau = h(\cdot, 1)$, $\mathrm{f}_t = \Delta \mathrm{f}(\cdot, \cdot, \Delta t)$. Compared to problem (1), we have $x_t + \mathrm{f}_t(x_t, u_t) = f_t(x_t, u_t)$.

8.1 Necessary Optimality Conditions

Necessary conditions for the continuous time control problem are known as Pontryagin's maximum principle, recalled below.

Theorem 8.1 (Pontryagin's maximum principle (Pontryagin et al., 1963)). *Define the Hamiltonian associated to problem* (43) *as*

$$\tilde{H}(x(t), u(t), \lambda(t), t) = h(x(t), u(t), t) + \lambda(t)^\top \mathrm{f}(x(t), u(t), t).$$

A trajectory $x \in \mathcal{C}^1([0,1])$ and a control function $u \in \mathcal{C}([0,1])$ are optimal if there exists $\lambda \in \mathcal{C}^1([0,1])$ such that

1. *$\dot{x}(t) = \nabla_{\lambda(t)} \tilde{H}(x(t), u(t), \lambda(t), t)$ for all $t \in [0,1]$ with $x(0) = \bar{x}_0$,*
2. *$\dot{\lambda}(t) = -\nabla_{x(t)} \tilde{H}(x(t), u(t), \lambda(t), t)$ for all $t \in [0,1]$ with $\lambda(1) = \nabla_{x(1)} h(x(1), 1)$,*
3. *$\tilde{H}(x(t), u(t), \lambda(t), t) = \min_{u \in \mathbb{R}^{n_u}} \tilde{H}(x(t), u, \lambda(t), t)$ for all $t \in [0,1]$.*

In comparison, necessary optimality conditions for the discretized problem (44) are given by considering the Karush–Kuhn–Tucker conditions of the problem, or equivalently by considering a sequence of controls such that the gradient of the objective is null.

Fact 8.2. *Define the Hamiltonian associated to problem* (44) *as*

$$\tilde{H}_t(x_t, u_t, \lambda_{t+1}) = h_t(x_t, u_t) + \lambda_{t+1}^\top \mathrm{f}_t(x_t, u_t).$$

A trajectory $x_0, \ldots, x_\tau \in \mathbb{R}^{n_x}$ and a sequence of controls $u_0, \ldots, u_{\tau-1} \in \mathbb{R}^{n_u}$ are optimal if there exists $\lambda_1, \ldots, \lambda_\tau \in \mathbb{R}^{n_x}$ such that

1. *$x_{t+1} - x_t = \nabla_{\lambda_{t+1}} \tilde{H}_t(x_t, u_t, \lambda_{t+1})$ for all $t \in \{0, \ldots, \tau-1\}$, with $x_0 = \bar{x}_0$,*
2. *$\lambda_{t+1} - \lambda_t = -\nabla_{x_t} \tilde{H}_t(x_t, u_t, \lambda_{t+1})$ for all $t \in \{1, \ldots, \tau-1\}$, with $\lambda_\tau = \nabla h_\tau(x_\tau)$,*
3. *$\nabla_{u_t} \tilde{H}_t(x_t, u_t, \lambda_{t+1}) = 0$ for all $t \in \{0, \ldots, \tau-1\}$.*

The first two conditions in Fact 8.2 are the discretizations of the first two conditions for the continuous time problem. The third condition differs since, in discrete time, the control variables need only to be stationary points of the Hamiltonian.

Note that the third condition of the continuous time case is a priori not necessary in the discrete case. For example, consider $\tau = 1$ such that the discrete-time control problem can be reduced to (ignoring the initial state) $\min_{u \in \mathbb{R}^{n_u}} h(u) + h_\tau(u + \mathrm{f}(u))$. If the third condition of the continuous case was necessary one would get that if $u^* = \arg\min_u h(u) + h_\tau(u + \mathrm{f}(u))$ then $u^* = \arg\min_u h(u) + {\lambda^*}^\top \mathrm{f}(u)$ for $\lambda^* = \nabla h_\tau(\mathrm{f}(u^*))$. As a numerical example, take $h(u) = u^2/2$, $f(u) = \cos(\exp(u))$ and $h_\tau(x) = x^2/2$. One verifies easily that $\arg\min_u h(u) + \nabla h_\tau(f(u^*))^\top f(u) \neq \arg\min_u h(u) + h_\tau(f(u))$.

8.2 Sufficient Optimality Conditions

Sufficient conditions for optimality of continuous control problems were presented by Mangasarian (1966); Arrow (1968); Kamien and Schwartz (1971). We present simply their translation in discrete time and refer the reader to, e.g., (Kamien and Schwartz, 1971) for the details in the continuous case. We rewrite problem (44) as

$$\begin{aligned} \min_{\substack{x_0,\ldots,x_\tau \in \mathbb{R}^{n_x} \\ \delta_0,\ldots,\delta_{\tau-1}\in\mathbb{R}^{n_x}}} \quad & \sum_{t=0}^{\tau-1} m_t(x_t,\delta_t) + h_\tau(x_\tau), \quad \text{where } m_t(x_t,\delta_t) = \inf_{\substack{u_t\in\mathbb{R}^{n_u} \\ \delta_t = \mathrm{f}_t(x_t,u_t)}} h_t(x_t,u_t) \\ \text{subject to} \quad & \delta_t = x_{t+1} - x_t, \; x_0 = \bar{x}_0. \end{aligned} \tag{45}$$

Sufficient conditions are related to the true Hamiltonian, presented by Clarke (1979), and defined as the convex conjugate of $m_t(x_t,\cdot)$, i.e.,

$$H_t(x,\lambda) = \sup_{\delta\in\mathbb{R}^{n_x}} \lambda^\top \delta - m_t(x,\delta),$$

for $x,\lambda\in\mathbb{R}^{n_x}$. Note that the minimum of the Hamiltonian can be expressed with the true Hamiltonian as

$$\tilde{H}_t^{\min}(x,\lambda) = \inf_{u\in\mathbb{R}^{n_u}} h_t(x,u) + \lambda^\top \mathrm{f}_t(x,u) = -H_t(x,-\lambda).$$

Theorem 8.3. *Assume that m_t defined in (45) is such that $m_t(x_t,\cdot)$ is convex for any x_t. If there exist $x_0^*,\ldots,x_\tau^*$ and $\lambda_1^*,\ldots,\lambda_\tau^*$ such that $H_t(\cdot,-\lambda_{t+1}^*)$ is concave, i.e., $\tilde{H}_t^{\min}(\cdot,\lambda_{t+1}^*)$ is convex and*

$$\lambda_t^* - \lambda_{t+1}^* \in \partial_{x_t}\tilde{H}_t^{\min}(x_t^*,\lambda_{t+1}^*) \quad \text{for } t\in\{1,\ldots,\tau-1\}, \qquad \lambda_\tau^* = \nabla h_\tau(x_\tau^*) \tag{46}$$

$$x_{t+1}^* - x_t^* \in \partial_{\lambda_{t+1}}\tilde{H}_t^{\min}(x_t^*,\lambda_{t+1}^*) \quad \text{for } t\in\{0,\ldots,\tau-1\}, \qquad x_0^* = \hat{x}_0. \tag{47}$$

Then $x_0^*,\ldots,x_\tau^*$ is an optimal trajectory for (45).

Proof. Since $m_t(x_t,\cdot)$ is convex for any x_t, problem (45) can be rewritten

$$\min_{\substack{x_0,\ldots,x_\tau\in\mathbb{R}^{n_x} \\ x_0=\hat{x}_0}} \sup_{\lambda_1,\ldots,\lambda_\tau\in\mathbb{R}^{n_x}} \sum_{t=0}^{\tau-1}\Big(-{\lambda_{t+1}}^\top(x_{t+1}-x_t) - H_t(x_t,-\lambda_{t+1})\Big) + h_\tau(x_\tau). \tag{48}$$

The above problem can be written as $\min_{\boldsymbol{x}\in\mathbb{R}^{\tau n_x}} \sup_{\boldsymbol{\lambda}\in\mathbb{R}^{\tau n_x}} f(\boldsymbol{x},\boldsymbol{\lambda})$ with $f(\boldsymbol{x},\cdot)$ concave for any $\boldsymbol{x}$. The assumptions amount to consider $\boldsymbol{x}^*,\boldsymbol{\lambda}^*$ such that (i) $0\in\partial_{\boldsymbol{\lambda}^*} f(\boldsymbol{x}^*,\boldsymbol{\lambda}^*)$, (ii) $f(\cdot,\boldsymbol{\lambda}^*)$ convex and $0\in\partial_{\boldsymbol{x}^*} f(\boldsymbol{x}^*,\boldsymbol{\lambda}^*)$. Then for any $\boldsymbol{x}\in\mathbb{R}^{\tau n_x}$,

$$\sup_{\boldsymbol{\lambda}\in\mathbb{R}^{\tau n_x}} f(\boldsymbol{x},\boldsymbol{\lambda}) \geq f(\boldsymbol{x},\boldsymbol{\lambda}^*) \overset{(ii)}{\geq} f(\boldsymbol{x}^*,\boldsymbol{\lambda}^*) \overset{(i)}{=} \sup_{\boldsymbol{\lambda}\in\mathbb{R}^{\tau n_x}} f(\boldsymbol{x}^*,\boldsymbol{\lambda}).$$

Hence $\boldsymbol{x}^*\in\arg\min_{\boldsymbol{x}\in\mathbb{R}^{\tau n_x}}\sup_{\boldsymbol{\lambda}\in\mathbb{R}^{\tau n_x}} f(\boldsymbol{x},\boldsymbol{\lambda})$, that is, $x_0^*,\ldots,x_\tau^*$ is an optimal trajectory. □

Conditions (46) and (47) of Proposition 8.3 may be made explicit as the existence of $u_t^*\in\arg\min_{u_t} h_t(x_t,u_t) + \lambda_{t+1}^\top f(x_t,u_t)$, $v_t^*\in\arg\min_{u_t} h_t(x_t,u_t)+\lambda_{t+1}^\top f(x_t,u_t)$ such that

$$\begin{aligned} \lambda_t^* - \lambda_{t+1}^* &= \nabla_{x_t} h_t(x_t^*,v_t^*) + \nabla_{x_t}\mathrm{f}_t(x_t^*,v_t^*)\lambda_{t+1}^* \\ x_{t+1}^* - x_t^* &= \mathrm{f}_t(x_t^*,u_t^*). \end{aligned}$$

While Theorem 8.3 presents original sufficient optimality conditions for the discretized control problem (44), it requires the problem to be reformulated in the form (48) by assuming the convexity of the functions m_t, which appears unclear to verify in simple instances. Moreover, Theorem 8.3 requires the concavity of the true Hamiltonian at the candidate solution which may be difficult to verify in practice.

9 Summary

In Fig. 2, we present a summary of the different algorithms presented in this paper and detailed in this section. Recall that our objective is

$$\mathcal{J}(\boldsymbol{u}) = \sum_{t=0}^{\tau-1} h_t(x_t, u_t) + h_\tau(x_\tau)$$
$$\text{s.t.} \quad x_{t+1} = f_t(x_t, u_t) \quad \text{for } t \in \{0, \ldots, \tau-1\}, \quad x_0 = \bar{x}_0.$$

that can be summarized as $\mathcal{J}(\boldsymbol{u}) = h(g(\boldsymbol{u}))$, where, for $\boldsymbol{u} = (u_0; \ldots; u_{\tau-1})$, $\boldsymbol{x} = (x_1; \ldots; x_\tau)$,

$$h(\boldsymbol{x}, \boldsymbol{u}) = \sum_{t=0}^{\tau-1} h_t(x_t, u_t) + h_\tau(x_\tau), \; g(\boldsymbol{u}) = (f^{[\tau]}(\bar{x}_0, \boldsymbol{u}), \boldsymbol{u}), \; f^{[\tau]}(x_0, \boldsymbol{u}) = (x_1; \ldots; x_\tau)$$
$$\text{s.t.} \quad x_{t+1} = f_t(x_t, u_t) \quad \text{for } t \in \{0, \ldots, \tau-1\}.$$

We present nonlinear control algorithms from a functional viewpoint by introducing finite difference, linear and quadratic expansions of the dynamics and the costs presented in the notations in Eq. (2).

For a function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^p$, with $p = 1$ (for the costs) or $p = n_x$ (for the dynamics), these expansions read for $x, u \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$,

$$\delta_f^{x,u} : y, v \to f(x+y, u+v) - f(x, u), \qquad \ell_f^{x,u} : y, v \to \nabla_x f(x,u)^\top y + \nabla_u f(x,u)^\top v \tag{49}$$
$$q_f^{x,u} : y, v \to \nabla_x f(x,u)^\top y + \nabla_u f(x,u)^\top v + \frac{1}{2}\nabla_{xx}^2 f(x,u)[y,y,\cdot] + \frac{1}{2}\nabla_{uu}^2 f(x,u)[v,v,\cdot] + \nabla_{xu}^2 f(x,u)[y,v,\cdot]$$

For $\lambda \in \mathbb{R}^p$, we denote shortly

$$\frac{1}{2}\nabla^2 f(x,u)[\cdot,\cdot,\lambda] : (y,v) \to \frac{1}{2}\nabla_{xx}^2 f(x,u)[y,y,\lambda] + \frac{1}{2}\nabla_{uu}^2 f(x,u)[v,v,\lambda] + \nabla_{xu}^2 f(x,u)[y,v,\lambda].$$

In the algorithms, we consider storing in memory linear or quadratic functions by storing the associated vectors, matrices or tensors defining the linear or quadratic functions. For example, to store the linear expansion ℓ_f^x or the quadratic expansion q_f^x of a function $f : \mathbb{R}^d \to \mathbb{R}^p$ around a point x, we consider storing $\nabla f(x) \in \mathbb{R}^{d\times p}$ and $\nabla^2 f(x) \in \mathbb{R}^{d\times d\times p}$. In the backward or roll-out passes, we consider that by having access to the the linear or quadratic functions, we have access to the associated matrices/tensors defining the operations as presented in, e.g., Algo. 2. The functional viewpoint help to isolate the main technical operations in the procedures LQBP in Algo. 2 or LBP in Algo. 3 and to identify the discrepancies between, e.g., the Newton oracle in Algo. 14 and a DDP oracle with quadratic approximations presented in Algo. 16. For a presentation of the algorithms in a purely algebraic viewpoint, we refer the reader to, e.g., (Wright, 1990; Liao and Shoemaker, 1992; Sideris and Bobrow, 2005).

In Algo. 7, 8, 9, we a priori need to check whether the sub-problems defined by the Bellman recursion are strongly convex or not. Namely in Algo. 7, 8, 9, we need to check that $q_t(x,\cdot) + c_{t+1}(\ell_t(x,\cdot))$ is strongly convex for any x. With the notations of Algo. 2, this amounts to check that $Q + B^\top J_{t+1} B \succ 0$. This can be done by checking the positivity of the minimum eigenvalue of $Q + B^\top J_{t+1} B$. In our implementation, we simply check that

$$j_t^0 - j_{t+1}^0 = -\frac{1}{2}(q + B^\top j_{t+1})^\top (Q + B^\top J_{t+1} B)^{-1} (q + B^\top j_{t+1}) < 0. \tag{50}$$

If condition (50) is not satisfied then necessarily $Q + B^\top J_{t+1} B \not\succ 0$. We chose to use condition (50) since this quantity is directly available and computing the eigenvalues of $Q + B^\top J_{t+1} B \succ 0$ can slow down the computations. Moreover, if criterion (50) is satisfied for all $t \in \{0, \ldots, \tau-1\}$, this means that, for the Guass-Newton and the Newton methods, the resulting direction is a descent direction for the objective. Algo. 4 details the aforementioned verification step.

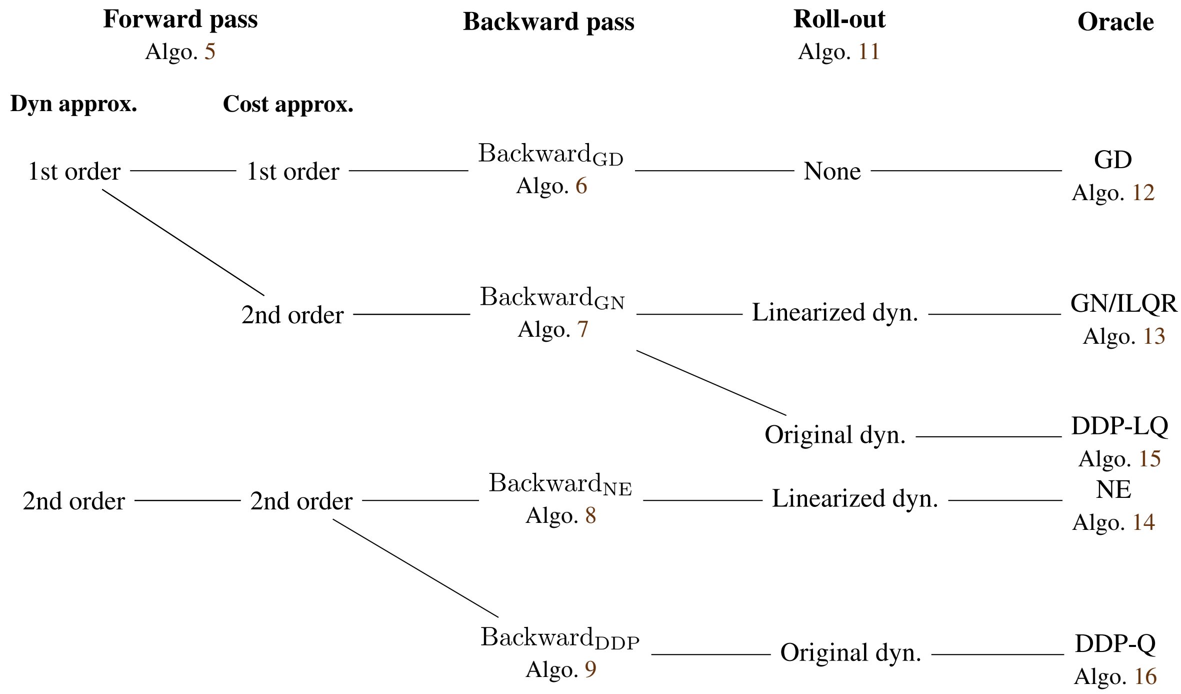

Figure 2: Taxonomy of non-linear control algorithms. GD stands for gradient Descent, GN for Gauss-Newton, NE for Newton. DDP-LQ and DDP-Q stand for DDP approaches based on linear quadratic or quadratic approximations respectively of Bellman's equation. Line-search procedure is presented in Algo. 17

Algorithm 2 Analytic solution of Bellman's equation (5) for linear dynamics, quadratic costs
[LQBP : $\ell_t, q_t, c_{t+1} \to c_t, \pi_t$]

Inputs:
1. Linear function ℓ_t parameterized as $\ell_t(x,u) = A_t x + B_t u$,
2. Quadratic function q_t parameterized as $q_t(x,u) = \frac{1}{2}x^\top P_t x + \frac{1}{2}u^\top Q_t u + x^\top R_t u + p_t^\top x + q_t^\top u$
3. Quadratic function c_{t+1} parameterized as $c_{t+1}(x) = \frac{1}{2}x^\top J_{t+1} x + j_{t+1}^\top x + j_{t+1}^0$.

Define $c_t : x \to \frac{1}{2}x^\top J_t x + j_t^\top x + j_t^0$ with

$$
\begin{aligned}
J_t &= P_t + A_t^\top J_{t+1} A_t - (R_t + A_t^\top J_{t+1} B_t)(Q_t + B_t^\top J_{t+1} B_t)^{-1}(R_t^\top + B_t^\top J_{t+1} A_t) \\
j_t &= p_t + A_t^\top j_{t+1} - (R_t + A_t^\top J_{t+1} B_t)(Q_t + B_t^\top J_{t+1} B_t)^{-1}(q_t + B_t^\top j_{t+1}), \\
j_t^0 &= j_{t+1}^0 - \frac{1}{2}(q_t + B_t^\top j_{t+1})^\top (Q_t + B_t^\top J_{t+1} B_t)^{-1}(q_t + B_t^\top j_{t+1}).
\end{aligned}
$$

Define $\pi_t : x \to K_t x + k_t$ with

$$
K_t = -(Q_t + B_t^\top J_{t+1} B_t)^{-1}(R_t^\top + B_t^\top J_{t+1} A_t), \qquad k_t = -(Q_t + B_t^\top J_{t+1} B_t)^{-1}(q_t + B_t^\top j_{t+1}).
$$

Output: Cost-to-go c_t and policy π_t at time t

Algorithm 3 Analytic solution of Bellman's equation (26) for linear dynamics, linear regularized costs
$\left[\mathrm{LBP} : \ell_t^f, \ell_t^h, c_{t+1}, \nu \to c_t, \pi_t\right]$

Inputs:
1. Linear function ℓ_f parameterized as $\ell_t^f(x,u) = A_t x + B_t u$
2. Linear function ℓ_h parameterized as $\ell_t^h(x,u) = p_t^\top x + q_t^\top u$
3. Linear function c_{t+1} parameterized as $c_{t+1}(x) = j_{t+1}^\top x + j_{t+1}^0$
4. Regularization $\nu \geq 0$

Define $c_t : x \to j_t^\top x + j_t^0$ with $j_t = p_t + A_t^\top j_{t+1},\ j_t^0 = j_{t+1}^0 - \|q_t + B_t^\top j_{t+1}\|_2^2/(2\nu)$.
Define $\pi_t : x \to k_t$ with $k_t = -(q_t + B_t^\top j_{t+1})/\nu$.
Output: Cost-to-go c_t and policy π_t at time t

Algorithm 4 Check if subproblems given by $q_t(y,\cdot) + c_{t+1}(\ell_t(y,\cdot))$ are valid for solving Bellman's equation (5)
$[\mathrm{CheckSubProblem} : \ell_t, q_t, c_{t+1} \to \mathrm{valid} \in \{\mathrm{True}, \mathrm{False}\}]$

Option: Check strong convexity of subproblems or check only if the result gives a descent direction
Inputs:
1. Linear function ℓ_t parameterized as $\ell_t(x,u) = A_t x + B_t u$,
2. Quadratic function q_t parameterized as $q_t(x,u) = \frac{1}{2}x^\top P_t x + \frac{1}{2}u^\top Q_t u + x^\top R_t u + p_t^\top x + q_t^\top u$
3. Quadratic function c_{t+1} parameterized as $c_{t+1}(x) = \frac{1}{2}x^\top J_{t+1} x + j_{t+1}^\top x + j_{t+1}^0$.

if check strong convexity **then**
  Compute the eigenvalues $\lambda_1 \leq \ldots \leq \lambda_{n_u}$ of $Q_t + B_t^\top J_{t+1} B_t$
  if $\lambda_1 > 0$ **then** valid $=$ True **else** valid $=$ False
else if check descent direction **then**
  Compute $j_t^0 - j_{t+1}^0 = -\frac{1}{2}(q_t + B_t^\top j_{t+1})^\top (Q_t + B_t^\top J_{t+1} B_t)^{-1} (q_t + B_t^\top j_{t+1})$
  if $j_t^0 - j_{t+1}^0 < 0$ **then** valid $=$ True **else** valid $=$ False
end if
Output: valid

Algorithm 5 Forward pass
$\left[\mathrm{Forward} : \boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f, o_h \to \mathcal{J}(\boldsymbol{u}), (m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}\right]$

Inputs: Command $\boldsymbol{u} = (u_0; \ldots; u_{\tau-1})$, dynamics $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau}$, initial state $\bar{x}_0$, order of the information to collect on the dynamics $o_f \in \{0,1,2\}$ and the costs $o_h \in \{0,1,2\}$
Initialize $x_0 = \bar{x}_0$, $\mathcal{J}(\boldsymbol{u}) = 0$
for $t = 0, \ldots \tau - 1$ **do**
  Compute $h_t(x_t, u_t)$, update $\mathcal{J}(\boldsymbol{u}) \leftarrow \mathcal{J}(\boldsymbol{u}) + h_t(x_t, u_t)$
  if $o_h \geq 1$ **then** Compute and store $\nabla h_t(x_t, u_t)$ defining $\ell_{h_t}^{x_t,u_t}$ as in (49)
  if $o_h = 2$ **then** Compute and store $\nabla^2 h_t(x_t, u_t)$ defining, with $\nabla h_t(x_t, u_t)$, $q_{h_t}^{x_t,u_t}$ as in (49)
  Compute $x_{t+1} = f_t(x_t, u_t)$
  if $o_f \geq 1$ **then** Compute and store $\nabla f_t(x_t, u_t)$ defining $\ell_{f_t}^{x_t,u_t}$ as in (49)
  if $o_f = 2$ **then** Compute and store $\nabla^2 f_t(x_t, u_t)$ defining, with $\nabla f_t(x_t, u_t)$, $q_{f_t}^{x_t,u_t}$ as in (49)
end for
Compute $h_\tau(x_\tau)$, update $\mathcal{J}(\boldsymbol{u}) \leftarrow \mathcal{J}(\boldsymbol{u}) + h_\tau(x_\tau)$
if $o_h \geq 1$ **then** Compute and store $\nabla h_\tau(x_\tau)$ defining $\ell_{h_\tau}^{x_\tau}$ as in (49)
if $o_h = 2$ **then** Compute and store $\nabla^2 h_\tau(x_\tau)$ defining, with $\nabla h_\tau(x_\tau)$, $q_{h_\tau}^{x_\tau}$ as in (49)
Output: Total cost $\mathcal{J}(\boldsymbol{u})$
Stored: (if $\min\{o_f, o_h\} \neq 0$) Approximations $(m_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (m_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, m_{h_\tau}^{x_\tau}$ defined by

$$m_{f_t}^{x_t,u_t} = \begin{cases} \ell_{f_t}^{x_t,u_t} & \text{if } o_f = 1 \\ q_{f_t}^{x_t,u_t} & \text{if } o_f = 2 \end{cases}, \quad m_{h_t}^{x_t,u_t} = \begin{cases} \ell_{h_t}^{x_t,u_t} & \text{if } o_h = 1 \\ q_{h_t}^{x_t,u_t} & \text{if } o_h = 2 \end{cases}, \quad m_{h_\tau}^{x_\tau} = \begin{cases} \ell_{h_\tau}^{x_\tau} & \text{if } o_h = 1 \\ q_{h_\tau}^{x_\tau} & \text{if } o_h = 2 \end{cases}$$

Algorithm 6 Backward pass for gradient oracle

$\left[\mathrm{Backward}_{\mathrm{GD}} : (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (\ell_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, \ell_{h_\tau}^{x_\tau}, \nu) \rightarrow (\pi_t)_{t=0}^{\tau-1}, c_0\right]$

Inputs: Linear expansions of the dynamics $(\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}$, linear expansions of the costs $(\ell_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, \ell_{h_\tau}^{x_\tau}$, regularization $\nu > 0$
Initialize $c_\tau = \ell_{h_\tau}^{x_\tau}$
for $t = \tau - 1, \dots 0$ **do**
  Define $\ell_t = \ell_{f_t}^{x_t,u_t}, \quad q_t : y_t, v_t \rightarrow \ell_{h_t}^{x_t,u_t}(y_t, v_t) + \frac{\nu}{2}\|v_t\|_2^2$
  Compute $c_t, \pi_t = \mathrm{LQBP}(\ell_t, q_t, c_{t+1}) = \mathrm{LBP}(\ell_{f_t}^{x_t,u_t}, \ell_{h_t}^{x_t,u_t}, c_{t+1}, \nu)$ where LBP is given in Algo. 3
end for
Outputs: Policies $(\pi_t)_{t=0}^{\tau-1}$, cost-to-go function at initial time c_0

Algorithm 7 Backward pass for Gauss-Newton oracle

$\left[\mathrm{Backward}_{\mathrm{GN}} : (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu) \rightarrow (\pi_t)_{t=0}^{\tau-1}, c_0\right]$

Inputs: Linear expansions of the dynamics $(\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}$, quadratic expansions of the costs $(q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}$, regularization $\nu \geq 0$
Initialize $c_\tau = q_{h_\tau}^{x_\tau}$
for $t = \tau - 1, \dots 0$ **do**
  Define $\ell_t = \ell_{f_t}^{x_t,u_t}, \quad q_t : y_t, v_t \rightarrow q_{h_t}^{x_t,u_t}(y_t, v_t) + \frac{\nu}{2}\|v_t\|_2^2$,
  if $\mathrm{CheckSubProblem}(\ell_t, q_t, c_{t+1})$ is True **then**
    Compute $c_t, \pi_t = \mathrm{LQBP}(\ell_t, q_t, c_{t+1})$ with LQBP given in Algo. 2
  else
    $\pi_s : x \rightarrow 0$ for $s \leq t$, $c_0 : x \rightarrow -\infty$, **break**
  end if
end for
Outputs: Policies $(\pi_t)_{t=0}^{\tau-1}$, cost-to-go function at initial time c_0

Algorithm 8 Backward pass for Newton oracle

$\left[\mathrm{Backward}_{\mathrm{NE}} : (q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu) \rightarrow (\pi_t)_{t=0}^{\tau-1}, c_0\right]$

Inputs: Quadratic expansions of the dynamics $(q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}$, quadratic expansions of the costs $(q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}$, regularization $\nu \geq 0$
Initialize $c_\tau = q_{h_\tau}^{x_\tau}$, $\lambda_\tau = \nabla h_\tau(x_\tau)$
for $t = \tau - 1, \dots 0$ **do**
  Define $\ell_t = \ell_{f_t}^{x_t,u_t}, \quad q_t : (y_t, v_t) \rightarrow q_{h_t}^{x_t,u_t}(y_t, v_t) + \frac{\nu}{2}\|v_t\|_2^2 + \frac{1}{2}\nabla^2 f_t(x_t, u_t)[\cdot, \cdot, \lambda_{t+1}](y_t, v_t)$
  Compute $\lambda_t = \nabla_{x_t} h_t(x_t, u_t) + \nabla_{x_t} f_t(x_t, u_t)\lambda_{t+1}$
  if $\mathrm{CheckSubProblem}(\ell_t, q_t, c_{t+1})$ is True **then**
    Compute $c_t, \pi_t = \mathrm{LQBP}(\ell_t, q_t, c_{t+1})$ with LQBP given in Algo. 2
  else
    $\pi_s : x \rightarrow 0$ for $s \leq t$, $c_0 : x \rightarrow -\infty$, **break**
  end if
end for
Outputs: Policies $(\pi_t)_{t=0}^{\tau-1}$, cost-to-go function at initial time c_0

Algorithm 9 Backward pass for a DDP approach with quadratic approximations
$\left[\mathrm{Backward}_{\mathrm{DDP}} : (q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu) \rightarrow (\pi_t)_{t=0}^{\tau-1}, c_0\right]$

Inputs: Quadratic expansions on the dynamics $(q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}$, quadratic expansions on the costs $(q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}$, regularization $\nu \geq 0$
Initialize $c_\tau = q_{h_\tau}^{x_\tau}$
for $t = \tau - 1, \ldots 0$ **do**
    Define $\ell_t = \ell_{f_t}^{x_t,u_t}, \quad q_t : y_t, v_t \rightarrow q_{h_t}^{x_t,u_t}(y_t, v_t) + \frac{\nu}{2}\|y_t\|_2^2 + \frac{1}{2}\nabla^2 f_t(x_t, u_t)[\cdot, \cdot, \nabla c_{t+1}(0)](y_t, v_t)$
    if $\mathrm{CheckSubProblem}(\ell_t, q_t, c_{t+1})$ is True **then**
        Compute $c_t, \pi_t = \mathrm{LQBP}(\ell_t, q_t, c_{t+1})$ with LQBP given in Algo. 2
    else
        $\pi_s : x \rightarrow 0$ for $s \leq t$, $c_0 : x \rightarrow -\infty$, **break**
    end if
end for
Outputs: Policies $(\pi_t)_{t=0}^{\tau-1}$, cost-to-go function at initial time c_0

Algorithm 10 Backward pass for Newton oracle with function storage

Inputs: Stored functions $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau}$, inputs $(u_t)_{t=0}^{\tau-1}$ with associated trajectory $(x_t)_{t=0}^{\tau}$
Compute the quadratic expansion $q_{h_\tau}^{x_\tau}$ of the final cost and the derivative $\nabla h_\tau(x_\tau)$ of the final cost on x_τ
Set $c_\tau = q_{h_\tau}^{x_\tau}$, $\lambda_\tau = \nabla h_\tau(x_\tau)$
for $t = \tau - 1, \ldots 0$ **do**
    Compute the linear approximation $\ell_{f_t}^{x_t,u_t}$ of the dynamic around x_t, u_t
    Compute the quadratic approximation $q_{h_t}^{x_t,u_t}$ of the cost around x_t, u_t
    Compute the Hessian of $x_t, u_t \rightarrow f_t(x_t, u_t)^\top \lambda_{t+1}$ on x_t, u_t which gives $\frac{1}{2}\nabla^2 f_t(x_t, u_t)[\cdot, \cdot, \lambda_{t+1}]$.
    Define $\ell_t = \ell_{f_t}^{x_t,u_t}, \quad q_t : (y_t, v_t) \rightarrow q_{h_t}^{x_t,u_t}(y_t, v_t) + \frac{\nu}{2}\|v_t\|_2^2 + \frac{1}{2}\nabla^2 f_t(x_t, u_t)[\cdot, \cdot, \lambda_{t+1}](y_t, v_t)$
    Compute $\lambda_t = \nabla_{x_t} h_t(x_t, u_t) + \nabla_{x_t} f_t(x_t, u_t)\lambda_{t+1}$
    if $\mathrm{CheckSubProblem}(\ell_t, q_t, c_{t+1})$ is True **then**
        Compute $c_t, \pi_t = \mathrm{LQBP}(\ell_t, q_t, c_{t+1})$
    else
        $\pi_s : x \rightarrow 0$ for $s \leq t$, $c_0 : x \rightarrow -\infty$, **break**
    end if
end for
Outputs: Policies $(\pi_t)_{t=0}^{\tau-1}$, cost-to-go function at initial time c_0

Algorithm 11 Roll-out on dynamics
$\left[\mathrm{Roll} : y_0, (\pi_t)_{t=1}^{\tau-1}, (\phi_t)_{t=0}^{\tau-1} \rightarrow \boldsymbol{v}\right]$

Inputs: Initial state y_0, sequence of policies $(\pi_t)_{t=0}^{\tau-1}$, dynamics to roll-on $(\phi_t)_{t=0}^{\tau-1}$
for $t = 0, \ldots, \tau - 1$ **do**
    Compute and store $v_t = \pi_t(y_t), \; y_{t+1} = \phi_t(y_t, v_t)$.
end for
Output: Sequence of controllers $\boldsymbol{v} = (v_0; \ldots; v_{\tau-1})$

Algorithm 12 Gradient oracle
$\left[\mathrm{GD}: \boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \nu \rightarrow \boldsymbol{v}\right]$

1: **Inputs:** Command $\boldsymbol{u}{=}(u_0; \ldots; u_{\tau-1})$, dynamics $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau}$, initial state $\bar{x}_0$, regularization $\nu{>}0$
2: Compute with Algo. 5
$$\mathcal{J}(\boldsymbol{u}), (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (\ell_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, \ell_{h_\tau}^{x_\tau} = \mathrm{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 1, o_h = 1)$$
3: Compute with Algo. 6
$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \mathrm{Backward}_{\mathrm{GD}}((\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (\ell_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu)$$
4: Compute with Algo. 11
$$\boldsymbol{v} = \mathrm{Roll}(0, (\pi_t)_{t=0}^{\tau-1}, (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1})$$
5: **Output:** Gradient direction $\boldsymbol{v} = \arg\min_{\tilde{\boldsymbol{v}}\in\mathbb{R}^{\tau n_u}} \left\{\ell_{h\circ g}^{\boldsymbol{u}}(\tilde{\boldsymbol{v}}) + \frac{\nu}{2}\|\tilde{\boldsymbol{v}}\|_2^2\right\} = -\nu^{-1}\nabla(h\circ g)(\boldsymbol{u})$

Algorithm 13 Gauss-Newton oracle
$\left[\mathrm{GN}: \boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \nu \rightarrow \boldsymbol{v}\right]$

1: **Inputs:** Command $\boldsymbol{u}{=}(u_0; \ldots; u_{\tau-1})$, dynamics $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau}$, initial state $\bar{x}_0$, regularization $\nu{\geq}0$
2: Compute with Algo. 5
$$\mathcal{J}(\boldsymbol{u}), (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau} = \mathrm{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 1, o_h = 2)$$
3: Compute with Algo. 7
$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \mathrm{Backward}_{\mathrm{GN}}((\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu)$$
4: Compute with Algo. 11
$$\boldsymbol{v} = \mathrm{Roll}(0, (\pi_t)_{t=0}^{\tau-1}, (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1})$$
5: **Output:** If $c_0(0) = +\infty$, returns `infeasible`, otherwise returns Gauss-Newton direction
$$\boldsymbol{v} = \arg\min_{\tilde{\boldsymbol{v}}\in\mathbb{R}^{\tau n_u}} \left\{q_h^{g(\boldsymbol{u})}(\ell_g^{\boldsymbol{u}}(\tilde{\boldsymbol{v}})) + \frac{\nu}{2}\|\tilde{\boldsymbol{v}}\|_2^2\right\} = -(\nabla g(\boldsymbol{u})\nabla^2 h(\boldsymbol{x},\boldsymbol{u})\nabla g(\boldsymbol{u}) + \nu\,\mathrm{I})^{-1}\nabla(h\circ g)(\boldsymbol{u})$$

Algorithm 14 Newton oracle
$\left[\mathrm{NE}: \boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \nu \rightarrow \boldsymbol{v}\right]$

1: **Inputs:** Command $\boldsymbol{u}{=}(u_0; \ldots; u_{\tau-1})$, dynamics $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau}$, initial state $\bar{x}_0$, regularization $\nu{\geq}0$
2: Compute with Algo. 5
$$\mathcal{J}(\boldsymbol{u}), (q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau} = \mathrm{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 2, o_h = 2)$$
3: Compute with Algo. 8
$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \mathrm{Backward}_{\mathrm{NE}}((q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu)$$
4: Compute with Algo. 11
$$\boldsymbol{v} = \mathrm{Roll}(0, (\pi_t)_{t=0}^{\tau-1}, (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1})$$
5: **Output:** If $c_0(0) = +\infty$, returns `infeasible`, otherwise returns Newton direction
$$\boldsymbol{v} = \arg\min_{\tilde{\boldsymbol{v}}\in\mathbb{R}^{\tau n_u}} \left\{q_{h\circ g}^{\boldsymbol{u}}(\tilde{\boldsymbol{v}}) + \frac{\nu}{2}\|\tilde{\boldsymbol{v}}\|_2^2\right\} = -(\nabla^2(h\circ g)(\boldsymbol{u}) + \nu\,\mathrm{I})^{-1}\nabla(h\circ g)(\boldsymbol{u})$$

Algorithm 15 Differential dynamic programming oracle with linear quadratic approximations
$\left[\text{DDP-LQ} : \boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \nu \to \boldsymbol{v}\right]$

1: **Inputs:** Command $\boldsymbol{u}=(u_0;\ldots;u_{\tau-1})$, dynamics $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau}$, initial state $\bar{x}_0$, regularization $\nu\geq 0$
2: Compute with Algo. 5

$$\mathcal{J}(\boldsymbol{u}), (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau} = \text{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 1, o_h = 2)$$

3: Compute with Algo. 7

$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \text{Backward}_{\text{GN}}((\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu)$$

4: Compute with Algo. 11, for $\delta_{f_t}^{x_t,u_t}(y_t, v_t) = f(x_t + y_t, u_t + v_t) - f(x_t, u_t)$,

$$\boldsymbol{v} = \text{Roll}(0, (\pi_t)_{t=0}^{\tau-1}, (\delta_{f_t}^{x_t,u_t})_{t=0}^{\tau-1})$$

5: **Output:** If $c_0(0) = +\infty$, returns `infeasible`, otherwise returns DDP oracle with linear-quadratic approximations $\boldsymbol{v}$

Algorithm 16 Differential dynamic programming oracle with quadratic approximations
$\left[\text{DDP-Q} : \boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, \nu \to \boldsymbol{v}\right]$

1: **Inputs:** Command $\boldsymbol{u}=(u_0;\ldots;u_{\tau-1})$, dynamics $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau}$, initial state $\bar{x}_0$, regularization $\nu\geq 0$
2: Compute with Algo. 5

$$\mathcal{J}(\boldsymbol{u}), (q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau} = \text{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 2, o_h = 2)$$

3: Compute with Algo. 9

$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \text{Backward}_{\text{DDP}}((q_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu)$$

4: Compute with Algo. 11, for $\delta_{f_t}^{x_t,u_t}(y_t, v_t) = f(x_t + y_t, u_t + v_t) - f(x_t, u_t)$,

$$\boldsymbol{v} = \text{Roll}(0, (\pi_t)_{t=0}^{\tau-1}, (\delta_{f_t}^{x_t,u_t})_{t=0}^{\tau-1})$$

5: **Output:** If $c_0(0) = +\infty$, returns `infeasible`, otherwise returns DDP oracle with quadratic approximations $\boldsymbol{v}$

Algorithm 17 Line-search
$\left[\text{LineSearch} : \boldsymbol{u}, (h_t)_{t=0}^{\tau}, (f_t)_{t=0}^{\tau-1}, (\phi_t)_{t=0}^{\tau-1}, (\text{Pol} : \gamma \to (\pi_t^\gamma)_{t=0}^{\tau-1}, c_0^\gamma) \to \boldsymbol{u}^{\text{next}}\right]$

Inputs: Current controls $\boldsymbol{u}$, costs $(h_t)_{t=0}^{\tau}$, initial state $\bar{x}_0$, original dynamics $(f_t)_{t=0}^{\tau-1}$, dynamics to roll-out on $(\phi_t)_{t=0}^{\tau-1}$, family of policies and corresponding costs given by $\gamma \to (\pi_t^\gamma)_{t=0}^{\tau-1}, c_0^\gamma$, decreasing factor $\rho_{\text{dec}} \in (0, 1)$, increasing factor $\rho_{\text{inc}} > 1$, previous stepsize γ_{prev}
Compute $\mathcal{J}(\boldsymbol{u}) = \text{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 0, o_h = 0)$
if condition (40) is used **then**
    Initialize $\gamma = 1$
else if condition (42) is used **then**
    Compute $\nabla h(\boldsymbol{x}, \boldsymbol{u})$ for $\boldsymbol{x} = f^{[\tau]}(\bar{x}_0, \boldsymbol{u})$
    Initialize $\gamma = \rho_{\text{inc}}\gamma_{\text{prev}}/\|\nabla h(\boldsymbol{x}, \boldsymbol{u})\|_2$
end if
Initialize $y_0 = 0$, accept = False, minimal stepsize $\gamma_{\min} = 10^{-12}$
while not accept **do**
    Get $\pi_t^\gamma, c_0^\gamma = \text{Pol}(\gamma)$
    Compute $\boldsymbol{v}^\gamma = \text{Roll}(y_0, (\pi_t^\gamma)_{t=1}^{\tau-1}, (\phi_t)_{t=0}^{\tau-1})$
    Set $\boldsymbol{u}^{\text{next}} = \boldsymbol{u} + \boldsymbol{v}^\gamma$
    Compute $\mathcal{J}(\boldsymbol{u}^{\text{next}}) = \text{Forward}(\boldsymbol{u}^{\text{next}}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 0, o_h = 0)$
    if $\mathcal{J}(\boldsymbol{u}^{\text{next}}) - \mathcal{J}(\boldsymbol{u}) \leq c_0^\gamma(0)$ **then** set accept = True **else** set $\gamma \to \rho_{\text{dec}}\gamma$
    if $\gamma \leq \gamma_{\min}$ **then break**
end while
if condition (42) is used **then** $\gamma := \gamma\|\nabla h(\boldsymbol{x}, \boldsymbol{u})\|_2$
Output: Next sequence of controllers $\boldsymbol{u}^{\text{next}}$, store value of the stepsize selected γ

Algorithm 18 Iterative Linear Quadratic Regulator/Gauss-Newton step with line-search on descent directions

Inputs: Command $\boldsymbol{u}$, dynamics $(f_t)_{t=0}^{\tau-1}$, costs $(h_t)_{t=0}^{\tau-1}$, inital state $\bar{x}_0$
Compute with Algo. 5

$$\mathcal{J}(\boldsymbol{u}), (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau} = \text{Forward}(\boldsymbol{u}, (f_t)_{t=0}^{\tau-1}, (h_t)_{t=0}^{\tau}, \bar{x}_0, o_f = 1, o_h = 2)$$

Compute with Algo. 7

$$(\pi_t)_{t=0}^{\tau-1}, c_0 = \text{Backward}_{\text{GN}}((\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, 0)$$

Set $\nu = \nu_{\text{init}}$ with, e.g., $\nu_{\text{init}} = 10^{-6}$
while $c_0(0) = +\infty$ **do**
    Compute $(\pi_t)_{t=0}^{\tau-1}, c_0 = \text{Backward}_{\text{GN}}((\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, (q_{h_t}^{x_t,u_t})_{t=0}^{\tau-1}, q_{h_\tau}^{x_\tau}, \nu)$
    Set $\nu \to \rho_{\text{inc}}\nu$ with, e.g., $\rho_{\text{inc}} = 10$
end while
Define $\text{Pol} : \gamma \to \begin{pmatrix} (\pi_t^\gamma : & y \to \gamma\pi_t(0) + \nabla\pi_t(0)^\top y)_{t=0}^{\tau-1}, \\ c_0^\gamma : & y \to \gamma c_0(y) \end{pmatrix}$
Compute with Algo. 17

$$\boldsymbol{u}^{\text{next}} = \text{LineSearch}(\boldsymbol{u}, (h_t)_{t=0}^{\tau}, (f_t)_{t=0}^{\tau-1}, (\ell_{f_t}^{x_t,u_t})_{t=0}^{\tau-1}, \text{Pol})$$

Output: Next sequence of controllers $\boldsymbol{u}^{\text{next}}$

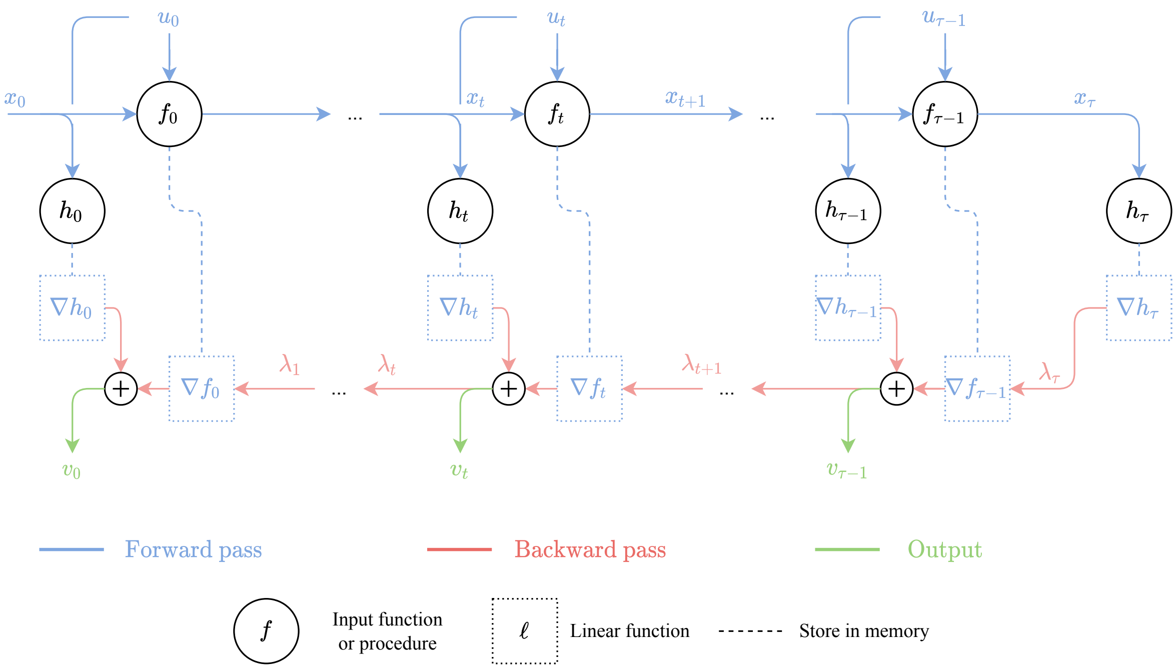

Figure 3: Computational scheme of a gradient oracle.

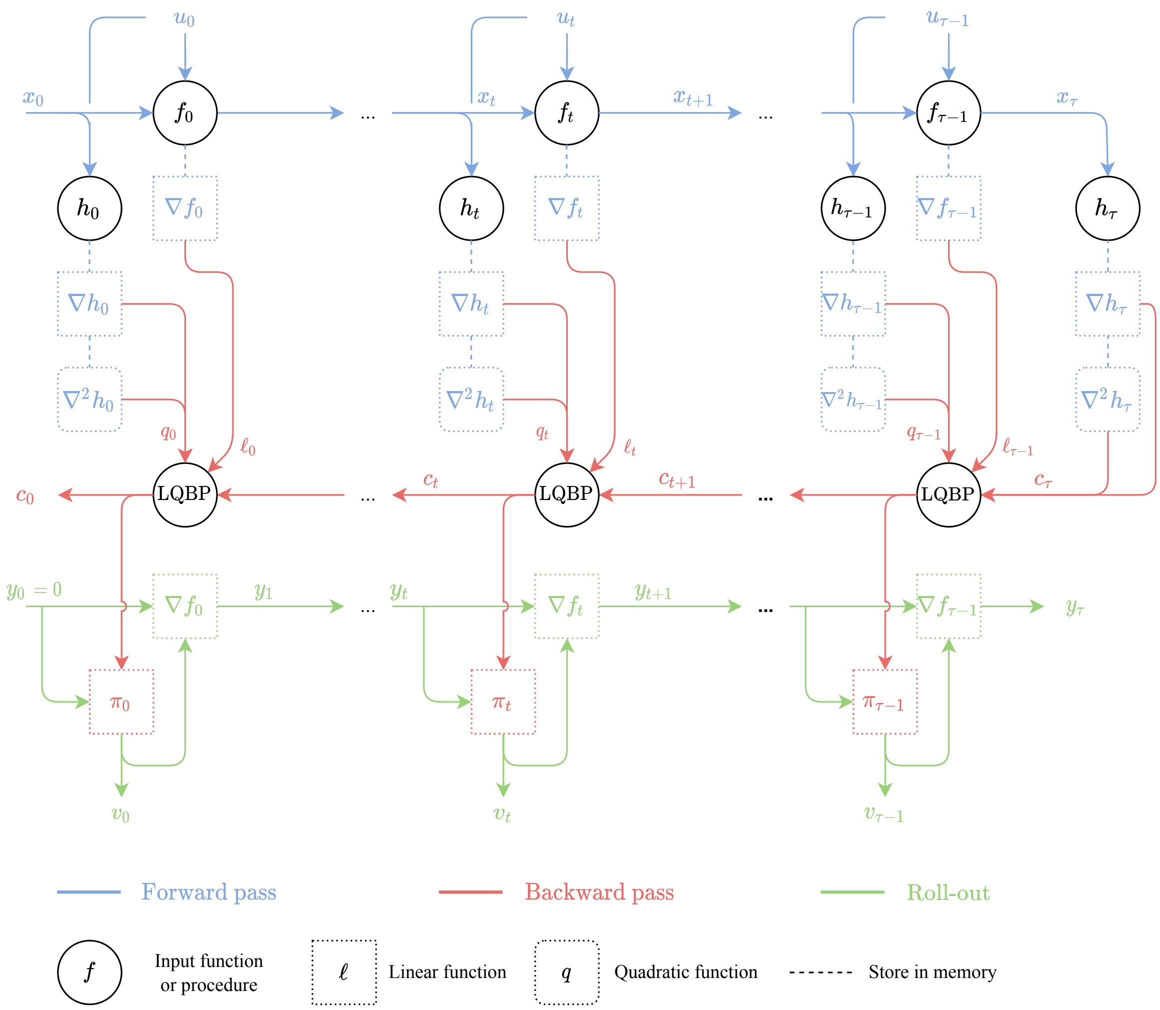

Figure 4: Computational scheme of a ILQR/Gauss-Newton oracle.

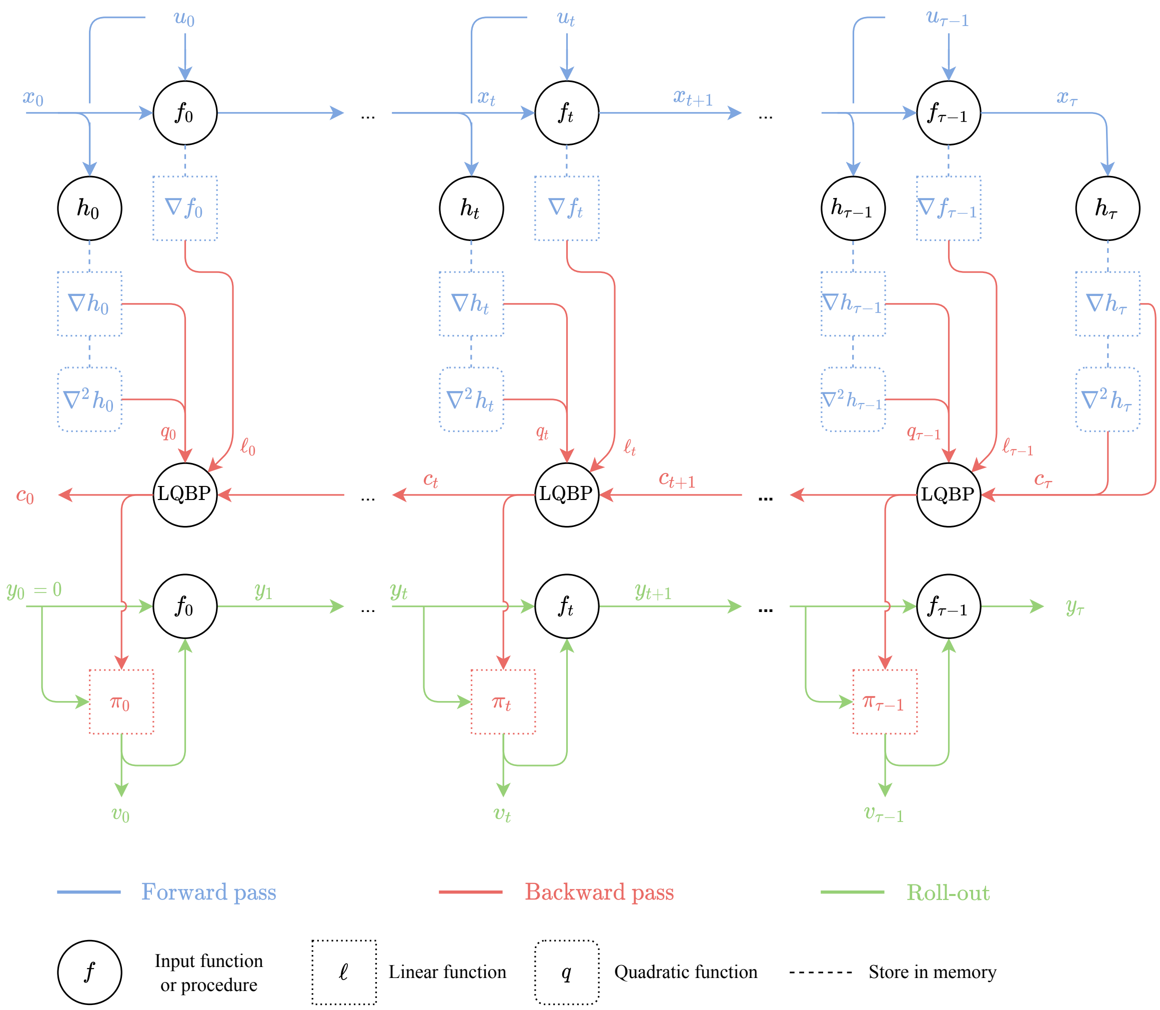

Figure 5: Computational scheme of a DDP oracle with linear quadratic approximations.

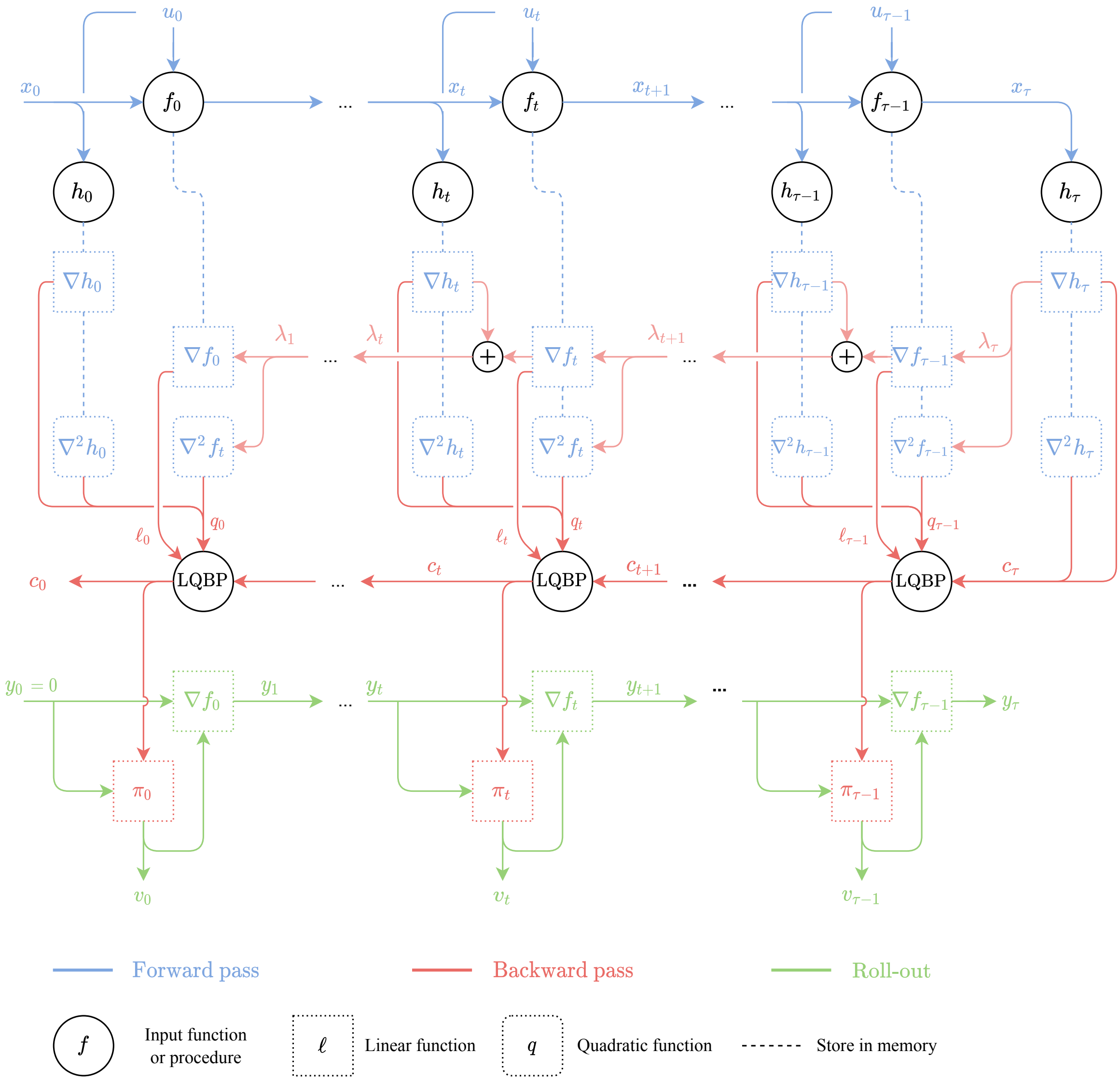

Figure 6: Computational scheme of a Newton oracle.

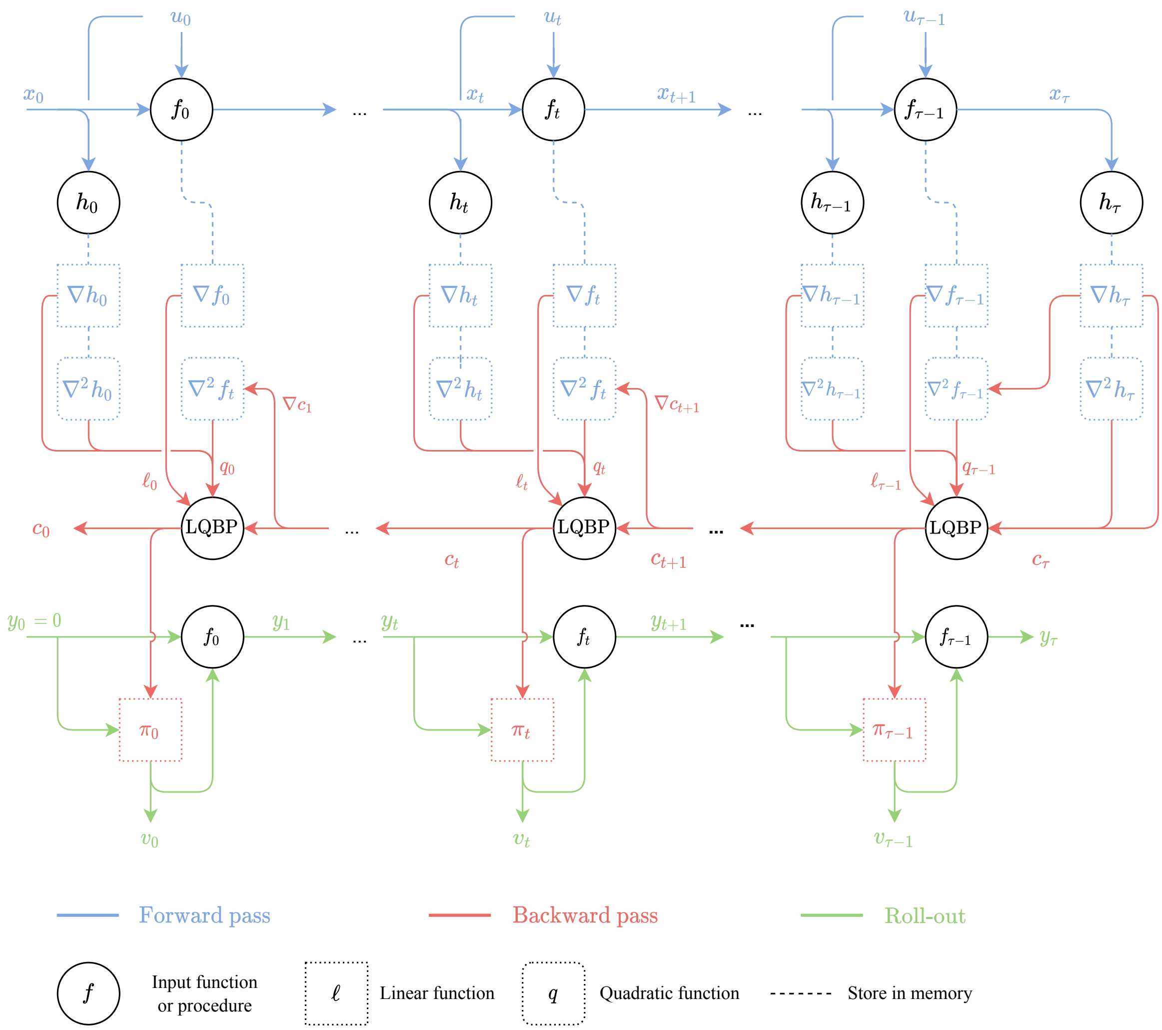

Figure 7: Computational scheme of a DDP oracle with quadratic approximations.

10 Experiments

We first describe in detail the continuous time systems studied in the experiments then present the numerical performances of the algorithms reviewed in this work[3]. Numerical constants are provided in the Appendix.

10.1 Discretization

In the following we denote by $z(t)$ the state of a system at time t. Given a control $u(t)$ at time t, we consider time-invariant dynamical systems governed by a differential equation of the form

$$\dot{z}(t) = \mathrm{f}(z(t), u(t)),$$

where f models the physics of the movement and are described below for each model.

Given a continuous time dynamic, the discrete time dynamics are given by a discretization method such that the states follow dynamics of the form

$$z_{t+1} = f(z_t, u_t) \quad \text{for } t \in \{0, \ldots\},$$

for a sequence of controls $u_0, u_1, \ldots$. One possible discretization method is the Euler method, which for a time-step Δ, reads

$$f(z_t, u_t) = z_t + \Delta \mathrm{f}(z_t, u_t).$$

Alternatively, we can consider a Runge-Kutta method of order 4 that defines the discrete-time dynamics as

$$\begin{aligned} f(z_t, u_t) &= z_t + \frac{\Delta}{6}(k_1 + k_2 + k_3 + k_4) \\ \text{where} \quad k_1 &= \mathrm{f}(z_t, u_t) \qquad & k_2 &= \mathrm{f}(z_t + \Delta k_1/2, u_t) \\ k_3 &= \mathrm{f}(z_t + \Delta k_2/2, u_t) \qquad & k_4 &= \mathrm{f}(z_t + \Delta k_3, u_t), \end{aligned}$$

where we consider the controls to be piecewise constant, i.e., constant on time intervals of size Δ. We can also consider a Runge-Kutta method with varying control inputs such that, for $u_t = (v_t, v_{t+1/3}, v_{t+2/3})$,

$$\begin{aligned} f(z_t, u_t) &= z_t + \frac{\Delta}{6}(k_1 + k_2 + k_3 + k_4) \\ \text{where} \quad k_1 &= \mathrm{f}(z_t, v_t) \qquad & k_2 &= \mathrm{f}(z_t + \Delta k_1/2, v_{t+1/3}) \\ k_3 &= \mathrm{f}(z_t + \Delta k_2/2, v_{t+1/3}) \qquad & k_4 &= \mathrm{f}(z_t + \Delta k_3, v_{t+2/3}). \end{aligned}$$

10.2 Swinging-up a Pendulum

We consider the problem of controlling a pendulum such that it swings up. Namely, the dynamics of a pendulum are given as

$$ml^2 \ddot{\theta}(t) = -mlg \sin\theta(t) - \mu\dot{\theta}(t) + u(t),$$

with θ the angle of the rod, m the mass of the blob, l the length of the blob, μ a friction coefficient, g the gravitational constant and u a torque applied to the pendulum (which defines the control we have on the system). Denoting the angle speed $\omega = \dot{\theta}$ and the state of the system $x = (\theta, \omega)$, the continuous time dynamics are given as

$$\mathrm{f} : (x = (\theta, \omega), u) \to \begin{pmatrix} \omega \\ -\frac{g}{l}\sin\theta - \frac{\mu}{ml^2}\omega + \frac{1}{ml^2}u \end{pmatrix},$$

such that the continuous time system is defined by $\dot{x}(t) = \mathrm{f}(x(t), u(t))$. After discretization by an Euler method, we get discrete time dynamics $f_t(x_t, u_t) = f(x_t, u_t)$ of the form

$$f(x_t, u_t) = x_t + \Delta \mathrm{f}(x_t, u_t) = \begin{pmatrix} \theta_t + \Delta\omega_t \\ \omega_t + \Delta\left(-\frac{g}{l}\sin\theta_t - \frac{\mu}{ml^2}\omega_t + \frac{1}{ml^2}u_t\right) \end{pmatrix},$$

where Δ is the discretization step and $x_t = (\theta_t; \omega_t)$.

A classical task is to enforce the pendulum to swing up and stop without using too much torque at each time step, i.e., for $\bar{x}_0 = (0, 0)$, the costs we consider are, for some non-negative parameters $\lambda \geq 0, \rho \geq 0$,

$$h_t(x_t, u_t) = \lambda\|u_t\|_2^2 \quad \text{for } t \in \{0, \ldots, \tau - 1\}, \quad h_\tau(x_\tau) = (\pi - \theta_\tau)^2 + \rho\|\omega_\tau\|_2^2.$$

[3]The code is available at https://github.com/vroulet/ilqc.

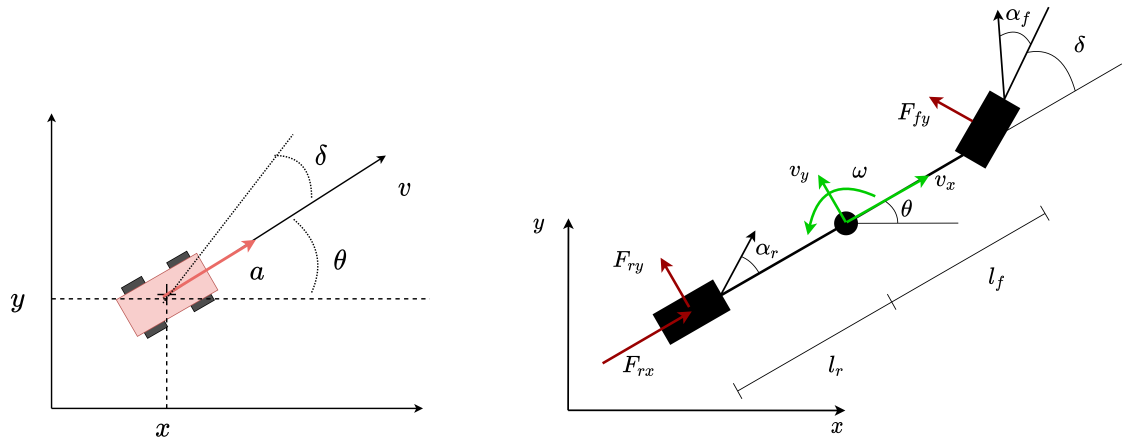

Figure 8: Left: Simple model of a car, Right: Bicycle model of a car

10.3 Autonomous Car Racing

We consider the control of a car on a track through two different dynamical models: a simple one where the orientation of the car is directly controlled by the steering angle, a more realistic one that takes into account the tire forces to control the orientation of the car. In the following, we present the dynamics, a simple tracking cost, a contouring cost enforcing the car to race the track at a reference speed or as fast as possible.

10.3.1 Dynamics

Simple model. A simple model of the car is described in Fig. 8. The state of the car is decomposed as $z(t) = (x(t), y(t), \theta(t), v(t))$, where (dropping the dependency w.r.t. time for simplicity)

1. x, y denote the position of the car on the plane,
2. θ denotes the angle between the orientation of the car and the x-axis, a.k.a. the yaw,
3. v denotes the longitudinal speed.

The car is controlled through $u(t) = (a(t), \delta(t))$, where

1. a is the longitudinal acceleration of the car,
2. δ is the steering angle.

For a car of length L, the continuous time dynamics are then

$$\dot{x} = v\cos\theta \qquad \dot{y} = v\sin\theta \qquad \dot{\theta} = v\tan(\delta)/L \qquad \dot{v} = a. \tag{51}$$

Bicycle model. We consider the model presented by Liniger et al. (2015) recalled below and illustrated in Fig. 8. In this model, the state of the car at time t is decomposed as $z(t) = (x(t), y(t), \theta(t), v_x(t), v_y(t), \omega(t))$ where

1. x, y denote the position of the car on the plane,
2. θ denotes the angle between the orientation of the car and the x-axis, a.k.a. the yaw,
3. v_x denotes the longitudinal speed,
4. v_y denotes the lateral speed,
5. ω denotes the derivative of the orientation of the car, a.k.a. the yaw rate.

The control variables are analogous to the simple model, i.e., $u(t) = (a(t), \delta(t))$, where

1. a is the PWM duty cycle of the car, this duty cycle can be negative to take into account braking,
2. δ is the steering angle.

These controls act on the state through the following forces.

1. A longitudinal force on the rear wheels, denoted $F_{r,x}$ modeled using a motor model for the DC electric motor as well as a friction model for the rolling resistance and the drag

$$F_{r,x} = (C_{m1} - C_{m2}v_x)a - C_{r0} - C_{rd}v_x^2$$

where $C_{m1}, C_{m2}, C_{r0}, C_{rd}$ are constants estimated from experiments, see the Appendix.

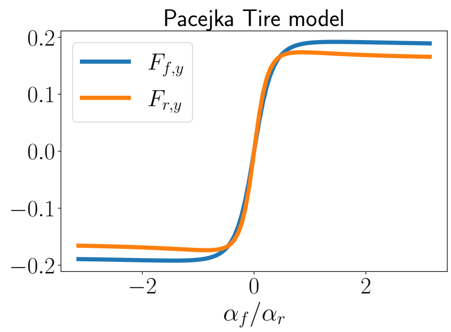

Figure 9: Pacejka model of the friction on the tires as a function of the slip angles

2. Lateral forces on the front and rear wheels, denoted $F_{f,y}, F_{y,r}$ respectively, modeled using a simplified Pacejka Tire Model

$$F_{f,y} = D_f \sin(C_f \arctan(B_f \alpha_f)) \quad \text{where } \alpha_f = \delta - \operatorname{arctan2}\left(\frac{\omega l_f + v_y}{v_x}\right)$$
$$F_{r,y} = D_r \sin(C_r \arctan(B_r \alpha_r)) \quad \text{where } \alpha_r = \operatorname{arctan2}\left(\frac{\omega l_r - v_y}{v_x}\right)$$

where α_f, α_r are the slip angles on the front and rear wheels respectively, l_f, l_r are the distance from the center of gravity to the front and the rear wheel respectively and the constants $B_r, C_r, D_r, B_f, C_f, D_f$ define the exact shape of the semi-empirical curve, presented in Fig. 9.

The continuous time dynamics are then (dropping the dependency w.r.t. time for simplicity):

$$\begin{aligned} \dot{x} &= v_x \cos\theta - v_y \sin\theta & \dot{v}_x &= \frac{1}{m}(F_{r,x} - F_{f,y}\sin\delta) + v_y\omega \\ \dot{y} &= v_x \sin\theta + v_y \cos\theta & \dot{v}_y &= \frac{1}{m}(F_{r,y} + F_{f,y}\cos\delta) - v_x\omega \\ \dot{\theta} &= \omega & \dot{\omega} &= \frac{1}{I_z}(F_{f,y} l_f \cos\delta - F_{r,y} l_r) \end{aligned} \tag{52}$$

where m is the mass of the car and I_z is the inertia.

10.3.2 Costs

Tracks. We consider tracks that are given as a continuous curve, namely a cubic spline approximating a set of points. As a result, for any time t, we have access to the corresponding point $\hat{x}(t), \hat{y}(t)$ on the curve. The track we consider is a simple track illustrated in Fig. 10.

Tracking cost. A simple cost on the states is

$$c_t(z_t) = \|x_t - \hat{x}(\Delta v^{\text{ref}} t)\|_2^2 + \|y_t - \hat{y}(\Delta v^{\text{ref}} t)\|_2^2 \quad \text{for } t = 1, \ldots, \tau, \tag{53}$$

for $z_t = (x_t, y_t)$, where Δ is some discretization step and v^{ref} is some reference speed. The cost above is the one we choose for the simple model of a car. The disadvantage of such costs is that it enforces the car to follow the track at a constant speed which may not be physically realizable. We consider in the following a contouring cost as done by Liniger et al. (2015).

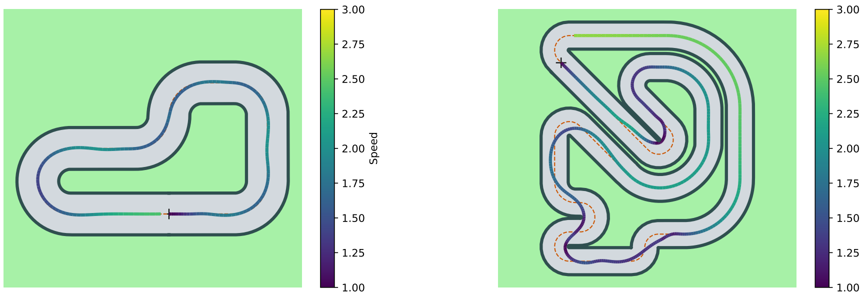

Figure 10: Simple and complex tracks used with a trajectory computed on the bicycle model (52).

Ideal cost. Given a track parameterized in continuous time, an ideal cost is to enforce the car to be as close as possible to the track, while moving along the track as fast as possible. Formally, define the distance from the car at position (x, y) to the track defined by the curve $\hat{x}(t), \hat{y}(t)$ as

$$d(x, y) = \min_{t \in \mathbb{R}} \quad \sqrt{((x - \hat{x}(t))^2 + (y - \hat{y}(t))^2}.$$

Denoting $t^* = t(x, y) = \arg\min_{t \in \mathbb{R}} \quad (x - \hat{x}(t))^2 + (y - \hat{y}(t))^2$, the reference time on the track for a car at position (x, y), the distance $d(x, y)$ can be expressed as

$$d(x, y) = \sin(\theta(t^*))\,(x - \hat{x}(t^*)) - \cos(\theta(t^*))\,(y - \hat{y}(t^*))\,,$$

where $\theta(t) = \frac{\partial \hat{y}(t)}{\partial \hat{x}(t)}$ is the angle of the track with the x-axis. The distance $d(x, y)$ is illustrated in Fig. 11. An ideal cost for the problem is then defined as $h(z) = h(x, y) = d(x, y)^2 - t(x, y)$, which enforces the car to be close to the track by minimizing $d(x, y)^2$, and also encourages the car to go as far as possible by adding the term $-t(x, y)$.

Contouring and lagging costs. The computation of t^* involves solving an optimization problem and is not practical. As Liniger et al. (2015), we rather augment the states with a flexible reference time. Namely, we augment the state of the car by adding a variable s whose objective is to approximate the reference time t^*. The cost is then decomposed into the *contouring cost* and the *lagging cost* illustrated in Fig. 11 and defined as

$$\begin{aligned}
e_c(x, y, s) &= \sin(\theta(s))\,(x - \hat{x}(s)) - \cos(\theta(s))\,(y - \hat{y}(s)) \\
e_l(x, y, s) &= -\cos(\theta(s))\,(x - \hat{x}(s)) - \sin(\theta(s))\,(y - \hat{y}(s))\,.
\end{aligned}$$

Rather than encouraging the car to make the most progress on the track, we enforce them to keep a reference speed. Namely we consider an additional penalty of the form $\|\dot{s} - v^{\text{ref}}\|_2^2$ where v^{ref} is a parameter chosen in advance. Moreover we want the reference time s not to go backward in time, namely we add a log-barrier term $-\varepsilon \log(\dot{s})$ for $\varepsilon = 10^{-6}$.

Finally, we let the system control the reference time through its second order derivative $\ddot{s}$. Overall this means that we augment the state variable by adding the variables s and $\nu := v_s$ and that we augment the control variable by adding the variable $\alpha := a_s$ such that the discretized problem is written for, e.g., the bicycle model, as

$$\begin{aligned}
\min_{(a_0,\delta_0,\alpha_0),\ldots,(a_{\tau-1},\delta_{\tau-1},\alpha_{\tau-1})} \quad & \sum_{t=0}^{\tau-1} \rho_c e_c(x_t, y_t, s_t)^2 + \rho_l e_l(x_t, y_t, s_t)^2 + \rho_v \|v_{s,t} - v^{\text{ref}}\|_2^2 - \varepsilon \log \nu_t \\
\text{s.t.} \quad & x_{t+1}, y_{t+1}, \theta_{t+1}, v_{x,t+1}, v_{y,t+1}, \omega_{t+1} = f(x_t, y_t, \theta, v_{x,t}, v_{y,t}, \omega_t, \delta_t, a_t) \\
& s_{t+1} = s_t + \Delta \nu_t, \quad \nu_{t+1} = \nu_t + \Delta \alpha_t \\
& z_0 = \hat{z}_0 \quad s_0 = 0 \quad \nu_0 = v^{\text{ref}},
\end{aligned}$$

where f is a discretization of the continuous time dynamics, δ is a discretization step and $\hat{z}_0$ is a given initial state where z_0 regroups all state variables at time 0 (i.e. all variables except a_0, δ_0).

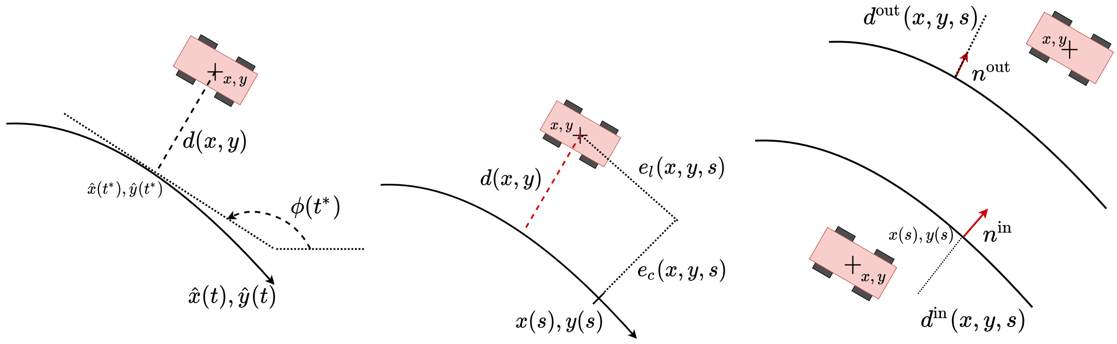

Figure 11: Left: Distance to the track, Middle: Approx. by contouring and lagging costs, Right: Border costs.

This cost is defined by the parameters $\rho_c, \rho_l, \rho_v, v^{\text{ref}}$ which are fixed in advance. The larger the ρ_c, the closer the car to the track. The larger the ρ_l, the closer the car to its reference time s. In practice, we want the reference time to be a good approximation of the ideal projection of the car on the track so ρ_l should be chosen large enough. On the other hand, varying ρ_c allows to have a car that is either conservative and potentially slow or a car that is fast but inaccurate, i.e., far from the track. The most important aspect of the trajectory is to ensure that the car remains inside the borders of the track defined in advance.

Border costs. To enforce the car to remain inside the track defined by some borders, we penalize the approximated distance of the car to the border when it goes outside the border as $e_b(x, y, s) = e_b^{\text{in}}(x, y, s) + e_b^{\text{out}}(x, y, s)$ with

$$\begin{aligned} e_b^{\text{in}}(x, y, s) &= \max((w + d^{\text{in}}(x, y, s))^2, 0) \qquad & d^{\text{in}}(x, y, s) &= -(z - z^{\text{in}}(s))^\top n^{\text{in}}(s) \\ e_b^{\text{out}}(x, y, s) &= \max((w + d^{\text{out}}(x, y, s))^2, 0) \qquad & d^{\text{out}}(x, y, s) &= (z - z^{\text{out}}(s))^\top n^{\text{out}}(s) \end{aligned} \tag{54}$$

for $z = (x, y)$, where $n^{\text{in}}(s)$ and $n^{\text{out}}(s)$ denote the normal at the borders at time s and w is the width of the car. In practice, we use smooth approximation of the max function in Eq. (54). The normals $n^{\text{in}}(s)$ and $n^{\text{out}}(s)$ can easily be computed by derivating the curves defining the inner and outer borders. These costs are illustrated in Fig. 11. Note the potential mismatch between the computed distance of the car to the border and the real distance. This mismatch is due to using the free parameter s rather than the real projection of the car to the border.

Constrain controls. We constrain the steering angle to be between $[-\pi/3, \pi/3]$ by parameterizing the steering angle as

$$\delta(\tilde{\delta}) = \frac{2}{3} \arctan(\tilde{\delta}) \quad \text{for } \tilde{\delta} \in \mathbb{R}.$$

Similarly, we constrain the acceleration a to be between $[c, d]$ (with $c = -0.1, d = 1.$), by parameterizing it as

$$a(\tilde{a}) = (d - c)\,\text{sig}(4\tilde{a}/(d - c)) + c$$

with $\text{sig} : x \to 1/(1 + e^{-x})$ the sigmoid function. The final set of control variables is then $\tilde{a}, \tilde{\delta}, \alpha$.

Control costs. In both cases, we use a square regularization on the control variables of the system, i.e., the cost on the control variables is $r_t(u_t) = \lambda \|u_t\|_2^2$ for some $\lambda \geq 0$ where u_t are the control variables at time t.

Overall contouring cost. The whole problem with contouring cost is then

$$\begin{aligned}\min_{(\tilde a_0,\tilde\delta_0,\alpha_0),\ldots,(\tilde a_{\tau-1},\tilde\delta_{\tau-1},\tilde\alpha_{\tau-1})} \quad & \sum_{t=0}^{\tau-1}\Big[\rho_c e_c(x_t,y_t,s_t)^2+\rho_l e_l(x_t,y_t,s_t)^2+\rho_v\|v_{s,t}-v^{\mathrm{ref}}\|_2^2-\varepsilon\log(\nu_t) \\ & \qquad +\rho_b e_b(x_t,y_t,s_t)^2+\lambda(\tilde a_t^2+\tilde\delta_t^2+\alpha_t^2)\Big] \qquad (55)\\ \text{s.t.}\quad & x_{t+1},y_{t+1},\theta_{t+1},v_{x,t+1},v_{y,t+1},\omega_{t+1}=f(x_t,y_t,\theta_t,v_{x,t},v_{y,t},\omega_t,\delta_t(\tilde\delta_t),a_t(\tilde a_t))\\ & s_{t+1}=s_t+\Delta\nu_t,\quad \nu_{t+1}=\nu_t+\Delta\alpha_t\\ & z_0=\hat z_0\quad s_0=0\quad \nu_0=v^{\mathrm{ref}},\end{aligned}$$

with parameters $\rho_c,\rho_l,\rho_v,v^{\mathrm{ref}},\rho_b,\lambda$ and f given in Eq. (52).

10.4 Results

All the following plots are in log-scale where the y-axis is computed as $\log\big((\mathcal{J}(\boldsymbol{u}^{(k)})-\mathcal{J}^*)/(\mathcal{J}(\boldsymbol{u}^{(0)})-\mathcal{J}^*)\big)$ with $\mathcal{J}$ the objective, $\boldsymbol{u}^{(k)}$ the set of controls at iteration k, and $\mathcal{J}^*=\min_{\boldsymbol{u}\in\mathbb{R}^{\tau n_u}}\mathcal{J}(\boldsymbol{u}))$ estimated from running the algorithms for more iterations than presented. The acronyms (GD, GN, NE, DDP-LQ, DDP-Q) correspond to the taxonomy of algorithms presented in Fig. 2.

For the realistic model of a car, we observed that gradient oracles were not reliable in the sense that the algorithm stopped after just 1 iteration. The highly non-linear dynamics may lead to vanishing/exploding gradients which hinder the use of gradients.

10.4.1 Linear Quadratic Approximations

In Fig. 12, we compare a gradient descent and nonlinear control algorithms with linear quadratic approximations, i.e., GN or DDP-LQ with different steps (i.e., moving along the oracle direction or using a regularized step) . We observe that GN or DDP-LQ always outperform a simple gradient descent algorithm. Similarly, we observe that the differential dynamic programming approach generally outperforms its classical counterpart, i.e., Gauss-Newton, for the same steps (descent direction or regularized steps). Finally, in terms of steps, for GN, taking a descent direction seems generally better than taking regularized steps for easy problems such as the pendulum. The regularized steps can be advantageous for harder problems as illustrated in the control of a bicycle model of a car. Overall DDP-LQ with regularized steps appears to outperform all other algorithms. In particular, regularized steps appear to outperform DDP-LQ with descent directions.

In Fig. 13, we plot the same algorithms but with respect to time. We observe that, though descent directions generally outperform their regularized counterparts in terms of iterations, they can be more demanding in terms of time, as numerous function evaluations are required to ensure a sufficient decrease. On the other hand, regularized steps allow for faster line-searches as they incorporate previous stepsizes. The line-searches for descent directions may be refined by using alternative line-search strategies.

Finally, in Fig. 14, we plotted the stepsizes taken by the algorithms for the pendulum and the simple model of a car. On the the pendulum example, the stepsizes used by moving along the oracle direction quickly tend to 1 which means that the algorithms (GN or DDP-LQ) are then taking the largest possible stepsize for this strategy. On the other hand, for the regularized steps, on the pendulum example, the regularizations (i.e. the inverse of the stepsizes) quickly converge to 0, which means that, as the number of iterations increase, the regularized and the descent directions strategy coincide. For the car example, the step sizes for the descent directions also converge quickly to one but then oscillate between one and approximately one half. Similarly, the regularized steps do not always exhibit a regularization that tends to 0.

10.4.2 Quadratic Approximations

In Fig. 15, 16, 17, we observe that the nonlinear control algorithms with quadratic approximations follow the same trends as their linear quadratic counterparts with the DDP approach with regularized steps generally outperforming all other algorithms based on quadratic approximations.

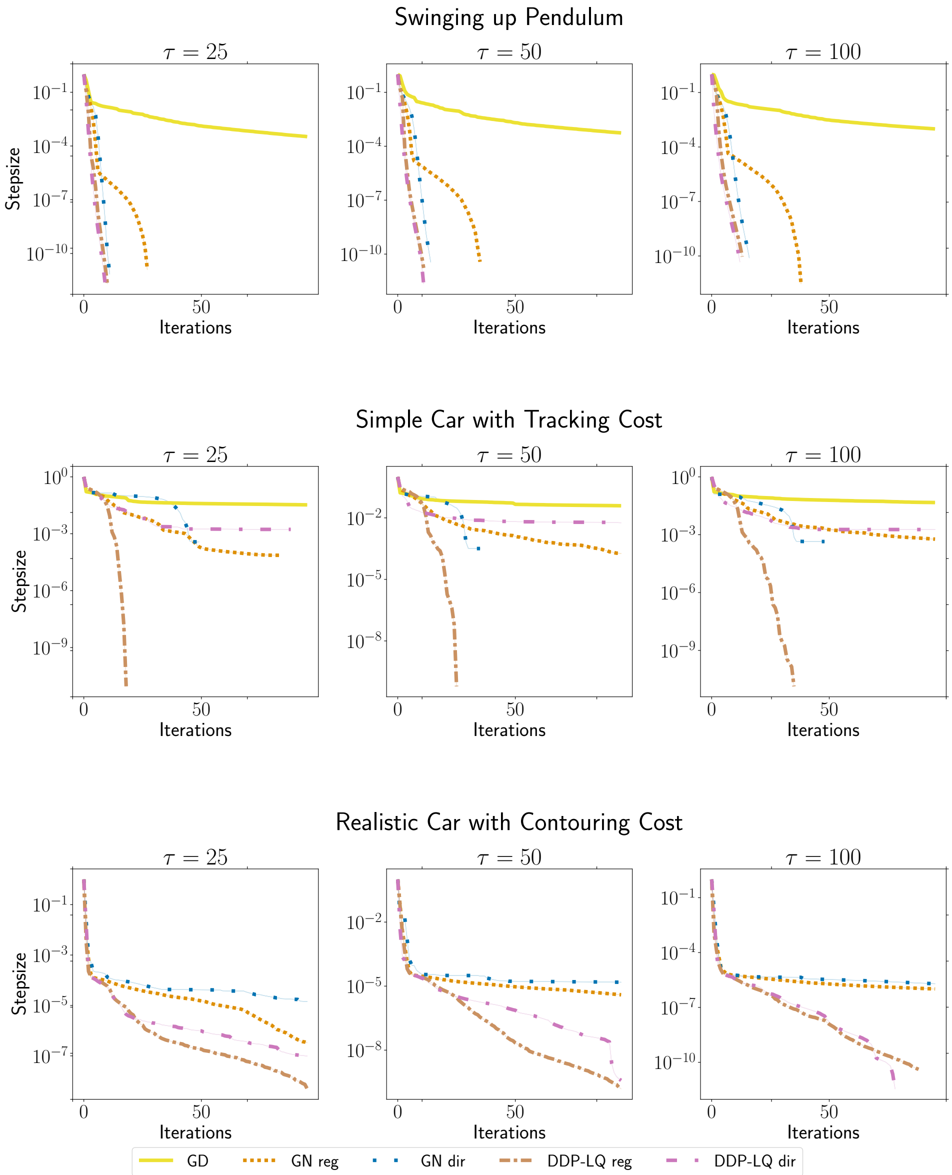

Figure 12: Convergence of algorithms with linear quadratic approximations in iterations.

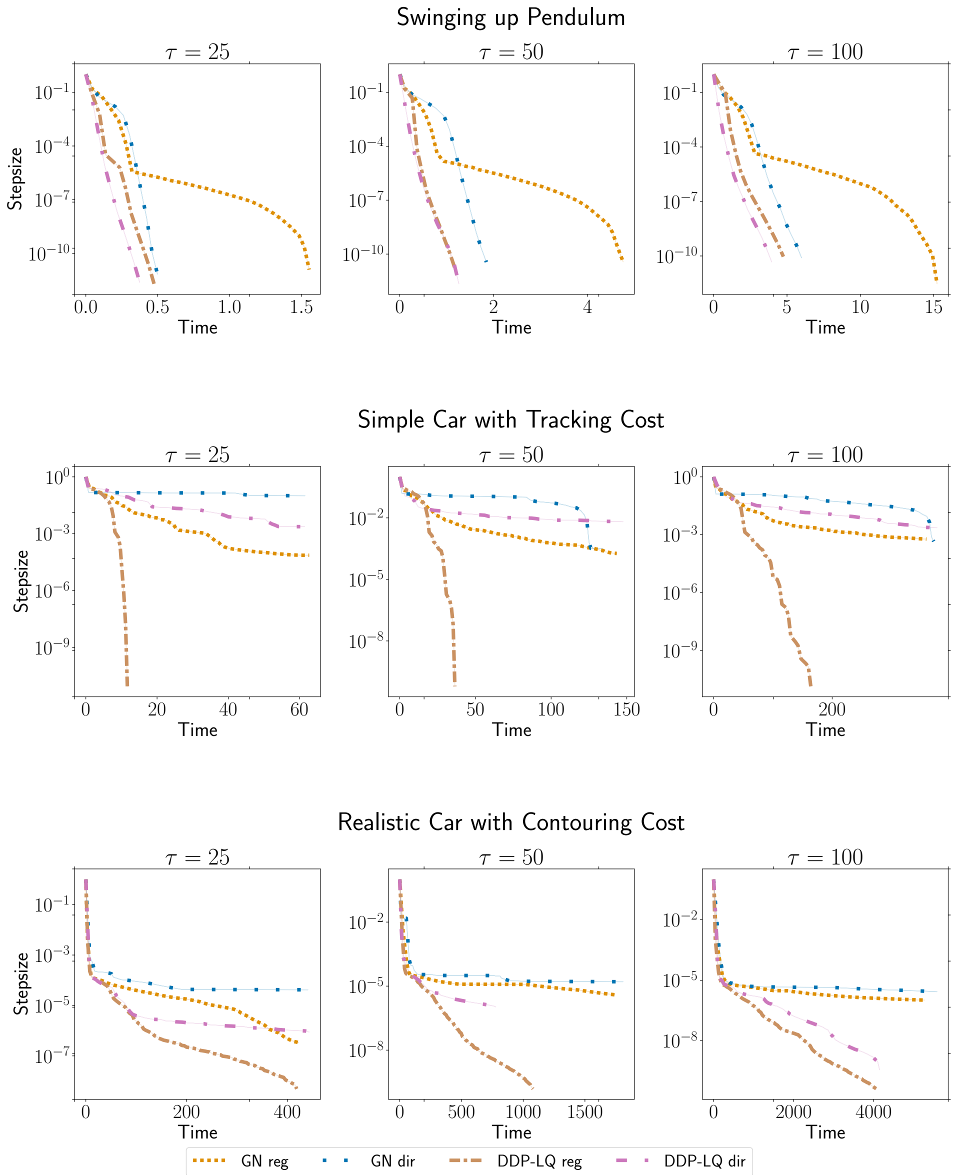

Figure 13: Convergence of algorithms with linear quadratic approximations in time.

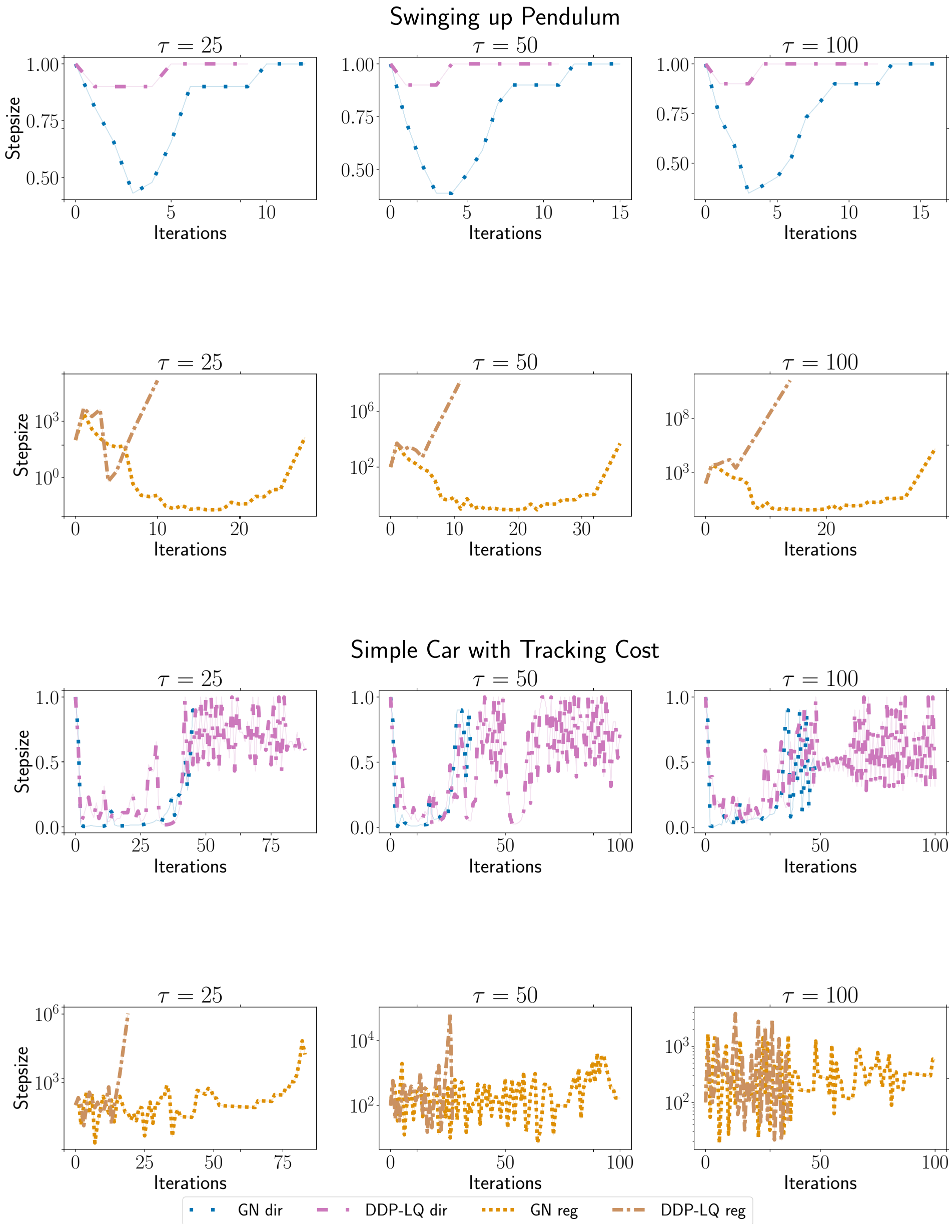

Figure 14: Stepsizes taken along the iterations for linear quadratic approximations.

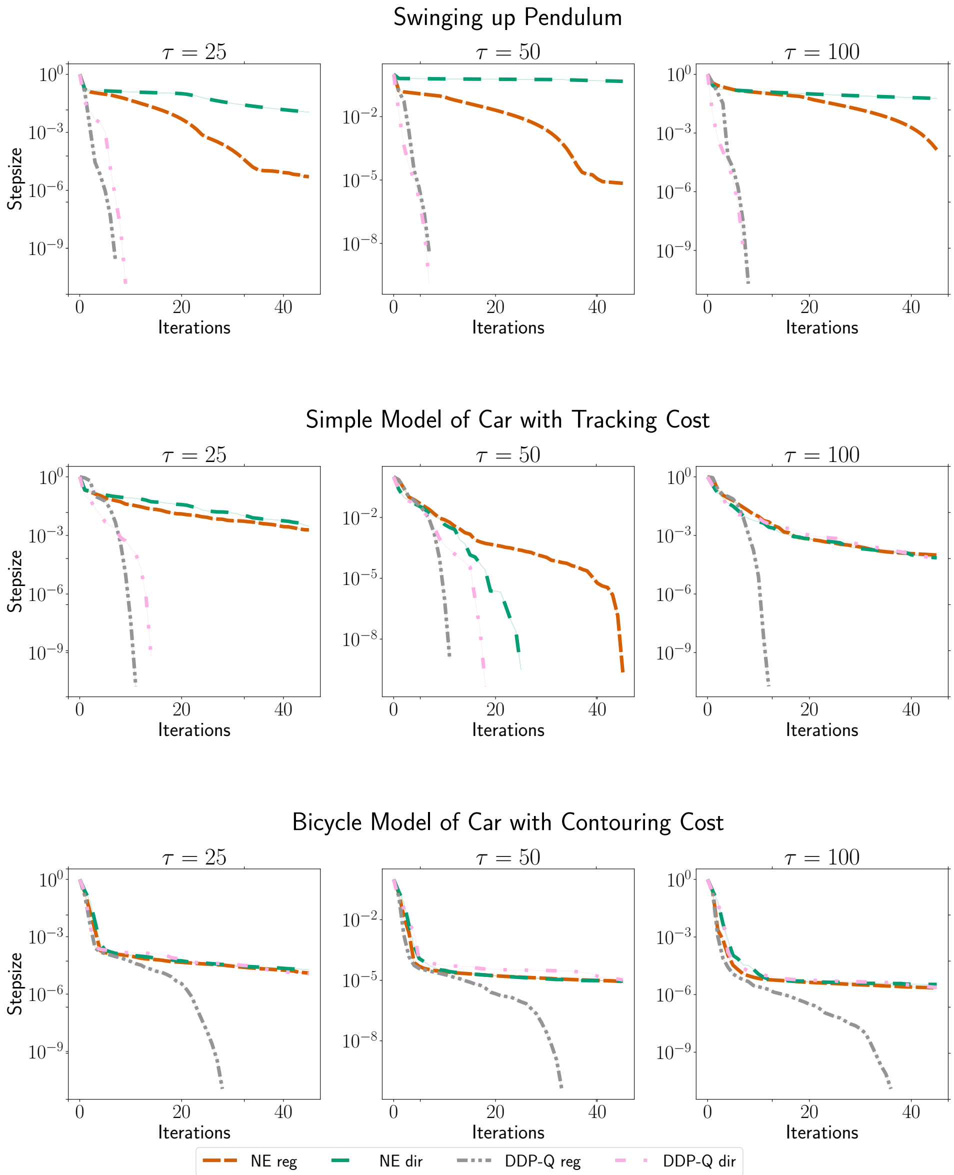

Figure 15: Convergence of algorithms with quadratic approximations in iterations.

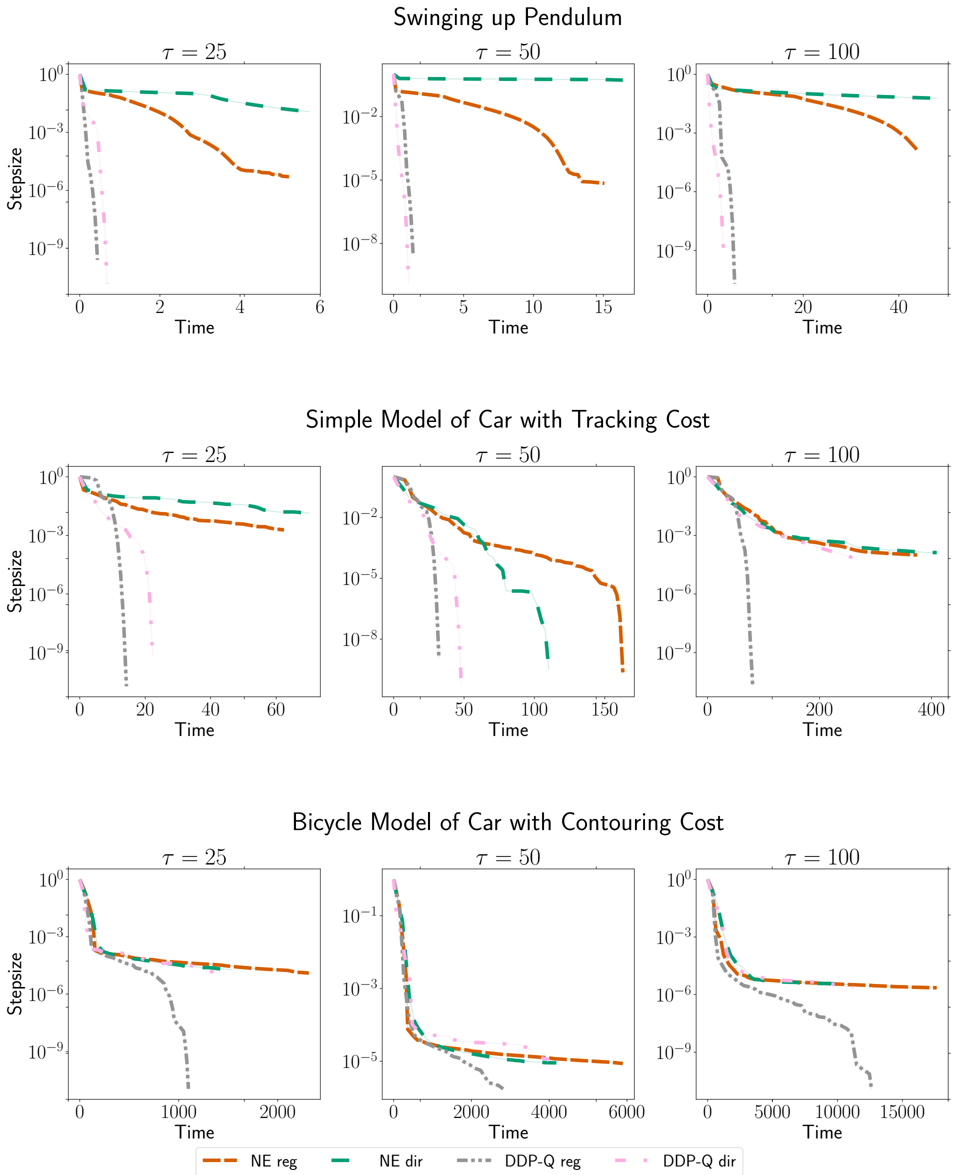

Figure 16: Convergence of algorithms with quadratic approximations in time.

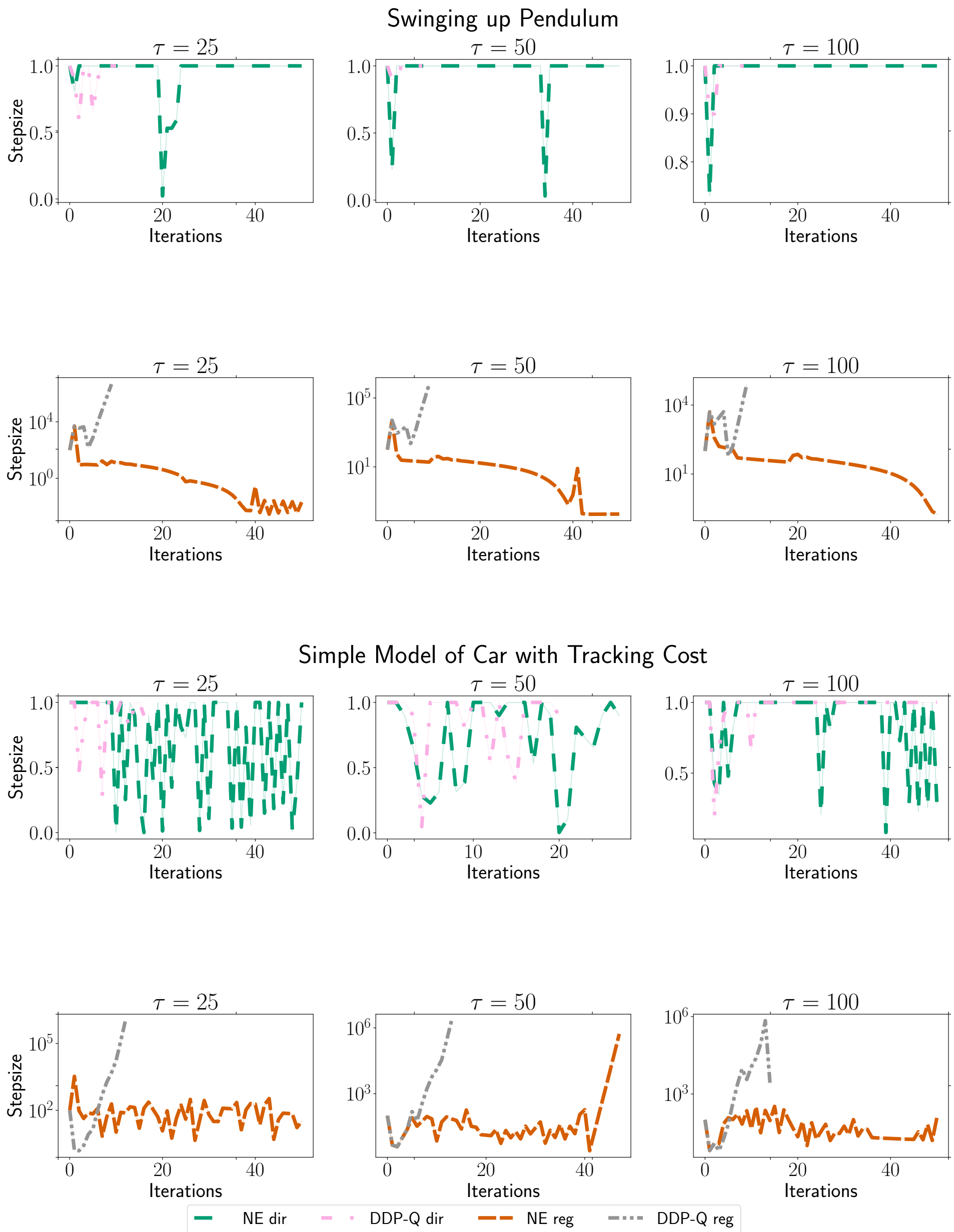

Figure 17: Stepsizes taken along the iterations for quadratic approximations.

Acknowledgments. This work was supported by NSF DMS-1839371, DMS-2134012, CCF-2019844, CIFAR-LMB, NSF TRIPODS II DMS-2023166 and faculty research awards. The authors thank Dmitriy Drusvyatskiy, Alexander Liniger, Krishna Pillutla and John Thickstun for fruitful discussions on the paper and their help to develop the numerical experiments.

References

Arrow, K. (1968). Applications of control theory to economic growth. In *Lectures on Applied Mathematics*, volume 12 (Mathematic of the Decision Science, Part 2).

Bellman, R. (1971). *Introduction to the mathematical theory of control processes*, volume 2. Academic press.

Betts, J. (2010). *Practical methods for optimal control and estimation using nonlinear programming*. SIAM.

Boyd, S. and Vandenberghe, L. (1997). Semidefinite programming relaxations of non-convex problems in control and combinatorial optimization. In *Communications, Computation, Control, and Signal Processing*, pages 279–287. Springer.

Clarke, F. (1979). Optimal control and the true hamiltonian. *SIAM Review*, 21(2):157–166.

Dunn, J. and Bertsekas, D. (1989). Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems. *Journal of Optimization Theory and Applications*, 63(1):23–38.

Jacobson, D. and Mayne, D. (1970). *Differential Dynamic Programming*. Elsevier.

Kamien, M. and Schwartz, N. (1971). Sufficient conditions in optimal control theory. *Journal of Economic Theory*, 3(2):207–214.

Li, W. and Todorov, E. (2007). Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic system. *International Journal of Control*, 80(9):1439–1453.

Liao, L.-Z. and Shoemaker, C. (1991). Convergence in unconstrained discrete-time differential dynamic programming. *IEEE Transactions on Automatic Control*, 36(6):692–706.

Liao, L.-Z. and Shoemaker, C. A. (1992). Advantages of differential dynamic programming over Newton's method for discrete-time optimal control problems. Technical report, Cornell University.

Liniger, A., Domahidi, A., and Morari, M. (2015). Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647.

Mangasarian, O. (1966). Sufficient conditions for the optimal control of nonlinear systems. *SIAM Journal on Control*, 4(1):139–152.

Mayne, D. and Polak, E. (1975). First-order strong variation algorithms for optimal control. *Journal of Optimization Theory and Applications*, 16(3):277–301.

Murray, D. and Yakowitz, S. (1984). Differential dynamic programming and Newton's method for discrete optimal control problems. *Journal of Optimization Theory and Applications*, 43(3):395–414.

Nganga, J. and Wensing, P. (2021). Accelerating second-order differential dynamic programming for rigid-body systems. *IEEE Robotics and Automation Letters*, 6(4):7659–7666.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

Pantoja, J. (1988). Differential dynamic programming and Newton's method. *International Journal of Control*, 47(5):1539–1553.

Pontryagin, L., Boltyansky, V., Gamkrelidze, R., and Mischenko, E. (1963). The mathematical theory of optimal processes. *Wiley-Interscience*.

Rao, C., Wright, S., and Rawlings, J. (1998). Application of interior-point methods to model predictive control. *Journal of optimization theory and applications*, 99(3):723–757.

Roulet, V., Srinivasa, S., Drusvyatskiy, D., and Harchaoui, Z. (2019). Iterative linearized control: stable algorithms and complexity guarantees. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5518–5527.

Sideris, A. and Bobrow, J. (2005). An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems. In *Proceedings of the 2005 American Control Conference*, pages 2275–2280.

Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913.

Tassa, Y., Mansard, N., and Todorov, E. (2014). Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE.

Wright, S. (1990). Solution of discrete-time optimal control problems on parallel computers. *Parallel Computing*, 16(2-3):221–237.

Wright, S. (1991a). Partitioned dynamic programming for optimal control. *SIAM Journal on optimization*, 1(4):620–642.

Wright, S. (1991b). Structured interior point methods for optimal control. In *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*, pages 1711–1716. IEEE.

A Experiments Details

The code is available at https://github.com/vroulet/ilqc. We add for ease of reference, the hyper-parameters used for each setting.

Pendulum. The constants we used in the experiment to implement the pendulum example are the following.

1. mass $m = 1$,
2. gravitational constant $g = 10$,
3. length of the blob $l = 1$,
4. friction coefficient $\mu = 0.1$,
5. speed regularization $\rho = 0.1$,
6. control regularization $\lambda = 10^{-6}$,
7. total time of the movement $T = 2$.

Simple Car with Tracking Cost. The constants we used in the experiment to implement the simple model of a car (51) with a tracking cost (53) are the following.

1. length of the car $L = 1$,
2. reference speed $v^{ref} = 3$,
3. control regularization $\lambda = 10^{-6}$,
4. total time of the movement $T = 1$.

Bicycle Model of a Car with a Contouring Objective. The parameters of the dynamics (52) are drawn from Liniger et al. (2015) and available in the code. We precise here the constants we sue for the cost defined in (55).

1. contouring error penalty $q_c = 0.1$,
2. lagging error penalty $q_l = 10$,
3. reference speed penalty $q_v = 0.1$,
4. barrier error penalty $q_b = 100$,
5. refrence speed $v^{\text{ref}} = 3$,
6. control regularization $\lambda = 10^{-6}$,
7. total time of the movement $T = 1$.