Constraining Gaussian Processes to Systems of Linear Ordinary Differential Equations

Andreas Besginow
Department of Electrical Engineering and Computer Science
OWL University of Applied Sciences and Arts
Lemgo, Germany
Institute industrial IT Lemgo, Germany
andreas.besginow@th-owl.de

Markus Lange-Hegermann
Department of Electrical Engineering and Computer Science
OWL University of Applied Sciences and Arts
Lemgo, Germany
Institute industrial IT Lemgo, Germany
markus.lange-hegermann@th-owl.de

Abstract

Data in many applications follows systems of Ordinary Differential Equations (ODEs). This paper presents a novel algorithmic and symbolic construction for covariance functions of Gaussian Processes (GPs) with realizations strictly following a system of linear homogeneous ODEs with constant coefficients, which we call LODE-GPs. Introducing this strong inductive bias into a GP improves modelling of such data. Using smith normal form algorithms, a symbolic technique, we overcome two current restrictions in the state of the art: (1) the need for certain uniqueness conditions in the set of solutions, typically assumed in classical ODE solvers and their probabilistic counterparts, and (2) the restriction to controllable systems, typically assumed when encoding differential equations in covariance functions. We show the effectiveness of LODE-GPs in a number of experiments, for example learning physically interpretable parameters by maximizing the likelihood.

1 Introduction

Many real world tasks have underlying dynamic behavior, for example chemical reactions [22], systems in bioprocess engineering [26], or population dynamics [68]. One currently widely debated application are compartmental models in epidemiology [20, 34, 50]. Many such systems are linear or can be decently linearized, such as in control theory [71], biology [16], process engineering [1, §9], or engines [7]. Including prior knowledge in the form of differential equations benefits the model fit and enhances interpretability for a model. Hence, modelling differential equations in Machine Learning (ML) has therefore been the focus of much research in both Gaussian Processes (GPs) (e.g. [37, 29, 49, 24, 61]) and Deep Learning (DL) (e.g. [47, 36, 62, 17]).

The class of probabilistic Ordinary Differential Equation (ODE) solvers allow to estimate the solution of ODE initial value problems through approximations, often based on e.g. Runge-Kutta methods or Kalman filters [50, 52, 51, 10], and thereby do not strictly guarantee to yield solutions of the ODE. This class of algorithms is commonly used to solve non-linear ODEs, but typically require that systems are well-posed, in particular the solution needs to be unique once a finite number of

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

initial conditions is known. While this second limitation is typically irrelevant in physical or biological systems, it is strongly relevant in systems in engineering, such as in control systems with their freely choosable inputs.

With early works like [25], the introduction of derivatives as linear operators in GPs became prominent for partial differential equations. This inspired several works that encode differential equations in the covariance structure of GPs [29, 49, 24, 61, 46] to introduce a strong inductive bias into GPs. This motivated [37] to make these approaches algorithmic, build a mathematical foundation, and show the hidden assumption in previous works of being limited to controllable systems.

This paper overcomes the necessity of approximations, the restriction to systems where initial conditions lead to unique solutions, and the restriction to controllable systems. For that purpose, we algorithmically construct LODE-GPs (Theorem 1), a novel class of GPs whose realizations strictly follow a given system of homogenuous linear ODEs with constant coefficients.

Our construction of LODE-GPs starts by describing our system of ODEs via a so-called operator matrix A. Calculating its Smith Normal Form (SNF) gives us a decoupled representation of the system via a diagonal operator matrix D, at the cost of going through two invertible base change matrices U, V with the relationship $U \cdot A \cdot V = D$. We can easily construct a multi-output GP $g = \mathcal{GP}(0, k)$ (cf. Lemma 1) with as many outputs as the system channels and whose covariance function encodes the decoupled system. Then, applying the base change matrix V to g yields the pushforward GP $V_* g = \mathcal{GP}(0, VkV')$ whose covariance function encodes the original system, i.e. the LODE-GP. We refer to Theorem 1 and its algorithmic proof for details.

Our paper makes the following contributions:

1. It develops and proves an algorithmic and symbolic construction of LODE-GPs, GPs with the strong inductive bias that their realizations strictly satisfy *any* given system of homogeneous linear ODEs with constant coefficients (see Theorem 1).
2. It demonstrates that the constructed GPs are numerically stable and robust (see Section 5).
3. It automatically includes ODE system parameters as GP parameters, learns them during the training process of the GP, and allows an interpretation of the data (see Subsection 5.2).
4. It provides a free and public implementation based on the GP library GPyTorch[1] [21].

We successfully test our approach on controllable and non-controllable systems of differential equations, with and without system parameters. The LODE-GP strictly satisfies the ODEs and outperforms GPs by several magnitudes in its precision, additionally it correctly reconstructs the system parameters used to generate the data with a small relative error, even despite noise. We also discuss how this strict behavior influences results for metrics like the RMSE with e.g. Figure 4.

2 Related Work

Using physical information in ML systems became a central research topic over recent years in both DL and GPs. With the GP research in two important categories: (1) Works that introduce derivative information as inductive bias into the GP [29, 37, 38, 25, 49, 24] and (2) works that train GPs without inductive bias on physical data, with specially selected standard kernels, as an approximate model for additional processing steps [67, 61, 55, 32, 9, 12, 13]. Of all the works on GPs, [37, 38] are the closest to us, as they also provide general algorithms to introduce inductive bias into GPs, based on Gröbner bases. These approaches are only applicable to controllable systems. By basing our algorithm on the SNF instead of Gröbner bases, we overcome this limitation to controllable systems, but are restricted to *ODEs*.

Probabilistic ODE solvers like [50, 33, 60, 10, 41, 51, 52, 8, 12] are able to include inductive bias of ODEs but sometimes require a number of comparably complex calculations and some approximations. We avoid any approximation through calculation of the SNF and the application of the pushforward with a linear differential operator. Hence, our work is limited to *linear* ODEs, whereas probabilistic ODE solvers are able to work with non-linear ODEs.

For GP priors for decoupled ODE systems with constant coefficients and right hand side functions see [2], which consideres the right hand side functions as latent, hence these models are called Latent

[1] `https://github.com/ABesginow/LODE-GPs`

Force Model (LFM). A GP prior on these latent forces and is pushed forward through differential operators and Green's operator. We empirically compare LODE-GPs to LFMs in the appendix.

Of the other works that introduce inductive bias, [25] and [49] are the earliest works and already discuss a general formulation of differential equations by viewing derivatives as linear operators that can be applied to GPs. Subsequent works target specific ODEs and create covariance functions with inductive bias [24, 61, 15]. They can be considered special cases of our approach with a specific manually constructed pushforward. Finally, others constructed covariance functions for *partial* differential equations from standard covariance functions with e.g. curl-free behaviour [67] to model physical behaviour [67, 61, 55, 32] or use the GP as a surrogate model to learn physical problems like the inverse pole problem [13] or a variety of applications [9].

Most DL models explore physically informed models through additional loss terms that punish solutions that deviate from the system [62, 36, 47], or in the case of [17] show the equivalence of the loss to their assumed physical behavior. This is often combined with different modifications to the networks like additional features [17] or specific network structures [36].

3 Background

3.1 Gaussian Processes

A GP $g = \mathcal{GP}(\mu, k)$ defines a probability distribution over the space of functions $\mathbb{R}^d \to \mathbb{R}^\ell$, such that the outputs $g(x_i)$ at any set of $x_i \in \mathbb{R}^d$ are jointly Gaussian [70]. Such a (multi-output) GP is defined by its mean function (often set to zero)

$$\mu : \mathbb{R}^d \to \mathbb{R}^\ell : x \mapsto \mathbb{E}(g(x))$$

and its (multi-output) positive semi-definite covariance function (also called kernel)

$$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^\ell_{\succeq 0} : (x, x') \mapsto \mathbb{E}((g(x) - \mu(x))(g(x') - \mu(x'))^T).$$

A popular kernel is the Squared Exponential (SE) kernel $k_{SE}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$, with its signal variance σ and lengthscale ℓ. It models smooth data very well and is thus usable for many datasets [70, p. 83], as its realizations are dense in the space of smooth, i.e. infinitely differentiable, functions $C^\infty(\mathbb{R}, \mathbb{R})$ [39, Prop. 1].

Manipulating existing GPs allows to introduce indutive biases in various ways [18, 19, 54, 11, 59]. One important example is the application of matrices of linear operators to a GP. Formally, this is the pushforward operation of an operator matrix B on the GP g as $B_* g = \mathcal{GP}(B\mu(x), Bk(x, x')(B')^T)$ [4], where B' denotes the operation of B on the second argument of $k(x, x')$ [38, Lemma 2.2]. These operators can, for example, be differential operators. The matrix B induces the strong bias such that all realizations lie in the image of B. This pushforward is typical for applications in differential equations [29, 37, 38, 25, 49] or geometry [27].

Example 1. GPs can be constrained to realizations satisfying a system of linear equations given by a matrix A, e.g. $A = [2 \quad -3]$

$$\mathrm{sol}_\mathcal{F}(A) := \left\{ [f_1(x) \quad f_2(x)]^T \in \mathcal{F}^{2\times 1} \,\middle|\, A \cdot [f_1(x) \quad f_2(x)]^T = 0 \right\}.$$

with $\mathcal{F} = C^\infty(\mathbb{R}, \mathbb{R})$ the space of smooth input functions. The matrix $B = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ is maximal in that it solves $A \cdot B = 0$ and hence $\mathrm{sol}_\mathcal{F}(A) = B \cdot \mathcal{F} = \{B \cdot f(x) \,|\, f(x) \in \mathcal{F}\}$. Taking a GP prior $g = \mathcal{GP}(0, k)$ for $f(x) \in \mathcal{F}$ and applying the pushforward then yields a new, constrained GP prior

$$B_* g = \mathcal{GP}(0, Bk(B')^T) = \mathcal{GP}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 9 \cdot \mathrm{k_{SE}} & 6 \cdot \mathrm{k_{SE}} \\ 6 \cdot \mathrm{k_{SE}} & 4 \cdot \mathrm{k_{SE}} \end{bmatrix} \right)$$

for $B \cdot f(x) \in \mathrm{sol}_\mathcal{F}(A)$, whose realizations are guaranteed to lie in $\mathrm{sol}_\mathcal{F}(A)$.

This example is similar to our use case of systems of differential equations for LODE-GPs, where we just replace the matrix B of numbers with a suitable matrix of differential operators.

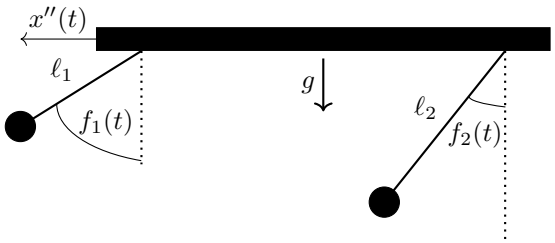

Figure 1: A visualization of the bipendulum with its components (rope lengths ℓ_1, ℓ_2) and states (angles $f_1(t), f_2(t)$, rod acceleration $x''(t)$).

3.2 Smith normal form

The SNF [53, 43] is a normal form for a matrix $A \in \mathbb{R}[x]^{m\times n}$ over the polynomial ring $\mathbb{R}[x]$, such that $U \cdot A \cdot V = D$. Here, $D \in \mathbb{R}[x]^{m\times n}$ is a (not necessarily square) diagonal matrix of same size as A and the base change matrices $U \in \mathbb{R}[x]^{m\times m}$ and $V \in \mathbb{R}[x]^{n\times n}$ are invertible square matrices, i.e. $\det(U), \det(V) \in \mathbb{R} \setminus \{0\}$ [23, 14]. Algorithms to construct the SNF are implemented in many computer algebra systems such as Matlab [42], Maple [40], or SageMath [58] (based on PARI [57]) which is a free and open source python library for computer algebra. Intuitively, the SNF can be compared to the eigendecomposition of a $\mathbb{R}^{n\times n}$ matrix, which—if it exists—produces a square matrix of eigenvectors W and the diagonal matrix of eigenvalues Λ. This analogy is lacking since e.g. the SNF produces two independent base change operator matrices V resp. U instead of a single eigenvector matrix W resp. its inverse W^{-1}. Still, the matrix D decouples and thereby simplifies a system given by the input matrix A, as does the diagonal eigenvalue matrix Λ [48, 14].

Since the SNF exists for matrices over polynomial rings over any field, we can also compute it over a polynomial ring over the function field $\mathbb{R}(a_1, \ldots, a_k)$. Hence, we can model differential equations containing parameters $a_1, \ldots, a_k$. The heating example in Subsection 5.2 demonstrates how such parameters are being used in the SNF and subsequently learned from data.

Example 2. A bipendulum acts as our running example. It consists of a rod with a pendulum attached to each end of the rod, see Figure 2. The linearized equations of an idealized bipendulum:

$$\begin{bmatrix} x''(t) + \ell_1 f_1''(t) + g f_1(t) \\ x''(t) + \ell_2 f_2''(t) + g f_2(t) \end{bmatrix} = \mathbf{0} = \underbrace{\begin{bmatrix} \partial_t^2 + \frac{g}{\ell_1} & 0 & -\frac{1}{\ell_1} \\ 0 & \partial_t^2 + \frac{g}{\ell_2} & -\frac{1}{\ell_2} \end{bmatrix}}_{A} \cdot \begin{bmatrix} f_1(t) \\ f_2(t) \\ u(t) \end{bmatrix} \tag{1}$$

with the operator matrix A, $\ell_i > 0$ the length of the i-th pendulum, $g = 9.81$ the gravitational constant, $u(t) = -x''(t)$ the rod's acceleration, and $f_i(t), i = 1, 2$ the angle between the i-th pendulum and the vertical axis. The SNF of A reveals its properties, specifically when $\ell_1 = \ell_2$ and $\ell_1 \neq \ell_2$. An algorithm calculates the SNF D of A and the base change matrices U, V as follows:

case $\ell_1 = 1 \neq \ell_2 = 2$:

$$\underbrace{\begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \partial_t^2 + g & 0 & -1 \\ 0 & \partial_t^2 + \frac{g}{2} & -\frac{1}{2} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} 0 & -\frac{4}{g} & \frac{2\partial_t^2 + g}{2} \\ 0 & -\frac{2}{g} & \frac{\partial_t^2 + g}{2} \\ -1 & -\frac{4\partial_t^2 + 4g}{g} & (\partial_t^2 + \frac{g}{2})(\partial_t^2 + g) \end{bmatrix}}_{V} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{D}$$

case $l_1 = l_2 = 1$:

$$\underbrace{\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \partial_t^2 + g & 0 & -1 \\ 0 & \partial_t^2 + g & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ -1 & 0 & \partial_t^2 + g \end{bmatrix}}_{V} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \partial_t^2 + g & 0 \end{bmatrix}}_{D}$$

In case of unequal rope length, the controllability is intuitive when imagining that, by accelerating the rod at the right time, the two ropes can reach every combination of valid positions. Formally,

this follows that $D \cdot [h_1(t) \quad h_2(t) \quad h_3(t)]^T = 0$ if and only if $h_1(t) = h_2(t) = 0$ and $h_3(t)$ is an arbitrarily choosable output. In case of equal rope length, the system is not controllable, indicated by the diagonal entry $\partial_t^2 + g$ of D [44, p. 154].

4 Constructing a GP for differential equations

We introduce a polynomial time algorithm to construct LODE-GPs, a class of GPs with realizations dense in the space of solutions of linear ODEs with constant coefficients. This ensures that the LODE-GP is able to produce all possible solutions to the ODEs and nothing but solutions for the ODEs. All this is guaranteed to be strictly accurate, since we make no approximations.

Consider a system of linear homogenous ordinary differential equations with constant coefficients

$$A \cdot \mathbf{f}(t) = 0 \tag{2}$$

with operator matrix $A \in \mathbb{R}[\partial_t]^{m \times n}$ determining the relationship between the smooth functions $f_i(t) \in C^\infty(\mathbb{R}, \mathbb{R})$ of $\mathbf{f}(t) = (f_1(t) \quad \dots \quad f_n(t))^T$. For such systems our main result holds.

Theorem 1. *(LODE-GPs) For every system as in Equation* (2) *there exists a GP g, such that the set of realizations of g is dense in the set of solutions of $A \cdot \mathbf{f}(t) = 0$.*

The following lemma play a crucial role in the proof of this theorem by constructing base covariance functions for a system as in Equation (2) that is *decoupled* (via the SNF) into scalar equations. The proof in Appendix D makes use of the solution space being decomposed, in particular is deal wih finite dimensional vector space like Bayesian linear regression.

Lemma 1. *Covariance functions for solutions of the scalar linear differential equations $d \cdot f = 0$ with constant coefficients, i.e. $d \in \mathbb{R}[\partial_t]$, are given by Table 1 for d is primary, i.e. a power of an irreducible real polynomial. In the case of a non-primary d, each primary factor d_i of $d = \prod_{i=0}^{\ell-1} d_i$ is first translated to its respective covariance function k_i separately before they are added up to give the full covariance function $k(t_1, t_2) = \sum_{i=0}^{\ell-1} k_i(t_1, t_2)$.*

Table 1: Primary operators d and their corresponding covariance function $k(t_1, t_2)$.

d	$k(t_1, t_2)$
1	0
$(\partial_t - a)^j$	$\left(\sum_{i=0}^{j-1} t_1^i t_2^i\right) \cdot \exp(a \cdot (t_1 + t_2))$
$((\partial_t - a - ib)(\partial_t - a + ib))^j$	$\left(\sum_{i=0}^{j-1} t_1^i t_2^i\right) \cdot \exp(a \cdot (t_1 + t_2)) \cdot \cos(b \cdot (t_1 - t_2))$
0	$\exp(-\frac{1}{2}(t_1 - t_2)^2)$

For the case of single (i.e. $j = 1$) zeroes, the sum in the covariance function simplifies to a factor 1.

Example 3. The ODE $d = \partial_t - 1$ has solutions of the form $a \cdot \exp(t)$ for $a \in \mathbb{R}$. This one-dimensional solution space is described by the covariance function $k(t_1, t_2) = \exp(t_1) \cdot \exp(t_2) = \exp(t_1 + t_2)$.

Proof of Theorem 1. We begin with a neutral multiplication of $A \cdot \mathbf{f} = 0$ with $V \cdot V^{-1}$ and a left multiplication by U to decouple the system using the SNF $D = U \cdot A \cdot V$ (cf. Equation (2)):

$$\begin{aligned} & U \cdot A \cdot V \cdot V^{-1} \cdot \mathbf{f} = 0 \\ \Leftrightarrow \quad & D \cdot V^{-1} \cdot \mathbf{f} = 0 \\ \Leftrightarrow \quad & D \cdot \mathbf{p} = 0 \\ \Leftrightarrow \quad & \bigwedge_{i=1}^{\min(n,m)} D_{i,i} \cdot \mathbf{p}_i = 0 \quad \wedge \quad \bigwedge_{i=\min(n,m)+1}^{n} 0 \cdot \mathbf{p}_i = 0 \end{aligned} \tag{3}$$

for decoupled latent vector $\mathbf{p} = V^{-1}\mathbf{f}$ of functions.

We assume the GP-prior $h \sim \mathcal{GP}(\mathbf{0}, k)$ for this vector $\mathbf{p}$ via a GP, where k is a multi-output diagonal covariance function such that each diagonal entry of k is given by Lemma 1. By this Lemma, the set of realizations of this prior h is dense in the set of solutions of $D \cdot \mathbf{p} = 0$.

The pushforward of h with V yields a GP g that can learn the original system of ODEs.

$$g \sim V_* h = \mathcal{GP}(\mathbf{0}, V \cdot k \cdot V') \tag{4}$$

with $V' = V^T$ the operation applied on the second entry of the kernel (i.e. t_2) using [38, Lemma 2.2]. As V is invertible, the set of realizations of g is dense in the set of solutions of $A \cdot \mathbf{f} = 0$. □

This proof shows that diagonalizing the system to $D \cdot \mathbf{p} = 0$ decouples the components of $\mathbf{p}$ to expose the intrinsic behavior of the system. We can also easily interpret the controllability of a system in this decoupled form. A zero column (or $0 \cdot \mathbf{p}_i = 0$) stands for a freely choosable function in $\mathbf{p}$, i.e. an output of the system. A one in a column (or $1 \cdot \mathbf{p}_i = 0$) stands for something we have no choice in, e.g. an input or a state that is fixed once the desired output was chosen. Any other entry leads to sinusoidal-exponential behavior, which is uncontrollable in the sense that it cannot be influenced. This behaviour is classic in systems of differential equations, as most of them can be split into a controllable and uncontrollable solution space, which can be clearly seen in the entries of D. Summing up, a system is controllable iff all diagonal entries of D are either zero or one.

This paper improves upon [37] for two reasons: first, it decouples the uncontrollable part of a system via the SNF as a direct summand and, second, it constructs a covariance function for this uncontrollable summand. This is generally impossible for partial differential equations due to [3, 45].

The runtime of the construction of LODE-GPs in Theorem 1 is dominated by computing the SNF. This has a deterministic polynomial complexity in the size of the (usually small) operator matrices A [63, 65, 35, 69] with quicker probabilistic Las Vegas algorithms [56], and it can be parallelized [64, 66]. For our examples, the SNF terminates instantaniously. The application of the base change matrix V is cubic in the (usually small) size of matrices, assuming constant time to apply operators to the covariance functions from Lemma 1. Once the covariance function is constructed, the complexity of $\mathcal{O}(n^3)$ for GPs applies, where n is the (potentially big) number of data points.

Example 4. We continue Example 2 and consider the SNF of A for $l_1 \neq l_2$. Since the three diagonal entries of the matrix D are $1, 1, 0$ we conclude that latent kernel k has diagonal entries $0, 0, \mathrm{k_{SE}}$ with Lemma 1. Now the pushforward creates the LODE-GP, which requires the following application of two operator matrices; we simplify by removing irrelevant zeroes of the latent covariance.

$$VkV' = \begin{bmatrix} \frac{2\partial_{t_1}^2 + g}{2} \\ \frac{\partial_{t_1}^2 + g}{2} \\ (\partial_{t_1}^2 + \frac{g}{2})(\partial_{t_1}^2 + g) \end{bmatrix} \cdot [\mathrm{k_{SE}}] \cdot \begin{bmatrix} \frac{2\partial_{t_2}^2 + g}{2} & \frac{\partial_{t_2}^2 + g}{2} & (\partial_{t_2}^2 + \frac{g}{2})(\partial_{t_2}^2 + g) \end{bmatrix}$$

Applying these two operators to the kernel $\mathrm{k_{SE}}$ results in the covariance function for the bipendulum.

5 Experimental evaluation

We demonstrate the effectiveness of LODE-GPs in three examples, where we constrain a LODE-GP using the system description and train it with datapoints, similar to solving an initial value problem. The Gröbner basis approach of [37] yields precisely the same results for the two controllable examples and is not applicable to the uncontrollable example. The only other method that could deal with our class of differential equations is [2][2], for which a comparison with the three tank system is discussed in the appendix, in the following we compare our model mainly to classic GPs.

Our comparison includes the error in satisfying the ODEs, specified by the median error the GPs posterior mean function has in satisfying the ODEs at evenly spread points, where we calculate derivatives through finite differences. The LODE-GPs hyperparameters (lengthscale and signal variance) are randomly initialized from a uniform distribution on $[-3, 3]$ and were trained using Adam [31]. The 25 training datapoints are created uniformly in the intervals $[1, 6]$ and $[-5, 5]$ for the bipendulum and three tank system, resp. the heating system. Training and evaluation is repeated

[2]Most probabilistic ODE solvers are not applicable to systems with free functions in their solution set, with the exception of [50], which can only estimate free functions of parameters.

Table 2: The median results and standard dev. for training and evaluation RMSE, loss value (negative marginal log likelihood per datapoint) and the mean ODE satisfaction error, over 20 experiments for the GP and LODE-GP on noisy training data. The mean ODE error is the average of each ODE error for a function in a system. The RMSEs were calculated with noiseless data in the training and evaluation interval, the ODE error only with the latter, for each respective experiment. Smaller is better.

		training RMSE	evaluation RMSE	loss	mean ODE error
Bipendulum	LODE-GP	0.060 ± 0.060	0.106 ± 0.258	-1.638 ± 0.567	**5.135e-07** ± 1.607e-06
	GP	**0.008** ± 0.030	**0.044** ± 0.0175	**-2.072** ± 0.215	0.064 ± 0.023
Heating	LODE-GP	**0.006** ± 0.001	**0.170** ± 0.068	**-2.089** ± 0.100	**0.008** ± 0.007
	GP	0.010 ± 0.001	0.333 ± 0.055	-1.628 ± 0.051	0.218 ± 0.070
Three tank	LODE-GP	**0.023** ± 0.004	0.078 ± 0.046	-0.949 ± 0.063	**1.360e-05** ± 4.919e-06
	GP	0.028 ± 0.005	**0.058** ± 0.031	**-0.974** ± 0.071	0.040 ± 0.020

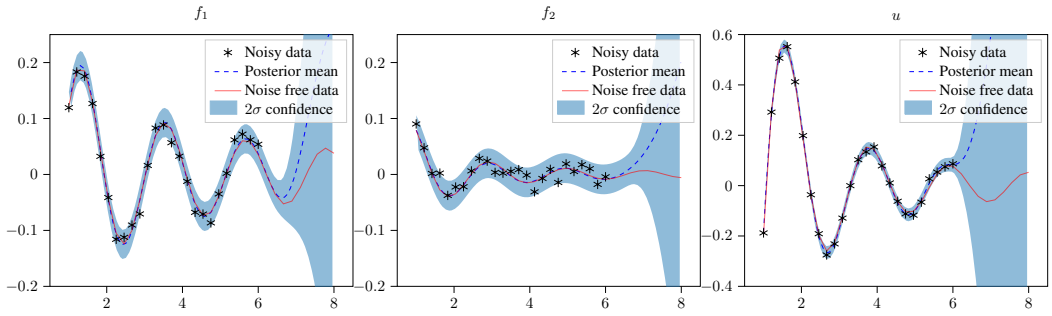

Figure 2: A posterior LODE-GP trained on noisy bipendulum data (black stars), its confidence (blue transparent), posterior mean (blue dashed line) and the original noise-free function (red line). The posterior mean fits the original data very well and shows that the behavior is learned despite noise.

ten times in each experiment using a GPyTorch [21] implementation of our LODE-GP construction with SageMath [58] to symbolically calculate the SNF. The training loss is the negative marginal log likelihood (c.f. [70, Eq. (2.30)]) which, in GPyTorch, is additionally divided by the number of training data points. Details of the training, data generation, verification process, additional experiments, e.g. without noise, and further system details can be found in the appendix.

5.1 Bipendulum - Controllable

We continue with the controllable variant of the bipendulum from Example 2 with the rope lengths $\ell_1 = 1$ and $\ell_2 = 2$. We create 25 points of training data from a solution of the ODEs as shown in Figure 7. We add white noise to the data with standard deviation of 2% of the maximal signal.

Figure 3 compares the error in satisfying the ODE of a LODE-GP and a GP. The LODE-GP is producing samples that satisfy the ODE with a median error[3] of $2\mathrm{e}{-6}$, for both differential equations. In comparison, the *symbolically computed solution* used to generate the training data, has an error of $1.75\mathrm{e}{-7}$, only one order of magnitude better than the LODE-GP. Hence, we conclude that, up to numerical precision, the LODE-GP produces samples that strictly satisfy the ODE, whereas regression models like GPs do not satisfy the differential equations. The LODE-GP produces roughly 30000 times more precise samples (0.06 to $2\mathrm{e}{-6}$) and deviates only slightly from this error.

Further we investigate the RMSE of the models by comparing their posterior mean to noiseless data in the training and evaluation intervals. The LODE-GP RMSE mostly performs similar compared to the GP RMSE (see Table 2). For the training interval, the LODE-GP achieves basically the

[3]The LODE-GPs trains into a local minimum at its value 10^{-10}, with high lengthscale and miniscule signal variance. This models constant zero behaviour which satisfies the ODEs, but does not fit the training data.

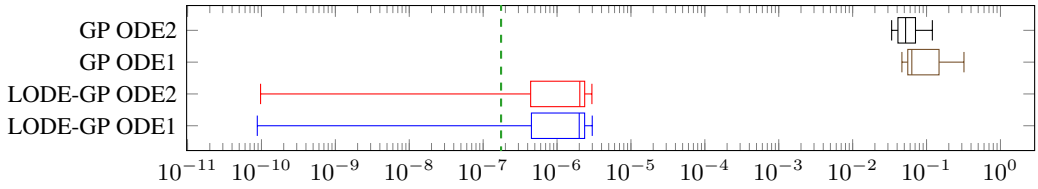

Figure 3: The error in satisfying the bipendulum ODEs in Equation (1) for a GP (top) and LODE-GP (bottom), for noisy training data. Smaller is better. The green dashed line shows the error of a solution in symbolic form. The low error shows that the LODE-GP strictly satisfies the ODEs, up to numerical precision.

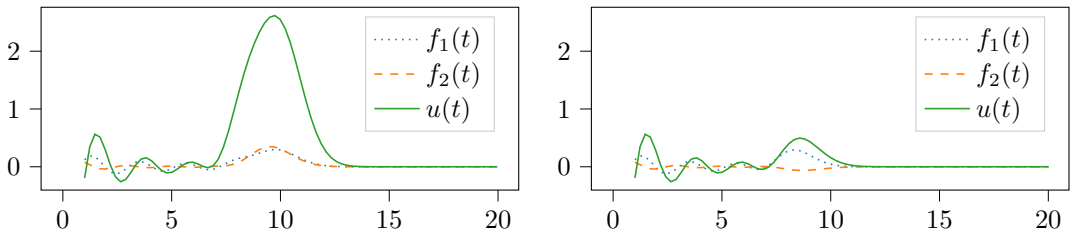

Figure 4: The posterior mean functions of a LODE-GP (left) and a GP (right) trained on the bipendulum data as described in the text.

same performance as the GP. For the evaluation interval the performance worsens due to a SE GP extrapolating to its (constant zero) prior mean. The GP can easily reach this mean zero, by smoothly modifying all channels without regard for the system equations. Since the ODE solution also approaches zero, this leads to a small error in extrapolation. The LODE-GP has to satisfy the system equations to move the mean function back to mean zero. Hence, the system produces a large spike[4] in the input channel $u(t)$, before the system can reach mean zero, as seen in Figure 4.

We also analyse the training runtime and see that the LODE-GP is as fast as the GP during the $\mathcal{O}(n^3)$ model inference and slower by a factor smaller than two in the $\mathcal{O}(n^2)$ covariance matrix calculation.

5.2 Heating system - Controllable with parameters

In this experiment we use a parameterized heating system to show that a LODE-GP can learn physically interpretable system parameters during GP hyperparameter training. The controllable heating system is depicted in Figure 5. It uses an input $u(t)$ to control a heating element $f_1(t)$ which exchanges heat with an object $f_2(t)$. Two physical parameters a and b determine how quickly the heat moves between the objects according to the ODEs:

$$f_1'(t) = -a \cdot (f_1(t) - f_2(t)) + u(t)$$
$$f_2'(t) = -b \cdot (f_2(t) - f_1(t))$$

We generate training data using a solution of the differential equation (see Figure 5) with parameter values $a = 3, b = 1$. The LODE-GP can be constructed with parameters in the ODEs.

Learning the physical parameters a and b reconstructs the original values successfully with a maximal relative error of less than 2.8% in ten training runs on data without noise. After adding white noise to the data with standard deviation of 1% of the maximal signal, the parameters were successfully reconstructed with a maximal relative error of 5.3% in ten training runs. Again, the LODE-GP satisfies the original ODEs (with parameters $a = 3$ and $b = 1$) with an median error of $1\mathrm{e}{-2}$, trained on the noisy data, we refer the reader to Appendix B.3 for a visualization similar to Figure 7. This is bigger than in the previous example, as the LODE-GP only uses approximate parameter values for

[4]The third data channel is particularly large due to its additional factor of $g = 9.81$, which has a different scale by an order of magnitude as the otherwise small data.

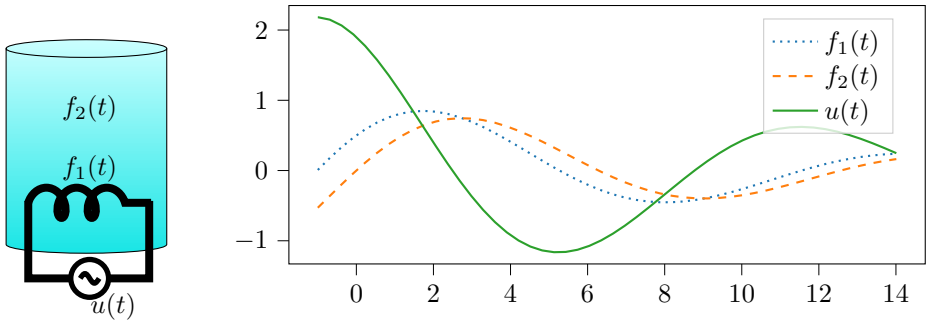

Figure 5: (left) A visualization of the heating system with input $u(t)$ and states $f_1(t), f_2(t)$. (right) A solution of the differential equations, which are used to create the data.

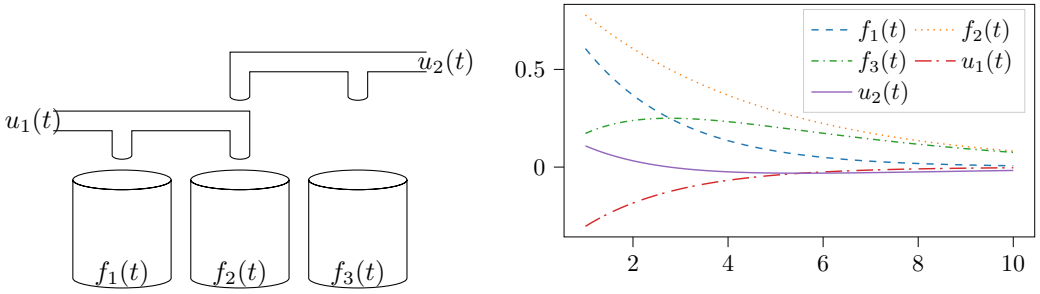

Figure 6: (left) A sketch of the three tank system and (right) a solution of the system.

a and b, but still smaller than the error of a GP of 0.20. Our interpretation of these results is that the LODE-GP can learn the physically interpretable parameters.

5.3 Three tank system - Non-controllable

We conclude the experiments with a non-controllable fluid system where the water level in three tanks is changed by two pipes. The system is non-controllable due to pipes' overlap over the center tank, whose changes directly affect the other tanks. This system requires multiple non-zero covariance functions in the latent GP to describe both the non-controllable subsystem and the two degrees of freedom, which corresponds to the columns of the SNFs matrix D via Lemma 1:

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\partial_t & 0 & 0 \end{bmatrix}$$

We use a solution to the system of ODEs (see Figure 6) to generate 25 datapoints, to which we add white noise with standard deviation of 10% of the maximal signal.

The system ODEs are again strictly satisfied by the LODE-GP with a median error of $1\mathrm{e}{-5}$ in each ODE. We refer the reader to Appendix B.4 for a visualization similar to Figure 7. Hence, LODE-GP can handle larger, non-controllable systems of ODEs, despite high noise.

6 Conclusion

In this paper we have introduced an algorithmical approach to automatically and symbolically create LODE-GP, a class of covariance functions for GPs such that their realizations strictly follow a given system of linear homogeneous ODEs. We have proven that this approach is mathematically sound and verified its effectiveness in experiments. We have demonstrated that the learned posterior mean strictly satisfies the given system of ODEs up to numerical precision and discussed the consequences of this strict behavior e.g. in their extrapolation, leading to drastic behavior when both following the

ODEs and going back to the prior mean. Additionally, we automatically trained the physically interpretable parameters of the system and were able to reconstruct their original values with low relative error. The runtime of the LODE-GP is still asymptotically dominated by the $\mathcal{O}(n^3)$ Cholesky decomposition of the covariance matrix, despite bigger constants in the $\mathcal{O}(n^2)$ part of constructing the covariance matrix and a (usually very small) constant time precomputation of the covariance function. Of course, approximate GPs are applicable to our covariance functions. Beyond applications where the dynamic behavior is known through corresponding ODEs, the combination with kernel search methods [30, 19, 28, 6, 5] has the potential to detect unknown dynamical behavior in data by systematic exploration and use these findings to generate new interpretable knowledge about the data through the corresponding system.

Acknowledgments and Disclosure of Funding

This research was supported by the research training group "Dataninja" (Trustworthy AI for Seamless Problem Solving: Next Generation Intelligence Joins Robust Data Analysis) funded by the German federal state of North Rhine-Westphalia.

We want to thank the anonymous reviewers for their valuable feedback.

References

[1] William A Adkins and Mark G Davidson. Linear systems of differential equations. In *Ordinary Differential Equations*, pages 629–721. 2012.

[2] Mauricio Alvarez, David Luengo, and Neil D Lawrence. Latent force models. In *Artificial Intelligence and Statistics*, pages 9–16. PMLR, 2009.

[3] Mohamed Barakat. Purity Filtration and the Fine Structure of Autonomy. In *Proceedings of the 19th International Symposium on Mathematical Theory of Networks and Systems*, 2010.

[4] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[5] Fabian Berns and Christian Beecks. Automatic Gaussian process model retrieval for big data. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020.

[6] Fabian Berns, Kjeld Schmidt, Ingolf Bracht, and Christian Beecks. 3cs algorithm for efficient Gaussian process model retrieval. In *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021.

[7] Yves Bertin, Etienne Videcoq, Sophie Thieblin, and Daniel Petit. Thermal behavior of an electrical motor through a reduced model. *IEEE Transactions on energy conversion*, 15(2): 129–134, 2000.

[8] LT Biegler, JJ Damiano, and GE Blau. Nonlinear parameter estimation: a case study comparison. *AIChE Journal*, 32(1):29–45, 1986.

[9] Ilias Bilionis, Nicholas Zabaras, Bledar A Konomi, and Guang Lin. Multi-output separable Gaussian process: Towards an efficient, fully Bayesian paradigm for uncertainty quantification. *Journal of Computational Physics*, 241:212–239, 2013.

[10] Nathanael Bosch, Philipp Hennig, and Filip Tronarp. Calibrated adaptive probabilistic ODE solvers. In *AISTATS*, 2021.

[11] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold Gaussian processes for regression. In *International Joint Conference on Neural Networks (IJCNN)*, 2016.

[12] Ben Calderhead, Mark Girolami, and Neil D Lawrence. Accelerating Bayesian inference over nonlinear differential equations with Gaussian processes. *NeurIPS*, 2009.

[13] Kian Chai, Christopher Williams, Stefan Klanke, and Vijayakumar Sethu. Multi-task Gaussian process learning of robot inverse dynamics. *NeurIPS*, 2008.

[14] Thomas Cluzeau, Victorita Dolean, Frédéric Nataf, and Alban Quadrat. Symbolic preconditioning techniques for linear systems of partial differential equations. 2011. URL `https://hal.archives-ouvertes.fr/hal-00664092`.

[15] Elizabeth J Cross and Timothy J Rogers. Physics-derived covariance functions for machine learning in structural dynamics. *IFAC-PapersOnLine*, 54(7):168–173, 2021.

[16] Michiel JL De Hoon, Seiya Imoto, Kazuo Kobayashi, Naotake Ogasawara, and Satoru Miyano. Inferring gene regulatory networks from time-ordered gene expression data of bacillus subtilis using differential equations. In *Biocomputing 2003*, pages 17–28.

[17] Claudia Drygala, Benjamin Winhart, Francesca di Mare, and Hanno Gottschalk. Generative modeling of turbulence. *Physics of Fluids*, 34(3):035114, 2022.

[18] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

[19] David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *ICML*, 2013.

[20] Duccio Fanelli and Francesco Piazza. Analysis and forecast of covid-19 spreading in china, italy and france. *Chaos, Solitons & Fractals*, 134:109761, 2020.

[21] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

[22] Alexandra Goeke, Christian Schilli, Sebastian Walcher, and Eva Zerz. Computing quasi-steady state reductions. *Journal of Mathematical Chemistry*, 50(6):1495–1513, 2012.

[23] Laurenz Göllmann. *Lineare Algebra*. Springer, 2017.

[24] Ítalo Gomes Gonçalves, Felipe Guadagnin, Sissa Kumaira, and Saulo Lopes Da Silva. A machine learning model for structural trend fields. *Computers & Geosciences*, 149:104715, 2021.

[25] Thore Graepel. Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. In *ICML*, 2003.

[26] Tanja Hernández Rodríguez, Anton Sekulic, Markus Lange-Hegermann, and Björn Frahm. Designing robust biotechnological processes regarding variabilities using multi-objective optimization applied to a biopharmaceutical seed train design. *Processes*, 10(5):883, 2022.

[27] Michael Hutchinson, Alexander Terenin, Viacheslav Borovitskiy, So Takao, Yee Teh, and Marc Deisenroth. Vector-valued Gaussian processes on Riemannian manifolds via gauge independent projected kernels. *NeurIPS 2021*, 34, 2021.

[28] Jan David Hüwel, Fabian Berns, and Christian Beecks. Automated kernel search for Gaussian processes on data streams. In *2021 IEEE International Conference on Big Data (Big Data)*, 2021.

[29] Carl Jidling, Niklas Wahlström, Adrian Wills, and Thomas B Schön. Linearly constrained Gaussian processes. In *NeurIPS 2017*, 2017.

[30] Hyunjik Kim and Yee Whye Teh. Scaling up the automatic statistician: Scalable structure discovery using Gaussian processes. In *AISTATS*, 2018.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[32] Edgar D Klenske, Melanie N Zeilinger, Bernhard Schölkopf, and Philipp Hennig. Gaussian process-based predictive control for periodic error correction. *IEEE Transactions on Control Systems Technology*, 24(1):110–121, 2015.

[33] Nicholas Krämer and Philipp Hennig. Linear-time probabilistic solution of boundary value problems. *NeurIPS*, 2021.

[34] Naresh Kumar and Seba Susan. COVID-19 pandemic prediction using time series forecasting models. In *International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2020.

[35] Salah Labhalla, Henri Lombardi, and Roger Marlin. Algorithmes de calcul de la réduction de hermite d'une matrice à coefficients polynomiaux. *Theoretical Computer Science*, 161(1-2): 69–92, 1996.

[36] Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

[37] Markus Lange-Hegermann. Algorithmic linearly constrained Gaussian processes. In *NeurIPS 2018*, pages 2141–2152, 2018.

[38] Markus Lange-Hegermann. Linearly constrained Gaussian processes with boundary conditions. In *AISTATS*, 2021.

[39] Markus Lange-Hegermann and Daniel Robertz. On boundary conditions parametrized by analytic functions. In François Boulier, Matthew England, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, pages 225–245, Cham, 2022. Springer International Publishing.

[40] Maplesoft, a division of Waterloo Maple Inc.. Maple.

[41] Alonso Marco, Philipp Hennig, Jeannette Bohg, Stefan Schaal, and Sebastian Trimpe. Automatic LQR tuning based on Gaussian process optimization: Early experimental results. In *Second Machine Learning in Planning and Control of Robot Motion Workshop at International Conference on Intelligent Robots and Systems*, 2015.

[42] MATLAB. The MathWorks Inc., Natick, Massachusetts.

[43] Morris Newman. The Smith normal form. *Linear algebra and its applications*, 254(1-3): 367–381, 1997.

[44] Ulrich Oberst. Multidimensional constant linear systems. *Acta Applicandae Mathematica*, 20 (1):1–175, 1990.

[45] Alban Quadrat. Grade Filtration of Linear Functional Systems. *Acta Appl. Math.*, 127:27–86, 2013.

[46] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.

[47] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[48] Daniel Robertz. *Formal computational methods for control theory*. PhD thesis, RWTH Aachen University, 2006.

[49] Simo Särkkä. Linear operators and stochastic partial differential equations in Gaussian process regression. In *International Conference on Artificial Neural Networks*, pages 151–158, 2011.

[50] Jonathan Schmidt, Nicholas Krämer, and Philipp Hennig. A probabilistic state space model for joint inference from differential equations and data. *NeurIPS 2021*, 34, 2021.

[51] Michael Schober, David K Duvenaud, and Philipp Hennig. Probabilistic ODE solvers with Runge-Kutta means. *NeurIPS 2014*, 27, 2014.

[52] Michael Schober, Simo Särkkä, and Philipp Hennig. A probabilistic model for the numerical solution of initial value problems. *Statistics and Computing*, 29(1):99–122, 2019.

[53] Henry John Stephen Smith. I. on systems of linear indeterminate equations and congruences. *Proceedings of the Royal Society of London*, (11):86–89, 1862.

[54] Edward Snelson, Zoubin Ghahramani, and Carl Rasmussen. Warped Gaussian processes. *NeurIPS*, 2003.

[55] Arno Solin, Manon Kok, Niklas Wahlström, Thomas B Schön, and Simo Särkkä. Modeling and interpolation of the ambient magnetic field by Gaussian processes. *IEEE Transactions on robotics*, 34(4):1112–1127, 2018.

[56] Arne Storjohann and George Labahn. A fast Las Vegas algorithm for computing the Smith normal form of a polynomial matrix. *Linear Algebra and its Applications*, 253(1-3):155–173, 1997.

[57] *PARI/GP version `2.11.2`*. The PARI Group, Univ. Bordeaux, 2019. available from `http://pari.math.u-bordeaux.fr/`.

[58] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.4)*, 2021. available from `https://www.sagemath.org`.

[59] Silja Thewes, Markus Lange-Hegermann, Christoph Reuber, and Ralf Beck. Advanced Gaussian process modeling techniques. *Design of Experiments (DoE) in Engine Development. Expert Verlag*, 2015.

[60] Filip Tronarp, Simo Särkkä, and Philipp Hennig. Bayesian ode solvers: The maximum a posteriori estimate. *Statistics and Computing*, 31(3):1–18, 2021.

[61] Selvakumar Ulaganathan, Ivo Couckuyt, Tom Dhaene, Joris Degroote, and Eric Laermans. Performance study of gradient-enhanced Kriging. *Engineering with computers*, 32(1):15–34, 2016.

[62] B Ph van Milligen, V Tribaldos, and JA Jiménez. Neural network differential equation and plasma equilibrium solver. *Physical review letters*, 75(20):3594, 1995.

[63] Gilles Villard. Computation of the Smith normal form of polynomial matrices. In *International symposium on Symbolic and algebraic computation*, pages 209–217, 1993.

[64] Gilles Villard. Fast parallel computation of the Smith normal form of polynomial matrices. In *International symposium on Symbolic and algebraic computation*, pages 312–317, 1994.

[65] Gilles Villard. Generalized subresultants for computing the Smith normal form of polynomial matrices. *Journal of Symbolic Computation*, 20(3):269–286, 1995.

[66] Gilles Villard. Fast parallel algorithms for matrix reduction to normal forms. *Applicable Algebra in Engineering, Communication and Computing*, 8(6):511–537, 1997.

[67] Niklas Wahlström, Manon Kok, Thomas B Schön, and Fredrik Gustafsson. Modeling magnetic fields using Gaussian processes. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3522–3526, 2013.

[68] Peter J Wangersky. Lotka-volterra population models. *Annual Review of Ecology and Systematics*, 9(1):189–218, 1978.

[69] Jon Wilkening and Jia Yu. A local construction of the Smith normal form of a matrix polynomial. *Journal of Symbolic Computation*, 46(1):1–22, 2011.

[70] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006.

[71] Eva Zerz. *Topics in Multidimensional Linear Systems Theory*, volume 256 of *Lecture Notes in Control and Information Sciences*. London, 2000.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] There is no potential negative societal impact outside of specially crafted edge cases like physical behavior of rockets, which apply to all works who solve differential equations and can't be mitigated for algorithmic solutions like ours.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See the appendix for a proof of Lemma 1
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the appendix
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See the appendix
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] Data was manually synthetically generated
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] Not present in our data
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [No] No human data
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No] No human data
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No] No human data

Appendices

A Constructing and training a LODE-GP

The construction of a LODE-GP, as discussed in the proof of Theorem 1, involves the following steps:

1. Calculate the SNF symbolically using SageMath [58]
2. Construct the covariance function for the GP-prior $h \sim \mathcal{GP}(\mathbf{0}, k)$, according to Lemma 1 and the last line of Equation 3
3. Calculate the final LODE-GP through the pushforward with the matrix V, including the matrix multiplication $V \cdot k \cdot V'$ and applying the derivatives to k

This construction gives us our LODE-GP, which we train using the standard GP training procedure. All SE kernels that are part of the LODE-GPs have randomly initialized signal variance σ and lengthscale ℓ values chosen uniformly from $[-3, 3]$, which are passed through an $\exp$, during calculation, to ensure numerical stability (e.g. preventing negative lengthscales) and smooth training. We use an Adam optimizer with a learning rate of 0.1 for 300 training iterations in all experiments, further we use the GPyTorch multitask likelihood, since the LODE-GP inherits the class of a GPyTorch multitask GP. We reduced the noise constraint for the likelihood from $1\mathrm{e}{-4}$ to $1\mathrm{e}{-10}$. GPyTorch uses the softplus operation to ensure the constraint. We performed calculations with 64 bit precision.

Training and evaluation was done on our server equipped with a Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz and 64GB RAM running at 3200MHz. All calculations were done on the CPU.

Training was done on 25 datapoints and was repeated ten times for noisy and noise-free, for both LODE-GP and GP.

At no point in training or evaluation is any data or GP output scaled, i.e. we use all the data as is.

B Additional details on experiments and results

We compare the values for the GP and LODE-GP parameters across the three experiments in Table 3. The values for the parameters are their actual values as used in the calculation, i.e. after applying the softplus and exp functions. The behaviour discussed in Section 5 is clearly visible in the maximum LODE-GP lengthscale for the bipendulum, where it sometimes learns a basically constant function. For the bipendulum and the three tank experiments, the LODE-GP generally learns a higher lengthscale, which translates to a smoother behaviour in the outputs.

Analyzing the eigenvalues for the LODE-GP and GP shows that the LODE-GP produces roughly half as many non-zero eigenvalues (i.e. eigenvalue $\lambda > 1\mathrm{e}{-6}$) compared to the GP, as shown in Table 4. This behaviour is expected since the LODE-GP is underlying additional strong constraints due to the inductive bias, which is reflected in its eigenvalues.

B.1 Calculating the ODE error

To calculate the error in satisfying the ODEs via finite differences, we generate uniformly distributed datapoints in the evaluation interval and add duplicate points, shifted by a step of $\Delta = 1e - 3$. We generate either 500 or 333 datapoints, dependent on whether only the first or also the second derivative is necessary, ensuring that always a total of 1000 datapoints are created.

We pass the data to the GP resp. LODE-GP and extract the resulting mean function. The first and second derivative are then approximated using forward difference.

To ensure numerical stability in the calculation of the second derivative, we pass the first order forward difference result through a low-pass filter, removing high frequency noise from the result.

To finally calculate the ODE error, we insert the finite difference values for the corresponding derivatives in the original ODEs and average over the error for each ODE.

Table 3: The trained hyperparameters for the bipendulum (top), heating (middle) and three tank (bottom) experiments, for the noisy and noise-free settings. The GP trains three, resp. five, signal variances, for simpler presentation we show the mean of the signal variance parameters. For the LODE-GP three tank system we also use the mean of the two SE kernels that are created.

	Parameter	Median	Std. deviation	Minimum	Maximum
LODE-GP noisy	noise	6.7521e-05	0.000149199	4.91994e-05	0.000470599
	signal variance	0.0617262	0.0351961	7.22543e-06	0.0879048
	lengthscale	1.24498	3.62295	1.18036	13.4538
LODE-GP no noise	noise	0.000306269	0.000178327	1.7952e-11	0.000413706
	signal variance	0.000298894	0.109973	6.91032e-06	0.354441
	lengthscale	1.82798	0.86266	0.69606	3.54208
GP noisy	noise	8.60043e-05	1.53343e-05	4.62593e-05	9.75798e-05
	signal variance	0.00222937	0.00135264	0.000539298	0.0056802
	lengthscale	0.786804	0.0557031	0.709304	0.86083
GP no noise	noise	2.66449e-09	2.13689e-08	5.74964e-11	5.48067e-08
	signal variance	0.0436976	0.0591215	3.70462e-05	0.152265
	lengthscale	1.25061	0.0776317	1.09688	1.34608
LODE-GP noisy	noise	0.000186034	3.64772e-05	0.000135545	0.000243457
	signal variance	10.0713	4.51968	0.412769	15.3753
	lengthscale	5.46155	0.553268	3.7435	5.86303
LODE-GP no noise	noise	1.41396e-06	1.97306e-06	4.83248e-10	5.48139e-06
	signal variance	6.43588	0.520707	5.76422	7.13915
	lengthscale	5.06497	0.109923	4.87297	5.2045
GP noisy	noise	0.000169646	2.93415e-05	0.000132707	0.000218817
	signal variance	1.26965	0.763973	0.542204	3.03235
	lengthscale	4.20254	0.198991	3.8563	4.47665
GP no noise	noise	1.61976e-11	5.0974e-13	1.57147e-11	1.72362e-11
	signal variance	6.29899	1.27418	3.36005	7.5202
	lengthscale	5.08887	0.175657	4.77775	5.26459
LODE-GP noisy	noise	0.00287473	0.000523729	0.00238355	0.00393084
	signal variance	0.58266	0.192348	0.323803	0.913425
	lengthscale	5.41985	1.00242	4.31772	7.55746
LODE-GP no noise	noise	1.18695e-11	6.09864e-13	1.05086e-11	1.23305e-11
	signal variance	44.2332	5.66572	32.4225	51.8143
	lengthscale	6.66897	0.116321	6.42639	6.8065
GP noisy	noise	0.00281939	0.000507194	0.00176329	0.00377263
	signal variance	0.00104692	0.000369078	0.000433981	0.00150992
	lengthscale	5.52465	1.15024	3.01939	6.5203
GP no noise	noise	1.6506e-11	3.734e-13	1.58496e-11	1.72428e-11
	signal variance	0.00304028	0.00140299	0.00170012	0.006194
	lengthscale	4.42061	0.113491	4.20928	4.69174

Table 4: Median number of eigenvalues greater than zero (i.e. eigenvalue $\lambda > 1\mathrm{e}{-6}$) for the experiments with the LODE-GP and the GP. The total number of eigenvalues for the bipendulum/heating and three tank are 3000 and 5000, respectively.

		eigenvalues greater zero
Bipendulum	LODE-GP	29
	GP	76
Heating	LODE-GP	14
	GP	39
Three tank	LODE-GP	18.5
	GP	32.5

B.2 Bipendulum

We illustrate how a LODE-GP covariance function might look like by showing the (1,1) entry for the bipendulum LODE-GP covariance function:

$$\sigma^2 \exp\left(-\frac{(x_1-x_2)^2}{2\ell^2}\right)\cdot\left(\frac{(x_1-x_2)^4}{\ell^8}-\frac{6\cdot(x_1-x_2)^2}{\ell^6}+\frac{3+g\cdot(x_1-x_2)^2}{\ell^4}-\frac{g}{\ell^2}+\frac{g^2}{4}\right)$$

System details Since the bipendulum has been thoroughly discussed in the paper, we just briefly mention the system equations, operator matrix and the SNF:

$$\begin{bmatrix} x''(t)+\ell_1 f_1''(t)+g f_1(t) \\ x''(t)+\ell_2 f_2''(t)+g f_2(t)\end{bmatrix} = \mathbf{0} = \underbrace{\begin{bmatrix} \partial_t^2+\frac{g}{\ell_1} & 0 & -\frac{1}{\ell_1} \\ 0 & \partial_t^2+\frac{g}{\ell_1} & -\frac{1}{\ell_2}\end{bmatrix}}_{A}\cdot\begin{bmatrix} f_1(t)\\ f_2(t)\\ u(t)\end{bmatrix}$$

case $\ell_1 = 1 \neq \ell_2 = 2$:

$$\underbrace{\begin{bmatrix}1 & 0\\ -\frac{1}{2} & 1\end{bmatrix}}_{U}\underbrace{\begin{bmatrix}\partial_t^2+g & 0 & -1\\ 0 & \partial_t^2+\frac{g}{2} & -\frac{1}{2}\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}0 & -\frac{4}{g} & \frac{2\partial_t^2+g}{2}\\ 0 & -\frac{2}{g} & \frac{\partial_t^2+g}{2}\\ -1 & -\frac{4\partial_t^2+4g}{g} & (\partial_t^2+\frac{g}{2})(\partial_t^2+g)\end{bmatrix}}_{V} = \underbrace{\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}}_{D}$$

To generate the data we use the following solution to the ODE (cf. Figure 7):

$$\begin{bmatrix} f_1(t)\\ f_2(t)\\ u(t)\end{bmatrix} = \begin{bmatrix} -\frac{41\,\sin(3\,t)}{100\,(t+1)}-\frac{3\,\cos(3\,t)}{5\,(t+1)^2}+\frac{\sin(3\,t)}{5\,(t+1)^3}\\ \frac{81\,\sin(3\,t)}{2000\,(t+1)}-\frac{3\,\cos(3\,t)}{10\,(t+1)^2}+\frac{\sin(3\,t)}{10\,(t+1)^3}\\ -\frac{3321\,\sin(3\,t)}{10000\,(t+1)}+\frac{987\,\cos(3\,t)}{500\,(t+1)^2}-\frac{3929\,\sin(3\,t)}{500\,(t+1)^3}-\frac{36\,\cos(3\,t)}{5\,(t+1)^4}+\frac{12\,\sin(3\,t)}{5\,(t+1)^5}\end{bmatrix} \tag{5}$$

Training details To train a LODE-GP we generate uniformly spaced training data from Equation 5 in the interval $[1, 6]$ to which we add noise of the form $0.012 \cdot \sigma_n$ for $\sigma_n \sim \mathcal{N}(0, 1)$, i.e. noise with standard deviation 0.012 (2% of the maximal signal value). The data is shown in Figure 7. To validate our results, e.g. using RMSE, we generate uniformly spaced evaluation data from Equation 5 in the interval $[1, 11]$, without adding noise.

B.3 Heating

System details Original system equations:

$$\begin{bmatrix} -f_1'(t)-a\cdot(f_1(t)-f_2(t))+u(t)\\ -f_2'(t)-b\cdot(f_2(t)-f_1(t))\end{bmatrix} = \mathbf{0} = \underbrace{\begin{bmatrix}\partial_t+a & -a & -1\\ -b & \partial_t+b & 0\end{bmatrix}}_{A}\cdot\begin{bmatrix} f_1(t)\\ f_2(t)\\ u(t)\end{bmatrix}$$

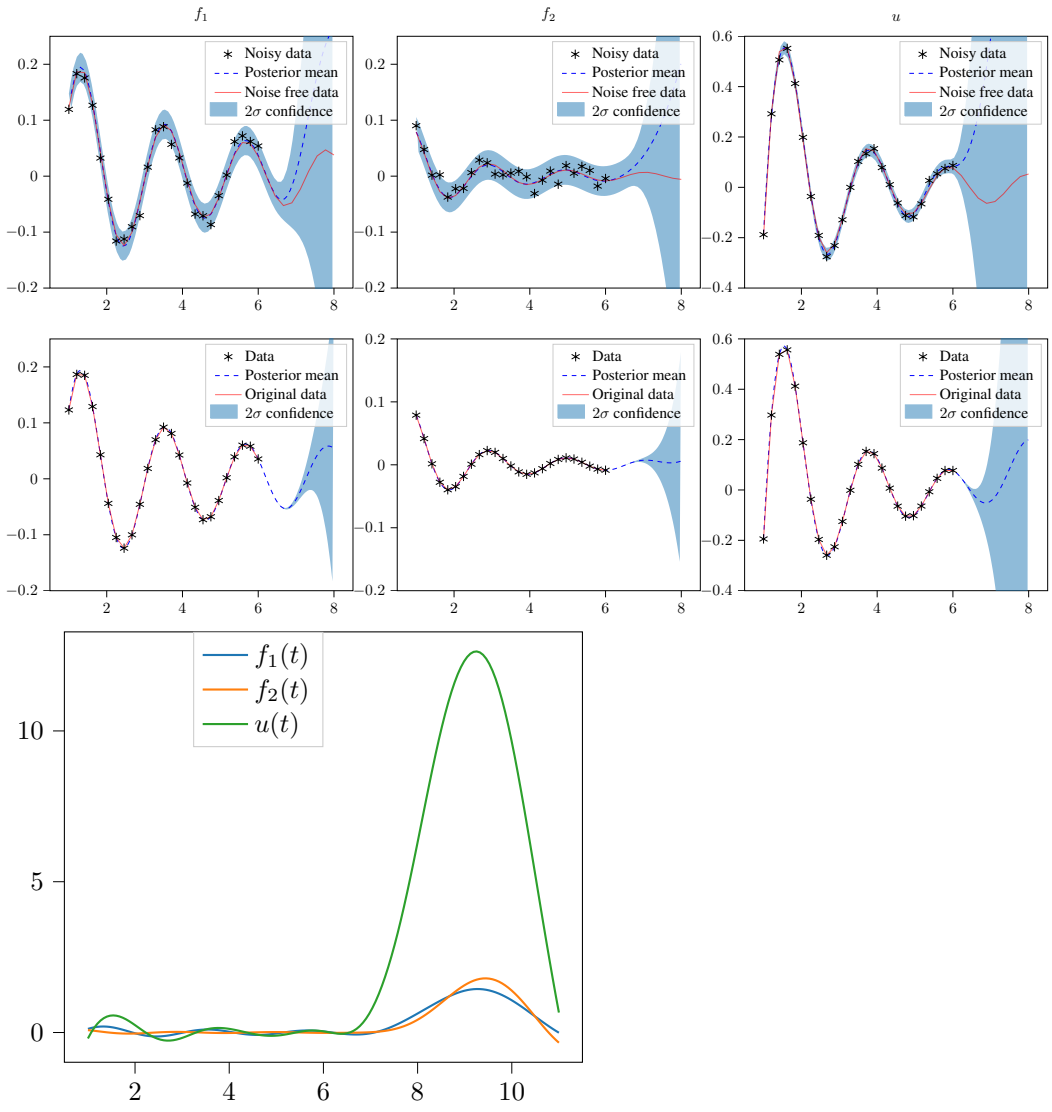

Figure 7: The posterior LODE-GP models for the bipendulum system, trained on noisy data (top) and noise-free data (bottom). The black stars indicate the datapoints (with or without noise), the red line is the solution to the ODEs, the blue dashed line is the LODE-GPs posterior mean, the transparent blue area is the 2σ confidence interval. In the bottom plot, a sample from the GP is shown.

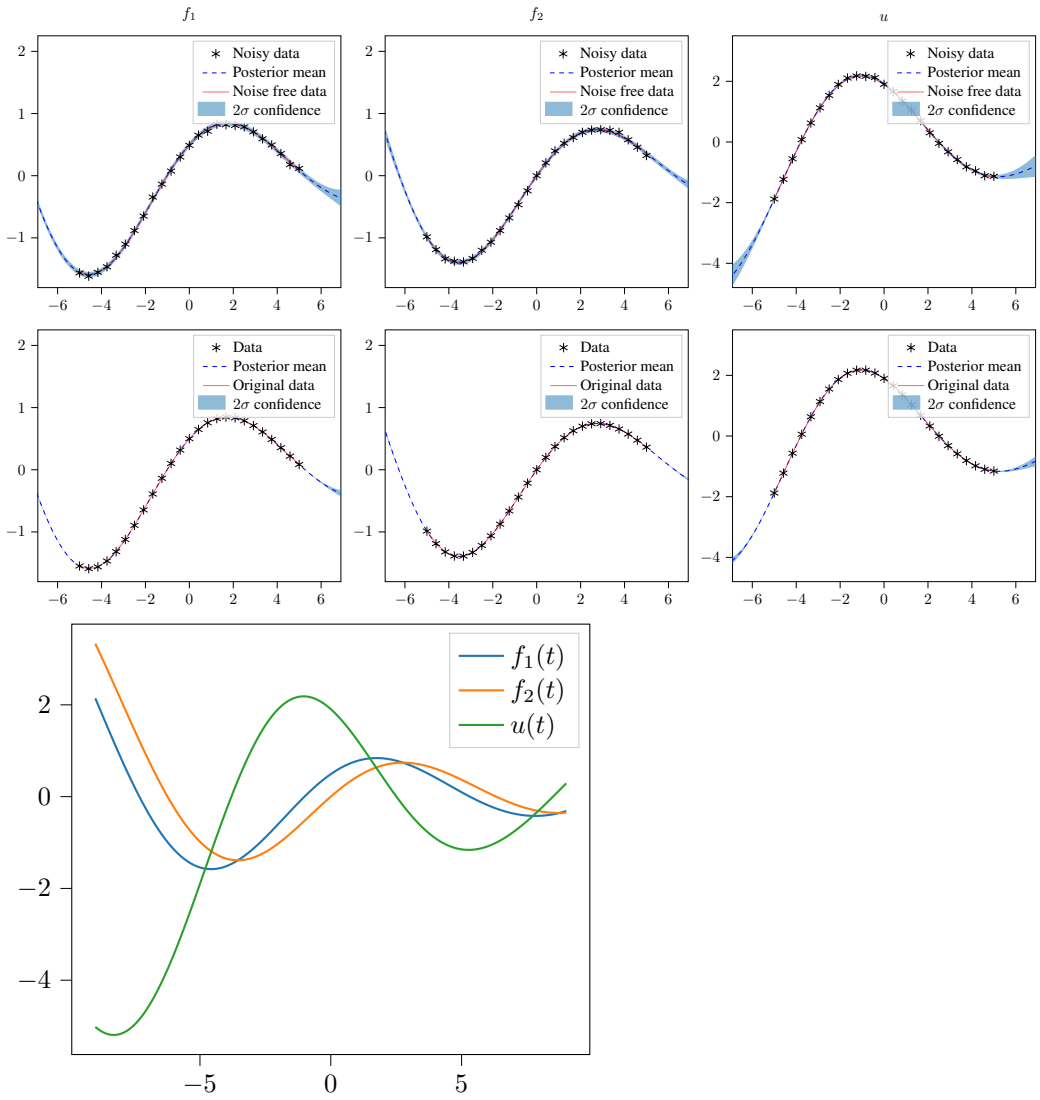

Figure 8: The posterior LODE-GP models for the heating system, trained on noisy data (top) and noise-free data (bottom). The black stars indicate the datapoints (with or without noise), the red line is the solution to the ODEs, the blue dashed line is the LODE-GPs posterior mean, the transparent blue area is the 2σ confidence interval. In the bottom plot, a sample from the GP is shown.

The SNF of the ODEs with all relevant matrices:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{D} = \underbrace{\begin{bmatrix} 0 & \frac{-1}{b} \\ b & \partial_t + a \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \partial_t + a & -a & -1 \\ -b & \partial_t + b & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} 1 & 0 & \partial_t + b \\ 0 & 0 & b \\ 0 & \frac{-1}{b} & \partial_t^2 + (b+a)\,\partial_t \end{bmatrix}}_{V}$$

To generate data, we use the following solution to the heating ODEs (cf. Figure 8).

$$\begin{bmatrix} f_1(t) \\ f_2(t) \\ u(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\cos\left(\frac{1}{2}\,t\right)\exp\left(-\frac{1}{10}\,t\right) + \frac{9}{10}\exp\left(-\frac{1}{10}\,t\right)\sin\left(\frac{1}{2}\,t\right) \\ \exp\left(-\frac{1}{10}\,t\right)\sin\left(\frac{1}{2}\,t\right) \\ \frac{19}{10}\cos\left(\frac{1}{2}\,t\right)\exp\left(-\frac{1}{10}\,t\right) - \frac{16}{25}\exp\left(-\frac{1}{10}\,t\right)\sin\left(\frac{1}{2}\,t\right) \end{bmatrix} \tag{6}$$

Training details To train a LODE-GP we generate uniformly spaced training data from Equation 6 in the interval $[-5, 5]$ to which we add noise of the form $0.02 \cdot \sigma_n$ for $\sigma_n \sim \mathcal{N}(0, 1)$, i.e. noise with

Table 5: The trained parameter values for a and b by the LODE-GP and their relative error from their actual value, for training data with noise (top) and without noise (bottom). Smaller rel. errors are better.

a	b	relative error a	relative error b
2.92218	0.983648	0.0259396	0.0163522
2.9831	0.997748	0.00563382	0.00225164
2.99933	0.991803	0.00022441	0.00819681
3.00912	1.01786	0.00304097	0.0178581
2.92706	0.984112	0.0243126	0.0158881
2.92016	0.968457	0.0266129	0.0315435
2.96962	0.989062	0.0101259	0.0109384
3.00482	0.990055	0.00160732	0.00994511
3.00571	1.00561	0.00190434	0.00560704
3.02601	0.998836	0.0086684	0.00116403
3.00065	1.00025	0.000217934	0.000254375
3.02079	0.989905	0.00693057	0.0100952
3.00042	1.00132	0.000139129	0.00132404
3.00002	1.00177	0.00000558486	0.00176581
2.97293	1.03124	0.00902484	0.0312367
2.98821	1.01434	0.00392941	0.0143379
3.0007	0.99991	0.00023333	0.0000902065
2.98481	1.02509	0.00506387	0.0250855
2.99362	1.01282	0.00212785	0.0128176
3.00045	1.00002	0.000149713	0.0000185651

standard deviation 0.02 (1% of the maximal signal value). The data is shown in Figure 8. To validate our results, e.g. using RMSE, we generate uniformly spaced evaluation data from Equation 6 in the interval $[-9, 9]$, without adding noise.

Additional results We present the exact parameter values trained by the LODE-GP in Table 5. It can be seen that the LODE-GP consistently learn the system parameters a and b with a small relative error, even despite noise.

B.4 Three tank

Original system equations:

$$\begin{bmatrix} -f_1'(t) + u_1(t) \\ -f_2'(t) + u_1(t) + u_2(t) \\ -f_3'(t) + u_2(t) \end{bmatrix} = \mathbf{0} = \underbrace{\begin{bmatrix} -\partial_t & 0 & 0 & 1 & 0 \\ 0 & -\partial_t & 0 & 1 & 1 \\ 0 & 0 & -\partial_t & 0 & 1 \end{bmatrix}}_{A} \cdot \begin{bmatrix} f_1(t) \\ f_2(t) \\ f_3(t) \\ u_1(t) \\ u_2(t) \end{bmatrix}$$

The SNF of the ODEs with all relevant matrices:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -1 & 1 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} -\partial_t & 0 & 0 & 1 & 0 \\ 0 & -\partial_t & 0 & 1 & 1 \\ 0 & 0 & -\partial_t & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 1 & -1 \\ 1 & 0 & 0 & -\partial_t & 0 \\ 0 & 1 & 0 & \partial_t & -\partial_t \end{bmatrix}}_{V} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\partial_t & 0 & 0 \end{bmatrix}}_{D}$$

We use the following solution to the ODEs (cf. Figure 9).

$$\begin{bmatrix} f_1(t) \\ f_2(t) \\ f_3(t) \\ u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} \exp(-\frac{t}{2}) \\ \exp(-\frac{t}{4}) \\ \exp(-\frac{t}{4}) - \exp(-\frac{t}{2}) \\ -\frac{\exp(-\frac{t}{2})}{2} \\ -\frac{\exp(-\frac{t}{4})}{4} + \frac{\exp(-\frac{t}{2})}{2} \end{bmatrix} \tag{7}$$

Training details To train a LODE-GP we generate uniformly spaced training data from Equation 7 in the interval $[1, 6]$ to which we add noise of the form $0.08 \cdot \sigma_n$ for $\sigma_n \sim \mathcal{N}(0, 1)$, i.e. noise with standard deviation 0.08 (10% of the maximal signal value). The data is shown in Figure 9. To validate our results, e.g. using RMSE, we generate uniformly spaced evaluation data from Equation 7 in the interval $[1, 11]$, without adding noise.

For the three tank system we hard coded the case $-x = x$, which is a legal operation since we can take out the -1 as part of a left matrix multiplication and integrate it in the U matrix, which does not influence the LODE-GP.

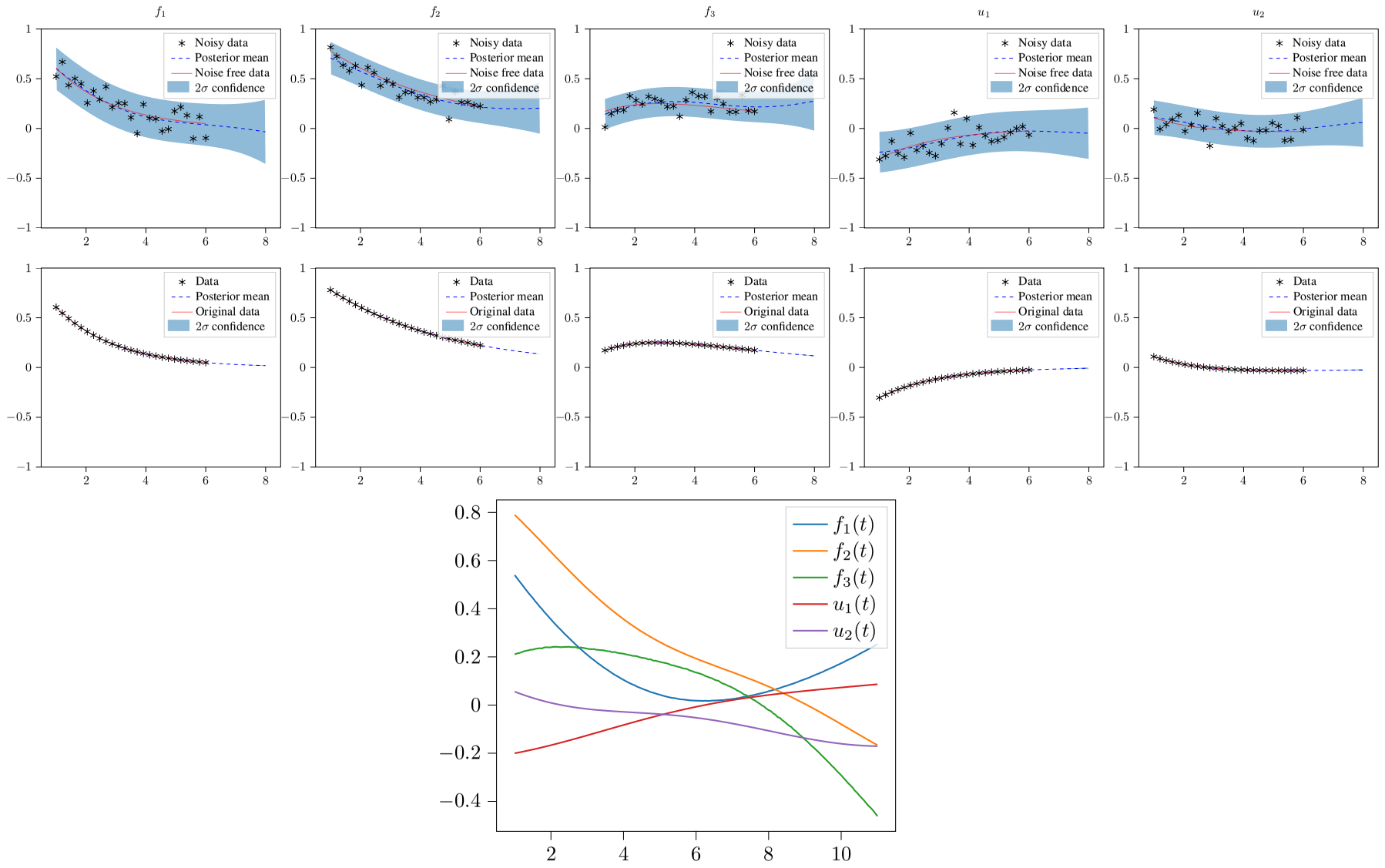

Figure 9: The posterior LODE-GP models for the three tank system, trained on noisy data (top) and noise-free data (bottom). The black stars indicate the datapoints (with or without noise), the red line is the solution to the ODEs, the blue dashed line is the LODE-GPs posterior mean, the transparent blue area is the 2σ confidence interval. In the bottom plot, a sample from the GP is shown.

Comparison to Latent Force Model We compare our LODE-GP with the LFM introduced by [2]. To do so we set the variables from Equation 3 in [2] as follows: $D_q = 0$, $B_q = 0$, $S_{rq} = \begin{bmatrix} -1 & 0 \\ -1 & -1 \\ 0 & -1 \end{bmatrix}$. Further we calculate the covariance function as the solution of the integral $k = \int_0^{t_2} \int_0^{t_1} \exp^{-(x_1 - x_2)^2} dx_1 dx_2$, effectively setting $\ell = 1$ and $\sigma = 1$. Following the steps of [2], we get a GP that can estimate the solution of the three tank systems differential equations.

The resulting GP marginalizes the (in their model considered) latent function u_1 and u_2 and only considered the three data channels f_1, f_2, and f_3. To calculate the ODE error, we inserted the original data for the channels u_1 and u_2 into the calculation.

For a fair comparison with our LODE-GP, we also marginalize u_1 and u_2 there. Similarly as for the LFM, we have set $\ell = 1$ and $\sigma = 1$.

The resulting ODE errors of the two models are as shown in Table 6. The performance of the marginalized LODE-GP is better but comparable to the LFM. The performance of the full LODE-GP, having learned all 5 channels and also set all lengthscales and signal variances to 1, shows a signifcant increase in performance.

Table 6: The ODE error of the LFM, the small LODE-GP, and the full LODE-GP. Smaller is better.

	ODE 1 error	ODE 2 error	ODE 3 error
(marginalized) LFM	0.042816	0.507017	0.013084
marginalized LODE-GP	0.031331	0.025065	0.008595
full LODE-GP	1.210e-05	1.285e-05	1.081e-05

C Discussion of the code and runtime

We use GPyTorch as the core of our code and build our LODE-GP on top of their work, which makes it an essential part of our implementation. Nevertheless, it could be replaced with other GP libraries if the kernel would be rewritten for those. We use SageMath to calculate the symbolic SNF, given the system of ODEs. In our current implementation, SageMath is a necessary library, but could be replaced by any other computer algebra system, with a Python interface, that symbolically solves the SNF for matrices $A \in \mathbb{R}[x]^{m \times n}$ over the polynomial ring $\mathbb{R}[x]$, and additionally the function field $\mathbb{R}(a_1, \ldots, a_k)[x]$ if there are parameters in the ODEs.

Training a LODE-GP on 25 multidimensional[5] datapoints with the stated training specifications needed, in the median, between 3 and 5 seconds per training and, 7 seconds for the three tank system, due to its greater size, as shown in Table 7. This result is in accordance to our findings in Section 5, where we discuss that the $\mathcal{O}(n^2)$ calculation takes longer, whereas the $\mathcal{O}(n^3)$ stays roughly the same. Due to the small number of datapoints, the $\mathcal{O}(n^2)$ calculations have a strong influence on the runtime.

The GP needed roughly 15 seconds per training run.

Additionally, generating the model, i.e. the SNF calculation, preparing the differentiated covariance function and initializing the model object, took averagely 0.3 seconds. Of this time, the SNF calculation takes, on average, 0.15 seconds. These calculations are performed only once for the model and are neglegtable, compared to the training runtime.

During development, we encountered an interesting problem we want to document for other developers. Namely that PyTorch only trains parameters that are part of the model object itself (i.e. which can be called using the `self` keyword). For example, we can't just use a list, that is part of the model object, in which we store the covariance functions, this would cause PyTorch to not recognize the parameters and not train them. In our case, the various kernels and ODE parameters, that are added dynamically, required us to give each parameter and kernel a name and add it to the kernel individually. We also store a position matrix with references to the respective kernel at that position, e.g. for

[5]The data is three dimensional for the bipendulum and heating system and five dimensional for the three tank system.

Table 7: Median training runtime, in seconds, for 300 iterations on 25 datapoints, for the LODE-GP and the GP, for each of the systems. Smaller is better.

		training runtime
Bipendulum	LODE-GP	5.02
	GP	2.34
Heating	LODE-GP	3.12
	GP	2.2
Three tank	LODE-GP	7.49
	GP	1.07

the bipendulum covariance function we have a 3×3 matrix with different kernel objects for each position.

Additionally, we want to highlight the On-Line Encyclopedia of Integer Sequences, which we used to find the construction formula of the number sequence for the general SE kernel derivative.[6]

D Proof for base covariance function construction

We prove the following Lemma

Lemma 1. *Covariance functions for solutions of the scalar linear differential equations $d \cdot f = 0$ with constant coefficients, i.e. $d \in \mathbb{R}[\partial_t]$, are given by Table 8 for d is primary, i.e. a power of an irreducible real polynomial. In the case of a non-primary d, each primary factor d_i of $d = \prod_{i=0}^{\ell-1} d_i$ is first translated to its respective covariance function k_i separately before they are added up to give the full covariance function $k(t_1, t_2) = \sum_{i=0}^{\ell-1} k_i(t_1, t_2)$.*

Table 8: Primary operators d and their corresponding covariance function $k(t_1, t_2)$.

d	$k(t_1, t_2)$
1	0
$(\partial_t - a)^j$	$\left(\sum_{i=0}^{j-1} t_1^i t_2^i\right) \cdot \exp(a \cdot (t_1 + t_2))$
$((\partial_t - a - ib)(\partial_t - a + ib))^j$	$\left(\sum_{i=0}^{j-1} t_1^i t_2^i\right) \cdot \exp(a \cdot (t_1 + t_2)) \cdot \cos(b \cdot (t_1 - t_2))$
mid 0	$\exp(-\frac{1}{2}(t_1 - t_2)^2)$

Proof. For a function f we have $1 \cdot f = 0$ if and only if $f = 0$. Such functions are described by the zero covariance function (and zero mean function).

The real differential equation $(\partial_t - a)^j \cdot f = 0$ only has the analytic solutions given symbolically by $\sum_{i=0}^{j-1} a_i \cdot t^i \cdot \exp(a \cdot t)$ for arbitrary $a_i \in \mathbb{R}$. This is a finite dimensional space and hence (as for linear regressions problems) a covariance function is given by $\sum_{i=0}^{j} \left(t_1^i \cdot \exp(a \cdot t_1)\right) \cdot \left(t_2^i \cdot \exp(a \cdot t_2)\right) = \left(\sum_{i=0}^{j-1} t_1^i t_2^i\right) \cdot \exp(a \cdot (t_1 + t_2))$.

The real differential equation $((\partial_t - a)^2 + b^2)^j = ((\partial_t - a - ib)(\partial_t - a + ib))^j$ only has the analytic solutions given symbolically by $\exp(a \cdot t) \cdot \sum_{i=0}^{j-1} t^i \cdot (a_i \cdot \cos(b \cdot t) + b_i \cdot \sin(b \cdot t))$ for arbitrary

[6] `http://oeis.org/A096713`

$a_i \in \mathbb{R}$. This is again finite dimensional space and hence a covariance function is given by

$$\begin{aligned}
&\sum_{i=0}^{j} \left(t_1^i \cdot \exp(a \cdot t_1) \cdot \cos(b \cdot t_1)\right) \cdot \left(t_2^i \cdot \exp(a \cdot t_2) \cdot \cos(b \cdot t_2)\right) \\
&+ \sum_{i=0}^{j} \left(t_1^i \cdot \exp(a \cdot t_1) \cdot \sin(b \cdot t_1)\right) \cdot \left(t_2^i \cdot \exp(a \cdot t_2) \cdot \sin(b \cdot t_2)\right) \\
&= \left(\sum_{i=0}^{j-1} t_1^i t_2^i\right) \cdot \exp(a \cdot (t_1 + t_2)) \cdot (\cos(b \cdot t_1) \cdot \cos(b \cdot t_2) + \sin(b \cdot t_1) \cdot \sin(b \cdot t_2)) \\
&= \left(\sum_{i=0}^{j-1} t_1^i t_2^i\right) \cdot \exp(a \cdot (t_1 + t_2)) \cdot \cos(b \cdot (t_1 - t_2)).
\end{aligned}$$

The factor $\cos(b \cdot (t_1 - t_2))$ is sometimes called the cosine covariance function.

For a smooth function $f \in C^\infty(\mathbb{R}, \mathbb{R})$, the equation $0 \cdot f = 0$ poses no restriction. Hence, by [39, Prop. 1], the squared exponential covariance function yields a GP with realizations densely contained in the space $C^\infty(\mathbb{R}, \mathbb{R})$. □

E Licenses and versions

We use Python 3.9.7, which uses the PSF license agreement.

We use GPyTorch version 1.5.0 for our experiments, GPyTorch is distributed under the MIT license.

We use SageMath version 9.5, released 2022-01-30, for our experiments, SageMath is distributed under the CC BY-SA 4.0 license.