Robert Planas
Department of Mechanical and Aerospace Engineering, University of California, Irvine, CA 92697
e-mail: planasr@uci.edu

Nick Oune
Department of Mechanical and Aerospace Engineering, University of California, Irvine, CA 92697
e-mail: ounen@uci.edu

Ramin Bostanabad
Department of Mechanical and Aerospace Engineering, University of California, Irvine, CA 92697
e-mail: raminb@uci.edu

Evolutionary Gaussian Processes

Emulation plays an important role in engineering design. However, most emulators such as Gaussian processes (GPs) are exclusively developed for interpolation/regression and their performance significantly deteriorates in extrapolation. To address this shortcoming, we introduce evolutionary Gaussian processes (EGPs) that aim to increase the extrapolation capabilities of GPs. An EGP differs from a GP in that its training involves automatic discovery of some free-form symbolic bases that explain the data reasonably well. In our approach, this automatic discovery is achieved via evolutionary programming (EP) which is integrated with GP modeling via maximum likelihood estimation, bootstrap sampling, and singular value decomposition. As we demonstrate via examples that include a host of analytical functions as well as an engineering problem on materials modeling, EGP can improve the performance of ordinary GPs in terms of not only extrapolation, but also interpolation/regression and numerical stability. [DOI: 10.1115/1.4050746]

Keywords: Gaussian processes, evolutionary programming, extrapolation, optimization

1 Introduction

Emulation or surrogate modeling has become an indispensable part of many scientific and engineering disciplines. Over the past few decades, many emulation methods have been developed. Some prominent examples include neural networks [1,2], Gaussian processes (GPs) [3–6], which are closely related to Kriging models [7–10], radial basis functions (RBF) [11], support vector regressors (SVR) [12], boosted trees [13], random forests [14], and evolutionary programming (EP) [15–17]. Among these methods, GPs have played a pivotal role in engineering design because they are easy to train and interpret, can reasonably quantify prediction uncertainty [18,19], have tractable conditional distributions, can emulate various function forms (e.g., smooth, fluctuating, rough, $\cdots$), and can handle qualitative inputs [20]. GPs, unlike deep learning models or any big data-based emulator, are also highly effective in interpolation/regression when the training data are sparse and noisy. However, similar to many other emulation methods, the predictive power of ordinary GPs drastically deteriorates in extrapolation, see Fig. 1.

To address this limitation, we build evolutionary Gaussian processes (EGPs) via a novel framework that systematically integrates GP modeling with EP, singular value decomposition (SVD), maximum likelihood estimation (MLE), and bootstrap sampling. Our results of comparing EGPs with ordinary GPs indicate that (i) EGPs generally outperform GPs in extrapolation, regression, and interpolation (see Fig. 1(*b*) for a simple example), and (ii) EGPs are numerically more stable than GPs.

As detailed in Sec. 2.1, the predictive power of ordinary GPs significantly decreases in extrapolation. This behavior is known as reversion to the mean and is a by-product of the additive nature of a GP predictor which has two parts. 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 predictions provided by the parametric mean functions (hence the term reversion to the mean). In ordinary GPs, no parametric functions are used and hence their extrapolation capabilities are minimal.

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 or kernels are designed to either discover and preserve patterns [21] or decay slower as the distance between a query point and the training data increases [22–24]. Pattern preserving kernels are built by modeling the spectral density—the Fourier transform of a kernel—by a parametric model (e.g., a Gaussian mixture) whose Fourier transform can be obtained analytically. While these kernels are powerful and provide insights into the data trends, their applications are limited to low-dimensional problems whose the patterns rely on trigonometric functions. GPs with slow decaying kernels break down in extrapolation if the distance between the query point and the training data is large.

In the second category, a set of parametric basis functions such as polynomials and trigonometric functions are employed in training to build the so-called universal GPs [9,25]. Each function's significance is then determined by their coefficients which are often estimated via MLE. A large (small) coefficient indicates the importance (irrelevance) of the corresponding function in explaining the relations between the inputs and outputs in the training data. If many basis functions are used, regularization must be exercised to avoid overfitting [14]. 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. This limitation also applies to methods that combine sparse regression with GPs.

Our approach for training GPs with extrapolation capabilities is similar to the second category but unlike prior research we automatically discover the parametric mean functions that must be used in GP modeling. Figure 2 demonstrates the simplified flowchart of our framework. We start by fitting an ordinary GP model to the original training data which can be sparse and/or noisy. We then use this GP model to generate training data for EP which discovers some bases (i.e., symbolic functions) that regress the new data. Next, we examine the bases found by EP based on a statistical measure that uses SVD, MLE, and bootstrap sampling. We select the bases with the best measure, include them in the discovered bases (DBs) repository, and repeat the preceding steps while adapting some of them to consider the most updated version of the DBs repository. Once the convergence criterion is met, we stop the iterations. The output of the framework is a GP model whose mean function is the DBs repository which improves not only the predictive power of GP (in both interpolation and extrapolation) but also its numerical stability. The rationales behind each step of our framework are detailed in Sec. 3.

Our efforts toward building emulators with extrapolation capabilities have some connections with (inverse) system identification and symbolic regression. The goal of system identification is to discover

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received November 16, 2020; final manuscript received March 18, 2021; published online May 28, 2021. Assoc. Editor: Mian Li.

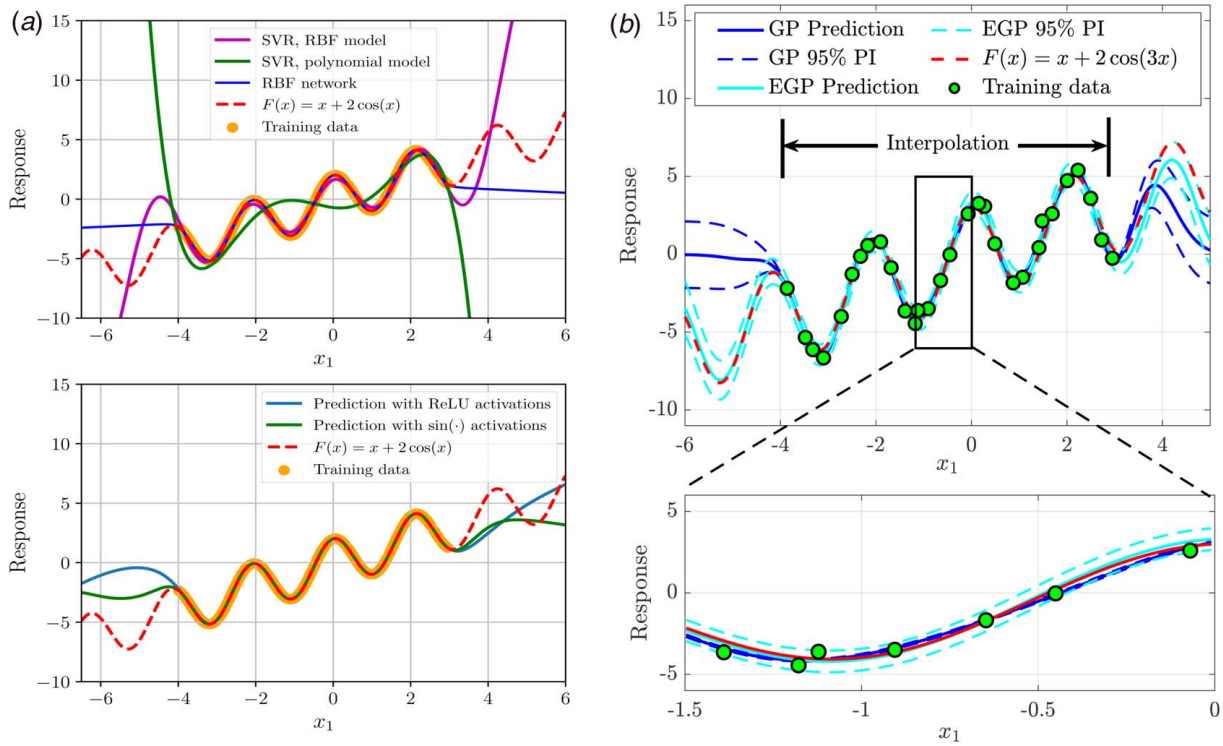

Fig. 1 **(*a*) *Extrapolation errors of various emulators*: $F(x) = x + 2\cos(3x)$ is learned over the interval $[-4, 3]$ and extrapolated over $[-6.5, 6]$. In the top panel, SVR (with two different kernels) and RBF are trained with 100 noiseless samples. In the bottom panel, 5000 noiseless samples are used to train two NNs with either ReLU or sin(.) activations (both NNs have 7 hidden layers and 15 neurons per layer). (*b*) *GP versus EGP*: GP and EGP both learn $F(x) = x + 2\cos(3x)$ using some noisy training data (noise variance is 0.25). EGP outperforms ordinary GP not only in extrapolation but also in interpolation. The prediction interval (PI) provided by EGP in extrapolation is also more accurate than the interval obtained via GP.**

the underlying governing dynamics of a system using some observed data [16,26–28]. The objective of symbolic regression is to discover a set of symbolic functions that regress the training data well [17,29–33]. With either of these two methodologies, the trained model provides extrapolation capabilities as long as the underlying physics of the data source (i) is effectively learned over the support of the training data and (ii) does not change outside of the training domain.

Different techniques such as EP, sparse regression [26,29,30], and NNs [31,32] can be used for system identification or symbolic regression but none is free from limitations. For instance, EP is very sensitive to noise and tends to output local optima that poorly extrapolate. Sparse regression requires a priori specification of the bases or symbolic functions. Hence, it is limited to simple and low-dimensional problems. NNs with symbolic activation functions provide limited representation power because with non-trivial activation functions the gradients quickly vanish/explode or the stochastic gradient descent fails to converge to sufficiently good local optima [1,34]. Among these methods, some of the limitations of EP can be methodically addressed by our framework, and hence, we employ EP to discover parametric bases in our approach.

The rest of the paper is organized as follows. In Sec. 2, we provide some technical background on GP modeling and EP. We elaborate on the algorithmic details of our approach in Sec. 3 and compare its performance with ordinary GPs and EP on a set of analytical functions and an engineering problem in Sec. 4. We summarize our contributions, discuss limitations, and provide future research directions in Sec. 5.

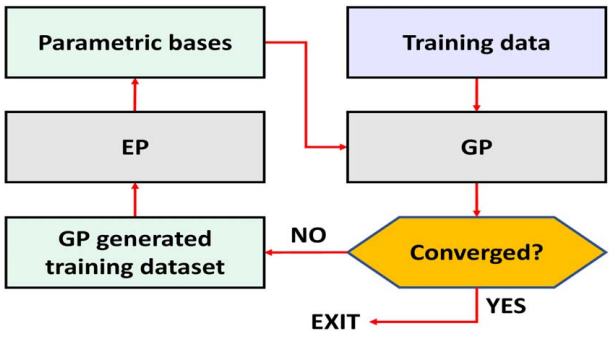

Fig. 2 ***Simplified flowchart of our approach*: We begin by learning an ordinary GP, i.e., one without any parametric bases. A large dataset is then generated with this GP and used in EP to find some bases. We analyze these bases and select the most fitting ones. Afterwards, we retrain the GP while using the selected bases. This iterative process is continued until the convergence criterion is met.**

2 Technical Background

In this section, we review GP modeling and EP since they are the backbones of our approach. We also discuss some of their limitations that our approach aims to address.

2.1 Emulation With Gaussian Processes. Let us denote the output and inputs in the training data by y and $\boldsymbol{x} = [x_1, x_2, \ldots, x_d]^T$, respectively, where $y \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^d$. For GP modeling, we assume that the input(s)–output relation is a single realization of the random process, $\eta(\boldsymbol{x})$, given by

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

where $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \ldots, f_h(\boldsymbol{x})]$ are some pre-determined set of bases (e.g., $\sin(x_1)$, $\log(x_1 + x_2)$, $(x_1^2 + x_2)$, $\ldots$), $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_h]^T$ are

unknown coefficients, and $\xi(\boldsymbol{x})$ is a zero-mean GP. $\xi(\boldsymbol{x})$ is completely characterized by its co-variance function, $cov(\cdot,\cdot)$, defined as

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

where σ^2 is the process variance and $r(\cdot,\cdot)$ is the correlation function. Many parametric correlation functions have been developed for GPs [4,5,7,9,24,35–37] with the Gaussian correlation function being the most commonly used one:

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

where $\boldsymbol{\omega} = [\omega_1, \ldots, \omega_d]^T, -\infty < \omega_i < \infty$ are the roughness or scale parameters (in practice the ranges are limited to $-10 < \omega_i < 6$ to avoid numerical errors). The collection of σ^2 and $\boldsymbol{\omega}$ is called the hyper-parameters of $\xi(\boldsymbol{x})$. Following the assumption in Eq. (1) and given the n training pairs of $(\boldsymbol{x}_{(i)}, y_{(i)})$, 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 some prior knowledge on these parameters is available. In this paper, the MLE approach is employed as it provides a high predictive power while minimizing the computational costs [24,35,38–41].

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

$$\left[\widehat{\boldsymbol{\beta}}, \widehat{\sigma^2}, \widehat{\boldsymbol{\omega}}\right] = \underset{\boldsymbol{\beta},\, \sigma^2,\, \boldsymbol{\omega}}{\arg\max} \mid 2\pi\sigma^2 \boldsymbol{R} \mid^{-\frac{1}{2}} \times \exp\left(\frac{-(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})^T(\sigma^2\boldsymbol{R})^{-1}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})}{2}\right)$$

which can also be written as

$$\left[\widehat{\boldsymbol{\beta}}, \widehat{\sigma^2}, \widehat{\boldsymbol{\omega}}\right] = \underset{\boldsymbol{\beta},\, \sigma^2,\, \boldsymbol{\omega}}{\arg\min} \frac{n}{2}\log(\sigma^2) + \frac{1}{2}\log(\mid \boldsymbol{R} \mid) + \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta})^T\boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{\beta}) \tag{4}$$

where $\log(\cdot)$ is the natural logarithm, $|\cdot|$ indicates the determinant operator, $\boldsymbol{y} = \left[y_{(1)}, \ldots, y_{(n)}\right]^T$ 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 $R_{ij} = r(\boldsymbol{x}_{(i)}, \boldsymbol{x}_{(j)})$ for $i, j = 1, \ldots, n$, and $\boldsymbol{F}$ is the $n \times h$ matrix with (k,l)th element $F_{kl} = f_l(\boldsymbol{x}_{(k)})$ for $k = 1, \ldots, n$ and $l = 1, \ldots, h$. The point estimates of $\boldsymbol{\beta}$ and σ^2 can be represented as a function of $\boldsymbol{\omega}$ by setting the partial derivative of Eq. (4) to zero:

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

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

Plugging these estimates in Eq. (4) and eliminating the constants results in

$$\widehat{\boldsymbol{\omega}} = \underset{\boldsymbol{\omega}}{\arg\min}\, n\log(\widehat{\sigma^2}) + \log(\mid \boldsymbol{R} \mid) = \underset{\boldsymbol{\omega}}{\arg\min}\, L \tag{7}$$

By numerically minimizing L in Eq. (7) one can find $\widehat{\boldsymbol{\omega}}$ and, subsequently, obtain $\widehat{\boldsymbol{\beta}}$ and $\widehat{\sigma^2}$ using Eqs. (5) and (6). Many heuristic global optimization methods such as genetic algorithms [42], pattern searches [43,44], and particle swarm optimization [45] 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 [3,35,46]. 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 $\boldsymbol{x}^*$:

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

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

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

where $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{f}^T(\boldsymbol{x}) - \boldsymbol{F}^T\boldsymbol{V}^{-1}\boldsymbol{g}(\boldsymbol{x})$. Equation (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 $\boldsymbol{g}(\boldsymbol{x}^*) \approx 0$, the posterior mean only depends on $\boldsymbol{f}(\boldsymbol{x}^*)\widehat{\boldsymbol{\beta}}$. Thus, 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 over-fitting) via the so-called nugget or jitter parameter, δ. To this end, $\boldsymbol{R}$ is replaced with

$$\boldsymbol{R}_\delta = \boldsymbol{R} + \delta\mathbb{I}_{n\times n} \tag{10}$$

If δ is used, the estimated (stationary) noise variance in the data is $\delta\widehat{\sigma^2}$.

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 notion of evolution to programming. This perspective on computer modeling has fathered an active field of research known as EP. The terms genetic programming [47,48], inverse system identification [15], grammatical evolution [17], symbolic regression [33], and structure learning [49] embody 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 from a selected pool of functions and operators an initial population of simple solutions known as individuals or candidates. Then, these solutions are iteratively evolved by mutation and crossover operations to create more complex individuals that hopefully better relate the inputs and output based on some predefined criterion. This process is continued until convergence when the desired individuals are returned. EP usually optimizes the fitness of the individuals but multiple criteria can be optimized at the same time by means of a Pareto frontier (see Sec. 3.2).

Koza provided the first systematic approach for using EP in symbolic regression and robot planning. In his paper [47], Koza models potential solutions to these problems as trees where nodes host primitive operations and functions while leaves store variables. In his approach, new solutions are found by combining these trees together and randomly changing parts of them until at least one of them can faithfully learn the data. Since Koza's seminal work, EP and its many variants have been successfully used in many complex problems. A particularly active application has been symbolic regression where EP is used to automatically find the governing equations of simple mechanical systems [15,16,27,28,30,33,50].

We conclude this subsection with two important notes. First, EP is fundamentally different than sparse regression [14,26,29,30,51,52]. While in EP, the input–output relation is discovered; in

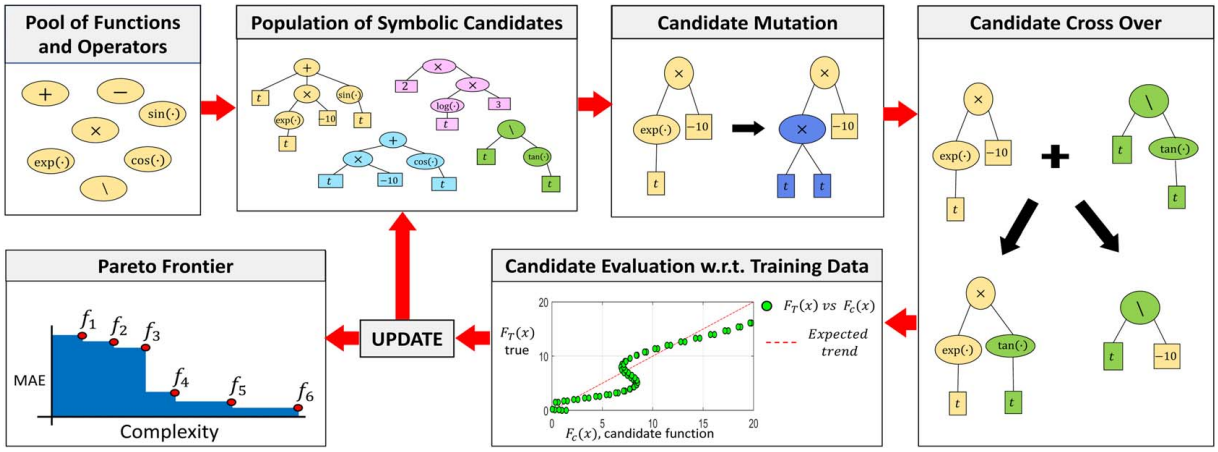

Fig. 3 ***Symbolic regression via evolutionary programming*: A pool of functions and operators is preinitialized. Then, random combinations of these functions are used to initiate a population of individuals. At each iteration, these individuals undergo mutation and crossover operations to produce individuals with higher fitness over the next generations. This evolutionary process is continued until convergence where a Pareto frontier of fitness versus complexity is obtained.**

sparse regression, a specific functional form is assumed and the data are used to estimate the parameters of that functional form. Second, 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.

3 Evolutionary Gaussian Processes

The algorithmic principles behind EGP are motivated by the posterior mean in Eq. (8). To improve the predictive power of this posterior (especially in extrapolation) a set of *free-form* parametric mean functions, $\boldsymbol{f}(\boldsymbol{x})$, can be learned. Hence, we design a data-driven framework to train a GP while simultaneously searching for some $\boldsymbol{f}(\boldsymbol{x})$ that capture the trend of the data over the training domain reasonably well. To find these $\boldsymbol{f}(\boldsymbol{x})$, we use EP to search the space of parametric functions spanned by polynomials of arbitrary degree, trigonometric functions, $\exp(\boldsymbol{x})$, $\log(\boldsymbol{x})$,…, and combinations thereof.

Figure 4 illustrates the mechanics of EGP which, in essence, is a systematic integration of GP and EP. Our framework is iterative (reasons to be discussed) and includes data-driven measures (inspired by the MLE and bootstrap sampling) that vet the bases found by EP.

Given a noisy and/or sparse dataset, we start the first iteration by fitting an ordinary GP, and computing and storing the iteration's convergence parameter (α^1). The computation procedures and motivations for the convergence parameter α^i are detailed in Sec. 3.1). Then, we use this GP to generate a new training dataset for EP. This new data have $\lfloor \mu \times n \rfloor$ samples ($\mu \in \mathbb{R}$ and $\mu \geq 1$) and are generated via Sobol sequence [53,54] over the same domain as the original data. They are better suited for EP since GP reduces noise and can generate as many data points as EP needs.

We then use EP for multi-objective symbolic regression to find a Pareto frontier of candidate solutions ($G^i(\boldsymbol{x})$) that optimizes for both fitness (mean absolute error, MAE) and complexity (for more details see Sec. 3.2). Next, we decompose each $G^i(\boldsymbol{x})$ into its components or *bases* $[g_1^i(\boldsymbol{x}), \ldots, g_q^i(\boldsymbol{x})]$ which are defined as its individual additive terms (excluding the coefficients and the constants). For instance, the bases in the candidate $G^i(\boldsymbol{x}) = x_1 +$

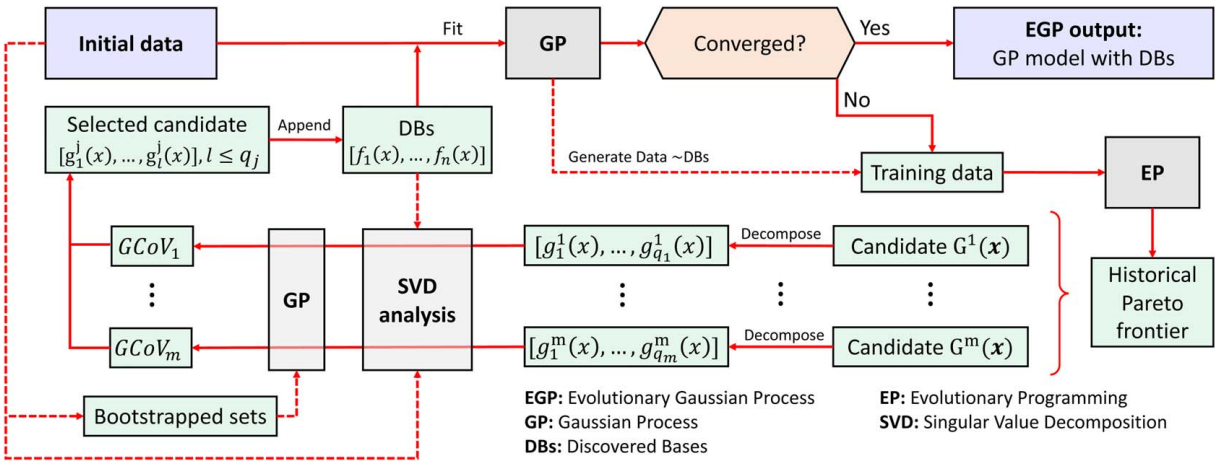

Fig. 4 ***Detailed flowchart of EGP*: We use the initial training data to fit a GP whose mean function is composed of the parametric functions stored in the discovered bases (DBs) repository. We then generate training data for EP via this GP such that the effect of DBs is excluded. We decompose the candidate solutions discovered by EP into their primitive bases, remove linear dependencies, and analyze their robustness via several trainings on bootstrapped sets. We select the best candidate and append its linearly independent bases to the DBs. We retrain the GP with the updated DBs and check for convergence.**

$x_2(x_3+2)+1$ are $[g_1^i(\boldsymbol{x}), g_2^i(\boldsymbol{x}), g_3^i(\boldsymbol{x})] = [x_1, x_2, x_2x_3]$). As explained below, this decomposition strategy is adopted to allow for not only removing redundant bases (that EP overfits to the data) but also refining the coefficient of the informative bases that capture the trend of the data.

To remove redundant bases in $G^i(\boldsymbol{x})$, we use SVD to eliminate the linearly dependent ones since they adversely affect the numerical stability of the ensuing steps that involve GP (if a set of linearly dependent bases are used in GP modeling, the $\boldsymbol{F}^T\boldsymbol{R}^{-1}\boldsymbol{F}$ term in Eq. (5) will not be invertible). Our SVD analysis is performed iteratively on the $\boldsymbol{F}$ matrix, which we build using the original training data and the bases. If dependencies are detected (i.e., the minimum singular value is below a threshold T), the most complex component is removed. This process is repeated until the set of bases is free from linear dependencies. Once the linearly independent bases of $G^i(\boldsymbol{x})$ are determined, we use them to calculate the global coefficient of variation (GCoV, detailed in Sec. 3.3) of $G^i(\boldsymbol{x})$. The candidate with the smallest GCoV is then selected and its linearly independent bases are temporarily included in the repository of DBs. Finally, we use the stored basis in the DBs and the original training data to train a GP and calculate α^2. α^2 is compared to α^1 and the result used as the convergence criterion (detailed in Sec. 3.1). If $\alpha^2 < \gamma\alpha^1$, with $\gamma \in (0, 1)$, the bases are permanently appended to the DBs repository and the algorithm progresses to the next iterations and continues until this convergence criterion is met. If $\alpha^2 >= \gamma\alpha^1$, the iteration is repeated and if the convergence parameter does not improve in this second trial, the algorithm is terminated.

Iterations two, three, … differ from the first one in two major aspects. First, the following equation is used to generate training data for EP:

$$y^*(\boldsymbol{x}^*) = y^*_{GP}(\boldsymbol{x}^*) - \boldsymbol{f}(\boldsymbol{x}^*)\widehat{\boldsymbol{\beta}}$$

where $\boldsymbol{f}$ and $\widehat{\boldsymbol{\beta}}$ are, respectively, all the bases from the DBs repository and its corresponding coefficients estimated via MLE. This strategy greatly helps EGP in that the evolutionary process no longer needs to discover what is learned in the previous iterations. The second major difference is that all the bases in the DBs repository are also included in the SVD analyses to prevent the addition of bases to the repository that are linearly dependent with the available ones. It is noted that to prevent forgetting what the algorithm has already learned, the SVD analyses of iteration i never eliminate the bases discovered in the previous iterations, no matter the complexity.

3.1 Convergence Criterion. The convergence parameter α^i, for a particular iteration i, ensures that the updates on the DBs after the iteration increase the predictive power of EGP (especially in extrapolation). α^i is calculated as follows. First, the original training dataset is divided into two mutually exclusive and collectively exhaustive parts: an interpolation set that encompasses 80% of the interior of the training domain (e.g. if $x \in [0, 1]$, the interpolation set is $x \in [0.1, 0.9]$) and an extrapolation set that includes the rest of data in the training domain. Then, a GP with the bases from DBs repository (if any) is fitted to the interpolation set and used to calculate α^i which is the mean-squared error (MSE) in the extrapolation set.

The value of α^i across two consecutive iterations governs convergence. In particular, EGP converges in iteration $i > 1$ if $\alpha^{i+1} \geq \gamma\alpha^i$ with $\gamma \in (0, 1)$, i.e., if the updates of the DBs in iteration $i+1$ incorrectly model the data or do not sufficiently increase the accuracy.

3.2 Historical Pareto Frontier. EP optimizes for both complexity and fitness (i.e., MAE) which are generally opposing objectives. That is, complex candidate solutions or individuals are unfavorable but they tend to have better fitness values. As a result, EP builds a Pareto frontier of individuals who indicate the fittest candidate for any complexity, see Fig. 5. The complexity of a candidate solution is calculated by summing the complexity numbers assigned to its entailed operations: Simple operations ($+, -, \times$) have complexity 1, division has complexity 2, and functions (e.g., trigonometric, exponential, logarithm,…) have complexity 3.

The Pareto frontier is usually built with the individuals of the last generation. However, our studies on EP indicate that these individuals do not necessarily provide the optimum compromise between fitness and complexity. This is because EP either achieves local optimality or overfits the noise that GP has failed to filter out. To address this issue, we track all the individuals generated during the entire evolution process (regardless of the population or generation number) and build the historical Pareto frontier.

3.3 Global Coefficient of Variation. The candidates obtained via EP generally overfit the data even though we use the historical Pareto frontier, filter out noise with GP, and learn them iteratively. To address this issue, we calculate the GCoV for each candidate. GCoV is inspired by MLE and bootstrap sampling and measures the robustness of the coefficients of a candidate's bases to variations in the training data. The candidate whose linearly independent bases are more robust has a smaller GCoV and is selected over other candidates.

To obtain GCoV for the candidate solutions, we first bootstrap the initial data to generate b datasets of size $n' < n$ where n is the size of the initial data. Then, for $G^i(\boldsymbol{x})$ with l linearly independent bases, we use the following equation to calculate the CoVs for

ID	Equation
1	x
2	$0.17+x$
3	$0.14+1.07x$
4	$x+0.06x^2$
5	$x+\cos(3.00x)$
6	$1.03x+0.94\cos(3.00x)-0.02$
7	$1.05x+0.92\cos(2.99x)-0.06x\ \cos(2.98+x)$
8	$1.05x+0.92\cos(2.98x)-0.04-0.07x\ \cos(2.54+x)$
9	$1.07x+0.95\cos(2.93x)-0.11-0.16x\ \cos(2.03+1.26x)$
10	$(1.06x+0.96\cos(2.91x)-0.12-0.18x\ \cos(1.93+1.27x)$ $-0.01x^2\cos(1.93+1.27x)$

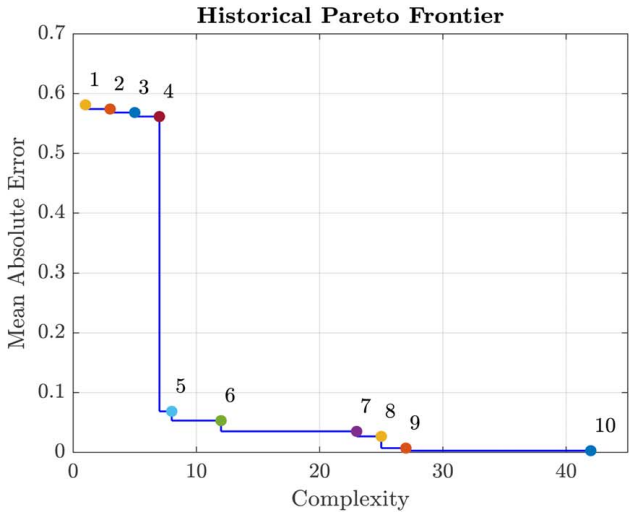

Fig. 5 ***Example of historical Pareto frontier*: EGP emulates the function $F(x) = x + 2\cos(3x)$ using some noisy training data. The individual candidates comprising this historical Pareto frontier are obtained by EP in the first iteration of EGP.**

Table 1 ***Analytical examples*: The functions posses different dimensionality, complexity, and ranges**

ID	Function	Min($\boldsymbol{x}$)	Max($\boldsymbol{x}$)	Mean range (y)
1	$y(x) = 943.61(x_1 - 2.8)^2 + 409.40\ (x_2 - 2.0941)^2 + 4.15(1 - \cos(2x_3)) + 3.28\left(\left(\frac{3.46}{x_4}\right)^3 - \left(\frac{3.46}{x_4}\right)^2\right)$	[2.12, 0.85, 0, 2]	[4.32, 3.35, 3.14, 7]	2602
2	$y(x) = x_7^3 - x_5 + 0.2x_7x_3x_1 + 4x_6 - x_2x_3 + x_1 - x_2 + 15\sin(2x_1) - 10\cos(x_3) - 5$	[−4.5, −6, 1, −3, −3, −2.5, −3.5]	[3.5, 2, 6, 4, 2, 3.5, 4.5]	188
3	$y(x) = 2\sin(x_1) - 3\cos(x_2) + x_1 + x_2^2$	[−3, −5]	[2, 2]	36
4	$y(x) = x_1^2 + \frac{x_1}{2x_2 - 1} + \sqrt{(2)}x_3x_4 + 2.3e^{-x_5} + \sqrt{(x_2)}$	[−3, 0.54, −2, −3, −4.5]	[3.5, 4.54, 3, 1, 0.5]	219.97
5	$y(x) = (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2 + x_1x_2 + 4x_2^4 - 4x_2^2$	[−2, −1]	[2, 1]	5.58
6	$y(x) = \log(\lvert x_3\rvert)x_1 + 3.2\tan(x_1) + \sqrt{\lvert x_2 + 3\rvert}$	[−1.2, −2, 0.7]	[1.2, +4, 5]	19.6
7	$y(x) = \frac{2\pi x_1(x_2 - x_3)}{\left(\log\left(\frac{x_4}{x_5}\right)\right)\left(1 + \frac{x_1}{x_6} + \frac{2x_1x_7}{\log(x_4/x_5)}x_5^2x_8^2\right)}$	[63070, 990, 700, 100, 0.05, 63.1, 1120, 9855]	[115600, 1110, 820, 50000, 0.15, 116, 1680, 12045]	223
8	$y(x) = \left(\exp(-(x_2 - 1)^2) + \exp(-0.8(x_2 + 1)^2) - 0.05\sin(8(x_2 + 0.1))\right)$ $\left(\exp(-(x_1 - 1)^2) + \exp(-0.8(x_1 + 1)^2) - 0.05\sin(8(x_1 + 0.1))\right)$	[−3, −3]	[3, 3]	1.11

Note: The average range of each function is calculated using multiple training sets that are generated over the input space.

the coefficients of its bases:

$$CoV_j = \frac{\text{std}(\beta_j^1, \ldots, \beta_j^b)}{\text{mean}(\beta_j^1, \ldots, \beta_j^b)}, \quad j = 1, \ldots, l$$

where β_j^k is the coefficient of basis j when the set k is used in MLE. The MLE process involves fitting a GP using the l linearly independent bases of the candidate and the n bases currently found in the DBs repository as parametric mean functions. Once these CoVs are calculated, we determine the GCoV of the corresponding candidate via penalized weighted averaging., i.e.,

$$GCoV = \frac{\sum_{j=1}^{l} (CoV_j)^p}{l^q}$$

where p and l^q penalize (i.e., increase) the GCoV of a candidate if it has unstable coefficients or too many bases, respectively.

3.4 Discussions. The underlying assumption behind EGP is that there are some free-form parametric bases that can model the data source reasonably well. As a result, if we can *discover* these bases via some training data and use them in GP modeling, we can build GPs that outperform ordinary GPs in extrapolation, interpolation, and regression. If this assumption is incorrect (or EGP fails to find bases that explain the data well), the output of our algorithm would be an ordinary GP.

As we show in Sec. 4.1, another potential advantage of EGP over other GP modeling approaches is increased numerical stability. We can quantify this advantage by analyzing the smallest eigenvalue of $\boldsymbol{R}_{opt} = \boldsymbol{R}_\delta(\boldsymbol{\omega} = \widehat{\boldsymbol{\omega}})$. In the extreme case where $\boldsymbol{f}(\boldsymbol{x})\widehat{\boldsymbol{\beta}}$ completely explains the data, $\xi(\boldsymbol{x})$ in Eq. (1) models either the noise (i.e., the data is noisy and $\boldsymbol{f}(\boldsymbol{x})\widehat{\boldsymbol{\beta}}$ regresses the data) or a random process that is zero everywhere (i.e., the data does not have noise and $\boldsymbol{f}(\boldsymbol{x})\widehat{\boldsymbol{\beta}}$ interpolates the data). In either of these scenarios, $\widehat{\omega}_i \ll 0$ (if the nugget is used correctly in the optimization [35]), and hence, $\boldsymbol{R}_\delta = \boldsymbol{R} + \delta\mathbb{I}_{n\times n} \approx \boldsymbol{I}_{n\times n} + \delta\mathbb{I}_{n\times n}$, which is well-conditioned as the smallest eigenvalue is $1 + \delta$.

In most problems, the extreme case considered above is not observed, and hence, $\boldsymbol{R} \neq \boldsymbol{I}_{n\times n}$. Since $\boldsymbol{R}$ is positive-definite, the smallest eigenvalue of $\boldsymbol{R}_{opt} = \boldsymbol{R}_\delta(\boldsymbol{\omega} = \widehat{\boldsymbol{\omega}}) = \boldsymbol{R}(\boldsymbol{\omega} = \widehat{\boldsymbol{\omega}}) + \delta\mathbb{I}_{n\times n}$ is close to δ and thus $\boldsymbol{R}_{opt}$ is again well-conditioned. It is also noted that our SVD analyses ensure that the term $\boldsymbol{F}^T\boldsymbol{R}^{-1}\boldsymbol{F}$ is of full rank and hence invertible.

Lastly, we point out that while the goal of EGP is not symbolic regression, it can be used for that purpose. We demonstrate this capability of EGP in Appendix B.

4 Results

In this section, we compare the performance of EGP in interpolation and extrapolation against ordinary GP and EP. In Sec. 4.1, we compare them using eight analytical examples that include different dimensions and noise levels. We also use these examples to demonstrate the superior numerical stability of EGP over ordinary GP. In Sec. 4.2, we apply these three emulation techniques to an engineering problem where very limited prior knowledge on the functional form of the response is available.

In all examples, the following parameters are selected for EGP: $\mu = 1.5$, $\gamma = 0.9$, $T = 10^{-4}$, $n' = \frac{1}{3} \times n$, $b = 10$, $p = 3$, and $q = 0.7$. The rationales behind these choices are explained in Appendix A.

We use the algorithm described in Ref. [35] for GP modeling which employs an adaptive mechanism to estimate the nugget variance. For symbolic regression, we use Eureqa [33] which is a highly sophisticated implementation of EP. In all of our studies, the convergence criterion of Eureqa is based on the stagnation of MAE across the candidates. We note that many aspects of EP (e.g., population size, crossover rate, and mutation rate) are internally determined in Eureqa and cannot be manually tuned.

4.1 Analytical Functions. The analytical functions we study are shown in Table 1. Many of them are relatively high dimensional

Table 2 ***Noise variance in analytical examples*: Normal noise is added to the training data**

	ID 1	ID 2	ID 3	ID 4	ID 5	ID 6	ID 7	ID 8
Small noise variance	0.75^2	1^2	0.8^2	1^2	0.04^2	0.1^2	1^2	0.01^2
Large noise variance	2.5^2	3^2	3^2	3.5^2	0.4^2	1^2	3^2	0.05^2

Note: For each example, two noise levels are considered (one small and one large). The range of the function (see the last column of Table 1) is used to determine the magnitude of the noise variances.

functions and include terms that possess asymptotic behavior, have a high-frequency, or contain nested functions. Each function is emulated under two different scenarios where the training data are corrupted by a Gaussian noise with either a small or large noise variance, see Table 2. The noise variances are selected based on the function range (see the last column in Table 1) where a small (large) noise variance corrupts the response by up to 1% (10%).

Sobol sequence [53,54] is used to generate the training and validation data. The size of the training dataset is $50 \times Dim$ where Dim indicates the dimensionality of the underlying function. The validation dataset for each case is corrupted by noise as well and comprises two mutually exclusive parts designed to assess the interpolation and extrapolation capabilities of GP, EP, and EGP. The interpolation error of each emulator is calculated over the training domain. However, the extrapolation error is measured over a domain that excludes the training region and its bounds are obtained by expanding $\min(\boldsymbol{x})$ and $\max(\boldsymbol{x})$ in Table 1 by 40%, unless the function loses continuty. For instance, if the training range is [0, 1], the interpolation and extrapolation errors are calculated over, respectively, [0, 1] and [−0.2, 1.2] ranges and if the function presents a discontinuty at −0.1, the extrapolation ranges become (−0.1, 1.2]. Errors are quantified via normalized root mean squared error (NRMSE) using $500 \times Dim$ samples. To account for sample-to-sample variations, the training-validation process is repeated five times. Lastly, in example 3, trigonometric functions are intentionally excluded from the primitive function set of EP.

4.1.1 Results. Figure 6 summarizes the results for the cases where the noise variance is small. Interpolation-wise, EGP outperforms EP across all our cases except for example 6 where EP results in slightly less variations across the repetitions. When comparing EGP to ordinary GP, two trends are observed. In examples where some bases are discovered by our algorithm (IDs 1,2,4,5,6), EGP achieves smaller interpolation errors than GP. In examples where no bases are discovered (IDs 3,7,8), the performance of EGP and GP is identical.

The primary advantage of EGP is observed in extrapolation (see Fig. 6). In examples 1, 2, 5, and 6 where some adequate bases are discovered, the extrapolation capability of EGP is more than EP and ordinary GP. This higher accuracy is because EGP (i) does not suffer from reversion to the mean as much as ordinary GP does and (ii) has multiple mechanisms (unlike a standalone EP software such as Eureqa) that exclude bases which incorrectly capture the trend of the data. In example 4, although some adequate bases are discovered, the extrapolation performance is not improved. This is because the base $\frac{x_1}{2x_2-1}$, which has a large asymptotic behavior on the training domain, is not properly learned. In examples 3, 7, and 8 where no bases are included in EGP, the performance is still better than EP (by avoiding overfitting) but similar to ordinary GP. We note that in one of the repetitions of example 7 (which is highly complex) an incorrect basis is accepted, and hence, the performance variability of EGP is more than GP (compare the height of the corresponding box plots)

Figure 7 summarizes the results for the cases where the noise variance is large. The observations are largely consistent with those reported in Fig. 6 with the exception of example 7. In this example, EP discovers some bases that capture part of the trend of the data in extrapolation and hence achieves smaller errors than EGP that excludes these bases as they insufficiently decrease

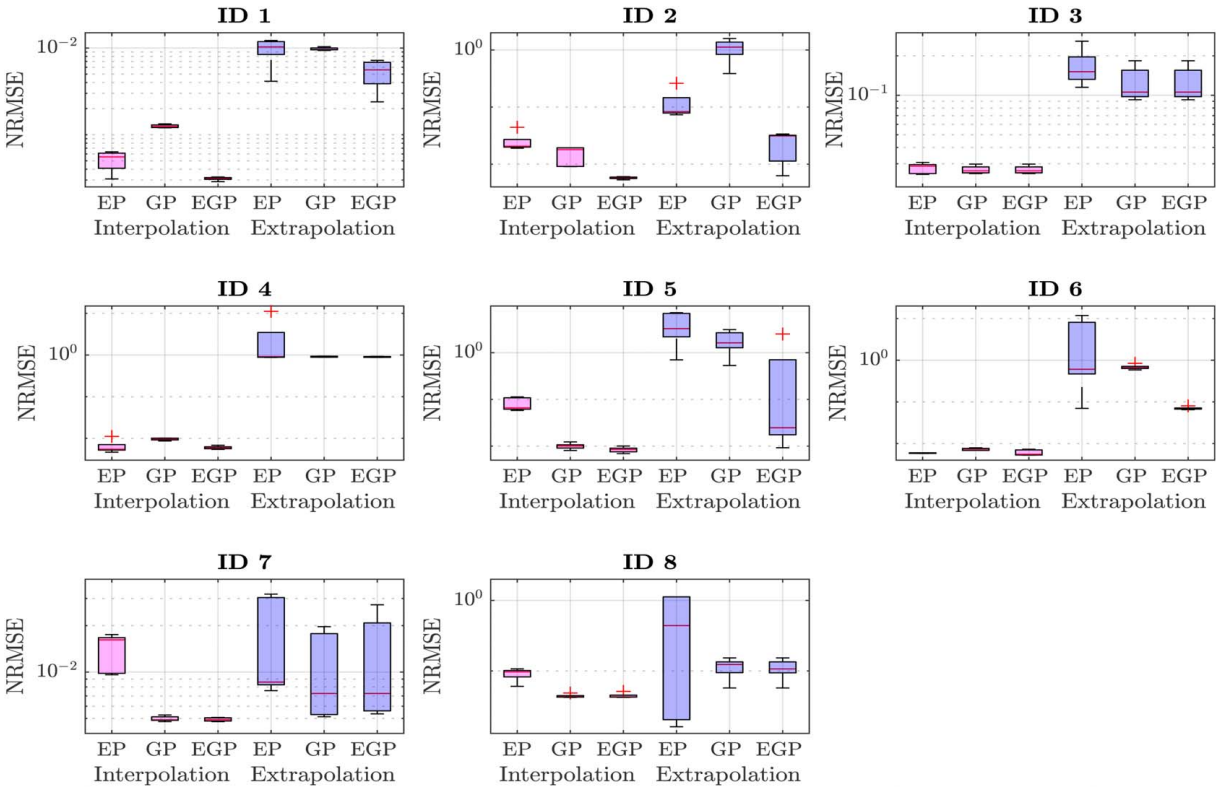

Fig. 6 ***Performance of EP, GP, and EGP (small noise scenario)*: Each subplot corresponds to one example and includes interpolation and extrapolation errors in terms of NRMSE on log scale**

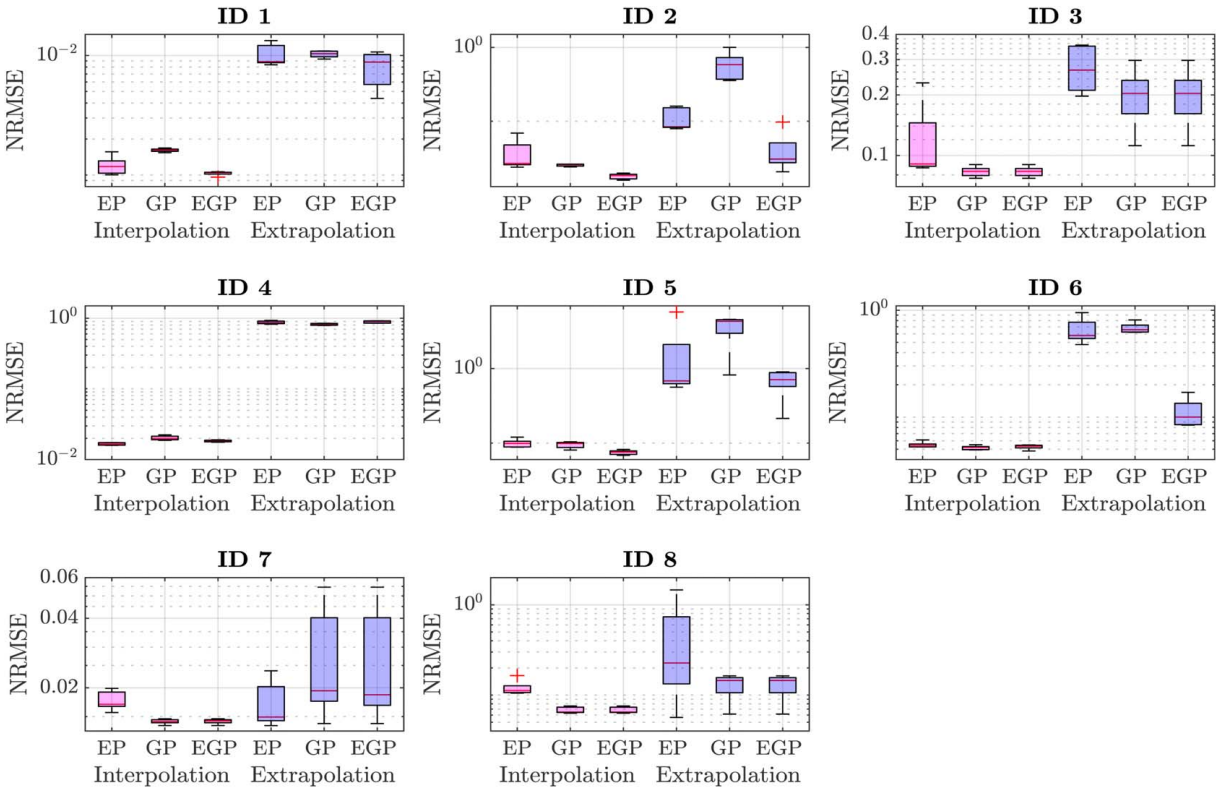

Fig. 7 ***Performance of EP, GP, and EGP (large noise scenario)*: Each subplot corresponds to one example and includes interpolation and extrapolation errors in terms of NRMSE on log scale.**

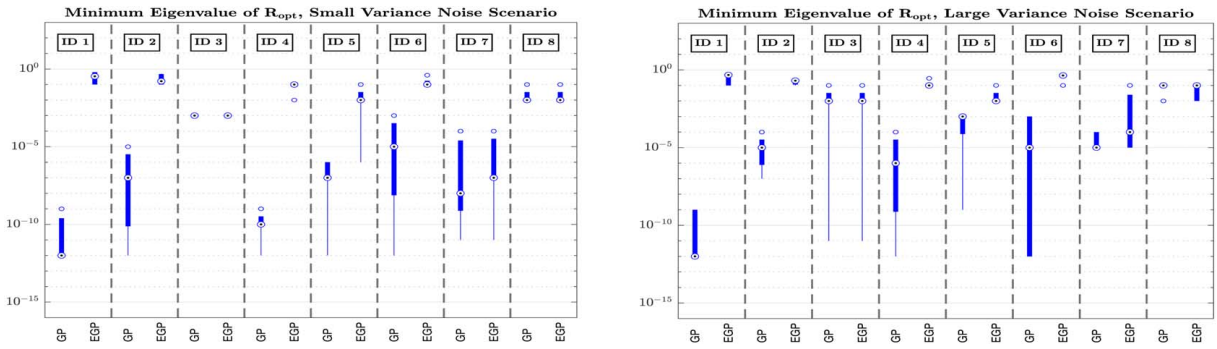

Fig. 8 ***Numerical stability of EGP versus ordinary GP:* The left (right) panel indicates the distribution of the smallest eigenvalue of R_{opt} across the eight examples when training data are corrupted by a Gaussian noise with small (large) variance.**

the convergence parameter α. Although tuning α in our algorithm can increase the performance of EGP in this example, we avoid tailoring EGP to specific problems.

With either large or small noise variance, two interesting observations are made. First, in example 3, where trigonometric functions are intentionally excluded from the evolutionary search process. This example is designed to not only resemble applications (such as the one in Sec. 4.2) where the underlying data source does not have a closed-form functional form, but also applications where the underlying physics is not effectively learned (e.g., due to the failure of the evolutionary search process). In this case, unlike EP, EGP avoids fitting high-degree polynomials to the data and hence achieves much better extrapolation accuracy. The second observation is on the numerical stability of EGP and how it compares to ordinary GP. Figure 8 quantifies the variations of smallest eigenvalue of $\boldsymbol{R}_{opt}$ in all of our studies and indicates that whenever some bases are discovered by EGP (examples 1, 2, 4, 5, and 6), this measure and thus the numerical stability are higher.

Table 3 ***Performance of GP, EGP, and EP in the engineering example*: MSE results of EP, GP and EGP on the different sets**

	Training error	Interpolation error	Extrapolation error
EP	27.08	48.66	169.18
GP	1.69e-05	0.185	92.73
EGP	1.28e-05	0.139	56.08

Note: The response range in the original training domain is 383.74.

4.2 Extrapolation for Material Modeling. In most real-world applications, the symbolic relation between the independent and dependent variables in a dataset is unknown. To study how our algorithm performs in these scenarios, we use EGP to learn the constitutive law of a 2D heterogeneous composite. The matrix material of the composite is a soft compressible elastomer modeled by the Arruda Boyce [55] hyperelastic constitutive model. The composite reinforcements are randomly distributed

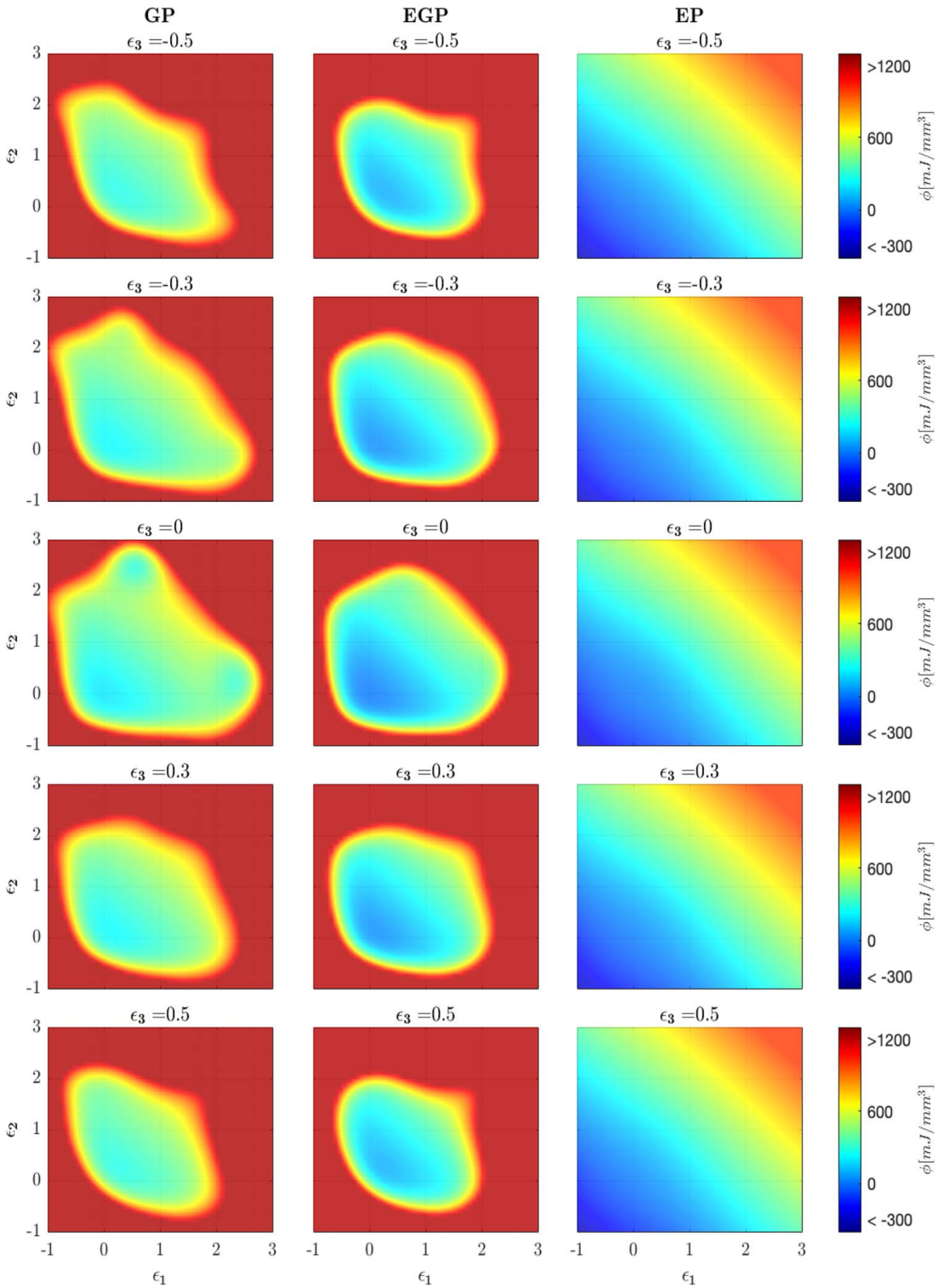

Fig. 9 ***Materials modeling with GP, EGP, and EP*: ϵ_1, ϵ_2, and ϵ_3 are Green strains and ϕ is the stored potential in the composite microstructure. The prediction domain is larger than the training domain which is $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -0.085 \leq \epsilon_1, \epsilon_2 \leq 1.275; -0.34 \leq \epsilon_3 \leq 0.34\}$.**

elliptical particles that are modeled as compressible Neo-Hookean elastomers (see more details in Refs. [56,57]). The constitutive law of our composite microstructure is unknown, and thus, numerical methods such as the finite element method (FEM) are required to find its response (i.e., the stored potential) to any applied strains. The high computational costs of FEM hinder multiscale simulations where the fine-scale behavior is governed by this microstructure. One solution for decreasing the costs is to replace FEM with a surrogate as follows. A training dataset is first built by computing (via FEM) the stored potential in the microstructure under various strain states. Then, an emulator is fitted to this dataset to serve as the constitutive law of the microstructure.

Our dataset has 1000 samples and its inputs and output are, respectively, Green strains (ϵ_1, ϵ_2, ϵ_3), and the stored deformation potential (ϕ) [mJ/mm^3]. The data has a small amount of noise which is primarily due to numerical instabilities, excessive mesh

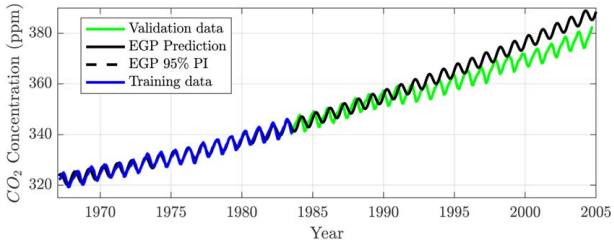

Fig. 10 ***Over confident extrapolations with EGP:*** **The monthly mean CO_2 concentration of Mauna Loa, Hawai, is learned and extrapolated. The uncertainties in extrapolation are smaller that 10^{-4}.**

distortion, and round off errors experienced in FEM (see Ref. [57] for more technical details). It is important to note that in this problem, obtaining samples that entail large deformations (i.e., large strains) takes much longer than samples that undergo small deformations. Hence, we are particularly interested in building an emulator that can predict the stored potential at large deformations using training data on small deformations.

We divide our dataset into three mutually exclusive and collectively exhaustive sets for training, interpolation testing, and extrapolation testing. The original domain sampled with FEM is $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -0.1 \leq \epsilon_1, \quad \epsilon_2 \leq 1.5; \ -0.4 \leq \epsilon_3 \leq 0.4\}$. For training and interpolation test sets, we select 85% of the original domain, i.e., $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -0.085 \leq \epsilon_1, \quad \epsilon_2 \leq 1.275; -0.34 \leq \epsilon_3 \leq 0.34\}$. This domain contains 616 data points. 75% of these data points (i.e., 462 samples) are randomly selected and used for training while the remaining 25% (i.e., 154 samples) are employed to obtain the interpolation accuracy. Data points that lay outside of this new domain (i.e., $1000 - 616 = 384$ samples) comprise the extrapolation set.

We fit three emulators to the training set via ordinary GP, EP, and EGP and predict the stored potential in the other two sets. As there is no prior knowledge on the underlying expression, we select the following primitive functions for symbolic regression in both EP and EGP: simple operations $(\pm, \times)$, division, and exponential function.

To compare the three emulators we use (i) MSE associated with interpolation and extrapolation test sets and (ii) the physical constraints that the emulators should satisfy: since they predict a stored potential, the emulators should be convex and also satisfy $\phi(0, 0, 0) = 0$ [58].

Table 3 summarizes the MSE results and indicates that EGP outperforms ordinary GP and EP in all cases. As for the physical constraints, the predicted stored potential at zero-deformation are $\phi(0, 0, 0)_{EP} = 6.71$, $\phi(0, 0, 0)_{GP} = 0.0027$, and $\phi(0, 0, 0)_{EGP} = 0.0021$ which show that EGP is more accurate. The constraint on convexity is evaluated by visually comparing the response surface of each emulator over the region $\{(\epsilon_1, \epsilon_2, \epsilon_3) \in \mathbb{R}^3 \mid -1 \leq \epsilon_1, \epsilon_2 \leq 3; -0.5 \leq \epsilon_3 \leq 0.5\}$ which is much larger than the original domain. The response surfaces are plotted in Fig. 9 and demonstrate that EGP neither fluctuates (unlike EP) nor regresses to the mean (unlike ordinary GP).

5 Conclusions

In this paper, we introduced EGP which leverages EP to build a GP model with automatically discovered symbolic bases. We tested the performance of EGP against ordinary GP and EP using some analytical functions as well as an engineering problem on learning the constitutive law of a heterogeneous microstructure. The analytical examples showed that, when our algorithm discovers bases that can explain the data reasonably well, EGP achieves lower extrapolation errors, is more numerically robust, and performs better in interpolation/regression. In the engineering example on materials modeling, EGP produced an emulator that better satisfies the physical constraints.

While we believe the primary advantages of EGP are realized in engineering problems where the underlying relation between the inputs and outputs are unknown, our studies on the analytical examples indicated the following three limitations. First, the computational cost of training an EGP is higher than that of an ordinary GP. The additional cost is unfavorable if our algorithm results in an ordinary GP which happens when the underlying assumptions do not hold or when EP does not discover any informative bases. In this case, the output of EGP is an ordinary GP, but several expensive computations are performed that add no performance improvement compared to an ordinary GP. The second limitation is on interfacing with Eureqa which we use for symbolic regression. Eureqa is not open-source and hence each iteration of our algorithm involves (i) manual copy-pasting of the GP-generated data set into Eureqa and (ii) writing subroutines that automatically convert the results of Eureqa into PYTHON-readable objects. While EGP generally converges in a few iterations, the manual data transfer is tedious and prone to error. We have tried open-source symbolic regression packages such as DEAP [48] which can be easily integrated with our codes. However, the symbolic regression power of Eureqa far exceeds other available packages and thus we used Eureqa in our approach. The third limitation arises when the discovered bases regress the data very well which results in over confident extrapolation so the PI for the extrapolation match the predictions, see Fig. 10.

Future research directions include extension to mixed-variable problems that include qualitative and quantitative inputs, improving interpolation/regression capabilities of GPs without the need for improving their extrapolation power, implementing an in-house symbolic regression package that can effectively interface with the rest of our algorithm in PYTHON, and extensions to handle very large [59,60] or high-dimensional [61] datasets.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The authors attest that all data for this study are included in the paper.

Appendix A: Selection of the EGP Parameters

The performance of EGP depends on μ, γ, n', T, b, p, and q. In this section, we specify the parameter choice used for all the examples in Sec. 4 and discuss the rationales behind them so inexperienced users can adapt them to their problems.

We use $\mu = 1.5$ to generate a denser dataset for EP which helps it in finding informative bases. We discourage the use of $\mu \gg 1$ as computational costs would increase, and the symbolic regression could overfit. $\gamma = 0.9$ is used to account for a minimum amount of improvement after each iteration. With a γ too close to one, highly complex bases will be accepted in the DBs repository. With $\gamma < 0.9$, it will be difficult to regress the least relevant terms correctly. Taking this into account, we recommend selecting γ based on the data configuration. In a low noise and dense dataset, where overfitting is not an issue as the noise will be properly filtered by the GP, we advise using a γ closer to 1. Otherwise, the user should be conservative on the selection of γ.

The choice for n' and T is related. We use $n' = \frac{1}{3} \times n$ while bootstrapping to allow for variations to be developed while having a decent number of samples. With very small bootstrapped sets, high variations will be observed across the sets but robustness (numerical stability) decreases. So, we select $T = 10^{-4}$ to avoid any numerical issues while training the GPs. Significantly denser (sparser) datasets than those used in the examples would require a lower (higher) value of n' while T can remain the same. In highly complex data, where linear dependencies between the bases are unlikely, a smaller value for T can be selected.

We use $b = 10$ to assess the sensitivity of the individual CoVs robustly. A higher value of b will be computationally expensive, but we encourage the user to increase it if the time and the computational power are not a constraint. We select $p = 3$ to penalize candidates with unstable bases whose CoV is high. When using a larger value of p, no improvements are usually observed. Finally, with $q = 0.7$, we penalize large, overfitting candidates. On the examples studied with $q \geq 1$, the linear independent bases of the overfitting candidates were usually accepted while with $q < 0.5$ single base candidates, which were not part of the true expression, were accepted. We recommend the inexperienced user to use the suggested defaults.

Appendix B: Symbolic Regression With Evolutionary Gaussian Process

While the goal of EGP is not symbolic regression, it can be used for that purpose. We illustrate this by comparing the discovered symbolic regressions from EGP (final components of the DBs repository) with the bases that Eureqa finds in examples 1, 3, and 7 (see Table 4).

In example 1, the EP emulator (from Eureqa) includes a wide variety of incorrect bases (x_3, x_3^4, x_4) while EGP does not. EGP generally includes the term $\frac{1}{x_4}$ which, while not part of the underlying expression, better captures the trend than x_4 (term usually included by EP). Similar results are observed in examples 2, 4, 5, and 6.

In example 3, the trigonometric functions are intentionally excluded from the primitive pool of functions when the evolutionary search process is conducted. As a result, EP tries to interpolate these trigonometric functions with high degree polynomial terms, which adversely affect extrapolation. EGP, however, detects that these terms are not part of the true underlying analytical function and discards them.

In example 7, the underlying expression is highly complex and the evolutionary search incorrectly finds simple terms. Most of these simple terms are excluded in the results of EGP as they cannot extrapolate. However, EP generally includes many incorrect bases in the final results. Similar results are observed in example 8.

Table 4 ***Regressed terms with EGP for the analytical examples***

Terms / Method	x_1^2	x_1	x_2^2	x_2	$\cos(2x_3)$	$\frac{1}{x_4^2}$	$\frac{1}{x_4^3}$	False terms (percentage of appearance %)
EGP Small noise	100%	100%	100%	100%	100%	40%	0%	x_4 (20%), $\frac{1}{x_4}$ (60%)
EGP Large noise	100%	100%	100%	100%	80%	40%	0%	x_4 (20%), x_4^2 (20%)
EP Small noise	100%	100%	100%	100%	40%	40%	0%	x_3(40%), x_3^2 (40%) x_4 (80%)
EP Large noise	100%	100%	100%	100%	40%	40%	0%	x_3(40%), x_3^2 (40%) x_4 (80%)

Terms / Method	$\sin(x_1)$	$\cos(x_2)$	x_1	x_2^2	False terms (percentage of appearance %)
EGP Small noise	0%	0%	0%	0%	–
EGP Large noise	0%	0%	0%	0%	–
EP Small noise	0%	0%	100%	100%	x_2^6(20%), x_2^4(60%), x_1^3(100%), $\frac{1}{x_2}$(20%), x_2^5(40%), x_2^3(40%), x_2(60%), x_1^2(20%), $x_2x_1^2$(20%), x_2x_1(40%)
EP High noise	0%	0%	100%	100%	x_2^3(40%), $x_1x_2^2$(20%), x_2(40%), x_1^3(40%), $\frac{1}{x_2}$(20%), $\frac{x_2}{x_1}$(20%), x_2^5(40%), $x_1^2x_2$(20%), x_1^2(20%), $\frac{1}{1.901x_1 - 0.389}$(20%), x_2^4(20%), x_2^2(20%), x_1x_2(20%), $x_2x_1^4$(20%)

Terms / Method	a	b	False terms (percentage of appearance %)
EGP Small noise	0%	0%	$x_2x_5^2$(20%), $x_3x_5^2$(20%), $x_7x_5^2$(20%), $x_8x_5^2$(20%)
EGP Large noise	0%	0%	–
EP Small noise	0%	0%	$x_2x_5^2$(40%), $x_8x_2^2x_5^2$(20%), $x_8x_3^2x_5^2$(20%), $x_7x_2^2x_5^2$(40%), $x_2x_5x_8$(40%), $x_2^2x_5^2$(40%), $x_3^2x_7^2$(40%), x_8(40%) $x_5x_7x_3^2$(40%), x_4(20%), sin (126 x_4)(20%), $x_2x_3x_8x_5^2$(20%), x_1x_6(20%)
EP Large noise	0%	0%	$\frac{x_8x_2^4x_5^2}{x_7x_3^3}$(20%), $\frac{1}{x_7x_3^3}$(20%), $x_2x_5x_8$(20%), $x_3^2x_7^2$(20%), x_8(20%), $x_5x_7x_3^2$(20%), x_5(20%), $x_8x_2^2x_5^2$(20%), x_3x_5(20%), $x_2x_5^2$(40%), $x_3x_7x_8x_5^2$(20%), $x_8x_5^2$(40%), $x_7x_5^2$(20%), $x_3x_5^2$(40%), $\frac{x_2^2x_5^2}{x_7}$(20%), $x_2^2x_5^2$(20%)

Note: From top to bottom, the tables show the symbolic regression results for examples 1, 3, and 7. For each true term, the tables include the percentage of appearance of it across the five different fits. The last column shows the regressed terms that are not part of the true underlying function and its percentage of appearance. $a = x_1x_3\left(\log\left(\frac{x_4}{x_5}\right)\left(1 + \frac{x_1}{x_6} + \frac{2x_1x_7}{\log(x_4/x_5)}x_5^2x_8^2\right)\right)^{-1}$, $b = x_1x_3\left(\log\left(\frac{x_4}{x_5}\right)\left(1 + \frac{x_1}{x_6} + \frac{2x_1x_7}{\log(x_4/x_5)}x_5^2x_8^2\right)\right)^{-1}$.

References

[1] Goodfellow, I., Bengio, Y., and Courville, A., 2016, *Deep Learning*, MIT Press, London, UK.
[2] Bostanabad, R., 2020, "Reconstruction of 3d Microstructures From 2d Images Via Transfer Learning," Comput.-Aided Design, **128**(2), p. 102906.
[3] Hassaninia, I., Bostanabad, R., Chen, W., and Mohseni, H., 2017, "Characterization of the Optical Properties of Turbid Media by Supervised Learning of Scattering Patterns," Sci. Rep., **7**(1), p. 15259.
[4] Bostanabad, R., Liang, B., Gao, J. Y., Liu, W. K., Cao, J., Zeng, D., Su, X. M., Xu, H. Y., Li, Y., and Chen, W., 2018, "Uncertainty Quantification in Multiscale Simulation of Woven Fiber Composites," Comput. Methods. Appl. Mech. Eng., **338**(21), pp. 506–532.
[5] Bostanabad, R., Chan, Y. C., Wang, L. W., Zhu, P., and Chen, W., 2019, "Globally Approximate Gaussian Processes for Big Data With Application to Data-Driven Metamaterials Design," ASME J. Mech. Des., **141**(11), p. 111402 (11 pages).
[6] Planas, R., Oune, N., and Bostanabad, R., 2020, "Extrapolation With Gaussian Random Processes and Evolutionary Programming," ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Virtual, Online.
[7] Cressie, N., 1990, "The Origins of Kriging," Math. Geology, **22**(3), pp. 239–252.
[8] Martin, J., and Simpson, T., 2005, "Use of Kriging Models to Approximate Deterministic Computer Models," AIAA. J., **43**(4), pp. 853–863.
[9] Stein, M. L., 2012, *Interpolation of Spatial Data: Some Theory for Kriging*, Springer Science & Business Media, New York.
[10] Bostanabad, R., Liang, B., van Beek, A., Gao, J., Liu, W. K., Cao, J., Zeng, D., Su, X., Xu, H., and Li, Y., 2020, *Multiscale Simulation of Fiber Composites With Spatially Varying Uncertainties*, Y. Wang and D. L. McDowell, eds., Elsevier, Atlanta, GA, pp. 355–384.
[11] Rasmussen, C. E., 2006, *Gaussian Processes for Machine Learning*, The MIT Press, Cambridge, MA.
[12] Awad, M., and Khanna, R., 2015, "Support Vector Regression," *Efficient Learning Machines*, Apress, Berkeley, CA, pp. 67–80.
[13] Chen, T., and Guestrin, C., 2016, "Xgboost: A Scalable Tree Boosting System," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, ACM, pp. 785–794.
[14] Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., and Tibshirani, R., 2009, *The Elements of Statistical Learning*, Vol. 2, Springer, New York, NY.
[15] Bongard, J., and Lipson, H., 2007, "Automated Reverse Engineering of Nonlinear Dynamical Systems," Proc. Natl. Acad. Sci. USA, **104**(24), pp. 9943–8.
[16] Schmidt, M., and Lipson, H., 2009, "Distilling Free-Form Natural Laws From Experimental Data," Science, **324**(5923), pp. 81–5.
[17] Tsoulos, I. G., and Lagaris, I. E., 2006, "Solving Differential Equations With Genetic Programming," Genetic Program. Evolvable Mach., **7**(1), pp. 33–54.
[18] Wang, Z., and Xu, H., 2021, "Quantitative Representation of Aleatoric Uncertainties in Network-Like Topological Structural Systems," ASME J. Mech. Des., **143**(3), p. 031713.
[19] Bae, S., Park, C., and Kim, N. H., 2020, "Estimating Effect of Additional Sample on Uncertainty Reduction in Reliability Analysis Using Gaussian Process," ASME J. Mech. Des., **142**(11), p. 111706.
[20] Oune, N., and Bostanabad, R., 2021, Latent Map Gaussian Processes for Mixed Variable Metamodeling.
[21] Wilson, A. G., and Adams, R. P., 2013, Gaussian Process Kernels for Pattern Discovery and Extrapolation.
[22] Ba, S., and Joseph, V. R., 2012, "Composite Gaussian Process Models for Emulating Expensive Functions," Ann. Appl. Stat., **6**(4), pp. 1838–1860.
[23] Zhang, N., and Apley, D. W., 2014, "Fractional Brownian Fields for Response Surface Metamodeling," J. Q. Tech., **46**(4), pp. 285–301.
[24] Plumlee, M., and Apley, D. W., 2017, "Lifted Brownian Kriging Models," Technometrics, **59**(2), pp. 165–177.
[25] Paulo, R., 2005, "Default Priors for Gaussian Processes," Ann. Stat., **33**(2), pp. 556–582.
[26] Brunton, S. L., Proctor, J. L., and Kutz, J. N., 2016, "Discovering Governing Equations From Data by Sparse Identification of Nonlinear Dynamical Systems," Proc. Natl. Acad. Sci. USA, **113**(15), pp. 3932–7.
[27] Long, Z., Lu, Y., Ma, X., and Dong, B., 2018, "PDE-NET: Learning PDEs from Data," Proceedings of the 35th International Conference on Machine Learning, Stockholmsmässan, Stockholm, Sweden, July 10–15.
[28] Somacal, A., Boechi, L., Jonckheere, M., Lefieux, V., Picard, D., and Smucler, E., 2020, Uncovering Differential Equations from Data With Hidden Variables.
[29] Schaeffer, H., 2017, "Learning Partial Differential Equations Via Data Discovery and Sparse Optimization," Proc. Math. Phys. Eng. Sci., **473**(2197), p. 20160446.
[30] Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N., 2017, "Data-Driven Discovery of Partial Differential Equations," Sci. Adv., **3**(4), p. e1602614.
[31] Martius, G., and Lampert, C. H., 2016, Extrapolation and Learning Equations.
[32] Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Čeperić, V., and Soljačić, M., 2020, "Integration of Neural Network-Based Symbolic Regression in Deep Learning for Scientific Discovery," IEEE Trans. Neural Netw. Learn. Syst., **31**(8), pp. 1–12.
[33] Schmidt, M., and Lipson, H., 2010, *Symbolic Regression of Implicit Equations*, Springer, Ithaca, NY, pp. 73–85.
[34] Chollet, F., 2017, *Deep Learning With Python*, Manning Publications Co, New York, NY.
[35] Bostanabad, R., Kearney, T., Tao, S. Y., Apley, D. W., and Chen, W., 2018, "Leveraging the Nugget Parameter for Efficient Gaussian Process Modeling," Int. J. Numer. Methods Eng., **114**(5), pp. 501–516.
[36] Zhang, W. Z., Bostanabad, R., Liang, B., Su, X. M., Zeng, D., Bessa, M. A., Wang, Y. C., Chen, W., and Cao, J., 2019, "A Numerical Bayesian-Calibrated Characterization Method for Multiscale Prepreg Preforming Simulations With Tension-Shear Coupling," Compos. Sci. Technol., **170**(3), pp. 15–24.
[37] Xu, H., 2020, "Constructing Oscillating Function-Based Covariance Matrix to Allow Negative Correlations in Gaussian Random Field Models for Uncertainty Quantification," ASME J. Mech. Des., **142**(7), p. 074501.
[38] Gramacy, R. B., and Apley, D. W., 2015, "Local Gaussian Process Approximation for Large Computer Experiments," J. Computat. Graphical Stat., **24**(2), pp. 561–578.
[39] MacDonald, B., Ranjan, P., and Chipman, H., 2015, "GPfit: An R Package for Fitting a Gaussian Process Model to Deterministic Simulator Outputs," J. Stat. Soft., **64**(12), pp. 1–23.
[40] Ranjan, P., Haynes, R., and Karsten, R., 2011, "A Computationally Stable Approach to Gaussian Process Interpolation of Deterministic Computer Simulation Data," Technometrics, **53**(4), pp. 366–378.
[41] Sacks, J., Schiller, S. B., and Welch, W. J., 1989, "Designs for Computer Experiments," Technometrics, **31**(1), pp. 41–47.
[42] Toal, D., Bressloff, N., and Keane, A., 2008, "Kriging Hyperparameter Tuning Strategies," AIAA. J., **46**(5), pp. 1240–1252.
[43] Audet, C., and Dennis Jr, J. E., 2002, "Analysis of Generalized Pattern Searches," SIAM J. Optim., **13**(3), pp. 889–903.
[44] Zhao, L., Choi, K. K., and Lee, I., 2011, "Metamodeling Method Using Dynamic Kriging for Design Optimization," AIAA. J., **49**(9), pp. 2034–2046.
[45] Toal, D., Bressloff, N. W., Keane, A. J., and Holden, C. M. E., 2011, "The Development of a Hybridized Particle Swarm for Kriging Hyperparameter Tuning. Engineering Optimization," Eng. Optim., **43**(6), pp. 675–699.
[46] Tao, S., Shintani, K., Bostanabad, R., Chan, Y.-C., Yang, G., Meingast, H., and Chen, W., 2017, "Enhanced Gaussian Process Metamodeling and Collaborative Optimization for Vehicle Suspension Design Optimization," ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Cleveland, OH.
[47] Koza, J. R., 1994, "Genetic Programming As a Means for Programming Computers by Natural-selection," Stat. Comput., **4**(2), pp. 87–112.
[48] Fortin, F. A., De Rainville, F. M., Gardner, M. A., Parizeau, M., and Gagne, C., 2012, "Deap: Evolutionary Algorithms Made Easy," J. Mach. Learn. Res., **13**(Jul.), pp. 2171–2175.
[49] Koller, D., and Friedman, N., 2009, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, London, UK.
[50] Izzo, D., Biscani, F., and Mereta, A., 2017, *Differentiable Genetic Programming*, J. McDermott, M. Castelli, L. Sekanina, I. Wolf, and P. García-Sánchez, eds., Springer, Amsterdam, The Netherlands, pp. 35–51.
[51] Xiong, J., Shi, S.-Q., and Zhang, T.-Y., 2020, "A Machine-Learning Approach to Predicting and Understanding the Properties of Amorphous Metallic Alloys," Mater. Des., **187**(6), p. 108378.
[52] Guyon, I., and Elisseeff, A., 2003, "An Introduction to Variable and Feature Selection," J. Mach. Learning Res., **3**(Mar.), pp. 1157–1182.
[53] Sobol', I. M., 1967, "On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals," Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, **7**(4), pp. 784–802.
[54] Sobol, I. M., 1998, "On Quasi-monte Carlo Integrations," Math. Comput. Simul., **47**(2–5), pp. 103–112.
[55] Arruda, E. M., and Boyce, M. C., 1993, "A Three-Dimensional Constitutive Model for the Large Stretch Behavior of Rubber Elastic Materials," J. Mech. Phys. Solids., **41**(2), pp. 389–412.
[56] Bostanabad, R., Chen, W., and Apley, D. W., 2016, "Characterization and Reconstruction of 3d Stochastic Microstructures Via Supervised Learning," J. Microsc., **264**(3), pp. 282–297.
[57] Bessa, M. A., Bostanabad, R., Liu, Z., Hu, A., Apley, D. W., Brinson, C., Chen, W., and Liu, W. K., 2017, "A Framework for Data-Driven Analysis of Materials Under Uncertainty: Countering the Curse of Dimensionality," Comput. Methods. Appl. Mech. Eng., **320**, pp. 633–667.
[58] Belytschko, T., Liu, W. K., Moran, B., and Elkhodary, K., 2013, *Nonlinear Finite Elements for Continua and Structures*, John Wiley & sons.
[59] Friedman, J. H., and Stuetzle, W., 1981, "Projection Pursuit Regression," J. Am. Stat. Assoc., **76**(376), pp. 817–823.
[60] Friedman, J. H., 1991, "Multivariate Adaptive Regression Splines," Ann. Stat., **19**(1), pp. 1–67.
[61] Shan, S., and Wang, G. G., 2010, "Metamodeling for High Dimensional Simulation-Based Design Problems," ASME J. Mech. Des., **132**(5), p. 051009.