AutoIP: A United Framework to Integrate Physics into Gaussian Processes

Da Long [1] **Zheng Wang** [1] **Aditi S. Krishnapriyan** [2 3] **Robert M. Kirby** [1] **Shandian Zhe** [1]
Michael W. Mahoney [2 3 4]

Abstract

Physical modeling is critical for many modern science and engineering applications. From a data science or machine learning perspective, where more domain-agnostic, data-driven models are pervasive, physical knowledge — often expressed as differential equations — is valuable in that it is complementary to data, and it can potentially help overcome issues such as data sparsity, noise, and inaccuracy. In this work, we propose a simple, yet powerful and general framework — AutoIP, for Automatically Incorporating Physics — that can integrate all kinds of differential equations into Gaussian Processes (GPs) to enhance prediction accuracy and uncertainty quantification. These equations can be linear or nonlinear, spatial, temporal, or spatio-temporal, complete or incomplete with unknown source terms, and so on. Based on kernel differentiation, we construct a GP prior to sample the values of the target function, equation-related derivatives, and latent source functions, which are all jointly from a multivariate Gaussian distribution. The sampled values are fed to two likelihoods: one to fit the observations, and the other to conform to the equation. We use the whitening method to evade the strong dependency between the sampled function values and kernel parameters, and we develop a stochastic variational learning algorithm. AutoIP shows improvement upon vanilla GPs in both simulation and several real-world applications, even using rough, incomplete equations.

[1]University of Utah [2]University of California, Berkeley [3]Lawrence Berkeley National Laboratory [4]International Computer Science Institute. Correspondence to: Shandian Zhe <zhe@cs.utah.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

1 Introduction

Physical modeling is omnipresent and critical to many modern science and engineering applications, including weather and climate forecasting, bridge design, etc. To model a physical system, one usually writes down a set of ordinary differential equations (ODEs) or partial differential equations (PDEs) that characterize the system behavior according to physical laws. One then identifies boundary and/or initial conditions and solves the equations, typically via numerical methods, to obtain the solution function on the domain of interest. The solution is then used in the subsequent steps, such as system evolution and design optimization.

Machine learning (ML) and data science use a different paradigm. Methods from these areas estimate or reconstruct target functions from observed data, rather than by solving physical equations. To learn target functions, ML methods typically optimize a data-dependent loss. Nonetheless, one would hope that the knowledge reflected in physical models, especially in ODEs and PDEs, is valuable to ML, in that this knowledge characterizes the local behaviors or properties of the target function, which then extrapolate to the entire domain of interest. Hence, as a complementary information source, physics knowledge can potentially help overcome data sparsity, noise, and inaccuracy in measurements of physical systems. These problems are ubiquitous in practice.

One example of an effort along these lines is provided by so-called physics-informed neural networks (PINNs) (Raissi et al., 2019), which use neural networks (NNs) to try to solve physical differential equations. PINNs simultaneously fit the boundary/initial conditions and minimize a residual term to conform to the equation. That is, from an optimization perspective (Boyd and Vandenberghe, 2004; Nocedal and Wright, 2006), PINNs do *not* solve a constrained optimization problem, where physical knowledge is included as a constraint. Instead, they adopt a penalty method approach (as opposed to an augmented Lagrangian method), solving a (related but non-equivalent) soft-constrained problem. Also, PINNs demand that the form of the equation be fully specified. From the data science perspective, this might restrict their capability to leverage physics knowledge in a broader sense. That is, the knowledge within incomplete equations, *e.g.,* those including latent sources (functions), cannot be

incorporated. In addition, the differentiation operators on the NN itself (in the residual) make the effective loss function quite complicated (Krishnapriyan et al., 2021), bringing challenges in optimization/training, robustness, and uncertainty quantification (Edwards, 2022).

In this work, we consider incorporating physics knowledge into Gaussian processes (GPs). GPs are a nonparametric Bayesian modeling approach, which is not only flexible enough to learn complex functions from data, but also is convenient to quantify the uncertainty (due to their closed-form posterior distribution). In most cases, GPs perform well with simple kernels, *e.g.,* Square Exponential (SE), without the need for complex architecture design and hyper-parameter tuning; and, in many cases, methods from randomized numerical linear algebra (RandNLA) (Mahoney, 2011; Drineas and Mahoney, 2016; Derezinski and Mahoney, 2021) can be used to speed up traditional algorithms for GP training and computation. To this end, we propose AutoIP, a framework for Automatically Incorporating Physics. AutoIP can incorporate all kinds of differential equations into GPs to enhance their prediction accuracy and uncertainty estimates. These equations can be linear or nonlinear, spatial, temporal, or spatio-temporal, complete or incomplete, including unknown source terms and coefficients, and so on. In this way, we can boost GPs with various sorts of physics knowledge.

In more detail, AutoIP first samples a set of collocation points in the domain to support the equation. It then uses kernel differentiation to construct a GP prior. This prior jointly samples from a multi-variate Gaussian distribution the values of the target function at the training inputs and the values of all the equation-related derivatives and latent sources at the collocation points. In this way, AutoIP couples the target function and its derivatives in a probabilistic framework, without the need for conducting differential operations on a nonlinear surrogate (like with NNs). Next, AutoIP feeds these samples to two likelihoods. One is to fit the training data. The other is a virtual Gaussian likelihood that encourages conformity to the equation. Since any differential equation is a combination of derivatives and source functions (if needed), it is straightforward to combine their sampled values correspondingly in the virtual likelihood. In doing so, we can flexibly incorporate any equation. For effective and efficient inference, AutoIP uses the whitening method to parameterize the latent random variables with a standard Gaussian noise, thereby evading their strong dependency on the kernel parameters. AutoIP then jointly estimates the kernel parameters and the posterior of the noise with a stochastic variational learning algorithm. We find an interesting insight that the approximate posterior process is still a GP but with a new kernel, which maintains the kernel differentiation property.

We illustrate our AutoIP framework in several benchmark physical systems, including nonlinear pendulums and the Allen-Cahn equation. We tested it with both complete and incomplete equations. For the latter, we hid a part of the ground-truth equation and view it as a latent source. In both cases, our approach largely improves upon the standard GP (*i.e.,* without physics knowledge incorporated) in prediction accuracy and uncertainty estimate when doing extrapolation. Next, on two real-world benchmark datasets, Swiss Jura and CMU motion datasets, we examined our approach when integrating a sensible physics model that includes latent sources. Here, the ground-truth governing equations are unknown. AutoIP shows better prediction accuracy and predictive log-likelihood, as compared with GP and latent force models, a classical approach that can integrate physics knowledge when Green's functions are available.

2 Gaussian Process Regression

Gaussian processes (GPs) are stochastic priors in function space. Due to their nonparametric nature, GPs can self-adapt to the complexity of the target function according to data, *e.g.,* from simple multilinear to highly nonlinear, not restricted to a specific parametric form. Suppose we aim to learn a function $f : \mathbb{R}^d \to \mathbb{R}$. When we place a GP prior over $f(\cdot)$, it means that f is sampled as a realization of a Gaussian process governed by some covariance function $\kappa(\cdot,\cdot)$, $f \sim \mathcal{GP}\left(m(\cdot), \kappa(\cdot,\cdot)\right)$ where m is the mean function, often set as the constant zero. The covariance function captures the stochastic correlation between the function values in terms of their inputs, and it is often chosen as a kernel function. For example, a popular choice is the Square Exponential (SE) kernel with Automatic Relevance Determination (ARD), $\operatorname{cov}\left(f(\mathbf{x}), f(\mathbf{x}')\right) = \kappa(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^\top \operatorname{diag}(\frac{1}{\mathbf{s}})(\mathbf{x}-\mathbf{x}')\right)$, where $\mathbf{s}$ are the length-scales (kernel parameters). The finite projection of the GP is a collection of the values of $f(\cdot)$ at an arbitrary finite set of inputs. This follows a multivariate Gaussian distribution, where the covariance matrix is the kernel matrix on the input set.

Consider a training dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]^\top$, $\mathbf{y} = [y_1, \cdots, y_N]^\top$, each $\mathbf{x}_n$ is an input, and y_n is a noisy observation of $f(\mathbf{x}_n)$. Then the function values at the training inputs, $\mathbf{f} = [f(\mathbf{x}_1), \cdots, f(\mathbf{x}_N)]^\top$, follow a multivariate Gaussian distribution, $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$ where each $[\mathbf{K}]_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Given $\mathbf{f}$, we can use a noisy model to sample the observation $\mathbf{y}$. A Gaussian noise model is the commonly used one, $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \beta^{-1}\mathbf{I})$ where β is the inverse noise variance. We can then marginalize out $\mathbf{f}$ to obtain the marginal likelihood of $\mathbf{y}$, *i.e.,* evidence,

$$p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \beta^{-1}\mathbf{I}). \tag{1}$$

To learn the model, one often maximizes the evidence to

estimate the kernel parameters and the inverse noise variance β. Given a new input $\mathbf{x}^*$, according to the GP prior, $[f(\mathbf{x}^*); \mathbf{y}]$ also follows a multivariate Gaussian distribution. Hence, the posterior (or predictive) distribution of $f(\mathbf{x}^*)$ is a conditional Gaussian,

$$p\big(f(\mathbf{x}^*)|\mathbf{x}^*, \mathbf{X}, \mathbf{y}\big) = \mathcal{N}\big(f(\mathbf{x}^*)|\mu^*, v^*\big), \tag{2}$$

where $\mu^* = \mathbf{k}_*^\top(\mathbf{K}+\beta^{-1}\mathbf{I})^{-1}\mathbf{y}$, $v^* = \kappa(\mathbf{x}^*, \mathbf{x}^*) - \kappa_*^\top(\mathbf{K}+\beta^{-1}\mathbf{I})^{-1}\mathbf{k}_*$ and $\mathbf{k}_* = [\kappa(\mathbf{x}^*, \mathbf{x}_1), \cdots, \kappa(\mathbf{x}^*, \mathbf{x}_N)]^\top$. Due to the closed-form posterior, GP models are convenient for quantifying and reasoning about under uncertainty.

3 Our AutoIP Framework

Model. To boost GPs with physics knowledge, we propose AutoIP— a framework for Automatically Incorporating Physics from all kinds of differential equations. Without loss of generality, we use a nonlinear, incomplete PDE in the Allen-Cahn family to illustrate the idea (Allen and Cahn, 1972). This PDE takes the form

$$\partial_t u - \nu \cdot \partial_x^2 u + \gamma \cdot u(u^2 - 1) + g(x, t) = 0, \tag{3}$$

where the target function $u(x, t)$ is a spatial-temporal function, $g(x, t)$ is an unknown source term, and ν and γ are coefficients. Note that $u(u^2 - 1)$ is a nonlinear term. Suppose we are given N training examples, $\mathcal{D} = \{(\mathbf{z}_1, y_1), \ldots, (\mathbf{z}_N, y_n)\}$, where each $\mathbf{z}_n = (x_n, t_n)$. We want our learned function not only to fit the observations but also to conform to (3), *i.e.,* to the known physics. To this end, we sample a set of M collocation points $\widehat{\mathcal{Z}} = \{\widehat{\mathbf{z}}_1, \ldots, \widehat{\mathbf{z}}_M\}$ in the domain of interest (*e.g.,* $[0, 2\pi] \times [0, 1]$), and we augment the GP model to encourage the L.H.S of the equation (3) evaluated at $\widehat{\mathcal{Z}}$ to be close to zero.

Specifically, we first construct a GP prior over u, g and the equation-related derivatives, *i.e.,* $\partial_t u$ and $\partial_x^2 u$. Naturally, we can sample $u \sim \mathcal{GP}\left(0, \kappa_u(\cdot, \cdot)\right)$ and $g \sim \mathcal{GP}\left(0, \kappa_g(\cdot, \cdot)\right)$. The key observation is that once u is drawn, all of its derivatives are determined — we do not need to draw them from separate GPs. The covariance and cross-covariance among u and its derivatives can be obtained from κ_u via kernel differentiation (Williams and Rasmussen, 2006),

$$\begin{aligned}
\operatorname{cov}\left(u(\mathbf{z}_1), u(\mathbf{z}_2)\right) &= \kappa_u(\mathbf{z}_1, \mathbf{z}_2), \\
\operatorname{cov}\left(\partial_t u(\mathbf{z}_1), \partial_t u(\mathbf{z}_2)\right) &= \frac{\partial^2 \kappa_u(\mathbf{z}_1, \mathbf{z}_2)}{\partial t_1 \partial t_2}, \\
\operatorname{cov}\left(\partial_x^2 u(\mathbf{z}_1), \partial_x^2 u(\mathbf{z}_2)\right) &= \frac{\partial^4 \kappa_u(\mathbf{z}_1, \mathbf{z}_2)}{\partial x_1^2 \partial x_2^2}, \\
\operatorname{cov}\left(\partial_t u(\mathbf{z}_1), \partial_x^2 u(\mathbf{z}_2)\right) &= \frac{\partial^3 \kappa_u(\mathbf{z}_1, \mathbf{z}_2)}{\partial t_1 \partial x_2^2}, \\
\operatorname{cov}\left(\partial_t u(\mathbf{z}_1), u(\mathbf{z}_2)\right) &= \frac{\partial \kappa_u(\mathbf{z}_1, \mathbf{z}_2)}{\partial t_1}, \\
\operatorname{cov}\left(\partial_x^2 u(\mathbf{z}_1), u(\mathbf{z}_2)\right) &= \frac{\partial^2 \kappa_u(\mathbf{z}_1, \mathbf{z}_2)}{\partial x_1^2},
\end{aligned} \tag{4}$$

where $\mathbf{z}_1 = (x_1, t_1)$ and $\mathbf{z}_2 = (x_2, t_2)$ are two arbitrary inputs (we abuse notation a bit here for convenience — the points $\mathbf{z}_1$ and $\mathbf{z}_2$ are different from the training inputs $\mathcal{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$.) In general, we can obtain the covariance of two arbitrary derivatives (of the same function) by taking the partial derivatives of the original kernel with respect to the corresponding inputs (using the same order). Since the commonly used kernels, *e.g.,* the SE kernel, are quite simple, we can obtain their derivatives analytically and directly apply the result for computation. We denote the values of the target function at the training inputs by $\mathbf{u} = (u(\mathbf{z}_1), \ldots, u(\mathbf{z}_N))^\top$, the values of u and u's derivatives at the collocation points by $\widehat{\mathbf{u}} = (u(\widehat{\mathbf{z}}_1), \ldots, u(\widehat{\mathbf{z}}_M))^\top$, $\widehat{\mathbf{u}}_t = (\partial_t u(\widehat{\mathbf{z}}_1), \ldots, \partial_t u(\widehat{\mathbf{z}}_M))^\top$ and $\widehat{\mathbf{u}}_{xx} = \left(\partial_x^2 u(\widehat{\mathbf{z}}_1), \ldots, \partial_x^2 u(\widehat{\mathbf{z}}_M)\right)^\top$, and the values of the latent source term at the collocation points by $\mathbf{g} = (g(\widehat{\mathbf{z}}_1), \ldots, g(\widehat{\mathbf{z}}_N))^\top$. Then, we can leverage the covariance functions in (4) and κ_g to construct a joint Gaussian prior over $\mathbf{f} = [\mathbf{u}; \widehat{\mathbf{u}}; \widehat{\mathbf{u}}_t; \widehat{\mathbf{u}}_{xx}; \mathbf{g}]$,

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{\Sigma}). \tag{5}$$

The covariance matrix $\mathbf{\Sigma}$ is block-diagonal, including a dense block for $[\mathbf{u}; \widehat{\mathbf{u}}; \widehat{\mathbf{u}}_t; \widehat{\mathbf{u}}_{xx}]$ computed from (4) and another dense block for $\mathbf{g}$ computed via κ_g. Note that we can further model the covariance between u and g if we have more prior knowledge. Here, we consider the general case that assumes they are sampled from two independent GPs.

Next, we feed the sampled $\mathbf{f}$ to two data likelihoods. One is to fit the actual observations from a Gaussian noise model,

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{u}, \beta^{-1}\mathbf{I}). \tag{6}$$

The other is a virtual Gaussian likelihood that integrates the physics knowledge into the differential equation (3), as

$$p(\mathbf{0}|\mathbf{f}) = \mathcal{N}(\mathbf{0}|\widehat{\mathbf{u}}_t - \nu\widehat{\mathbf{u}}_{xx} + \gamma\widehat{\mathbf{u}} \circ (\widehat{\mathbf{u}} \circ \widehat{\mathbf{u}} - \mathbf{1}) + \mathbf{g}, v\mathbf{I}), \tag{7}$$

where v is the variance and $\circ$ is element-wise product. As we can see, the mean of the Gaussian in (7) is the evaluation of the L.H.S of (3) at the collocation points. The variance v indicates how it is close to zero. The smaller v is, the more consistent the sampled functions are with the differential equation. In practice, we can either tune v or learn v to enforce the conformity to a certain degree. Finally, the joint probability of our model is given by

$$\begin{aligned}
p(\mathbf{f}, \mathbf{y}, \mathbf{0}) = &\mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{\Sigma})\mathcal{N}(\mathbf{y}|\mathbf{u}, \beta^{-1}\mathbf{I}) \\
&\cdot \mathcal{N}(\mathbf{0}|\widehat{\mathbf{u}}_t - \nu\widehat{\mathbf{u}}_{xx} + \gamma\widehat{\mathbf{u}} \circ (\widehat{\mathbf{u}} \circ \widehat{\mathbf{u}} - \mathbf{1}) + \mathbf{g}, v\mathbf{I}).
\end{aligned} \tag{8}$$

As we can see, by leveraging the kernel differentiation, our model constructs a GP prior to sample jointly the target function and all the basic components of the differential equation, *i.e.,* all kinds of derivatives and latent source terms (if needed). We naturally couple them into a probabilistic

framework, without the need for taking explicit differentiation over some (complex) function surrogates. Then, through the virtual Gaussian likelihood (7), we can combine these components following arbitrary differential equation (in the mean) to encode the physics knowledge. If there are unknown coefficients, *e.g.,* ν and γ in (3), we can estimate them jointly during model inference. While simple, our augmented GP is flexible enough to incorporate a variety of differential equations to benefit learning and prediction.

Algorithm. In general, the exact posterior of the latent random variables $\mathbf{f}$ is intractable to compute or marginalize out (as in standard GP regression), because the virtual likelihood (7) couples the components of $\mathbf{f}$ to reflect the equation, which can be nonlinear and nontrivial. Hence, we develop a general variational inference algorithm to estimate jointly the posterior of $\mathbf{f}$ and kernel parameters, inverse noise variance β, v, etc. However, we found that a straightforward implementation to optimize the variational posterior $q(f)$ often gets stuck at an inferior estimate. This might be due to the strong coupling of $\mathbf{f}$ and the kernel parameters in the prior (5). To address this issue, we use the whitening method (Murray and Adams, 2010) in MCMC sampling. That is, we parameterize $\mathbf{f}$ with a Gaussian noise,

$$\mathbf{f} = \mathbf{A}\boldsymbol{\eta} \tag{9}$$

where $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\mathbf{A}$ is the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma}$, *i.e.,* $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^\top$. Therefore, the joint probability of the model can be rewritten as

$$p(\text{Joint}) = \mathcal{N}(\boldsymbol{\eta}|\mathbf{0}, \mathbf{I})p(\mathbf{y}|\mathbf{L}\boldsymbol{\eta})p(\mathbf{0}|\mathbf{L}\boldsymbol{\eta}). \tag{10}$$

See (6) and (7) for $p(\mathbf{y}|\mathbf{L}\boldsymbol{\eta})$ and $p(\mathbf{0}|\mathbf{L}\boldsymbol{\eta})$, respectively. We then introduce a Gaussian variational posterior for the noise, $q(\boldsymbol{\eta}) = \mathcal{N}(\boldsymbol{\eta}|\boldsymbol{\mu}, \mathbf{L}\mathbf{L}^\top)$, where $\mathbf{L}$ is a lower-triangular matrix to ensure the positive definiteness of the covariance matrix. Since the prior of $\boldsymbol{\eta}$ is the standard normal distribution, it does not depend on the kernel parameters any more. We then construct a variational evidence lower bound,

$$\begin{aligned}\mathcal{L} = &-\text{KL}\left(q(\boldsymbol{\eta})\|\mathcal{N}(\boldsymbol{\eta}|\mathbf{0}, \mathbf{I})\right) \\ &+ \mathbb{E}_q\left[\log p(\mathbf{y}|\mathbf{L}\boldsymbol{\eta})\right] + \mathbb{E}_q\left[\log(p(\mathbf{0}|\mathbf{L}\boldsymbol{\eta}))\right],\end{aligned} \tag{11}$$

where $\text{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence. We maximize $\mathcal{L}$ to estimate $q(\boldsymbol{\eta})$ and the other parameters. We use the reparameterization trick (Kingma and Welling, 2013) to conduct stochastic optimization. Once we obtain $q(\boldsymbol{\eta})$, from (9) we can immediately obtain the variational posterior of $\mathbf{f}$, $q(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{A}\boldsymbol{\mu}, \mathbf{A}\mathbf{L}\mathbf{L}^\top\mathbf{A}^\top)$, according to which we can compute the predictive distribution of the function values at new inputs. We do not consider the computational challenge when the number of training examples (N) and/or collocation points (M) is big. However, one can extend our algorithm to a variety of sparse GP frameworks, *e.g.,* (Hensman et al., 2013), and/or use methods from RandNLA (Mahoney, 2011; Drineas and Mahoney, 2016; Derezinski and Mahoney, 2021) in order to handle large data.

Remarks. With the Gaussian variational approximation, the posterior process is still a GP and maintains the link between the function and its derivatives in terms of kernel differentiation. But the kernel has changed. This can be seen from the predictive distribution of an arbitrary finite set of the function and its derivative values, say $\mathbf{h} = (u(\mathbf{z}_1), u(\mathbf{z}_2), \partial_x u(\mathbf{z}_2), \ldots)$, which is,

$$\begin{aligned}p(\mathbf{h}|\mathcal{D}) = &\int p(\mathbf{h}|\mathbf{f})p(\mathbf{f}|\mathcal{D})\mathrm{d}\mathbf{f} \approx \int p(\mathbf{h}|\mathbf{f})\mathcal{N}(\mathbf{f}|\mathbf{m}_f, \mathbf{V}_f)\mathrm{d}\mathbf{f} \\ = &\mathcal{N}\left(\mathbf{h}|\mathbf{m}_h, \text{cov}(\mathbf{h}, \mathbf{h}) - \text{cov}(\mathbf{h}, \mathbf{f}) \cdot \mathbf{B} \cdot \text{cov}(\mathbf{f}, \mathbf{h})\right),\end{aligned} \tag{12}$$

where $\mathcal{D}$ is the data (including $\mathbf{y}$ and the virtual observation $\mathbf{0}$), $\text{cov}(\cdot)$ is obtained from the kernel κ_u and its differentiation (see (4)), and $\mathbf{m}_f$ and $\mathbf{V}_f$ are the estimated posterior mean and covariance of $\mathbf{f}$, $\mathbf{m}_h = \text{cov}(\mathbf{h}, \mathbf{f})\boldsymbol{\Sigma}^{-1}\mathbf{m}_f$, and $\mathbf{B} = \boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\mathbf{V}_f\boldsymbol{\Sigma}^{-1}$.

The result (12) defines a GP for $u(\cdot)$ with a new kernel: $\overline{\text{cov}}(u(\mathbf{z}_1), u(\mathbf{z}_2)) = \rho(\mathbf{z}_1, \mathbf{z}_2)$, where $\mathbf{z}_1$ and $\mathbf{z}_2$ are two arbitrary inputs, $\rho(\mathbf{z}_1, \mathbf{z}_2) = \kappa_u(\mathbf{z}_1, \mathbf{z}_2) - \tilde{\kappa}(\mathbf{z}_1, \mathbf{Z}) \cdot \mathbf{B} \cdot \tilde{\kappa}(\mathbf{z}_2, \mathbf{Z})$, $\mathbf{Z} = \{\mathcal{Z}, \widehat{\mathcal{Z}}\}$ are the corresponding inputs of $\mathbf{f}$, and $\tilde{\kappa}(\mathbf{z}, \mathbf{Z}) = \text{cov}(u(\mathbf{z}), \mathbf{f})$, which applies κ_u or its partial derivatives over $\mathbf{z}$ and each input in $\mathbf{Z}$; see (4). To verify if the link between u and its derivatives is still there, we examine the derivatives of the new kernel $k(\mathbf{z}_1, \mathbf{z}_2)$ w.r.t its inputs $\mathbf{z}_1$ and $\mathbf{z}_2$. Since $\mathbf{B}$ and $\mathbf{Z}$ are both constant to the inputs of ρ, the differentiation is only applied to κ_u and $\tilde{\kappa}$ on $\mathbf{z}_1$ and/or $\mathbf{z}_2$. For example, $\partial\rho(\mathbf{z}_1, \mathbf{z}_2)/\partial x_2 = \partial\kappa_u(\mathbf{z}_1, \mathbf{z}_2)/\partial x_2 - \tilde{\kappa}(\mathbf{z}_1, \mathbf{Z}) \cdot \mathbf{B} \cdot \partial\tilde{\kappa}(\mathbf{Z}, \mathbf{z}_2)/\partial x_2 = \text{cov}\left(u(\mathbf{z}_1), \partial_x u(\mathbf{z}_2)\right) - \text{cov}(u(\mathbf{z}_1), \mathbf{f}) \cdot \mathbf{B} \cdot \text{cov}(\mathbf{f}, \partial_x u(\mathbf{z}_2))$ (note $\mathbf{z}_2 = (x_2, t_2)$). Hence, the kernel differentiation gives the same covariance (between the function and its derivatives) as in the predictive distribution (12), *i.e.,* $\partial\rho(\mathbf{z}_1, \mathbf{z}_2)/\partial x_2 = \overline{\text{cov}}(u(\mathbf{z}_1), \partial_x u(\mathbf{z}_2))$. That means the kernel links are still maintained in the posterior/predictive process (*i.e.,* conditioned on the data and differential equation).

4 Related Work

Physics-informed machine learning has become a rapidly growing area (Karniadakis et al., 2021; Edwards, 2022). Consider, for example, Raissi et al. (2019) and and subsequent work such as Mao et al. (2020); Zhang et al. (2020); Chen et al. (2020); Penwarden et al. (2021); Lou et al. (2021). The core idea is to use an NN to represent the solution function. The training objective includes a loss term to fit the boundary/initial condition and a residual term to fit the differential equation. The residual term is computed by applying the differential operators on the NN and then evaluating at a set of collocation points. The closer

the residual is to zero, the more the NN surrogate fits the equation. While PINNs have been successfully used to solve many forward and inverse problems, the differential operators in the residual term have also brought challenges in optimization (Krishnapriyan et al., 2021; Wang et al., 2022a; Edwards, 2022). This suggests the need for more refined optimization methods (such as augmented Lagrangian methods or sequential quadratic programming methods) to provide a more principled basis with which to combine domain-driven and data-driven models.

GPs have also been used for modeling or learning from physical systems. Early works (Graepel, 2003; Raissi et al., 2017) leveraged kernel differential methods to solve linear equations with observable sources: $Lu = g$, where L is a linear operator, u is the solution, and g is the source term. Graepel (2003) assume the data consists of noisy observation of g. Given the covariance (kernel) function of u, the covariance of g is obtained via kernel differentiation with operator L. Hence the kernel parameters and noise variance can be estimated from maximizing the marginal likelihood (1). It is also straightforward to calculate the predictive distribution of u given the observation of g via cross-covariance between u and g. Raissi et al. (2017) assumed both u and g have noisy observations, and hence a joint GP prior over u and g is constructed via kernel differentiation. Recent work (Wang et al., 2022b) instead used a similar formulation to PINNs to conduct deep kernel learning, *i.e.,* applying differential operators on the posterior function samples. While it is quite effective with deep kernels, this method does not perform well when reducing the deep kernels to commonly used shallow kernels.

Recently, Chen et al. (2021) used kernel differentiation to solve linear and nonlinear PDEs. This work minimizes the RKHS norm of the solution with constraints and/or regularizations that the equation is satisfied on a set of collocation points. From a high-level view, our method uses a similar strategy to integrate physics, *i.e.,* applying kernel differentiation, and learning from data fitting plus regularization on collocation points (*i.e.,* the data likelihood and virtual likelihood). However, both the modeling and inference are different. Our model is a nonparametric Bayesian model that creates a joint distribution over the solution function, its derivative functions, the noisy data and virtual observations, while Chen et al. (2021) used kernel ridge regression (square loss plus RKHS norm), a typical frequentist based kernel learning framework (Kanagawa et al., 2018). One might hope that that a Bayesian model is more amenable for reasoning under uncertainty. Second, our variational inference estimates the posterior distribution of the solution function values and its derivatives, rather than providing a point estimation (Chen et al., 2020). We also find an interesting insight that the posterior process (with Gaussian variational approximations) is still a GP and maintains the kernel differentiation property.

Another related work involves latent force models (LFMs) (Alvarez et al., 2009), which aim to integrate incomplete equations with unknown latent forces for GP learning. Based on the kernel of latent forces, the LFM convolves with the Green's function to derive the kernel of the target function, thereby encoding the physics into the induced kernel. However, LFMs are restricted to linear equations with available Green's functions. To overcome this issue, Alvarez et al. (2013) linearized the nonlinear terms in the equation. Hartikainen et al. (2012); Ward et al. (2020) focused on ODEs, using a linear time-invariant (LTI) stochastic differential equation (SDE) to represent the temporal GP prior over the latent forces, and converting the original ODE to an SDE. While successful, these methods do not apply to PDEs and time-spatial source functions.

5 Empirical Results

In this section, we evaluate AutoIP on two illustrative and two more realistic problems. The illustrative problems include a nonlinear pendulum system and a diffusion-reaction system, where the exact equations and the ground-truth solutions are known. Here, we can inspect the performance when AutoIP incorporates the full equation and when AutoIP uses only a part of the equation. The realistic problems are the prediction tasks of metal pollution and joint motion trajectories, for which the underlying governing equations are unknown. Here, we examined if AutoIP can improve the prediction accuracy by integrating a sensible physics model (not necessarily the ground-truth equation).

5.1 Nonlinear Pendulum

First, we evaluated AutoIP on a nonlinear pendulum system. Consider that a pendulum starts from an initial angle and velocity, and swings back and forth under the influence of gravity. We are interested in how the angle θ varies with time t. The equation is given by

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \sin(\theta) = 0, \tag{13}$$

where $\sin(\theta)$ is a nonlinear term, and we choose units so that the ratio between the magnitude of gravity field and the length of the string is one.

We set the initial angle to $\frac{3}{4}\pi$ and the initial velocity to zero. The change of θ exhibits apparent periodicity. See Fig. 1 and 2 first row (the black curves). We randomly collected 50 training examples from $t \in [0, 7.3]$ that covers around $\frac{3}{4}$ period. Then we randomly sampled 800 test examples from $t \in [0, 28.8]$ which covers around three periods. We implemented both AutoIP and standard GPR with Pytorch (Paszke et al., 2019), and we performed stochastic optimization with ADAM (Kingma and Ba, 2014). We used learning rate 10^{-2} and ran both methods with 10K epochs.

To overcome the perturbation of accuracy caused by the stochastic training, we examined the prediction accuracy after each epoch and used the best accuracy for comparison. We used the SE-ARD kernel for both AutoIP and GPR, with the same initialization. For AutoIP, we let κ_g and κ_u share the same kernel parameters. We examined our method with two settings: AutoIP-C, running with the *complete* equation (13); and AutoIP-I, running with an *incomplete* differential equation, in which the nonlinear term is replaced by an unknown source term $g(t)$,

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + g(t) = 0. \tag{14}$$

In both settings, we randomly sampled 20 collocation points across the whole domain $[0, 28.8]$ to integrate the equation. To obtain the ground-truth and to generate the training and test data, we used the `scipy` library to solve the initial value problem. We considered two training settings: (1) using exact training examples from the solution; and (2) using noisy training examples, where we added independent Gaussian noises sampled from $\mathcal{N}(0, 0.1\mathbf{I})$ to the solution outputs to form the training set.

In addition, we examined the case where the governing equation includes a damping term,

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \sin(\theta) + b\frac{\mathrm{d}\theta}{\mathrm{d}t} = 0, \tag{15}$$

where $b > 0$ is a constant and was set to 0.2. The damping can be due to a type of energy loss, such as friction. We used the typical assumption that the friction is proportional to the velocity. We randomly sampled 16 examples from $t \in [0, 6]$ for training and 800 examples across $t \in [0, 24.3]$ for testing. Again, we ran our methods in two ways. One is to integrate the complete equation (15), *i.e.,* AutoIP-C; the other is to integrate an incomplete equation in the form of (14), *i.e.,* AutoIP-I. In the latter case, the latent source g thereby subsumes both the nonlinear and damping terms. The number of collocation points was set to 20. While AutoIP-C leverages the complete equation, we do not assume the coefficient b of the damping term is known. Instead, we view it as an unknown equation parameter, and we jointly estimate it during the inference. We optimized b in the log domain to ensure its positiveness. Identical to the no-damping case, we adopted two training settings: exact examples; and noisy examples (with additive Gaussian noise generated from $\mathcal{N}(0, 0.1\mathbf{I})$).

For each case (damping/no-damping, exact/noisy training), we repeated the experiment five times, and we examined the average Root Mean-Square-Error (RMSE), average Mean-Negative-Log-Likelihood (MNLL), and their standard deviation. See Table 1. In all the cases, our method outperforms the standard GPR by a large margin. Even with an incomplete equation (including some unknown latent source), our method (AutoIP-I) still achieves a big improvement upon GPR, showing the advantage of effectively using physics knowledge. When integrating with the complete equation, our approach obtains even much better prediction accuracy (AutoIP-C). This is reasonable, because more precise and refined physics knowledge is leveraged. We have also compared with physics-informed neural networks. See the results and discussion in the Appendix.

No damping	RMSE	MNLL
GPR	1.354 ± 0.005	1.97 ± 0.015
AutoIP-I	0.585 ± 0.017	1.02 ± 0.013
AutoIP-C	$\mathbf{0.416 \pm 0.050}$	$\mathbf{0.892 \pm 0.032}$
With damping		
GPR	0.262 ± 0.0003	0.744 ± 0.008
AutoIP-I	0.212 ± 0.014	0.678 ± 0.02
AutoIP-C	$\mathbf{0.096 \pm 0.0035}$	$\mathbf{0.155 \pm 0.01}$

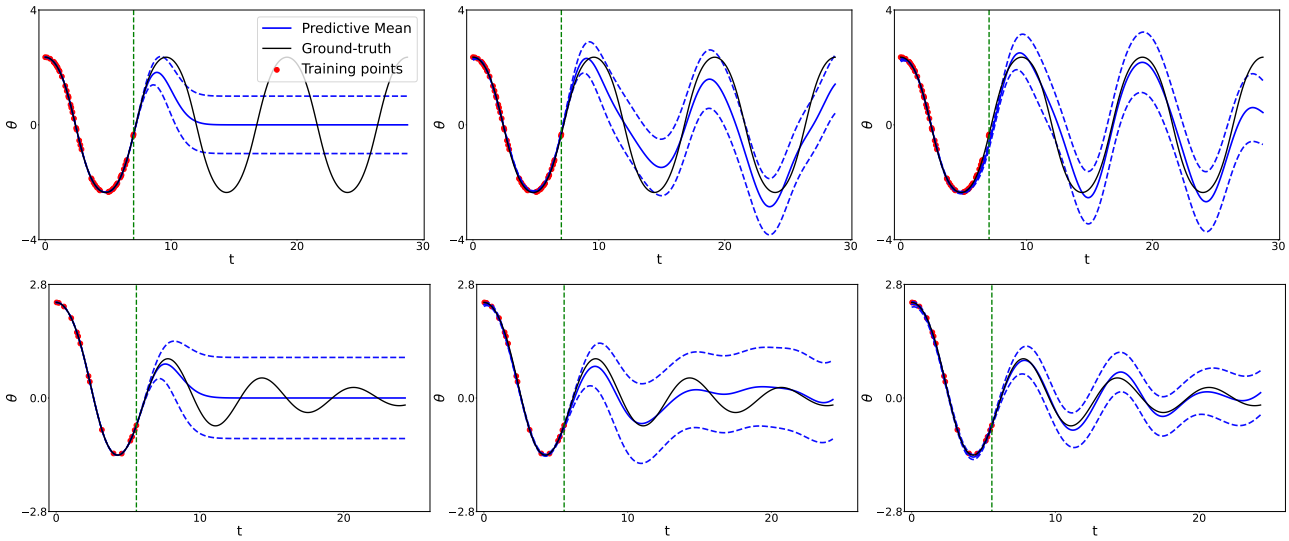
(a) Exact training data

No damping	RMSE	MNLL
GPR	1.44 ± 0.017	2.242 ± 0.055
AutoIP-I	0.691 ± 0.030	1.206 ± 0.024
AutoIP-C	$\mathbf{0.488 \pm 0.036}$	$\mathbf{1.061 \pm 0.028}$
With damping		
GPR	0.381 ± 0.018	1.07 ± 0.029
AutoIP-I	0.268 ± 0.013	0.937 ± 0.011
AutoIP-C	$\mathbf{0.133 \pm 0.010}$	$\mathbf{0.428 \pm 0.017}$

(b) Noisy training data

Table 1: Prediction accuracy in nonlinear pendulum systems with/without training noise and with/without the damping term, in terms of root-mean-square-error (RMSE) and mean-negative-log-likelihood (MNLL). AutoIP-I and AutoIP-C refer to our method using incomplete and complete equations, respectively. The results were averaged over five runs.

Next, we showcase the predictive mean and standard deviation of each method of one experiment, in Fig. 1 and 2, in contrast to the ground-truth. In all cases, GPR performs well in the training region. However, when moving away from the training region, the prediction of GPR quickly converges to the prior mean (zero), leaving a large predictive variance (uncertainty). See Fig. 1a and Fig. 2a. By contrast, with the effective use of differential equations, AutoIP can predict the target function quite accurately at places very far way from the training region, exhibiting much better extrapolation performance. It is surprising that even with an unknown source g (see (14)), with the key nonlinear term $\sin(\theta)$ and damping term θ' missing, AutoIP can still capture the variation of the target function quite well over a long range. See Fig. 1b and Fig. 2b. When integrating with the complete equation, AutoIP predicts the function values even closer to the ground-truth (see Fig. 1c and 2c). These together have shown the advantage of AutoIP in effectively leveraging different equations.



(a) GPR (b) AutoIP-I (c) AutoIP-C

Figure 1: Prediction in a nonlinear pendulum system with exact training examples. First row shows results without damping. Second row shows results with damping. Dashed lines are predictive mean $\pm$ standard deviation. Vertical line is the boundary of the training region.

Finally, we examined the estimated b in (15) by our method. For both noisy and exact training data, the estimation is quite good. For example, the estimated value by AutoIP-C in Fig. 1c and 2c is 0.2302 and 0.2352, respectively, giving 85% and 82.4% relative accuracy. Note that we only used 16 training examples and 20 collocation points. The average estimation from the five experiments for exact and noisy training data is 0.228 ± 0.002 and 0.232 ± 0.004, respectively.

5.2 Diffusion-Reaction System

We evaluated AutoIP on a diffusion-reaction system examined previously (Raissi et al., 2019). This system is governed by an Allen-Cahn equation along with periodic boundary conditions,

$$\frac{\partial u}{\partial t} - 0.0001\frac{\partial^2 u}{\partial x^2} + 5u^3 - 5u = 0, \qquad (16)$$

where $x \in [-1, 1]$, $t \in [0, 1]$, $u(0, x) = x^2 \cos(\pi x)$, $u(t, -1) = u(t, 1)$ and $u_x(t, -1) = u_x(t, 1)$.[1] We randomly sampled 256 training examples from $t \in [0, 0.28]$, and collected 100 collocation points from the whole input domain. Again, we tested our method using the complete equation (16), denoted by AutoIP-C, and the incomplete equation in the form of

$$\frac{\partial u}{\partial t} - 0.0001\frac{\partial^2 u}{\partial x^2} + g(x, t) = 0, \qquad (17)$$

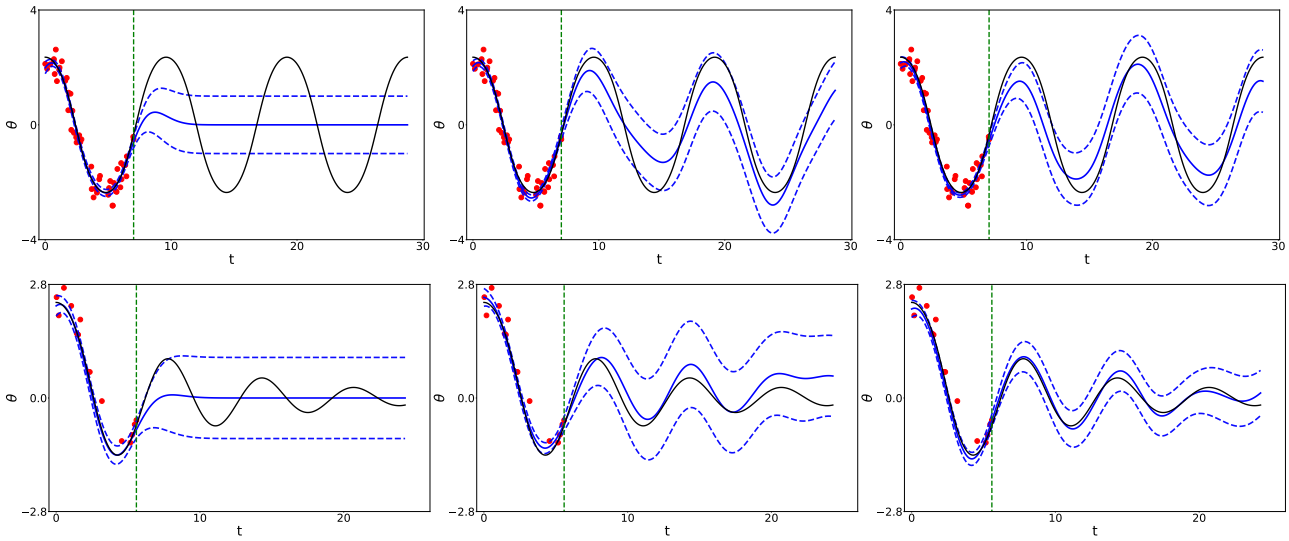
where g is an unknown source term (AutoIP-I). We ran GPR and our method for 200K epochs with the learning rate 10^{-3} (a larger learning rate will hurt the performance). The ground-truth solution is given by Fig. 3a. We show the prediction of GPR, our method with the incomplete equation (AutoIP-I), and our method with the complete equation (AutoIP-C) in Fig. 3b, 3c and 3d, respectively. As we can see, AutoIP is better able to capture the two reaction patterns, which look like two yellow flames. GPR, however, predicts a quite uniform reaction strength, losing its time variation. The overall RMSE confirms that AutoIP achieves a much better prediction accuracy. In Fig. 3e-g, we show the predictive variance of each method across the domain. Both AutoIP-I and AutoIP-P reduce the predictive uncertainty at places distant from the training region; see the red and yellow part on the right. The reduction from AutoIP with complete equation is even more significant (AutoIP-C), especially at the upper-half of the right end.

5.3 Motion Capture

We evaluated AutoIP on the prediction of the joint trajectories in motion capture. We used the CMU motion capture database.[2] We used the trajectories of subject 35 during walking and jogging, which lasted for 2,644 seconds. We considered joint 1 and joint 50. From each joint, we randomly sampled 100 examples from the first half of the trajectory for training, and we randomly collected another 800 examples across the whole trajectory for testing. Since the ground-truth differential equation that can characterize the motions is actually unknown, we used an incomplete one

[1] We used the solution data released in `https://github.com/maziarraissi/PINNs`.

[2] `http://mocap.cs.cmu.edu/`



(a) GPR (b) AutoIP-I (c) AutoIP-C

Figure 2: Prediction in a nonlinear pendulum system with noisy training examples. First and second row show the results without damping and with damping, respectively. Dashed lines are predictive mean $\pm$ standard deviation. Vertical green line is the boundary of the training region.

with a latent source term (Alvarez et al., 2009; 2013),

$$\frac{\partial u}{\partial t} + b \cdot u(t) - c = g(t), \tag{18}$$

where $b, c > 0$ are unknown coefficients and $g(t)$ is the latent source. In our method, we jointly estimate b and c in the log domain. To examine how the location of the collocation points will influence the performance of our method, we tested three settings: (1) AutoIP-T that uses the training inputs as the collocation points; (2) AutoIP-H that employs 200 random collocation points in the training region only, *i.e.,* half of the time span; and (3) AutoIP-W that employs 200 random collocation points across the whole time span of the trajectory. In addition to GPR, we also compared with latent force models (LFMs) proposed in Alvarez et al. (2009; 2013). LFMs use the kernel for the latent source g and the Green's function of the equation to perform convolution so as to derive an induced kernel for u, which includes b and c as the kernel parameters. We also used ADAM to train LFMs. We ran every method for 3K epochs with learning rate 10^{-2}, and we compared their best prediction accuracy (after each epoch). We repeated the experiments for five times, and calculated the average RMSE and NMLL, as listed in Table 2. As we can see, AutoIP always outperforms the competing methods. Since LFM on Joint 50 cannot give a reasonable test log likelihood (NMLL), we marked it as N/A, although its predictive mean is quite normal.[3] We can see that AutoIP-T and AutoIP-H are comparable in most cases. Since their collocations points are both from the time span of the first half trajectory, this shows that the randomness of the collocation points seem not have a major influence on the predictive performance. By contrast, AutoIP-W achieves much better prediction accuracy than AutoIP-T and AutoIP-H. This implies that a wider range of the collocation points (not the number) is more critical to improve the performance, especially in extrapolation.

Method	Joint 1	Joint 50
GPR	1.727 ± 0.026	0.257 ± 0.007
LFM	1.671 ± 0.016	0.257 ± 0.006
AutoIP-T	1.511 ± 0.007	0.224 ± 0.006
AutoIP-H	1.489 ± 0.03	0.225 ± 0.005
AutoIP-W	$\mathbf{1.103 \pm 0.027}$	$\mathbf{0.215 \pm 0.009}$

(a) RMSE

Method	Joint 1	Joint 50
GPR	1.368 ± 0.020	3.431 ± 0.242
LFM	1.721 ± 0.020	N/A
AutoIP-T	1.138 ± 0.024	2.615 ± 0.149
AutoIP-H	1.208 ± 0.081	2.664 ± 0.154
AutoIP-W	$\mathbf{1.121 \pm 0.084}$	$\mathbf{2.495 \pm 0.111}$

(b) NMLL

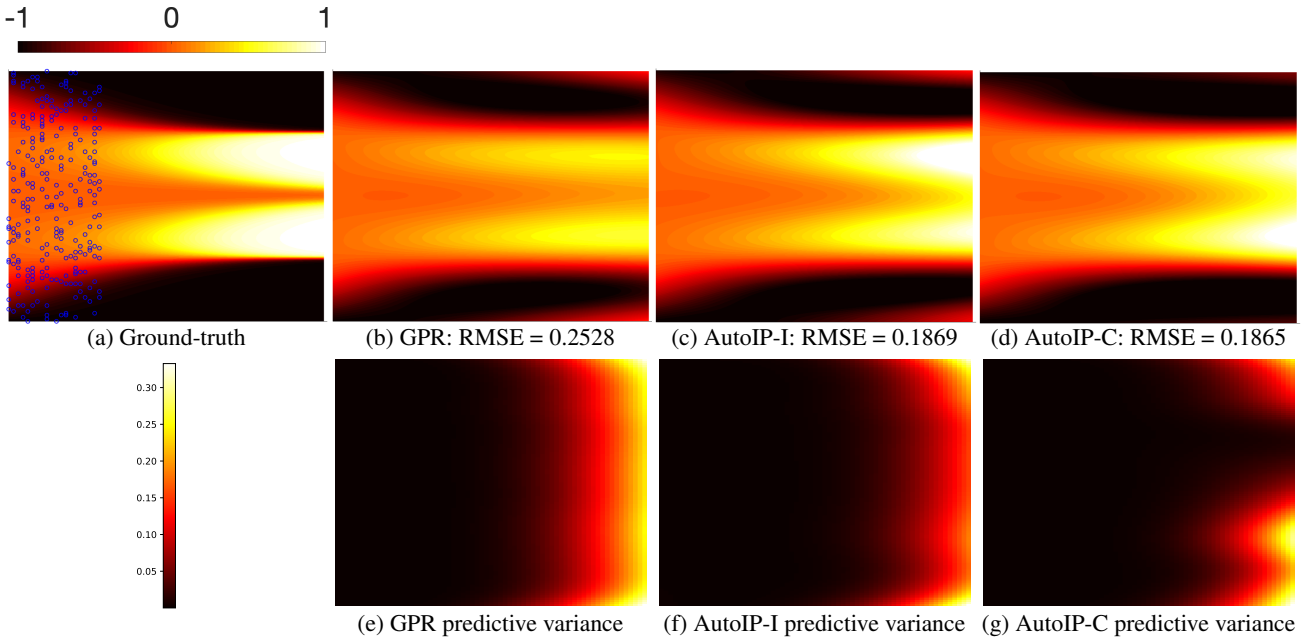
Table 2: Prediction accuracy on motion capture datasets.

5.4 Metal Pollution in Swiss Jura

We evaluated AutoIP on an application to predict the meta concentration with the Swiss Jura dataset.[4] The dataset

[3] We tried a variety of learning rates and initializations, but it either ended up with a non-positive definite covariance matrix (and crashed) or with very small test log likelihoods (10 times smaller than the competing methods), indicating a failure of learning. These might be due to some numerical issue in optimization with the induced kernel.

[4] `https://rdrr.io/cran/gstat/man/jura.html`



(a) Ground-truth (b) GPR: RMSE = 0.2528 (c) AutoIP-I: RMSE = 0.1869 (d) AutoIP-C: RMSE = 0.1865

(e) GPR predictive variance (f) AutoIP-I predictive variance (g) AutoIP-C predictive variance

Figure 3: Prediction in a diffusion reaction system. The horizontal axis is t while vertical axis is x. The first row consists of the ground-truth solution (where the blue circles indicate the training points used by all the methods), and the prediction made by GPR, AutoIP-I and AutoIP-C. The second row comprises of the predictive variance of each method across the domain.

	GPR	LFM	AutoIP
Task 1	0.299 ± 0.009	0.384 ± 0.010	$\mathbf{0.284 \pm 0.011}$
Task 2	0.304 ± 0.012	0.381 ± 0.011	$\mathbf{0.284 \pm 0.008}$
Task 3	0.232 ± 0.009	0.358 ± 0.005	$\mathbf{0.224 \pm 0.006}$
Task 4	0.261 ± 0.005	0.296 ± 0.005	$\mathbf{0.247 \pm 0.004}$

(a) RMSE

	GPR	LFM	AutoIP
Task 1	1.16 ± 0.064	1.36 ± 0.058	$\mathbf{1.10 \pm 0.069}$
Task 2	1.274 ± 0.093	1.471 ± 0.157	$\mathbf{1.219 \pm 0.129}$
Task 3	0.979 ± 0.058	1.31 ± 0.044	$\mathbf{0.849 \pm 0.067}$
Task 4	1.383 ± 0.098	1.496 ± 0.097	$\mathbf{1.303 \pm 0.091}$

(b) NMLL

Table 3: Prediction accuracy on Jura datasets.

includes measurements of seven metals (Zn, Ni, Cr, etc.) at 300 locations in a region of 14.5 km^2. The concentration is normally modeled by a diffusion equation, $\frac{\partial u}{\partial t} = \alpha \cdot \Delta u$, where Δ is the Laplace operator, $\Delta u = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}$. However, the dataset does not include the time information when these concentrations were measured. We followed prior work (Alvarez et al., 2009) to assume a latent time point t_s and estimate the solution at t_s, namely $h(x_1, x_2) = u(x_1, x_2, t_s)$. Thereby, the equation can be rearranged as,

$$\Delta h = g(x_1, x_2)$$

where $g(x_1, x_2) = \frac{1}{\alpha} \frac{\partial u(x_1, x_2, t)}{\partial t}|_{t=t_s}$ is viewed as a latent source term. Note that LFM views $u(x_1, x_2, 0)$ as the latent source, yet uses a convolution operation to derive an induced kernel for h in terms of locations, where t_s is considered as a kernel parameter jointly learned from data. We tested four tasks, namely predicting: (1) Zn with the location and Cd, Ni concentration; (2) Zn with the location and Co, Ni, Cr concentration; (3) Ni with the location and Cr concentration; and (4) Cr with the location and Co concentration. For each task, we randomly sampled 50 example for training and another 200 examples for testing. The experiments were repeated for five times, and we computed the average RMSE, average NMLL and their standard deviation. For our method, we used the training inputs as the collocation points. The results are reported in Table 3. AutoIP consistently outperforms the competing approaches, again confirming the advantage of our method.

6 Conclusion

AutoIP is a framework for Automatically Incorporating Physics into GPs. This approach samples the target functions and their derivatives in a probabilistic space and uses their relationships via a virtual likelihood defined by the differential equation. In the future, we will use RandNLA to extend our approach in large-scale applications.

Acknowledgements. This work was supported by MURI AFOSR grant FA9550-20-1-0358, NSF IIS-1910983 and NSF CAREER Award IIS-2046295. ASK was supported by LDRD funding under Contract Number DE-AC02-05CH11231 at LBNL and the Alvarez Fellowship in the CRD at LBNL. MWM would like to thank the NSF, DOE, and ONR for support of this work.

References

Allen, S. M. and Cahn, J. W. (1972). Ground state structures in ordered binary alloys with second neighbor interactions. Acta Metallurgica, 20(3):423–433.

Alvarez, M., Luengo, D., and Lawrence, N. D. (2009). Latent force models. In Artificial Intelligence and Statistics, pages 9–16.

Alvarez, M. A., Luengo, D., and Lawrence, N. D. (2013). Linear latent force models using Gaussian processes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(11):2693–2705.

Boyd, S. and Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press, Cambridge, UK.

Chen, Y., Hosseini, B., Owhadi, H., and Stuart, A. M. (2021). Solving and learning nonlinear PDEs with Gaussian processes. arXiv preprint arXiv:2103.12959.

Chen, Y., Lu, L., Karniadakis, G. E., and Dal Negro, L. (2020). Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics Express, 28(8):11618–11633.

Derezinski, M. and Mahoney, M. W. (2021). Determinantal point processes in randomized numerical linear algebra. Notices of the AMS, 68(1):34–45.

Drineas, P. and Mahoney, M. W. (2016). RandNLA: Randomized numerical linear algebra. Communications of the ACM, 59:80–90.

Edwards, C. (2022). Neural networks learn to speed up simulations. Communications of the ACM, 65(5):27–29.

Graepel, T. (2003). Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. In ICML, pages 234–241.

Hartikainen, J., Seppänen, M., and Särkkä, S. (2012). State-space inference for non-linear latent force models with application to satellite orbit prediction. In ICML.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI).

Kanagawa, M., Hennig, P., Sejdinovic, D., and Sriperumbudur, B. K. (2018). Gaussian processes and kernel methods: A review on connections and equivalences. arXiv preprint arXiv:1807.02582.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed machine learning. Nature Reviews Physics, 3(6):422–440.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. arXiv preprint arXiv:1312.6114.

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., and Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems, 34.

Lou, Q., Meng, X., and Karniadakis, G. E. (2021). Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation. Journal of Computational Physics, 447:110676.

Mahoney, M. W. (2011). Randomized algorithms for matrices and data. Foundations and Trends in Machine Learning, 3(2):123–224.

Mao, Z., Jagtap, A. D., and Karniadakis, G. E. (2020). Physics-informed neural networks for high-speed flows. Computer Methods in Applied Mechanics and Engineering, 360:112789.

Murray, I. and Adams, R. P. (2010). Slice sampling covariance hyperparameters of latent Gaussian models. In Advances in Neural Information Processing Systems.

Nocedal, J. and Wright, S. (2006). Numerical Optimization. Springer, New York.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32:8026–8037.

Penwarden, M., Zhe, S., Narayan, A., and Kirby, R. M. (2021). Multifidelity modeling for physics-informed neural networks (pinns). Journal of Computational Physics, page 110844.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Machine learning of linear differential equations using Gaussian processes. Journal of Computational Physics, 348:683–693.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). 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.

Wang, S., Yu, X., and Perdikaris, P. (2022a). When and why PINNs fail to train: A neural tangent kernel perspective. Journal of Computational Physics, 449:110768.

Wang, Z., Xing, W., Kirby, R., and Zhe, S. (2022b). Physics informed deep kernel learning. In International Conference on Artificial Intelligence and Statistics, pages 1206–1218. PMLR.

Ward, W., Ryder, T., Prangle, D., and Alvarez, M. (2020). Black-box inference for non-linear latent force models. In International Conference on Artificial Intelligence and Statistics, pages 3088–3098. PMLR.

Williams, C. K. and Rasmussen, C. E. (2006). Gaussian processes for machine learning, volume 2. MIT press Cambridge, MA.

Zhang, D., Guo, L., and Karniadakis, G. E. (2020). Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks. SIAM Journal on Scientific Computing, 42(2):A639–A665.

Appendix

We provide additional results for comparing with physics-informed neural networks (PINNs). Since PINNs only incorporate complete differential equations and are not Bayesian methods, we examined PINNs in the nonlinear pendulum system (Sec. 5.1) and the diffusion-reaction system (Sec. 5.2) with the complete equations given, and report the average RMSE and standard deviation.

A Nonlinear Pendulum

We tested PINNs with the same four settings in Sec. 5.1, namely, the equation with/without a damping term ((13) and (15)), combined with exact/noisy training data. We used the same training and test datasets for each run. For the NN architecture, we used `tanh` activation and two hidden layers. We varied the layer width from {5, 10, 50, 100}. For a fair comparison, we tested the PINN with the same number of collocation points as used by AutoIP, *i.e.,* 20 points. We also ran the PINN with 10K random collocation points. For training, we first ran 1,000 ADAM epochs with learning rate 10^{-3} and then ran L-BFGS with 50K maximum iterations and 50K maximum function evaluations. This is a popular practice of training PINNs [5]. The implementation is based on the code of Raissi et al. (2019)[6]. The average RMSE for five runs and standard deviation are reported in Table 4. As we can see, AutoIP-C largely outperforms the PINN in all the cases except when the equation includes a damping term and the training data does not include any noise. In that case, the PINN with 50 or 100 neurons per layer, and using 10K collocation points can solve the equation very accurately. However, when using the same few number of collocation points (20), the PINN with different architectures is consistently much worse than AutoIP, even when AutoIP only incorporates incomplete equations (*i.e.,* AutoIP-I). These results show that the performance of the PINN can be sensitive to the architecture design, the number of collocation points, and data quality, while AutoIP is quite promising and robust to different types of data, equations, and can work well with only a small number of collocation points.

Method	No damping/Exact training	No damping/Noisy training	Damping/Exact training	Damping/Noisy training
PINN-5 (20)	1.955 ± 0.214	1.895 ± 0.261	0.310 ± 0.019	0.310 ± 0.050
PINN-10 (20)	2.122 ± 0.179	1.824 ± 0.231	0.290 ± 0.018	0.342 ± 0.020
PINN-50 (20)	2.238 ± 0.541	1.927 ± 0.250	0.297 ± 0.044	0.361 ± 0.017
PINN-100 (20)	2.042 ± 0.273	2.407 ± 0.353	0.320 ± 0.074	0.384 ± 0.066
PINN-5 (10K)	1.479 ± 0.115	1.783 ± 0.297	0.110 ± 0.015	0.248 ± 0.037
PINN-10 (10k)	1.852 ± 0.320	1.548 ± 0.141	0.049 ± 0.023	0.194 ± 0.044
PINN-50 (10k)	1.367 ± 0.575	1.658 ± 0.074	$\mathbf{0.00007 \pm 0.00001}$	0.157 ± 0.051
PINN-100 (10k)	1.862 ± 0.584	1.993 ± 0.357	$\mathbf{0.00007 \pm 0.00002}$	0.186 ± 0.045
AutoIP-I	0.585 ± 0.017	0.691 ± 0.030	0.212 ± 0.014	0.268 ± 0.013
AutoIP-C	$\mathbf{0.416 \pm 0.050}$	$\mathbf{0.488 \pm 0.036}$	0.096 ± 0.004	$\mathbf{0.133 \pm 0.010}$

Table 4: Root Mean Square Error (RMSE). The results were averaged over five runs. "-{5, 10, 50, 100}" mean 5, 10, 50, and 100 neurons per layer; "(20)" means using 20 random collocation points while "(10K)" means 10K collocation points. Both AutoIP-I and AutoIP-C used 20 collocation points.

GPR	AutoIP-I	AutoIP-C	PINN (100)	PINN (10K)
0.2528	0.1869	0.1865	0.4388	**0.0169**

Table 5: RMSE in the diffusion-reaction system. "(100)" means using 100 random collocation points while "(10K)" means 10K collocation points. Both AutoIP-I and AutoIP-C used 100 collocation points.

B Diffusion-Reaction System

We used the same training and test datasets in Sec. 5.2. We followed (Raissi et al., 2019) to use four hidden layers with 200 neurons per layer, and `tanh` activation, to solve the Allen-Cahn equation. We tested the PINN with the same set of 100 collocation points as used by AutoIP, and 10K random collocation points sampled from the same domain. The training was done by first running 1,000 ADAM epochs with learning rate 10^{-3} and then L-BFGS with 50K maximum iterations and

[5] `https://github.com/lululxvi/deepxde`

[6] `https://github.com/maziarraissi/PINNs`

50K maximum function evaluations. The RMSE is given in Table 5. We can see that, with the same 100 collocation points, PINN is much worse than AutoIP. But with 100 times more collocation points, the PINN's performance is greatly improved. The results confirm the advantage of AutoIP when using a small number of of collocation points.