Proceedings of the ASME 2020
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE2020
August 17-19, 2020, Virtual, Online

DETC2020-22381

Extrapolation with Gaussian Random Processes and Evolutionary Programming

Robert Planas, Nicholas Oune, Ramin Bostanabad[1]
Department of Mechanical and Aerospace Engineering, University of California, Irvine
Irvine, CA, USA

ABSTRACT

Emulation plays an indispensable role in engineering design. However, the majority of emulation methods are formulated for interpolation purposes and their performance significantly deteriorates in extrapolation. In this paper, we develop a method for extrapolation by integrating Gaussian processes (GPs) and evolutionary programming (EP). Our underlying assumption is that there is a set of free-form parametric bases that can model the data source reasonably well. Consequently, if we can find these bases via some training data over a region, we can do predictions outside of that region. To systematically and efficiently find these bases, we start by learning a GP without any parametric mean function. Then, a rich dataset is generated by this GP and subsequently used in EP to find some parametric bases. Afterwards, we retrain the GP while using the bases found by EP. This retraining essentially allows to validate and/or correct the discovered bases via maximum likelihood estimation. By iterating between GP and EP we robustly and efficiently find the underlying bases that can be used for extrapolation. We validate our approach with a host of analytical problems in the absence or presence of noise. We also study an engineering example on finding the constitutive law of a composite microstructure.

Keywords: Gaussian Processes, Evolutionary Programming, Extrapolation, Emulation, Maximum Likelihood Estimation.

1 INTRODUCTION

Emulation (aka metamodeling, surrogate modeling, or supervised learning) plays an instrumental role in engineering design. Over the past few decades, many emulation methods have been developed including deep neural networks (NNs) [1], Gaussian processes (GPs) [2-4], radial basis functions [5], boosted trees [6], random forests [7], and many more. Each of these methods has its own pros and cons. However, they all share one limitation: Their performance deteriorates in extrapolation. As schematically illustrated in **Fig. 1**, our goal is to address this limitation by systematically integrating GPs and evolutionary programming (EP).

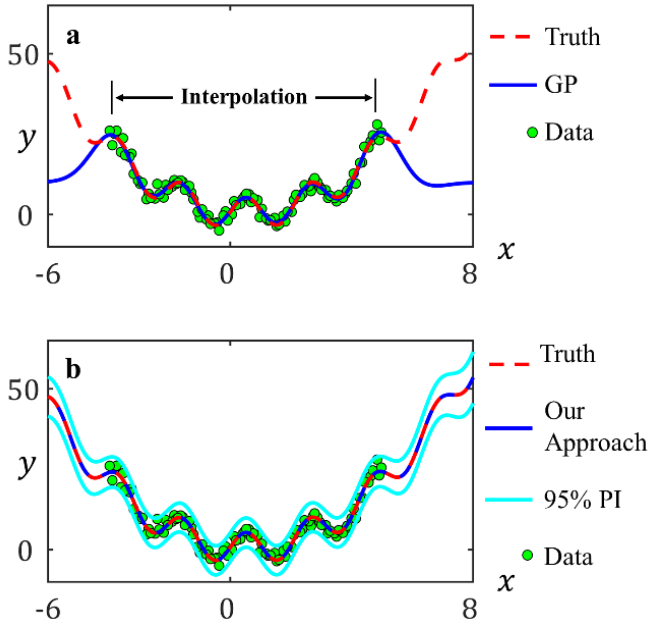


Fig. 1 Extrapolation: (a) GPs are accurate in interpolation but reverse to their mean in extrapolation. **(b)** With our appraoch GPs can extrapolate the data. The figure corresponds to example 1 in Sec. 4.1.

GPs are one of the most common emulators and have been long used to replace expensive computer simulations or real experiments with inexpensive but accurate emulators. GPs have many attractive features that make them exceptionally desirable for computationally intensive design tasks such as robust- and reliability-based design [8], uncertainty quantification [3, 9], and

[1] Corresponding author. Email: Raminb@uci.edu

topology optimization [4]. GPs attractive features primarily stem from their tractable conditional distributions, flexibility in emulating various functional forms (e.g., smooth, fluctuating, rough, …), ability to interpolate or regress, effective learning from small dataset, relatively low computational costs (esp. on small data), and ease of use.

However, similar to many other emulation methods the predictive power of GPs significantly deteriorates in extrapolation. This behavior is known as *reversion to the mean* and is a by-product of the additive nature of a GP predictor. As we explain in Sec. 2, a GP predictor has two parts where the first part consists of a set of parametric mean functions while the second part relies on correlations between the query point and the training data. In extrapolation, these correlations become extremely weak, so the GP predictor relies primarily on the mean (hence the term reversion to the mean).

Reversion to the mean in GPs has been studied previously and prior works can be divided into two primary categories. In the first approach, new covariance functions are designed that decay slower as the distance between a query point and the training data increases [10-12]. These approaches break down if the distance between the query points and the training data is large.

In the second category, a set of parametric functions such as polynomials and trigonometric functions are employed in GP training. The significance of each function is then determined by their coefficients which are generally estimated via maximum likelihood estimation (MLE). A large (small) coefficient indicates the importance (irrelevance) of the corresponding basis function in explaining the relations between the inputs and outputs of the training data. If many basis functions are used in GP training, regularization must be exercised to avoid overfitting. The primary limitation of addressing reversion to the mean with this approach is that the basis functions are designed by humans who generally include simple and application-dependent functions. We believe that our contributions address this shortcoming by automating the process of finding parametric mean functions that can have high degrees of nonlinearity and/or dimensionality.

As depicted in **Fig. 2**, our approach relies on GP emulation and EP. Given a training dataset, we start by learning a GP without any parametric mean function. Then, a rich dataset is generated with this GP and subsequently used in EP to find some basis functions. Afterwards, we retrain the GP while using the bases found by EP. This process is continued until convergence. The original training dataset is not directly used in EP for three primary reasons. Firstly, its size might be too small to be directly used in EP. Secondly, EP generally overfits if the dataset is noisy. Thirdly, EP may require homogeneously or heterogeneously distributed data (e.g., dense in some regions while sparse in other regions) to find highly complex bases. In Sec.3, we elaborate on how the iterative process enables (i) validating the bases found by EP and (ii) robustly detecting convergence.

Our approach has connections with sparse regression. However, there is a key difference between the two methods: in sparse regression the basis functions are chosen a priori by the designer and the sparsity is enforced by $L1$ regularization. In contrast, we automatically find the bases which considerably increases the flexibility and applicability of our approach as compared to sparse regression.

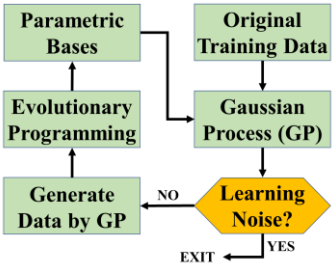


Fig. 2 Flowchart of our approach for extrapolation: We begin by learning a GP without any parametric bases. A large dataset is then generated with this GP and used in EP to find some bases. Afterwards, we retrain the GP while using the bases found by EP. This process is continued until GP starts to learn noise.

The above discussions revolve around GPs, but other emulation methods struggle with extrapolation as well. For instance, if a random forest or an NN is trained to emulate the simple function $y(x) = 2\sin(3x)$ over $x \in [-2\pi, 2\pi]$, neither of them can produce accurate predictions at $x = 3\pi$.

We close this section by a short discussion on a relevant concept known as *generalization*. A typical supervised learning procedure in machine learning or statistics involves estimating some model parameters while minimizing the generalization error of the model. This error is calculated by evaluating the model's performance on some test data assumed to possess the same distributional characteristics as the training data [1, 7]. This procedure gives rise to the so-called i.i.d. assumptions which imply that the training and test data assume similar supports. In this paper, our goal is to make accurate predictions outside of the training support without using any test data. The idea and its underlying assumptions are further discussed in Sec. 3

The rest of the paper is organized as follows. In Sec. 2 we provide some background on GPs and EP. In Sec. 3, we introduce our approach and elaborate on its advantages, underlying assumptions, and convergence mechanism. We test the capabilities of our approach on some analytical examples (with and without noise) in Sec. 4. An engineering application on materials modeling is studied in Sec. 4.2. We summarize our contributions and provide future research directions in Sec. 5.

2 TECHNICAL BACKGROUND

2.1 EMULATION WITH GAUSSIAN PROCESSES

In this section, we describe how GP emulators are fitted to a training dataset. The data can be noisy and is obtained from either computer simulations or laboratory experiments. Note that in practice the data are scaled or normalized to ensure numerical stability.

Denote the output and inputs in the training data by, respectively, y and $\boldsymbol{x} = [x_1, x_2, \dots, x_d]^T$ where $\boldsymbol{x} \in \mathbb{R}^d$. Assume the input-output relation is a single realization from the random process $\eta(\boldsymbol{x})$:

$$\eta(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x})\boldsymbol{\beta} + \xi(\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \dots, f_h(\boldsymbol{x})]$ are some pre-determined set of bases (e.g., $\sin(x_1), \log(x_1 + x_2), x_1^2 x_2, \dots$), $\boldsymbol{\beta} = [\beta_1, \dots, \beta_h]^T$ are unknown coefficients, and $\xi(\boldsymbol{x})$ is a zero-mean GP. $\xi(\boldsymbol{x})$ is completely characterized with its parametric covariance function, $c(\cdot,\cdot)$, given by:

$$cov\big(\xi(\boldsymbol{x}), \xi(\boldsymbol{x}')\big) = c(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 r(\boldsymbol{x}, \boldsymbol{x}'), \tag{2}$$

where σ^2 is the process variance and $r(\cdot)$ is the correlation function. Many parametric correlation functions have been developed for GPs [3, 4, 12-16] with the Gaussian correlation function being the most commonly used one:

$$r(\boldsymbol{x}, \boldsymbol{x}') = \exp\Big\{-\sum_{i=1}^{d} 10^{\omega_i}(x_i - x_i')^2\Big\}, \tag{3}$$

where $\boldsymbol{\omega} = [\omega_1, \dots, \omega_d]^T$, $-\infty < \omega_i < \infty$ are the roughness or scale parameters. In practice, the ranges are changed to $-10 < \omega_i < 6$ for numerical stability. The collection of σ^2 and $\boldsymbol{\omega}$ are called the hyperparameters of $\xi(\boldsymbol{x})$.

Following the assumption in Eq. (1) and given the n training pairs of $\big(\boldsymbol{x}_{(i)}, y_{(i)}\big)$, GP emulation requires finding a point estimate for $\boldsymbol{\beta}$, $\boldsymbol{\omega}$, and σ^2 via either MLE or cross-validation (CV). Alternatively, Bayes' rule can be employed to find the posterior distributions if there is some prior knowledge on these parameters. In this paper, the MLE approach is employed as it provides a high predictive power while minimizing the computational costs [12, 13, 17-20].

The MLE estimates of $\boldsymbol{\beta}$, $\boldsymbol{\omega}$, and σ^2 maximize the likelihood that the n training data are generated by $\eta(\boldsymbol{x})$, that is:

$$\big[\widehat{\boldsymbol{\beta}}, \hat{\sigma}^2, \widehat{\boldsymbol{\omega}}\big] = \underset{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\omega}}{\operatorname{argmax}} |2\pi\sigma^2\boldsymbol{R}|^{\frac{-1}{2}} \times \exp\left(\frac{-(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})^T(\sigma^2\boldsymbol{R})^{-1}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})}{2}\right),$$

which can be written as:

$$\big[\widehat{\boldsymbol{\beta}}, \hat{\sigma}^2, \widehat{\boldsymbol{\omega}}\big] = \underset{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\omega}}{\operatorname{argmin}} \frac{n}{2} log(\sigma^2) + \frac{1}{2}\log(|\boldsymbol{R}|) + \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})^T\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta}), \tag{4}$$

where $\log(\cdot)$ is the natural logarithm, $|\cdot|$ indicates the determinant operator, $\boldsymbol{y} = \big[y_{(1)}, \dots, y_{(n)}\big]^T$ is the $n \times 1$ vector of outputs in the training data, $\boldsymbol{R}$ is the $n \times n$ correlation matrix with $(i,j)^{th}$ element $\boldsymbol{R}_{ij} = r\big(\boldsymbol{x}_{(i)}, \boldsymbol{x}_{(j)}\big)$ for $i, j = 1, \dots, n$, and $\boldsymbol{F}$ is the $n \times h$ matrix with $(k,l)^{th}$ element $\boldsymbol{F}_{kl} = f_l(\boldsymbol{x}_{(k)})$ for $k = 1, \dots, n$ and $l = 1, \dots, h$. Setting the partial derivatives with respect to $\boldsymbol{\beta}$ and σ^2 to zero yields:

$$\widehat{\boldsymbol{\beta}} = [\boldsymbol{F}^T\boldsymbol{R}^{-1}\boldsymbol{F}]^{-1}\boldsymbol{F}^T\boldsymbol{R}^{-1}\boldsymbol{y}, \tag{5}$$

$$\hat{\sigma}^2 = \frac{1}{n}\big(\boldsymbol{y} - \boldsymbol{F}\widehat{\boldsymbol{\beta}}\big)^T\boldsymbol{R}^{-1}\big(\boldsymbol{y} - \boldsymbol{F}\widehat{\boldsymbol{\beta}}\big). \tag{6}$$

$\widehat{\boldsymbol{\beta}}$ and $\hat{\sigma}^2$ are functions of $\boldsymbol{\omega}$ (since $\boldsymbol{R}$ is a function of $\boldsymbol{\omega}$). Plugging these estimates in Eq. (4) and eliminating the constants results in:

$$\widehat{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega}}{\operatorname{argmin}}\ nlog(\hat{\sigma}^2) + \log(|\boldsymbol{R}|) = \underset{\boldsymbol{\omega}}{\operatorname{argmin}}\ L. \tag{7}$$

By numerically minimizing L in Eq. (7) one can find $\widehat{\boldsymbol{\omega}}$ and, subsequently, obtain $\widehat{\boldsymbol{\beta}}$ and $\hat{\sigma}^2$ using equations (5) and (6). Many heuristic global optimization methods such as genetic algorithms [21], pattern searches [22, 23], and particle swarm optimization [24] have been previously employed to solve Eq. (7). However, gradient-based optimization techniques are commonly preferred due to their ease of implementation and superior computational efficiency [2, 13, 25]. To guarantee global optimality in this case, the optimization is done numerous times with different initial guesses.

Upon completion of MLE, the following closed-form formula can be used to predict the response at any $\boldsymbol{x}^*$:

$$\mathbb{E}[y^*] = \boldsymbol{f}(\boldsymbol{x}^*)\widehat{\boldsymbol{\beta}} + \boldsymbol{g}^T(\boldsymbol{x}^*)\boldsymbol{V}^{-1}\big(\boldsymbol{y} - \boldsymbol{F}\widehat{\boldsymbol{\beta}}\big), \tag{8}$$

where $\mathbb{E}$ denotes expectation, $\boldsymbol{f}(\boldsymbol{x}^*) = [f_1(\boldsymbol{x}^*), \dots, f_h(\boldsymbol{x}^*)]$, $\boldsymbol{g}(\boldsymbol{x}^*)$ is an $n \times 1$ vector with i^{th} element $c\big(\boldsymbol{x}_{(i)}, \boldsymbol{x}^*\big) = \hat{\sigma}^2 r\big(\boldsymbol{x}_{(i)}, \boldsymbol{x}^*\big)$, and $\boldsymbol{V}$ is the covariance matrix with $(i,j)^{th}$ element $\hat{\sigma}^2 r\big(\boldsymbol{x}_{(i)}, \boldsymbol{x}_{(j)}\big)$. The posterior covariance between the responses at the two inputs $\boldsymbol{x}^*$ and $\boldsymbol{x}'$ reads:

$$cov(y^*, y') = c(\boldsymbol{x}^*, \boldsymbol{x}') - \boldsymbol{g}^T(\boldsymbol{x}^*)\boldsymbol{V}^{-1}\boldsymbol{g}(\boldsymbol{x}') + \boldsymbol{h}(\boldsymbol{x}^*)^T(\boldsymbol{F}^T\boldsymbol{V}^{-1}\boldsymbol{F})^{-1}\boldsymbol{h}(\boldsymbol{x}'), \tag{9}$$

where $\boldsymbol{h}(\boldsymbol{x}^*) = \big(\boldsymbol{f}^T(\boldsymbol{x}^*) - \boldsymbol{F}^T\boldsymbol{V}^{-1}\boldsymbol{g}(\boldsymbol{x}^*)\big)$.

Eq. (8) clearly demonstrates the reversion to the mean property of GPs in extrapolation: As the Euclidean distance between $\boldsymbol{x}^*$ and the training data increases, the elements of $\boldsymbol{g}(\boldsymbol{x}^*)$ approach zero. When $\boldsymbol{g}(\boldsymbol{x}^*) \sim \boldsymbol{0}$, the posterior mean only depends on $\boldsymbol{f}(\boldsymbol{x}^*)\widehat{\boldsymbol{\beta}}$. If an incorrect or incomplete set of bases is used in training, a GP returns inaccurate predictions when extrapolating.

Finally, we note that GPs can address noise and smooth the data (i.e., avoid interpolation) via the so-called nugget or jitter parameter, δ. To this end, $\boldsymbol{R}$ is replaced with $\boldsymbol{R}_\delta = \boldsymbol{R} + \delta\boldsymbol{I}_{n\times n}$. If δ is used, the estimated (stationary) noise variance in the data would be $\delta\hat{\sigma}^2$. We have recently developed an automatic method to robustly detect and estimate noise based on MLE and leave-one-out CV [13].

2.2 EVOLUTIONARY PROGRAMMING

Biological structures that are more successful in grappling with their environment (i.e., they are *fitter*) survive and reproduce at a higher rate. In other words, over time the fitness of a living individual begets its structure through natural selection, genetic crossover, and mutation. Realizing computer models as complex structures, many scholars have applied the

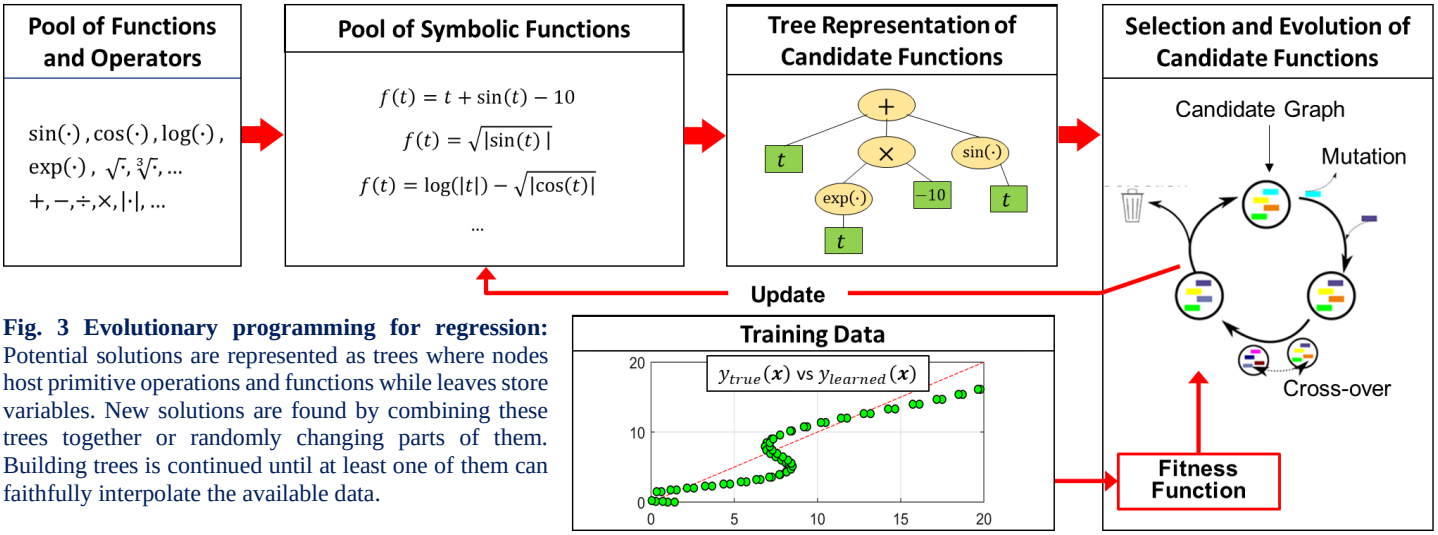


Fig. 3 Evolutionary programming for regression: Potential solutions are represented as trees where nodes host primitive operations and functions while leaves store variables. New solutions are found by combining these trees together or randomly changing parts of them. Building trees is continued until at least one of them can faithfully interpolate the available data.

notion of evolution to programming. This perspective on computer modeling has fathered an active field of research known as EP. The terms genetic programming [26, 27], inverse system identification [28], grammatical evolution [29], symbolic regression [30], and structure learning [31] are very similar lines of research.

Since the early 1990s, EP and its many variants have been exercised to find a symbolic relation between the input(s) and output of a training dataset. As illustrated in **Fig. 3**, the essential idea behind EP is to first build an initial population of simple solutions known as individuals. Then, these solutions are iteratively evolved to build more complex individuals that hopefully better relate the inputs and output based on some predefined criterion. This iterative process is continued until convergence.

Koza provided the first systematic approach for using EP in symbolic regression and robot planning. In his book [26], Koza models potential solutions to these problems as trees where nodes host primitive operations and functions while leaves store variables. New solutions are found by combining these trees together or randomly changing parts of them. Building trees is continued until at least one of them can faithfully interpolate the available data (in regression) or satisfy the governing equation of the system (in robot planning). Since Koza's seminal work, EP and its many variants have been successfully used to, e.g., automatically find the governing ordinary differential equation of a dynamical system such as a mass-spring oscillator or a double pendulum [28, 30, 32]. EP has also been recently employed to find simple partial differential equations that relate the dependent and independent variables in a training dataset [33-37].

We conclude this subsection with three important notes. Firstly, EP has also been used to find the optimum NN architecture [38-41] by adjusting the value of the weights, turning the hidden units on/off, or evolving the number of hidden layers and their units. However, this evolutionary approach for fitting NNs is no longer practiced as it is inferior to the current fitting practices that integrate efficient regularization, backpropagation, and mini-batch stochastic gradient [1]. Secondly, EP is fundamentally different than sparse regression [7, 42-46]. While in EP the input-output relation is discovered, in sparse regression a specific parametric form is assumed, and its parameters are estimated using the data. Thirdly, applications of EP have been primarily limited to problems of low dimensionality/complexity since EP tends to overfit the data by generating extremely complicated expressions. Although regularization alleviates this issue, the proposed penalty functions have been mostly ad hoc (e.g., based on the physics of the problem).

3 OUR APPROACH FOR EXTRAPOLATIVE EMULATION

Our *underlying assumption* is that there is a set of free-form parametric bases that can model the data source reasonably well. Consequently, if we can find these bases via some training data over a region, we can do predictions outside of that region. In this sense, our approach has the same spirit as prior works on symbolic regression and system identification (see Sec. 2.2 for a short review). However, *there are four primary differences*. Firstly, we propose a systematic and general method based on MLE that addresses the overfitting issue of EP. Secondly, the computational efficiency, robustness, and convergence rate of our approach is quite high. Thirdly, we can validate and correct the constants/coefficients found by EP. Lastly, our approach provides prediction intervals and is applicable to both noisy and noiseless data. Having stated the assumptions and main contributions, we next provide the details of our approach.

Our approach is *motivated* by the formulation in Eq. (1) and its direct consequence on the GP predictor in Eq. (8). Generally, no parametric mean function is employed in GP emulation, i.e., $\boldsymbol{f}(\boldsymbol{x})\boldsymbol{\beta} = \sum_{i=1}^{h} \beta_i f_i(\boldsymbol{x}) = \beta$ in Eq. (1). This is a valid assumption for interpolation purposes because the second term on the right-hand side (RHS) of Eq. (8) can emulate a

significantly wide range of relations between $\boldsymbol{x}$ and y. However, two interesting observations are made if the set $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \ldots, f_h(\boldsymbol{x})]$ is chosen such that it contains *some* bases that adequately regress or interpolate the training data. Firstly, the second term on the RHS of Eq. (8) will either be a constant (regardless of $\boldsymbol{x}^*$) or resemble a non-stationary noise source. Secondly, the β's corresponding to the insignificant bases (if there are any) will be estimated as zero. We elaborate on these observations below.

In Eq. (1), assume $\boldsymbol{f}(\boldsymbol{x})$ is chosen such that $\boldsymbol{f}(\boldsymbol{x})\boldsymbol{\beta}$ can regress or interpolate the training data very well. Then:

- These $\boldsymbol{\beta}$ can be estimated systematically and efficiently with MLE, see Eq. (5). In particular, if there are some redundant bases their corresponding $\hat{\beta}$'s will be estimated as zero.
- $\hat{\boldsymbol{\omega}} = [\hat{\omega}_1, \ldots, \hat{\omega}_d]^T$ in Eq. (7) will be all either very small or very large. In particular, if the MLE is performed over $-10 < \omega_i < 6$, the results of Eq. (7) will be $\hat{\omega}_1 = \hat{\omega}_2 = \ldots = \hat{\omega}_d = s$ where $s = -10$ or $s = 6$.

The effect on $\hat{\boldsymbol{\omega}}$ can be explained as follows. If the training data is noisy but the GP is forced to interpolate (i.e., nugget is not used), $\xi(\boldsymbol{x})$ in Eq. (1) will (wrongly) learn noise. Since there are no correlations in noise, $\hat{\omega}_1 = \hat{\omega}_2 = \ldots = \hat{\omega}_d = 6$ to force the spatial correlations of $\xi(\boldsymbol{x})$ to die very fast along all directions in the input space. However, if the nugget is used and systematically estimated via MLE, $\xi(\boldsymbol{x})$ will (correctly) learn the noise mean. Since we have assumed $\boldsymbol{f}(\boldsymbol{x})\hat{\boldsymbol{\beta}}$ regresses the training data very well, the noise mean is zero. Hence, $\xi(\boldsymbol{x})$ will be zero everywhere which indicates that there is a very high correlation between any two points in the input space. Such high correlations can be achieved by $\hat{\omega}_1 = \hat{\omega}_2 = \ldots = \hat{\omega}_d = -10$. **Fig. 4** illustrates these results in $1D$. If the training data is noiseless, $\boldsymbol{f}(\boldsymbol{x})\hat{\boldsymbol{\beta}}$ interpolates the data accurately. In this case, $\xi(\boldsymbol{x})$ will be zero everywhere and $\hat{\omega}_1 = \hat{\omega}_2 = \ldots = \hat{\omega}_d = -10$. Thus, the estimate on $\hat{\boldsymbol{\omega}}$ provides a clear signal on whether $\boldsymbol{f}(\boldsymbol{x})\hat{\boldsymbol{\beta}}$ regresses or interpolates the data accurately; regardless of whether the data is noisy or not.

The above discussions assumed that $\boldsymbol{f}(\boldsymbol{x})$ are given. In high dimensions or complex problems, it is impractical to guess these bases or build a very large set of bases and hope that MLE can filter out the redundant ones. As demonstrated in **Fig. 2**, we find these bases with EP. In particular, given a training dataset, we first fit a GP without any parametric mean, i.e., $\boldsymbol{f}(\boldsymbol{x})\boldsymbol{\beta} = \sum_{i=1}^{h} \beta_i f_i(\boldsymbol{x}) = \beta$. This GP is accurate as long as we are interpolating. So, we use this GP to generate a new training dataset over the interpolation region. This new dataset is then used in EP to find $\boldsymbol{f}(\boldsymbol{x})$, i.e., the set of parametric bases. Next, we refit the GP but this time use the newly found $\boldsymbol{f}(\boldsymbol{x})$. This process is continued until convergence, i.e., until $\hat{\omega}_1 = \hat{\omega}_2 = \ldots = \hat{\omega}_d = s$ where $s = -10$ or $s = 6$.

It is important to note that EP generates a population of solutions, i.e., many sets of $\boldsymbol{f}(\boldsymbol{x})$'s. In our implementation, we only choose the best individual for the next iteration and employ it in GP training. Other approaches may be exercised instead. For example, one could use the union of the top 10 individuals to increase the diversity of bases in $\boldsymbol{f}(\boldsymbol{x})$ and let MLE determine their relevance in the next iteration of our approach.

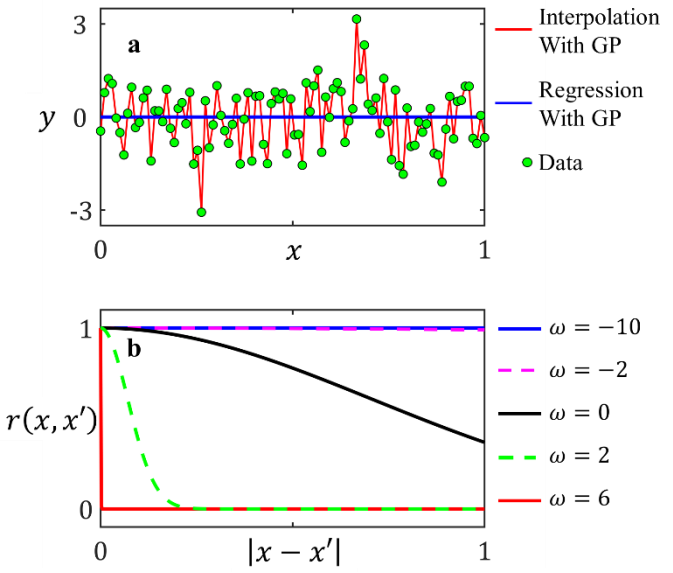


Fig. 4 Learning noise with GPs: (a) A noisy dataset that follows a normal distribution. **(b)** Effeect of ω on $r(x, x')$ in 1D, see also Eq. (3). When interpolating/regressing the noisy data, a very large/small ω is needed. A GP with $\omega = -4$ ($\omega = 4$) can regress (interpolate) the data shown in **(a)**. It is assumed that x is scaled to $[0, 1]$.

Table 1 Analytical examples: Our examples have combinations of polynomials, trigonometric functions, logarithm, and exponentiation.

EX ID	Function	$\boldsymbol{Min}(\boldsymbol{x})$	$\boldsymbol{Max}(\boldsymbol{x})$	$\boldsymbol{Range}(y)$
1	$y(x) = {x_1}^2 - x_1 + 5\sin(3x_1) + 1$	-4	5	27.595
2	$y(\boldsymbol{x}) = {x_1}^2 + 3x_1 - \exp(x_1) + \cos(x_1) + 4$	-4	5	111.333
3	$y(\boldsymbol{x}) = ({x_1}^2 - 10)/({x_2}^2 + x_2 + 2)$	$[-4.5, -5]$	$[6, 7]$	20.324
4	$y(\boldsymbol{x}) = x_1x_2 + {x_2}^2 + 5\sin(3x_1) + 1$	$[-4, -4]$	$[-1, 2]$	42.498
5	$y(\boldsymbol{x}) = {x_1}^2 + \cos(x_2) + \log(\lvert x_3 \rvert) + \exp(x_2) + 4$	$[-1, -2, 0.25]$	$[4, 3, 6]$	36.888
6	$y(\boldsymbol{x}) = x_2x_3 + x_1 - x_2 + 3\sin(2x_1) - 2\cos(x_3) - 5$	$[-2, -6, -2]$	$[4, 2, 3]$	37.164
7	$y(\boldsymbol{x}) = x_1x_2x_3 + {x_2}^2 + x_2 + 5\sin(3x_1) + \sin(x_3) + 1$	$[-1, -2, -1]$	$[3, 2, 5]$	62.265
8	$y(\boldsymbol{x}) = x_1x_2 - x_3x_4 + x_1 - 2\sin(x_1x_3) + 2$	$[-4, -3, 3.5, -5]$	$[2, 4, 6, 4]$	76.568

The original training dataset is not directly used in EP because (i) EP generally overfits if the dataset is noisy. GP allows to automatically detect and remove noise. (ii) The number of training data might be too small to be directly used in EP. This would lead to overfitting in EP as well. (iii) We normally don't have any control over the spatial distribution of the original training data. Using GP to generate data allows to homogeneously or heterogeneously distribute them (e.g., dense in some regions while sparse in other regions). As we show in Sec. 4.2, this helps in finding highly complex bases.

Note that EP tends to overfit even with noiseless and informative data made by GP. Hence, once a set of bases are found by EP, we refit the GP to employ MLE and determine if any of the bases is redundant. We believe this approach is more robust, efficient, and flexible than exercising ad-hoc regularization in EP. For instance, EP produces deeply nested functions such as $\sin(sin(cos(\ldots)))$ in many of our examples in Sec. 4.1. Once these functions are used in GP training, the corresponding β's are consistently estimated as zero by MLE.

4 VALIDATION

In this section, we apply our approach to a set of analytical examples in Sec. 4.1 and an engineering one in Sec. 4.2. In all the examples we use the Python package DEAP [27] for EP and train the GPs following the algorithm in [13]. We initialize DEAP with a set of primitive functions and let the algorithm evolve them to build more complex functions. Our primitive set includes the following functions and operations: $\log(|\cdot|), \exp(\cdot), +, \times, -, \div, \sin(\cdot)$, and $\cos(\cdot)$. In all of our examples, the population size, maximum number of generations, probability of crossover, and mutation probability are set to, respectively, 1200, 350, 0.3, and 0.4. The breaking parameter is set to 50, i.e., the evolution stops if the best individual does not improve after 50 generations. The fitness function in DEAP is set to minimize the root mean squared errors. These errors are calculated over the data that GP generates at each iteration of our approach.

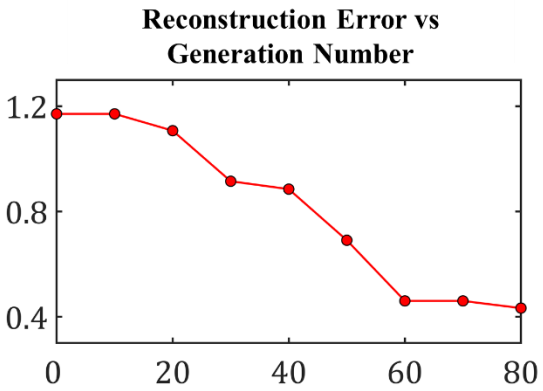
4.1 ANALYTICAL EXAMPLES

As shown in **Table 1**, the analytical functions we study are quite complex: many of them have high frequency or non-trivial terms that render guess-based function discovery for extrapolation very difficult. We study these examples in two scenarios. In the first one, the original training data is noiseless; allowing us to directly compare the performance of our approach to EP while changing the size of the training data. In the second scenario, the training data is noisy, so we only use our approach because EP overfits noisy data. We use Sobol sequence [47, 48] for taking samples from each function. Sample-to-sample variations are not observed (and thus not reported) in our simulations because (i) the sample sizes are sufficiently large, and (ii) Sobol sequence generates high quality space-filling designs.

Table 2 summarizes the results in the first scenario and indicates that, in the absence of noise, both approaches are quite efficient. In Ex 6, EP is unable to find the high frequency term $\sin(2x_1)$ but increasing the size of the training data alleviates this issue. In Ex 7, however, EP cannot learn the $\sin(x_3)$ term even with many data points. Instead of $\sin(x_3)$, EP wrongly finds other terms that are generally highly nonlinear and change with each time the algorithm is run. Two sample extra terms found by EP in Ex 7 are $x_1 + \cos({x_1}^2)$ and $x_1 + (\cos(2x_1) + \cos({x_1}^2))\sin(x_1)$. This error in Ex 7 is primarily because $-1 \leq \sin(x_3) \leq 1$ while $y(\boldsymbol{x})$ changes in a much larger range. To

Table 2 Results on analytical examples: Our approach (GP+EP) and EP can find the true function form in most cases. In Ex 6, EP missed $\sin(2x_1)$ with small training data. In Ex 7, EP could not recover the sinusodal term $\sin(x_3)$ on its own. Our approach never produces redundent bases because of MLE verification. However, EP wrongly produces extra terms if it cannot find the exact function form. Our simulation results are not affected by the randomness of the original training data. This is primarily due to the data size (which is at least $100d$) and the space-filling property of Sobol sequence.

Ex ID (dim)	Size of Original Training Data	Exact Form Recovered?	
		EP	GP+EP
1 (1)	$100 * d$	Yes	Yes
	$200 * d$	Yes	Yes
	$300 * d$	Yes	Yes
2 (1)	$100 * d$	Yes	Yes
	$200 * d$	Yes	Yes
	$300 * d$	Yes	Yes
3 (2)	$100 * d$	Yes	Yes
	$200 * d$	Yes	Yes
	$300 * d$	Yes	Yes
4 (2)	$100 * d$	Yes	Yes
	$200 * d$	Yes	Yes
	$300 * d$	Yes	Yes
5 (3)	$100 * d$	Yes	Yes
	$200 * d$	Yes	Yes
	$300 * d$	Yes	Yes
6 (3)	$100 * d$	No	Yes
	$200 * d$	Yes	Yes
	$300 * d$	Yes	Yes
7 (3)	$100 * d$	No	Yes
	$200 * d$	No	Yes
	$300 * d$	No	Yes
8 (4)	$100 * d$	Yes	Yes
	$200 * d$	Yes	Yes
	$300 * d$	Yes	Yes



Generation	Function
0	$x_1(x_1-1)$
10	$x_1(x_1-1)$
20	$x_1{}^2 - x_1 - \sin(2x_1)/\sin(1) + \sin(\sin(x_1{}^2)) + 1$
30	$x_1{}^2 - x_1 + \sin(\sin(3x_1)) + 1 - \sin(x_1)/x_1$
40	$x_1{}^2 - x_1 + \sin(3x_1) + 0.4 - \sin(4)$
50	$x_1{}^2 - x_1 + 2\sin(3x_1) + 0.4$
60	$x_1{}^2 - x_1 + 2\sin(3x_1) + \sin(\sin(3x_1)) + 1$
70	$x_1{}^2 - x_1 + 2\sin(3x_1) + \sin(\sin(3x_1)) + 1$
80	$x_1{}^2 - x_1 + 3\sin(3x_1) + 1$

Fig. 5 Evolution of a solution in evolutionary programming: In each iteration of our approach, EP is used to find a parametric function that can interpolate or regress the data generated via GP. The obtained solution is then validated via MLE in the next iteration. This figure corresponds to the first iteration in Ex 1 (no noise, 100 original training samples). The final solution does not match the true function form because the coefficient of $\sin(3x_1)$ is not 5. The MLE corrects this issue in the next iteration.

minimize the reconstruction error, EP focuses on the polynomial terms as well as $\sin(3x_1)$ which has a higher amplitude.

As opposed to EP, our approach which integrates GP and EP can consistently find the true underlying function. We iterate between GP and EP per **Fig. 2** and stop once $\hat{\boldsymbol{\omega}}$ converge. In all cases in **Table 2**, we generate 1000 data points with GP for EP and iterate between them at least twice to arrive at $\hat{\omega}_1 = \hat{\omega}_2 = \ldots = \hat{\omega}_d = -10$. Note that, our algorithm always ends with GP where MLE is carried out. At each iteration, MLE validates the bases found by EP in the previous iteration. In some of our examples, EP misidentifies some of the coefficients or includes some extra bases. These inaccuracies are all corrected by MLE in the next iteration. This is shown for Ex 1 in **Fig. 5** where in the final solution the coefficient of $\sin(3x_1)$ is incorrectly estimated as 3 instead of 5. When the bases x_1^2, x_1, and $\sin(3x_1)$ are used in GP emulation in the next iteration, the coefficient of $\sin(3x_1)$ is corrected by MLE. Note that we do not include the constants found by EP in GP as bases because we always let MLE determine if a constant term is required. This is done by always including a constant number such as 1 in the set of bases.

Table 3 Results on analytical examples with noise: EP overfits noisy data if used alone. So, we only use GP+EP which obviates ad-hoc regularization of EP. We never get extra terms since MLE associates a coeficient (i.e., β) of zero to redundent bases. The added noise to the data is zero-mean normal with variance as given in the table. Our simulation results are not affected by the randomness of the original training data. This is primarily due to the data size (which is $400d$) and the space-filling property of Sobol sequence. The iteration numbers do not include the first time that GP is fitted where no parametric function is used.

Ex ID (dim)	Noise Variance	Exact Form Recover?		Iterations (EP+GP)
		EP	EP + GP	
1 (1)	0.25^2	No	Yes	1
	2^2	No	Yes	2
2 (1)	0.2^2	No	Yes	1
	1.5^2	No	Yes	1
3 (2)	0.25^2	No	Yes	1
	2^2	No	Yes	1
4 (2)	0.2^2	No	Yes	1
	2^2	Yes	Yes	1
5 (3)	0.25^2	No	Yes	2
	2^2	No	Yes	3
6 (3)	0.2^2	Yes	Yes	1
	2^2	No	Yes	1
7 (3)	0.1^2	No	Yes	2
	1.5^2	No	Yes	1
8 (4)	0.2^2	No	Yes	2
	2^2	No	Yes	2

Table 3 summarizes the results in the second scenario where the original training data is noisy. EP, on its own, overfits noisy data and rarely obtains the exact form.. We test two noise levels to assess the sensitivity of our approach to noise variance. As the tabulated data indicate, we can successfully recover the true function forms given only noisy data. In this case, GP automatically filters out the noise using the nugget parameter. Hence, the data that it generates for EP at each iteration is noiseless.

4.2 EXTRAPOLATION FOR MATERIALS MODELING

Unlike the examples in Sec. 4.1, in real-world applications the exact relation between the independent and dependent variables is unknown. In this subsection we study how our algorithm performs in these scenarios.

Establishing microstructure-property relations plays a key role in materials design and, thereby, in many industrial and technological sectors [49-51]. Here, we examine the case of learning the constitutive law of a $2D$ hyperelastic composite microstructure. The constitutive law of our microstructure is unknown and thus numerical methods (e.g., finite element analysis, FEA) are required to find the response of the

microstructure to any applied strains. The high computational costs of numerical methods hinder designing multiscale materials whose fine-scale behavior is governed by our microstructure. Hence, the goal is to find the constitutive law and, in turn, be able to predict the microstructure behavior (stored potential in our case) under any applied strains.

We take a data-driven approach to achieve this goal. In particular, a training dataset of strains-potential is built by deforming the microstructure under various strain states via FEA. The dataset has a total of 512 samples and its inputs and output are, respectively, principal strains $(\varepsilon_1, \varepsilon_2)$ and the stored potential (ϕ). As demonstrated in **Fig. 6(a)**, the data are heterogeneously distributed in the input space. The data have a small amount of noise that is primarily due to numerical instabilities, excessive mesh distortion, and round off errors experienced in FEA. The reader is referred to [52] for more technical details on how the data is generated.

We divide this dataset into three mutually exclusive and collectively exhaustive sets for training, interpolation testing, and extrapolation testing with sample sizes of, respectively, $412, 50,$ and 50. Then, we fit an emulator to the training set and use it to predict the stored potential in the other two sets. To evaluate the performance of our approach (method 1), we compare it against three other emulation strategies: a polynomial model (method 2) a GP without any parametric mean (method 3), and a feed forward NN (method 4).

With our method, in each iteration, we generate heterogeneously distributed data with GP for EP. In particular, we first build a space-filling design of size 500 over the $(\varepsilon_1, \varepsilon_2)$ space with Sobol sequence and then augment this set with 100 data points close to $\varepsilon_1 = \varepsilon_2 = 0$ to capture the small strain behavior well. In each stage, we ensure the points are located in the strain space such that GP interpolates. The reason for placing more points close to the origin is that with large strains, ϕ is large so EP finds bases that are primarily predictive of the large strain behavior. To help EP capture the small strain behavior, we generate more data points that correspond to that region, see **Fig. 6(b)**.

We evaluate these four methods using two criteria: (1) The performance on the interpolation and extrapolation test sets, and (2) the physical constraints that the resulting models should satisfy. **Table 4** summarizes the performance of each method based on the first criteria and indicates that our method and NN outperform other methods. These results are consistent with **Fig. 6(c)-(f)** where only our method and NN produce models with physically acceptable trends over the strain ranges shown in **Fig. 6(a)**. In particular, in a hyperelastic material, such as rubber, the potential is zero in the absence of any strains and monotonically increases as the strain magnitude grows [53]. These properties are only preserved consistently with our approach and NN where the derivative of ϕ with respect to either ε_1 or ε_2 is positive over the strain space of **Fig. 6(a)** and the potential at (0,0) is equal to 0.

As opposed to NN which is a black box, our approach provides (1) an analytical expression for the dynamics of the model that can be used for inspection and design, (2) prediction

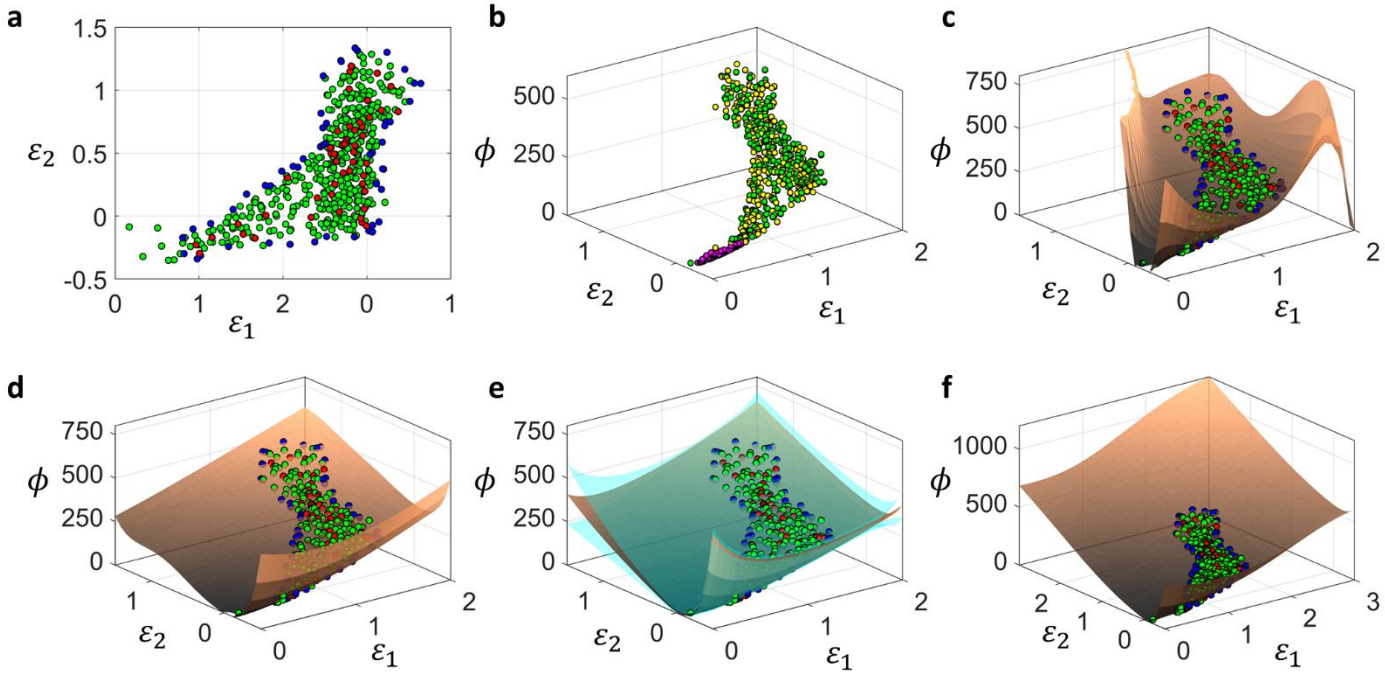


Fig. 6 Extrapolation for materials modeling: (a) The training data. ε_1 and ε_2 are the principal strains and ϕ is the stored potential in the composite microstructure upon deformation. The green, red, and blue data points denote, respectively, the training set, interpolation test set, and extrapolation test set. **(b)** Data generated with GP for EP in one of the iterations of our appraoch. Colors indicate the **(c)** Emulation via cross-validated polynomials, **(d)** Emulation via GP **(e)** Emulation using our approach. The light blue surfaces mark the 95% prediction intervals. **(f)** Emulation with cross validated NN. The NN model is plotted over a larger strain range to indicate its shortcoming in excessive extrapolation. This behavior is not observed in our appraoch.

intervals which are useful in sequential sampling and assessing the reliability of the extrapolation and interpolation, and (3) faithful extrapolations over large ranges (as shown in **Fig. 6(e)** extrapolation with NN is inaccurate for very large strains).

Table 4 Mean squared error (MSE): Our method outperforms other approaches in predicting the test data.

Method	MSE: Training set	MSE: Interpolation test set	MSE: Extrapolation test set
Polynomial	**0.112**	0.370	0.644
GP Only	0.116	0.361	0.417
Neural Network	0.114	0.355	0.386
EP + GP	0.116	**0.352**	**0.376**

5 CONCLUSIONS AND FUTURE WORKS

In this paper, we introduced a method for extrapolative emulation. Our underlying assumption was that there is a set of free-form parametric functions that can emulate the data source reasonably well. The by-product of this assumption is that if we can find these bases via some training data over a region, we can do predictions outside of that region. To systematically and efficiently find these bases, we introduced an approach by integrating GPs with evolutionary programming. We start by learning a GP without any parametric mean function. Then, a rich dataset is generated by this GP and subsequently used in EP to find some parametric bases. Afterwards, we retrain the GP while using the bases found by EP. This retraining essentially allows to validate and/or correct the discovered bases via maximum likelihood estimation. By iterating between GP and EP we robustly and efficiently find the underlying bases that can be used for extrapolation.

We illustrated our approach with eight analytical problems (with or without noise) where it performed better than EP alone. We also studied an engineering example on finding the constitutive law of a composite microstructure. The emulators we found using our approach were more consistent with the physics of the problem than those found by other approaches.

In our approach, we used a GP as an interphase between the original data and EP. Other methods can be used in place of GPs, but care must be exercised because the chosen method must have the capability of detecting overfitting and convergence. In the engineering example, we verified our results by checking the monotonicity constraint. A more rigorous and quantitative validation procedure, perhaps by obtaining more data from FEA, will be useful.

6 REFERENCES

1. Goodfellow, I., Y. Bengio, and A. Courville, *Deep learning*. 2016: MIT press.
2. Hassaninia, I., et al., *Characterization of the Optical Properties of Turbid Media by Supervised Learning of Scattering Patterns.* Sci Rep, 2017. **7**(1): p. 15259.
3. Bostanabad, R., et al., *Uncertainty quantification in multiscale simulation of woven fiber composites.* Computer Methods in Applied Mechanics and Engineering, 2018. **338**: p. 506-532.
4. Bostanabad, R., et al., *Globally Approximate Gaussian Processes for Big Data With Application to Data-Driven Metamaterials Design.* Journal of Mechanical Design, 2019. **141**(11).
5. Rasmussen, C.E., *Gaussian processes for machine learning.* 2006.
6. Chen, T. and C. Guestrin. *Xgboost: A scalable tree boosting system*. in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016. ACM.
7. Hastie, T., et al., *The elements of statistical learning*. Vol. 2. 2009: Springer.
8. Du, X. and W. Chen, *Sequential Optimization and Reliability Assessment Method for Efficient Probabilistic Design.* Journal of Mechanical Design, 2004. **126**(2): p. 225-233.
9. Bostanabad, R., et al., *Multiscale Simulation of Fiber Composites with Spatially-Varying Uncertainties*, in *Uncertainty Quantification in Multiscale Materials Modeling*. 2019, Elsevier.
10. Ba, S. and V.R. Joseph, *Composite Gaussian process models for emulating expensive functions.* The Annals of Applied Statistics, 2012: p. 1838-1860.
11. Zhang, N. and D.W. Apley, *Fractional brownian fields for response surface metamodeling.* Journal of Quality Technology, 2014. **46**(4): p. 285.
12. Plumlee, M. and D.W. Apley, *Lifted Brownian Kriging Models.* Technometrics, 2017. **59**(2): p. 165-177.
13. Bostanabad, R., et al., *Leveraging the nugget parameter for efficient Gaussian process modeling.* International Journal for Numerical Methods in Engineering, 2018. **114**(5): p. 501-516.
14. Zhang, W., et al., *A numerical Bayesian-calibrated characterization method for multiscale prepreg preforming simulations with tension-shear coupling.* Composites Science and Technology, 2019. **170**: p. 15-24.
15. Cressie, N., *The origins of kriging.* Mathematical geology, 1990. **22**(3): p. 239-252.
16. Stein, M.L., *Interpolation of spatial data: some theory for kriging*. 2012: Springer Science & Business Media.
17. Gramacy, R.B. and D.W. Apley, *Local Gaussian process approximation for large computer experiments.* Journal of Computational and Graphical Statistics, 2015. **24**(2): p. 561-578.
18. MacDonald, B., P. Ranjan, and H. Chipman, *GPfit: AnRPackage for Fitting a Gaussian Process Model to Deterministic Simulator Outputs.* Journal of Statistical Software, 2015. **64**(12).
19. Ranjan, P., R. Haynes, and R. Karsten, *A computationally stable approach to Gaussian process interpolation of deterministic computer simulation data.* Technometrics, 2011. **53**(4): p. 366-378.

20. Sacks, J., S.B. Schiller, and W.J. Welch, *Designs for Computer Experiments.* Technometrics, 1989. **31**(1): p. 41-47.
21. Toal, D.J.J., N.W. Bressloff, and A.J. Keane, *Kriging hyperparameter tuning strategies.* Aiaa Journal, 2008. **46**(5): p. 1240-1252.
22. Audet, C. and J.E. Dennis Jr, *Analysis of generalized pattern searches.* SIAM Journal on optimization, 2002. **13**(3): p. 889-903.
23. Zhao, L., K. Choi, and I. Lee, *Metamodeling method using dynamic kriging for design optimization.* AIAA journal, 2011. **49**(9): p. 2034-2046.
24. Toal, D.J., et al., *The development of a hybridized particle swarm for kriging hyperparameter tuning.* Engineering optimization, 2011. **43**(6): p. 675-699.
25. Tao, S., et al. *Enhanced Gaussian Process Metamodeling and Collaborative Optimization for Vehicle Suspension Design Optimization*. in *ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. August 6–9, 2017. Cleveland, OH: American Society of Mechanical Engineers.
26. Koza, J.R., *Genetic programming as a means for programming computers by natural selection.* Statistics and computing, 1994. **4**(2): p. 87-112.
27. Fortin, F.-A., et al., *DEAP: Evolutionary algorithms made easy.* Journal of Machine Learning Research, 2012. **13**(Jul): p. 2171-2175.
28. Bongard, J. and H. Lipson, *Automated reverse engineering of nonlinear dynamical systems.* Proceedings of the National Academy of Sciences, 2007. **104**(24): p. 9943-9948.
29. Tsoulos, I.G. and I.E. Lagaris, *Solving differential equations with genetic programming.* Genetic Programming and Evolvable Machines, 2006. **7**(1): p. 33-54.
30. Schmidt, M. and H. Lipson, *Symbolic regression of implicit equations*, in *Genetic Programming Theory and Practice VII*. 2010, Springer. p. 73-85.
31. Koller, D. and N. Friedman, *Probabilistic graphical models: principles and techniques*. 2009: MIT press.
32. Schmidt, M. and H. Lipson, *Distilling free-form natural laws from experimental data.* science, 2009. **324**(5923): p. 81-85.
33. Izzo, D., F. Biscani, and A. Mereta. *Differentiable genetic programming*. in *European Conference on Genetic Programming*. 2017. Springer.
34. Long, Z., et al., *PDE-net: Learning PDEs from data.* arXiv preprint arXiv:1710.09668, 2017.
35. Atkinson, S., et al., *Data-driven discovery of free-form governing differential equations.* arXiv preprint arXiv:1910.05117, 2019.
36. Bassenne, M. and A. Lozano-Durán, *Computational model discovery with reinforcement learning.* arXiv preprint arXiv:2001.00008, 2019.
37. Jonckheere, M., et al., *Uncovering differential equations from data with hidden variables.* 2019.
38. Stanley, K.O., et al., *Designing neural networks through neuroevolution.* Nature Machine Intelligence, 2019. **1**(1): p. 24-35.
39. Gauci, J. and K.O. Stanley, *Autonomous evolution of topographic regularities in artificial neural networks.* Neural computation, 2010. **22**(7): p. 1860-1898.
40. Schmidhuber, J., *Deep learning in neural networks: An overview.* Neural Networks, 2015. **61**: p. 85-117.
41. Yao, X., *Evolving artificial neural networks.* Proceedings of the IEEE, 1999. **87**(9): p. 1423-1447.
42. Brunton, S.L., J.L. Proctor, and J.N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems.* Proceedings of the National Academy of Sciences, 2016. **113**(15): p. 3932-3937.
43. Rudy, S.H., et al., *Data-driven discovery of partial differential equations.* Science Advances, 2017. **3**(4): p. e1602614.
44. Schaeffer, H., *Learning partial differential equations via data discovery and sparse optimization.* Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2017. **473**(2197): p. 20160446.
45. Xiong, J., S.-Q. Shi, and T.-Y. Zhang, *A machine-learning approach to predicting and understanding the properties of amorphous metallic alloys.* Materials & Design, 2020. **187**: p. 108378.
46. Guyon, I. and A. Elisseeff, *An introduction to variable and feature selection.* Journal of machine learning research, 2003. **3**(Mar): p. 1157-1182.
47. Sobol', I.y.M., *On the distribution of points in a cube and the approximate evaluation of integrals.* Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 1967. **7**(4): p. 784-802.
48. Sobol, I.M., *On quasi-Monte Carlo integrations.* Mathematics and Computers in Simulation, 1998. **47**(2-5): p. 103-112.
49. Bostanabad, R., et al., *Stochastic microstructure characterization and reconstruction via supervised learning.* Acta Materialia, 2016. **103**: p. 89-102.
50. Bostanabad, R., W. Chen, and D.W. Apley, *Characterization and reconstruction of 3D stochastic microstructures via supervised learning.* J Microsc, 2016. **264**(3): p. 282-297.
51. Bostanabad, R., et al., *Computational microstructure characterization and reconstruction: Review of the state-of-the-art techniques.* Progress in Materials Science, 2018. **95**: p. 1-41.
52. Bessa, M.A., et al., *A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality.* Computer Methods in Applied Mechanics and Engineering, 2017. **320**: p. 633-667.
53. Belytschko, T., et al., *Nonlinear finite elements for continua and structures*. 2013: John wiley & sons.