European Journal of Control xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Control

journal homepage: www.sciencedirect.com/journal/european-journal-of-control

Gauss–Newton Runge–Kutta integration for efficient discretization of optimal control problems with long horizons and least-squares costs ☆

Jonathan Frey [a,b,*], Katrin Baumgärtner [b], Moritz Diehl [a,b]

[a] Department of Microsystems Engineering (IMTEK), University Freiburg, Germany
[b] Department of Mathematics, University Freiburg, Germany

ARTICLE INFO

Recommended by T. Parisini

Keywords:
Optimal control
Optimization algorithms
Optimization

ABSTRACT

This work proposes an efficient treatment of continuous-time optimal control problems with long horizons and nonlinear least-squares costs. In particular, we present the Gauss–Newton Runge–Kutta (GNRK) integrator which provides a high-order cost integration. Crucially, the Hessian of the cost terms required within an SQP-type algorithm is approximated with a Gauss–Newton Hessian. Moreover, L_2 penalty formulations for constraints are shown to be particularly effective for optimization with GNRK. An efficient implementation of GNRK is provided in the open-source software framework `acados`. We demonstrate the effectiveness of the proposed approach and its implementation on an illustrative example showing a reduction of relative suboptimality by a factor greater than 10 while increasing the runtime by only 10%.

1. Introduction

Model predictive control (MPC) is an optimization-based control strategy which relies on the (approximate) solution of nonlinear optimization problems in real-time. In direct optimal control, starting from a continuous-time optimal control problem (OCP), a variety of choices have to be made to derive a discrete-time formulation that adequately approximates the continuous-time problem but can be solved efficiently within an online optimization context.

Direct methods for optimal control first discretize then optimize the original problem and are the focus of this paper. In particular, we consider a multiple shooting (Bock & Plitt, 1984) discretization approach. Sequential quadratic programming (SQP) is a widely used algorithm in the field of real-time nonlinear model predictive control (NMPC) to tackle the resulting discrete-time OCP. Especially its application via the real-time iteration (RTI) scheme (Diehl, Bock, & Schlöder, 2005) is of particular interest in the context of online optimization. Within the RTI framework, a single SQP iteration is performed at each sampling time, which allows one to further split the required computation into a preparation and a feedback phase minimizing feedback delays.

One essential component of SQP software for NMPC based on direct multiple shooting are integration routines that solve initial value problems with possibly nonlinear and stiff differential equations and compute the sensitivities of the result with respect to the initial state and the control input (Frey, De Schutter, & Diehl, 2023; Quirynen, 2017). Often, these integration methods are simply referred to as *integrators*.

The above ingredients are implemented in the open-source software package `acados` which provides high-performance algorithms for optimal control (Verschueren et al., 2021). It internally uses the linear algebra package `BLASFEO`, which provides performance-optimized routines for small to medium sized matrix operations (Frison, Kouzoupis, Sartor, Zanelli, & Diehl, 2018). The `acados` software offers a very flexible optimization problem formulation supporting a wide range of optimal-control structured problems, such as classic optimal control problems (OCP) and moving horizon estimation (MHE) problems. Various discretization options are available, such as nonuniform grids, explicit and implicit integrators and dedicated functionalities to handle nonlinearities and linearities in cost and constraint functions efficiently. Moreover, a variety of quadratic programming (QP) solvers, such as `HPIPM`, `qpOASES`, `DAQP`, `OSQP`, `qpDUNES`, (Arnstrom, Bemporad, & Axehill, 2022; Ferreau, Kirches, Potschka, Bock, & Diehl, 2014; Frasch, Vukov, Ferreau, & Diehl, 2013; Frison & Diehl, 2020; Stellato, Geyer, & Goulart, 2017) are interfaced, which either tackle the OCP-structured QP directly or after applying full or partial condensing to it (Axehill, 2015; Frison, Kouzoupis, Jørgensen, & Diehl, 2016).

This paper focuses on the discretization and Hessian approximation of the cost function. We investigate how the control performance, both in terms of closed-loop cost and computation time, can be improved

☆ This research was supported by DFG via Research Unit FOR 2401 and projects 424107692, 504452366, by BMWK 03EN3054B, and by the EU via ELO-X 953348.

* Corresponding author at: Department of Microsystems Engineering (IMTEK), University Freiburg, Germany.
E-mail addresses: jonathan.frey@imtek.uni-freiburg.de (J. Frey), moritz.diehl@imtek.uni-freiburg.de (M. Diehl).

https://doi.org/10.1016/j.ejcon.2024.101038
Received 2 May 2024; Accepted 10 June 2024
Available online 12 June 2024

using a sophisticated cost discretization scheme. In particular, we focus on nonlinear-least squares objectives, which are common in control applications and allows us to use intrinsically positive-semidefinite Gauss–Newton Hessian approximations. We efficiently implemented the integration of a nonlinear least-squares Lagrange cost term together with its derivatives within an integrator, resulting in a Gauss–Newton Runge–Kutta (GNRK) integration method, recently proposed in Katliar (2022). In addition, we propose a simple but effective penalty formulation to incorporate state constraints with an L_2 penalty. The combination of the above ingredients are especially effective, in terms of accuracy and associated computational complexity, when applied to problems with relatively long horizons. The GNRK implementation is described and its effectiveness is demonstrated together with the use of RTI and a nonuniform discretization grid in terms of computation time and closed-loop cost on the illustrative example of a pendulum on a cart.

The remainder of the paper is structured as follows. Section 2 presents the continuous-time OCP and discusses in detail how to transform it into an NLP using multiple shooting. Section 3 describes the GNRK integrator. Section 4 presents numerical experiments and Section 5 concludes the paper and discuss the handling of state constraints via penalties.

2. Optimal control problem formulation

In this section, we start with a continuous-time optimal control problem (OCP) which we aim at approximating with a direct multiple shooting formulation that is suitable for real-time MPC. We give an overview and recommendations on the various discretization choices within the direct multiple shooting framework.

2.1. Continuous-time optimal control problem

We consider optimal control problems of the form

$$\min_{x(\cdot),u(\cdot)} \quad \int_0^\infty \ell(x(t),u(t))\,\mathrm{d}t \tag{1a}$$

$$\text{s.t.} \quad x(0) = \bar{x}_0 \tag{1b}$$

$$0 = f(t, x(t), \dot{x}(t), u(t)), \quad t \in [0,\infty) \tag{1c}$$

$$0 \geq g(x(t),u(t)), \qquad t \in [0,\infty), \tag{1d}$$

where $x(\cdot) : [0,\infty) \to \mathbb{R}^{n_x}$, $u(\cdot) : [0,\infty) \to \mathbb{R}^{n_u}$ are the state and control trajectories respectively, $\bar{x}_0$ is the initial state value, $f(\cdot)$ describes the implicit system dynamics and $g(\cdot)$ denotes the inequality constraints. The cost function consists of the integral of the Lagrange cost term $\ell(\cdot)$, which we assume to have the following nonlinear least-squares form

$$\ell(x,u) = \frac{1}{2}\|r(x,u)\|^2_W, \tag{2}$$

where $W \in \mathbb{R}^{n_y \times n_y}$ is positive definite and $r(\cdot) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$ is the potentially nonlinear residual function.

2.2. Discretization of the optimal control problem

In order to arrive at a finite dimensional approximation of the continuous problem (1), which can be solved online, a finite time horizon T and the number of shooting intervals N have to be chosen. The shooting intervals are $[t_n, t_{n+1}]$ with $t_0 = 0$ and $t_N = T$. The time steps are $\Delta t_n = t_{n+1} - t_n$ for $n = 0, \dots, N-1$. The multiple shooting OCP corresponding to (1) can then be stated as

$$\min_{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}}} \quad \sum_{n=0}^{N-1} L_n(x_n,u_n) + M(x_N) \tag{3a}$$

$$\text{s.t.} \quad x_0 = \bar{x}_0 \tag{3b}$$

$$x_{n+1} = \phi_n(x_n,u_n), \quad n = 0,\dots,N-1 \tag{3c}$$

$$0 \geq g_n(x_n,u_n), \quad n = 0,\dots,N-1 \tag{3d}$$

$$0 \geq g_{\text{terminal}}(x_N). \tag{3e}$$

Its optimization variables are the discrete control inputs u_n acting on $[t_n, t_{n+1}]$, $n = 0,\dots,N-1$, and the discrete states x_n at t_n, $n = 0,\dots,N$. The values x_n and x_{n+1} are coupled by integration methods (integrators) $\phi_n(\cdot)$ that discretize the continuous-time dynamics in (1c) and that can be different for all stages $n = 0,\dots,N-1$. The cost terms $L_n(\cdot)$ approximate the integral of the continuous cost over the shooting interval $[t_n, t_{n+1}]$. The constraints $g_n(\cdot)$ represent the continuous-time constraints on $[t_n, t_{n+1}]$. Most direct methods in optimal control only enforce the constraints at the shooting nodes. Lastly, the terminal constraint $g_{\text{terminal}}(\cdot)$ and the terminal cost term $M(\cdot)$ can be used to approximately summarize the infinite remainder of the horizon.

2.3. Constraint handling via direct penalty

In the context of NMPC, it is not recommended to impose hard constraints on the state, since this can render the OCP infeasible (Rawlings, Mayne, & Diehl, 2017). This issue is typically mitigated by *softening* all constraints which depend on the state. This means that a scalar constraint of the form $h(z) \leq 0$ in the variables z is replaced by

$$h(z) \leq s \tag{4}$$

using an additional optimization variable s, commonly referred to as *slack*, which is constrained to be nonnegative, $s \geq 0$. The slack is penalized in the cost function, by adding a term $\rho_s(s)$, which typically consists of an L_1 and/or L_2 penalty. Such slack variables can be considered as a control input in the context of optimization problem (3). However, many OCP specific solvers allow to handle them in a more dedicated fashion, exploiting the fact that they do not enter the dynamics (Frey, Di Cairano, & Quirynen, 2020; Frison & Diehl, 2020).

Since constraints are typically only imposed on the shooting nodes in the discrete-time OCP, the continuous-time trajectories corresponding to the discrete solution may violate the constraints between shooting nodes. This issue can be mitigated by directly adding the cost term corresponding to the constraint violation to the continuous objective, i.e.

$$\rho_s(\max(0, h(z))). \tag{5}$$

This can lead to a more accurate incorporation of the constraint cost if an accurate cost integration is employed, see below, especially if longer intervals are used.

In order to fit into the nonlinear least-squares framework, we propose to penalize constraint violations of (4) with an L_2 penalty with weighting parameter γ, by adding

$$\rho(z) = \gamma(\max(h(z), 0))^2 \tag{6}$$

to the cost function. The combination of such penalty functions for multiple constraints is visualized in Fig. 1. Note that for this penalty formulation, the Gauss–Newton Hessian corresponds to the exact Hessian, which is in this case not continuous. However, the Newton iterations can be analyzed within the framework of semismooth Newton methods, compare e.g. Hintermüller (2010).

In contrast, using an L_1 penalty instead of the L_2 penalty yields a nondifferentiable objective, rendering this formulation not directly suitable for the numerical method presented in this paper. Instead, L_1 penalties require the reformulation via a slack variable and inequalities. However, an extension of formulation (6) to convex penalties with a continuous gradient, such as the Huber loss, would also be suitable for direct numerical treatment and can be handled with a generalized or extended Gauss–Newton Hessian (Baumgärtner & Diehl, 2022).

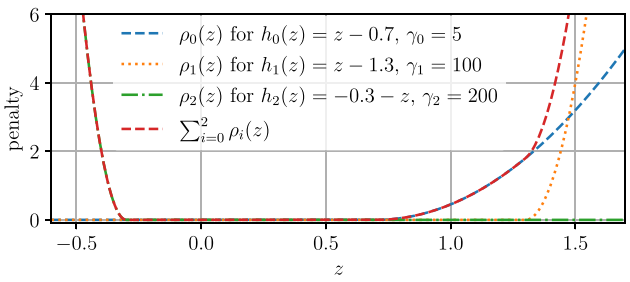

Fig. 1. Multiple constraints penalized via (6).

2.4. Cost integration scheme

Now that state constraints are incorporated into the cost function, we want to define $L_n(\cdot)$ to approximate the continuous-time cost term on the interval $[t_n, t_{n+1})$, i.e.

$$L_n(x_n, u_n) \approx \int_{t_n}^{t_{n+1}} \ell(x(t), u(t))\mathrm{d}t. \tag{7}$$

We consider two possible integration schemes:

(i) The shooting node cost discretization (SN) approximates the cost term as

$$L_n^{\mathrm{SN}}(x_n, u_n) = \Delta t_n \ell(x_n, u_n), \tag{8}$$

and corresponds to an Euler integration of the cost.

(ii) The (implicit) Runge–Kutta (RK) integration uses the same integration scheme which is used to integrate the dynamics, represented by $\phi_n(\cdot)$. The Runge–Kutta integration of the cost and, in particular, a Hessian approximation of this cost term, which is efficient to compute, are described in detail in Section 3.

Note that the two options coincide if the dynamics are discretized using one step of an explicit Euler scheme.

2.5. Practical considerations and nonuniform grids

In the context of real-time MPC, there are practical limitations relevant for the design of the MPC controller. Firstly, the plant or actuators typically have a minimum sampling time $\Delta t_{\mathrm{plant}}$, for which a control input should be applied. Operating the plant at a lower frequency restricts the control law unnecessarily, potentially sacrificing control performance. Thus, a controller should be able to output controls with the same frequency, i.e., the sampling time of the controller T_s should equal $\Delta t_{\mathrm{plant}}$, which we assume in the following. Secondly, the control input applied for one sampling period is typically constant. These considerations motivate a constant control input on the first shooting interval $[0, t_1]$ with $t_1 = T_s$.

While the remaining degrees of freedom in choosing a time grid are massive, the majority of practical applications of MPC restrict themselves to a uniform time grid. It is difficult to make general suggestions about this choice, but we want to motivate the use of a nonuniform in the following.

If the first shooting interval is longer, i.e., $\Delta t_0 > T_s$, this can lead to a loss of optimality, since the controls have to be chosen more conservatively, such that they do not drive the system away from a desired trajectory, even if applied for the longer period Δt_0. On the other hand, if $\Delta t_0 < T_s$, the controller might choose too aggressive actions for T_s, which are only safe to apply for the shorter period of Δt_0.

The prediction model and its discretization should be very accurate on the first shooting interval to avoid suboptimality of the open-loop trajectory due to model-plant mismatch on the first part of the horizon, which is applied to the real system.

Additionally, it is essential to have a sufficiently long time horizon such that the optimizer is aware of future constraints or future changes in the cost, such as reference changes. The crucial task of the latter part of the horizon which is not applied to the plant is to capture the cost-to-go accurately. However, since the computational cost of MPC algorithms scales at least linearly in the number of shooting nodes, a trade-off between a long time horizon and a low number of shooting nodes has to be made, which motivates the use of a nonuniform time grid.

Overall, these considerations encourage the use of a fine cost integration in contrast to the widely used shooting node discretization.

3. Implicit Runge–Kutta integration of the Lagrange term and Gauss–Newton Hessian approximation

In the following, we describe how the Lagrange term within the continuous cost (1a) can be integrated with the same integration scheme used for the system dynamics (1c). In particular, we show how the first-order, as well as an approximation of the second-order derivatives of the integrated Lagrange term can be computed with little computational overhead and a low memory footprint. Our presentation closely follows the one given in Katliar (2022).

3.1. Integrated cost and its derivatives

We assume that each integration interval $[t_n, t_{n+1})$ is subdivided into n_{steps} equidistant subintervals $[t_n^i, t_n^{i+1})$ with $t_n = t_n^0$, $t_{n+1} = t_n^{n_{\mathrm{steps}}}$ and $t_n^i = t_n + i\frac{\Delta t_n}{n_{\mathrm{steps}}}$ where $\frac{\Delta t_n}{n_{\mathrm{steps}}}$ is the length of each subinterval, $i = 1, \dots, n_{\mathrm{steps}}$. On each subinterval $[t_n^i, t_n^{i+1}]$, the following system of equations is solved:

$$s_n^{i,j} = x_n^i + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{l=1}^{n_{\mathrm{stages}}} a_{j,l} k_n^{i,l}, \tag{9}$$

$$0 = f(t_n^{i,j}, s_n^{i,j}, k_n^{i,j}, u_n), \tag{10}$$

for $j = 1, \dots, n_{\mathrm{stages}}$ and where $t_n^{i,j} = t_n^i + c_j \frac{\Delta t_n}{n_{\mathrm{steps}}}$. The final state at the end of the subinterval is obtained as

$$x_n^{i+1} = x_n^i + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{j=1}^{n_{\mathrm{stages}}} b_j k_n^{i,j}. \tag{11}$$

The coefficients $a_{j,l}$, b_j, c_j are given by the Butcher tableau defining a specific RK method. We obtain the integrated value of the cost at t_n^i as:

$$L_n^{i+1} = L_n^i + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{j=1}^{n_{\mathrm{stages}}} \frac{b_j}{2} \left\| r_n^{i,j} \right\|_W^2 \tag{12}$$

where $r_n^{i,j} = r(s_n^{i,j}, u_n)$. Differentiating (12) with respect to $w_n := (x_n, u_n)$, we obtain

$$\frac{\mathrm{d}L_n^{i+1}}{\mathrm{d}w_n} = \frac{\mathrm{d}L_n^i}{\mathrm{d}w_n} + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{j=1}^{n_{\mathrm{stages}}} b_j r_n^{i,j\top} W \tilde{J}_n^{i,j}, \tag{13}$$

where $\tilde{J}_n^{i,j} = \frac{\mathrm{d}r_n^{i,j}}{\mathrm{d}w_n}$. For the Hessian, we differentiate the above again to obtain

$$\frac{\mathrm{d}^2 L_n^{i+1}}{\mathrm{d}w_n^2} = \frac{\mathrm{d}^2 L_n^i}{\mathrm{d}w_n^2} + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{j=1}^{n_{\mathrm{stages}}} b_j \cdot \left(\tilde{J}_n^{i,j\top} W \tilde{J}_n^{i,j} + \sum_{l=1}^{n_y} r_n^{i,j,l} W_{l,\cdot} \frac{\mathrm{d}^2 r_n^{i,j,l}}{\mathrm{d}w_n^2} \right) \tag{14}$$

where $r_n^{i,j,l}$ is the lth component of $r_n^{i,j}$. Discarding the second term within the sum in (14), we obtain the Gauss–Newton (GN) Hessian approximation

$$\frac{\mathrm{d}^2 L_n^{i+1}}{\mathrm{d}w_n^2} \approx H_n^{i+1} := H_n^i + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{j=1}^{n_{\mathrm{stages}}} b_j \tilde{J}_n^{i,j\top} W \tilde{J}_n^{i,j}. \tag{15}$$

Note that the product $\tilde{J}_n^{i,j^\top} W \tilde{J}_n^{i,j}$ in (15) is positive semidefinite. Thus, if the coefficients b_j are nonnegative,[1] the GN Hessian approximation is positive semidefinite as well.

In summary, the GNRK cost integration technique is defined by using the cost

$$L_n^{\mathrm{GNRK}}(x_n, u_n) := L_n^{n_{\mathrm{steps}}}. \tag{16}$$

where $L_n^{n_{\mathrm{steps}}}$ as in (12). It is used together with its exact gradient (13) and the GN Hessian approximation in (15).

If used within an SQP-type algorithm, the Gauss–Newton Hessian approximation yields in general local linear convergence (Bock, 1983). The asymptotic linear rate depends on the deviation of the Gauss–Newton Hessian approximation from the exact Hessian at the solution (Messerer, Baumgärtner, & Diehl, 2021). The deviation is explicitly given by

$$E_n^{i+1} := E_n^i + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{j=1}^{n_{\mathrm{stages}}} b_j \left(\sum_{l=1}^{n_y} r_n^{i,j,l} W_{l,\cdot} \frac{\mathrm{d}^2 r_n^{i,j,l}}{\mathrm{d}w_n^2} \right). \tag{17}$$

Thus, we expect fast linear convergence if the residuals $r_n^{i,j,l}$ are close to zero.

3.2. Efficient computation of the Gauss–Newton Hessian

This section describes how the Gauss–Newton Hessian of the integrated cost can be obtained with minor additional computations within an integrator that already delivers first-order derivatives. Applying the chain rule, we can express $\tilde{J}_n^{i,j}$ as

$$\tilde{J}_n^{i,j} = J_n^{i,j} S_n^{i,j} \tag{18}$$

where we introduced

$$J_n^{i,j} = \frac{\partial r_n^{i,j}}{\partial w}$$

$$S_n^{i,j} = \frac{\mathrm{d}(s_n^{i,j}, u_n)}{\mathrm{d}w_n} = \begin{bmatrix} \frac{\mathrm{d}s_n^{i,j}}{\mathrm{d}x_n} & \frac{\mathrm{d}s_n^{i,j}}{\mathrm{d}u_n} \\ \mathbb{0}_{n_u \times n_x} & \mathbb{1}_{n_u \times n_u} \end{bmatrix}.$$

Differentiating (9), we obtain

$$\frac{\mathrm{d}s_n^{i,j}}{\mathrm{d}w_n} = \frac{\mathrm{d}x_n^i}{\mathrm{d}w_n} + \frac{\Delta t_n}{n_{\mathrm{steps}}} \sum_{l=1}^{n_{\mathrm{stages}}} a_{j,l} \frac{\mathrm{d}k_n^{i,l}}{\mathrm{d}w_n}. \tag{19}$$

The derivatives $\frac{\mathrm{d}k_n^{i,l}}{\mathrm{d}w_n}$ are available within the forward propagation of any Runge–Kutta integrator that applies internal numerical differentiation and can be reused. For a detailed description of forward sensitivity propagation within implicit integrators, we refer to Quirynen (2017).

The additional computations for using GNRK instead of SN on a shooting interval are $n_{\mathrm{stages}} n_{\mathrm{steps}}$ evaluations of $r(\cdot)$ and $\frac{\partial r}{\partial w}(\cdot)$ and matrix–matrix multiplications with dimension $n_y \times (n_x + n_u)$, $(n_x + n_u) \times n_x$, respectively $(n_x + n_u) \times (n_x + n_u)$. The additional linear algebra operations are all of order $(n_x + n_u)^2$ or less, assuming that $n_y \leq (n_x + n_u)$. Thus, it is of a lower order compared to the fastest QP solution algorithms, which require computations with order n_x^3 and n_u^3 (Frey et al., 2020; Frison, 2015). On the other hand, in the SN discretization, the functions $r(\cdot)$ and $\frac{\partial r}{\partial w}(\cdot)$ would only be evaluated once instead of $n_{\mathrm{stages}} n_{\mathrm{steps}}$ times, which might dominate the computational cost. However, if these evaluations would dominate, the functions are likely to be very nonlinear and a more accurate integration is desirable. Since the Hessian contributions of each point within the RK integrator can be accumulated on the fly using (15), (18) and (19), the additional memory footprint is small and independent of n_{steps} and n_{stages}. Note that these considerations also hold for explicit Runge–Kutta method, such as the widely used RK4 method. However, the relative additional computational cost would be higher compared to the implicit GNRK implementation considered in this work, since explicit integrators are typically faster.

[1] Note that this is the case for Gauss–Radau IIA and Gauss–Legendre tableaus with $n_{\mathrm{stages}} = 1, \ldots, 9$ and all explicit tableaus implemented in `acados` at the time of writing, but not true in general, e.g. some DIRK methods (Hairer & Wanner, 1991) use negative coefficients b_j.

Table 1
Overview on discretization and solver options varied in the benchmark of this paper.

Option	Variant I	Variant II
Hessian (approximation)	Gauss–Newton (GN)	Exact Hessian (EH)
cost discretization	Shooting Node (SN)	Runge–Kutta (RK)
discretization grid	uniform (a)	nonuniform (b)
algorithm type	converged SQP	RTI
shooting intervals N	20	200
Time horizon T	0.4	4.0

3.3. Comparison to generic quadrature states

Note that a common alternative to the approach described in the previous sections is to propagate the cost with the same accuracy as the dynamics via a cost state. Efficient integrators support a dedicated treatment of quadrature variables, i.e., variables which do not enter the dynamics, such that the implicit system of equations can be decoupled and solved in the original space (Hindmarsh et al., 2005). This idea has been extended to integrators that exploit more general linear structures within the dynamic system (Frey et al., 2019; Quirynen, Gros, & Diehl, 2013).

The generic quadrature state approach however does not take the cost function's nonlinear least-squares structure into account. Thus, it is limited to an exact Hessian propagation, which might result in an indefinite Hessian and the associated problems within an NLP solver. From a computational perspective, an exact Hessian propagation requires at least an additional adjoint sweep and thus more computational resources. Different methods for Hessian propagation exist which trade-off computations and memory footprint (Quirynen, 2017).

3.4. Implementation in `acados`

The GNRK algorithm described in Sections 3.1 and 3.2 has been implemented in the open-source software package `acados`, which provides high-performance, embedded solvers for nonlinear optimal control. The cost integration was implemented as an option in the `acados` IRK module for nonlinear-least squares cost functions (2).

The GNRK algorithm is compatible with additional Hessian contributions form the constraints. In particular, when including Hessian contributions from constraints with a convex-over-nonlinear structure as described in Verschueren, van Duijkeren, Quirynen, and Diehl (2016), this results in a sequential convex quadratic programming (SCQP) scheme with integrated cost.

4. Numerical experiments

In this section, we illustrate the effectiveness of the presented strategies in terms of closed-loop cost with two numerical simulation studies. All experiments have been carried out using `acados` v0.3.1 via its `Python` interface on a Laptop with an Intel i5-8365U CPU, 16 GB of RAM running Ubuntu 22.04. The code to reproduce the results is publicly available.[2]

4.1. Inverted pendulum on cart problem

In order to demonstrate the importance of cost discretization, we regard the widely studied control problem of stabilizing an inverted

[2] https://github.com/FreyJo/GNRK_benchmark

Table 2
Closed-loop performance of different controllers measured by relative suboptimality, maximum and minimum computation time, and SQP iterations n_{iter} over the scenario depicted in Fig. 2.

Hessian approximation and cost discretization	N	T[s]	RTI	uniform	rel. subopt.	max n_{iter}	median n_{iter}	$t_{\min}$[ms]	$t_{\max}$[ms]	in Fig. 2
GNRK	200	4.0		x	**0.0** %	21	4.89	25.3	207.4	A
GNSN	200	4.0		x	**0.0** %	20	4.97	25.9	206.2	
GNRK	20	4.0	x		**3.6** %	1	1.00	**0.9**	1.1	B
GNRK	20	4.0			**3.7** %	15	4.08	1.8	15.5	
GNSN	20	4.0	x		68.3%	1	1.00	**0.8**	**1.0**	C
GNSN	20	4.0			34.3%	33	4.42	1.7	31.5	
EHSN	20	4.0			34.3%	400	8.48	1.8	393.1	
EHRK	20	4.0			**3.7** %	50	5.85	4.8	72.7	
GNRK	20	4.0	x	x	992.3%	1	1.00	**0.9**	**1.0**	D
GNRK	20	4.0		x	845.4%	23	5.70	2.5	20.6	
GNSN	20	4.0	x	x	960.9%	1	1.00	**0.8**	**1.0**	
GNSN	20	4.0		x	823.4%	48	6.38	2.7	46.9	
GNRK	20	0.4	x	x	3524.8%	1	1.00	**0.9**	1.8	
GNRK	20	0.4		x	3356.3%	400	164.53	2.7	688.0	

pendulum mounted onto a cart. The differential state of the model is $x = [p, \theta, s, \omega]^\top$ with cart position p, cart velocity s, angle of the pendulum θ and angular velocity ω. The system dynamics can be found e.g. in Verschueren et al. (2021). The control input u is a force acting on the cart in the horizontal plane and constrained to be in $[-40, 40]$. The example simulation starts with an initial state $\bar{x}_0 = [0, \bar{\theta}_0, 0, 0]^\top$ with $\bar{\theta}_0 = \frac{\pi}{5}$. The goal is to drive all states to zero, i.e. the unstable upright position. We formulate the nonlinear least squares cost

$$l_{\text{pend}}(x, u) = x^\top Q x + u^\top R u + \rho_0(x) + \rho_1(x), \tag{20}$$

consisting of quadratic costs on states and controls, with weights $Q = \text{diag}(100, 10^3, 0.01, 0.01)$, $R = 0.2$, and additional penalty terms $\rho_i(x)$ corresponding to the inequalities $p_{\min} - p \leq 0$ and $p - p_{\max} \leq 0$ with $p_{\min} = -1, p_{\max} = 1$ and $\gamma = 5 \cdot 10^4$, c.f. (6). The terminal cost term is set to $M_{\text{pend}}(x) = x^\top P x$, where P is obtained as solution of the discrete algebraic Riccati equation with cost and dynamics linearized at the steady-state.

4.2. Controller variants in closed-loop

We study the behavior of different controller variants in a closed-loop simulation of 4s. The plant is represented by a single-step IRK integrator that uses the Radau IIA Butcher tableau with $n_{\text{stages}} = 4$ with a sampling time of $T_s = 0.02$s. It internally uses a model that is augmented with a cost state to accurately capture the evolution of (20) over time.

All controllers use `HPIPM` without condensing as a QP solver and a single step of IRK with Radau IIA Butcher tableau and $n_{\text{stages}} = 4$ on each shooting interval solving the system of RK equations to a tolerance of $\epsilon_{\text{IRK}} = 10^{-12}$. The time horizon is chosen to be $T = 4$s if not otherwise stated and divided using one of the following time grids:

(a) uniform time grid with $\Delta t_n = \frac{T}{N}$, $n = 0, \ldots, N-1$
(b) nonuniform time grid using the sampling time $T_s = 0.02$s on the first interval, $\Delta t_0 = T_s$, and dividing the remainder equally between the other intervals, i.e., $\Delta t_n = \frac{T - T_s}{N-1}$, $n = 1, \ldots, N-1$.

In terms of cost-discretization, we compare the shooting node (SN) and the Runge–Kutta (RK) versions, which can be combined either with the Exact Hessian (EH) or the Gauss–Newton Hessian (GN), such that the proposed approach is GNRK. Additionally, we look at converged SQP and SQP-RTI and different number of shooting intervals N. An overview on the discretization and solver options varied in this benchmark is given in Table 1.

In Fig. 2, the closed-loop trajectories of different controllers are visualized. Key performance indicators of even more variants are listed in Table 2. The minimum and maximum computation time $t_{\min}$, respectively $t_{\max}$ are evaluated after running the exact same simulation 5 times and taking the minimum of each execution to remove artifacts.

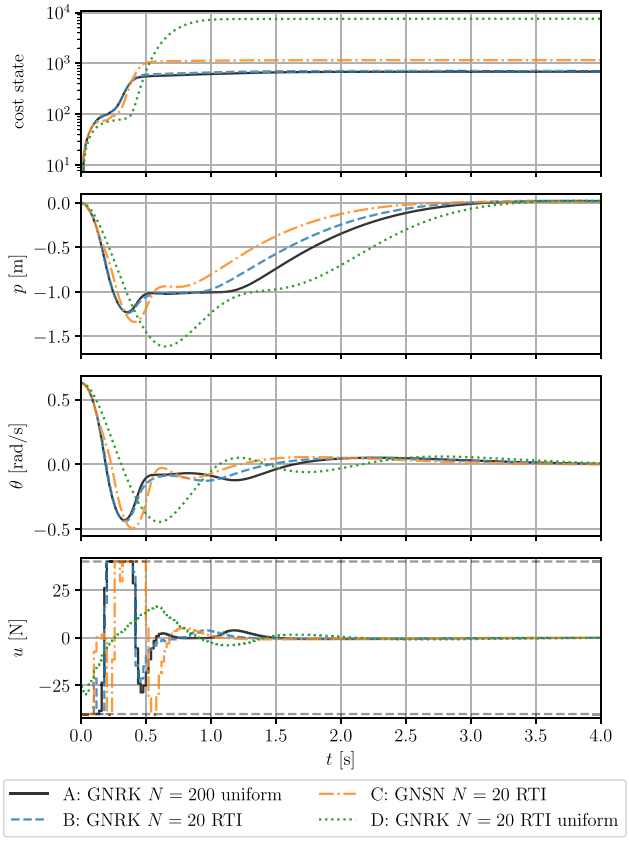

Fig. 2. Closed-loop state and control trajectories for different controllers stabilizing a pendulum on cart system. The corresponding computation times are given in Table 2.

The relative suboptimality is obtained by comparing the total closed-loop cost, i.e., the integrated cost state, with the one of an ideal controller, i.e. without model-plant mismatch.

The black line in Fig. 2 shows the reference controller with a fine uniform time grid and GNRK cost discretization. In this case, there is no model-plant mismatch and the difference between GNRK and SN cost discretization is negligible, see Table 2. The controller variants with GNRK cost discretization, a nonuniform grid and $N = 20$ result in a very similar closed-loop cost with only around 3.7% of relative suboptimality with respect to the baseline. In contrast, using the standard SN cost discretization with otherwise the same settings, results in a suboptimality of 34% and 68% for converged SQP and RTI respectively. When using controller variants with a uniform grid

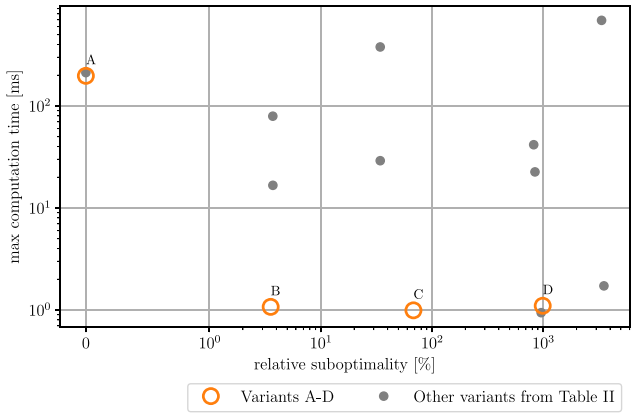

Fig. 3. Pareto plot comparing computation time and relative suboptimality of different controllers, see Table 1. The x-axis is linearly scaled in $[0, 1]$.

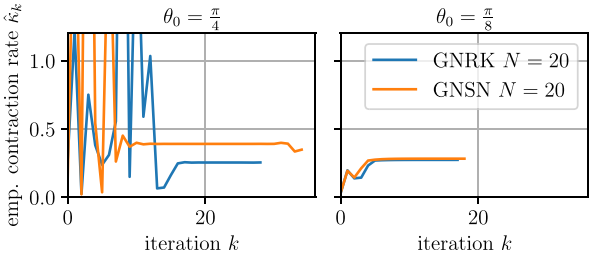

Fig. 4. Empirical contraction rate for different initial states of the pendulum example. Both controllers use a uniform grid with $T = 4$s and $n_{\text{stages}} = 4$.

and $\Delta t_0 = 10T_s$, the control performance drastically degrades, resulting in a relative suboptimality of over 800%. On the other hand, using a uniform grid with $\Delta t_0 = T_s$ results in a very short horizon length T, when keeping $N = 20$ fix. The corresponding controller does not stabilize the pendulum and results in over 3000% of relative suboptimality.

The computation times in Table 2 show that all Gauss–Newton variants with $N = 20$ and RTI have a similar runtime. As expected, the variants with converged SQP have a much higher variance in CPU time. Comparing the versions with exact Hessian in Table 2 to their GN counterparts, we observe that they converge to the same solution. However, the minimum runtime is more than twice as high. The computation times of the GN variants with $N = 200$ are roughly tenfold of the corresponding version with $N = 20$. Regarding the number of SQP iterations in Table 2, we observe that while the median number of iterations is similar for GNRK and GNSN, the maximum number is roughly half for GNRK, indicating better convergence properties.

Fig. 3 visualizes the Pareto front of different controllers from Table 2 in terms of relative suboptimality and maximum computation time. The latter ultimately determines if a controller is real-time feasible. It can be seen that the proposed controller variant B results in a reduction of relative suboptimality by a factor of 18 while increasing the maximum computation time by less than 10%, compared to controller variant C, i.e., the same controller without cost integration, which was regarded as an attractive variant before this work.

Overall, the results indicate that using GNRK allows one to drastically reduce the number of shooting intervals as long as the first interval is kept at T_s.

4.3. Empirical contraction rate

In order to give insights into why the maximum number of iterations in Table 2 is higher for GNSN than for GNRK, we regard the empirical contraction rate $\hat{\kappa}_k = \frac{\|d_{k+1}\|}{\|d_k\|}$, where d_k denotes the step in all variables at SQP iteration k. The empirical contraction rate is plotted in Fig. 4 for two initial states, $\bar{x}_0 = [0, \theta_0, 0, 0]^\top$. We observe that GNRK converges with a faster rate. For the easier initial state with $\theta_0 = \frac{\pi}{4}$, the difference is not significant, while for the initial state with $\theta_0 = \frac{\pi}{8}$, the $\hat{\kappa}_k$ values close to the solution are significantly smaller.

5. Conclusion & outlook

The GNRK integrator has been shown to handle nonlinear least-squares OCPs with long horizons effectively, as it trades off accuracy and computational complexity. We showed that soft L_2 constraints can be handled accurately without additional slack variables by integrating the constraint violation penalty, which perfectly fits the GNRK framework. The effectiveness of GNRK combined with the use of nonuniform discretization grids and L_2 penalties for state constraints has been demonstrated on an illustrative example. Furthermore, this paper gave some recommendations that can help MPC practitioners to formulate and discretize their problems to obtain a competitive solver implementation.

Possible future work includes an extension of the current GNRK implementation in `acados` to handle generalized and extended Gauss–Newton Hessian terms (Baumgärtner & Diehl, 2022), which would allow one to handle more general convex-over-nonlinear cost and penalty functions.

CRediT authorship contribution statement

Jonathan Frey: Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Software, Writing – original draft. **Katrin Baumgärtner:** Conceptualization, Data curation, Formal analysis, Investigation, Visualization, Writing – original draft, Writing – review & editing. **Moritz Diehl:** Conceptualization, Formal analysis, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Arnstrom, D., Bemporad, A., & Axehill, D. (2022). A dual active-set solver for embedded quadratic programming using recursive LDL^T updates. *IEEE Transactions on Automatic Control*, http://dx.doi.org/10.1109/TAC.2022.3176430.

Axehill, D. (2015). Controlling the level of sparsity in MPC. *Systems & Control Letters*, *76*, 1–7.

Baumgärtner, K., & Diehl, M. (2022). The extended Gauss-Newton method for nonconvex loss functions and its application to time-optimal model predictive control. In *Proceedings of the American control conference*.

Bock, H. G. (1983). Recent advances in parameter identification techniques for ODE. In *Numerical treatment of inverse problems in differential and integral equations* (pp. 95–121). Birkhäuser.

Bock, H. G., & Plitt, K. J. (1984). A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the IFAC world congress* (pp. 242–247). Pergamon Press.

Diehl, M., Bock, H. G., & Schlöder, J. P. (2005). A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, *43*(5), 1714–1736. http://dx.doi.org/10.1137/S0363012902400713.

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., & Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, *6*(4), 327–363.

Frasch, J. V., Vukov, M., Ferreau, H., & Diehl, M. (2013). A dual Newton strategy for the efficient solution of sparse quadratic programs arising in SQP-based nonlinear MPC. Optimization Online 3972.

Frey, J., De Schutter, J., & Diehl, M. (2023). Fast integrators with sensitivity propagation for use in casADi. In *Proceedings of the European control conference*.

Frey, J., Di Cairano, S., & Quirynen, R. (2020). Active-set based inexact interior point QP solver for model predictive control. In *Proceedings of the IFAC world congress*.

Frey, J., Quirynen, R., Kouzoupis, D., Frison, G., Geisler, J., Schild, A., et al. (2019). Detecting and exploiting generalized nonlinear static feedback structures in DAE systems for MPC. In *Proceedings of the European control conference*.

Frison, G. (2015). *Algorithms and methods for high-performance model predictive control* (Ph.D. thesis), Technical University of Denmark (DTU).

Frison, G., & Diehl, M. (2020). HPIPM: A high-performance quadratic programming framework for model predictive control. In *Proceedings of the IFAC world congress*. Berlin, Germany.

Frison, G., Kouzoupis, D., Jørgensen, J. B., & Diehl, M. (2016). An efficient implementation of partial condensing for nonlinear model predictive control. In *Proceedings of the IEEE conference on decision and control* (pp. 4457–4462).

Frison, G., Kouzoupis, D., Sartor, T., Zanelli, A., & Diehl, M. (2018). BLASFEO: Basic linear algebra subroutines for embedded optimization. *ACM Transactions on Mathematical Software*, *44*(4), 42:1–42:30.

Hairer, E., & Wanner, G. (1991). *Solving ordinary differential equations II – stiff and differential-algebraic problems* (2nd ed.). Berlin Heidelberg: Springer.

Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D., et al. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, *31*(3), 363–396.

Hintermüller, M. (2010). *Semismooth Newton methods and applications*. Department of Mathematics, Humboldt-University of Berlin.

Katliar, M. (2022). *Optimal control of motion simulators* (Ph.D. thesis), Albert-Ludwigs-Universität Freiburg.

Messerer, F., Baumgärtner, K., & Diehl, M. (2021). Survey of sequential convex programming and generalized Gauss-Newton methods. *ESAIM: Proceedings and Surveys*, *71*, 64–88.

Quirynen, R. (2017). *Numerical simulation methods for embedded optimization* (Ph.D. thesis), KU Leuven and University of Freiburg.

Quirynen, R., Gros, S., & Diehl, M. (2013). Efficient NMPC for nonlinear models with linear subsystems. In *Proceedings of the IEEE conference on decision and control* (pp. 5101–5106).

Rawlings, J. B., Mayne, D. Q., & Diehl, M. M. (2017). *Model predictive control: Theory, computation, and design* (2nd ed.). Nob Hill.

Stellato, B., Geyer, T., & Goulart, P. J. (2017). High-speed finite control set model predictive control for power electronics. *Institute of Electrical and Electronics Engineers. Transactions on Automatic Control*, *32*(5), 4007–4020.

Verschueren, R., van Duijkeren, N., Quirynen, R., & Diehl, M. (2016). Exploiting convexity in direct optimal control: A sequential convex quadratic programming method. In *Proceedings of the IEEE conference on decision and control*.

Verschueren, R., Frison, G., Kouzoupis, D., Frey, J., van Duijkeren, N., Zanelli, A., et al. (2021). Acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 147–183. http://dx.doi.org/10.1007/s12532-021-00208-8.