Master's thesis

Physics-Consistent Surrogate Models with Uncertainty Quantification for Model Predictive Control

Jan-David Ridder
Matr. 217599

First examiner:	Prof. Dr.-Ing. Sergio Lucia
Second examiner:	Prof. Dr.-Ing Hannsjörg Freund
Advisor:	Niklas Kemmerling, M.Sc.
Year:	2025
Ident.:	BCI-PAS-2025-M01

Technische Universität Dortmund
Faculty of Biochemical and Chemical Engineering
Laboratory for Process Automation Systems

technische universität dortmund

process automation systems

Eidesstattliche Versicherung

(Affidavit)

Name, Vorname
(surname, first name)

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

☐ Masterarbeit
(Master's thesis)

Titel
(Title)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**

Kurzzusammenfassung

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Abstract

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

Notation

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{a}_{[1:N]}$	A sequence of vectors from 1 to N, such that $\boldsymbol{a}_{[1:N]} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N)$
$\boldsymbol{A}$	A matrix
A	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \ldots, 0, 1, 0, \ldots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \ldots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b

$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1.
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{A}$
$\mathbf{A}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Linear Algebra Operations

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A}^+$	Moore-Penrose pseudoinverse of $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathsf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$

$\int f(\boldsymbol{x})d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x})d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \perp \mathrm{b}$	The random variables a and b are independent
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	They are conditionally independent given c
$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x}\sim P}[f(x)]$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P\|Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$

Datasets and Distributions

p_{data}	The data generating distribution
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set
$\mathbb{X}$	A set of training examples
$\boldsymbol{x}^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

List of Figures

List of Tables

Chapter 1

Introduction

In the chemical process industry, system models are a fundamental tool for design and operation. Due to its techno-scientific nature, chemical engineering is about modelling physical and chemical phenomena in an accurate enough manner to realize a save and profitable conversion process. These models vary in complexity and prediction potency from single equations such as the Bernoulli equation to detailed partial differentail equation systems such as multi-phase Navier-Stokes. Physico-chemical models are at the core of optimal design or optimal control of most unit operations.
As these *first principle models* rely on intrinsic parameters such as activation energies, friction coefficients, etc. that are usually fitted by experiments, they exhibit some kind of prediction uncertainty. This uncertainty is heavily tied to the measurement uncertainty of the underlying experiments and the non-linearity in the propagation by the model. For example parametric uncertainties of ± 50 % can occur in multi-component reaction systems [1]. As most phenomena in chemical engineering exhibit strong non-lineatities, such as reaction kinetics or activity coefficients, uncertainty quantification represents a great challenge.
Model Predictive Control (MPC) is a potent optimal control algorithm that allows the optimal operation of various systems. The algorithm optimizes a custom loss or reward function under a set of constraints and the dynamics of the system model. The optimal inputs of the system are determined at every time instant as the solution of the optimization. MPC is able do deal with uncertain parameter scenarios in the form of *multi-stage* MPC, where all possible scenarios are optimized over simultaneously. This leads to tremendous computational effort especcialy for detailed system models. The computational effort growths multiplicatory with the amount of parameters and exponentially with the robust horizon [1]. These optimization tasks are impossible to solve in real time with current hardware.
Neural Networks (NNs) are able to approximate any continuous function that maps from a finite dimensional space to another up to an arbitrary small error [1]. Thus, NNs are a key component of machine learning advancements of the 2010s enabling data approximation of almost any kind such as autoregressive language generation [2], protein folding [1] or reduction of partial differential equations (PDEs). Their universal approximation ability, differentiatability and fast evaluation makes NNs greatly suitable to reduce first principle models in the context of optimal control.
For NNs to be applied successfully and savely in process engineering, another measure is of great importance – The alignment of predictions with physical conservation laws such

as mass or energy. By definition, NN predictions seek to minimize a loss function to the ground truth data. The aligment of the predictions with real balance equations is only achieved until test accuracy. Deviations accumulate especcialy in propagation scenarios where predictions are propagated through different networks or through a time horizon in autoregressive manner. This leads to bad predictions or even unstable behavior in late propagation stages and makes MPC implementations challenging.
This work investigates the usage of physics-consistent (PC) NNs in the context of robust model predictive control under parametric uncertainty. The NNs in focus shall significantly reduce the computational effort due to different parameter scenarios as well as guarantee physical correctness of the predictions over the whole prediction horizon. To account for uncertainties, quantile regression is utilized which captures a prediction confidence intervall. PC is enforced by implementing a recent type of activation function that elegantly maps a prediction into the allowed output space without altering the inference time [3].
The developed surrogate models are later evaluated regarding their simulation accuracy using a fictional plug flow tubular reactor (PFTR) model of the ethene oxide synthesis.

Chapter 2

Model Predictive Control under Uncertainty

Before diving into the application of a special model predictive (MPC) algorithm. Uncertain optimization is treated more generally. From economics, over self-driving cars to chemical plants, optimization problems are formulated in terms of uncertain variables in countless variations. These uncertainties arise in three different types – fluctuating disturbances such as wind or weather conditions, parametric offsets such as wrongly determined friction coefficients or model mismatches like unseen side reactions. As the straight maximization or minimzation of an objective often puts a problem to its boundaries, unforeseen disturbances may lead to *constraint violation* or even *instability* [4]. Therefore, *robust* optimization formulations are needed, that are proof against mentioned challenges as well as retain the *performance* with respect to the objective function.

2.1 Robust Optimization

Considering the following optimization task (2.1) under an uncertainty vector $\boldsymbol{w}$.

Definition 2.1. Semi-infinite program with the uncertainty $\boldsymbol{w}$.

$$\begin{aligned} \min_{\boldsymbol{u}} \max_{\boldsymbol{w}} \quad & F_0(\boldsymbol{u}, \boldsymbol{w}) \\ \text{s.t.} \quad & F_i(\boldsymbol{u}, \boldsymbol{w}) \leq 0 \quad \forall \boldsymbol{w} \in \mathbb{W} \end{aligned}$$

Here, the optimization of the objective is formulated in a bilevel form. The scalar objective function F_0 is to be minimized w.r.t the decision variables $\boldsymbol{u}$ for the worst case scenario w.r.t the disturbance vector $\boldsymbol{w}$. It can be interpreted as a player-vs-player setup [4]. One player (the optimizer) seeks a minimum solution for the input $\boldsymbol{u}$, while the opposing player (nature) chooses a maximum solution for the disturbance variables $\boldsymbol{w}$. The inequality constraints F_i in addition, should be satisfied for all possible disturbances, that lie in a defined set $\mathbb{W}$. If this set is not finite, the number of possible constraints to be satisifed is infintely big. Thus, this formulation is named as *semi-infinite-program* [5]. To reduce the number of constraints, the worst-case estimate can also be applied here (2.2).

Definition 2.2. Bi-level formulation of the semi-infinite program

$$\begin{aligned} &\min_{\boldsymbol{u}} \max_{\boldsymbol{w}} \quad F_0(\boldsymbol{u}, \boldsymbol{w}) \\ \text{s.t.} \quad &\varphi_i(\boldsymbol{u}) \coloneqq \max_{\boldsymbol{w}} \quad F_i(\boldsymbol{u}, \boldsymbol{w}) \leq 0 \quad \boldsymbol{w} \in \mathbb{W} \end{aligned}$$

Now, the inequality constraints are also formulated in a bi-level manner. The maximum of F_i is constant in $\boldsymbol{w}$ and only a function of the decision variables $\boldsymbol{u}$. The function φ_i exactly describes the wost-case for F_i parametrized by $\boldsymbol{u}$. In fact, if certain assumptions hold regarding the uncertainty set $\mathbb{W}$ and the concavity of the maximization, φ_i can be exactly computed [1]. This would simplify the problem dramatically. In reality however, an analytical solution for φ_i is not possible to compute. But with the universal approximation capability of neural networks (NNs) in mind [6], a data-driven approach is promising, that delivers an approximated function $\tilde{\varphi}_i(\boldsymbol{u})$.

2.2 Model Predictive Control

Model predictive control (MPC) is a type of optimization, in which an optimal control problem is solved in a dynamic manner. The main goal of this control strategy is to find the best inputs to a system that maximize or minimize a defined objective. In a chemical process, for instance, the energy consumption for heating should be minimized. It should be minimized in such a way, that other variables remain within their bounds. For example, the product specification and, most importantly, the safety of the process needs to be ensured under all circumstances. In contrast to the *optimal design* of a process, that is done during the engineering phase, an MPC controller operates *online*. It acts on the process while it is running. Thus, it is coined *dynamic optimization*. To do so, the optimization problem needs to solved recursively at given time instants. Chemical plants usually have large *time constants*, meaning, they react slowly to their inputs. Other physical systems with low time constants, like quadrocopters behave much faster. So, the computation time of an MPC algorithm is to be kept low [7] to be able to react to the system. To predict the future behavior of the system, MPC requires a system model.

In total, an MPC controller is based on three pillars – an objective function, a set of constraints and, crucially, the model of the system.

Definition 2.3. Continuous-time formulation of an MPC problem.

$$\begin{aligned} &\min_{\boldsymbol{u}} \quad J(t_0, t_{\text{end}}, \boldsymbol{x}, \boldsymbol{u}) \\ \text{s.t.} \quad &\dot{\boldsymbol{x}} = f(t, \boldsymbol{x}, \boldsymbol{u}) \\ &F_i(t, \boldsymbol{x}, \boldsymbol{u}) \leq 0 \quad \forall t \in [t_0, t_{\text{end}}] \\ &\boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}} \end{aligned}$$

Optimal system inputs $\boldsymbol{u}$ are to be determined, that minimize the objective function J. This objective varies with the inputs and the states of the system $\boldsymbol{x}$. It is to be minimized over a time horizon from t_0 to t_{end}, called the *prediction horizon*. The states of the system evolve along the dynamics of the system, which are described by the system model $f(t, \boldsymbol{x}, \boldsymbol{u})$. Thus, the quality of the controller heavily depends on the prediction accuracy of this model [8]. The system states however should remain in a set of

constraints at all times whereas the inputs are bounded, all summarized by the constraint functions F_i. In a physical sense, constraint violation can lead to instability and thus damage of the real system. Here is a vital example. For a chemical reactor a *reaction runaway* must be avoided at all costs. This happens, if the exothermic heat release exceeds the maximum possible cooling rate. Higher temperatures increase the reaction rate exponentially leading to a greater enthalpy release, which again causes a temperature rise. That is why, stability and guaranteed mathematical constraint satisfaction are a great field of research in the discipline of optimal control **<empty citation>**

2.3 Multi-Stage Model Predictive Control

Coming from this general perspective on robust optimization and basic MPC, *robust MPC* specializes *robustness* to a dynamic closed-loop control environment.

Definition 2.4. Continuous-time formulation of an MPC task under uncertainty.

$$\begin{aligned} & \min_{\boldsymbol{u}} \max_{\boldsymbol{w}} \quad J(t_0, t_{\text{end}}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \\ \text{s.t.} \quad & \dot{\boldsymbol{x}} = f(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \\ & F_i(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \leq 0 \quad \forall t \in [t_0, t_{\text{end}}],\ \forall \boldsymbol{w} \in \mathbb{W} \\ & \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}} \end{aligned}$$

The only difference to the basic MPC formulation (2.3) is that the model, the objective and the constraints are functions of the uncertainty, denoted with $\boldsymbol{w}$. This MPC formulation (def. 2.4) is now infinite in time and in the uncertain parameters. To make computation feasible, the problem can be discretized in the time dimension eiher using single shooting, multiple shooting or orthogonal collocation on finite elements [9]. One possibility to roughly discretize the uncertain parameter set $\mathbb{W}$ is done by *multi-stage-MPC* [10].

It generates a scenario tree (fig. **??**) for all possible combinations of disturbances. If the uncertainty is discrete, this formulation is exact. In most cases however, when the uncertainties lie in a continuous space, the scenario tree is an extremely rough approximation. With a non-linear system, the scenarios are not monotonous regarding the disturbance scenarios. This means, even if the scenario branches cover the extreme values of the disturbances themselves, they can leave extreme scenarios caused by intermediate parameter combinations. Robust constraint satisfaction is not guaranteed and must be checked via a reachability analysis [10]. Another drawback of multi-stage MPC is, that it includes redundant scenarios not causing extreme cases due to the non-monotonicity. These scenarios still are part of the optimization leading to unnecessary complexity. On top of this, the number of optimization variables growths exponentially with the *robust horizon* N_{r}, the number of parameters and their respective values (eq. 2.1) [1].

$$n_{\text{s}} = \left(\prod_{i}^{n_{\text{w}}} \nu_i \right)^{N_{\text{r}}} \tag{2.1}$$

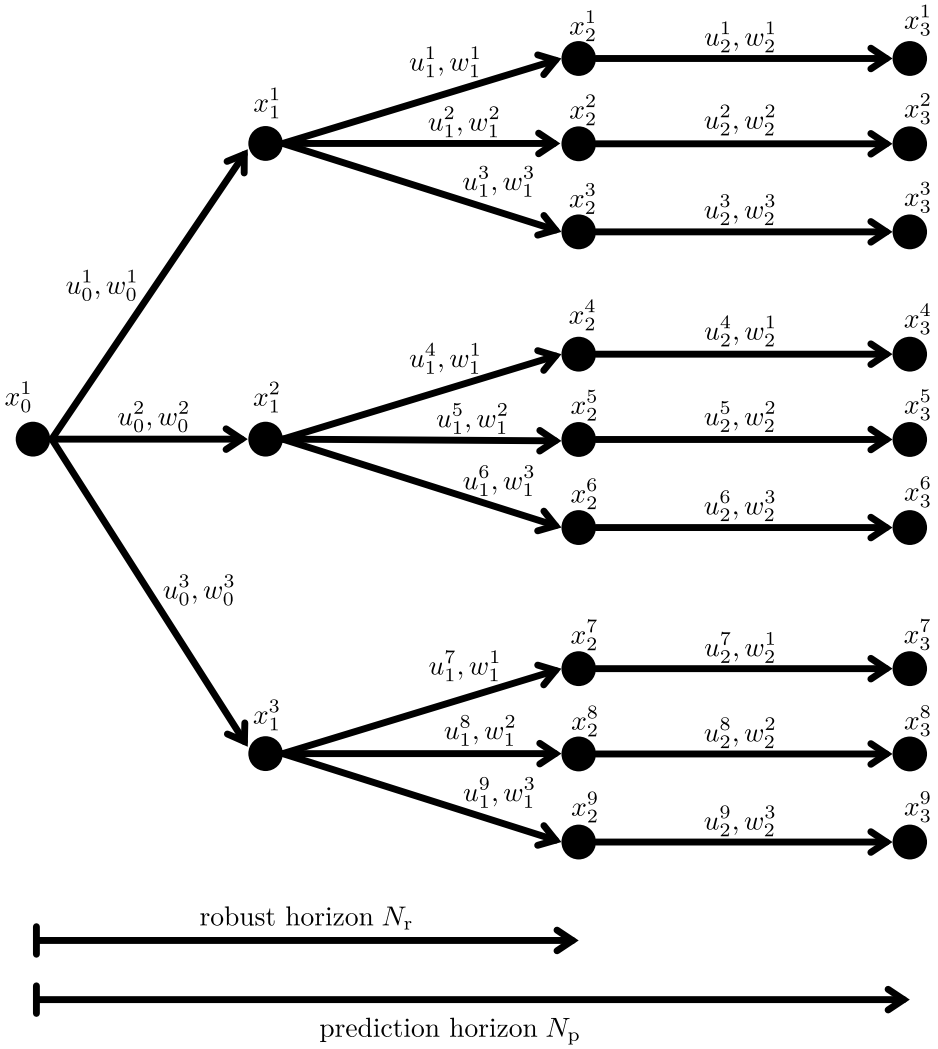

Figure 2.1: Multi-stage MPC discretizies the uncertainty set via user-defined possible cases $w_k^{(\cdot)}$. All possible case combinations are branched through the robust horizon N_r. To limit the complexity, the uncertainties are considered constant thereafter for the rest of the prediction horizon N_p.

Here, n_w denotes the number of different disturbances and ν_i their respective discrete values. n_s describes the number of scenarios in the scenario tree (fig. **??**).

2.4 Conservativeness and Performance

2.5 Neural Networks

Feed forward neural networks (NNs), sometimes referred to as *multi-layer-perceptrons*, provide a stunning capability of approximating any multivariate function from any finite dimensional space to another. According to the *Universal approximation theorem*, any finite dimensional function can be reproduced if the number of hidden layers and respective neurons is high enough, up to an arbitrarily small error [6]. Thus, NNs play a major role in recent developments in machine learning. The breakthrough of large language models such as ChatGPT heavily rely on neural networks [2]. NNs are not only succesfully applied in natural language processing but in the chemical process industry. Applications range from thermodynamic property prediction over unit operation design to corporate level decision making [11]. Their ability to approximate any non-linear function while beeing differentiable, makes NNs especcialy suitable for dynamic optimization such as MPC. In contrast to detailed first principle models, NNs have the advantage of making a prediction in a fraction of the inference time of the first principle model. As MPC operates in real time parallel to the system to be controlled, the computation duration is a key measure for a succesfull MPC implementation. That is why, surrogate models in form of NNs are a crucial tool for dynamic optimization and heavily investigated in current research [12]. May a NN be given as an alternating series of function evaluations (def. 2.5).

Definition 2.5. Mathematical formulation of a neural network as a nested function evaluation.

$$\begin{aligned} \mathcal{NN}(\boldsymbol{\xi}, \boldsymbol{\theta}) &= g_{l+1} \circ h_{l+1} \circ ... \circ g_1 \circ h_1(\boldsymbol{\xi}) \\ h(\boldsymbol{a}_{l-1}) &= \boldsymbol{\theta}_{Wl} \boldsymbol{a}_{l-1} + \boldsymbol{\theta}_{bl} \\ \boldsymbol{a}_l &= g_l(h(\boldsymbol{a}_{l-1})) \end{aligned}$$

In fact, a NN is a series of function combinations denoted by the $\circ$-symbol alternating between a linear function $h(\boldsymbol{a})$ and a *non-linear* activation function $g(h(\boldsymbol{a}))$. The linear transformation matrices $\boldsymbol{\theta}_{Wl}$ and offsets $\boldsymbol{\theta}_{bl}$, are called weights and biases respectively and are the trainable parameters of a NN. The activation functions g are important to introduce non-linearity into every linear projection. Otherwise, the complete function combination can be written as one singular linear transformation and the universal approximations capability is lost. Given a set of training data, the optimal parameters for $\boldsymbol{\theta}$ are found by solving an optimization task that involves the minimzation of a prediction loss function (eq. 2.2).

$$\begin{aligned} \mathbb{X}_\mathrm{train} &\coloneqq \{\boldsymbol{\xi}^{(0)}, \ldots, \boldsymbol{\xi}^{(m)}\} \\ \mathbb{Y}_\mathrm{train} &\coloneqq \{\boldsymbol{y}^{(0)}, \ldots, \boldsymbol{y}^{(m)}\} \\ \mathcal{L}(\boldsymbol{\theta}, \mathbb{X}_\mathrm{train}, \mathbb{Y}_\mathrm{train}) &= \frac{1}{m} \sum_{i=0}^{m-1} \left|\left| \boldsymbol{y}^{(i)} - \tilde{\boldsymbol{y}}(\boldsymbol{\theta}, \boldsymbol{\xi}^{(i)}) \right|\right|_2^2 \end{aligned} \tag{2.2}$$

In *supervised learning*, the training data is separated in *features* $\mathbb{X}_{\text{train}}$, the inputs of the NN, and *labels* $\mathbb{Y}_{\text{train}}$, the outputs of the NN. During training, the parameters $\boldsymbol{\theta}$ are determined such that the NN approximates the data in best possible manner. In this example the loss function $\mathcal{L}$ portrays the mean squarred error (MSE), that is used in most regression tasks. The difference between the model prediction, denoted as $\tilde{\boldsymbol{y}}(\boldsymbol{\xi}^{(i)})$, and the true data point $\boldsymbol{y}^{(i)}$ is penalized quadratically.

2.5.1 Quantile Regression

By utilizig a loss function in form of the MSE, a model is trained to achieve regression predictions in form of the quarred mean. To tweak the prediction characteristics of a NN for the task at hand, the type of training loss function is decisive. In a probabilistic environment, the prediction of confidence intervalls in the form of quantiles may be of mayor interest. To learn the quantiles of a distribution of data points, the *quantile loss* or *pinball loss* can be used (eq. 2.3).

$$\mathcal{L}_\tau = \begin{cases} \tau \cdot (y - \tilde{y}(\boldsymbol{\theta})), & y \geq \tilde{y}(\boldsymbol{\theta}) \\ (1-\tau) \cdot (\tilde{y}(\boldsymbol{\theta}) - y), & y < \tilde{y}(\boldsymbol{\theta}) \end{cases} \tag{2.3}$$

The hyperparameter τ denotes the type and therefore position of the quantile. A small value for τ, e.g. 0.05 or 0.1, strongly penalizes data points that are below the threshold $y < \tilde{y}(\boldsymbol{\theta})$ via the $(1-\tau)$-term to keep the quantile small. A high value for τ, e.g. 0.95, focuses on the penalization of data points above the threshold $y \geq \tilde{y}(\boldsymbol{\theta})$ to cover a greater amount of points. For robust optimization, *quantile regression* can be applied to learn upper or lower bounds for the uncertainty-dependent constraints $F_i(\boldsymbol{u}, \boldsymbol{w})$ (eq. 2.2) in terms of confidence quantiles. This would reduce the semi-infinite program to an uncertainty-finite-program.

2.6 NARX Models

Because of their universal approximation capability [6] and their differentiability, NNs are well suited as *surrogate models* in MPC algorithms to approximate the system behavior represented by the function f (def. 2.3). This can significantly reduce the solution time in contrast to discretized systems of ordinary differential equations (ODEs) **<empty citation>**

Definition 2.6. Time-discrete form of a NN to predict the future state, based on past information contained in the vector $\boldsymbol{\xi}_k$.

$$\begin{aligned} \boldsymbol{x}_{k+1} &= f(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \tilde{\boldsymbol{x}}_{k+1} &= \mathcal{NN}(\boldsymbol{x}_k, \boldsymbol{x}_{k-1}, ..., \boldsymbol{x}_{k-h}, \boldsymbol{u}_k, \boldsymbol{u}_{k-1}, ..., \boldsymbol{u}_{k-h}) \end{aligned}$$

In the discrete-time notation, the state at the next time instant $\boldsymbol{x}_{k+1}$ is calculated by propagating the current state $\boldsymbol{x}_k$ trough a function $f(\boldsymbol{x}_k, \boldsymbol{u}_k)$ that discribes the system dynamic (def. 2.6). In the original MPC formulation (def. 2.3) this is some numerical integration sceme for a system of differential equations [9] with the abbreviation *right-hand-side* (rhs). The state propagation by the function f is a classical example of a time series extrapolation. NNs have shown significant capabilities in time series

forecasting. Deep learning based architectures exist for time series forecasting in various complexities. Convolutional NNs (CNNs), Long-Short-Term-Memory (LSTMs) cells and transformers are the most promiment for time series prediction with varying complexity [2], [13]. These provide a great possibility to apply deep learning based forecasting within optimal control algorithms, where the next state $\tilde{\boldsymbol{x}}_{k+1}$ is the output of a neural network $\mathcal{NN}$. In context of this work, shallow feedforward NNs or *single-layer perceptrons* (SLPs) are used for this purpose which technically do not belong to *deep learning* architectures.

In the realm of control, the non-linear system indendification is summarized with the term *Non-linear AutoRegressive with eXogenous inputs* (NARX). Here, the term *autoregressive* means, that the future prediction is a function of a series of past system states $\{\boldsymbol{x}_k, \boldsymbol{x}_{k-1}, ..., \boldsymbol{x}_{k-h}\}$, while *exogenous* implies, that a series of outside inputs $\{\boldsymbol{u}_k, \boldsymbol{u}_{k-1}, ..., \boldsymbol{u}_{k-h}\}$ is passed to the function as well (def. 2.6). The hyperparameter h denotes the length of the time horizon of the past inputs. A high value for this parameter includes more past information, which possibly allows better predictions while increasing the network input. Between states and inputs, h can be choosen differently. In this work, h is set equal for states and inputs.

2.6.1 Input Compression

$$\boldsymbol{\phi}_k \coloneqq (\boldsymbol{x}_k, \boldsymbol{x}_{k-1}, ..., \boldsymbol{x}_{k-h}, \boldsymbol{u}_k, \boldsymbol{u}_{k-1}, ..., \boldsymbol{u}_{k-h}) \tag{2.4}$$

The concatenated input feature vector $\boldsymbol{\phi}_k$ contains all past states and inputs within the time horizon h, leading to a vector length of $(n_x + n_u) \cdot (h + 1)$ where n_x and n_u represent the number of states and inputs respectively. For large systems, the input length increases the network size unaccetably quick while the input time serieses often exhibit strong redundacies. To reduce the input size, the feature vector $\boldsymbol{\phi}_k$ is passed to an *encoder* [14]. The encoder is based on feature compression via *principal component analysis* (PCA), that maps the feature space to a lower dimensional subspace. This subspace represents the space of the input data with the most information.

2.6.2 Singular Value Decomposition

The *singular value decomposition* (SVD) is a fundamental building block in linear algebra. Its applications are extremely versatile from image compression, over exploratory data analysis to model reduction in fluid mechanics. The power of the SVD is, that it maps a high dimensional space into a low dimensional subspace. The mapping is achieved by a simplistic matrix transformation. It not only identifies a subspace, it also identifies underlying patterns in the data encoded via the respective *singular vectors*. The compression and component indendification is guaranteed to exist for every matrix $\boldsymbol{X} \in \mathbb{C}^{n \times m}$ in contrast to an eigendecomposition [15].

The SVD is extremely useful, when high dimensional data is presented, that embeds some kind of pattern. For example, simulations of partial differential equations (PDEs), images as collections of pixels and time series exhibit strong patterns within the data. This is especcialy true for data points in close temporal or spatial distance. Because the SVD is the basis for PCA to compress the feature vector $\boldsymbol{\phi}_k$ and *model order reduction*

(MOR) of ODEs or PDEs it is explained in further detail. According to the SVD, any matrix $\boldsymbol{X} \in \mathbb{C}^{n \times m}$ can be decomposed into three submatrices (th. 2.1).

Theorem 2.1. *Singular value decomposition of a data matrix* $\boldsymbol{X}$*.*

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$$

Here, the matrices $\boldsymbol{U} \in \mathbb{C}^{n \times n}$ and $\boldsymbol{V}^* \in \mathbb{C}^{m \times m}$ are quadratic, orthonormal matrices. The real, non-negative, diagonal matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times m}$ contains the so called *singular values* of the matrix with the biggest as the top-left entry. The notation $\boldsymbol{V}^*$ stands for the transposed complex-conjugate of $\boldsymbol{V}$. The computation in `python` can be done in `numpy` via the method `linalg.svd()` or directly in a training environment with `torch.linalg.svd()`. For large data matrices $\boldsymbol{X}$, that do not fit into memory, a batched or incremental SVD must be considered. A current `torch` implementation with GPU support exists under `torch_incremental_pca` [16].

The compression itself is done by truncating the

Chapter 3

Surrogate Modelling

3.1 Physics-Constrained Neural Networks

By definition, NNs are designed to reproduce their training data in a best possible manner, defined by the characteristics of the loss function used during the training process. NN predictions minimize the loss towards the training data seen beforehand. Circumstances exist, in which the output of a NN should strictly satisfy a certain set of equations. In case of physical modeling, the prediction of a surrogate should follow the same conservation laws as a first principle model. Especcialy, when predicions are propagated through several NNs or simulated in an autoregressive manner, errors accumulate. Final predictions are untrustworthy or simulations exhibit instability [3]. As a MPC algorithm needs a reasonable approximation of the system behavior over the complete prediction horizon, the accumulation of conservation violations become an issue.

To realize physics consistent NN outputs, several strategies exists. Physics-informed neural networks (*PINN*s) for instance, embed physical balance equations into the loss function similar to a regularization term [17]. This serves as a soft constraint and comes with two downsides. The NN output does not strictly follow the constraint equation, because the physical consistency is fullfilled up to the trade off between data fit and constraint satisfaction. Every new balance equation to be included introduces a new hyperparameter worsening the ability to include network parameter regularization. To guarantee strict constraint satisfaction and thus perfect physics consistency up to machine precision, a new type of activation function is presented equivalently to the work by Chen et al. [3]. This activation function serves as a linear projection from the NN output space into the physical allowed subspace without adding neither learnable parameters nor hyperparameters. Let the output of a NN be defined as well as a set of linear equality contraints that summarize some physical conservation quantity (def. 3.1).

Definition 3.1. Crude NN output with a given set of linear equality constraints.

$$\tilde{\boldsymbol{y}}(\boldsymbol{\theta}, \boldsymbol{\xi}) \coloneqq \mathcal{NN}(\boldsymbol{\theta}, \boldsymbol{\xi})$$
$$\boldsymbol{A}(\boldsymbol{y} - \boldsymbol{z}_0) = \boldsymbol{b}$$
$$\text{with} \quad \tilde{\boldsymbol{y}}(\boldsymbol{\theta}, \boldsymbol{\xi}), \boldsymbol{z}_0 \in \mathbb{R}^{n_\text{out}} \quad \boldsymbol{A} \in \mathbb{R}^{n_\text{c} \times n_\text{out}} \quad \boldsymbol{b} \in \mathbb{R}^{n_\text{c}}$$

$\tilde{\boldsymbol{y}}$ denotes the NN output, $\boldsymbol{\xi}$ the input and the matrices $\boldsymbol{A}$ and $\boldsymbol{b}$ portray the feasible space of points. The variable $\boldsymbol{z}_0$ illustrates an external input into the constraint. This

may be a NN input, a constant, or another batched reference point. In a physical sense, $\boldsymbol{z}_0$ is often introduced as a boundary or initial condition. A feasible point $\boldsymbol{y}$ now needs to be determined that fullfills the set of linear equality constraints. For a straight forward implementation, the formulation appears in such a way that it supports batching of a certain batch size $n_\mathcal{B}$. n_out and n_c describe the number of output features and numbers of linear equality constraints respectively. The main idea is, to find a point in the feasible subspace that is closest to the NN prediction. This is forced by following quadratic programm (QP) (def. 3.2).

Definition 3.2. Minimization of the euclidian distance towards the NN output as a linear equality constrained optimization that supports batching.

$$\begin{aligned} \boldsymbol{y}^* &= \underset{\boldsymbol{y}}{\operatorname{argmin}} ||\boldsymbol{y} - \tilde{\boldsymbol{y}}||_2^2 \\ \text{s.t.} \quad & (\boldsymbol{y} - \boldsymbol{z}_0)\boldsymbol{A}^\top - \boldsymbol{b} = \boldsymbol{0} \end{aligned}$$

This means, the output of the network $\tilde{\boldsymbol{y}}$, the optimal solution $\boldsymbol{y}^*$ and $\boldsymbol{z}_0$ are matrices of dimension $n_\mathcal{B} \times n_\text{out}$. Computing frameworks that support broadcasting such as `numpy` or `pytorch` allow the implementation of $\boldsymbol{z}_0$ in the shape of $1 \times n_\text{out}$ to reduce memory usage as long as it remains constant [1]. The transposed constraint matrix $\boldsymbol{A}^\top$ must be of shape $n_\text{c} \times n_\mathcal{B}$. The vector $\boldsymbol{b}$ represents the offset of the linear equality constraints. This optimization task can be solved analytically by setting up the *Karush-Kuhn-Tucker* (KKT) conditions (th. 3.1).

Theorem 3.1. *Langrangian and KKT conditions of the QP*

$$\begin{aligned} \mathcal{L}(\boldsymbol{y}, \boldsymbol{\nu}) &= ||\boldsymbol{y} - \tilde{\boldsymbol{y}}||_2^2 + \boldsymbol{\nu}^\top (\boldsymbol{A}(\boldsymbol{y} - \boldsymbol{z}_0)^\top - \boldsymbol{b}) \\ \nabla_{\boldsymbol{y}} \mathcal{L}(\boldsymbol{y}, \boldsymbol{\nu}) &= 2(\boldsymbol{y}^* - \tilde{\boldsymbol{y}}) + \boldsymbol{\nu}^\top \boldsymbol{A} \overset{!}{=} 0 \\ \nabla_{\boldsymbol{\nu}} \mathcal{L}(\boldsymbol{y}, \boldsymbol{\nu}) &= (\boldsymbol{y}^* - \boldsymbol{z}_0)\boldsymbol{A}^\top - \boldsymbol{b} \overset{!}{=} 0 \end{aligned}$$

The Lagrangian $\mathcal{L}$ arises as the sum of the objective function and all linear equality constraints. The vector $\boldsymbol{\nu}$ contains the dual variables of the linear equality constraints. The optimality conditions lead to a linear system of equations (cor. 3.1).

Corollary 3.1. *The optimal solution in the feasible space as projection of an activation function*

$$\begin{pmatrix} \boldsymbol{y}^* & (\boldsymbol{\nu}^*)^\top \end{pmatrix} \begin{pmatrix} 2\boldsymbol{I} & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} 2\tilde{\boldsymbol{y}} & \boldsymbol{z}_0\boldsymbol{A}^\top + \boldsymbol{b} \end{pmatrix}$$

$$g_{pc}(\tilde{\boldsymbol{y}}, \boldsymbol{z}_0) \coloneqq \begin{pmatrix} 2\tilde{\boldsymbol{y}} & \boldsymbol{z}_0\boldsymbol{A}^\top + \boldsymbol{b} \end{pmatrix} \begin{pmatrix} 2\boldsymbol{I} & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0} \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{y}^* & (\boldsymbol{\nu}^*)^\top \end{pmatrix}$$

The optimal solution denoted with "*" is directly yielded by projecting the NN output $\tilde{\boldsymbol{y}}$ using the additional input $\boldsymbol{z}_0$ and a projection matrix. This projection matrix only contains the constraint matrix $\boldsymbol{A}$ can be computed before runtime which drastically speeds up forward and backward passes in contrast to implicit methods [18]. The solution elegantly serves as an activation function $g_\text{pc}(\tilde{\boldsymbol{y}}, \boldsymbol{z}_0)$, that maps the crude NN

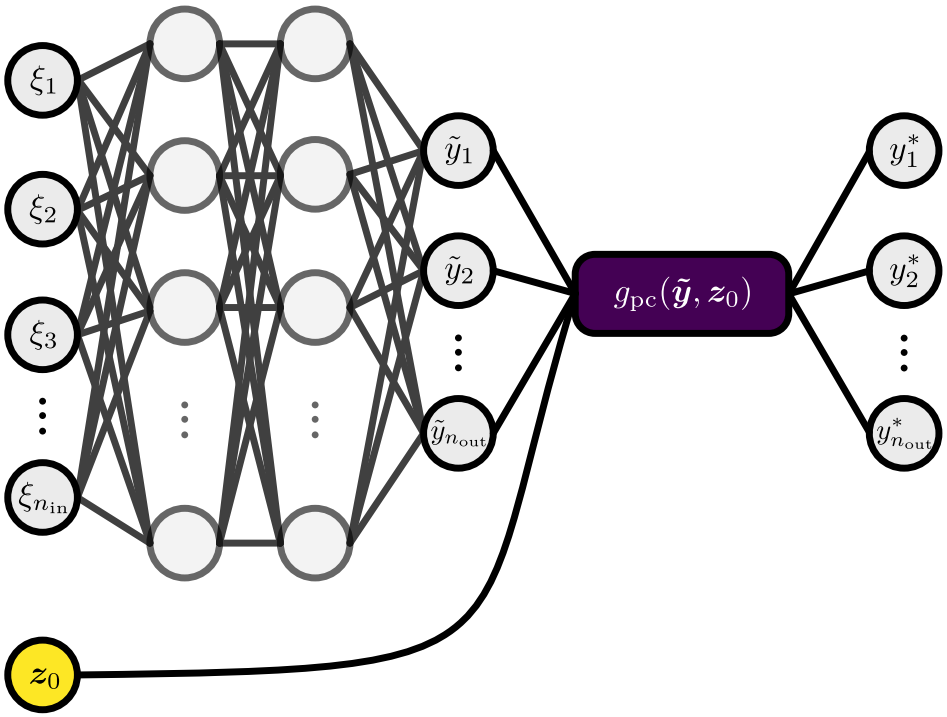

Figure 3.1: The physics constrained NN turns the input vector $\boldsymbol{\xi}$ into the crude output $\tilde{\boldsymbol{y}}$ via a feedforward NN, which is the projected by the physics constrained activation function g_{pc} into the feasible output $\boldsymbol{y}^*$. It allows to include data points such as boundary conditions as needed in the form of $\boldsymbol{z}_0$

output into a subspace defined by $\boldsymbol{A}$ and $\boldsymbol{b}$. The final output for $\boldsymbol{y}^*$ satisfies the linear equality constraints by definition, while it is located as close to the crude output $\tilde{\boldsymbol{y}}$ as possible. This is not only applicable in the field of chemical engineering but a general formulation that allows to integrate linear equality constraints into NNs. Additionally, the type of network architecture is independent of the mapping. It can be appended to any network type such as recurrent NNs, convolutional NNs or transformer-based architectures.

Due to its general applicability, the activation function could be used within a network to correct intermediate values. In most use cases howerver, it is applied as a final correction function at the output (fig. 3.1). The input $\boldsymbol{\xi}$ is propagated through a feedforward NN with the physics constrained activation function behind the last layer (purple). It takes the reference value $\boldsymbol{z}_0$ (yellow) without beeing forwarded through the network. In recent literature, NNs that utilze such a type of activation function can be found as *KKT-hPINN* [3], which is short for *Karush-Kuhn-Tucker-hard-physics-informed-neural-network*. For later reference, this activation function g_{pc} will be referred to as *physics constrained activation* (cor. 3.1) to have a short name in the context of chemical engineering. Furthermore, this procedure could be extended to support arbitrary non-linear

equality constraints by splitting the network output in frozen and unfrozen variables [19]. It enables the integration of enthalpy balances that are of non-linear nature. As this work combines the implementation of physics consistent NNs with parametric uncertainty for MPC, the focus remains on linear equality constraints.

Algorithm 1 Forward pass of the physics constrained activation function.

Require: $\boldsymbol{K} = \begin{pmatrix} 2\boldsymbol{I} & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0} \end{pmatrix}^{-1}$, $\tilde{\boldsymbol{y}}, \boldsymbol{z}_0, \boldsymbol{A}, \boldsymbol{b}$

$\boldsymbol{h} \leftarrow \begin{pmatrix} 2\tilde{\boldsymbol{y}} & \boldsymbol{z}_0\boldsymbol{A}^\top + \boldsymbol{b} \end{pmatrix}$

$\boldsymbol{y}^* \leftarrow \boldsymbol{h}\boldsymbol{K}$

$\boldsymbol{y}^* \leftarrow \boldsymbol{y}^*_{:,:n_{\text{out}}}$

return $\boldsymbol{y}^*$

The forward pass through the physics constrained activation involves the setup of a temporary vector $\boldsymbol{h}$ (alg. 1) and the truncation of the dual variables in the solution. The matrix inversion to yield $\boldsymbol{K}$ should be done offline and then saved as untrainable model parameter. In `pytorch` this is can be done using the `register_buffer()` method. An implementation example of the physics consistent activation can be found in the appendix (XX).

Chapter 4

Results on Surrogate Model Performance

Please choose a more expressive title for this chapter. If it makes sense for your work you can also have multiple chapters presenting your results.

4.1 Training Effectivity

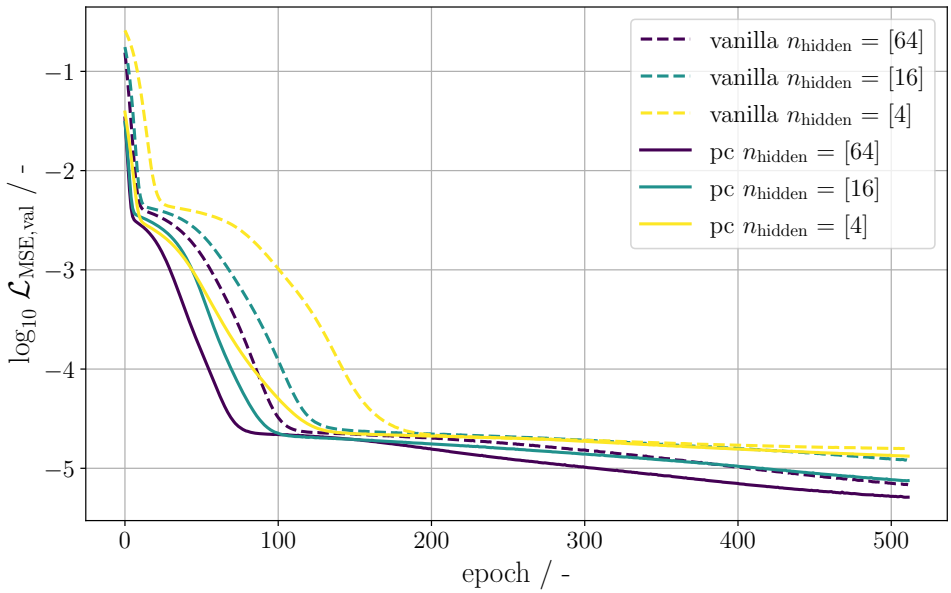

Figure 4.1: The validation loss calculated as MSE $\mathcal{L}_{\text{MSE,val}}$ during the training of the vanilla NARX (dashed line) and the physics-constrained NARX (solid line). The three data rows each indicate *one hidden layer* with $64, 16$ and 4 hidden neurons respectively. The physics-constrained NARX converges visibly faster than the vanilla NARX.

Chapter 5

Conclusions

Appendix A

Datasheets

Did your work require long tables of parameters, e.g. to configure a system model. This table would distract the main part of the thesis but including it is an important aspect of scientific work. Please attach it here.

A.1 Parameter for ...

...

Appendix B

Proofs

Did you introduce lemmas or theorems with lengthy proofs? Including them in the main part of the thesis might be distracting. You can also include them in the appendix.

B.1 Proof of Theorem X

...

Bibliography

[1] JD, *Placeholder citation*, 0.

[2] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[3] H. Chen, G. E. C. Flores, and C. Li, "Physics-informed neural networks with hard linear equality constraints," *Computers & Chemical Engineering*, vol. 189, p. 108 764, 2024. DOI: https://doi.org/10.1016/j.compchemeng.2024.108764.

[4] B. Houska, "Robust optimization of dynamic systems," Ph.D. dissertation, KU Leuven, 2011, ISBN: 978-94-6018-394-2.

[5] O. Stein, "Bi-level methods for semi-infinite programming," in *Bi-Level Strategies in Semi-Infinite Programming*. Boston, MA: Springer US, 2003, pp. 145–169, ISBN: 978-1-4419-9164-5. DOI: 10.1007/978-1-4419-9164-5_5. [Online]. Available: https://doi.org/10.1007/978-1-4419-9164-5_5.

[6] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. DOI: https://doi.org/10.1016/0893-6080(91)90009-T.

[7] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2397–2404, Apr. 2023. DOI: 10.1109/lra.2023.3246839.

[8] S. Lucia and S. Engell, "Robust nonlinear model predictive control of a batch bioreactor using multi-stage stochastic programming," in *2013 European Control Conference (ECC)*, 2013, pp. 4124–4129. DOI: 10.23919/ECC.2013.6669521.

[9] "Dynamic optimization methods with embedded dae solvers," in *Nonlinear Programming*, ch. 9, pp. 251–286. DOI: 10.1137/1.9780898719383.ch9. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719383.ch9. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898719383.ch9.

[10] S. Lucia and S. Engell, "Potential and limitations of multi-stage nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 48, no. 8, pp. 1015–1020, 2015. DOI: https://doi.org/10.1016/j.ifacol.2015.09.101.

[11] K. McBride and K. Sundmacher, "Overview of surrogate modeling in chemical process engineering," *Chemie Ingenieur Technik*, vol. 91, no. 3, pp. 228–239, 2019. DOI: https://doi.org/10.1002/cite.201800091. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cite.201800091.

[12] T. Salzmann, J. Arrizabalaga, J. Andersson, M. Pavone, and M. Ryll, "Learning for casadi: Data-driven models in numerical optimization," in *Learning for Dynamics and Control Conference (L4DC)*, 2024.

[13] K. Benidis, S. S. Rangapuram, V. Flunkert, *et al.*, "Deep learning for time series forecasting: Tutorial and literature survey," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–36, Dec. 2022. DOI: 10.1145/3533382.

[14] N. Kemmerling and S. Lucia, "Uncertainty-aware reduced-order modeling for robust control of zeolite crystallization," Manuscript in preparation, TU Dortmund University, Dortmund, Germany, 2025.

[15] S. L. Brunton and J. N. Kutz, "Reduced order models (roms)," in *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, POD for model reduction, Cambridge University Press, 2019, pp. 375–402.

[16] Anonymous, *Torch-incremental-pca: A pytorch implementation of incremental pca*, https://pypi.org/project/torch-incremental-pca/, PyTorch implementation inspired by scikit-learn's IncrementalPCA and PCAonGPU. Provides GPU-accelerated, memory-efficient incremental PCA for large datasets., 2025.

[17] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. DOI: https://doi.org/10.1016/j.jcp.2018.10.045.

[18] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems*, 2019.

[19] G. Lastrucci, T. Karia, Z. Gromotka, and A. M. Schweidtmann, *Picard-kkt-hpinn: Enforcing nonlinear enthalpy balances for physically consistent neural networks*, 2025. arXiv: 2501.17782 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2501.17782.