Trajectory Design for Space Systems

EN.530.626 (Fall 2025)

Lecture 26

Instructor: Prof. Abhishek Cauligi
Course Assistant 1: Arnab Chatterjee
Course Assistant 2: Mark Gonzales

Agenda

- Last time: discussed connections between optimal control and reinforcement learning
- Today: differentiable optimization
  - What
  - How
  - Why

Computing gradients

Gradients with respect to decision variables

- We have focused on computing gradients with respect to *decision variables*

$$\min_x \quad F(x) \quad \longleftrightarrow \quad J_x F(x)$$

Computing gradients

Gradients with respect to decision variables

- We have focused on computing gradients with respect to decision variables

$$\min_x \quad F(x) \quad \longleftrightarrow \quad J_x F(x)$$

- For constrained optimization, this extended to the dual variables

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0, \\ & h(x) = 0. \end{aligned} \quad \longrightarrow \quad \begin{aligned} \min_x \quad & f(x) + \Phi(s) \\ \text{s.t.} \quad & g(x) + s = 0, \\ & h(x) = 0, \\ & s \geq 0. \end{aligned} \quad \longrightarrow \quad \mathscr{L}(x, y, z) = f(x) + \Phi(s) + y^T h(x) + z^T (g(x) + s)$$

Computing gradients

Gradients with respect to decision variables

- We have focused on computing gradients with respect to decision variables

$$\min_x \quad F(x) \quad \longleftrightarrow \quad J_x F(x)$$

- For constrained optimization, this extended to the dual variables

$$r(x, y, z, s) = \begin{pmatrix} \nabla_x \mathscr{L} \\ \nabla_y \mathscr{L} \\ \nabla_z \mathscr{L} \\ \nabla_s \mathscr{L} \end{pmatrix} = 0$$

Computing gradients

Gradients with respect to parameters

- There are also situations that arise when we compute gradients with respect to the *parameters* of the function

$$\min_{\theta} \quad F(x, \theta) \longleftrightarrow J_{\theta} F(x, \theta)$$

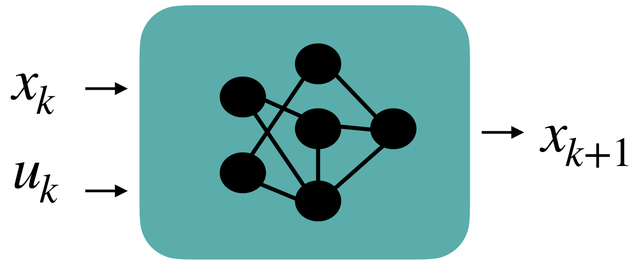
Computing gradients

Gradients with respect to parameters

- There are also situations that arise when we compute gradients with respect to the *parameters* of the function

$$\min_{\theta} \quad F(x, \theta) \longleftrightarrow J_\theta F(x, \theta)$$

- Recall: model-based reinforcement learning requires learning ϕ in $\hat{f}_\phi$



Estimate $\hat{f}_\phi(x_k, u_k)$

$$\mathscr{L}(\mathscr{D}) = \frac{1}{N_d} \sum_{i=1}^{N_d} \| x_{k+1}^{(i)} - \hat{f}_\phi(x_k^{(i)}, u_k^{(i)}) \|_2^2$$

Need to compute $\nabla_\phi \mathscr{L}(\mathscr{D})$

Computing gradients through MPC

Gradients through MPC

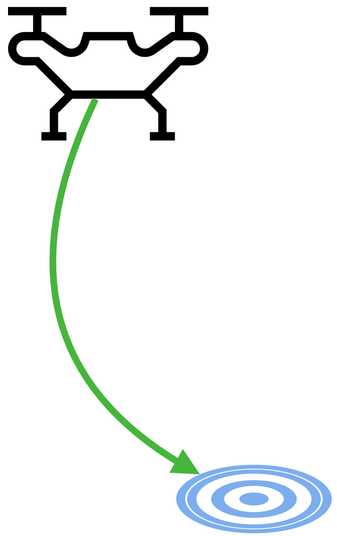
- How can we compute gradients of an optimization problem with respect to θ?

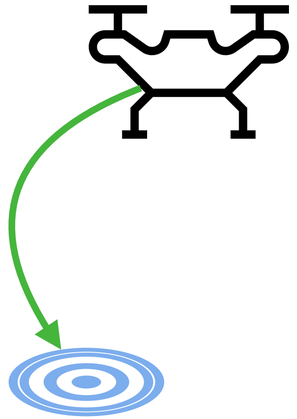
$$\theta \longrightarrow \begin{aligned} \min_{x} \quad & f(x;\theta) \\ \text{s.t.} \quad & g(x;\theta) \leq 0, \\ & h(x;\theta) = 0. \end{aligned} \longrightarrow (x^*, y^*, z^*, s^*)$$

Computing gradients through MPC

Physical intuition

How does a small change in $x_0 = x_{\text{init}}$ change the optimal trajectory x^*?





Generalize this to: dynamics, cost function, constraints, etc.

Implicit function theorem

- Suppose $F \in \mathscr{C}^1$ and $F : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_x}$. Given (x^*, θ^*) such that $F(x^*, \theta^*) = 0$, if $J_x F(x^*, \theta^*)$ is invertible, then:

  (1) There exists a neighborhood $U \subset \mathbb{R}^{n_\theta}$ of θ^* and a differentiable function $x(\theta)$ such that $F(x(\theta), \theta) = 0$ for all $\theta \in U$, and $x(\theta)$ is the unique solution in a neighborhood of x^*.

  (2) $x(\,\cdot\,)$ is $\mathscr{C}^1$ differentiable on U with

$$J_\theta x(\theta) = -\left(J_x F(x, \theta)\right)^{-1} J_\theta F(x, \theta)$$

$$\frac{\partial x(\theta)}{\partial \theta} = -\left(\frac{\partial F(x, \theta)}{\partial x}\right)^{-1} \frac{\partial F(x, \theta)}{\partial \theta}$$

Implicit function theorem

- For optimization problems, let $F(x, \theta)$ be the residual function:

$$r(x, y, z, s) = \begin{pmatrix} \nabla_x \mathscr{L} \\ \nabla_y \mathscr{L} \\ \nabla_z \mathscr{L} \\ \nabla_s \mathscr{L} \end{pmatrix} = 0 \qquad \longleftrightarrow \qquad \begin{aligned} r(\bar{x}, \theta) &= 0 \\ \bar{x} &= (x, y, z, s) \end{aligned}$$

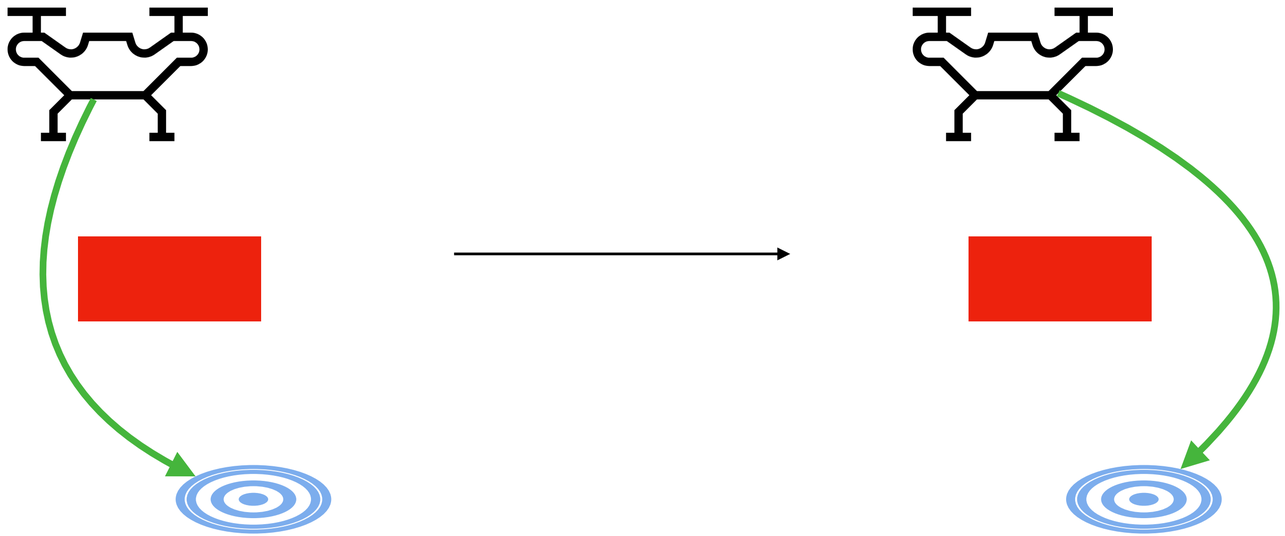
- IFT then gives:

$$\theta \longrightarrow \begin{aligned} \min_x \quad & f(x; \theta) \\ \text{s.t.} \quad & g(x; \theta) \leq 0, \\ & h(x; \theta) = 0. \end{aligned} \longrightarrow \bar{x}^*$$

$$\frac{\partial \bar{x}^*}{\partial \theta} = \left(\frac{\partial r}{\partial \bar{x}} \right)^{-1} \frac{\partial r}{\partial \theta}$$

Implicit function theorem

- How do we know the gradient exists?



A small change in x_{init} could yield a large jump in $\bar{x}^*$

Implicit function theorem

- How do we know the gradient exists?

Mathematical Programming 10 (1976) 287–311.
North-Holland Publishing Company

SENSITIVITY ANALYSIS FOR NONLINEAR PROGRAMMING USING PENALTY METHODS*

Anthony V. FIACCO
The George Washington University, Washington, U.S.A.

Received 26 July 1974
Revised manuscript received 18 August 1975

In this paper we establish a theoretical basis for utilizing a penalty-function method to estimate sensitivity information (i.e., the partial derivatives) of a local *solution* and its associated Lagrange multipliers of a large class of nonlinear programming problems with respect to a general parametric variation in the problem functions. The local solution is assumed to satisfy the second order sufficient conditions for a strict minimum. Although theoretically valid for higher order derivatives, the analysis concentrates on the estimation of the first order (first partial derivative) sensitivity information, which can be explicitly expressed in terms of the problem functions. For greater clarity, the results are given in terms of the mixed logarithmic-barrier quadratic-loss function. However, the approach is clearly applicable to *any algorithm* that generates a once differentiable "solution trajectory".

Suppose $r(\bar{x}^*, \theta^*) = 0$ is a minimizer of KKT residuals and the following three assumptions hold:

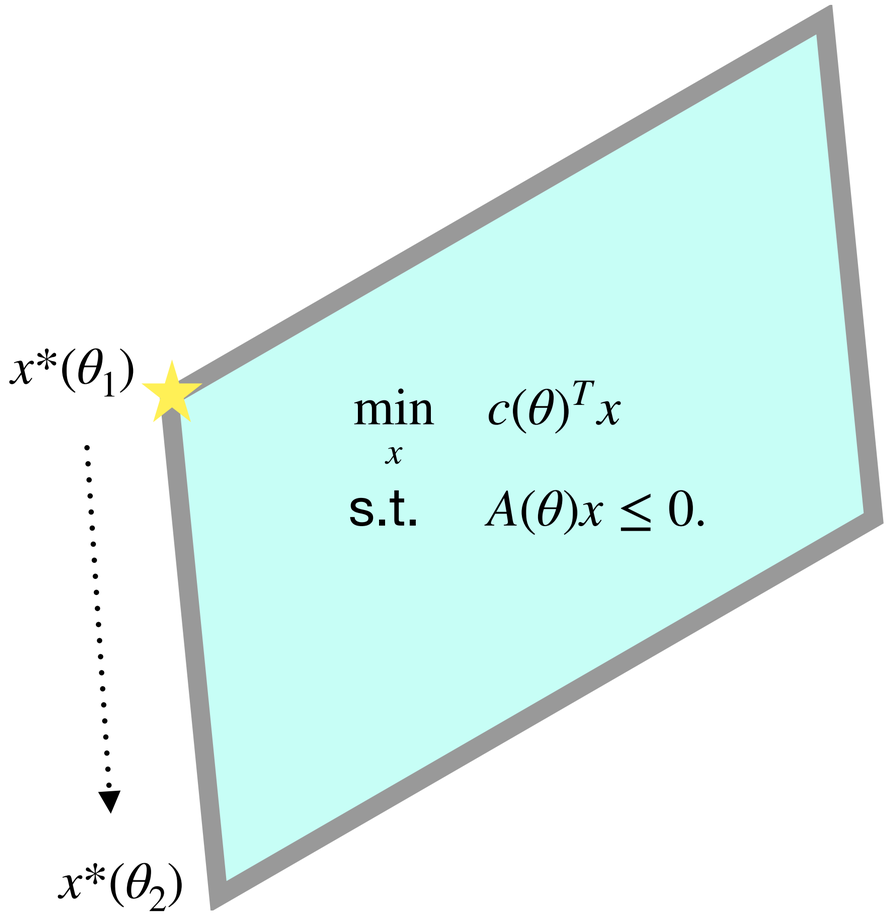
1. Second-order sufficiency conditions
2. Linear-independence constraint qualifications
3. Strict complementarity slackness

Then the IFT holds in the neighborhood U of θ^* .

Implicit function theorem

- How do we know the gradient exists?

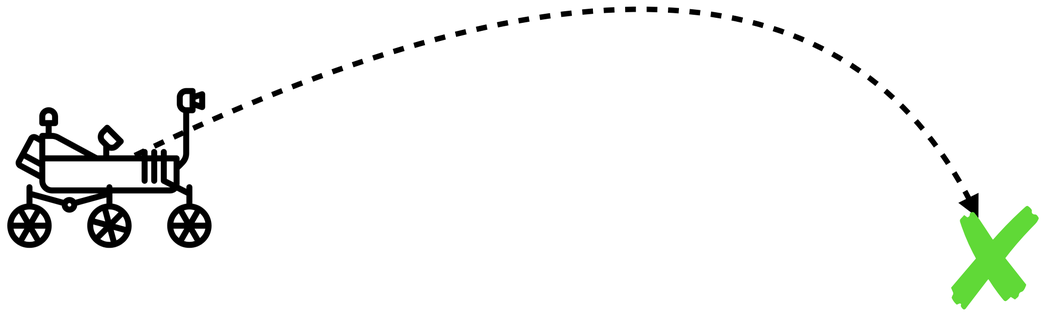
In this case, the previous theorem would state that a differentiable neighborhood does not exist in the neighborhood of θ_1 as a small change in parameters from, e.g., θ_1 to θ_2 yields a discontinuous jump in solutions $x^*(\theta_1)$ and $x^*(\theta_2)$, respectively



Case study

Actor-critic model predictive control

- Motivation: MPC is deployed to ultimately accomplish some *task*
- Can the MPC cost be automatically tuned to minimize downstream task loss?



Example: for Lunar rover driving, the MPC formulation might change across different mission modes and be subject to varying sets of constraints. How can the MPC formulation be efficiently updated to ensure the downstream task loss of reaching some goal is minimized?

Case study

Actor-critic model predictive control

- Uses an MPC-controller as the “actor” within RL actor-critic
- The “actor” network predicts the MPC cost function parameters and is trained to minimize task loss
- The aim is to update MPC cost parameters efficiently to complete downstream task of flying through track as quickly as possible

2024 IEEE International Conference on Robotics and Automation (ICRA)
May 13-17, 2024. Yokohama, Japan

Actor-Critic Model Predictive Control

Angel Romero, Yunlong Song, Davide Scaramuzza

***Abstract*— An open research question in robotics is how to combine the benefits of model-free reinforcement learning (RL)—known for its strong task performance and flexibility in optimizing general reward formulations—with the robustness and online replanning capabilities of model predictive control (MPC). This paper provides an answer by introducing a new framework called *Actor-Critic Model Predictive Control*. The key idea is to embed a differentiable MPC within an actor-critic RL framework. The proposed approach leverages the short-term predictive optimization capabilities of MPC with the exploratory and end-to-end training properties of RL. The resulting policy effectively manages both short-term decisions through the MPC-based actor and long-term prediction via the critic network, unifying the benefits of both model-based control and end-to-end learning. We validate our method in both simulation and the real world with a quadcopter platform across various high-level tasks. We show that the proposed architecture can achieve real-time control performance, learn complex behaviors via trial and error, and retain the predictive properties of the MPC to better handle out of distribution behaviour.**

SUPPLEMENTARY MATERIAL

Video of the experiments: `https://youtu.be/mQqm_vFo7e4`

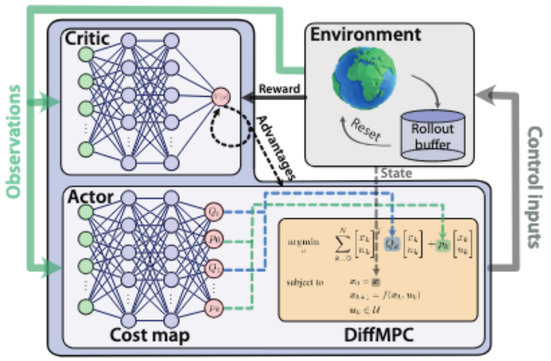


Fig. 1: A block diagram of the approach. We combine the strength of actor-critic RL and the robustness of MPC by placing a differentiable MPC as the last layer of the actor policy. At deployment time the commands for the environment are drawn from solving an MPC, which leverages the dynamics of the system and finds the optimal solution given the current state.

[10], [12]. Often, conservative assumptions about the task are made, leading to potentially sub-optimal task performance,

Case study

Actor-critic model predictive control

from a non-linear optimization problem to a Quadratic Program (QP). Therefore, a more general cost function can be written as in Eq. (2).

$$J_Q(\boldsymbol{x}) = \sum_{k=0}^{N} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T Q_k \begin{bmatrix} x_k \\ u_k \end{bmatrix} + p_k \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{2}$$

In our paper, we propose to directly search for the matrix coefficients of Eq. (2). This way, by varying Q_k and p_k, we are able to capture a larger family of problems, without suffering from the dependency on a feasible trajectory.

D. Actor-Critic Model Predictive Control

This paper proposes an Actor-Critic MPC controller architecture where the MPC is differentiable [26] and the cost function is learned end-to-end using RL. This differentiable MPC supports input constraints but not state constraints. Hence we add $\boldsymbol{u}_k \in \mathcal{U}$ in (1). The MPC block is introduced as the differentiable layer of the actor in an actor-critic

MPC loss: LQR cost objective

2) Rewards: For all the experiments, one reward term in common is the gate progress reward, which encourages fast flight through the track. The objective is to directly maximize progress toward the center of the next gate. Once the current gate is passed, the target gate switches to the next one. At each simulation time step k, the reward function is defined by:

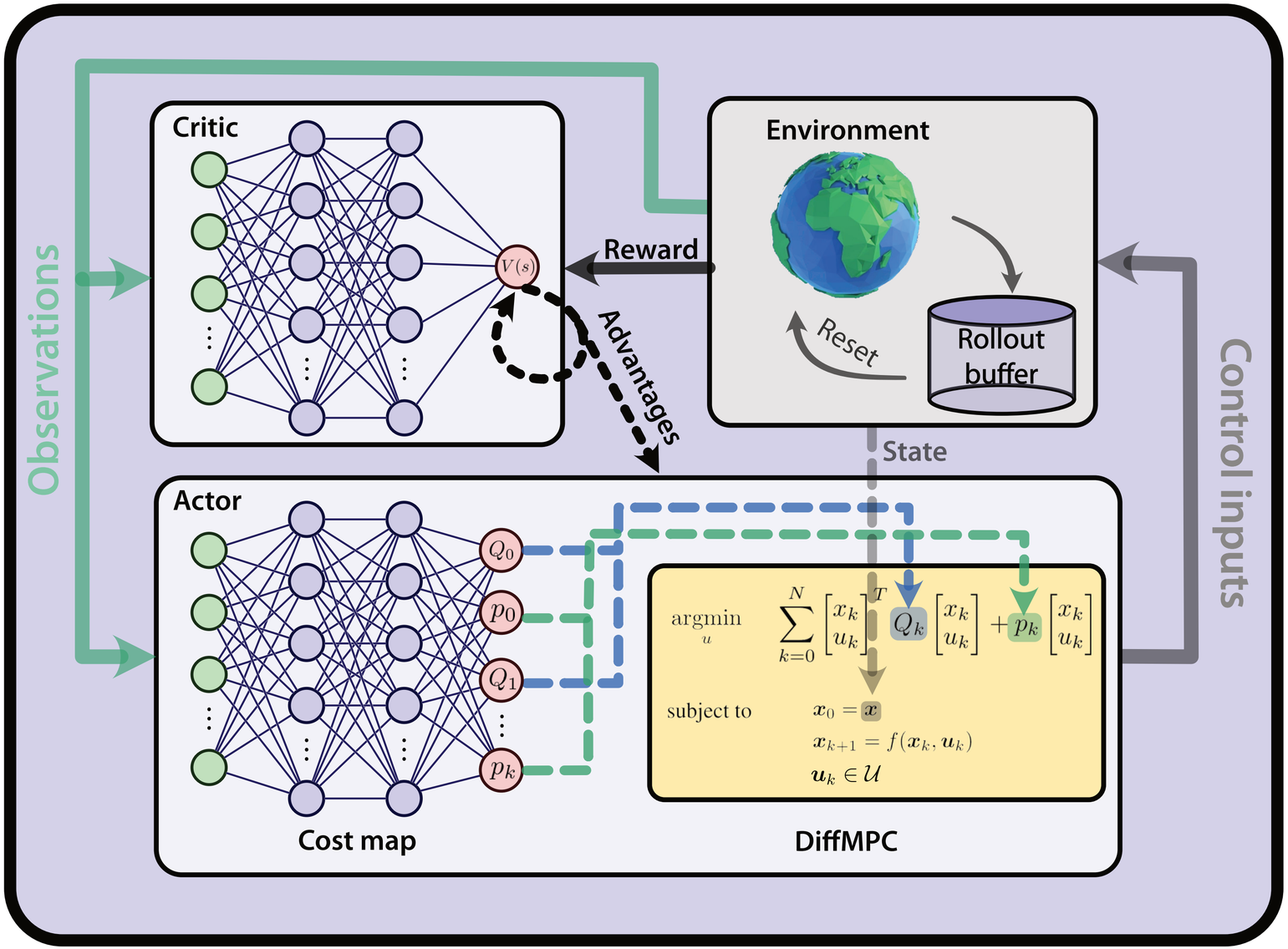
$$r(k) = \|g_k - p_{k-1}\| - \|g_k - p_k\| - b\|\boldsymbol{\omega}_k\|, \tag{5}$$

where g_k represents the target gate center, and p_k and p_{k-1} are the vehicle positions at the current and previous time steps, respectively. Here, $b\|\boldsymbol{\omega}_k\|$ is a penalty on the bodyrate multiplied by a coefficient $b = 0.01$. To discourage collisions with the environment, a penalty ($r(k) = -10.0$) is imposed when the vehicle experiences a collision. To encourage gate passing, a positive reward ($r(k) = +10.0$) is added after each gate passing. The agent is also rewarded with a positive reward ($r(k) = +10.0$) upon finishing the race.

Task loss: fast flying through the track

Romero, Song, & Scaramuzza [ICRA 2024]

AC-MPC (Ours)



Romero, Song, & Scaramuzza [ICRA 2024]

Case study

Actor-critic model predictive control

- The actor neural network predicts the Q_k and p_k parameters used in cost term
- The network outputs are propagated through an MPC solver to compute x^*
- The gradient $\nabla_\theta J(x_k)$ is computed through the MPC routine and used to train the actor NN

Algorithm 1: Actor-Critic Model Predictive Control

Input: initial neural cost map, initial value function V
for $i = 0, 1, 2, \cdots$ **do**
  Collect set of trajectories $\mathcal{D}_i\{\tau\}$ with
    $u_k \sim \mathcal{N}\{\text{diffMPC}(x_{k_i} Q(s_k), p(s_k)), \Sigma\}$
  Compute reward-to-go R_k
  Compute advantage estimates $\hat{A}_k$ based on value function $V(s_k)$
  Update the cost map by policy gradient (e.g., PPO-clip objective) and diffMPC backward [26]
  Fit value function by regression on mean-squared error
Output: Learned cost map

Romero, Song, & Scaramuzza [ICRA 2024]

Research directions

Differentiable NMPC

Differentiable Nonlinear Model Predictive Control

Jonathan Frey
University Freiburg
jonathan.frey@imtek.uni-freiburg.de

Katrin Baumgärtner
University Freiburg

Gianluca Frison
University Freiburg

Dirk Reinhardt
NTNU Trondheim

Jasper Hoffmann
University Freiburg

Leonard Fichtner
University Freiburg

Sebastien Gros
NTNU Trondheim

Moritz Diehl
University Freiburg

Abstract

The efficient computation of parametric solution sensitivities is a key challenge in the integration of learning-enhanced methods with nonlinear model predictive control (MPC), as their availability is crucial for many learning algorithms. While approaches presented in the machine learning community are limited to convex or unconstrained formulations, this paper discusses the computation of solution sensitivities of general nonlinear programs (NLPs) using the implicit function theorem (IFT) and smoothed optimality conditions treated in interior-point methods (IPM). We detail sensitivity computation within a sequential quadratic programming (SQP) method which employs an IPM for the quadratic subproblems. The publication is accompanied by an efficient open-source implementation within the `acados` framework, providing both forward and adjoint sensitivities for general optimal control problems, achieving speedups exceeding 3x over the state-of-the-art solver `mpc.pytorch`.

Differentiable Model Predictive Control on the GPU

Emre Adabag, Marcus Greiff, John Subosits, Thomas Lew
Toyota Research Institute

ABSTRACT

Differentiable model predictive control (MPC) offers a powerful framework for combining learning and control. However, its adoption has been limited by the inherently sequential nature of traditional optimization algorithms, which are challenging to parallelize on modern computing hardware like GPUs. In this work, we tackle this bottleneck by introducing a GPU-accelerated differentiable optimization tool for MPC. This solver leverages sequential quadratic programming and a custom preconditioned conjugate gradient (PCG) routine with tridiagonal preconditioning to exploit the problem's structure and enable efficient parallelization. We demonstrate substantial speedups over CPU- and GPU-based baselines, significantly improving upon state-of-the-art training times on benchmark reinforcement learning and imitation learning tasks. Finally, we showcase the method on the challenging task of reinforcement learning for driving at the limits of handling, where it enables robust drifting of a Toyota Supra through water puddles.

Code: `https://github.com/ToyotaResearchInstitute/diffmpc`
Video: `https://youtu.be/r42iJBw-L4E`

https://youtu.be/r42iJBw-L4E

On the Differentiability of the Primal-Dual Interior-Point Method

Kevin Tracy and Zachary Manchester

The Robotics Institute, Carnegie Mellon University
{ktracy, zacm}@cmu.edu

Abstract. Primal-Dual Interior-Point methods are capable of solving constrained convex optimization problems to tight tolerances in a fast and robust manner. The derivatives of the primal-dual solution with respect to the problem matrices can be computed using the implicit function theorem, enabling efficient differentiation of these optimizers for a fraction of the cost of the total solution time. In the presence of active inequality constraints, this technique is only capable of providing discontinuous subgradients that present a challenge to algorithms that rely on the smoothness of these derivatives. This paper presents a technique for relaxing primal-dual solutions with a logarithmic barrier to provide smooth derivatives near active inequality constraints, with the ability to specify a uniform and consistent amount of smoothing. We pair this with an efficient primal-dual interior-point algorithm for solving an always-feasible ℓ_1-penalized variant of a convex quadratic program, eliminating the issues surrounding learning potentially infeasible problems. This parallelizable and smoothly differentiable solver is demonstrated on a range of robotics tasks where smoothing is important. An open source implementation in JAX is available at `www.github.com/kevin-tracy/qpax`.

Keywords: Differentiable Optimization · Interior-Point Methods · Convex Optimization.

Sensitivity analysis

What is old is new

R-1036-PR
April 1974

Implicit Function Theorems
for Optimization Problems and
for Systems of Inequalities

James H. Bigelow and Norman Z. Shapiro

A Report prepared for

UNITED STATES AIR FORCE PROJECT RAND

Implicit function formulas for differentiating the solutions of mathematical programming problems satisfying the conditions of the Kuhn-Tucker theorem are motivated and rigorously demonstrated. The special case of a convex objective function with linear constraints is also treated with emphasis on computational details. An example, an application to chemical equilibrium problems, is given. Implicit function formulas for differentiating the unique solution of a system of simultaneous inequalities are also derived.

Sensitivity analysis

What is old is new

Mathematical Programming 10 (1976) 287–311.
North-Holland Publishing Company

SENSITIVITY ANALYSIS FOR NONLINEAR PROGRAMMING USING PENALTY METHODS*

Anthony V. FIACCO
The George Washington University, Washington, U.S.A.

Received 26 July 1974
Revised manuscript received 18 August 1975

In this paper we establish a theoretical basis for utilizing a penalty-function method to estimate sensitivity information (i.e., the partial derivatives) of a local *solution* and its associated Lagrange multipliers of a large class of nonlinear programming problems with respect to a general parametric variation in the problem functions. The local solution is assumed to satisfy the second order sufficient conditions for a strict minimum. Although theoretically valid for higher order derivatives, the analysis concentrates on the estimation of the first order (first partial derivative) sensitivity information, which can be explicitly expressed in terms of the problem functions. For greater clarity, the results are given in terms of the mixed logarithmic-barrier quadratic-loss function. However, the approach is clearly applicable to *any algorithm* that generates a once differentiable "solution trajectory".

Annals of Operations Research 27 (1990) 215–236

SENSITIVITY AND STABILITY ANALYSIS FOR NONLINEAR PROGRAMMING *

Anthony V. FIACCO
The George Washington University, School of Engineering and Applied Science, Washington, DC 20052, USA

Yo ISHIZUKA
Sophia University, Tokyo 102, Japan

Abstract

We give a brief overview of important results in several areas of sensitivity and stability analysis for nonlinear programming, focusing initially on "qualitative" characterizations (e.g., continuity, differentiability and convexity) of the optimal value function. Subsequent results concern "quantitative" measures, in particular optimal value and solution point parameter derivative calculations, algorithmic approximations, and bounds. Our treatment is far from exhaustive and concentrates on results that hold for smooth well-structured problems.

Mathematical Programming 70 (1995) 159–172

Directional derivatives of the solution of a parametric nonlinear program

D. Ralph [a,*,1], S. Dempe [b]

[a] *Department of Mathematics, University of Melbourne, Parkville, Vic. 3052, Australia*
[b] *Department of Mathematics, Technical University of Chemnitz, Germany*

Received 26 April 1994

Abstract

Consider a parametric nonlinear optimization problem subject to equality and inequality constraints. Conditions under which a locally optimal solution exists and depends in a continuous way on the parameter are well known. We show, under the additional assumption of constant rank of the active constraint gradients, that the optimal solution is actually piecewise smooth, hence B-differentiable. We show, for the first time to our knowledge, a practical application of quadratic programming to calculate the directional derivative in the case when the optimal multipliers are not unique.

Keywords: Parametric nonlinear programming; Directional differentiability; B-derivative; Piecewise smooth function; Nonunique multipliers; Degeneracy

Takeaways

- Applying ideas from reinforcement learning to optimal control sometimes necessitates differentiating through an optimization problem
- Implicit function theorem (IFT) provides the machinery to compute the sensitivity analysis between problem parameters θ and solution x^*
- A nascent area of research is to develop differentiable solvers that satisfy the assumptions of IFT to differentiate through neural networks

References

- F. Pacaud, “Sensitivity Analysis for Parametric Nonlinear Programming: A Tutorial”, *arXiv:2504.1585*, 2025.