Please cite: M. Ławryńczuk, R. Nebeluk: Beyond the quadratic norm: Computationally efficient constrained nonlinear MPC using a custom cost function, ISA Transactions, vol. 134, pp. 336-356, 2023.

Beyond the Quadratic Norm: Computationally Efficient Constrained Nonlinear MPC Using a Custom Cost Function

Maciej Ławryńczuk and Robert Nebeluk
Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Control and Computation Engineering
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
Maciej.Lawrynczuk@pw.edu.pl, Robert.Nebeluk@pw.edu.pl

Abstract: A new approach to nonlinear Model Predictive Control (MPC) is discussed in this work. A custom user-defined cost function is used in place of the typically considered quadratic norm. An approximator of the cost function is applied to obtain a computationally simple procedure and linearization of two trajectories is carried out online. The predicted output trajectory of the approximator and the predicted trajectory of the manipulated variable, both over the prediction horizon, are repeatedly linearized online. It yields a simple quadratic programming task. The algorithm is implemented for a simulated neutralization benchmark modeled by a neural Wiener model. The resulting control quality is excellent, identical to that observed in the MPC scheme with nonlinear optimization. Validity of the described MPC algorithms is demonstrated when only simple box constraints are considered on the process input variable and in a more demanding case when additional soft limitations are put on the predicted output. Two structures of the approximator are compared: polynomial and neural; the advantages of the latter one are shown and stressed.

Keywords: Model Predictive Control, Cost function, Neutralization reactor, Neural networks.

1 Introduction

Model Predictive Control (MPC) algorithms compute online the control policy from an optimization problem the objective of which is to describe the predicted quality of control [24, 34]. Typically, the squared predicted control errors are considered over some time horizon. Such a classical approach works efficiently for numerous processes, example applications of MPC are: chemical reactors [25, 30], evaporators [29], distillation columns [33] and boiler–turbine systems [18]. MPC methods have been also utilized for fast systems, e.g. drones [38], robotic manipulators [16], carrier-based aircrafts [35],

open-loop unstable mechanical systems [32], electro-mechanical systems [15], inverter-based electrical drives [12] and even stochastic systems [1]. Recently, the classical formulation of MPC has been extended. For example, MPC algorithms may control networked systems affected by Denial-of-Service (DoS) attacks [4]. An interesting idea is cooperation of MPC with the sliding mode control method [8]. Such an approach can be further extended by imposing a contractive Lyapunov condition to guarantee the closed-loop stability [7].

Although the classical sum of squared predicted discrepancies between the setpoint and the predicted process output, i.e., the predicted control errors (the L_2 cost function), is successfully used very frequently, some alternatives are also reported. Firstly, a penalty term may be added to the cost function. Such modifications are necessary for MPC algorithms with the infinite prediction horizon; the penalty term approximates the predictions made from the end of the finite horizon to infinity [27]. Other penalty terms are motivated by economic reasons. The MPC algorithms presented in [3, 22, 39] measure the economic cost in the penalty term of MPC; the additional economic penalty term is also minimized during online control. In all cited solutions, the predicted control errors are measured in the MPC cost function in the simplest way, i.e., the L_2 cost function is used. Such a choice has one essential advantage because a quadratic optimization problem is obtained assuming a linear (or successively linearized online) process model.

In some approaches, however, the L_2 cost function is not utilized to assess the impact of the predicted control errors in MPC. The most frequent choice is the sum of absolute values of predicted control errors (the L_1 cost function). Provided that a linear model is used, a simple to solve linear optimization task may be derived [6, 17]. When a nonlinear model is necessary, online nonlinear programming repeated at each algorithm execution is used [5, 9] or online model linearization may be employed to reduce the computational complexity [21]; an alternative is to use the trust-region sequential quadratic programming method [2]. As pointed out in [5], the MPC-L_1 approach is expected to give better control quality than the classical MPC-L_2 one and may be beneficial for stability [26]. In addition to the L_1 cost function, the maximum predicted control error (the L_∞ norm) might also be utilized [23]. If a powerful optimization solver is available, in theory, we can use different nonlinear cost functions in MPC. Unfortunately, in the case of a nonlinear cost function and a nonlinear dynamical model used for prediction, we obtain a complex nonlinear optimization task that has to be solved at each algorithm's execution.

This work presents an MPC algorithm in which the predicted control errors are measured using a user-defined, custom cost function. The function may generally be defined analytically or in a graphical form. A differentiable approximator approximates the cost function. In general, different approximator structures are possible. In this work, two representations are considered: polynomial and neural ones. We study the influence of some imperfections of polynomials, even of a relatively high degree, on the control quality. The neural approximator has excellent accuracy and is recommended. The trajectory of the predicted controlled variable and the trajectory of the predicted control errors embedded in the custom cost function are linearized online along some trajectory of the future changes of the manipulated variable to get a computationally not demanding algorithm. As a result, a simple to solve quadratic programming task is derived. This work extends [21] in which the L_1 cost function is utilized

in linearization-based MPC. In this work, practically any cost function is possible since a differentiable approximator is used and the constraints imposed on the predicted process output are allowed. To guarantee the existence of the feasible set, soft constraints are used, which may be temporarily violated when necessary. The presented MPC methods are compared with the classical alternatives, i.e., the MPC scheme with nonlinear optimization (and the custom cost function) and the MPC scheme relying on the classical L_2 norm. Moreover, the influence of the cost function approximator on the resulting control quality is also studied. All things considered, the following features of the novel approach presented in this work must be pointed out:

- A custom cost function is used in nonlinear MPC, in which a nonlinear model is used for prediction, but a computationally simple calculation scheme is derived. Non-classical cost functions (L_1 and L_∞) have been used in MPC, but efficient implementations are possible if a linear model is used [6, 17, 23]. When a nonlinear model is considered, a nonlinear optimization task is solved on-line [5, 9]. As a result of an advanced trajectory linearization method, a relatively simple quadratic task is solved in our approach.
- The derived MPC optimization task has only one global solution.
- Feasibility of the optimization is guaranteed by soft constraints put on the controlled variable.
- The cost function used is practically not restricted. It is approximated by an approximator that is required to be differentiable. Different structures of approximators may be used, e.g., neural networks of different kinds.
- The number of optimized decision variables is the same as in the classical approach to MPC.
- Easiness of implementation.
- The structure of the dynamical model used is not limited provided that it is differentiable.

A preliminary, simplified version of one of our approaches was shortly described in the conference work [28] in which only simple box constraints on the manipulated variable were possible, trajectory linearization was performed in a simplified manner, computational efficiency was not studied and the influence of the approximator type on control quality was not discussed.

Efficacy of the new approach to nonlinear MPC is checked for a pH reactor process [10] which is a typical benchmark to validate nonlinear control methods. Two variants of the MPC algorithm are compared in simulations as only one or multiple trajectory linearizations at single sampling instants may be possible. High accuracy and efficacy in MPC of the neural approximator are compared with properties of polynomial approximators. A thorough comparison of a few configurations of the MPC algorithm with two custom cost functions is discussed; the results are compared with those possible when the classical squared norm is used. Finally, the satisfaction efficiency of nonlinear constraints related to the predicted process output is validated.

The outline of this work is the following. The problem is formulated in Section 2. Section 3 derives the computationally fast MPC algorithm with a custom cost function; in particular, the quadratic programming task with soft constraints put on the predicted controlled process output variable is

formulated. Efficacy of the discussed methods is thoroughly discussed in Section 4 which presents simulation results for a pH (neutralization) chemical reactor benchmark. Finally, concluding remarks are formulated in Section 5.

2 Preliminaries and Problem Definition

The symbol u is used throughout this work to denote the manipulated process input variable while the symbol y stands for the controlled process output, respectively. The decision vector computed at each execution of MPC, i.e., each sampling instant, is

$$\triangle \boldsymbol{u}(k) = [\triangle u(k|k) \dots \triangle u(k+N_{\mathrm{u}}-1|k)]^{\mathrm{T}} \tag{1}$$

where $\triangle u(k+p|k)$ stands for the change of the manipulated variable for the future sampling $k+p$ computed at the sampling k, i.e., $\triangle u(k+p|k) = u(k+p|k) - u(k+p-1|k)$; N_{u} denotes the control horizon. The decision vector is found from an optimization problem carried out online. The rudimentary constrained MPC optimization task is

$$\begin{aligned}
&\min_{\triangle \boldsymbol{u}(k)} \{J(k)\} \\
&\text{s. t.} \\
&u^{\min} \leq u(k+p|k) \leq u^{\max},\ p = 0, \dots, N_{\mathrm{u}}-1 \\
&\triangle u^{\min} \leq \triangle u(k+p|k) \leq \triangle u^{\max},\ p = 0, \dots, N_{\mathrm{u}}-1 \\
&y^{\min} \leq \hat{y}(k+p|k) \leq y^{\max},\ p = 1, \dots, N \\
&\hat{y}(k+p|k) = f_{\mathrm{model}}(k+p|k) + d(k),\ p = 1, \dots, N
\end{aligned} \tag{2}$$

where the acronym "s. t." stands for "subject to". In general, the limitations are put on manipulated and controlled variables. The scalars $u^{\min}$ and $u^{\max}$ define the range of the manipulated input variable, the scalars $\triangle u^{\min}$ and $\triangle u^{\max}$ define the restrictions related to the rate of change of the same variable, the quantities $y^{\min}$ and $y^{\max}$ specify the limits put on the predicted values of the process output. $\hat{y}(k+p|k)$ stands for the predicted value of the controlled variable for the sampling instant $k+p$ determined at the instant k. Since a dynamical model is used to calculate the predictions, the general prediction equation

$$\hat{y}(k+p|k) = f_{\mathrm{model}}(k+p|k) + d(k) \tag{3}$$

is explicitly taken into account as the last constraint over the prediction horizon, i.e., for all $p = 1, \dots, N$. $f_{\mathrm{model}}(k+p|k)$ denotes the model output for the sampling instant $k+p$ computed at the sampling k and $d(k)$ stands for an estimation of the unmeasured disturbance that affects the process output at the time k. The prediction of the controlled variable cannot be computed as the model output, i.e., $\hat{y}(k+p|k) = f_{\mathrm{model}}(k+p|k)$, because, in such a case, the MPC algorithm does not have the integral action which compensates for all model errors, measurement noise and actual process disturbances [34]. Typically, we assume that the disturbance estimation does not change over the prediction horizon [34]. It is found as the difference between the current measured process output and the model output. When the MPC optimization task is solved, only the first element of the optimal

vector (1), i.e., $\triangle u(k|k)$, is sent to the process and the computation procedure is executed at the following sampling instants.

Let us recall definition of the classical L_2-type MPC cost function

$$J_2(k) = \sum_{p=1}^{N} \left(y^{\mathrm{sp}}(k+p|k) - \hat{y}(k+p|k)\right)^2 + \lambda \sum_{p=0}^{N_\mathrm{u}-1} \left(\triangle u(k+p|k)\right)^2 \tag{4}$$

Square norms are used in both parts of the function $J_2(k)$. The first part of the cost function measures the squared future deviations between the setpoint and the predicted process output, i.e., the future control errors; they are considered over the prediction horizon N. The predictions $\hat{y}(k+p|k)$ are calculated from the general prediction equation (3). The symbol $y^{\mathrm{sp}}(k+p|k)$ stands for the setpoint value of the controlled variable for the sampling instant $k+p$ known at the current instant k. The prediction is computed from a dynamical model of the considered process. The second part of the function $J_2(k)$ measures the squared changes of the manipulated variable; $\lambda > 0$ is a penalty coefficient. The role of the penalty term is twofold. Firstly, it is used to reduce unfavorable abrupt and huge changes of the manipulated variable. Secondly, the quadratic penalty term leads to good numerical properties of the whole MPC optimization task (2).

An alternative, custom, user-defined cost function is used to measure the predicted control errors. The following form is postulated

$$J_\mathrm{c}(k) = \sum_{p=1}^{N} F(y^{\mathrm{sp}}(k+p|k) - \hat{y}(k+p|k)) + \lambda \sum_{p=0}^{N_\mathrm{u}-1} \left(\triangle u(k+p|k)\right)^2 \tag{5}$$

The custom function $F\colon \mathbb{R} \to \mathbb{R}$ may be defined analytically, employing an explicit mathematical formula, or in a graphical form. In both cases, a differentiable approximation of that function is used as it is explained next. The penalty term deliberately has the classical quadratic norm, i.e., to reduce unwanted moves of the process input variable and get good numerical properties.

The majority of applications use the quadratic L_2 cost function. Provided that linear models are used, the L_1 cost function [6, 17] and the L_∞ one [23] are possible. As pointed out in [5], the L_1 cost function gives better control quality compared with the typically used L_2 function. The algorithms presented in this work have two features: a custom cost function is used and prediction is computed from a nonlinear model, not a linear one. The role of a custom function is to weigh the predicted errors depending on the value of the error. For example, different costs may be used for negative and positive errors or different costs are possible for some specific errors.

Let us note that the number of decision variables of the optimization task (2) is always equal to N_u, regardless of the type of cost function used ((4) or (5)).

The process control diagram is sketched in Fig. 1. At each discrete sampling instant, for the given setpoint trajectory vector $\boldsymbol{y}^{\mathrm{sp}}(k)$, the optimization procedure calculates the set of increments $\triangle \boldsymbol{u}(k)$ that minimizes the cost function $J(k)$ subject to the existing constraints. The optimization procedure requires the predictions of the controlled variable, denoted as the vector $\hat{\boldsymbol{y}}(k)$, that correspond to the currently calculated decision vector $\triangle \boldsymbol{u}(k)$. A dynamical process model finds successively online the predicted trajectory. Some past measurements of process signals are necessary during optimization and prediction.

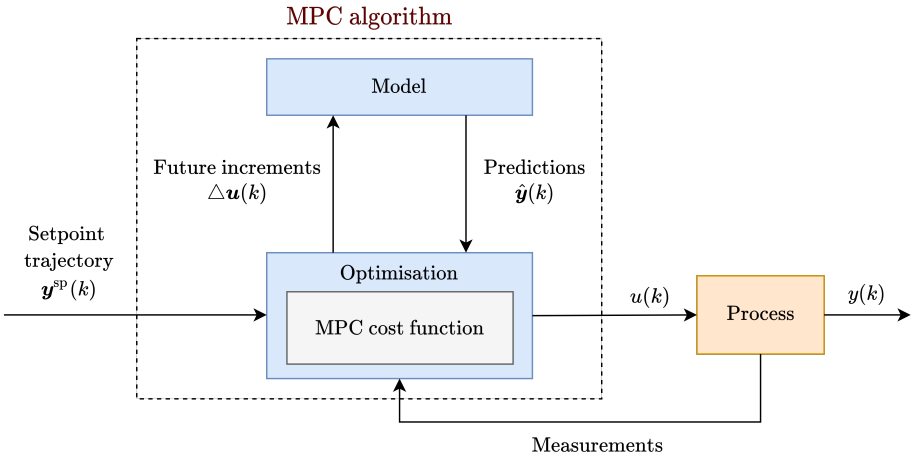

Figure 1: The process control diagram

3 Computationally Efficient MPC Using a Custom Cost Function

From the general MPC problem (2) and using the $\mathrm{L_c}$ cost function (5), the rudimentary MPC-$\mathrm{L_c}$ optimization task is

$$\begin{aligned}
\min_{\triangle \boldsymbol{u}(k)} \Bigg\{ J_{\mathrm{c}}(k) = & \sum_{p=1}^{N} F(y^{\mathrm{sp}}(k+p|k) - \hat{y}(k+p|k)) \\
& + \lambda \sum_{p=0}^{N_{\mathrm{u}}-1} (\triangle u(k+p|k))^2 \Bigg\} \\
\text{s. t.} & \\
& u^{\min} \leq u(k+p|k) \leq u^{\max}, \ p = 0, \ldots, N_{\mathrm{u}} - 1 \\
& \triangle u^{\min} \leq \triangle u(k+p|k) \leq \triangle u^{\max}, \ p = 0, \ldots, N_{\mathrm{u}} - 1 \\
& y^{\min} \leq \hat{y}(k+p|k) \leq y^{\max}, \ p = 1, \ldots, N \\
& \hat{y}(k+p|k) = f_{\mathrm{model}}(k+p|k) + d(k), \ p = 1, \ldots, N
\end{aligned} \tag{6}$$

This work assumes that the model used for prediction computation and the function F are nonlinear. That means that the first part of the custom cost function $J_{\mathrm{c}}(k)$ is nonlinear in terms of the MPC decision vector $\triangle \boldsymbol{u}(k)$. Due to model nonlinearity, the last two constraints are also nonlinear in terms of that vector. Furthermore, that also means that the rudimentary MPC-$\mathrm{L_c}$ optimization problem (6) is nonlinear. Hence, a nonlinear optimization procedure must be utilized at each sampling instant to calculate the decision vector. A computationally simpler alternative is explored next in which a sophisticated online linearization is carried out. As a result, a quadratic optimization task is obtained, with no need for nonlinear optimization. During derivation, we will remember that for linearization, differentiability is necessary. As we do not restrict the function F in any way, differentiability must be assured by the approximator of the rudimentary function F. Moreover, model differentiability is required.

3.1 Cost Function Approximator

In this work, it is postulated not to use directly the custom MPC cost function in the first part of the performance index (5) but its approximation. There are two reasons for that. Firstly, we will find a universal representation of different cost functions that is convenient for online use in MPC. Secondly, when the function F is given in a non-parametric graphical way, an approximator is a natural solution. We will only use approximators that assure differentiability. We require that the approximator satisfies the general equation

$$(\alpha(e(k+p|k)))^2 = F(e(k+p|k)) \tag{7}$$

for $p = 1, \ldots, N$, where

$$e(k+p|k) = y^{\mathrm{sp}}(k+p|k) - \hat{y}(k+p|k) \tag{8}$$

stands for the future control error for the time step $k+p$ computed at the current instant k. The predictions $\hat{y}(k+p|k)$ are determined from Eq. (3). The approximator's output for the sampling instant $k+p$ is $\alpha(e(k+p|k))$. Using Eqs. (5) and (7), the cost function $J_{\mathrm{c}}(k)$ becomes

$$J_{\mathrm{c}}(k) = \sum_{p=1}^{N} (\alpha(e(k+p|k)))^2 + \lambda \sum_{p=0}^{N_{\mathrm{u}}-1} (\triangle u(k+p|k))^2 \tag{9}$$

It is essential to explain why the first part of the cost function (9) has a particular form, i.e., the predicted control errors are valued by the function α first and next, the results are squared. It is because in both parts of the custom cost function, squared terms are used, as in the classical cost function $J_2(k)$ defined by Eq. (4). Let us remind that if the model utilized for prediction is linear with constant parameters or time-varying but linear in parameters, the use of the classical cost function $J_2(k)$ simplifies derivations as we can yield simple to solve quadratic programming tasks, e.g., [19, 20, 34]. A similar idea is used in this work, but in a sophisticated manner. In our case, the fundamental difficulty is that the control errors are measured with the nonlinear function α.

Two classes of differentiable approximators are defined and used in simulations presented next. A simple polynomial approximator is defined by

$$\alpha(e(k+p|k)) = w_0 + \sum_{i=1}^{n} w_i (e(k+p|k))^i \tag{10}$$

Coefficients of the polynomial are denoted by $w_0, \ldots, w_n$, n is the degree of the polynomial. We also use a neural approximator. For this purpose, the Multi-Layer Perceptron (MLP) neural network [13] is utilized. Such an approximator consists of two layers: the first one is nonlinear with K nodes, the second one is linear. The neural approximator is

$$\alpha(e(k+p|k)) = w_0^2 + \sum_{i=1}^{K} w_i^2 \tanh\left[w_{i,0}^1 + w_{i,1}^1 e(k+p|k)\right] \tag{11}$$

The network's weights are $w_{i,0}^1$, $w_{i,1}^1$, w_i^2 for $i = 1, \ldots, K$ and w_0^2. The tanh activation function is used in this work, which means that the neural approximator is differentiable. Although the network (11) uses the tanh activation function, other differentiable alternatives are possible. Polynomial and MLP neural approximators are described in this work; details of MPC algorithms for these two structures

are given. Both approximator structures are quite simple and they are utilized online in a fairly simple way. As it will be shown next, they offer very different approximation quality and resulting control quality. In addition to polynomial and neural approximators of the MLP type, we may use numerous other options such as Support Vector Machines or Radial Basis Function (RBF) neural networks.

The approximator is obtained from an identification procedure. For a given set of data points, i.e., the values of e and $F(e)$, the approximator's parameters are calculated. The type of the algorithm used for this purpose depends on the approximator's structure. For polynomial approximators, or, in general, for approximators which are linear-in-parameters functions of the error e, the simple least-squares problem is obtained, which is easily solved analytically. Similarly, for RBF approximators with fixed parameters of the basis functions, we also obtain least-squares problems. Conversely, training of MLP neural networks requires nonlinear optimization, which must be done numerically, not analytically. In this work, the Levenberg-Marquardt algorithm is used for training neural approximators.

3.2 Linearization of the Trajectory $\boldsymbol{\alpha}(k)$

We have to perform online linear approximations of two trajectories: the trajectory of the predicted control errors weighted by the function α (the trajectory $\boldsymbol{\alpha}(k)$) and the trajectory of the predicted values of the controlled variable (the trajectory $\hat{\boldsymbol{y}}(k)$). Linearizations will allow to substitute the complex nonlinear optimization with quadratic programming. In general, multiple solutions (minima) are possible in nonlinear optimization, while a quadratic task has only one (global) solution. That dramatically simplifies the online computational procedure.

The custom cost function $J_{\mathrm{c}}(k)$, in which the polynomial (Eq. (10)) or the neural (Eq. (11)) approximator is used, is given by Eq. (9). Due to nonlinearity of the process and nonlinear nature of the cost function, the whole function $J_{\mathrm{c}}(k)$ depends in a nonlinear way on the MPC decision vector $\triangle\boldsymbol{u}(k)$ (Eq. (1)). To solve the problem, we will adopt and extend the general trajectory linearization method discussed in [20] for the Wiener class of dynamical systems. However, in our case linearization is much more difficult since we have to find a linear approximation of the predicted control errors measured by the nonlinear function α. The objective is to find a linear approximation of the approximator output, i.e., the signal $\alpha(e(k+p|k))$, for the whole prediction horizon, i.e., for all $p = 1, \ldots, N$. It leads to a quadratic custom cost function $J_{\mathrm{c}}(k)$. Using the vector notation, we expect to derive a linear approximation of the vector

$$\boldsymbol{\alpha}(k) = \left[\alpha(e(k+1|k)) \ldots \alpha(e(k+N|k))\right]^{\mathrm{T}} \tag{12}$$

The linearized substitute for the trajectory $\boldsymbol{\alpha}(k)$ is found not in a simple way, for a past or current operating point of the process, but along a future trajectory of the process input

$$\boldsymbol{u}^{\mathrm{traj}}(k) = \left[u^{\mathrm{traj}}(k|k) \ldots u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k)\right]^{\mathrm{T}} \tag{13}$$

The trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ plays the role of the linearization point in scalar linearization. The assumed trajectory should be close to the true future process trajectory, which is, of course, unknown at the time instant k. A straightforward guess of that trajectory is to define it using the manipulated variable applied at the previous sampling instant or the elements of the optimal solution of the MPC

optimization task derived at the previous sampling instant. Example choices of the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ are discussed in the last part of Section 3.4. Taylor series expansion is applied to find the linearized substitute for the nonlinear function $\alpha(e(k+p|k))$. The consecutive elements of the vector (12) are obtained from

$$\begin{aligned}\alpha(e(k+p|k)) = \alpha(e^{\mathrm{traj}}(k+p|k)) \\ + \sum_{r=0}^{N_{\mathrm{u}}-1} \frac{\partial \alpha(e^{\mathrm{traj}}(k+p|k))}{\partial u^{\mathrm{traj}}(k+r|k)} (u(k+r|k) - u^{\mathrm{traj}}(k+r|k))\end{aligned} \tag{14}$$

for all $p = 1, \ldots, N$. Let us note that the future values of the input process variable, $u(k+r|k)$ for $r = 0, \ldots, N_{\mathrm{u}} - 1$, are the arguments of the linear approximations of the functions $\alpha(e(k+p|k))$. Conversely, the quantities $u^{\mathrm{traj}}(k+r|k)$ are fixed numbers, they define the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ (Eq. (13)) which is utilized for linearization. The control errors for the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ are determined from Eq. (8) which gives

$$e^{\mathrm{traj}}(k+p|k) = y^{\mathrm{sp}}(k+p|k) - \hat{y}^{\mathrm{traj}}(k+p|k) \tag{15}$$

where the values $\hat{y}^{\mathrm{traj}}(k+p|k)$ for all $p = 1, \ldots, N$ are derived using a dynamical process model embedded in MPC, using the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ and the general prediction equation (3), i.e.

$$\hat{y}^{\mathrm{traj}}(k+p|k) = f_{\mathrm{model}}^{\mathrm{traj}}(k+p|k) + d(k) \tag{16}$$

The quantities $\alpha(e^{\mathrm{traj}}(k+p|k))$ over the whole prediction horizon are found from the controller errors $e^{\mathrm{traj}}(k+p|k)$ and a chosen type of the approximator, e.g., given by Eq. (10) or Eq. (11). The partial derivatives in Eq. (14) are numbers; they are computed for the particular structure of the approximator used. In the case of the polynomial approximator (10), we have

$$\frac{\partial \alpha(e^{\mathrm{traj}}(k+p|k))}{\partial u^{\mathrm{traj}}(k+r|k)} = -\sum_{i=1}^{n} i w_i \left(e^{\mathrm{traj}}(k+p|k)\right)^{i-1} \frac{\partial \hat{y}^{\mathrm{traj}}(k+p|k)}{\partial u^{\mathrm{traj}}(k+r|k)} \tag{17}$$

for $p = 1, \ldots, N$ and $r = 0, \ldots, N_{\mathrm{u}} - 1$. The MLP neural approximator (11) with the tanh function used in the hidden layer requires the formula

$$\frac{\partial \alpha(e^{\mathrm{traj}}(k+p|k))}{\partial u^{\mathrm{traj}}(k+r|k)} = -\sum_{i=1}^{K} w_{i,1}^1 w_i^2 \left(1 - (\tanh(z_i^{\mathrm{traj}}(k+p|k)))^2\right) \frac{\partial \hat{y}^{\mathrm{traj}}(k+p|k)}{\partial u^{\mathrm{traj}}(k+r|k)} \tag{18}$$

The consecutive inputs of hidden nodes, i.e., for $i = 1, \ldots, K$, for the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$, are

$$z_i^{\mathrm{traj}}(k+p|k) = w_{i,0}^1 + w_{i,1}^1 (y^{\mathrm{sp}}(k+p|k) - \hat{y}^{\mathrm{traj}}(k+p|k)) \tag{19}$$

For convenient derivations, the matrix-vector notation is utilized in place of the scalar one. Using Eq. (14), the vector of the linearized approximator's outputs over the whole prediction horizon used, i.e., the vector (12), becomes

$$\boldsymbol{\alpha}(k) = \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} (\boldsymbol{u}(k) - \boldsymbol{u}^{\mathrm{traj}}(k)) \tag{20}$$

Similarly to Eq. (12), the vector of the approximator's outputs for the chosen trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ is

$$\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) = \left[\alpha(e^{\mathrm{traj}}(k+1|k)) \dots \alpha(e^{\mathrm{traj}}(k+N|k))\right]^{\mathrm{T}} \tag{21}$$

The partial derivatives comprise the following $N \times N_{\mathrm{u}}$ matrix

$$\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} = \begin{bmatrix} \dfrac{\partial\alpha(e^{\mathrm{traj}}(k+1|k))}{\partial u^{\mathrm{traj}}(k|k)} & \cdots & \dfrac{\partial\alpha(e^{\mathrm{traj}}(k+1|k))}{\partial u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k)} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial\alpha(e^{\mathrm{traj}}(k+N|k))}{\partial u^{\mathrm{traj}}(k|k)} & \cdots & \dfrac{\partial\alpha(e^{\mathrm{traj}}(k+N|k))}{\partial u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k)} \end{bmatrix} \tag{22}$$

The independent variable vector of length N_{u} in Eq. (20) is

$$\boldsymbol{u}(k) = \left[u(k|k) \dots u(k+N_{\mathrm{u}}-1|k)\right]^{\mathrm{T}} \tag{23}$$

It is easily calculated from the corresponding vector of increments $\triangle\boldsymbol{u}(k)$ (Eq. (1))

$$\boldsymbol{u}(k) = \boldsymbol{J}\triangle\boldsymbol{u}(k) + \boldsymbol{u}(k-1) \tag{24}$$

The $N_{\mathrm{u}} \times N_{\mathrm{u}}$ matrix $\boldsymbol{J}$ has its all diagonal and below-diagonal entries equal to 1 while all above-diagonal entries are 0. The $N_{\mathrm{u}} \times 1$ vector is $\boldsymbol{u}(k-1) = u(k-1)\,[1 \dots 1]^{\mathrm{T}}$. Using the simple relation between $\boldsymbol{u}(k)$ and $\triangle\boldsymbol{u}(k)$ defined by Eq. (24), the linearized trajectory of the approximator's output over the entire prediction horizon (Eq. (20)) becomes

$$\boldsymbol{\alpha}(k) = \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\triangle\boldsymbol{u}(k) + \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \tag{25}$$

We clearly see that the linearized trajectory $\boldsymbol{\alpha}(k)$ is really a linear function of the MPC decision vector $\triangle\boldsymbol{u}(k)$. It means that the custom cost function (9) is a quadratic function of that vector.

3.3 Linearization of the Trajectory $\hat{\boldsymbol{y}}(k)$

Due to nonlinearity of the process, the constraints put on the predicted controlled variable in the rudimentary MPC optimization task (6) depend in a nonlinear way on the MPC decision variable vector $\triangle\boldsymbol{u}(k)$ (Eq. (1)). To formulate a quadratic optimization task, linear approximations of these constraints must be also found. Once again, we use the general trajectory linearization method discussed in [20], but now we have to find a linear approximation of the vector of the controlled variable over the prediction horizon, i.e., the vector

$$\hat{\boldsymbol{y}}(k) = \left[\hat{y}(k+1|k)) \dots \hat{y}(k+N|k))\right]^{\mathrm{T}} \tag{26}$$

Linearization is determined along the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ defined by Eq. (13). We use a similar mathod that is used for linearization of the trajectory $\boldsymbol{\alpha}(k)$ defined by Eq. (12). Similarly to the vector (20), the linearized predicted trajectory in terms of the future values of the manipulated process input, i.e., the vector $\boldsymbol{u}(k)$, is

$$\hat{\boldsymbol{y}}(k) = \hat{\boldsymbol{y}}^{\mathrm{traj}}(k) + \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k) - \boldsymbol{u}^{\mathrm{traj}}(k)) \tag{27}$$

where the predicted vector of the controlled process output related to the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ is

$$\hat{\boldsymbol{y}}^{\mathrm{traj}}(k) = \left[\hat{y}^{\mathrm{traj}}(k+1|k)) \dots \hat{y}^{\mathrm{traj}}(k+N|k))\right]^{\mathrm{T}} \tag{28}$$

and the partial derivatives comprise the $N \times N_{\mathrm{u}}$ matrix, which is similar to the matrix (22)

$$\frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} = \begin{bmatrix} \dfrac{\partial \hat{y}^{\mathrm{traj}}(k+1|k)}{\partial u^{\mathrm{traj}}(k|k)} & \cdots & \dfrac{\partial \hat{y}^{\mathrm{traj}}(k+1|k)}{\partial u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k)} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial \hat{y}^{\mathrm{traj}}(k+N|k)}{\partial u^{\mathrm{traj}}(k|k)} & \cdots & \dfrac{\partial \hat{y}^{\mathrm{traj}}(k+N|k)}{\partial u^{\mathrm{traj}}(k+N_{\mathrm{u}}-1|k)} \end{bmatrix} \tag{29}$$

Using Eq. (24), it is easy to express the linearized predicted trajectory of the process output variable in terms of the decision vector of MPC

$$\hat{\boldsymbol{y}}(k) = \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\triangle\boldsymbol{u}(k) + \hat{\boldsymbol{y}}^{\mathrm{traj}}(k) + \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \tag{30}$$

Similarly to Eq. (25), we clearly see that the linearized trajectory $\hat{\boldsymbol{y}}(k)$ is also a linear function of the MPC optimal solution vector $\triangle\boldsymbol{u}(k)$.

The MPC formulation recommended in this work relies on trajectory linearization, i.e., the linear approximations of the trajectories $\boldsymbol{\alpha}(k)$ and $\hat{\boldsymbol{y}}(k)$ are directly carried out. Some computationally efficient MPC algorithms [20, 31] apply model linearization at each sampling instant and the linearized model serves for finding the predicted trajectories. As shown in [20], direct trajectory linearization typically gives better control quality of MPC than model linearization. It is because model linearization is performed for the past measurements of manipulated and controlled variables, the current controlled value is also utilized. Conversely, trajectory linearization is computed for a future trajectory of the manipulated variable.

3.4 Formulation of the MPC Optimization Task

Using the approximator in the first part of the custom cost function (9) and having found the linearized trajectories (25) and (30), from the basic task (6), we obtain the quadratic optimization problem

$$\begin{aligned} \min_{\triangle\boldsymbol{u}(k)} \Bigg\{ J_{\mathrm{c}}(k) = \Bigg\| & \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\triangle\boldsymbol{u}(k) + \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) \\ & + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \Bigg\|^2 + \|\triangle\boldsymbol{u}(k)\|^2_{\boldsymbol{\Lambda}} \Bigg\} \\ \text{s. t.} & \\ \boldsymbol{u}^{\min} \le{}& \boldsymbol{J}\triangle\boldsymbol{u}(k) + \boldsymbol{u}(k-1) \le \boldsymbol{u}^{\max} \\ \triangle\boldsymbol{u}^{\min} \le{}& \triangle\boldsymbol{u}(k) \le \triangle\boldsymbol{u}^{\max} \\ \boldsymbol{y}^{\min} \le{}& \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\triangle\boldsymbol{u}(k) + \hat{\boldsymbol{y}}^{\mathrm{traj}}(k) \\ & + \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \le \boldsymbol{y}^{\max} \end{aligned} \tag{31}$$

The $N_{\mathrm{u}} \times N_{\mathrm{u}}$ matrix is $\boldsymbol{\Lambda} = \lambda \mathrm{diag}(1, \ldots, 1)$; the $N_{\mathrm{u}} \times 1$ vectors are: $\boldsymbol{u}^{\min} = u^{\min}\left[1 \ldots 1\right]^{\mathrm{T}}$, $\boldsymbol{u}^{\max} = u^{\max}\left[1 \ldots 1\right]^{\mathrm{T}}$, $\triangle\boldsymbol{u}^{\min} = \triangle u^{\min}\left[1 \ldots 1\right]^{\mathrm{T}}$ and $\triangle\boldsymbol{u}^{\max} = \triangle u^{\max}\left[1 \ldots 1\right]^{\mathrm{T}}$; the $N \times 1$ vectors are: $\boldsymbol{y}^{\min} = y^{\min}\left[1 \ldots 1\right]^{\mathrm{T}}$ and $\boldsymbol{y}^{\max} = y^{\max}\left[1 \ldots 1\right]^{\mathrm{T}}$. The optimization task's (31) number of decision variables is N_{u}.

Let us note that the last constraints in optimization tasks (2) and (6), constituted by the prediction equation (3), are nonlinear in terms of the future increments $\triangle\boldsymbol{u}(k)$. The prediction equation (16), necessary to compute the trajectory $\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)$, is nonlinear in terms of the assumed trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$, but is is independent of the vector $\triangle\boldsymbol{u}(k)$. Hence, it is not used in the set of constraints in the derived optimization task (31).

The derived task (31) is of the quadratic optimization type as the minimized objective function is linear in respect to the optimal solution vector $\triangle\boldsymbol{u}(k)$ and all limitations are linear. It means that it has only one solution, which is the global solution. As far as the feasibility problem (31) is concerned, it is always feasible, i.e., there exists a non-empty set of possible solutions, provided that there are only the constraints imposed on the magnitude and rate of change of the manipulated variable. Unfortunately, the inclusion of the constraints imposed on the predicted controlled variable may result in an empty feasible set, in particular when the model error is significant and/or an important disturbance affects the process.

Because the limitations associated with the predicted process output variable are likely to yield an empty set of possible solutions, these constraints are implemented in the soft version [19, 20, 34]. We derive the following optimization task in place of the optimization problem (31) with hard constraints

$$
\begin{aligned}
\min_{\substack{\triangle\boldsymbol{u}(k) \\ \varepsilon^{\min}(k) \\ \varepsilon^{\max}(k)}} & \left\{ J_{\mathrm{c}}(k) = \left\| \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \boldsymbol{J} \triangle\boldsymbol{u}(k) + \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) \right.\right. \\
& \qquad \left. + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} (\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \right\|^2 \\
& \qquad \left. + \|\triangle\boldsymbol{u}(k)\|^2_{\boldsymbol{\Lambda}} + \rho^{\min}(\varepsilon^{\min}(k))^2 + \rho^{\max}(\varepsilon^{\max}(k))^2 \right\} \\
& \text{s. t.} \\
& \boldsymbol{u}^{\min} \leq \boldsymbol{J} \triangle\boldsymbol{u}(k) + \boldsymbol{u}(k-1) \leq \boldsymbol{u}^{\max} \\
& \triangle\boldsymbol{u}^{\min} \leq \triangle\boldsymbol{u}(k) \leq \triangle\boldsymbol{u}^{\max} \\
& \boldsymbol{y}^{\min} - \underline{\boldsymbol{\varepsilon}}^{\min}(k) \leq \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \boldsymbol{J} \triangle\boldsymbol{u}(k) + \hat{\boldsymbol{y}}^{\mathrm{traj}}(k) \\
& \qquad + \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} (\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \leq \boldsymbol{y}^{\max} + \underline{\boldsymbol{\varepsilon}}^{\max}(k) \\
& \varepsilon^{\min}(k) \geq 0, \ \varepsilon^{\max}(k) \geq 0
\end{aligned}
\tag{32}
$$

The additional positive calculated variables $\varepsilon^{\min}(k)$ and $\varepsilon^{\max}(k)$ temporarily modify the original hard constraints which enlarges the admissible set. The $N \times 1$ vectors are: $\underline{\boldsymbol{\varepsilon}}^{\min}(k) = \varepsilon^{\min}(k)\left[1 \ldots 1\right]^{\mathrm{T}}$ and $\underline{\boldsymbol{\varepsilon}}^{\max}(k) = \varepsilon^{\max}(k)\left[1 \ldots 1\right]^{\mathrm{T}}$. The number of optimization task's (32) decision variables is $N_{\mathrm{u}} + 2$. The obtained MPC optimization task (32) with soft output constraints is always feasible, i.e., the set of possible solutions is never empty, which is enforced by the variables $\varepsilon^{\min}(k)$ and $\varepsilon^{\max}(k)$. As the cost

function is quadratic and all constraints are linear in terms of the decision variables ($\triangle \boldsymbol{u}(k)$, $\varepsilon^{\min}(k)$ and $\varepsilon^{\max}(k)$), the obtained MPC optimization task has only one global solution.

Next, we must transform the MPC optimization task with soft constraints (32) to the classical quadratic optimization form. The new $(n_{\mathrm{u}}N_{\mathrm{u}}+2)\times 1$ vector of the optimal solution is

$$\boldsymbol{x}(k) = \left[\triangle \boldsymbol{u}^{\mathrm{T}}(k)\ \varepsilon^{\min}(k)\ \varepsilon^{\max}(k)\right]^{\mathrm{T}} \tag{33}$$

The auxiliary matrices given below

$$\boldsymbol{N}_1 = \left[\begin{array}{cc} \boldsymbol{I}_{N_{\mathrm{u}}\times N_{\mathrm{u}}} & \boldsymbol{0}_{N_{\mathrm{u}}\times 2} \end{array}\right] \tag{34}$$

$$\boldsymbol{N}_2 = \left[\begin{array}{ccc} \boldsymbol{0}_{1\times N_{\mathrm{u}}} & \boldsymbol{I}_{1\times 1} & \boldsymbol{0}_{1\times 1} \end{array}\right] \tag{35}$$

$$\boldsymbol{N}_3 = \left[\begin{array}{cc} \boldsymbol{0}_{1\times (N_{\mathrm{u}}+1)} & \boldsymbol{I}_{1\times 1} \end{array}\right] \tag{36}$$

are of dimensionality $N_{\mathrm{u}}\times(N_{\mathrm{u}}+2)$, $1\times(N_{\mathrm{u}}+2)$ and $1\times(N_{\mathrm{u}}+2)$, respectively. The matrices (34)-(36) are used to determine the vector of the MPC original optimal solution and the additional quotients that enlarge the feasible set from the vector of decision variables $\boldsymbol{x}(k)$

$$\triangle \boldsymbol{u}(k) = \boldsymbol{N}_1\boldsymbol{x}(k) \tag{37}$$

$$\varepsilon^{\min}(k) = \boldsymbol{N}_2\boldsymbol{x}(k) \tag{38}$$

$$\varepsilon^{\max}(k) = \boldsymbol{N}_3\boldsymbol{x}(k) \tag{39}$$

For further transformations, the vectors $\underline{\boldsymbol{\varepsilon}}^{\min}(k)$ and $\underline{\boldsymbol{\varepsilon}}^{\max}(k)$ are rewritten

$$\underline{\boldsymbol{\varepsilon}}^{\min}(k) = \boldsymbol{I}_{N\times 1}\otimes \varepsilon^{\min}(k) \tag{40}$$

$$\underline{\boldsymbol{\varepsilon}}^{\max}(k) = \boldsymbol{I}_{N\times 1}\otimes \varepsilon^{\max}(k) \tag{41}$$

where the operator $\otimes$ stands for the vector Kronecker product. From Eqs. (38)-(39) and (40)-(41), we get

$$\underline{\boldsymbol{\varepsilon}}^{\min}(k) = \boldsymbol{I}_{N\times 1}\otimes \boldsymbol{N}_2\boldsymbol{x}(k) \tag{42}$$

$$\underline{\boldsymbol{\varepsilon}}^{\max}(k) = \boldsymbol{I}_{N\times 1}\otimes \boldsymbol{N}_3\boldsymbol{x}(k) \tag{43}$$

The standard quadratic programming problem is

$$\begin{aligned} &\min_{\boldsymbol{x}(k)} \left\{0.5\boldsymbol{x}^{\mathrm{T}}(k)\boldsymbol{H}_{\mathrm{QP}}(k)\boldsymbol{x}(k) + \boldsymbol{f}_{\mathrm{QP}}^{\mathrm{T}}(k)\boldsymbol{x}(k)\right\} \\ &\text{s. t.} \\ &\boldsymbol{A}(k)\boldsymbol{x}(k) \le \boldsymbol{b}(k) \end{aligned} \tag{44}$$

Using Eqs (37), (42) and (43), the quadratic optimization problem (32) is rewritten using only the decision vector $\boldsymbol{x}(k)$

$$\min_{\boldsymbol{x}(k)} \left\{ J_{\mathrm{c}}(k) = \left\| \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)} \boldsymbol{J}\boldsymbol{N}_1\boldsymbol{x}(k) + \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) \right.\right.$$

$$
\begin{aligned}
&+ \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k))\Big\|^2 \\
&+ \|\boldsymbol{N}_1\boldsymbol{x}(k)\|^2_{\boldsymbol{\Lambda}} + \rho^{\min}(\boldsymbol{N}_2\boldsymbol{x}(k))^2 + \rho^{\max}(\boldsymbol{N}_3\boldsymbol{x}(k))^2 \Big\}
\end{aligned}
$$

s. t. (45)

$$
\begin{aligned}
&\boldsymbol{u}^{\min} \le \boldsymbol{J}\boldsymbol{N}_1\boldsymbol{x}(k) + \boldsymbol{u}(k-1) \le \boldsymbol{u}^{\max} \\
&\triangle\boldsymbol{u}^{\min} \le \boldsymbol{N}_1\boldsymbol{x}(k) \le \triangle\boldsymbol{u}^{\max} \\
&\boldsymbol{y}^{\min} - \boldsymbol{I}_{N\times 1}\otimes\boldsymbol{N}_2\boldsymbol{x}(k) \le \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\boldsymbol{N}_1\boldsymbol{x}(k) + \hat{\boldsymbol{y}}^{\mathrm{traj}}(k) \\
&\qquad + \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k)) \le \boldsymbol{y}^{\max} + \boldsymbol{I}_{N\times 1}\otimes\boldsymbol{N}_3\boldsymbol{x}(k) \\
&\hat{y}^{\mathrm{traj}}(k+p|k) = f^{\mathrm{traj}}_{\mathrm{model}}(k+p|k) + d(k),\ p = 1,\ldots,N \\
&\boldsymbol{N}_2\boldsymbol{x}(k) \ge 0,\ \boldsymbol{N}_3\boldsymbol{x}(k) \ge 0
\end{aligned}
$$

Having differentiated the cost function $J_{\mathrm{c}}(k)$ in (45), we have

$$
\begin{aligned}
\frac{\mathrm{d}J_{\mathrm{c}}(k)}{\mathrm{d}\boldsymbol{x}(k)} = {} & 2\boldsymbol{N}_1^{\mathrm{T}}\boldsymbol{J}^{\mathrm{T}}\left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\right)^{\mathrm{T}}\left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\boldsymbol{N}_1\boldsymbol{x}(k) + \boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))\right. \\
& \left. + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k))\right) \\
& + 2\boldsymbol{N}_1^{\mathrm{T}}\boldsymbol{\Lambda}^{\mathrm{T}}\boldsymbol{N}_1\boldsymbol{x}(k) + 2\rho^{\min}\boldsymbol{N}_2^{\mathrm{T}}\boldsymbol{N}_2\boldsymbol{x}(k) + 2\rho^{\max}\boldsymbol{N}_3^{\mathrm{T}}\boldsymbol{N}_3\boldsymbol{x}(k)
\end{aligned} \tag{46}
$$

From Eq. (46), it is straightforward to read the hessian matrix necessary in the quadratic optimization task (44) is

$$
\begin{aligned}
\boldsymbol{H}_{\mathrm{QP}}(k) = {} & 2\boldsymbol{N}_1^{\mathrm{T}}\boldsymbol{J}^{\mathrm{T}}\left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\right)^{\mathrm{T}}\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\boldsymbol{N}_1 \\
& + 2\boldsymbol{N}_1^{\mathrm{T}}\boldsymbol{\Lambda}^{\mathrm{T}}\boldsymbol{N}_1 + 2\rho^{\min}\boldsymbol{N}_2^{\mathrm{T}}\boldsymbol{N}_2 + 2\rho^{\max}\boldsymbol{N}_3^{\mathrm{T}}\boldsymbol{N}_3
\end{aligned} \tag{47}
$$

The vector $\boldsymbol{f}_{\mathrm{QP}}(k)$ is

$$
\boldsymbol{f}_{\mathrm{QP}}(k) = 2\boldsymbol{N}_1^{\mathrm{T}}\boldsymbol{J}^{\mathrm{T}}\left(\frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\right)^{\mathrm{T}}\left(\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k)) + \frac{\mathrm{d}\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\mathrm{traj}}(k))\right) \tag{48}
$$

The limitations subject to which optimization is performed are specified by the matrix

$$
\boldsymbol{A}(k) = \begin{bmatrix}
-\boldsymbol{J}\boldsymbol{N}_1 \\
\boldsymbol{J}\boldsymbol{N}_1 \\
-\boldsymbol{I}_{N\times 1}\otimes\boldsymbol{N}_2 - \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\boldsymbol{N}_1 \\
-\boldsymbol{I}_{N\times 1}\otimes\boldsymbol{N}_3 + \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\mathrm{traj}}(k)}\boldsymbol{J}\boldsymbol{N}_1 \\
-\boldsymbol{N}_1 \\
\boldsymbol{N}_1 \\
-\boldsymbol{N}_2 \\
-\boldsymbol{N}_3
\end{bmatrix} \tag{49}
$$

and the vector

$$\boldsymbol{b}(k) = \begin{bmatrix} -\boldsymbol{u}^{\min} + \boldsymbol{u}(k-1) \\ \boldsymbol{u}^{\max} - \boldsymbol{u}(k-1) \\ -\boldsymbol{y}^{\min} + \hat{\boldsymbol{y}}^{\text{traj}}(k) + \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\text{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\text{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\text{traj}}(k)) \\ \boldsymbol{y}^{\max} - \hat{\boldsymbol{y}}^{\text{traj}}(k) - \frac{\mathrm{d}\hat{\boldsymbol{y}}^{\text{traj}}(k)}{\mathrm{d}\boldsymbol{u}^{\text{traj}}(k)}(\boldsymbol{u}(k-1) - \boldsymbol{u}^{\text{traj}}(k)) \\ -\triangle\boldsymbol{u}^{\min} \\ \triangle\boldsymbol{u}^{\max} \\ 0 \\ 0 \end{bmatrix} \tag{50}$$

Let us compare the discussed MPC methods with state-space feedback control methods. Firstly, the derivation presented in this work is concerned with input-output models, while feedback control utilizes state-space models. A modification of the presented MPC approach to deal with state-space process description is possible. Secondly, the classical state-space control methods rely on linear models while nonlinear ones are considered in this work. Nonlinear state-space control methods have been developed recently for specific model structures, e.g., [36, 37]. Conversely, the model structure and properties of the model considered in this work are not limited; the only requirement is model differentiability; for non-differentiable models, the presented approach does not work. Thirdly, the classical state-space control methods frequently neglect constraints (extensions are necessary to use constraints, e.g., [36, 37]) while the described MPC approach allows to use different kinds of constraints on manipulated and controlled variables. Finally, the described MPC algorithms use online quadratic optimization, not explicit control laws. It may turn out that the use of the state-space feedback control laws requires a shorter time than optimization. Of course, it depends on the performance of the hardware used for algorithm implementation.

Fig. 2 shows flowcharts of two discussed MPC algorithms, i.e., organization of calculations performed at each time step. We name two algorithms:

1. MPC-NPLT-$\mathrm{L_c}$: the MPC algorithm with Nonlinear Prediction and Linearization along the Trajectory using the $\mathrm{L_c}$ norm. The following calculations are performed at each sampling step:

   (a) At first, the algorithm calculates the predicted trajectories $\boldsymbol{\alpha}(\boldsymbol{e}^{\text{traj}}(k))$ (Eq. (21)) and $\hat{\boldsymbol{y}}^{\text{traj}}(k)$ (Eq. (28)) for the assumed trajectory $\boldsymbol{u}^{\text{traj}}(k)$ (Eq. (13)) using Eqs. (15) and (16), respectively. A dynamical model of the process chosen by the user is used to determine the predictions.

   (b) Secondly, linear approximations of the trajectories $\boldsymbol{\alpha}(k)$ (Eq. (12) and $\hat{\boldsymbol{y}}(k)$ (26) are computed from Eqs. (25) and (30), respectively. Linearization of both trajectories is performed using the chosen dynamical model. In particular, the partial derivatives (22) and (29) are calculated for the model used.

   (c) Finally, the algorithm solves the quadratic optimization task (45) and sends the first element of the obtained vector to the process.

   Three variants of the algorithm are used: MPC-NPLT1-$\mathrm{L_c}$, MPC-NPLT2-$\mathrm{L_c}$ and MPC-NPLT3-$\mathrm{L_c}$. The difference is the way the trajectory utilized during linearization (Eq. (13)) is chosen.

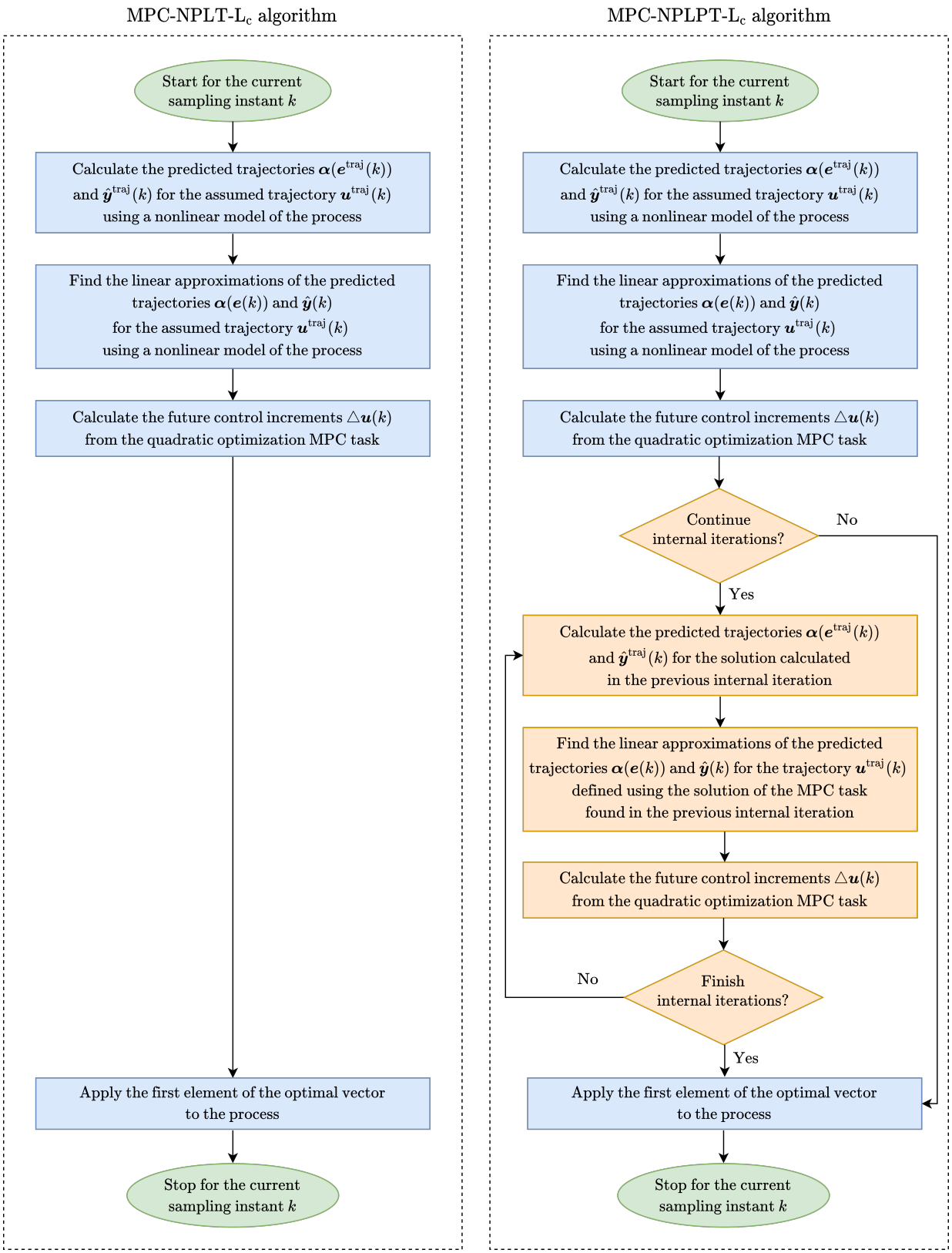

Figure 2: Flowcharts of two discussed MPC algorithms

In the first case, its all elements are the same as the manipulated variable just recently used, i.e., $u(k-1)$. In the second one, that trajectory is defined by the last $N_{\mathrm{u}}-1$ elements of the optimal solution computed at the sampling instant $k-1$. The third alternative is to use an inverted static process model to define every element of the trajectory by the manipulated variable corresponding to the current setpoint.

2. MPC-NPLPT-$\mathrm{L_c}$: the MPC algorithm with Nonlinear Prediction and Linearization along the Predicted Trajectory using the $\mathrm{L_c}$ norm. The following calculations are performed at each sampling step:

   (a) Initially, the algorithm calculates the predicted trajectories $\boldsymbol{\alpha}(\boldsymbol{e}^{\mathrm{traj}}(k))$ and $\hat{\boldsymbol{y}}^{\mathrm{traj}}(k)$ and linear approximations of the trajectories $\boldsymbol{\alpha}(k)$ and $\hat{\boldsymbol{y}}(k)$ for the assumed trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$, exactly in the same way it is done in the MPC-NPLT-$\mathrm{L_c}$ method.

   (b) The quadratic optimization task (45) is solved for the first time.

   (c) The above calculations may be repeated in internal iterations in which the trajectory $\boldsymbol{u}^{\mathrm{traj}}(k)$ is defined using the optimal solution of the MPC optimization task performed at the previous internal iteration. Such iterations are started if the predicted value of the controlled variable is reasonably different from its actual value. The iterations are stopped when the solutions of the MPC optimization task in two successive iterations are close or the possible number of iterations is exceeded.

4 Simulations

4.1 Process Description

Properties of the presented above approach to MPC are demonstrated for a neutralization reactor process [10]. Because of its nonlinear static and dynamic behavior, this process is very frequently utilized as a benchmark in model identification tasks and nonlinear control, e.g., [10, 11, 14, 20]. A base (NaOH) stream flow rate q_1 (ml/s) is the manipulated variable, the value of the pH is the controlled process output.

4.2 Process Modeling for MPC

In simulations the results of which are discussed next, the fundamental model based on first principles of the reactor [10] is used. Conversely, in all MPC algorithms, a Wiener model of the process is used. Its linear dynamic block is

$$v(k) = b_1 u(k-1) + b_2 u(k-2) - a_1 v(k-1) - a_2 v(k-2) \tag{51}$$

where v is an auxiliary model variable, the nonlinear static block is

$$y(k) = g_{\mathrm{static}}(v(k)) \tag{52}$$

The $g_{\mathrm{static}}\colon \mathbb{R} \to \mathbb{R}$ scalar function is represented by a two layered neural network with five nonlinear sigmoid units in the first layer. Identification and validation of the model defined by Eqs. (51)-(52) is

thoroughly discussed in [20]. The sampling time of the model, equal to the sampling time of MPC, is 10 seconds. Process variables are scaled: $u = q_1 - \bar{q}_1,\ y = \mathrm{pH} - \overline{\mathrm{pH}}$ where for the initial point $\bar{q}_1 = 15.5$ and $\overline{\mathrm{pH}} = 7$.

4.3 Implementation Details of the Computationally Efficient Nonlinear MPC Algorithm Using the Custom Cost Functions

During linearization of the trajectory $\boldsymbol{\alpha}(k)$, the elements of the derivative matrix (22) are computed from Eqs. (17) and (18) for the polynomial and neural approximators, respectively. The partial derivatives of the predicted controlled variable used in Eqs. (17) and (18) and in the matrix (29) that is necessary during linearization of the trajectory $\hat{\boldsymbol{y}}(k)$ are found for the particular type of the model defined by Eqs. (51)-(52) as detailed in [20].

4.4 Simulation Results and Discussion

In simulations, we will compare the following MPC methods:

1. MPC-NO-$\mathrm{L_c}$: the MPC approach with Nonlinear Optimization using the $\mathrm{L_c}$ norm. The underlying general nonlinear task is constituted by Eq. (2), but the soft constraints are put on the controlled variable, as in the formulation (32). Although nonlinear optimization is computationally expensive and nonlinear solvers are significantly more complex than quadratic ones, the MPC-NO-$\mathrm{L_c}$ method is a reference to assess the linearization-based MPC algorithms.

2. MPC-NPLT1-$\mathrm{L_c}$, MPC-NPLT2-$\mathrm{L_c}$ and MPC-NPLT3-$\mathrm{L_c}$: three versions of the MPC-NPLT-$\mathrm{L_c}$ algorithm using the $\mathrm{L_c}$ norm. The quadratic optimization task is given by Eq. (32). In the last variant, the trajectory utilized during linearization (Eq. (13)) is determined from an inverse static model represented by a two-layered MLP neural structure; ten hidden tanh nodes are used.

3. MPC-NPLPT-$\mathrm{L_c}$ with the $\mathrm{L_c}$ norm. At each execution of the algorithm, initially, the trajectory used for linearization is obtained from the inverse static model (as in the MPC-NPLT3-$\mathrm{L_c}$ approach). The obtained solution is used for possible consecutive linearization; five such repetitions are possible.

4. MPC-NPLPT-$\mathrm{L_2}$: the algorithm with the classical $\mathrm{L_2}$ norm and multiple trajectory linearizations at one sampling instant.

In all versions of MPC algorithms, the same set of parameters is used: the horizon N is 10, the horizon N_u is 3, the weight λ is 1. The prediction horizon and the weight λ have been tuned to obtain stable process operation even in the case of modeling errors and additional disturbances (as shown in Section 4.4.7). These two parameters are typically used to obtain stability in practice [34]. To guarantee stability, it is recommended to use the described MPC algorithms supplemented by an additional stabilizing controller that works in the neighborhood of the equilibrium points (the dual-mode MPC approach) [19]. The minimal and maximal box limits put on the process input are 0 and

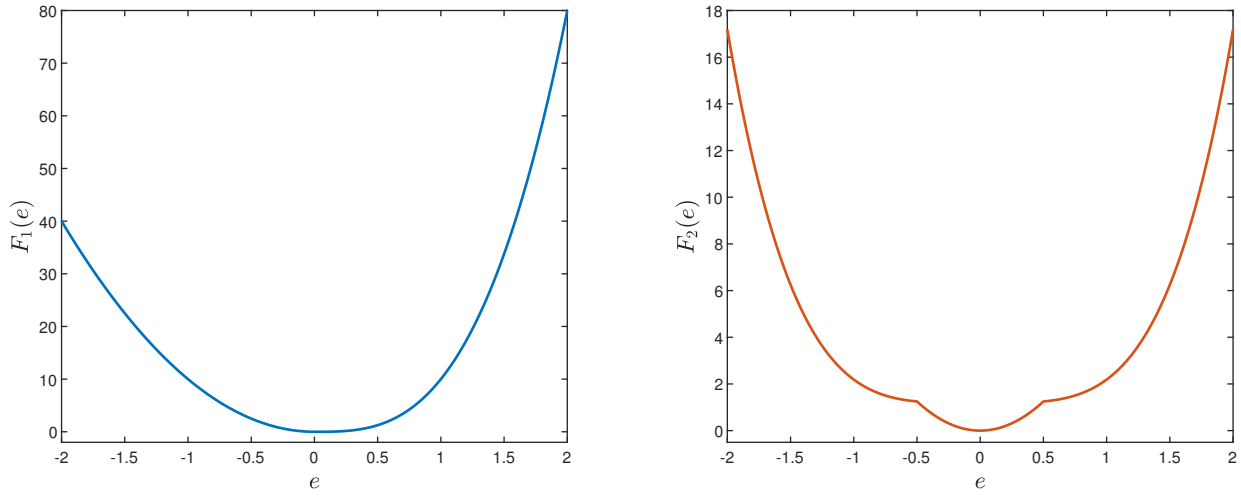

Figure 3: The custom cost functions F_1 and F_2

30, respectively. They are taken into account in all simulation scenarios and all algorithms. The limits put on the process output are only considered in Section 4.4.6.

In this work, two custom cost functions are used. They are characterized by the following analytical expressions (exact functions)

$$F_1(e) = \begin{cases} 10e^2 & \text{if } e \leq 0 \\ 10e^3 & \text{if } e > 0 \end{cases} \tag{53}$$

and

$$F_2(e) = \begin{cases} e^4 + 1.1875 & \text{if } e < -0.5 \\ 5e^2 & \text{if } -0.5 \leq e < 0.5 \\ e^4 + 1.1875 & \text{if } e \geq 0.5 \end{cases} \tag{54}$$

Both functions are sketched in Fig. 3.

The following part of the simulation section demonstrates different aspects of the approximators and MPC algorithms.

4.4.1 Accuracy of Neural and Polynomial Approximation of Custom Cost Functions

At first, the efficacy of neural and polynomial approximation of the cost functions is discussed. Both classes of approximators are used in MPC. Fig. 4 presents the cost function F_1 vs. its neural and polynomial approximations. The MLP network with ten tanh hidden nodes is used; the polynomials of the degree $n = 2, 3, 4, 5, 6$ are considered. For each configuration of the approximator, the approximation error is also drawn. In general, the higher the degree of the polynomial approximator, the better its accuracy. Unfortunately, even the most complex polynomial is characterized by much greater errors when compared with the neural one. Of note, the accuracy of the neural approximator is excellent.

Fig. 5 compares the cost function F_2 vs. its approximations. The MLP network with ten tanh hidden nodes is used. The symmetric shape of the cost function indicates that polynomials of the even degree should be considered; the results for $n = 2, 4, 6, 8, 10$ are compared. The polynomial approximation

for the second cost function is worse than in the first case. We may observe that the increment of the polynomial degree has a minimal effect on its accuracy. The approximation error is large even for the most complex polynomial ($n = 10$). Conversely, the accuracy of the neural approximator is excellent.

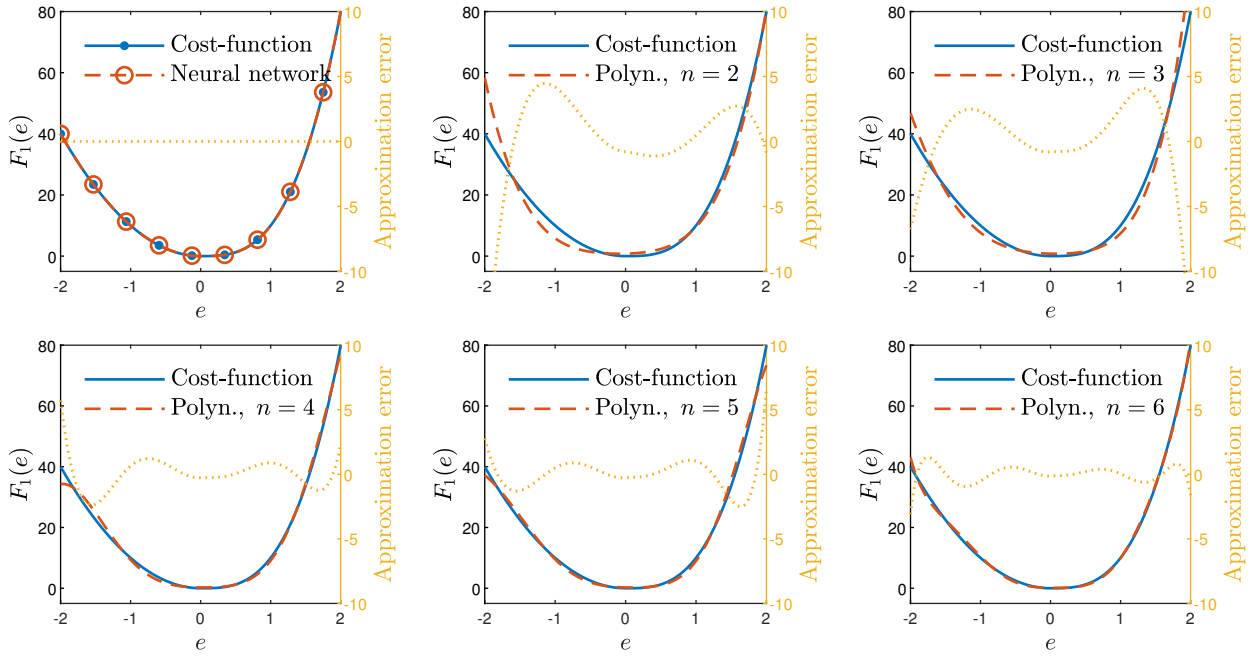

Figure 4: The custom cost function F_1 vs. its neural and polynomial approximations; approximation errors are also given

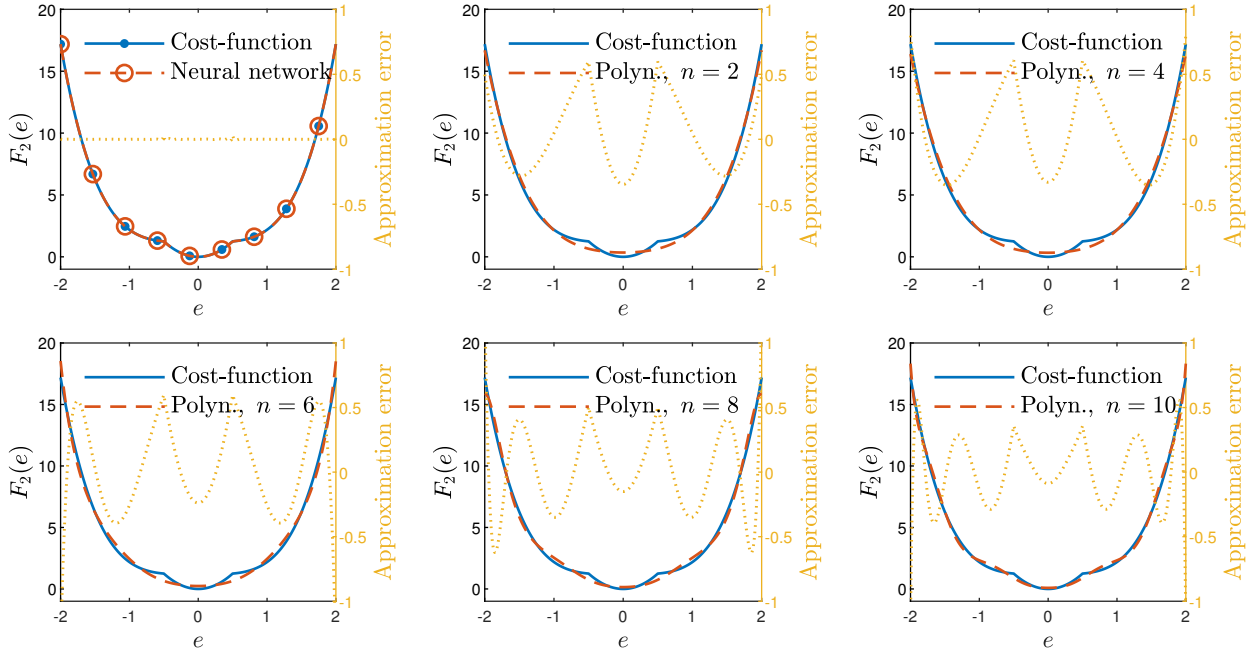

Figure 5: The custom cost function F_2 vs. its neural and polynomial approximations; approximation errors are also given

4.4.2 Efficiency of the MPC-NO-L$_c$ Approach: Neural vs. Polynomial Approximation

The MPC-NO-L$_c$ algorithm is first evaluated with neural and polynomial approximators. We expect that the use of an approximator results in a process trajectory the same or very similar to the case when the analytical exact cost functions, defined by Eqs. (53) and (54), are utilized. The basic question is which approximator type can efficiently replace the rudimentary exact cost function.

Figs. 6 and 7 depict the simulated process trajectories for the MPC-NO-L$_c$ algorithm in which the exact cost functions F_1 and F_2 and their neural approximators are used, respectively. Similarly, Figs. 8 and 9 compares the trajectories when the exact cost functions and their polynomial approximators of different degrees are used. We can observe that the neural approximators give excellent results as the trajectories are practically the same as those when the exact cost functions are used. That is the expected result. Unfortunately, the polynomial approximators give much worse results. Of course, the higher the degree of the approximators' polynomial, the better the control, but it is much worse than when the neural approximator is used. Interestingly, the MPC algorithm is sensitive to the inaccuracy of the approximator. Although in the case of the cost function F_1, for a high degree of the polynomial, approximation accuracy is not bad as shown in Fig. 4 but when such an approximation is used in MPC, as shown in Fig. 8, the control quality is below expectations. Hence, the neural approximators are only used in the following simulation, not the polynomial one.

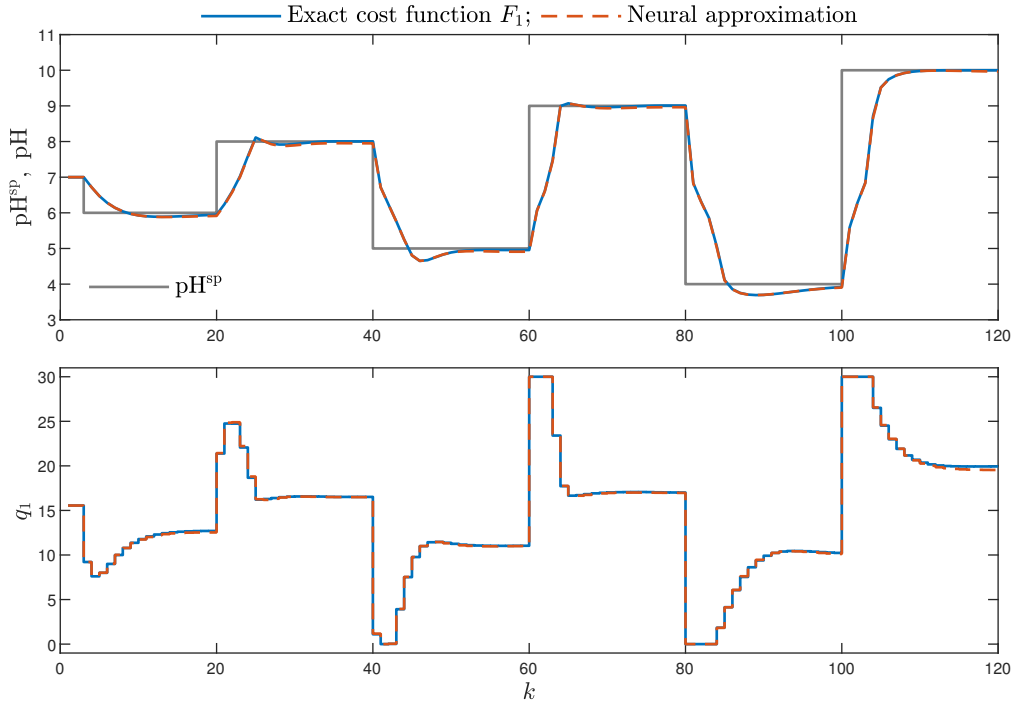

Figure 6: The process trajectories: the MPC-NO-L$_c$ control scheme using the exact custom cost function F_1 vs. the MPC-NO-L$_c$ algorithm using the neural approximation

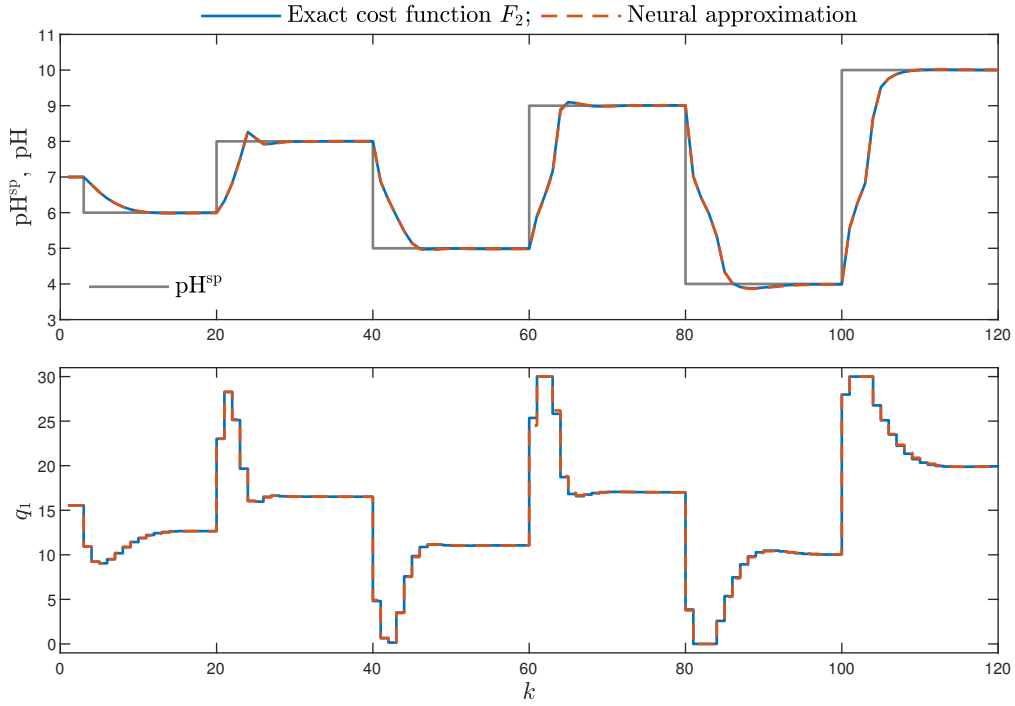

Figure 7: The process trajectories: the MPC-NO-L$_c$ method using the exact custom cost function F_2 vs. the MPC-NO-L$_c$ algorithm using the neural approximation

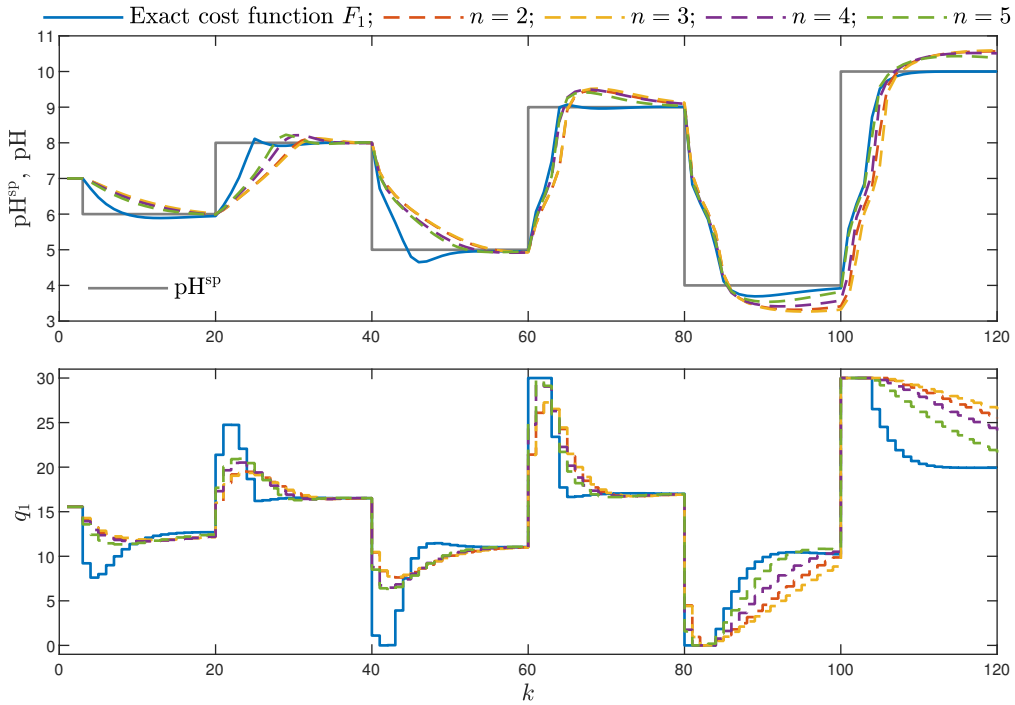

Figure 8: The process trajectories: the MPC-NO-L$_c$ control scheme using the exact custom cost function F_1 vs. the MPC-NO-L$_c$ algorithm using the polynomial approximation; n stands for the polynomial degree

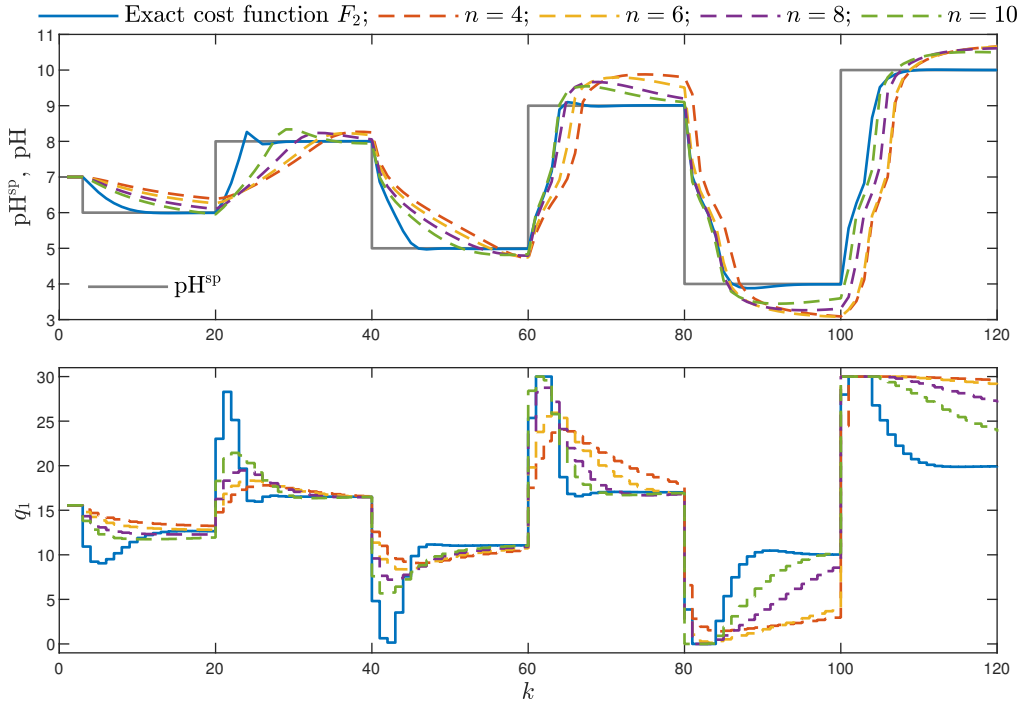

Figure 9: The process trajectories: the MPC-NO-L$_c$ method using the exact custom cost function F_2 vs. the MPC-NO-L$_c$ approach using the polynomial approximation; n stands for the polynomial degree

4.4.3 Efficiency of MPC-NPLT1-L$_c$, MPC-NPLT2-L$_c$ and MPC-NPLT3-L$_c$ methods (with One Trajectory Linearization at Each Sampling Instant) and Neural Approximation

Figs. 10 and 11 presents simulation results of the MPC-NPLT1-L$_c$, MPC-NPLT2-L$_c$ and MPC-NPLT3-L$_c$ algorithms with one trajectory linearization at each execution of the MPC and the neural approximation of the custom cost functions F_1 and F_2, respectively; the algorithms differ in the way the trajectory for linearization is chosen, initialized and then updated at every discrete time instant. For the first custom cost function, F_1, the first initialization method gives not good results, particularly for the first two setpoint changes. The third initialization method is the best one. For the second custom cost function, F_2, the second initialization method fails for the second setpoint change (it may be rectified by increasing the penalty factor λ, but it would slow down control). The first initialization method gives slightly slower trajectories for the first, the third and the fourth setpoint changes. Hence, the MPC-NPLT3-L$_c$ algorithm is selected for further comparison. It turns out that for the considered neutralization system, the trajectory for linearization should be defined by the manipulated variable's value related to the setpoint.

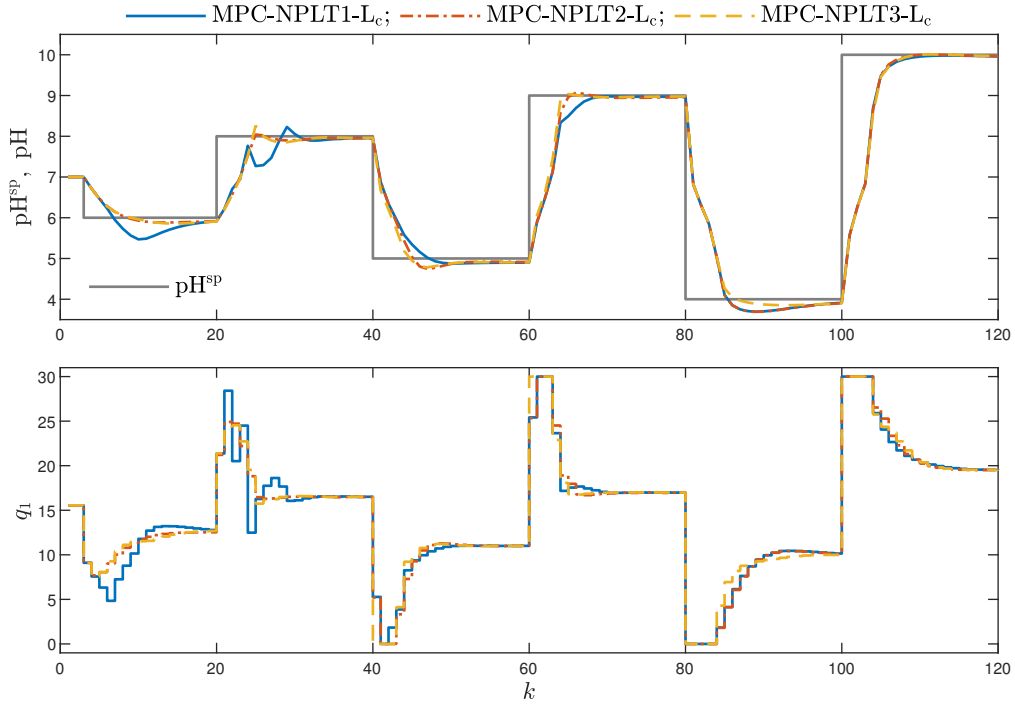

Figure 10: The process trajectories: the MPC-NPLT1-L_c, MPC-NPLT2-L_c and MPC-NPLT3-L_c approaches using the neural approximation of the custom cost function F_1

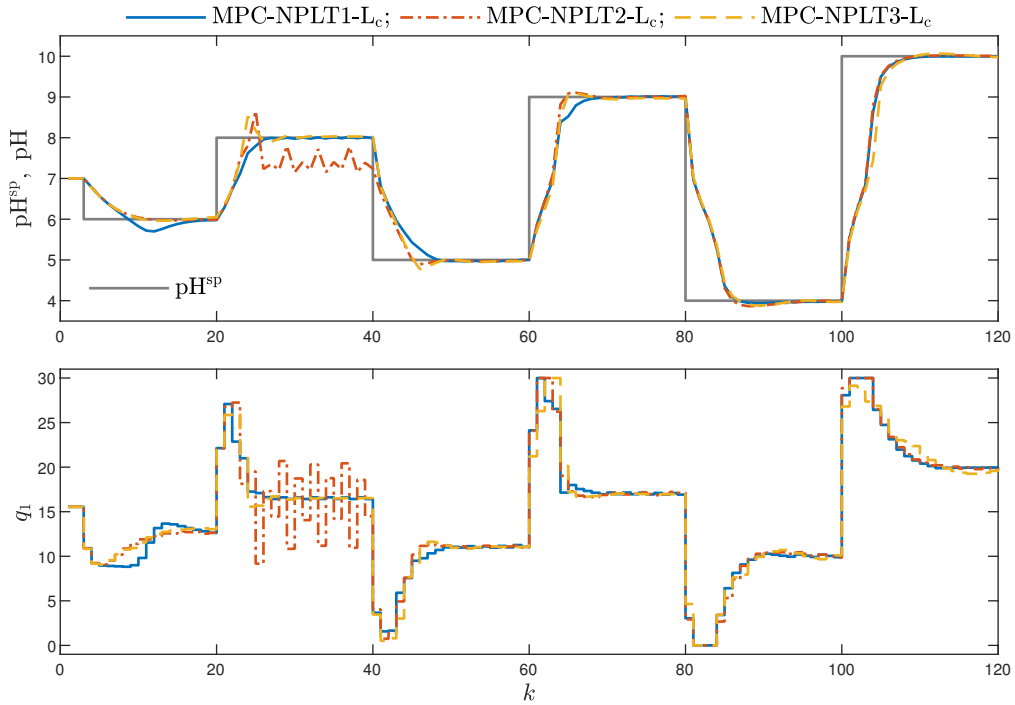

Figure 11: The process trajectories: the MPC-NPLT1-L_c, MPC-NPLT2-L_c and MPC-NPLT3-L_c methods using the neural approximation of the custom cost function F_2

4.4.4 Efficiency of the MPC-NPLPT-L$_c$ Control Scheme (with Multiple Trajectory Linearizations) and Neural Approximation

Figs. 12 and 13 depict the results of simulations for the following three algorithms: MPC-NO-L$_c$, MPC-NPLT3-L$_c$ and MPC-NPLPT-L$_c$; in all cases the neural approximations of the custom cost functions F_1 and F_2, respectively, are used. From the presented results, we find out that multiple linearization at each time step (MPC-NPLPT-L$_c$) produces slightly better outcomes than the MPC-NPLT3-L$_c$ scheme with one linearization but the differences are really insignificant. Of course, for other processes multiple linearization is likely to be necessary [20]. Moreover, the trajectories in the MPC-NPLT3-L$_c$ and MPC-NPLPT-L$_c$ methods with quadratic optimization are very similar to that possible in the reference MPC-NO-L$_c$ scheme requiring nonlinear optimization.

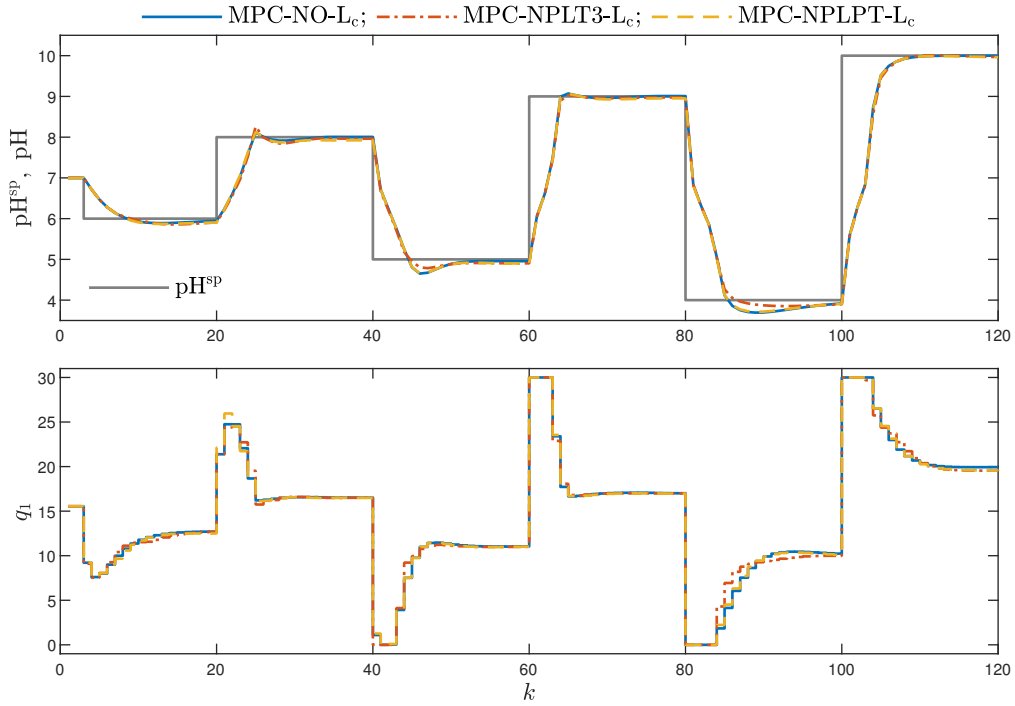

Figure 12: The process trajectories: the MPC-NO-L$_c$, MPC-NPLT3-L$_c$ and MPC-NPLPT-L$_c$ method using the neural approximation of the custom cost function F_1

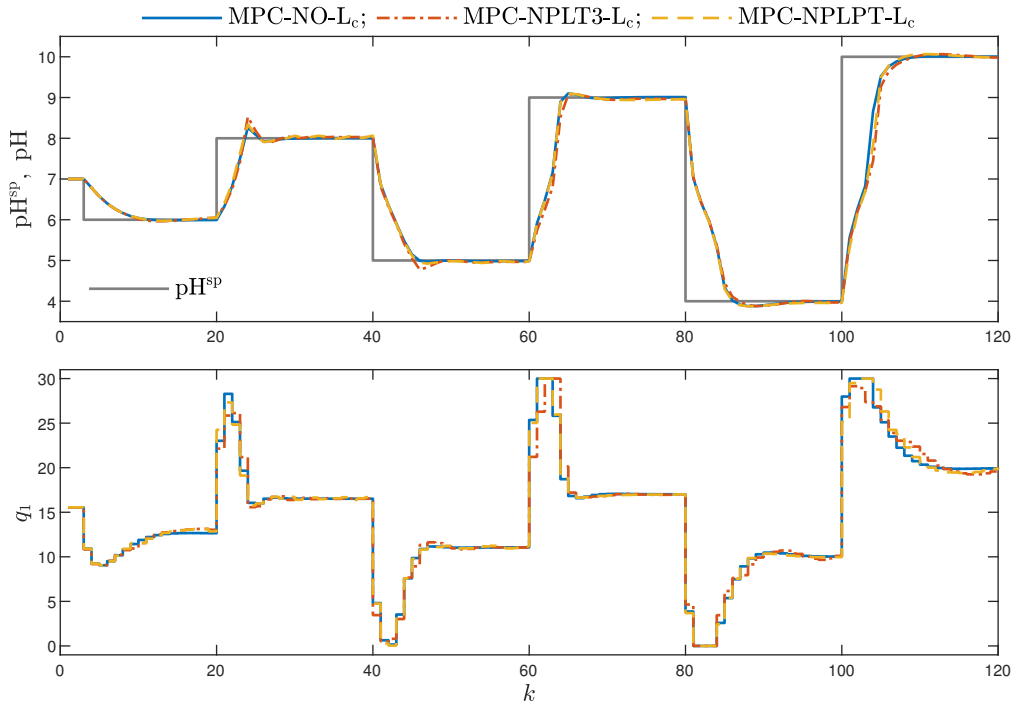

Figure 13: The process trajectories: the MPC-NO-L_c, MPC-NPLT3-L_c and MPC-NPLPT-L_c algorithms using the neural approximation of the custom cost function F_2

4.4.5 The MPC-NPLPT-L_c Method with the Custom Cost Function vs. the MPC-NPLPT-L_2 Algorithm with the Classical L_2 Norm

Figs. 14 and 15 compare the MPC-NPLPT-L_c control method in which the neural approximation of the custom cost functions F_1 and F_2, respectively, is used with the MPC-NPLPT-L_2 algorithm in which the classical cost function L_2 is applied. We can easily observe that the use of custom cost functions results in different trajectories of both manipulated and controlled variables when compared with the classical L_2 cost function. In particular, for two considered custom cost functions F_1 and F_2, control quality is better since the process output faster reaches its setpoint. Moreover, in the case of the cost function F_2, overshoot is lower.

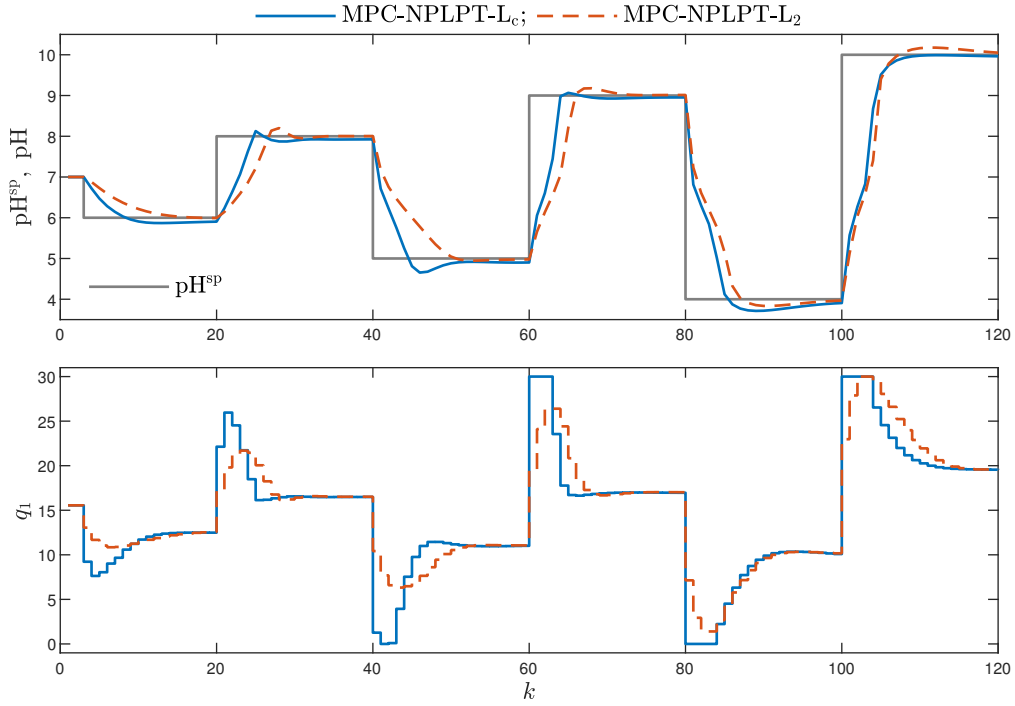

Figure 14: The process trajectories: the MPC-NPLPT-L$_c$ algorithm using the neural approximation of the custom cost function F_1 and the MPC-NPLPT-L$_2$ algorithm using the classical cost function L$_2$

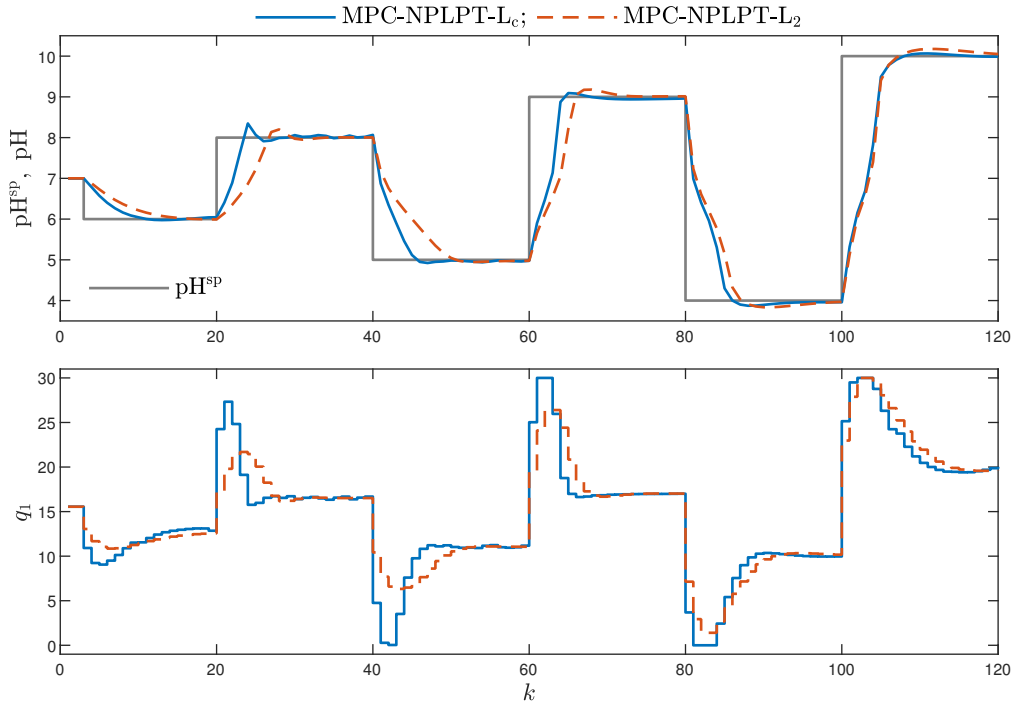

Figure 15: The process trajectories: the MPC-NPLPT-L$_c$ algorithm using the neural approximation of the custom cost function F_2 and the MPC-NPLPT-L$_2$ algorithm using the classical cost function L$_2$

The results presented above in figures are compared numerically in Tables 1 and 2. The first of the considered quality indexes measures summarized squared control errors

$$E_2 = \sum_{k=k_{\min}}^{k_{\max}} (y^{\mathrm{sp}}(k) - y(k))^2 \tag{55}$$

where $y(k)$ is the process output's value, $k_{\min}$ and $k_{\max}$ indicate the range of simulation horizon. The second of the control performance indexes corresponds with the custom cost function F

$$E_{\mathrm{c}} = \sum_{k=k_{\min}}^{k_{\max}} F(y^{\mathrm{sp}}(k) - y(k)) \tag{56}$$

Let us note that the indicators E_2 and E_{c} are not actually optimized in MPC, but they are calculated afterward. The MPC cost function has two parts; they are minimized over the prediction and control horizons. Interestingly, the custom cost function gives better results in terms of the performance index E_{c} and the index E_2. For the precise neural approximator, the obtained values of E_2 are lower when the custom cost function is used, not the classical quadratic one.

Considering Figs. 6-15 as well as Tables 1 and 2, for cost functions F_1 and F_2, we observe the following:

1. In the MPC-NO-$\mathrm{L_c}$ approaches, the application of the neural approximator leads to very similar results as the use of the exact function. Conversely, the polynomial approximator does not give good results, although high degrees are used.

2. The MPC-NPLT1-$\mathrm{L_c}$, MPC-NPLT2-$\mathrm{L_c}$ and MPC-NPLT3-$\mathrm{L_c}$ control methods with one linearization and optimization at each sampling instant give good results.

3. The MPC-NPLPT-$\mathrm{L_c}$ algorithm with multiple linearizations and optimizations gives practically the same results as the MPC-NO-$\mathrm{L_c}$ method.

4. The MPC-NPLPT-$\mathrm{L_2}$ control approach with the classical $\mathrm{L_2}$ cost function gives significantly different (worse) results.

Tables 1 and 2 also specify the calculation time of all algorithms. We find out the following:

1. The MPC-NO-$\mathrm{L_c}$ algorithm, no matter the exact cost function is used or its polynomial and neural approximators, is characterized by the longest calculation time.

2. The MPC-NPLT1-$\mathrm{L_c}$, MPC-NPLT2-$\mathrm{L_c}$ and MPC-NPLT3-$\mathrm{L_c}$ algorithms with one repetition of linearization and optimization are approximately three times faster.

3. The MPC-NPLPT-$\mathrm{L_c}$ scheme with multiple repetitions of linearization and optimization needs a slightly longer calculation time, but it is approximately two times faster than the MPC-NO-$\mathrm{L_c}$ method.

4. As a result of using the neural approximator of the custom cost function, the MPC-NPLPT-$\mathrm{L_c}$ approach is slightly slower than the MPC-NPLPT-$\mathrm{L_2}$ algorithm with the simple quadratic cost function.

Let us note that the online linearization used in the aforementioned MPC algorithms makes them computationally efficient in two ways. Firstly, they solve quadratic tasks with only one global minimum, not nonlinear ones, with possibly multiple local solutions. It is a fundamental difference. Secondly, quadratic programming is demonstrated to be much faster.

Table 1: Simulation results: comparison of control quality indexes E_2 and E_c and the calculation time for MPC algorithms using the custom cost function F_1; for the MPC-NPLPT-L_2 algorithm the classical L_2 norm is used

Algorithm	Approximator	E_2	E_c	Calculation time
MPC-NO-L_c	Exact cost function	1.858×10^2	5.816×10^3	123.28%
MPC-NO-L_c	Neural	1.865×10^2	5.882×10^3	100.00%
MPC-NO-L_c	Polynomial, $n = 2$	2.778×10^2	9.607×10^3	81.51%
MPC-NO-L_c	Polynomial, $n = 3$	2.980×10^2	1.077×10^4	78.77%
MPC-NO-L_c	Polynomial, $n = 4$	2.438×10^2	8.090×10^3	82.88%
MPC-NO-L_c	Polynomial, $n = 5$	2.132×10^2	6.450×10^3	87.67%
MPC-NO-L_c	Polynomial, $n = 6$	2.030×10^2	6.165×10^3	89.73%
MPC-NO-L_c	Polynomial, $n = 7$	1.971×10^2	5.957×10^3	93.15%
MPC-NPLT1-L_c	Neural	1.935×10^2	5.996×10^3	33.56%
MPC-NPLT2-L_c	Neural	1.904×10^2	5.995×10^3	33.56%
MPC-NPLT3-L_c	Neural	1.869×10^2	5.916×10^3	34.25%
MPC-NPLPT-L_c	Neural	1.859×10^2	5.872×10^3	55.47%
MPC-NPLPT-L_2	Exact cost function	2.329×10^2	7.048×10^3	52.08%

Table 2: Simulation results: comparison of control quality indexes E_2 and E_c and the calculation time for MPC algorithms using the custom cost function F_2; for the MPC-NPLPT-L_2 algorithm the classical L_2 norm is used

Algorithm	Approximator	E_2	E_c	Calculation time
MPC-NO-L_c	Exact cost function	1.894×10^2	3.319×10^3	121.99%
MPC-NO-L_c	Neural	1.900×10^2	3.329×10^3	100.00%
MPC-NO-L_c	Polynomial, $n = 2$	4.377×10^2	1.113×10^4	63.83%
MPC-NO-L_c	Polynomial, $n = 4$	4.342×10^2	1.106×10^4	64.54%
MPC-NO-L_c	Polynomial, $n = 6$	3.808×10^2	9.588×10^3	73.76%
MPC-NO-L_c	Polynomial, $n = 8$	2.968×10^2	6.559×10^3	73.05%
MPC-NO-L_c	Polynomial, $n = 10$	2.359×10^2	4.571×10^3	77.31%
MPC-NPLT1-L_c	Neural	1.925×10^2	3.340×10^3	34.75%
MPC-NPLT2-L_c	Neural	1.901×10^2	3.258×10^3	35.46%
MPC-NPLT3-L_c	Neural	2.011×10^2	3.478×10^3	35.46%
MPC-NPLPT-L_c	Neural	1.958×10^2	3.479×10^3	67.08%
MPC-NPLPT-L_2	Exact cost function	2.329×10^2	3.856×10^3	55.03%

4.4.6 Satisfaction of Constraints Put on Predicted Process Output

Let us consider the cost function F_1 and additional constraints the objective of which is to reduce the overshoot. In order to do it, we have to impose the constraints on the predicted process output. The quotients $y^{\min}$ and $y^{\max}$ are changed in the following sampling instants as follows

$$y^{\min}(k) = \begin{cases} 6 & \text{for } 3 \leq k < 19 \\ 5 & \text{for } 40 \leq k < 59 \\ 4 & \text{for } 80 \leq k < 99 \end{cases} \tag{57}$$

and

$$y^{\max}(k) = \begin{cases} 8 & \text{for } 20 \leq k < 39 \\ 9 & \text{for } 60 \leq k < 79 \\ 10 & \text{for } 100 \leq k \leq 120 \end{cases} \tag{58}$$

The constraints do not exist for the sampling instants other than specified. Of course, we cannot guarantee that the additional constraints are always satisfied. There are two reasons for that. Firstly, the constraints rely on predictions characterized by some error since the model is good but not perfect. Secondly, the additional limitations are implemented as soft, which always guarantees non-empty feasible set of solutions of the MPC optimization task (45). Weights associated with the additional penalty terms are: $\rho_{\min} = \rho_{\max} = 10^3$.

Table 3 specifies the control quality indicators E_2 and E_c for all considered MPC control techniques with additional output constraints. We observe the following:

1. The use of the neural approximator in the MPC-NO-L_c algorithm gives excellent results while the application of polynomials gives poor performance. Increasing the polynomial degree improves the outcome, but the results are still far from those possible when the neural network is used. The same is observed when the predicted controlled variables are not constrained (Figs. 6 and 7).

2. When only one trajectory linearization is used at each time step, the the MPC-NPLT1-L_c and the MPC-NPLT2-L_c algorithms do not work correctly for the first four setpoint steps as depicted in Fig. 16. Conversely, the MPC-NPLT3-L_c algorithm is much better. It is also true when the additional constraints are not considered (Figs. 10 and 11).

3. The MPC-NPLPT-L_c algorithm with multiple repetitions of linearization gives somewhat better outcomes than the MPC-NPLT3-L_c one. The trajectories are shown in Fig. 17.

4. In terms of the performance function E_c, the MPC-NPLPT-L_2 control approach with the classical L_2 cost function gives significantly worse results than the MPC-NPLPT-L_c scheme. Fig. 18 confirms that observation; the custom cost function gives faster control. The same is observed when the predicted controlled variable is not constrained as shown in Figs. 14 and 15.

As far as the calculation time of MPC algorithms is concerned, the additional constraints help to slightly shorten the calculation time of all MPC algorithms with linearization when compared with the scenario in which only the manipulated variable is limited, as shown in Tables 1 and 2.

Table 3: Simulation results when output constraints are utilized: comparison of control quality indexes E_2 and E_c and the calculation time for MPC algorithms using the custom cost function F_1; for the MPC-NPLPT-L_2 algorithm the classical L_2 norm is used

Algorithm	Approximator	E_2	E_c	Calculation time
MPC-NO-L_c	Exact cost function	1.805×10^2	5.589×10^3	120.07%
MPC-NO-L_c	Neural	1.795×10^2	5.581×10^3	100.00%
MPC-NO-L_c	Polynomial, $n = 2$	2.253×10^2	6.317×10^3	92.83%
MPC-NO-L_c	Polynomial, $n = 3$	2.272×10^2	6.338×10^3	93.19%
MPC-NO-L_c	Polynomial, $n = 4$	2.026×10^2	5.909×10^3	94.27%
MPC-NO-L_c	Polynomial, $n = 5$	1.993×10^2	5.867×10^3	97.49%
MPC-NO-L_c	Polynomial, $n = 6$	1.916×10^2	5.722×10^3	97.85%
MPC-NO-L_c	Polynomial, $n = 7$	1.902×10^2	5.701×10^3	98.92%
MPC-NPLT1-L_c	Neural	2.010×10^2	5.909×10^3	21.51%
MPC-NPLT2-L_c	Neural	1.909×10^2	5.890×10^3	21.15%
MPC-NPLT3-L_c	Neural	1.807×10^2	5.596×10^3	21.15%
MPC-NPLPT-L_c	Neural	1.790×10^2	5.586×10^3	31.96%
MPC-NPLPT-L_2	Exact cost function	2.315×10^2	7.396×10^3	30.53%

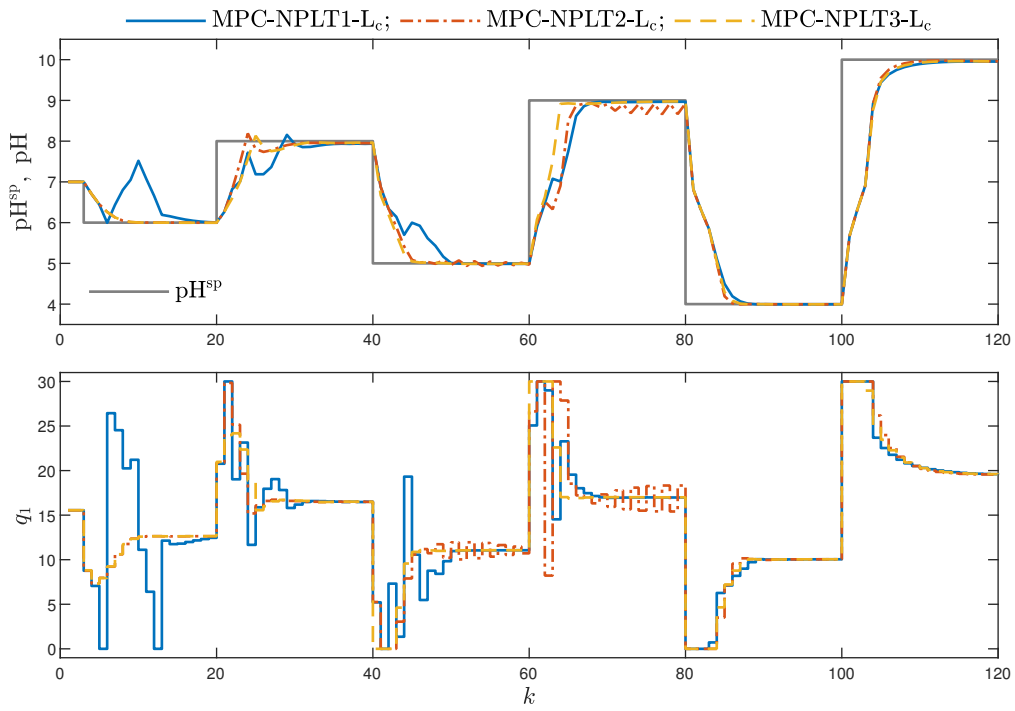

Figure 16: The process trajectories when output constraints are utilized: the MPC-NPLT1-L_c, MPC-NPLT2-L_c and MPC-NPLT3-L_c algorithms using the neural approximation of the custom cost function F_1

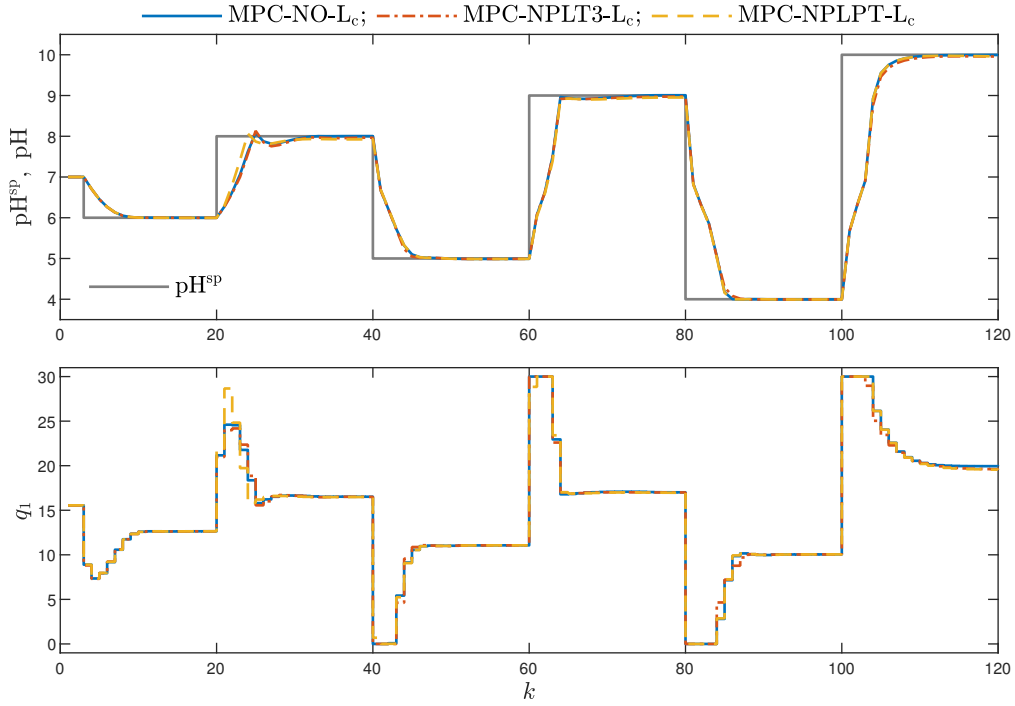

Figure 17: The process trajectories when output constraints are utilized: the MPC-NO-L$_c$, MPC-NPLT3-L$_c$ and MPC-NPLPT-L$_c$ algorithms using the neural approximation of the custom cost function F_1

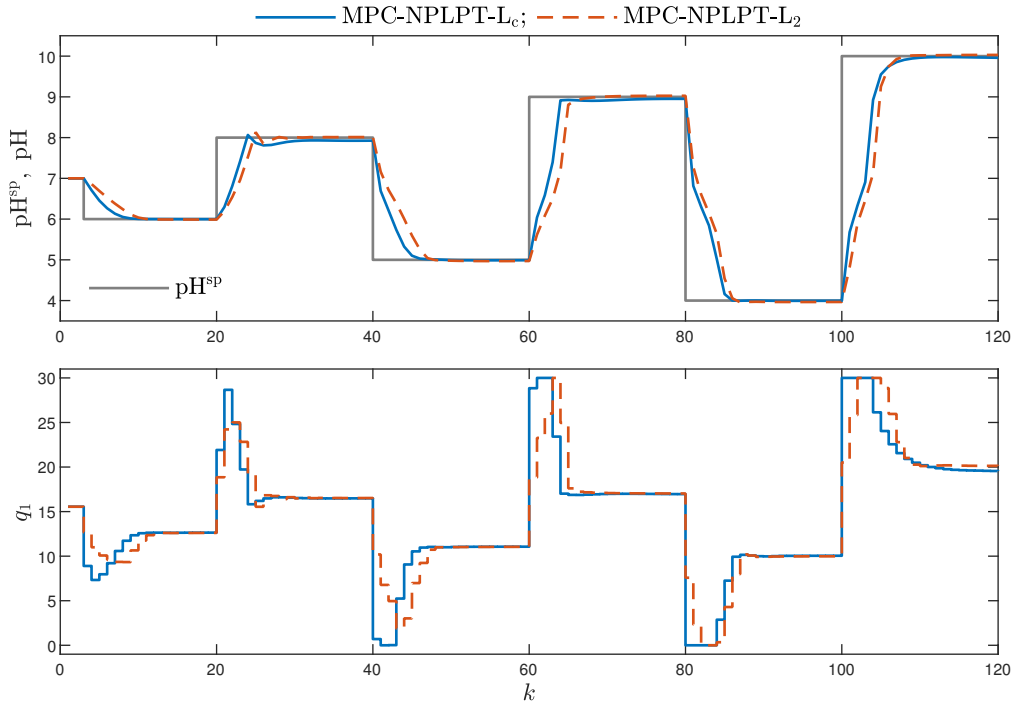

Figure 18: The process trajectories when output constraints are utilized: the MPC-NPLPT-L$_c$ control schemes using the neural approximation of the custom cost function F_1 and the MPC-NPLPT-L$_2$ algorithm using the classical cost function L$_2$

4.4.7 Compensation of Model Errors and Disturbances

A fundamental model of the reactor is used to simulate the process, while a Wiener model of the process is used in all considered MPC algorithms. Of course, the Wiener model is not perfect; it has some inaccuracy. Nevertheless, the presented MPC algorithms can compensate for modeling errors since the required setpoint is always reached in all simulations. Now, let us consider larger model errors and disturbances. To achieve this, the model static gain is decreased by 10% and an unmeasured additive unit disturbance affects the process output from the sampling instant 70. The values of the quality indexes are given in Table 4 and Fig. 19 depicts the trajectories obtained in three example MPC algorithms. The custom cost function F_1 is considered. The following observations are possible:

1. Due to the integral action, all considered MPC algorithms make it possible to achieve the required setpoint. The influence of the disturbance is very quickly compensated.

2. The MPC-NPLT1-L_c, MPC-NPLT2-L_c and MPC-NPLT3-L_c algorithms give quite good control quality whereas the MPC-NPLPT-L_c scheme is slightly better, very close to the MPC-NO-L_c scheme.

3. The use of the classical cost function in the MPC-NPLPT-L_2 algorithm gives worse control quality indicators.

4. The use of polynomial approximators, even in the MPC-NO-L_c algorithm, gives worse results than utilization of the neural structure.

5. The calculation time of MPC algorithms is fairly similar to the results obtained when the model error and additional disturbances are not considered, as shown in Table 1.

Table 4: Simulation results when model errors and disturbances are present: comparison of quality indexes E_2 and E_c and the calculation time for MPC algorithms using the custom cost function F_1; for the MPC-NPLPT-L_2 algorithm the classical L_2 norm is used

Algorithm	Approximator	E_2	E_c	Calculation time
MPC-NO-L_c	Exact cost function	2.951×10^2	1.121×10^4	124.00%
MPC-NO-L_c	Neural	2.957×10^2	1.127×10^4	100.00%
MPC-NO-L_c	Polynomial, $n = 2$	3.382×10^2	1.193×10^4	84.00%
MPC-NO-L_c	Polynomial, $n = 3$	3.406×10^2	1.195×10^4	85.60%
MPC-NO-L_c	Polynomial, $n = 4$	3.227×10^2	1.161×10^4	86.40%
MPC-NO-L_c	Polynomial, $n = 5$	3.169×10^2	1.147×10^4	93.60%
MPC-NO-L_c	Polynomial, $n = 6$	3.110×10^2	1.136×10^4	90.40%
MPC-NO-L_c	Polynomial, $n = 7$	3.076×10^2	1.128×10^4	92.00%
MPC-NPLT1-L_c	Neural	2.986×10^2	1.130×10^4	40.80%
MPC-NPLT2-L_c	Neural	2.967×10^2	1.130×10^4	40.80%
MPC-NPLT3-L_c	Neural	2.602×10^2	8.991×10^3	40.00%
MPC-NPLPT-L_c	Neural	2.827×10^2	1.041×10^4	67.51%
MPC-NPLPT-L_2	Exact cost function	3.399×10^2	1.246×10^4	61.76%

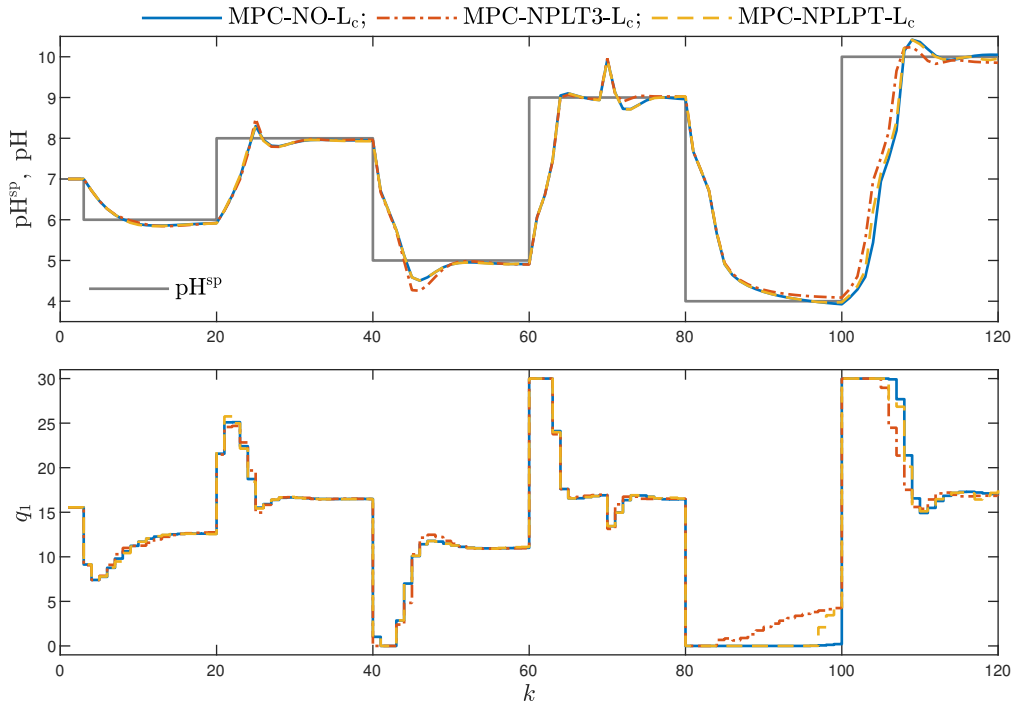

Figure 19: The process trajectories when when model errors and disturbances are present: the MPC-NO-L$_c$, MPC-NPLT3-L$_c$ and MPC-NPLPT-L$_c$ algorithms using the neural approximation of the custom cost function F_1

5 Concluding Remarks

A computationally efficient approach to constrained nonlinear MPC which uses a custom cost function is discussed in this work. It makes it possible to consider a different control objective than the classical case in which the quadratic norm is used, i.e., the summarized squared predicted control errors. The custom cost function, which may be defined analytically or graphically, is represented by an approximator. To obtain a computationally simple computational scheme, two predicted trajectories, i.e., the approximator's predicted output trajectory and the process output's predicted trajectory, are linearized at each algorithm's execution. The first linearized trajectory is used in the minimized MPC objective function; the second one is employed in the constraints. In consequence, a simple quadratic programming task is derived. For a simulated pH reactor, the efficiency of the presented approach is discussed. It is shown that when one trajectory linearization is performed at each algorithm's execution, the control quality is very good, provided that the trajectory used for linearization is determined using an inverse static model for the current setpoint of the controlled variable. Furthermore, when multiple linearizations are used when the sampling instant changes significantly, the resulting control quality is excellent, practically the same as in the general MPC control method with nonlinear programming. Validity of the described MPC algorithms is demonstrated for two custom cost functions when only simple box constraints are assumed on the manipulated process input and in a more demanding case when additional constraints are put on the predicted process output.

Of note, the results presented in this work indicate that the proposed MPC approach is very

efficient, provided that the accuracy of the approximator is very good. That is why neural approximators are recommended in this work, as their accuracy is much better than that of the polynomial approximators. It is necessary to emphasize that a precise approximator is necessary for good control quality. Unfortunately, not precise polynomial approximators yield unacceptable control quality, which is a limitation of the presented method. Conversely, precise neural approximators result in excellent control quality. Furthermore, the differentiability of the approximator is required for trajectory linearization. Neural networks, e.g., of MLP type, satisfy this requirement. It is important to note that the presented method does not limit the dynamical model's structure that is used for prediction. The only condition is model differentiability.

The extension of the presented MPC method to deal with multivariable processes is straightforward. In such a case, different cost functions may be used to assess the influence of control errors of the consecutive process outputs. Moreover, it is also possible to devise MPC algorithms with custom cost functions for dynamical systems defined by state-space nonlinear models.

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.

Acknowledgment

This research was supported by Warsaw University of Technology.

References

[1] P. Bania. An information based approach to stochastic control problems. *International Journal of Applied Mathematics and Computer Science*, 30(1):23–34, 2020.

[2] D. Boiroux and J. B. Jørgensen. Sequential ℓ_1 quadratic programming for nonlinear model predictive control. *IFAC–PapersOnLine*, 52(1):474–479, 2019.

[3] G. De Souza, D. Odloak, and A. C. Zanin. Real time optimization (RTO) with model predictive control (MPC). *Computers and Chemical Engineering*, 34(12):1999–2006, 2010.

[4] Y. Deng, X. Yin, and S. Hu. Event-triggered predictive control for networked control systems with DoS attacks. *Information Sciences*, 542:71–91, 2021.

[5] P. Domański and M. Ławryńczuk. Impact of MPC embedded performance index on control quality. *IEEE Access*, 9:24787–24795, 2021.

[6] A. Dötlinger and R. M. Kennel. Near time-optimal model predictive control using an L1-norm based cost functional. In *2014 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 3504–3511, 2014 IEEE Energy Conversion Congress and Exposition (ECCE), Pittsburgh, PA, USA, 2014.

[7] M. Farbood, M. Shasadeghi, T. Niknam, and B. Safarinejadian. Fuzzy Lyapunov-based model predictive sliding-mode control of nonlinear systems: an ellipsoid recursive feasibility approach. *IEEE Transactions on Fuzzy Systems*, 30(6):1929–1938, 2022.

[8] M. Farbood, M. Veysi, M. Shasadeghi, A. Izadian, T. Niknam, and J. Aghaei. Robustness improvement of computationally efficient cooperative fuzzy model predictive-integral sliding mode control of nonlinear systems. *IEEE Access*, 9:147874–147887, 2021.

[9] M. Fehér, O. Straka, and V. Šmídl. Model predictive control of electric drive system with L1-norm. *European Journal of Control*, 56:242–253, 2020.

[10] J. C. Gómez, A. Jutan, and E. Baeyens. Wiener model identification and predictive control of a pH neutralisation process. *Proceedings of IEE, Part D, Control Theory and Applications*, 151(3):329–338, 2004.

[11] A. Grancharova, J. Kocijan, and T. A. Johansen. Explicit output-feedback nonlinear predictive control based on black-box models. *Engineering Applications of Artificial Intelligence*, 24(2):388–397, 2011.

[12] O. Gulbudak and M. Gokdag. Finite control set model predictive control approach of nine switch inverter-based drive systems: Design, analysis, and validation. *ISA Transactions*, 110:283–304, 2021.

[13] S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, Upper Saddle River, New Jersey, 2009.

[14] A. W. Hermansson and S. Syafiie. Model predictive control of pH neutralization processes: a review. *Control Engineering Practice*, 45:98–109, 2015.

[15] D. Horla. Experimental results on actuator/sensor failures in adaptive GPC position control. *Actuators*, 10(3):1–18, 2021.

[16] E. Kang, H. Qiao, J. Gao, and W. Yang. Neural network-based model predictive tracking control of an uncertain robotic manipulator with input constraints. *ISA Transactions*, 109:89–101, 2021.

[17] P. Karamanakos, T. Geyer, and R. Kennel. On the choice of norm in finite control set model predictive control. *IEEE Transactions on Power Electronics*, 33(8):7105–7117, 2018.

[18] L. Kong and J. Yuan. Generalized discrete-time nonlinear disturbance observer based fuzzy model predictive control for boiler–turbine systems. *ISA Transactions*, 90:89–106, 2019.

[19] M. Ławryńczuk. *Computationally Efficient Model Predictive Control Algorithms: a Neural Network Approach*, volume 3 of *Studies in Systems, Decision and Control*. Springer, Cham, 2014.

[20] M. Ławryńczuk. *Nonlinear Predictive Control Using Wiener Models: Computationally Efficient Approaches for Polynomial and Neural Structures*, volume 389 of *Studies in Systems, Decision and Control*. Springer, Cham, 2022.

[21] M. Ławryńczuk and R. Nebeluk. Computationally efficient nonlinear model predictive control using the L1 cost-function. *Sensors*, 21(17):1–23, 2021.

[22] M. Ławryńczuk and P. Tatjewski. Efficient predictive control integrated with economic optimisation based on neural models. In L. Rutkowski, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, editors, *Proceedings of the 9th International Conference on Artificial Intelligence and Soft Computing (ICAISC 2008)*, volume 5097 of *Lecture Notes in Artificial Intelligence*, pages 111–122. Springer, Berlin, 2008.

[23] M. Lazar, W. P. M. H. Heemels, S. Weiland, A. Bemporad, and O. Pastravanu. Infinity norms as Lyapunov functions for model predictive control of constrained PWA systems. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *Lecture Notes in Computer Science*, pages 417–432. Springer, Berlin, 2005.

[24] J.M. Maciejowski. *Predictive control with constraints*. Prentice Hall, Harlow, 2002.

[25] P. M. Marusak. A numerically efficient fuzzy MPC algorithm with fast generation of the control signal. *International Journal of Applied Mathematics and Computer Science*, 31(1):59–71, 2021.

[26] M. A. Müller and K. Worthmann. Quadratic costs do not always work in MPC. *Automatica*, 82:269–277, 2017.

[27] K. R. Muske and J. B. Rawlings. Model predictive control with linear models. *AIChE Journal*, 39(2):262–287, 1993.

[28] R. Nebeluk and M. Ławryńczuk. Fast nonlinear model predictive control using a custom cost-function: Preliminary results. In *The 30th Mediterranean Conference on Control and Automation (MED)*, pages 13–18, Vouliagmeni, Greece, 2022.

[29] R. Nebeluk and P. M. Marusak. Efficient MPC algorithms with variable trajectories of parameters weighting predicted control errors. *Archives of Control Sciences*, 30(2):325–363, 2020.

[30] A. Perez and Y. Yang. Offset-free ARX-based adaptive model predictive control applied to a nonlinear process. *ISA Transactions*, 123:251–262, 2022.

[31] B. B. Schwedersky, R. C. C. Flesch, and H. A. S. Dangui. Practical nonlinear model predictive control algorithm for long short-term memory networks. *IFAC–PapersOnLine*, 52(1):468–473, 2019.

[32] M. Stogiannos, A. Alexandridis, and H. Sarimveis. Model predictive control for systems with fast dynamics using inverse neural models. *ISA Transactions*, 72:161–177, 2018.

[33] H. Sun, T. Zou, J. Liu, and M. Wang. Double-layer model predictive control integrated with zone control. *ISA Transactions*, 114:206–216, 2021.

[34] P. Tatjewski. *Advanced control of industrial processes, structures and algorithms*. Springer, London, 2007.

[35] L. Wang, X. Jiang, Z. Zhang, and Z. Wen. Lateral automatic landing guidance law based on risk-state model predictive control. *ISA Transactions*. In press.

[36] J. Yu, P. Shi, J. Liu, and C. Lin. Command-filtered neuroadaptive output-feedback control for stochastic nonlinear systems with input constraint. *IEEE Transactions on Cybernetics*. In press.

[37] J. Yu, P. Shi, J. Liu, and C. Lin. Neuroadaptive finite-time control for nonlinear MIMO systems with input constraint. *IEEE Transactions on Cybernetics*, 52:6676–6683, 2022.

[38] Q. Yuan, J. Zhan, and X. Li. Outdoor flocking of quadcopter drones with decentralized model predictive control. *ISA Transactions*, 71:84–92, 2017.

[39] A.C. Zanin, M. Tvrzská de Gouvêa, and D. Odloak. Integrating real-time optimization into model predictive controller of the FCC system. *Control Engineering Practice*, 10(8):819–831, 2002.