Coversheet: Gaussian Processes for Efficient Numerical Simulations in Physics

In this preamble to the final project report we describe progress since the midterm report and respond to comments and questions made by reviewers on that midterm report. Code for this project can be found at `https://drive.google.com/open?id=18CeHPPehNv9HcdRpsnpZowbKgjLSNbuN`.

We would like to acknowledge the fact that our final report is 9 pages instead of the specified 8. Due to the two-part nature of our project, we required a significant amount of background and methodological description, and as such were unable to further reduce the content of the report without impacting clarity and completeness. Some less important details are relegated to our appendix.

1 Progress

This project is separated into two components, both based on Gaussian process techniques. The first, investigated by Christophe Bonneville, is the interpolation of the solution of stationary linear partial differential equation (PDE) systems with respect to domain geometry and boundary conditions. The second, investigated by Maxwell Jenquin, is the numerical integration of time-dependent nonlinear PDE systems learned from two adjacent snapshots of the system's behavior and an assumption on the functional form of governing equation.

In the realm of domain and boundary condition interpolation, a number of new experiments and extensions have been made. As of the midterm report, estimation of the maximum temperature value for the heat equation on rectangular and arbitrary quadrilateral domains had been completed. In the final report, we also report maximum temperature results for arbitrary convex octagonal domains, and begin to explore predicting the temperature on a sparse set of systematically placed points within the domain using a number of independent GP models. Results for more complex boundary conditions are also described. In addition we apply this method to the linear elasticity equation, obtaining encouraging performance overall. One goal, extrapolation of characteristics beyond the range of training data, was not achieved, and is relegated to future work.

In the realm of time extrapolation, one core extension has been made. As of the midterm report, the model discovery scheme detailed in section 4.3 of the report had been used with an RBF kernel to extrapolate the dynamics of several 1-dimensional systems. In the final report, we derive and implement the corresponding model for the spectral mixture (SM) kernel, and test it on those same systems. We also report error statistics and detail the process of adapting the model to the SM kernel, with mixed results.

2 Responses to Selected Review Comments and Questions

- Some confusion on the part of the reviewers seems to have come from connecting domain interpolation to time extrapolation. These two sections of the project address different problems entirely, with different methods developed and employed. The common thread between them is the use of GPs to simulate PDE system solutions, but they do so in very different contexts and should be considered separate efforts with their own distinct methodology. We have included further description of that methodology in order to clarify this distinction.

- Both models have been explained in more detail, including data generation and kernel choice for the domain interpolation section. The functional form of the RBF kernel is assumed to be known and much of the theory of Gaussian processes is relegated to a citation of Rasmussen, but overall much more background is explicit.

- Error quantification in the time extrapolation section has been included, both in graphical and tabular form.

- With regards to computational cost of our proposed methods: In general GPs provide $\mathcal{O}(n^3)$ inference (n being the number of training points), but can be massively sped up using new techniques such as Structured Kernel Interpolation. Here we do not implement such techniques, but note that they provide the potential for significant speed improvements in future work. Regardless, naïve GPs tie general methods for time integration (without problem-specific structure, which is rare in practice and will be nearly impossible to beat with these methods). They can also massively outperform existing methods for FEM simulation, which are $\mathcal{O}(n_p^3)$, where n_p is the number of discretized domain points, easily much larger than the size of the training set. In fact, the GP methods described in the domain interpolation section are $\mathcal{O}(1)$ with respect to n_p.

- Further justification for several design choices (i.e. squared OU kernel vs no exponent or cubed) has been included.

- To the best of our knowledge, the time extrapolation framework is not extensible to true nonlinearity of the governing equation involved. This is due to the reliance on linearity of the operator $(I - \Delta t \mathcal{L}_x)$ to induce a joint GP prior by placing only a simple GP prior on one timestep.

- In the domain interpolation portion of this project we compare our methods only to FEM simulation, and not to finite difference methods. This is due to the fact that finite difference methods are only effective on very regular domains, whereas finite element approaches are adaptable to arbitrary domain geometry. Thus we have only compared computational efficiency of the GP approach to a similarly flexible option.

Gaussian Processes for Efficient Numerical Simulations in Physics
ORIE 6741 Project Final Report

Christophe Bonneville (cpb97@cornell.edu) [*1] **Maxwell Jenquin (mrj89@cornell.edu)** [*1]

[1]Cornell University, Ithaca NY, USA [*]Equal Contribution

Abstract

Two distinct problems in approximation of PDE systems are approached with Gaussian process methods. Domain and boundary condition interpolation for 2-dimensional linear PDE is explored and high-accuracy results are reported in a number of contexts with an Ornstein-Uhlenbeck kernel. Efficiency of this GP model is evident, and steps towards full solution prediction are also made. Time extrapolation of nonlinear systems with a kernel structure which encodes the PDE system is also explored. With a base RBF kernel, we report strong results, and with a base spectral mixture kernel mixed success is described. Efficiency in this context relies on future work in structured kernel interpolation, but flexibility of domain discretization is an immediate improvement over existing general methods of integration.

1. Introduction

Many physical phenomena, such as heat transfer, fluid behavior, and quantum mechanics, are governed by partial differential equations (PDE). As a result solving these equations accurately over a wide array of space and time domains is of critical interest to scientific and engineering applications. However, as is well known, there is no general method to solve PDE, and analytic solutions exist for only very few and usually simplistic problems. To address this issue, mechanistic numerical analysis methods such as finite elements and numerical integration schemes have been used to find approximate solutions. These methods have had tremendous success in the last 30 years and are widely used for simulations in physics and engineering. However, they are computationally expensive (complex or high-resolution simulations can take days to run) and some equations remain difficult to approximate. More efficient approaches which retain accuracy are clearly needed and would be a breakthrough in applied physics fields.

Machine learning provides a potential framework for avoiding the expensive computation required by mechanistic methods. In particular, recent papers (Umetani & Bickel, 2018; Raissi & Karniadakis, 2018) suggest that Gaussian processes (GPs) can be used along with results from simulations or measurements to predict key traits of related systems, or learn parameters from small snapshots of known systems. Even for the Navier-Stokes equation, these papers find that high-quality results can be recovered and interpolated using GP-based techniques. Additionally, GPs are inherently well-suited to problems of this nature because datasets are limited in size by computational expense. The confidence intervals resulting from such techniques are also useful tools for estimating the reliability of a solution (which is critical in engineering applications).

This project extends the approaches of the above papers. In section 2 we develop and present interpolation strategies for domain and boundary condition parameters. In section 3 we develop and test a method for extrapolation of time-dependent partial differential equation systems.

1.1. Gaussian Processes

Both sections of this project make use of Gaussian Processes, or GPs. This class of data model is a primary tool of Bayesian nonparametric machine learning, and can be conceptualized as a Gaussian distribution over functions, in that any finite number of point evaluations have a joint multivariate Gaussian distribution. As such, a GP is specified by its mean function μ and its covariance function k, that is for some function f which is distributed as $\mathcal{GP}(\mu, k)$ we have

$$\mathbb{E}[f(x)] = \mu(x), \quad \text{cov}[f(x), f(x')] = k(x, x')$$

While common practice is to set the prior mean $\mu = 0$, the covariance function or "kernel" k encodes important inductive biases into the model. Choice of the functional form of k, and then likelihood regression on its parameters, referred to as hyperparameters, with respect to the relevant dataset constitutes training a GP model. The model can then be used to predict the value of the generating function at new points in the domain, an $\mathcal{O}(n^3)$ operation, where n is the number of training points. Modern techniques such as Structured Kernel Interpolation (SKI) (Wilson & Nickish, 2015) allow for dramatically faster inference, which helps moti-

vate the use of Gaussian processes in this project, although implementation of these methods is reserved for later work. For a full discussion of the theory of Gaussian processes we refer the reader to Rasmussen & Williams (2005).

2. Interpolating Domain and Boundary Conditions

2.1. Related Work

2.1.1. Standard Methods for Steady-State PDE Approximation

To develop our method in this portion of the project we focus on the heat equation, a linear PDE describing the temperature behavior inside a domain Ω, given arbitrary scalar heat input q and prescribed temperatures $\bar{u}$ at the boundaries of Ω. The heat equation for steady-state (time independent) 2D problems is given by:

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = q, & (x,y) \in \Omega \subset \mathbb{R}^2 \\ u : \Omega \to \mathbb{R}, & u = \bar{u} \quad \text{for} \quad (x,y) \in \partial\Omega \end{cases} \tag{1}$$

Although no analytic solution exists, we may solve this equation approximately using numerical approaches. The most flexible approach is the finite element method (FEM), as described by Hughes (1987). The core concept of FEM is to approximate the solution on a set of n_p points throughout Ω. Doing so first requires creating a mesh of the domain, which takes up to $\mathcal{O}(n_p^2)$ computations (Ruppert, 1995). We can then build a linear system of equations and solve it to find an approximate solution vector $\tilde{u}$ of length n_p, which requires up to $\mathcal{O}(n_p^3)$ computations. Although FEM is a very popular technique in engineering and physics, it is time consuming, especially when a fine discretization is needed (large n_p). Note that while techniques exist to decrease the number of computations for solving the linear system, what takes the longest in practice if often mesh generation as it is an iterative process and difficult to parallelize (also n_p is not generally known in advance). In engineering design processes, we often need to re-iterate simulations to obtain a satisfactory design, leading to potentially redundant computations. We are also usually interested in the extrema of $\tilde{u}$ rather than $\tilde{u}$ itself (e.g. Ω could represent a chip heating inside a computer and we want to compare the maximum temperature with the material melting point), which motivates a more efficient manner of extracting these relevant values rather than going through the full process of FEM simulation.

2.1.2. Existing Work Towards Solution Prediction

Despite advances in parallel computing, the drawbacks of FEM techniques described above are still inherent to the approach. Machine learning techniques are a promising tool for improving efficiency in that respect. One major issue standing in the way has historically been domain parameterization: in order to be suitable inputs for machine learning techniques, a wide range of domains must be described by a few consistent parameters. This issue has been partly addressed by Umetani & Bickel (2018), with the development of a novel parameterization method to describe complex geometries with a systematic number of features. Subsequently, they created a dataset of FEM simulation results for the flow of air around car bodies, on which they trained a Gaussian process with an ARD kernel. They were able to obtain fair flow predictions for interpolant car models with relatively little computation time. The core assumption in developing such a method is that once the data is generated and the GP model is trained, the computational cost of making a prediction is significantly lower than using FEM. Their work is a valuable proof of concept and shows that predicting PDE solutions with Gaussian processes is feasible and efficient. In this portion of the project we propose to extend their work by finding an expressive kernel especially suited for linear PDE and robust to complex domains and advanced boundary conditions.

2.2. Interpolation Model

Following the work of Umetani & Bickel (2018), we used an FEM solver to generate a dataset of results for the heat equation in a rectangular domain $\Omega = a \times b \subset \mathbb{R}^2$ with an intermediate level of discretization. For each simulation, four input features were randomly drawn from a uniform distribution: the length and width of the domain (a and b), the scalar heat input q (with units of temperature per area), and the boundary condition $\bar{u}$ (with units of temperature). The resulting input feature vector was $\mathrm{X}_i = [a, b, q, \bar{u}]$ and the target variable was $y_i = \max\{\tilde{u}\}$ (maximum temperature within Ω). Arbitrarily, we choose each input feature to have values within the range $[0, 100]$. The procedure, outlined in algorithm 1 is used to generate both the training set and the test set. Although existing work uses an ARD kernel, we can exploit the linear nature of the heat equation for better prediction. Indeed, if one of the features changes while the other three remain fixed, the temperature throughout the domain varies linearly. Therefore, we base our model on the Ornstein-Ulhenbeck (OU) kernel which acts as a strong pointwise interpolator and is robust to noise (as will be seen later however, a squared Ornstein-Ulhenbeck kernel surprisingly tends to provide slightly better results).

Algorithm 1. Data Generation

Initialize X, **y**
for $i = 1$ **to** n **do**
  Draw a, b, q, $\bar{u} \sim$ U[0,100]
  $mesh \Leftarrow$ MeshGenerator(a, b)
  $\tilde{u} \Leftarrow$ FEMSolver($mesh$, q, $\bar{u}$)
  $\mathrm{X}_i = [a, b, q, \bar{u}]$
  $y_i = \max\{\tilde{u}\}$
end for

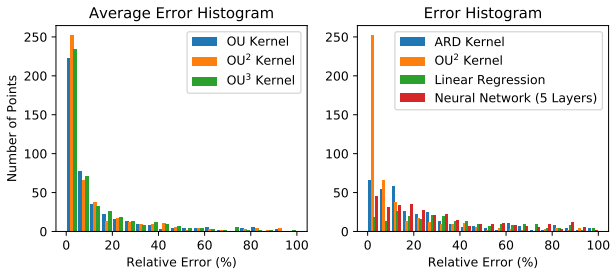


Figure 1. Relative error histograms between the predicted maximum temperature and the test set for rectangular domains using different kernels and ML models ($n_{\text{train}} = n_{\text{test}} = 500$)

The prior and the kernel have the form:

$$\mathbf{f} \sim \mathcal{GP}(0, k(x, x')) \tag{2}$$

$$k(x,x') = \theta_i k_{\text{OU}}^2(x, x') \tag{3}$$

$$k_{\text{OU}}(x,x') = \exp\big(-\frac{|x - x'|}{\theta_j}\big) \tag{4}$$

As shown in Rasmussen & Williams (2005), the corresponding predictive distribution is:

$$f_* \sim \mathcal{GP}(\mu(x_*, x, \mathbf{f}), \Sigma(x_*, x)) \tag{5}$$

Therefore, the predictive mean maximum temperature is $u_*^{\max} = \mu(x_*, x, \mathbf{f}) = k(x_*, x)k(x, x)^{-1}\mathbf{f}$. To implement our GP model we use the Python library GPyTorch (Gardner et al., 2018).

MODEL	OU	OU^2	OU^3	ARD	NN	LR
MSE/10^5	6.6	3.8	3.8	6.8	88	$7 \cdot 10^5$

Table 1. Maximum temperature prediction MSE for rectangular domains using different kernels and ML models

2.3. Results

2.3.1. Rectangular Domains

We train our interpolation model using 500 data points. Figure 1 shows relative error histograms between our prediction and the test set. The squared OU tends to give slightly better results then both the standard OU kernel and the cubed OU kernel, with a larger number of test points falling in the lowest-error region. Note that the left histogram shows an average error over three training attempts, so the differences in accuracy should be attributed to the kernels themselves and not the stochastic optimization process. The squared OU kernel also outperforms the ARD kernel, standard linear regression and a 5-layer neural network (see appendix A). The mean squared error (MSE) for each model is summarized in table 1. Although the MSE for the squared OU kernel and the cubed OU kernel are similar, we ultimately prefer the squared OU kernel for its slightly higher accuracy. Note that the absolute MSE values are large because the temperature targets lie in the range $y_i \in [0, 7{\cdot}10^5]$ as per FEM simulation.

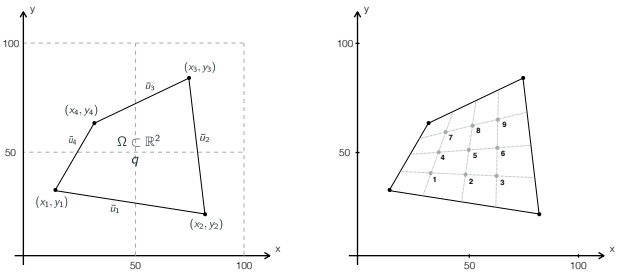


Figure 2. Parameterization of the PDE problem for any given quadrilateral and non uniform boundary conditions (left), and internal reference points (right)

2.3.2. Polygon Domains and Advanced Boundary Conditions

We now extend our approach to more complex domains and boundary conditions. Following the same process described in algorithm 1, we created a new data set for random quadrilaterals with varying prescribed temperatures. Each quadrilateral is fully described by the Cartesian coordinates of its vertices listed in counter clockwise order, along with the boundary conditions (Figure 2, left). This yield an input vector with 13 features of the form $\mathrm{X}_i = (x_1, y_1, ..., x_4, y_4, q, \bar{u}_1, ..., \bar{u}_4)$. Similarly, we create another dataset for octagonal domains, leading to 25 features. Figure 3 shows the error histogram with respect to the test set (200 points). There is no drop in accuracy between quadrilaterals and octagons, even though the number of features almost doubles and the same number of training points were used for both cases (500 points).

	DOMAIN	OU^2	NN	LR
$MSE/10^5$	QUAD	14.7	62.7	75.7
$MSE/10^5$	OCT	0.25	9.7	0.7
$MSE/10^5$	OCT V2	0.22	14.7	0.6

Table 2. Maximum temperature prediction MSE for quadrilateral and octagon domains

When generating these datasets, even if the parameters $\bar{u}_i$ and q are drawn uniformly, the corresponding target values are not strictly uniformly distributed. Therefore, in order to improve the prediction accuracy we pad y where points in the data space are too sparse. If there is no heat input nor boundary conditions ($q = \bar{u}_i = 0$), the maximum temperature will be 0, so we also include additional data points to make sure our model accurately captures such special cases. Figure 4 shows the results for octagon domains using an updated dataset. The accuracy improves quite significantly compared to the raw dataset used in figure 3. The MSE are summarized in table 2. Again, the squared OU kernel outperforms other machine learning models.

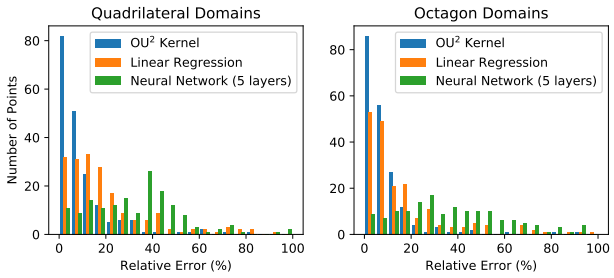


Figure 3. Relative error histograms between the predicted maximum temperature and the test set for quadrilateral and octagon domains ($n_{\text{train}} = 500$, $n_{\text{test}} = 200$)

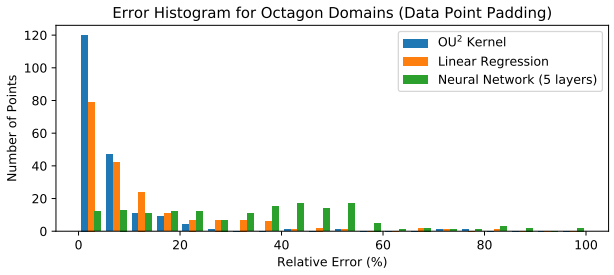


Figure 4. Relative error histograms between the predicted maximum temperature and the test set for octagon domains with updated training set ($n_{\text{train}} = 500$, $n_{\text{test}} = 200$)

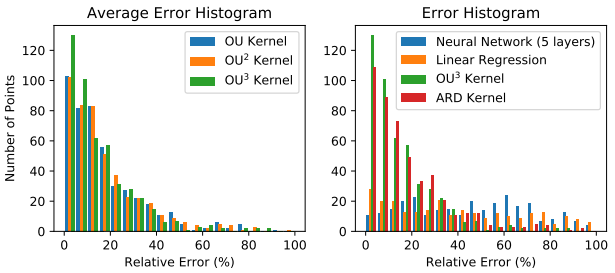


Figure 5. Relative error histograms between the predicted maximum stress and the test set for rectangular domains ($n_{\text{train}} = n_{\text{test}} = 500$)

2.3.3. APPLICATION TO OTHER LINEAR PDE

Next this model was applied to the linear elasticity problem (equation 6). This equation is a set of tensor-valued linear PDE describing the deformation of objects given their material elasticity constant E, weight p and some prescribed displacement $\bar{u}$ (e.g when the object is stretched or squeezed). The goal is to solve for the stresses S describing the internal pressure inside the domain (when the maximum stress exceed the material strength, the object breaks).

$$\begin{cases} \nabla \cdot \mathrm{S}(x,y) = -\mathrm{p} & (x,y) \in \Omega \subset \mathbb{R}^2 \\ \mathrm{S}(x,y) = \mathrm{E} \cdot (\nabla u^{\mathrm{T}} + \nabla u)/2 & \\ \mathrm{S} : \Omega \to \mathbb{R}, \qquad u = \bar{u} \quad \text{for} \quad (x,y) \in \partial\Omega & \end{cases} \tag{6}$$

As in section 2.3.1, we consider a rectangular domain but we are now interested in the maximum stress throughout Ω. The feature vector has the form $\mathrm{X}_i = [a, b, \mathrm{E}, \mathrm{p}, \bar{u}]$ and the target is $y_i = \max\{\tilde{\mathrm{S}}\}$. Figure 5 shows the error histogram for stress prediction. Surprisingly the cubed OU kernel now performs significantly better then both the standard OU kernel and the squared OU kernel. It also performs better than the ARD kernel and clearly outperforms linear regression and neural networks. The predictions in this case are less accurate than the heat equation, as a large number of test points are above 20% error. Given time constraints we could not improve the accuracy significantly, but the results we obtained suggest that the model itself is promising and that by tuning an OU kernel with the relevant power we may make accurate predictions for many linear PDE.

2.3.4. TOWARDS FULL SOLUTION PREDICTION

Although so far we have emphasized extrema prediction, let us now consider full solution prediction for the heat equation on arbitrary quadrilaterals. This is a challenging problem because it requires the prediction of temperature readings at different locations throughout the domain. In order to standardize the location of these points, we consider 9 internal reference points inside the domain. The location of these

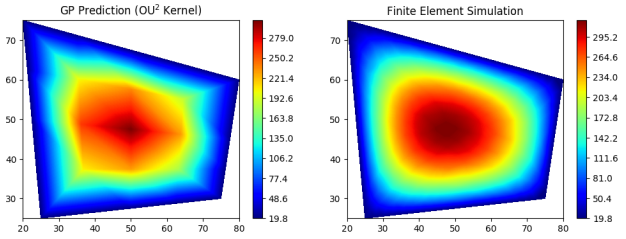


Figure 6. Full solution prediction using GP with squared OU kernel (left) and FEM (right). $q = 2$ and $\bar{u} = 20$

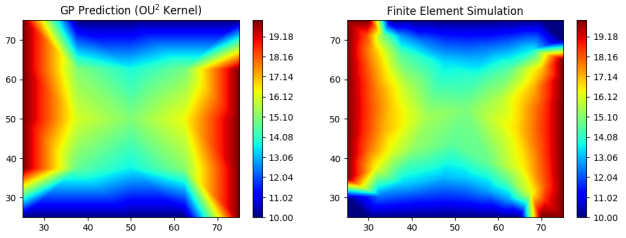


Figure 7. Full solution prediction using GP with squared OU kernel (left) and FEM (right). $q = 0$ and $\bar{u}_1 = \bar{u}_3 = 10$, $\bar{u}_2 = \bar{u}_4 = 20$

reference points only depends on the Cartesian coordinates of the vertices, so we can calculate them analytically using the input feature vector (Figure 2, right). As a result, we only need to predict 9 temperature readings but not their corresponding locations (which limits the number of task and the risk of error). Therefore when generating the dataset, instead of considering the maximum temperature we sample 9 temperature readings at each reference point, yielding an output target vector of the form $y_i = [\tilde{u}_1,...,\tilde{u}_9]$. The outputs are clearly correlated, so we could use a multitask GP for better accuracy and computational efficiency. Unfortunately, the current multitask implementation in GPyTorch does not provide satisfactory results so we train 9 independent single-output models instead. We may then make predictions at the 9 corresponding reference points and linearly interpolate the predicted temperatures over the whole domain. As shown in figure 6 and 7, the GP prediction captures very well the temperature behavior inside the domain and well-approximates the temperature readings (as compared to the finite element simulation).

2.4. Discussion

As we have seen, GPs can make very accurate predictions if the dataset fully represents the behavior of the PDE. Using an Ornstein-Ulhenbeck to the appropriate power (2 or 3 in practice), GPs outperform other machine learning models and require much smaller datasets. They are also much faster than finite elements (up to 80 times speedup). Table 3 shows the average time taken by a GP to make a new prediction (once the model has been trained), as compared to FEM simulation for the same problem. Even for full solution prediction where 9 different models made predictions, the improvement in efficiency is still significant (5 times faster). Note that for the full solution, using multitask GP models could yield faster computation, as well as using an inducing point method like SKI (Wilson & Nickish, 2015). Finally, it should be noticed that the computation time only depends on the training set size. Therefore, we may use highly discretized FEM simulation when creating the datasets, which would lead to more accurate solutions while maintaining identical prediction time.

DOMAIN/EXAMPLE	GP	FEM
RECTANGLE	0.10S	8.05S
QUADRILATERAL	0.20S	8.88S
OCTAGON	0.22S	13.90S
FULL SOLUTION (FIG 6)	2.06S	9.85S
FULL SOLUTION (FIG 7)	3S	15.9S

Table 3. Average computation time for new predictions with GPs, compared to FEM

3. Extrapolating in Time

3.1. Introduction

Time-dependent partial differential equation (PDE) systems govern the majority of fundamental physical processes and phenomena, but are often difficult to describe and simulate with numerical methods. Even more challenging perhaps is identifying (and extrapolating the dynamics of) a PDE system given a small number of measurements of its behavior. In pursuit of this goal, Raissi & Karniadakis (2018) have proposed and demonstrated a method for learning the parameters of a homogeneous PDE system using linearization of the differential operator and Gaussian process (GP) regression, via so-called numerical Gaussian processes (Raissi et al., 2018). In this section of the project we use and modify their model to explore the extrapolation of dynamics of a semi-undiscovered PDE system, as a potential alternative to standard methods of integration when PDE parameters are not known.

3.2. Related Work

3.2.1. STANDARD METHODS FOR PDE INTEGRATION

If the PDE which governs a time-dependent system is explicitly defined, there exist many standard methods for integration, most of which are specified to the particular type of PDE involved in the problem. In addition to time-dependent finite-element methods related to those described in section 2, finite-difference methods such as Crank-Nicolson (LeV-

eque, 2007) are popular choices for certain problems due to properties like energy conservation (for elliptic PDE) and efficiency for certain linear problems. For some problems, spectral methods (Kopriva, 2009) are employed, resulting in a similar computational cost to finite element methods.

All of these methods suffer from some form of matrix inversion as a requirement, therefore making them $\mathcal{O}(n^3)$ without problem-dependent special structure. In addition they fix the domain discretization for the length of the entire simulation, and generally restrict that discretization to some level of detail to ensure convergence. Because of this there is room for Gaussian process-based approaches to be not only strong predictors in the case where the system is not precisely known, but also to compete when the system is known exactly. Modern advancements in inducing-point methods such as structured kernel interpolation (Wilson & Nickish, 2015) also provide the potential for future work on GP extrapolation to be more efficient than standard methods for many extrapolation tasks.

3.2.2. Existing Work Towards PDE Discovery

Following Raissi and Karniadakis (2018), suppose we have a PDE system on a 1-D domain governed by the following equation, where $\mathcal{L}_x$ is a linear differential operator which contains only spatial derivatives. If a PDE has a nonlinear operator we make a linear approximation (see appendix B) and proceed.

$$\frac{\partial}{\partial t}u(x,t) = \mathcal{L}_x u(x,t) \tag{7}$$

Further suppose that we measure the system at some points along its domain twice, with measurements close enough in time that we may approximate the evolution of the system via the backwards Euler finite difference equation,

$$u_n \approx u_{n-1} + \Delta t \mathcal{L}_x u_n. \tag{8}$$

Here we will use the notation $u_j = u(x, t_j)$. Then by placing a GP prior on u_n we use equation 8 to induce a GP prior on u_{n-1}, since the two are related by a linear operator and any linear transformation applied to a GP results in a GP:

$$u_n \sim \mathcal{GP}(0,k) \Rightarrow u_{n-1} \sim (I - \Delta t \mathcal{L}_x)\mathcal{GP}(0,k) \tag{9}$$

where I is the identity. This induces a joint GP distribution over our two measurements,

$$\begin{bmatrix} u_n \\ u_{n-1} \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k & k_{1,0} \\ k_{0,1} & k_{0,0} \end{bmatrix} \right) \tag{10}$$

where we have defined

$$k_{1,0}(x,x') = (I - \Delta t \mathcal{L}_x)k(x,x') \tag{11}$$
$$k_{0,1}(x,x') = (I - \Delta t \mathcal{L}_{x'})k(x,x') \tag{12}$$
$$k_{0,0}(x,x') = (I - \Delta t \mathcal{L}_{x'})(I - \Delta t \mathcal{L}_x)k(x,x') \tag{13}$$

For further details refer to appendix B. Existing work uses this joint distribution, along with a radial basis function (RBF) kernel as $k(x,x')$ with analytic derivatives to define the other sub-kernels (the RBF kernel is by far the most standard smooth interpolation kernel). The resulting model is used to learn the coefficients of $\mathcal{L}_x$ as hyperparameters of the joint kernel, given the functional form of the linear operator. The powerful analytic structure encoded in the joint kernel captures both the backwards Euler integration scheme and the overall structure of $\mathcal{L}_x$. These inductive biases will allow us to use the joint GP prior to extrapolate the time dynamics of the system.

It should be noted here that Raissi and Karniadakis have also done work on placing GP priors on general Runge-Kutta methods to create what they describe as a "Numerical Gaussian Process" (Raissi et al., 2018). As such, this backwards Euler integration scheme has the potential to be replaced by a scheme of higher-order accuracy or one which conserves energy for more faithful reproduction of system evolution. Here we have chosen the backwards Euler scheme for simplicity and demonstration, but certainly more accurate methods would improve results at the cost of more computation.

3.2.3. The Spectral Mixture Kernel

Introduced by Wilson & Adams (2013), the spectral mixture (SM) kernel approximates any stationary kernel via a mixture of Gaussian distributions in its spectral density. In one dimension it is given by

$$k(\tau) = \sum_{q=1}^{Q} w_q \exp(-2\pi^2\tau^2\nu_q)\cos(2\pi\tau\mu_q) \tag{14}$$

where $\tau = x - x'$, and the spectral density of k is a mixture of Q Gaussians, with means μ_q and variances ν_q, weighted by w_q. In addition it has reasonably simple analytic derivatives (see appendix C), which makes it convenient for use in the framework described by Raissi and Karniadakis. The SM kernel has strong inductive biases which make it suitable for extrapolating periodic behavior, and as such this project was begun with the idea that those biases may prove helpful in improving the accuracy of the model put forth by Raissi and Karniadakis, and extended here.

3.3. Extrapolation Model

In existing work, the dynamics of a PDE system have been encoded into a GP model and trained. But, no attempt to explore the use of that model to simulate dynamics has been made. Here we extend and develop the method in section 3.2.2 to extrapolate the dynamics of the discovered PDE system. After measuring u_n and u_{n-1} and training the joint GP model described, we treat the joint distribution between u_n and u_{n+1} (where $t_{n+1} - t_n = t_n - t_{n-1}$) as a predictive distribution for u_{n+1} (Rasmussen & Williams, 2005). That is, with trained kernel functions, our predictive distribution yields:

$$\begin{bmatrix} u_{n+1} \\ u_n \end{bmatrix} \sim \mathcal{GP}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k & k_{1,0} \\ k_{0,1} & k_{0,0} \end{bmatrix} \right) \tag{15}$$

$$\Rightarrow \mathbb{E}[u_{n+1}] = k_{1,0}(x_n, x_{n+1}) \Big(k(x_n, x_n) + \sigma I \Big)^{-1} u_n \tag{16}$$

where σ is the learned noise variance of the posterior model, x_n is the set of sampled locations at the second measurement and x_{n+1} is the set of x values at which we would like to predict the solution into the future. This process is repeated iteratively, evaluating the predictive distribution for the induced joint GP posterior over u_{n+1} and u_{n+2}, and onward into the future.

To motivate this process, we note that any and all standard methods of integration are both non-scalable and rigid in discretization. That is, no fast techniques exist to approximate them as discretization becomes fine or domain becomes large, and the domain approximation is fixed for all subsequent timesteps, often in a grid pattern. In contrast, a GP model would allow both scalability (via structured kernel interpolation), and flexibility of domain discretization from timestep to timestep. For example, a GP model could be used to increase resolution adaptively near sharp variations in a system as it evolves, a concept only attainable by complicated multiscale parameterizations or slow adaptive grid techniques in standard literature.

3.4. Methodology and Experiment Design

Raissi and Karniadakis provided MatLab code for their "Hidden Physics Model", which we adapted to our purposes. Beginning with integration using the RBF model, we examined two nonlinear systems, namely a 1-dimensional Kortweg-de Vries system with solition peaks, and a 1-dimensional Kuramoto-Sivashinksy system with a single peak devolving into chaotic interactions after a short time. For the RBF model we also explored a 1-dimensional nonlinear Schrödinger equation, which can be found in appendix D.

The Kuramoto-Sivashinsky system was governed by the equation

$$\frac{\partial u}{\partial t} = -\lambda_1 u \frac{\partial u}{\partial x} - \lambda_2 \frac{\partial^2 u}{\partial x^2} - \lambda_3 \frac{\partial^4 u}{\partial x^4} \quad \text{where } \lambda_1 = \lambda_2 = \lambda_3 = 1, \tag{17}$$

and the Kortweg-de Vries system studied was governed by the equation

$$\frac{\partial u}{\partial t} = -\lambda_1 u \frac{\partial u}{\partial x} - \lambda_2 \frac{\partial^3 u}{\partial x^3}, \quad \text{where } \lambda_1 = 6, \lambda_2 = 1. \tag{18}$$

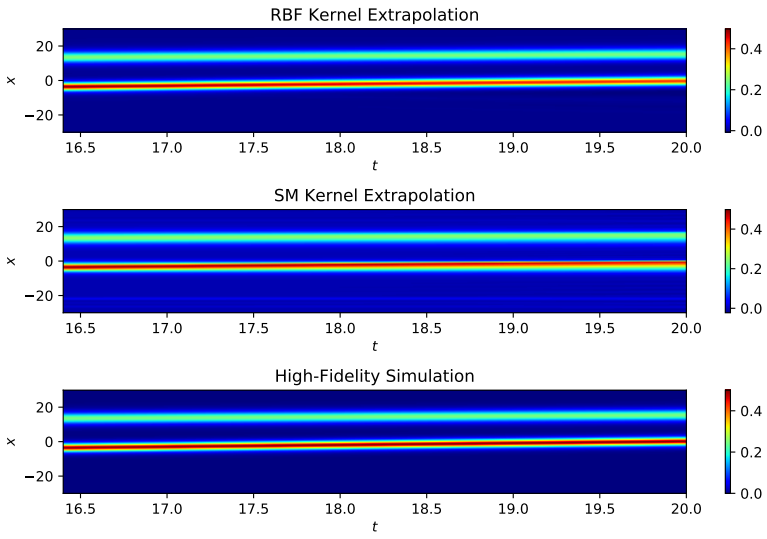
While Raissi and Karniadakis focused on determining the coefficients λ_i, our goal was to recreate the dynamics of the system. For both systems, high-fidelity simulations were used as "ground truth" for comparison, and the system was learned from samples at two adjacent timesteps of those simulations (as in Raissi & Karniadakis (2018)). After sampling, the hyperparameters of the model were initialized and trained using Rasmussen's `minimize.m` function, a standard tool when analytic derivatives are known. Once trained, the model was used to iteratively predict the following timestep as in equation 16. This process was repeated with 1% noise added to the sampled data.

Next, for both systems, the spectral mixture kernel model was employed as well. Here we note that the SM kernel must be initialized carefully, with the empirical spectrum of the data taken into account. Because the relationship between the spectrum of u_{n-1} and the implied spectrum of the complex joint kernel is unclear, the base SM kernel $k(\tau)$ was initialized to hyperparameters based on the spectrum of only u_n, then trained for several iterations based again only on data in u_n. Then, to discover coefficients and train the remainder of the hyperparameters, the SM kernel hyperparameters were fixed and `minimize.m` was employed. As in the RBF case, the trained GP model was then used to predict the dynamics of the systems iteratively. This process was repeated with 1% noise added to the sampled data.

Q	1	2	3	4	5	6
n_{SUCCESS}	11	8	28	55	56	49

Table 4. Number of successful fits of the KdV model (less than .1 relative error in 2-norm) to u_n following training, with 100 trials for each Q value.

One design choice in the SM case that does not exist for the RBF model is the number of mixture components Q. In both models we chose $Q = 4$, since as can be seen in table 4 the frequency of high-quality (less than 1% overall relative error on a test prediction of u_n) fit was near-optimal, and choosing $Q = 5$ would have resulted in more computational



RBF MODEL

DATA	CLEAN	NOISY
LOG(NOISE)	-17.18	-13.80
λ_1	6.11	5.72
λ_2	1.05	0.96
PEAK ERROR	0.11	0.13
AVERAGE ERROR	.08	.10

SM MODEL

DATA	CLEAN	NOISY
LOG(NOISE)	-15.67	-12.80
λ_1	2.04	2.38
λ_2	1.21E-7	-8.65E-7
PEAK ERROR	.47	.45
AVERAGE ERROR	.37	.34

Figure 8. Results of an example KdV simulation for RBF kernel and SM kernel with clean data (left), and statistics on error and discovered coefficients (right). Here "Peak Error" refers to the largest error magnitude across the simulation, and "Average Error" refers to the average relative error in 2-norm over all simulated timesteps.

cost with only a small marginal benefit. This analysis was not performed for the KS system due to the length of training time for $Q \geq 4$.

3.5. Results and Discussion

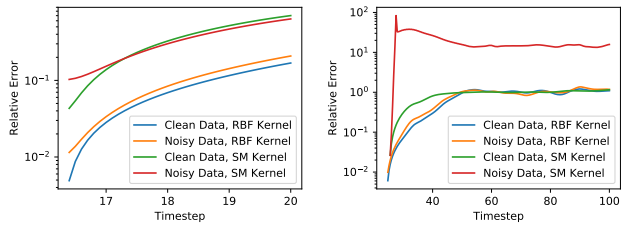
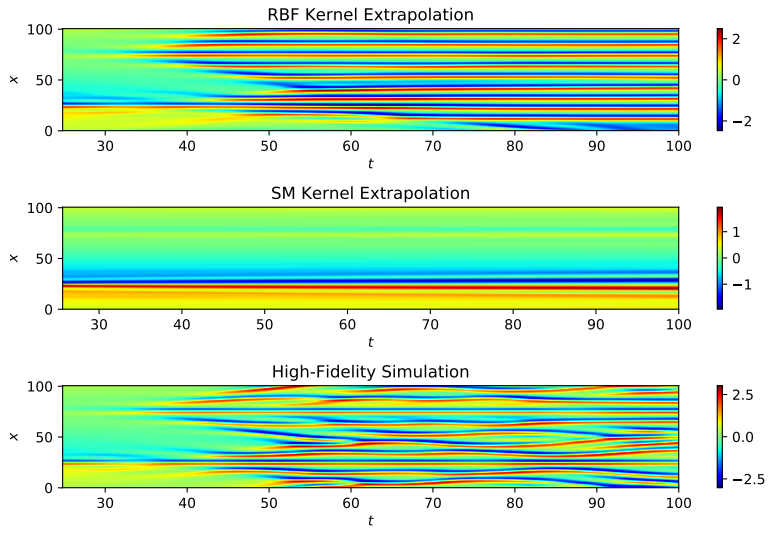


Figure 9. Relative error vs timestep for KdV system (left) and KS sytem (right).

In figure 8, we see both models in action on the Kortweg-de Vries (KdV) system. Both models recreate the qualitative dynamics of the system with reasonable accuracy, although a close inspection reveals that the RBF model maintains a more symmetric peak with respect to the larger maximum than the SM kernel. The error statistics are significantly higher for the SM model, and surprisingly the resulting estimates for coefficients λ_1 and λ_2 are completely inaccurate. Repeated trials of training the SM model yielded no improvement in coefficient estimates, with different local minima found and dynamics extrapolated to varying degrees of success.

This result, along with the relative difficulty of training the SM model as compared to the RBF model, indicates that the likelihood surface for the SM model is extremely rough. At many parameter values the model also produces singular covariance matrices, further complicating training. However, the capability of the SM model to still extrapolate qualitatively reasonable results is astonishing, hinting that the model itself is under-constrained (with $Q = 4$, the total number of hyperparameters for the SM model is 15). However, the RBF model produces high-quality results, and despite some energy leakage from the higher peak seems to be a promising step towards data-model integration of PDE systems. Comparison of the error patterns of the two systems in figure 9 demonstrates this point further - the SM model performs worse across the board. Interestingly, the noisy models seem to have slightly lower error overall.

In figure 10, we see a more extreme case of the same story for the Kuramoto-Sivashinsky (KS) system. The RBF model recovers the correct coefficients, although accuracy suffers in the case of noisy data. More significantly however the model reproduces the onset of the chaotic regime seen in the simulation with surprisingly high accuracy. However, the SM model fails completely, and produces a high-quality fit of the u_n timestep but is unable to extrapolate the system's dynamics. Note that the higher peak and average errors in the RBF model are here due to the fact that by linearizing the governing PDE, we have removed the ability of the model to capture the intricate nonlinear interactions in the chaotic regime. Still, the ability to accurately predict the onset of that chaos speaks to the RBF model's utility as a flexible predictor.



RBF MODEL

DATA	CLEAN	NOISY
LOG(NOISE)	-15.57	-10.40
λ_1	.97	.79
λ_2	.99	.83
λ_3	.98	.83
PEAK ERROR	5.03	5.35
AVERAGE ERROR	.81	.83

SM MODEL

DATA	CLEAN	NOISY
LOG(NOISE)	-7.78	-6.71
λ_1	-.03	-.04
λ_2	.06	-.05
λ_3	.05	-.06
PEAK ERROR	3.54	106.44
AVERAGE ERROR	.90	18.42

Figure 10. Results of an example KS simulation for RBF kernel and SM kernel with clean data (left), and statistics on error and discovered coefficients (right). Here "Peak Error" refers to the largest error magnitude across the simulation, and "Average Error" refers to the average relative error in 2-norm over all simulated timesteps.

In the case of the KS system the complex likelihood surface of the SM model proves too difficult to optimize. Indeed despite training with hundreds of restarts, no trial converged to a usable model across a wide range of parameter settings and adjustments. We conclude as a result that while the spectral mixture kernel has the ability to be a robust extrapolator, especially for periodic behavior, its parameter complexity and sensitivity to initialization make it unsuitable for this model discovery and PDE system integration task. In figure 9 we see that the noisy data case completely diverges, producing meaningless results.

With these results we have demonstrated that while the GP extrapolation technique described in section 3.3 may not produce physically precise descriptions of the evolution of a PDE system, qualitative dynamics and key features were well reproduced in several contexts (one additional context can be found in appendix D). With flexibility in choice of domain sample points from timestep to timestep as well as the potential for efficient implementation using inducing point methods or structured kernel interpolation, GP integration becomes a potentially attractive means of approximating key features of a system with unknown coefficients.

4. Conclusions and Future Work

We have developed and tested two methods for approximating PDE system solutions with Gaussian processes. Domain interpolation for linear PDE with an OU kernel or some power thereof reported promising results, and significant computational speedup over finite element approaches. Steps were taken towards complex domains, demonstrating that this method is robust to reasonably parameterized domains and has the potential to be extended to many analogous situations. Additionally, the use of multiple models to approximate solutions on a rough grid within an arbitrary quadrilateral domain is a step towards full solution prediction with GP models. Future work in this area includes models for extrapolation of quantities beyond the range of existing data, and extension to numerous other linear PDE and domain types.

Time extrapolation of nonlinear PDE systems reported solid results with an RBF kernel model, and mixed results with a spectral mixture kernel model. The flexibility of domain discretization between timesteps and the possibility for fast inference makes this portion of the project a promising direction for solution approximation, but the current model still has certain obvious issues with respect to energy conservation and precise description of nonlinear interactions. Future work towards alleviating these issues includes adoption of higher-order or symplectic integration schemes, as well as efficiency improvements via structured kernel interpolation.

References

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *NeurIPS*, 2018.

Hughes, T. J. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 1987.

Kopriva, D. *Implementing Spectral Methods for Partial Differential Equations*. Springer, Dordrecht, 2009.

LeVeque, R. J. *Finite Difference Methods for Ordinary and Partial Differential Equations*. SIAM, Philadelphia, PA, 2007.

Raissi, M. and Karniadakis, G. E. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357, 2018.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Numerical gaussian processes for time-dependent and non-linear partial differential equations. *SIAM J. Sci. Computation*, 40(1), 2018.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

Ruppert, J. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3), 1995.

Umetani, N. and Bickel, B. Learning three-dimensional flow for interactive aerodynamic design. *ACM Trans. Graph.*, 37(4), 2018.

Wilson, A. and Adams, R. Gaussian process kernels for pattern discovery and extrapolation. *International Conference on Machine Learning*, 2013.

Wilson, A. and Nickish, H. Kernel interpolation for scalable structured gaussian processes (kiss-gp). *International Conference on Machine Learning*, 2015.

Appendix: Gaussian Processes for Efficient Numerical Simulations in Physics

A Neural Network from Section 2.3.1

For comparison with the GP model, a 5-layer simple neural network was used. This neural network had 100 hidden units per layer, and was fully connected (100 activations between layers).

B Details of Time Extrapolation Model

In this section we describe the time extrapolation model for the Kortweg-de Vries system and for the Kuramoto-Sivashinsky system in further detail. Recall the nonlinear operators for the two systems, given by equations REF and REF in the main text:

$$\text{Kortweg-de Vries: } \frac{\partial u}{\partial t} = -\lambda_1 u \frac{\partial u}{\partial x} - \lambda_2 \frac{\partial^3 u}{\partial x^3},$$

$$\text{Kuramoto-Sivashinksy: } \frac{\partial u}{\partial t} = -\lambda_1 u \frac{\partial u}{\partial x} - \lambda_2 \frac{\partial^2 u}{\partial x^2} - \lambda_3 \frac{\partial^4 u}{\partial x^4}$$

We begin with the Kortweg-de Vries (KdV) equation. The nonlinear operator (where we use subscripts to denote derivatives)

$$\mathcal{N}_x u = -\lambda_1 u u_x - \lambda_2 u_{xx} - \lambda_3 u_{xxx}$$

must be linear when applied to u_n. So, we must approximate the coefficient u in the first term by something which does not depend on u_n. Since we assume that the time between measurements Δt is small, we approximate $u_n \approx u_{n-1}$. That is, we approximate $\mathcal{N}_x$ by the linear operator

$$\mathcal{L}_x u = -\lambda_1 \bar{u} u_x - \lambda_2 u_{xx} - \lambda_3 u_{xxx}$$

where $\bar{u}(x,t) = u(x, t - \Delta t)$. Then, after placing the joint GP prior specified in section 3.3 of the main text, we interpret the joint kernel as

$$\begin{bmatrix} u_n \\ u_{n-1} \end{bmatrix} = \mathcal{GP}\left(0, \begin{bmatrix} k_{1,1} & k_{1,0} \\ k_{0,1} & k_{0,0} \end{bmatrix}\right)$$

where we have defined, with $\mathcal{L}_x$ referring to derivatives with respect to x, and $\mathcal{L}_{x'}$ referring to derivatives with respect to x', the first and second arguments of k respectively,

$$\begin{aligned} k_{1,0}(x, x') &= (I - \Delta t \mathcal{L}_x)k(x, x') \\ k_{0,1}(x, x') &= (I - \Delta t \mathcal{L}_{x'})k(x, x') \\ k_{0,0}(x, x') &= (I - \Delta t \mathcal{L}_{x'})(I - \Delta t \mathcal{L}_x)k(x, x') \end{aligned}$$

Here we note that since k is assumed to be a stationary kernel, we have $k(x, x') = k(x - x') = k(\tau)$, and so

$$\frac{\partial^n}{\partial x^n} k(x, x') = (-1)^n \frac{\partial^n}{\partial x'^n} k(x, x') = \frac{\partial^n}{\partial \tau^n} k(\tau)$$

Notating $\frac{\partial^n}{\partial \tau^n} k(\tau)$ by $k^{(n)}$, we then can compute, for the KdV model,

$$k_{1,1} = k$$

$$\begin{aligned} k_{1,2} &= (I - \Delta t \mathcal{L}_{x'})k \\ &= k - \Delta t \lambda_1 \bar{u}' k^{(1)} + \Delta t \lambda_2 k^{(3)} \end{aligned}$$

$$\begin{aligned} k_{2,2} &= (I - \Delta t \mathcal{L}_x)(I - \Delta t \mathcal{L}_{x'})k \\ &= k + \Delta t \lambda_1 (\bar{u} - \bar{u}')k^{(1)} - \Delta t^2 \lambda_1^2 \bar{u}' \bar{u} k^{(2)} - \Delta t^2 \lambda_1 \lambda_2 (\bar{u} + \bar{u}')k^{(4)} - \Delta t^2 \lambda_2^2 k^{(6)} \end{aligned}$$

(recall $\bar{u}(x, t) = u(x, t - \Delta t)$ and $\bar{u}'(x, t) = u(x', t - \Delta t)$). From this we may compute derivatives with respect to parameters λ_i, which are used for optimization:

$$\begin{aligned} \frac{\partial k_{1,1}}{\partial \lambda_1} &= \frac{\partial k_{1,1}}{\partial \lambda_2} = 0 \\ \frac{\partial k_{1,0}}{\partial \lambda_1} &= -\Delta t \bar{u}' k^{(1)} \\ \frac{\partial k_{1,0}}{\partial \lambda_2} &= \Delta t k^{(3)} \\ \frac{\partial k_{0,0}}{\partial \lambda_1} &= \Delta t (\bar{u} - \bar{u}')k^{(1)} - 2\Delta t^2 \lambda_1 \bar{u}' \bar{u} k^{(2)} - \Delta t^2 \lambda_2 (\bar{u} + \bar{u}')k^{(4)} \\ \frac{\partial k_{0,0}}{\partial \lambda_2} &= -\Delta t^2 \lambda_1 (\bar{u} + \bar{u}')k^{(4)} - 2\Delta t^2 \lambda_2 k^{(6)} \end{aligned}$$

Doing the same for the KS system, which has precisely the same type of nonlinearity, we obtain

$$k_{1,1} = k$$

$$\begin{aligned} k_{1,0} &= (I - \Delta t \mathcal{L}_{x'})k \\ &= k - \Delta t \lambda_1 \bar{u}' k^{(1)} + \Delta t \lambda_2 k^{(2)} + \Delta t \lambda_3 k^{(4)} \end{aligned}$$

$$\begin{aligned} k_{0,0} &= (I - \Delta t \mathcal{L}_x)(I - \Delta t \mathcal{L}_{x'})k \\ &= k + \Delta t \lambda_1 (\bar{u} - \bar{u}')k^{(1)} + (2\Delta t \lambda_2 - (\Delta t \lambda_1)^2 \bar{u}\bar{u}')k^{(2)} + (\Delta t^2 \lambda_1 \lambda_2 (\bar{u} - \bar{u}'))k^{(3)} \\ &\quad + (2\Delta t \lambda_3 + \Delta t^2 \lambda_2^2)k^{(4)} + (\Delta t^2 \lambda_1 \lambda_3 (\bar{u} - \bar{u}'))k^{(5)} + 2\Delta t^2 \lambda_2 \lambda_3 k^{(6)} + \Delta t^2 \lambda_3^2 k^{(8)} \end{aligned}$$

and for derivatives,

$$\begin{aligned} \frac{\partial k_{1,1}}{\partial \lambda_1} &= \frac{\partial k_{1,1}}{\partial \lambda_2} = \frac{\partial k_{1,1}}{\partial \lambda_3} = 0 \\ \frac{\partial k_{1,0}}{\partial \lambda_1} &= -\Delta t \bar{u}' k^{(1)} \\ \frac{\partial k_{1,0}}{\partial \lambda_2} &= \Delta t k^{(2)} \\ \frac{\partial k_{1,0}}{\partial \lambda_3} &= \Delta t k^{(4)} \\ \frac{\partial k_{0,0}}{\partial \lambda_1} &= \Delta t (\bar{u} - \bar{u}')k^{(1)} - 2\Delta t^2 \lambda_1 \bar{u}\bar{u}' k^{(2)} + \Delta t^2 \lambda_2 (\bar{u} - \bar{u}')k^{(3)} \\ &\quad + \Delta t^2 \lambda_3 (\bar{u} - \bar{u}')k^{(5)} \\ \frac{\partial k_{0,0}}{\partial \lambda_2} &= 2\Delta t k^{(2)} + \Delta t^2 \lambda_1 (\bar{u} - \bar{u}')k^{(3)} + 2\Delta t^2 \lambda_2 k^{(4)} + 2\Delta t^2 \lambda_3 k^{(6)} \\ \frac{\partial k_{0,0}}{\partial \lambda_3} &= 2\Delta t k^{(4)} + \Delta t^2 \lambda_1 (\bar{u} - \bar{u}')k^{(5)} + 2\Delta t^2 \lambda_2 k^{(6)} + 2\Delta t^2 \lambda_3 k^{(8)} \end{aligned}$$

C Spectral Mixture Kernel Derivatives

In one dimension, the spectral mixture (SM) kernel is given by equation 14 in the main text,

$$k(\tau) = \sum_{q=1}^{Q} w_q k_q(\tau) = \sum_{q=1}^{Q} w_q \exp(-2\pi^2 \tau^2 \nu_q) \cos(2\pi \tau \mu_q)$$

where Q is the number of mixture components. In order to compute the covariances in appendix A, we require analytic derivatives of $k(\tau)$. From now on we consider only derivatives of a single mixture component,

$$k_q(\tau) = \exp(-2\pi^2\tau^2\nu_q)\cos(2\pi\tau\mu_q)$$

and note that when computing a full derivative one sums over all such mixture components. Making the substitutions $a = -2\pi^2\nu_q$ and $b = 2\pi\mu_q$, we can easily see that arbitrary derivatives take the form

$$k_q^{(n)}(\tau) = e^{a\tau^2}\big(\cos(b\tau)P_c^n(\tau) + \sin(b\tau)P_s^n(\tau)\big)$$

where $P_s^n(\tau)$ and $P_c^n(\tau)$ are polynomials functions of τ which also involve a and b. From this form we can define these polynomials recursively:

$$P_c^{n+1} = 2a\tau P_c^n + \frac{d}{d\tau}P_c^n + bP_s^n, \quad P_s^{n+1} = 2a\tau P_s^n + \frac{d}{d\tau}P_s^n - bP_c^n$$

We compute the first eight such polynomials in order to analytically represent all eight derivatives of k for the KS system:

$$P_c^0(\tau) = 1$$
$$P_s^0(\tau) = 0$$

$$P_c^1(\tau) = 2a\tau$$
$$P_s^1(\tau) = -b$$

$$P_c^2(\tau) = 4a^2\tau^2 + (2a - b^2)$$
$$P_s^2(\tau) = -4ab\tau$$

$$P_c^3(\tau) = 8a^3\tau^3 + (12a^2 - 6ab^2)\tau$$
$$P_s^3(\tau) = -12a^2b\tau^2 + (-6ab + b^3)$$

$$P_c^4(\tau) = 16a^4\tau^4 + (48a^3 - 24a^2b^2)\tau^2 + (12a^2 - 12ab^2 + b^4)$$
$$P_s^4(\tau) = -32a^3b\tau^3 + (-48a^2b + 8ab^3)\tau$$

$$P_c^5(\tau) = 32a^5\tau^5 + (160a^4 - 80a^3b^2)\tau^3 + (120a^3 - 120a^2b^2 + 10ab^4)\tau$$
$$P_s^5(\tau) = -80a^4b\tau^4 + (-240a^3b + 40a^2b^3)\tau^2 + (-60a^2b + 20ab^3 - b^5)$$

$$P_c^6(\tau) = 64a^6\tau^6 + (480a^5 - 240a^4b^2)\tau^4 + (720a^4 - 720a^3b^2 + 60a^2b^4)\tau^2 + \cdots$$
$$+ (120a^3 - 180a^2b^2 + 30ab^4 - b^6)$$
$$P_s^6(\tau) = -192a^5b\tau^5 + (-960a^4b + 160a^3b^3)\tau^3 + (-960a^3b + 240a^2b^3 - 12ab^5)\tau$$

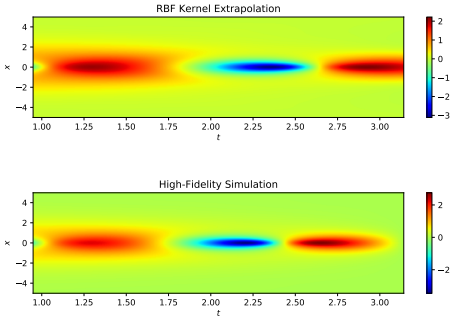
$$P_c^7(\tau) = 128a^7\tau^7 + (1344a^6 - 672a^5b^2)\tau^5 + (3360a^5 - 3360a^4b^2 + 280a^3b^4)\tau^3 + \cdots$$
$$+ (1680a^4 - 2760a^3b^2 + 420a^2b^4 - 14ab^6)\tau$$
$$P_s^7(\tau) = -488a^6b\tau^6 + (-3360a^5b + 560a^4b^3)\tau^4 + (-5520a^4b + 1680a^3b^3 - 84a^2b^5)\tau^2 + \cdots$$
$$+ (-1080a^3b + 420a^2b^3 - 42ab^5 + b^7)$$

$$P_c^8(\tau) = 256a^8\tau^8 + (3584a^7 - 1832a^6b^2)\tau^6 + (13440a^6 - 13440a^5b^2 + 1120a^4b^4)\tau^4 + \cdots$$
$$+ (13440a^5 - 21120a^4b^2 + 3360a^3b^4 - 112a^2b^6)\tau^2 + \cdots$$
$$+ (1680a^4 - 3840a^3b^2 + 840a^2b^4 - 56ab^6 + b^8)$$
$$P_s^8(\tau) = -1104a^7b\tau^7 + (-10992a^6b + 1792a^5b^3)\tau^5 + (-27840a^5b + 8960a^4b^3 - 448a^3b^5)\tau^3 + \cdots$$
$$+ (-14880a^4b + 6960a^3b^3 - 672a^2b^5 + 16ab^7)\tau$$

D Additional Experiment for the RBF Time Extrapolation Model

In addition to the KS and KdV systems, we examined the Nonlinear Schrödinger (NLS) equation with the RBF model. However due to time constraints and a qualitatively different nonlinearity we were not able to implement the SM model for this system. Results are given in figure 1 below. All results are measured on the real part of the solution for space considerations. Note that the governing equation for the NLS system is

$$i\frac{\partial u}{\partial t} = -\lambda_1\frac{\partial^2 u}{\partial x^2} - \lambda_2|u|^2u,\ u \in \mathbb{C},\ \lambda_1 = \frac{1}{2},\ \lambda_2 = 1$$



	RBF Model	
Data	Clean	Noisy
log(noise)	-17.11	-10.41
λ_1	0.51	0.48
λ_2	1.00	0.97
Peak Error	3.86	2.67
Average Error	1.17	1.00

Figure 1: Results of an example NLS simulation for RBF kernel with clean data (left), and statistics on error and discovered coefficients (right). Here "Peak Error" refers to the largest error magnitude across the simulation, and "Average Error" refers to the average relative error in 2-norm over all simulated timesteps.

As can be seen in figure 1, the NLS system's dynamics were well recaptured by the GP model, although due to sensitivity to hyperparameters the periodicity

of the system was not perfectly captured, leading to error estimates which appear large but in fact are simply due to a deviation in synchronization between the simulation and extrapolated system. Once again the RBF model accurately recovers the desired coefficients, and once again qualitatively reproduces key dynamics. In figure 2, we see that as described in the figure 1 table the value of the total error in simulation is in fact quite high due to the desynchronized behavior of the extrapolation as compared to the simulation. However the relative error statistics are inflated by the fact that the NLS system occasionally has zero signal, hence inflating relative error as compared to absolute error.

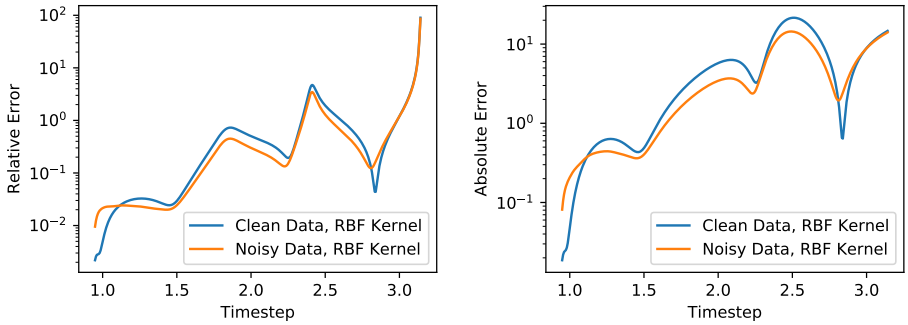


Figure 2: Relative error (left) and absolute error (right) vs timestep for NLS system.