NOSNOC: A Software Package for Numerical Optimal Control of Nonsmooth Systems

Armin Nurkanović and Moritz Diehl

***Abstract*—This letter introduces the NOnSmooth Numerical Optimal Control (NOSNOC) open-source software package. It is a modular MATLAB tool based on CasADi and IPOPT for numerically solving Optimal Control Problems (OCP) with piecewise smooth systems (PSS). The tool supports: 1) automatic reformulation of systems with state jumps into PSS (via the time-freezing reformulation [1]) and of PSS into computationally more convenient forms, 2) automatic discretization of the OCP via, e.g., the recently introduced Finite Elements with Switch Detection [2] which enables high accuracy optimal control and simulation of PSS, 3) solution methods for the resulting discrete-time OCP. The nonsmooth discrete-time OCP are solved with techniques of continuous optimization in a homotopy procedure, without the use of integer variables. This enables the treatment of a broad class of nonsmooth systems in a unified way. Two tutorial examples are given. A benchmark shows that NOSNOC provides both faster and more accurate solutions than conventional approaches, including mixed-integer formulations.**

***Index Terms*—Software, hybrid systems, optimal control, numerical algorithms.**

I. Introduction

NONSMOOTH and hybrid dynamical systems are a powerful tool to model complex physical and cyber-physical phenomena. Their theory is well established and many good numerical simulation algorithms exist [3]. However, optimal control of nonsmooth systems is yet not wide spread, mainly due to the computational difficulty and lack of software. A notable exception are mixed integer optimization approaches [4]. However, they become intractable as soon as nonconvexities appear or exact junction times need to be computed. The open-source software package `NOSNOC` is designed to reduce this gap [5]. We regard a nonsmooth OCP of the following form:

$$\min_{x(\cdot),u(\cdot)} \int_0^T f_{\mathrm{q}}(x(t),u(t))\mathrm{d}t + f_{\mathrm{T}}(x(T)) \tag{1a}$$

$$\text{s.t. } x_0 = s_0, \tag{1b}$$

$$\dot{x}(t) = f_i(x(t),u(t)), \text{ if } x \in R_i,\ i \in \mathcal{I}, t \in [0,T], \tag{1c}$$

$$0 \geq G_{\mathrm{ineq}}(x(t),u(t)),\ t \in [0,T], \tag{1d}$$

$$0 \geq G_{\mathrm{T}}(x(T)), \tag{1e}$$

where $f_{\mathrm{q}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is the stage cost and $f_{\mathrm{T}} : \mathbb{R}^{n_x} \to \mathbb{R}$ is the terminal cost, $s_0 \in \mathbb{R}^{n_x}$ is a given parameter. The path and terminal constraints are collected in the functions $G_{\mathrm{ineq}} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_{g1}}$ and $G_{\mathrm{T}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_{g2}}$, respectively. The ODE (1c) is a piecewise smooth system (PSS), where $\mathcal{I} := \{1, \ldots, n_f\}$. The regions R_i are disjoint, nonempty, connected and open. The functions $f_i(\cdot)$ are smooth on an open neighborhood of $\overline{R_i}$, which denotes the closure of R_i.

The event of x reaching some boundary ∂R_i is called a *switch*. The right-hand side (r.h.s.) of (1c) is in general discontinuous in x. Several classes of systems with state jumps can be brought into the form of (1c) via the time-freezing reformulation [1], [6], [7]. Thus, the focus on PSS enables a unified treatment of many different kinds of nonsmooth systems.

One might wonder why not just to apply standard direct methods and existing software to problem with a smoothed version of the r.h.s. of (1c)? The necessity for tailored methods and software follows from two important results from the seminal paper of Stewart and Anitescu [8]. First, in standard direct approaches for (1c), the numerical sensitivities are wrong no matter how small the integrator step-size is. This often yields artificial local minima and impairs the optimization progress [9]. Second, smoothing delivers correct sensitivities only if the step-size shrinks faster than the smoothing parameter. Consequently, even for moderate accuracy, many optimization variables are needed.

These two difficulties are overcome by the recently introduced Finite Elements with Switch Detection (FESD) method [2]. In this method, the ODE (1c) is transformed into a Dynamic Complementarity System (DCS). FESD relies on Runge-Kutta (RK) discretizations of the DCS, but the integrator step-sizes are left as degrees of freedom as first proposed by [10]. Additional constraints ensure implicit and exact switch detection and eliminate spurious degrees of freedom. The discretization yields Mathematical Programs with Complementarity Constraints (MPCC). They are highly degenerate and nonsmooth Nonlinear Programs (NLP) [11], [12], but with suitable reformulations and homotopy procedures they

Manuscript received March 21, 2022; revised May 14, 2022; accepted June 3, 2022. Date of publication June 9, 2022; date of current version June 24, 2022. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) through Research Unit FOR 2401 and Project 424107692, and in part by the European Union (EU) through ELO-X under Grant 953348. Recommended by Senior Editor M. Arcak. *(Corresponding author: Armin Nurkanović.)*

Armin Nurkanović is with the Department of Microsystems Engineering (IMTEK), University of Freiburg, 79110 Freiburg, Germany (e-mail: armin.nurkanovic@imtek.uni-freiburg.de).

Moritz Diehl is with the Department of Microsystems Engineering (IMTEK) and the Department of Mathematics, University of Freiburg, 79110 Freiburg, Germany (e-mail: moritz.diehl@imtek.uni-freiburg.de).

Digital Object Identifier 10.1109/LCSYS.2022.3181800

can be solved efficiently using techniques for smooth NLP, without any integer variables.

The MATLAB tool NOSNOC [5] aims to automate the whole tool-chain and to make nonsmooth optimal control problems solvable for non-experts. In particular, it supports:

- automatic model reformulation of the PSS (1c) into the computationally more suitable DCS.
- time-freezing reformulation for systems with state jumps, reformulations to solve time-optimal control problems both for PSS and systems with state jumps,
- automatic discretization of the OCP (1) via FESD or RK,
- several algorithms for solving the MPCC with a homotopy approach,
- rapid prototyping with different formulations and algorithms for nonsmooth OCP.

It builds on the open-source software packages: CasADi [13] which is a symbolic framework for nonlinear optimization and the NLP solver IPOPT [14]. Having these packages as a backend enables good computational performance, despite the fact that all user inputs are provided in MATLAB. All steps above can be performed in a couple of lines of code without needing a deep understanding of the numerical methods and implementation details. In NOSNOC, the user has only to specify the functions in (1) and the sets R_i via constraint functions $c(x)$, cf. Section II. The reformulation, discretization and solution of the nonsmooth OCP is completely automated.

Notation: The complementarity conditions for two vectors $a, b \in \mathbb{R}^n$ read as $0 \le a \perp b \ge 0$, where $a \perp b$ means $a^\top b = 0$. The so-called C-functions $\Phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ have the property $\Phi(a,b) = 0 \iff 0 \le a \perp b \ge 0$, e.g., $\Phi(a,b) = \min(a,b)$. The concatenation of two column vectors $a \in \mathbb{R}^{n_a}$, $b \in \mathbb{R}^{n_b}$ is denoted by $(a,b) := [a^\top, b^\top]^\top$, the concatenation of several column vectors is defined in an analogous way. A column vector with all ones is denoted by $e = (1, 1, \ldots, 1) \in \mathbb{R}^n$, its dimensions is clear from the context. The closure of a set C is denoted by $\overline{C}$, its boundary as ∂C. Given a matrix $M \in \mathbb{R}^{n \times m}$, its i-th row is denoted by $M_{i,\bullet}$ and its j-th column is denoted by $M_{\bullet,j}$.

Outline: Section II describes the reformulation of PSS into DCS. Section III describes the discretization methods in NOSNOC with a focus on FESD. In Section IV, solution strategies for the discrete-time OCP are discussed. Section V provides two tutorials for the use of NOSNOC and a numerical benchmark. Section VI outlines some future developments.

II. Problem Reformulation

System with state jumps do not fit in the form of (1c). However, we use the time-freezing reformulation [1], [6], [7] to automatically reformulate them into the from of (1c). An example is given in Section V-C.

In this section, we detail how to compactly represent the systems (1c) and a how to transform them into a Dynamic Complementarity System (DCS) via Stewart's approach [15].

It is assumed that $\overline{\bigcup_{i\in\mathcal{I}} R_i} = \mathbb{R}^n$ and that $\mathbb{R}^n \setminus \bigcup_{i\in\mathcal{I}} R_i$ is a set of measure zero. Moreover, we assume that R_i are defined via the zero level sets of the components of the smooth function $c : \mathbb{R}^{n_x} \to \mathbb{R}^{n_c}$. We use a sign matrix $S \in \mathbb{R}^{n_f \times n_c}$ with non repeating rows for a compact representations as follows:

$$S = \begin{bmatrix} 1 & 1 & \ldots & 1 & 1 \\ 1 & 1 & \ldots & 1 & -1 \\ \vdots & \vdots & \ldots & \vdots & \\ -1 & -1 & \ldots & -1 & -1 \end{bmatrix}, \tag{2a}$$

$$R_i = \{x \in \mathbb{R}^{n_x} \mid \operatorname{diag}(S_{i,\bullet})c(x) > 0\}. \tag{2b}$$

For example, for the sets $R_1 = \{x \in \mathbb{R} \mid x > 0\}$ and $R_2 = \{x \in \mathbb{R} \mid x < 0\}$, we have $c(x) = x$ and $S = \begin{bmatrix} 1 & -1 \end{bmatrix}^\top$.

The dynamics are not defined on ∂R_i and to have a meaningful notion of solution for the PSS (1c) we use the Filippov convexification and define the following differential inclusion [16]:

$$\dot{x} \in F_{\mathrm{F}}(x,u) = \Big\{ F(x)\theta \Big| \sum_{i\in\mathcal{I}} \theta_i = 1,\ \theta_i \ge 0,\ \theta_i = 0 \text{ if } x \notin \overline{R_i}, \forall i \in \mathcal{I} \Big\}, \tag{3}$$

where $\theta = (\theta_1, \ldots, \theta_{n_f}) \in \mathbb{R}^{n_f}$ and $F(x) := [f_1(x), \ldots, f_{n_f}(x)] \in \mathbb{R}^{n_x \times n_f}$. Note that in the interior of a set R_i we have $F_{\mathrm{F}}(x) = \{f_i(x)\}$ and on the boundary between some regions the resulting vector field is a convex combination of the neighboring vector fields. To have a computationally useful representation of the Filippov system (3), we transform it into a DCS via Stewart's reformulation [15]. In this reformulation, it is assumed that the sets R_i are represented via the *discriminant functions* $g_i(\cdot)$:

$$R_i = \{x \in \mathbb{R}^{n_x} \mid g_i(x) < \min_{j\in\mathcal{I}, j\neq i} g_j(x)\}. \tag{4}$$

Given the more intuitive representation via the sign matrix S in Eq. (2), it can be shown that the function $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_f}$ whose components are $g_i(x)$ can be found as [2]:

$$g(x) = -Sc(x). \tag{5}$$

With this representation, the convex multipliers in the r.h.s. of (3) can be found as a solution of a suitable Linear Program (LP) [15], and (3) is equivalent to

$$\dot{x} = F(x,u)\theta(x), \tag{6a}$$

$$\theta(x) \in \arg\min_{\tilde{\theta}\in\mathbb{R}^{n_f}} \ g(x)^\top \tilde{\theta} \quad \text{s.t.} \quad e^\top\tilde{\theta} = 1,\ \tilde{\theta} \ge 0. \tag{6b}$$

We use a C-function $\Phi(\cdot,\cdot)$ for the complementarity conditions and write the KKT conditions of the LP (6b) as a nonsmooth equation

$$G_{\mathrm{LP}}(x,\theta,\lambda,\mu) := \begin{bmatrix} g(x) - \lambda - \mu e \\ 1 - e^\top\theta \\ \Phi(\theta,\lambda) \end{bmatrix} = 0, \tag{7}$$

where $\lambda \in \mathbb{R}^{n_f}_{\ge 0}$ and $\mu \in \mathbb{R}$ are the Lagrange multipliers associated with the constraints of the LP (6b). Note that $\mu = \min_{j\in\mathcal{I}} g_j(x)$. Finally, the Filippov system is equivalent to the following DCS, which can be interpreted as a nonsmooth differential algebraic equation:

$$\dot{x} = F(x,u)\theta, \quad 0 = G_{\mathrm{LP}}(x,\theta,\lambda,\mu). \tag{8}$$

A fundamental property of the multipliers $\lambda(\cdot)$ and $\mu(\cdot)$ is their continuity in time [2, Lemma 5], whereas $\theta(\cdot)$ is in general a discontinuous function in time.

III. The Standard and FESD Discretizations for a Single Control Interval

This section describes the discretization of a single control interval in NOSNOC via standard RK methods and FESD. We start with a standard RK method for the DCS (8). We subsequently introduce step-by-step the additional constraints which lead to FESD.

A. Standard Runge-Kutta Discretization

We consider a single control interval $[0, T]$ with a given constant control input q and a given initial value $x_0 = s_0$. We divide the control interval into N_{FE} finite elements (i.e., integration intervals) $[t_n, t_{n+1}]$ via the grid points $0 = t_0 < t_1 < \ldots < t_{N_{\mathrm{FE}}} = T$. On each of these intervals, we apply an n_{s}-stage RK scheme, which is defined by its Butcher tableau entries $a_{i,j},\ b_i,\ c_i,\ i,j \in \{1,\ldots,n_{\mathrm{s}}\}$ [17]. We denote the step-size as $h_n = t_{n+1} - t_n,\ n = 0,\ldots,N_{\mathrm{FE}} - 1$. The approximation of the state at the grid points t_n is denoted by $x_n \approx x(t_n)$. The time derivative of the state at the stage points $t_n + c_i h_n,\ i = 1,\ldots,n_{\mathrm{s}}$, for a single finite element are collected in the vector $V_n := (v_{n,1},\ldots,v_{n,n_{\mathrm{s}}}) \in \mathbb{R}^{n_{\mathrm{s}}\cdot n_x}$. The stage values for the algebraic variable $\theta(\cdot)$ are collected in $\Theta_n := (\theta_{n,1},\ldots,\theta_{n,n_{\mathrm{s}}}) \in \mathbb{R}^{n_{\mathrm{s}}\cdot n_f}$. The vectors $\Lambda_n \in \mathbb{R}^{n_{\mathrm{s}}\cdot n_f}$ and $M_n \in \mathbb{R}^{n_{\mathrm{s}}}$ are defined accordingly. Let x_n^{next} denote the value at the next time step t_{n+1}, which is obtained after a single RK step.

Now we can write the RK equations for the DCS (8) in a compact *differential* form. We summarize all RK equations of a finite element in $G_{\mathrm{rk}}(x_n^{\mathrm{next}}, Z_n, h_n, q) = 0$, where $Z_n = (x_n, \Theta_n, \Lambda_n, M_n, V_n)$ collects all internal variables, and define

$$
\begin{aligned}
&G_{\mathrm{rk}}(x_n^{\mathrm{next}}, Z_n, h_n, q)\\
&:= \begin{bmatrix}
v_{n,1} - F(x_n + h_n \sum_{j=1}^{n_{\mathrm{s}}} a_{1,j} v_{n,j}, q)\theta_{n,1}\\
\vdots\\
v_{n,n_{\mathrm{s}}} - F(x_n + h_n \sum_{j=1}^{n_{\mathrm{s}}} a_{n_{\mathrm{s}},j} v_{n,j}, q)\theta_{n,n_{\mathrm{s}}}\\
G_{\mathrm{LP}}(x_n + h_n \sum_{j=1}^{n_{\mathrm{s}}} a_{1,j} v_{n,j}, \theta_{n,1}, \lambda_{n,1}, \mu_{n,1})\\
\vdots\\
G_{\mathrm{LP}}(x_n + h_n \sum_{j=1}^{n_{\mathrm{s}}} a_{n_{\mathrm{s}},j} v_{n,j}, \theta_{n,n_{\mathrm{s}}}, \lambda_{n,n_{\mathrm{s}}}, \mu_{n,n_{\mathrm{s}}})\\
x_n^{\mathrm{next}} - x_n - h_n \sum_{i=1}^{n_{\mathrm{s}}} b_i v_{n,i}
\end{bmatrix}.
\end{aligned}
$$

To summarize all conditions for a single control interval in a compact way, we introduce some new notation. The variables for all finite elements of a single control interval are collected in the following vectors $\mathbf{x} = (x_0, x_0^{\mathrm{next}}, \ldots, x_{N_{\mathrm{FE}}}) \in \mathbb{R}^{(2N_{\mathrm{FE}}+1)n_x}$, $\mathbf{V} = (V_0,\ldots,V_{N_{\mathrm{FE}}-1}) \in \mathbb{R}^{N_{\mathrm{FE}} n_{\mathrm{s}} n_x}$ and $\mathbf{h} := (h_0,\ldots,h_{N_{\mathrm{FE}}-1}) \in \mathbb{R}^{N_{\mathrm{FE}}}$. The vectors $\Theta \in \mathbb{R}^{N_{\mathrm{FE}} n_{\mathrm{s}} n_f}$, $\Lambda \in \mathbb{R}^{N_{\mathrm{FE}} n_{\mathrm{s}} n_f}$ and $\mathbf{M} \in \mathbb{R}^{N_{\mathrm{FE}} n_{\mathrm{s}}}$ are defined analogously. The vector $\mathbf{Z} = (\mathbf{x}, \Theta, \Lambda, \mathbf{M}, \mathbf{V})$ collects all *internal* variables.

Finally, we can summarize all computations over a single control interval and interpret it as a discrete-time nonsmooth system:

$$
s_1 = F_{\mathrm{std}}(\mathbf{Z}),\ 0 = G_{\mathrm{std}}(\mathbf{Z}, \mathbf{h}, s_0, q) \tag{9}
$$

with $F_{\mathrm{std}}(\mathbf{Z}) = x_{N_{\mathrm{FE}}}$ and

$$
G_{\mathrm{std}}(\mathbf{Z}, \mathbf{h}, s_0, q) := \begin{bmatrix}
x_0 - s_0\\
G_{\mathrm{rk}}(x_0^{\mathrm{next}}, Z_0, h_0, q)\\
x_1 - x_0^{\mathrm{next}}\\
\vdots\\
G_{\mathrm{rk}}(x_{N_{\mathrm{FE}}-1}^{\mathrm{next}}, Z_{N_{\mathrm{FE}}-1}, h_{N_{\mathrm{FE}}-1}, q)\\
x_{N_{\mathrm{FE}}} - x_{N_{\mathrm{FE}}-1}^{\mathrm{next}}
\end{bmatrix}.
$$

Note that we keep a dependency on h_n in (9), but h_n is implicitly given by the chosen discretization grid. This also means that for a standard RK scheme for DCS, higher order accuracy can be achieved only if the grid points t_n coincide with all switching points, which is in practice impossible to achieve.

B. Cross-Complementarity

In FESD, the step-sizes h_n are left as degrees of freedom such that the grid points t_n can coincide with the switching times. Consequently, the switches should not happen on the stages inside a finite element. To exploit the additional degrees of freedom and to achieve these two effects we introduce additional conditions to the RK equations (9) called *cross complementaries*. A key assumption, of course, is that there are more grid points in the interior of the grid than switching points.

For ease of exposition, we focus on the case where the right-boundary point of a finite element is also an RK-stage point, i.e., $c_{n_{\mathrm{s}}} = 1$ and $t_{n+1} = t_n + c_{n_{\mathrm{s}}} h_n$. Extensions can be found in [2]. To achieve implicit and exact switch detection at the boundaries of $[t_n, t_{n+1}]$ and to avoid switching inside an element we exploit the fact that $\lambda(\cdot)$ and $\mu(\cdot)$ are continuous functions. We need their values at t_n and t_{n+1} which are denoted by $\lambda_{n,0},\ \mu_{n,0}$ and $\lambda_{n,n_{\mathrm{s}}},\ \mu_{n,n_{\mathrm{s}}}$, respectively. Due to continuity, we impose that $\lambda_{n,n_{\mathrm{s}}} = \lambda_{n+1,0}$ and $\mu_{n,n_{\mathrm{s}}} = \mu_{n+1,0}$ and use only the right boundary points of the finite elements ($\lambda_{n,n_{\mathrm{s}}}$ and $\mu_{n,n_{\mathrm{s}}}$) in the sequel.

To achieve the effects described above, we introduce the cross complementarity conditions which read as [2]:

$$
0 = G_{\mathrm{cross}}(\Theta, \Lambda) := \begin{bmatrix}
\sum_{i=1}^{n_{\mathrm{s}}} \sum_{j=1, j\neq i}^{n_{\mathrm{s}}} \theta_{1,i}^{\top} \lambda_{1,j}\\
\vdots\\
\sum_{i=1}^{n_{\mathrm{s}}} \sum_{\substack{j=0,\\ j\neq i}}^{n_{\mathrm{s}}} \theta_{N_{\mathrm{FE}}-1,i}^{\top} \lambda_{N_{\mathrm{FE}}-1,j}
\end{bmatrix}. \tag{10}
$$

This additional constraint ensures two very important properties: (i) we have the same active-set in (9) in $\Phi(\theta_{n,m}, \lambda_{n,m})$ for all m and changes can happen only for different n, i.e., at grid points t_n, (ii) whenever the active-sets for two neighboring finite elements differ in the i-th and j-th components of $\Phi(\theta_{n,m}, \lambda_{n,m})$, then these two components of $\lambda_{n,n_{\mathrm{s}}}$ must be zero [2]. This will implicitly result in the constraint $0 = g_i(x_{n+1}) - g_j(x_{n+1})$ (which comes from (7) and the fact that $\mu_{n,n_{\mathrm{s}}} = \min_j g_j(x_{n+1})$). This defines the boundary between regions and R_i and R_j, cf. (4). Thus, it implicitly forces h_n to adapt for exact switch detection.

C. Step-Equilibration

If no switches occur then also no active-set changes happen, hence the constraints (10) are trivially satisfied. Consequently,

the step-size h_n can vary in a possibly undesired way and the optimizer can play with the discretization accuracy. To remove the spurious degrees of freedom we introduce an indicator function $\eta(\cdot)$ evaluated at the *inner* grid points t_n, $n = 1, \ldots, N_{\mathrm{FE}} - 1$ and its value at t_n is denoted by η_n. It has the following property: if a switch happens at t_n its value is zero, otherwise it is strictly positive. We omit the details on how a function $\eta(\cdot)$ is derived and refer to [2]. The discrete-time function $\eta(\cdot)$ depends on the values of Θ_n and Λ_n of neighboring finite elements and we define

$$\eta_n(\Theta, \Lambda) \coloneqq \eta(\Theta_{n-1}, \Lambda_{n-1}, \Theta_n, \Lambda_n).$$

Thus, the constraint $0 = G_{\mathrm{eq}}(\mathbf{h}, \Theta, \Lambda)$ removes the possible spurious degrees of freedom in h_n, where:

$$G_{\mathrm{eq}}(\mathbf{h}, \Theta, \Lambda) \coloneqq \begin{bmatrix} (h_1 - h_0)\eta_1(\Theta, \Lambda) \\ \vdots \\ (h_{N_{\mathrm{FE}}-1} - h_{N_{\mathrm{FE}}-2})\eta_{N_{\mathrm{FE}}-1}(\Theta, \Lambda) \end{bmatrix}. \quad (11)$$

We call the condition (11) step-equilibration. A consequence of (11) are locally equidistant state discretization grids between switching point, within a single control interval. Since this constraint can be quite nonlinear, NOSNOC offers several reformulations and heuristics that help numerical convergence.

D. Finite Elements With Switch Detection

We now use the ingredients explained above to state the FESD method. Similar to the standard RK scheme (9), we summarize all computations over a single control interval and interpret it as a discrete-time nonsmooth system where internally exact switch detection is happening. The next step is computed by

$$s_1 = F_{\mathrm{fesd}}(\mathbf{Z}), \;\; 0 = G_{\mathrm{fesd}}(\mathbf{Z}, \mathbf{h}, s_0, q, T), \quad (12)$$

and $F_{\mathrm{fesd}}(\mathbf{Z}) = x_{N_{\mathrm{FE}}}$ renders the state transition map and the equation $0 = G_{\mathrm{fesd}}(\mathbf{x}, \mathbf{Z}, q)$ collects all other internal computations including all RK steps within the regarded control interval:

$$G_{\mathrm{fesd}}(\mathbf{Z}, \mathbf{h}, s_0, q, T) \coloneqq \begin{bmatrix} G_{\mathrm{std}}(\mathbf{Z}, \mathbf{h}, s_0, q) \\ G_{\mathrm{cross}}(\Theta, \Lambda) \\ G_{\mathrm{eq}}(\mathbf{h}, \Theta, \Lambda) \\ \sum_{n=0}^{N_{\mathrm{FE}}-1} h_n - T \end{bmatrix}.$$

The last condition ensures that the length of the considered time-interval is unaltered. In contrast to (9), h_n are now degrees of freedom, s_0, q and T are given parameters. The formulation (12) can be used as an integrator with exact switch detection for PSS (1c). This feature is implemented in NOSNOC via the function `integrator_fesd()`. It can automatically handle all kinds of switching cases such as: crossing a discontinuity, sliding mode, leaving a sliding mode or spontaneous switches [16].

IV. Discretizing and Solving a Nonsmooth Optimal Control Problem

This section outlines how a nonsmooth OCP is discretized in NOSNOC and how the resulting MPCC is solved.

A. Multiple Shooting-Type Discretization With FESD

One of the main goals of NOSNOC is to numerically solve a discretized version of the OCP (1). We consider $N_{\mathrm{stg}} \geq 1$ control intervals of equal length, indexed by k, with piecewise constant controls collected in $\mathbf{q} = (q_0, \ldots, q_{N_{\mathrm{stg}}-1}) \in \mathbb{R}^{N_{\mathrm{stg}} n_u}$. All internal variables are additionally equipped with an index k. On every control interval k, we apply an FESD discretization (12) with N_{FE} internal finite elements. The state values at the control interval boundaries are collected in $\mathbf{s} = (s_0, \ldots, s_{N_{\mathrm{stg}}}) \in \mathbb{R}^{(N_{\mathrm{stg}}+1)n_x}$. The vector $\mathcal{Z} = (\mathbf{Z}_0, \ldots, \mathbf{Z}_{N_{\mathrm{stg}}-1})$ collects all internal variables and $\mathcal{H} = (\mathbf{h}_0, \ldots, \mathbf{h}_{N_{\mathrm{stg}}-1})$ all step-sizes. Finally the discretized OCP reads as:

$$\begin{aligned} \min_{\mathbf{s},\mathbf{q},\mathcal{Z},\mathcal{H}} \;\; & \sum_{k=0}^{N_{\mathrm{stg}}-1} \hat{f}_q(s_k, \mathbf{x}_k, q_k) + \hat{f}_{\mathrm{T}}(s_{N_{\mathrm{stg}}}) && \text{(13a)} \\ \text{s.t.} \;\; & s_0 = \bar{x}_0, && \text{(13b)} \\ & s_{k+1} = F_{\mathrm{fesd}}(\mathbf{x}_k), \;\; k = 0, \ldots, N_{\mathrm{stg}} - 1, && \text{(13c)} \\ & 0 = G_{\mathrm{fesd}}(\mathbf{x}_k, \mathbf{Z}_k, q_k), \;\; k = 0, \ldots, N_{\mathrm{stg}} - 1, && \text{(13d)} \\ & 0 \geq G_{\mathrm{ineq}}(s_k, q_k), \;\; k = 0, \ldots, N_{\mathrm{stg}} - 1, && \text{(13e)} \\ & 0 \geq G_{\mathrm{T}}(s_{N_{\mathrm{stg}}}), && \text{(13f)} \end{aligned}$$

where $\hat{f}_q : \mathbb{R}^{n_x} \times \mathbb{R}^{(N_{\mathrm{FE}}+1)n_s n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ and $\hat{f}_{\mathrm{T}} : \mathbb{R}^{n_x} \to \mathbb{R}$ are the discrteized stage and terminal costs, respectively.

B. Reformulating and Solving MPCC

The discrete-time OCP (13) is an MPCC. It can be written more compactly as

$$\begin{aligned} \min_{w} \;\; & f(w) && \text{(14a)} \\ \text{s.t.} \;\; & 0 \leq h(w), && \text{(14b)} \\ & 0 \leq w_1 \perp w_2 \geq 0, && \text{(14c)} \end{aligned}$$

where $w = (w_0, w_1, w_2) \in \mathbb{R}^{n_w}$ is a given decomposition of the problem variables. MPCC are difficult nonsmooth NLP which violate, e.g., the MFCQ at all feasible points [12]. Fortunately, they can often be solved efficiently via reformulations and homotopy approaches [11], [12]. We briefly discuss the different ways of solving MPCC that are implemented in NOSNOC. They differ in how Eq. (14c) is handled. In all cases, $w_1, w_2 \geq 0$ is kept unaltered and the bilinear constraint $w_1^\top w_2 = 0$ is treated differently.

In a homotopy procedure, we solve a sequence of more regular, relaxed NLP related to (14) and parameterized by a homotopy parameter $\sigma_i \in \mathbb{R}_{\geq 0}$. Every new NLP is initialized with the solution of the previous one. In all approaches the homotopy parameter is updated via the rule: $\sigma_{i+1} = \kappa\sigma_i$, $\kappa \in (0, 1)$, $\sigma_0 > 0$, where i is the index of the NLP in the homotopy. In the limit as $\sigma_i \to 0$ (or often even for a finite i and σ_i) the solution of the relaxed NLP matches a solution of (14). NOSNOC supports the following approaches:

Smoothing and Relaxation: In smoothing the bilinear term is replaced by the simpler constraint $w_1^\top w_2 = \sigma_i$ and in *relaxation* by $w_1^\top w_2 \leq \sigma_i$. Under certain assumptions for $\sigma_i \to 0$ a solution of the initial MPCC (14) is obtained [11].

ℓ_1-Penalty: In this approach, the bilinear constraint is discarded and the term $\frac{1}{\sigma_i} w_1^\top w_2$ is added to the objective, which

is a penalized ℓ_1 norm of the complementarity residual. When the penalty $\frac{1}{\sigma_i}$ exceeds a certain (often finite) threshold we have $w_1^\top w_2 = 0$ and the solution of such an NLP is a solution to (14) [12].

Elastic Mode: In *elastic mode* (sometimes called ℓ_∞-approach) [12], a bounded scalar slack variable $\gamma \in [0, \bar{\gamma}]$ is introduced. The relaxed bilinear constraint reads as $w_1^\top w_2 \leq \gamma$ and we add to the objective $\frac{1}{\sigma_i}\gamma$. Variants with $w_1^\top w_2 = \gamma$ and $-\gamma \leq w_1^\top w_2 \leq \gamma$ are supported as well. Once the penalty $\frac{1}{\sigma_i}$ exceeds a certain (often finite) threshold, we have $\gamma = 0$ and we recover a solution of (14) [12].

V. NOSNOC Tutorials and a Benchmark

In this section, we provide two short tutorials on the use of NOSNOC. A numerical benchmark where we compare our software to conventional approaches is presented as well.

A. Solving a Time-Optimal Control Problem

We regard a time-optimal control problem of a double-integrator car model with a normal and turbo mode. The state vector $x = (q, v) \in \mathbb{R}^2$ consists of the car's position q and velocity v. The PSS reads as

$$\dot{x} = \begin{cases} (v, u), & \text{if } v < \bar{v} \\ (v, 3u), & \text{if } v > \bar{v} \end{cases}. \tag{15}$$

Following Section II, we have $f_1(x, u) = (q, u)$ (nominal), $f_2(x, u) = (q, 3u)$ (turbo). The two regions R_1 and R_2 described by $c(x) = v - \bar{v}$ and $S = \begin{bmatrix} -1 & 1 \end{bmatrix}^\top$. The car should reach the state $x_{\text{goal}} = (200, 0)$ in optimal time T. Additionally, we have constraints on the velocity $|v| \leq v_{\max}$ and control $|u| \leq u_{\max}$. The parameters are $v_{\max} = 25$, $u_{\max} = 5$ and $\bar{v} = 10$. This OCP is formulated and solved with NOSNOC using the code:

```
1  [settings] = default_settings_nosnoc();
2  settings.time_optimal_problem = 1;
3  settings.n_s = 2;
4  model.N_stg = 10; model.N_FE = 3; model.T = 1;
5  q = MX.sym('q'); v = MX.sym('v');
6  model.x = [q;v]; model.x0 = [0;0];
7  model.lbx = [-inf;-25]; model.ubx = [inf;25];
8  u = SX.sym('u'); model.u = u;
9  model.lbu = -5; model.ubu = 5;
10 f_1 = [v;u]; f_2 = [v;3*u];
11 model.F = [f_1 f_2];
12 model.c = v-10; model.S = [-1;1];
13 model.g_terminal = [q-200;v-0];
14 [results,stats,model,settings] = nosnoc_solver(
       model,settings);
```

The function `default_settings_nosnoc()` returns a MATLAB `struct` with default values for all possible tuning parameters. The needed time-transformations are automated by the flag `settings.time_optimal_problem = 1`. For the FESD-RK method we keep the default choice of a Radau II-A, hence we have with $n_s = 2$ an accuracy order of 3 [17]. The MATLAB `struct` named `model` stores user input data, given in lines 4 to 13, which defines the OCP (1). NOSNOC automates all definitions, reformulations and updates the `model` with all CasADi expressions for the DCS (8). Moreover, possible inconsistencies in the provided `settings` are refined. Finally, in line 14 we solve the discretized OCP with a homotopy as described in Section IV-B. The solution trajectory is given in Fig. 1. The user has access to all tuning parameters, intermediate results for all homotopy iterations and to all CasADi symbolic expressions and `Function` objects. This facilitates rapid prototyping and detailed analysis of solutions.

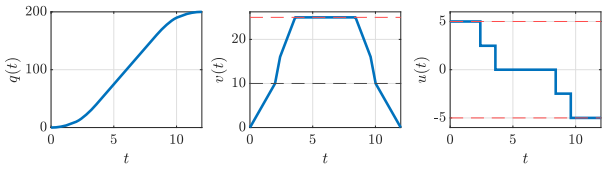

Fig. 1. The position of the car $q(t)$ is shown in the left plot, the velocity $v(t)$ in the middle plot. Note the increase in acceleration in the turbo mode for $v > \bar{v}$. The right plots shows the optimal control $u(t)$.

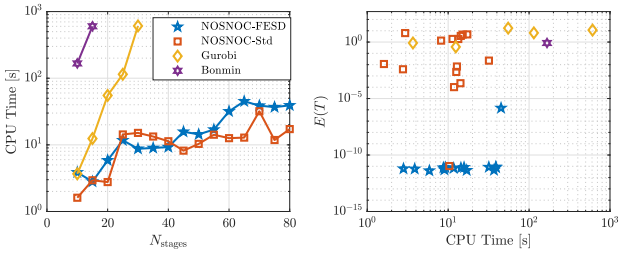

Fig. 2. Comparison of NOSNOC to mixed integer formulations. The left plot show CPU time as function of number of control intervals N_{stg}. The right plot show the solution accuracy as function of CPU time in a Pareto plot.

B. Numerical Benchmark

We solve the OCP from the last section with four different approaches. We use NOSNOC with the FESD discretization (12) and NOSNOC with the standard discretization (9). The latter approach is closely related to the smoothing approach in [8] and [9]. For the MPCC, we use in both cases the relaxation approach as it is usually the most robust one. Additionally, we make a *big M* reformulation of the PSS (15) and solve a mixed integer nonlinear program (MINLP). Switches are allowed only at the control interval boundaries, hence we have two binary variables per control interval. We solve the MINLP with the dedicated solver Bonmin [18]. Moreover, since the only non-linearity is in time T, we fix it and make a bisection-type search in T. For every fixed T we solve a MILP with Gurobi. The MILP with the smallest T that is still feasible delivers the optimal solution. We vary N_{stg} from 10 to 80 with steps of 5. The computations are aborted if the time-limit of 10 minutes is exceeded.

The results of the benchmark are depicted in Fig. 2. NOSNOC-FESD is slightly slower than NOSNOC-Std, since it has $N_{\text{stg}}N_{\text{FE}}$ more variables, as h_n are degrees of freedom. We compare also the solution quality by making a high-accuracy simulation $x_{\text{sim}}(t)$ of (15) with the obtained optimal controls $\mathbf{q}$. We compare the terminal constraint satisfaction $E(T) = \|x_{\text{sim}}(T) - x_{\text{goal}}\|$. Due to the exact switch detection property, NOSNOC-FESD has by far the most accurate solutions. The outlier where NOSNOC-Std achieves high accuracy corresponds to a local minima without switches. We see that even a simple nonsmooth OCP is difficult to solve with

conventional approaches, whereas NOSNOC-FESD provides faster and several orders of magnitude more accurate solutions. Further detailed comparisons of FESD to the standard approach can be found in [2].

C. An Example With State Jumps and Time-Freezing

In this subsection we illustrate how to use NOSNOC with systems with state jumps. We consider a planar bouncing ball with elastic impacts. The state vector is defined as $x = (q, v) \in \mathbb{R}^4$ with $q = (q_1, q_2) \in \mathbb{R}^2$ and $v = (v_1, v_2) \in \mathbb{R}^2$ being the ball's position and velocity, respectively. The initial state is $x(0) = (0, 0.5, 0, 0)$ and the ball is controlled with some force $u \in \mathbb{R}^2$. The ODE with state jumps reads as:

$$\dot{q} = v, \ \dot{v} = u - (0, g), \tag{16a}$$

$$v_2(t^+) = -ev_2(t^-), \text{ if } q_2(t) = 0 \text{ and } v_2(t) < 0. \tag{16b}$$

where $e \in (0, 1]$ is the coefficient of restitution and determines the post impact velocity. The goal is to reach $q_f = (4, 0.5)$ with a minimal quadratic control effort modeled with the stage cost $f_q(x, u) = u^\top u$ and a minimal terminal velocity expressed via $f_T(x) = 100v^\top v$, with $T = 4$. The control force is bounded such that it is weaker than the gravitational force, i.e., $u^\top u \leq u_{\max}^2$. The chosen parameters are $e = 0.9$, $g = 9.81$, $u_{\max} = 9$. The following NOSNOC code solves the described nonsmooth OCP with state jump:

```
[settings] = default_settings_fesd();
settings.time_freezing = 1; settings.n_s = 3;
model.T = 4; model.N_stg = 20; model.N_FE = 3;
q = MX.sym('q',2); v = MX.sym('v',2);
model.x = [q;v]; model.x0 = [0;0.5;0;0];
u = MX.sym('u',2);  model.u = u;
model.c = q(2);  model.e = 0.9;
model.f = [v;u-[0;9.81]];
model.f_q = u'*u; model.f_q_T = 100*v'*v;
model.g_ineq = u'*u-9^2;
model.g_terminal = q-[4;0.5];
[results,stats,model,settings] = nosnoc_solver(
    model,settings);
```

The flag `settings.time_freezing = 1` ensures that system with state jumps (16) is transformed into a PSS of the form of (1c) via the time-freezing reformulation [1]. A solution trajectory is given in Figure 3, note the state jumps in $v_2(t)$ in the middle plot.

Many more settings can be changed by the user, for example, one can choose between different MPCC reformulations via `mpcc_mode`, control the sparsity of the cross complementarities `cross_complementarity_mode` and so on. A few more examples and a detailed user manual are available NOSNOC's repository [5].

VI. Conclusion and Outlook

In this letter we presented NOSNOC, an open-source software package for nonsmooth numerical optimal control. With the help of the Finite Elements with Switch Detection (FESD) method and the time-freezing reformulation, it enables practical and high accuracy optimal control of several different classes of nonsmooth system in a unified way. The discretized OCP are solved with techniques solely from continuous optimization, without the need for any integer variables. All reformulations and details are hidden but accessible such that a convenient use for users with different knowledge levels of the field is ensured.

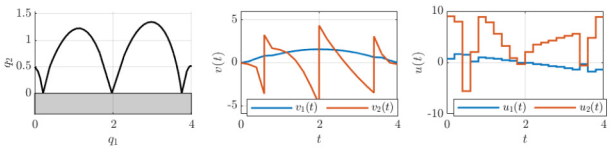

Fig. 3. The illustration of the optimal solution $q(t)$ is shown in the left plot. The middle plot shows the optimal velocities $v(t)$ as a function of the *physical time* t, where the state jumps are recovered. The right plot shows the optimal controls $u(t)$.

In future work, we aim to implement a python version of NOSNOC. Moreover, further algorithmic developments in FESD, e.g., different reformulations of PSS into DCS, support for time-freezing for other classes of hybrid systems will be implemented.

References

[1] A. Nurkanović, T. Sartor, S. Albrecht, and M. Diehl, "A time-freezing approach for numerical optimal control of nonsmooth differential equations with state jumps," *IEEE Control Syst. Lett.*, vol. 5, no. 2, pp. 439–444, Apr. 2021.

[2] A. Nurkanović, M. Sperl, S. Albrecht, and M. Diehl, "Finite elements with switch detection for direct optimal control of nonsmooth systems," 2022, *arXiv:2205.05337*.

[3] B. Brogliato and A. Tanwani, "Dynamical systems coupled with monotone set-valued operators: Formalisms, applications, well-posedness, and stability," *SIAM Rev.*, vol. 62, no. 1, pp. 3–129, 2020.

[4] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[5] "NOSNOC." 2022. [Online]. Available: https://github.com/nurkanovic/nosnoc

[6] A. Nurkanović, S. Albrecht, B. Brogliato, and M. Diehl, "The time-Freezing reformulation for numerical optimal control of complementarity lagrangian systems with state jumps," 2021, *arXiv:2111.06759*.

[7] A. Nurkanović and M. Diehl, "Continuous optimization for control of hybrid systems with hysteresis via time-freezing," 2022, *arXiv:2203.11510*.

[8] D. E. Stewart and M. Anitescu, "Optimal control of systems with discontinuous differential equations," *Numerische Mathematik*, vol. 114, no. 4, pp. 653–695, 2010.

[9] A. Nurkanović, S. Albrecht, and M. Diehl, "Limits of MPCC formulations in direct optimal control with nonsmooth differential equations," in *Proc. Eur. Control Conf. (ECC)*, 2020, pp. 2015–2020.

[10] B. T. Baumrucker and L. T. Biegler, "MPEC strategies for optimization of a class of hybrid dynamic systems," *J. Process Control*, vol. 19, no. 8, pp. 1248–1256, 2009.

[11] S. Scholtes, "Convergence properties of a regularization scheme for mathematical programs with complementarity constraints," *SIAM J. Optim.*, vol. 11, no. 4, pp. 918–936, 2001.

[12] M. Anitescu, P. Tseng, and S. J. Wright, "Elastic-mode algorithms for mathematical programs with equilibrium constraints: Global convergence and stationarity properties," *Math. Program.*, vol. 110, no. 2, pp. 337–371, 2007.

[13] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi—A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.

[14] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, 2006.

[15] D. E. Stewart, "A high accuracy method for solving ODEs with discontinuous right-hand side," *Numerische Mathematik*, vol. 58, no. 1, pp. 299–328, 1990.

[16] A. F. Filippov, *Differential Equations with Discontinuous Right-Hand Side*. Dordrecht, The Netherlands: Kluwer, 1988.

[17] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II—Stiff and Differential-Algebraic Problems*, 2nd ed. Berlin, Germany: Springer, 1991.

[18] P. Bonami *et al.*, "An algorithmic framework for convex mixed integer nonlinear programs," *Discr. Optim.*, vol. 5, no. 2, pp. 186–204, 2008.