2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Reinforced Model Predictive Control via Trust-Region Quasi-Newton Policy Optimization

Dean Brandner and Sergio Lucia

Abstract— **Model predictive control can optimally deal with nonlinear systems under consideration of constraints. The control performance depends on the model accuracy and the prediction horizon. Recent advances propose to use reinforcement learning applied to a parameterized model predictive controller to recover the optimal control performance even if an imperfect model or short prediction horizons are used. However, common reinforcement learning algorithms rely on first order updates, which only have a linear convergence rate and hence need an excessive amount of dynamic data. Higher order updates are typically intractable if the policy is approximated with neural networks due to the large number of parameters.**

In this work, we use a parameterized model predictive controller as policy, and leverage the small amount of necessary parameters to propose a trust-region constrained Quasi-Newton training algorithm for policy optimization with a superlinear convergence rate. We show that the required second order derivative information can be calculated by the solution of a linear system of equations. A simulation study illustrates that the proposed training algorithm outperforms other algorithms in terms of data efficiency and accuracy.

I. Introduction

Optimal control strategies such as model predictive control (MPC) enable the control of nonlinear systems while taking constraints into rigorous consideration. MPC repeatedly solves the underlying optimal control problem at each time instance and applies the first control action to the plant [1]. However, a good performance typically requires an accurate system model and a large prediction horizon, which can render the optimization problem intractable for real-time applications. Real-time capability can be recovered by, e.g. using simpler system models or a shorter prediction horizon, both at the expense of accuracy for faster computation.

While MPC relies on the prediction of a state trajectory using a system model, reinforcement learning provides model-free methods to solve the dynamic optimization problem, as for instance policy optimization [2]. To do so, an agent computes an action according to its policy and applies the action to an environment. The agent's policy is then updated iteratively based on the next state and stage cost to find the optimal policy. State-of-the-art performance for control tasks with continuous action spaces using deterministic policies can be obtained using neural networks (NNs) to approximate the policy [3]. In these algorithms, the NN parameters are updated iteratively using a deterministic policy gradient algorithm [4] until the parameters converge. Due to the mostly random initialization of the weights and biases of NNs, their lack of structure, and the linear convergence rate of gradient descent algorithms [5], the demand for training data is usually extremely high in reinforcement learning.

Different studies suggest to decrease the demand of data by taking more elaborate update steps such as natural policy gradients [6], [7], which scales the gradient by the inverse of the Fisher information matrix, or Quasi-Newton update steps [5], [8], [9], which scales the gradient by the inverse of an approximation of the Hessian. Although showing practical improvements, natural policy gradient methods still have a linear convergence rate. Quasi-Newton methods however can have a superlinear convergence rate, which can significantly reduce the demand on training data. Standard implementations of reinforcement learning algorithms rely on heavily parameterized NNs as policy approximators, which can render the training process for second order methods intractable due to the large resulting matrices and linear systems of equations. For this reason, first order optimization methods are almost exclusively considered in literature.

In this work, we propose to use a parameterized MPC as policy approximator instead of large NNs, as it has been recently proposed [10], [11], [12]. The central advantage of this strategy is that the parameterized MPC is an optimization problem in which the different parts, such as the objective or the constraint functions, can be parameterized. This leads typically to significantly less parameters than if large NNs are considered as policy approximators. Tools from reinforcement learning can then be used to recover the optimal policy by updating the MPC parameters, even if the system model is inaccurate or a short prediction horizon is used. In addition, using MPC as a policy approximator profits from a reasonably good initial policy when expert knowledge is supplied, e.g. in the form of a rough dynamic model. However, it appears that still a significant amount of data is typically required for the MPC policy to converge when employing first order updates. Alleviating this challenge is the main motivation of this work.

The main contributions of our work are the following. We exploit the small number of parameters, which typically arise when using MPC as a policy approximator in reinforcement learning, by using Quasi-Newton update steps to reduce the demand of training data. We propose a method to calculate the second order sensitivities of the optimal control actions with respect to the parameters to compute an approximation

The authors are with the chair of Process Automation Systems at the department of Biochemical and Chemical Engineering, TU Dortmund University, 44227 Dortmund, Germany (e-mail: dean.brandner@tu-dortmund.de; sergio.lucia@tu-dortmund.de).

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 466380688 – within the Priority Program "SPP 2331: Machine Learning in Chemical Engineering".

of the deterministic policy Hessian. We integrate the approximation in a trust-region constrained Quasi-Newton policy optimization algorithm for episodic reinforcement learning to further improve the data efficiency and accuracy.

The paper is structured as follows. Section II introduces the background of Markov decision processes and MPC as a policy approximator. Section III shows how Quasi-Newton update steps can be computed and introduces the prerequisites. In section IV we show how trust region constrained Quasi-Newton updates can be embedded in an episodic reinforcement learning setting. Lastly, we demonstrate the performance of the proposed algorithm in section V before we summarize the results in section VI.

II. Background

A. Markov Decision Processes

Reinforcement learning can solve Markov decision processes via interaction of an agent with an environment. A transition to the subsequent state $s' \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$ in the possibly stochastic environment is modelled as a transition possibility distribution $p(s'|s,a)$ with current state $s \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$ and action $a \in \mathcal{A} \subseteq \mathbb{R}^{n_a}$. In addition to the next state, the environment also responds with a scalar stage cost $\ell(s,a) \in \mathbb{R}$ also known as negative reward, which indicates how good a defined objective is fulfilled when being in state s and taking action a. Also, the stage cost can penalize constraint violations by large costs.

Action a is computed via the agent's policy $\pi : \mathcal{S} \to \mathcal{A}$. The objective is to find the optimal policy π^*, which minimizes the expected closed-loop cost $J(\pi)$. For an episodic process of $N_{\mathrm{ep}} \in \mathbb{N}$ steps, the closed-loop cost is defined as

$$J(\pi) = \mathbb{E}_{s_0 \sim S_0}\left[\sum_{i=0}^{N_{\mathrm{ep}}} \gamma^i \ell(s_i, \pi(s_i))\right] \tag{1}$$

with $\gamma \in (0,1]$ being a discount factor. The operator $\mathbb{E}_{s_0\sim S_0}[\cdot]$ denotes the expected value taken over the initial states s_0 of an episode when sampled from some distribution S_0. The optimal policy can then be obtained via

$$\pi^* = \arg\min_{\pi} J(\pi). \tag{2}$$

The state-value function V^π now gives information on the closed-loop cost given state s and following policy π, while the action-value function $Q^\pi(s,a)$ gives information on the closed-loop cost given state s, taking action a, and following policy π afterwards. They are recursively defined using the Bellman equations as

$$V^\pi(s) = Q^\pi(s, \pi(s)), \tag{3a}$$
$$Q^\pi(s,a) = \ell(s,a) + \gamma \mathbb{E}_{s'}\left[V^\pi(s')\right]. \tag{3b}$$

B. Parameterized MPC as Policy Approximator in Reinforcement Learning

MPC is an advanced control scheme, which repeatedly computes a sequence of optimal control actions $\mathbf{u}^* = (u_0^*, \dots u_{N-1}^*)^\top$ with $u_i^* \in \mathcal{A}$ by solving (4) at each time instance t_k and applies the first control action u_0^* to the plant, so $a = u_0^*$

$$\begin{aligned}\mathbf{u}_\theta^*(s) = \arg\min_{\mathbf{u}} \quad & \gamma^N \left(V_{\mathrm{f},\theta}(x_N) + w_{\mathrm{f}}^\top \sigma_N\right) \\ & + \sum_{k=0}^{N-1} \gamma^k \left(\ell_\theta(x_k, u_k) + w^\top \sigma_k\right) && \text{(4a)} \\ \text{s.t.} \quad & x_{k+1} = \hat{f}_\theta(x_k, u_k), \quad x_0 = s && \text{(4b)} \\ & h_\theta(x_k, u_k) \le \sigma_k, \quad \sigma_k \ge 0, && \text{(4c)} \\ & h_{\mathrm{f},\theta}(x_N) \le \sigma_N, \quad \sigma_N \ge 0. && \text{(4d)}\end{aligned}$$

The objective function (4a) is composed of the sum of discounted stage costs $\ell_\theta(x_k, u_k) \ge 0$ over a finite horizon $N - 1 \in \mathbb{N}$, and the discounted terminal cost $V_{\mathrm{f},\theta}(x_N) \ge 0$. The states x_k evolve following the underlying system model $\hat{f}_\theta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ starting from a given initial state $x_0 = s \in \mathcal{S}$ shown in (4b) while satisfying constraints $h_\theta(x_k, u_k) \in \mathbb{R}^{n_h}$ at each time instance (4c) and $h_{\mathrm{f},\theta}(x_N) \in \mathbb{R}^{n_{h_\mathrm{f}}}$ at the end of the horizon (4d). The constraints (4c) and (4d) are relaxed as soft constraints by σ_k and σ_N.

All functions with index $\theta \in \mathbb{R}^{n_\theta}$ are freely parameterizable. Since it is shown in [10] that the parameterized MPC (4) can approximate the optimal policy, reinforcement learning can be used to adapt the parameters θ such that the closed-loop cost is minimized.

C. Iterative Policy Optimization in Reinforcement Learning

Reinforcement learning considers different options to compute the agent's optimal behaviour. One method is called policy optimization [2], which uses an approximation π_θ to learn the optimal policy π^*. The policy parameters θ are then updated iteratively using the update vector $\Delta\theta \in \mathbb{R}^{n_\theta}$ according to the general update scheme

$$\theta \leftarrow \theta + \Delta\theta. \tag{5}$$

A commonly used update rule to minimize the closed-loop cost is motivated by gradient descent [2]

$$\Delta\theta = -\alpha \nabla_\theta J(\theta), \tag{6}$$

with learning rate $\alpha > 0$, and the deterministic policy gradient $\nabla_\theta J(\theta) \in \mathbb{R}^{n_\theta}$, which is derived in [4] as

$$\nabla_\theta J(\theta) = \mathbb{E}_s\left[\nabla_\theta \pi_\theta^\top(s) \nabla_a Q^{\pi_\theta}(s,a)|_{a=\pi_\theta(s)}\right]. \tag{7}$$

Since we propose to use a parameterized MPC (4) as policy approximators, the Jacobian $\nabla_\theta \pi_\theta$ requires to differentiate the solution of (4) with respect to its parameters. Section III-A shows how these first order sensitivities can be computed. As we use NNs in this work to approximate the Q-function, automatic differentiation can be used to obtain $\nabla_a Q$.

Second order methods typically have higher convergence rates and hence require less data. One can derive an update of the form

$$\Delta\theta = -\alpha \nabla_\theta^2 J(\theta)^{-1} \nabla_\theta J(\theta), \tag{8}$$

which is also known as a Newton step. An exact expression for the deterministic policy Hessian $\nabla_\theta^2 J(\theta) \in \mathbb{R}^{n_\theta \times n_\theta}$ is derived in [9].

III. Quasi-Newton Iteration for Policy Optimization

Due to computational complexity, the exact deterministic policy Hessian $\nabla^2_\theta J(\theta)$ is typically intractable [9]. However, under some conditions the convergence rate can still be superlinear even if the deterministic policy Hessian is not known exactly but only an approximation $H(\theta) \approx \nabla^2_\theta J(\theta)$. The update then looks similar to (8)

$$\Delta\theta = -\alpha H(\theta)^{-1} \nabla_\theta J(\theta). \tag{9}$$

The deterministic policy Hessian can be approximated as [9]

$$\begin{aligned} H(\theta) = \mathbb{E}_s \big[&\nabla^2_\theta \pi_\theta(s) \otimes \nabla_a Q^{\pi_\theta}(s,a)|_{a=\pi_\theta} + \\ &\dots \nabla_\theta \pi_\theta^\top(s) \nabla^2_a Q^{\pi_\theta}(s,a)|_{a=\pi_\theta} \nabla_\theta \pi_\theta(s) \big], \end{aligned} \tag{10}$$

which can still give a superlinear convergence rate under some assumptions [9]. The expression requires the second order sensitivity tensor $\nabla^2_\theta \pi_\theta(s) \in \mathbb{R}^{n_\theta \times n_\theta \times n_a}$, as well as the first order sensitivity matrix $\nabla_\theta \pi$. The $\otimes$ operator denotes the tensor vector product [9]. Section III-B introduces a method to compute these second order sensitivities.

A. First Order Sensitivities of Nonlinear Programs

The deterministic policy gradient (7) and approximate Hessian (10) require $\nabla_\theta \pi_\theta(s)$ and $\nabla^2_\theta \pi_\theta(s)$, which are the first and second order sensitivities of the solution of (4).

As a general case of (4), consider the nonlinear program (11) with the objective function $\Phi : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to [0, \infty)$, decision variables $z \in \mathbb{R}^{n_z}$, parameters $p \in \mathbb{R}^{n_p}$, and the equality and inequality constraints $h : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_h}$ and $g : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_g}$

$$z^*(p) = \arg\min_z \quad \Phi(z,p) \tag{11a}$$
$$\text{s.t.} \quad h(z,p) = 0 \tag{11b}$$
$$g(z,p) \leq 0. \tag{11c}$$

Let $z^*(p)$ denote the solution of (11) in dependency of the parameter vector p and let $\mathcal{L} : \mathbb{R}^{n_z} \times \mathbb{R}^{n_g} \times \mathbb{R}^{n_h} \times \mathbb{R}^{n_p} \to \mathbb{R}$ be the Lagrangian associated to (11) with Lagrange multipliers $\lambda \in \mathbb{R}^{n_g}, \nu \in \mathbb{R}^{n_h}$

$$\mathcal{L}(z, \lambda, \nu, p) = \Phi(z,p) + \lambda^\top g(z,p) + \nu^\top h(z,p), \tag{12}$$

then the optimal primal-dual solution vector $\xi^{*\top} = [z^{*\top}, \lambda^{*\top}, \nu^{*\top}] \in \mathbb{R}^{n_\xi}$ with $n_\xi = n_z + n_g + n_h$ satisfies the KKT-conditions [13]. When omitting the inequalities of the KKT-conditions, the reduced KKT-conditions can be considered as an implicit function $F : \mathbb{R}^{n_\xi} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_\xi}$

$$F(\xi^*(p), p) = \begin{pmatrix} \nabla_{z^*} \mathcal{L}(z^*, \lambda^*, \nu^*, p) \\ h(z^*, p) \\ \lambda^* \odot g(z^*, p) \end{pmatrix} = 0. \tag{13}$$

The $\odot$ operator denotes the Hadamard product.

Via implicit differentiation of (13) [14], the first order sensitivity matrix $\nabla_p \xi^*(p) \in \mathbb{R}^{n_\xi \times n_p}$ of the primal-dual solution with respect to the parameters can be obtained by solving the linear system of equations

$$\nabla_{\xi^*} F \, \nabla_p \xi^* = -\nabla_p F. \tag{14}$$

The coefficient matrix $\nabla_{\xi*} F \in \mathbb{R}^{n_\xi \times n_\xi}$ and the right hand side matrix $\nabla_p F \in \mathbb{R}^{n_\xi \times n_p}$ are the Jacobians of the reduced KKT-conditions F with respect to the optimal primal-dual solution ξ^* and the parameters p of (11) respectively.

B. Second Order Sensitivities of Nonlinear Programs

The second order sensitivity tensor $\nabla^2_p \xi^* \in \mathbb{R}^{n_p \times n_p \times n_\xi}$ of (11) can be computed by consideration of the differentiated KKT conditions (14) as an implicit function $\tilde{F} : \mathbb{R}^{n_\xi} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_\xi \times n_p}$

$$\tilde{F}(\xi^*(p), p) = \nabla_{\xi^*} F \, \nabla_p \xi^* + \nabla_p F = 0. \tag{15}$$

We define the matrix formulation $S \in \mathbb{R}^{n_\xi \times n_p^2}$ of the second order sensitivity tensor $\nabla^2_p \xi^*$ as

$$S = \begin{bmatrix} \frac{\partial^2 \xi^*}{\partial p_1 \partial p} & \dots & \frac{\partial^2 \xi^*}{\partial p_{n_p} \partial p} \end{bmatrix}. \tag{16}$$

In the following we contribute the expression for the matrix formulation S of the second order sensitivity tensor $\nabla^2_p \xi^*$.

Theorem 1 (Second Order Sensitivities):
Given the primal-dual solution $\xi^*(p)$ of (11) and the differentiated KKT conditions (15), the matrix formulation of the second order sensitivities S of (11) can be obtained by

$$\nabla_{\xi^*} F \, S = -C. \tag{17}$$

The right hand side block matrix $C \in \mathbb{R}^{n_\xi \times n_p^2}$ is composed of submatrices $C_j \in \mathbb{R}^{n_\xi \times n_p}$, with $j = 1, \dots, n_p$ given as

$$C = \begin{bmatrix} C_1 & \dots & C_{n_p} \end{bmatrix}, \tag{18a}$$
$$C_j = D_j + E_j \nabla_p \xi^*, \tag{18b}$$
$$D_j = \frac{\partial^2 F}{\partial p_j \partial p} + \begin{bmatrix} \frac{\partial^2 F}{\partial p_1 \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} & \dots & \frac{\partial^2 F}{\partial p_{n_p} \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} \end{bmatrix}, \tag{18c}$$
$$E_j = \frac{\partial^2 F}{\partial p_j \partial \xi^*} + \begin{bmatrix} \frac{\partial^2 F}{\partial \xi^*_1 \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} & \dots & \frac{\partial^2 F}{\partial \xi^*_{n_\xi} \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} \end{bmatrix}. \tag{18d}$$

Proof: See appendix. ∎

The second order sensitivities can therefore be obtained by solving the linear system of equations (17). The matrix $\nabla_{\xi^*} F$ is the same as in (14). The block matrix C depends on the first order sensitivities $\nabla_p \xi^*$, which must be obtained first, and mixed derivatives of F with respect to p and ξ^*.

To compute the approximation of the deterministic policy Hessian (10), the general optimization problem (11) must be cast into (4) with $z^* = \mathbf{u}^*$ and $p = \theta$. The relevant sensitivities $\nabla_\theta \pi = \nabla_\theta u_0^*$ and $\nabla^2_\theta \pi = \nabla^2_\theta u_0^*$ can be extracted from the relevant row of $\nabla_p \xi^*$ or from the relevant blocks of $\nabla^2_\theta \xi^*$, which are computed via (14) and (17).

C. Q-Function Approximation

To compute $\nabla_\theta J(\theta)$ and $H(\theta)$ as in (7) and (10), the action-value function $Q^\pi(s,a)$ must be approximated, e.g. using a generic function approximator $Q_v(s,a) \approx Q^\pi(s,a)$ with parameters $v \in \mathbb{R}^{n_v}$. The episodic setting of the proposed policy optimization algorithm reduces the approximation task to a supervised learning task.

We propose to build an approximation of the Q-function by first learning the stage cost $\hat{Q}^\pi_0(s,a)$ and then improving step by step by taking k-step look-aheads $\hat{Q}^\pi_k(s,a)$ based on

the previous approximation $Q_{v_{k-1}}$. The k-step look-ahead estimation $\hat{Q}_k^\pi(s,a)$ of $Q^\pi(s,a)$ is recursively defined as

$$\hat{Q}_k^\pi(s,a) = \begin{cases} \ell(s,a) & \text{if } k=0, \\ \ell(s,a) + \gamma Q_{v_{k-1}}(s',a') & \text{else.} \end{cases} \tag{19}$$

Let $\mathcal{R}$ be a so called replay buffer of length $n_\mathcal{R} \in \mathbb{N}$, gathering the last $n_\mathcal{R}$ encountered transitions $\langle s,a,\ell,s'\rangle$ using an arbitrary exploration policy π_{exp}

$$\mathcal{R} = \left\{ \langle s,a,\ell,s' \rangle^{(i)} \right\}_{i=1}^{n_\mathcal{R}}, \tag{20}$$

then we can define the set $\mathcal{M}$ of encountered transitions $\langle s,a,\ell,s'\rangle$ and suggested actions $a' = \pi(s')$ as

$$\mathcal{M} = \{ \langle s,a,\ell,s',a' \rangle \,|\, \langle s,a,\ell,s' \rangle \in \mathcal{R} \}. \tag{21}$$

The parameter v_k can then be obtained by the solution of

$$\min_{v_k} \quad \mathbb{E}_{\langle s,a,\ell,s',a' \rangle \sim \mathcal{M}} \left[\Psi \left(\hat{Q}_k^\pi(s,a), Q_{v_k}(s,a) \right) \right]. \tag{22}$$

The function $\Psi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ can be any suitable function for regression tasks, e.g. mean squared error. The solution process then alternates between label generation (19) and parameter regression (22). The process is repeated until the desired horizon $N_Q \in \mathbb{N}$ is reached. The choice of N_Q is a trade-off between approximation accuracy and computational cost. The steps are summarized in Algorithm 1.

Algorithm 1 Q-Function approximation

Require: $\mathcal{R}, \pi(s), N_Q$
  $\mathcal{M} \leftarrow \emptyset$
  for all $\langle s,a,\ell,s' \rangle \in \mathcal{R}$ **do**
    $a' \leftarrow \pi(s')$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{ \langle s,a,\ell,s',a' \rangle \}$
  end for
  for $k = 0, \ldots, N_Q$ **do**
    Compute $\hat{Q}_k^\pi(s,a)$ with (19) on $\mathcal{M}$
    Learn $Q_{v_k}(s,a)$ by solving (22)
  end for

IV. A Trust-Region Quasi-Newton Policy Optimization Algorithm

A rigorous choice of the learning rate α or a restriction of the maximum update step length can improve the stability of iterative optimization algorithms and can reduce the number of iterations until convergence. Two common approaches are line search and trust-region methods. It turns out that line search methods cannot be used properly in reinforcement learning because the objective function $J(\theta)$ is unknown. In contrast to that, trust-region methods can adapt the maximum step length based on measurements of the closed-loop cost of each episode only.

The proposed trust-region Quasi-Newton policy optimization algorithm consists three steps below: 1) Sampling of closed-loop trajectories, 2) Trust region update, 3) MPC parameter update. These steps are repeated until convergence to a stationary point $\|\nabla_\theta J(\theta)\|_2 \leq \epsilon$ with $\epsilon > 0$.

1) Sampling: The objective is to minimize $J(\theta)$. For a Quasi-Newton update step (9), $\nabla_\theta J(\theta)|_{\theta=\theta_j}$ and $H(\theta_j)$ must be known for the current policy π_{θ_j}. All three terms require to take expected values over a distribution of initial conditions S_0 as shown in (1), (7) and (10). To take the expected value over the initial conditions, a fixed set of initial conditions $\mathcal{S}_0 = \{ s_0^{(i)} | s_0^{(i)} \sim S_0 \}_{i=0}^{N_{\mathcal{S}_0}}$ is defined, which will be used to evaluate the closed-loop cost in step 2. For each initial condition in $\mathcal{S}_0$ a full trajectory of length N_{ep} is conducted with an exploration policy $\pi_{\theta_j,\text{exp}}(s)$. All observed tuples $\langle s,a,\ell,s' \rangle$ are stored in a replay buffer $\mathcal{R}$ of length $n_\mathcal{R} \in \mathbb{N}$. Also, the measured cumulative cost $V^{\pi_{\theta_j}}(s_0)$ is added to the set $\mathcal{J}_j$.

2) Trust-Region radius update: The trust-region radius $\delta_j > 0$ limits the maximum length of the update step $\|\Delta\theta_j\|_2$. If the observed closed-loop cost $J(\theta_j)$ is close to the predicted closed-loop cost $q(\theta_j)$, the prediction can be trusted, hence δ_j can be increased, and vice versa. The ratio ρ_j measures the agreement of the exact closed-loop cost function $J(\theta)$ and the closed-loop cost model $q(\theta) \approx J(\theta)$

$$\rho_j = \frac{J(\theta_{j-1}) - J(\theta_j)}{J(\theta_{j-1}) - q(\theta_j)}. \tag{23}$$

The better the model fits the observation, the closer the ratio gets to one. The trust-region radius is then updated depending on the observed value of ρ_j as commonly done in optimization algorithms [13].

3) Update of parameters: To update the parameters θ, the closed-loop cost is approximated as $q(\theta)$. The approximate second order Taylor expansion of $J(\theta)$ around θ_j reads as

$$q(\theta_j + \Delta\theta_j) = J(\theta_j) + \Delta\theta_j^\top \nabla_\theta J(\theta)|_{\theta=\theta_j} + \ldots \frac{1}{2} \Delta\theta_j^\top H(\theta_j) \Delta\theta_j. \tag{24}$$

The update step $\Delta\theta_j$ at iteration j within the iterative optimization algorithm is then the solution of the trust-region constrained optimization problem

$$\Delta\theta_j = \arg\min_{\Delta\hat{\theta}_j} \quad q(\theta_j + \Delta\hat{\theta}_j) \tag{25a}$$

$$\text{s.t.} \quad \|\Delta\hat{\theta}_j\|_2 \leq \delta_j. \tag{25b}$$

Since $\nabla_\theta J(\theta)|_{\theta=\theta_j}$ and $H(\theta_j)$ require $\nabla_\theta \pi_{\theta_j}(s)$, $\nabla_\theta^2 \pi_{\theta_j}(s)$, $\nabla_a Q^{\pi_{\theta_j}}(s,a)$ and $\nabla_a^2 Q^{\pi_{\theta_j}}(s,a)$, all these must be computed for all items in the replay buffer $\mathcal{R}$. First, $Q^{\pi_{\theta_j}}(s,a)$ is approximated by $Q_v(s,a)$ based on Algorithm 1 using $\mathcal{R}$. Then, the policy's action $a_\pi = \pi_{\theta_j}(s)$ together with $\nabla_\theta \pi_\theta(s)$ and $\nabla_\theta^2 \pi_{\theta_j}(s)$ are computed according to (14) and (17) for all states s in $\mathcal{R}$. Lastly, $\nabla_a Q^{\pi_{\theta_j}}(s,a)$ and $\nabla_a^2 Q^{\pi_{\theta_j}}(s,a)$ are computed for all s and their related a_π. Once all subterms are gathered, $\nabla_\theta J(\theta)|_{\theta=\theta_j}$ and $H(\theta_j)$ can be calculated with (7) and (10). The update $\Delta\theta_j$ is then obtained from (25).

These steps only have to be applied if the proposed update $\Delta\theta_{j-1}$ improves the closed-loop cost that is $\rho_j > 0$. Otherwise, if $\rho_j < 0$, the update is reverted, so $\theta_{j-1} \leftarrow \theta_{j-1} - \Delta\theta_{j-1}$, such that the old values of $\nabla_\theta J(\theta)|_{\theta=\theta_{j-1}}$ and $H(\theta_{j-1})$ can be reused in (25) but with a smaller trust-region radius $\delta_j < \delta_{j-1}$. Algorithm 2 summarizes all steps.

Algorithm 2 Trust-Region Quasi-Newton Iteration

Require: Empty replay buffer $\mathcal{R}$ of length $n_{\mathcal{R}}$
Require: Trust-Region parameters: $\delta_0, \epsilon, \rho_0 > 0$
Require: Policy parameters: θ_0, NN parameters: v_0
  $j \leftarrow 0$
  while $j = 0$ **or** $\left\|\nabla_\theta J(\theta)|_{\theta=\theta_{j-1}}\right\|_2 > \epsilon$ **do**
    $\mathcal{J}_j \leftarrow \emptyset$ ▷ Sampling
    for all $s_0 \in \mathcal{S}_0$ **do**
      Sample full trajectory for s_0 with π_{exp}
      Store all $\langle s, a, \ell, s' \rangle$ in $\mathcal{R}$
      $\mathcal{J}_j \leftarrow \mathcal{J}_j \cup \{V^{\pi_{\theta_j}}(s_0)\}$
    end for
    Compute mean $J(\theta_j)$ over $\mathcal{J}_j$ ▷ Trust-Region update
    if $j > 0$ **then**
      Compute ρ_j with (23) using $J(\theta_j)$ and $J(\theta_{j-1})$
      Update δ_j based on ρ_j
    end if
    if $\rho_j > 0$ **then** ▷ Update computation
      Train $Q_v(s,a)$ with Algorithm 1 using $\mathcal{R}$
      for all $\langle s, a, \ell, s' \rangle \in \mathcal{R}$ **do**
        $a_\pi \leftarrow \pi_{\theta_j}(s)$
        Compute $\nabla_\theta \pi_{\theta_j}(s)$ with (14)
        Compute $\nabla^2_\theta \pi_{\theta_j}(s)$ with (17)
        Get $Q_v(s, a_\pi), \nabla_a Q_v(s, a_\pi), \nabla^2_a Q_v(s, a_\pi)$
      end for
      Get $\nabla_\theta J(\theta)|_{\theta=\theta_j}$ from (7)
      Get $H(\theta_j)$ from (10)
    else
      $\theta_{j-1} \leftarrow \theta_{j-1} - \Delta\theta_{j-1}$
      $\nabla_\theta J(\theta)|_{\theta=\theta_j} \leftarrow \nabla_\theta J(\theta)|_{\theta=\theta_{j-1}}$
      $H(\theta_j) \leftarrow H(\theta_{j-1})$
    end if
    Get $\Delta\theta_j$ from (25)
    $\theta_j \leftarrow \theta_{j-1} + \Delta\theta_j$
    $j \leftarrow j + 1$
  end while

V. Case Study

We consider a two dimensional linear system model to demonstrate the performance of the proposed algorithm

$$s' = \begin{pmatrix} 0.9 & 0.35 \\ 0 & 1.1 \end{pmatrix} s + \begin{pmatrix} 0.0813 \\ 0.2 \end{pmatrix} a. \tag{26}$$

The control goal is to regulate the states and actions to the origin, while not violating the constraints

$$h(s,a) = \begin{pmatrix} s_{\text{lb}} - s \\ s - s_{\text{ub}} \\ a_{\text{lb}} - a \\ a - a_{\text{ub}} \end{pmatrix} \leq 0, \tag{27}$$

with $s_{\text{lb}} = (0, -1)^\top$, $s_{\text{ub}} = (1, 1)^\top$, $a_{\text{lb}} = -1$ and $a_{\text{ub}} = 1$.

The stage cost $\ell(s,a)$ penalizes the deviation from the origin and constraint violations

$$\ell(s,a) = s^\top s + \frac{1}{2} a^\top a + 100^\top \max\{0, h(s,a)\}. \tag{28}$$

The $\max(\cdot)$ operator is applied elementwise to each row of the vectors.

The agent with MPC structure (4) is constructed using

$$\ell_\theta(x_k, u_k) = x_k^\top x_k + \frac{1}{2} u_k^\top u_k, \quad w = w_{\text{f}} = 100, \tag{29a}$$

$$V_{\text{f},\theta}(x_N) = x_N^\top \begin{pmatrix} 5.7 & 1.3 \\ 1.3 & 3.3 \end{pmatrix} x_N, \quad \gamma = 1, \tag{29b}$$

$$\hat{f}_\theta(x_k, u_k) = \begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{pmatrix} x_k + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} u_k + \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}, \tag{29c}$$

$$h_\theta(x_k, u_k) = \begin{pmatrix} s_{1,\text{lb}} + \Delta x_1 - x_{1,k} \\ s_{2,\text{lb}} - x_{2,k} \\ x_k - s_{\text{ub}} \\ a_{\text{lb}} - u_k \\ u_k - a_{\text{ub}} \end{pmatrix}, \tag{29d}$$

$$h_{\text{f},\theta}(x_N) = h_{\theta,x}(x_N), \quad N = 10. \tag{29e}$$

The resulting parameter vector θ is defined as

$$\theta = (a_{11}, a_{12}, a_{22}, b_1, b_2, d_1, d_2, \Delta x_1)^\top, \tag{30}$$

with initial values $\theta_0 = (1, 0.25, 1, 0.1, 0.3, 0, 0, 0)^\top$.

The Q-function is approximated using a feed-forward NN with two hidden layers with 20 neurons each and tanh-activation function. The inputs $(s^\top, a^\top)^\top$ and labels $\hat{Q}^\pi_k(s,a)$ are all scaled using custom scalers. Note that the scaling also affects $\nabla_a Q_v(s,a)$ and $\nabla^2_a Q_v(s,a)$, which has to be taken into account. The horizon of the Q-function is set as $N_Q = 10$, which is a trade-off between computational complexity and accuracy. The Huber loss function [15] is used in (22) together with the Adam optimizer [16].

Algorithm 2 is initialized with the values given in Table I. The trust-region radius update follows the suggestion in [13] with $\delta_{\max}$ denoting the maximum allowed stepsize.

TABLE I
Hyperparameters of proposed Algorithm 2.

Parameter	Value	Parameter	Value
n_{IC}	50	N_{ep}	50
ϵ	10^{-6}	$n_{\mathcal{R}}$	250
δ_0	10^{-2}	$\delta_{\max}$	10^{-1}

Algorithm 2 is compared to three training algorithms:

1) First order updates without trust region (6)
2) First order updates with trust region
3) Second order updates without trust region (9)

All agents use the same initial conditions, but vary in their hyperparameter settings. In case 1), the learning rate is set to $\alpha = 10^{-4}$, which compromises stability and convergence speed. In case 2), the agent is initialized with a trust-region radius of $\delta_0 = 10^{-3}$ and a maximum trust-region radius of $\delta_{\max} = 10^{-1}$. In case 3), the learning rate is set to $\alpha = 10^{-2}$, which is the largest possible investigated learning rate without losing stability of the training process. All methods are compared to a benchmark MPC, which uses the exact model (26) and a prediction horizon of $N = 50$, and the untrained MPC using $\theta = \theta_0$.

Figure 1 shows the decrease of $J(\theta_j)$ over the reinforcement learning iterations j for all training algorithms and compares them to the benchmark MPC. It can be seen that the proposed algorithm (right subfigure, solid line) outperforms all other methods with respect to convergence speed as it needs less than 20 iterations to converge to the performance of the benchmark MPC while the others still keep decreasing. Also, less oscillations are observed during the training process, which suggests a higher stability during training.

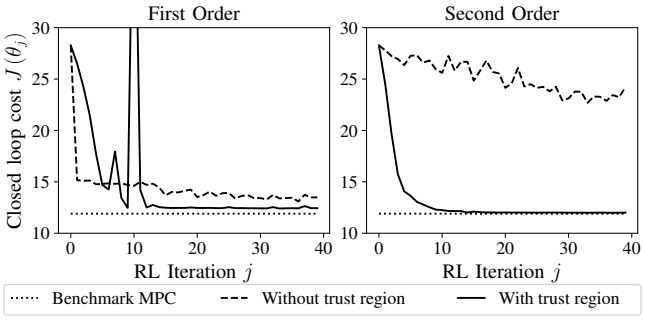

Fig. 1. Evolution of the closed-loop cost $J(\theta_j)$ over the reinforcement learning (RL) iterations j. The plots show the results for first order training (left) and second order training (right) with and without a trust region.

The differently trained MPC agents are evaluated on closed-loop simulations of a test set. The test set is created by taking 2,500 randomly distributed initial states in the feasible state space and performing an episode using the benchmark MPC. All initial conditions which encounter any infeasible point in their closed-loop trajectory are discarded. The final test set consists of $n_{\mathrm{T}} = 1,579$ initial conditions with a total of $n_{\mathrm{P}} = 78,950$ points. The performance measures are the number of infeasible trajectories $n_{\mathrm{T,if}}$, the closed-loop cost on all n_{T} trajectories J, the number of infeasible points $n_{\mathrm{P,if}}$, the maximum constraint violations $\mathrm{CV}_{\max}$, and the average constraint violation $\overline{\mathrm{CV}}$ on the set of infeasible points. The number of infeasible points $n_{\mathrm{P,if}}$ is the portion of all points n_{P} for which the MPC agent controls the system into the infeasible state space. A trajectory is then infeasible and added to the number of infeasible trajectories $n_{\mathrm{T,if}}$ if any point of the closed-loop trajectory is an infeasible point.

The results are summarized in Table II. The agent trained with the proposed approach (2^{nd} order (TR)), is feasible on all initial conditions and outperforms all other trained agents with respect to the obtained closed-loop cost, which is almost the closed-loop cost obtained by the benchmark MPC.

We want to emphasize that each update step is performed offline and does not influence the online solution time of the applied MPC controller. Also, the offline computation time of each update step of the proposed trust-region constrained Quasi-Newton updates is observed to be in the same order of magnitude as the established first order updates.

All implementations were done in Python, using CasADi [17], do-mpc [18], Ipopt [19], and Tensorflow [20].

TABLE II

PERFORMANCE WITH RESPECT TO THE CLOSED-LOOP COST J, NUMBER OF INFEASIBLE TRAJECTORIES $n_{\mathrm{T,if}}$ AND POINTS $n_{\mathrm{P,if}}$ AS WELL AS MAXIMUM AND AVERAGE CONSTRAINT VIOLATION $\mathrm{CV}_{\max}$, $\overline{\mathrm{CV}}$.

	J	$n_{\mathrm{T,if}}$	$n_{\mathrm{P,if}}$	$\mathrm{CV}_{\max}$	$\overline{\mathrm{CV}}$
Benchmark	3.61	0	0	0	–
Untrained	10.66	1579	12,001	18.6	0.86
1^{st} order	4.34	0	0	0	–
1^{st} order (TR)	3.88	0	0	0	–
2^{nd} order	8.89	1539	4,876	15.6	1.57
2^{nd} order (TR)	**3.64**	0	0	0	–

The code to reproduce the results is available online[1].

VI. CONCLUSION

In this work, we propose a trust-region Quasi-Newton policy optimization algorithm for episodic reinforcement learning using a parameterized MPC as a policy approximator. We show that the computation of the second order sensitivity tensor for nonlinear programs boils down to the solution of a linear system of equations. We apply the proposed algorithm to an example system and show empirically that the proposed algorithm outperforms other investigated algorithms with respect to the data efficiency and also with respect to the achieved control performance of the learned policy.

Future work will investigate how the method scales to larger and potentially nonlinear systems. Also, different options to approximate the Q-function such as the MPC scheme itself as well as a direct comparison of the proposed method with established state-of-the-art reinforcement learning algorithms will be investigated.

REFERENCES

[1] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Santa Barbara, California: Nob Hill Publishing, 2nd ed., 2020.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series, Cambridge, Massachusetts: The MIT Press, 2nd ed., 2018.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[4] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, PMLR, June 2014.

[5] T. Furmston, G. Lever, and D. Barber, "Approximate Newton Methods for Policy Search in Markov Decision Processes," *Journal of Machine Learning Research*, vol. 17, no. 226, pp. 1–51, 2016.

[6] S. M. Kakade, "A natural policy gradient," in *Advances in Neural Information Processing Systems* (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, MIT Press, 2001.

[7] J. Andrew Bagnell and J. Schneider, "Covariant Policy Search," in *International Joint Conference on Artificial Intelligence*, p. 142282 Bytes, Carnegie Mellon University, 2003.

[1] `https://github.com/DeanBrandner/ECC24_TR_improved_QN_PO_for_MPC_in_RL`

[8] D. K. Jha, A. U. Raghunathan, and D. Romeres, "Quasi-newton trust region policy optimization," in *Proceedings of the Conference on Robot Learning* (L. P. Kaelbling, D. Kragic, and K. Sugiura, eds.), vol. 100 of *Proceedings of Machine Learning Research*, pp. 945–954, PMLR, 2020-10-30/2020-11-01.

[9] A. B. Kordabad, H. Nejatbakhsh Esfahani, W. Cai, and S. Gros, "Quasi-Newton Iteration in Deterministic Policy Gradient," in *2022 American Control Conference (ACC)*, (Atlanta, GA, USA), pp. 2124–2129, IEEE, June 2022.

[10] S. Gros and M. Zanon, "Data-Driven Economic NMPC Using Reinforcement Learning," *IEEE Transactions on Automatic Control*, vol. 65, pp. 636–648, Feb. 2020.

[11] A. B. Kordabad, D. Reinhardt, A. S. Anand, and S. Gros, "Reinforcement Learning for MPC: Fundamentals and Current Challenges," *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 5773–5780, 2023.

[12] D. Brandner, T. Talis, E. Esche, J.-U. Repke, and S. Lucia, "Reinforcement learning combined with model predictive control to optimally operate a flash separation unit," in *Computer Aided Chemical Engineering*, vol. 52, pp. 595–600, Elsevier, 2023.

[13] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer Series in Operations Research, New York: Springer, 2nd ed., 2006.

[14] A. V. Fiacco and Y. Ishizuka, "Sensitivity and stability analysis for nonlinear programming," *Annals of Operations Research*, vol. 27, no. 1, pp. 215–235, 1990.

[15] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, pp. 73–101, Mar. 1964.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[17] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[18] F. Fiedler, B. Karg, L. Lüken, D. Brandner, M. Heinlein, F. Brabender, and S. Lucia, "Do-mpc: Towards FAIR nonlinear and robust model predictive control," *Control Engineering Practice*, vol. 140, p. 105676, Nov. 2023.

[19] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, Mar. 2006.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.

[21] I. N. Bronštejn, K. A. Semendjaev, G. Musiol, and H. Mühlig, *Taschenbuch der Mathematik*. Edition Harri Deutsch, Haan-Gruiten: Verlag Europa-Lehrmittel - Nourney, Vollmer GmbH & Co. KG, 10th ed., 2016.

APPENDIX
PROOF OF THEOREM 1

Proof: Consider the j-th column vector $\tilde{F}_j(\xi^*(p), p) \in \mathbb{R}^{n_\xi}$ of the implicit matrix function $\tilde{F}(\xi^*(p), p)$ defined in (15), then implicit differentiation results in

$$\frac{\partial \tilde{F}_j}{\partial \xi^*} \nabla_p \xi^* + \frac{\partial \tilde{F}_j}{\partial p} = 0. \tag{31}$$

Lets define matrix E_j from (18d) as $E_j = \frac{\partial \tilde{F}_j}{\partial \xi^*}$

$$E_j = \frac{\partial \tilde{F}_j}{\partial \xi^*} = \frac{\partial}{\partial \xi^*} \left(\frac{\partial F}{\partial \xi^*} \frac{\partial \xi^*}{\partial p_j} + \frac{\partial F}{\partial p_j} \right). \tag{32}$$

Application of the differentiation operator to each summand and using the interchangeability of partial derivatives [21]

$$E_j = \frac{\partial}{\partial \xi^*} \left(\frac{\partial F}{\partial \xi^*} \frac{\partial \xi^*}{\partial p_j} \right) + \frac{\partial^2 F}{\partial p_j \partial \xi^*}. \tag{33}$$

Application of the chain rule [21] gives

$$E_j = \begin{bmatrix} \left(\frac{\partial^2 F}{\partial \xi_1^* \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} + \frac{\partial F}{\partial \xi^*} \frac{\partial^2 \xi^*}{\partial \xi_1^* \partial p_j} \right)^\top \\ \vdots \\ \left(\frac{\partial^2 F}{\partial \xi_{n_\xi}^* \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} + \frac{\partial F}{\partial \xi^*} \frac{\partial^2 \xi^*}{\partial \xi_{n_\xi}^* \partial p_j} \right)^\top \end{bmatrix}^\top + \frac{\partial^2 F}{\partial p_j \partial \xi^*}, \tag{34}$$

which can be simplified by looking at the right summand of each matrix entry. The Jacobian is derived to be $\frac{\partial \xi^*}{\partial \xi^*} = I$, leading to 0 when the derivative with respect to p_j is applied. The simplified expression then reads as

$$E_j = \frac{\partial^2 F}{\partial p_j \partial p} + \begin{bmatrix} \frac{\partial^2 F}{\partial \xi_1^* \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} & \cdots & \frac{\partial^2 F}{\partial \xi_{n_\xi}^* \partial \xi^*} \frac{\partial \xi^*}{\partial p_j} \end{bmatrix}. \tag{35}$$

The procedure is also conducted for $\frac{\partial \tilde{F}_j}{\partial p}$ until (34) giving

$$\frac{\partial \tilde{F}_j}{\partial p} = \begin{bmatrix} \left(\frac{\partial^2 F}{\partial p_1 \partial p} \frac{\partial \xi^*}{\partial p_j} + \frac{\partial F}{\partial \xi^*} \frac{\partial^2 \xi^*}{\partial p_j \partial p_1} \right)^\top \\ \vdots \\ \left(\frac{\partial^2 F}{\partial p_{n_p} \partial p} \frac{\partial \xi^*}{\partial p_j} + \frac{\partial F}{\partial \xi^*} \frac{\partial^2 \xi^*}{\partial p_j \partial p_{n_p}} \right)^\top \end{bmatrix}^\top + \frac{\partial^2 F}{\partial p_j \partial p}. \tag{36}$$

The right summands of the matrix entries do not vanish

$$\frac{\partial \tilde{F}_j}{\partial p} = \frac{\partial F}{\partial \xi^*} \frac{\partial^2 \xi^*}{\partial p_j \partial p} + D_j, \tag{37a}$$

$$\text{with} \quad D_j = \frac{\partial^2 F}{\partial p_j \partial p} + \begin{bmatrix} \frac{\partial^2 F}{\partial p_1 \partial p} \frac{\partial \xi^*}{\partial p_j} & \cdots & \frac{\partial^2 F}{\partial p_{n_p} \partial p} \frac{\partial \xi^*}{\partial p_j} \end{bmatrix}. \tag{37b}$$

Plugging (35) and (37) into (31) then delivers

$$E_j \nabla_p \xi^* + D_j + \frac{\partial F}{\partial \xi^*} \frac{\partial^2 \xi^*}{\partial p_j \partial p} = 0. \tag{38}$$

Rearranging the equation as a linear system of equations for the j-th slice of the second order sensitivity tensor leads to

$$\nabla_{\xi^*} F \frac{\partial^2 \xi^*}{\partial p_j \partial p} = -C_j, \tag{39a}$$

$$\text{with} \quad C_j = D_j + E_j \nabla_p \xi^*. \tag{39b}$$

The matrix $\nabla_{\xi^*} F$ is equal for all slices of the second order sensitivity tensor. Hence, according to (16), the matrix slices $\frac{\partial^2 \xi^*}{\partial p_j \partial p}$ can be stacked into a matrix representation S of the second order sensitivity tensor. The same is done for the right-hand-side matrices C_j according to (18b). All this combined leads to

$$\nabla_{\xi^*} F \, S = -C. \tag{40}$$

■