Optimally Emulating Smooth Functions for Bayesian Analyses

Scott Pratt, Oleh Savchuk, Eren Erdogan, Ekaksh Kataria
Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory Michigan State University, East Lansing, MI 48824 USA

Christal Martin
Department of Physics, University of Tennessee, Knoxville, TN 37996

Syed Afrid Jahan
Department of Physics and Astronomy, Wayne State University, Detroit, MI 48202

Frederi Viens
Department of Statistics, Rice University, Houston, TX 77005

(Dated: October 15, 2025)

For Bayesian analyses aimed at constraining the parameter space of computationally expensive models by comparison with heterogeneous data sets, model emulators play a pivotal role. Model emulators enable searches through high-dimensional parameter spaces, which might otherwise require millions of full model runs. Here, an emulation strategy is presented based on the assumption that the model output can be well described by an analytic polynomial expansion with all terms being independent of one another and with higher order terms being given decreasing weights according to a convergence parameter. It is found that if the expansion's form is constrained to being invariant to rotation and translation of the parameter space that the polynomial form becomes equivalent to that of a Gaussian process emulator using a Gaussian correlation kernel. In the limit that the function is smooth, quantified by a convergence parameter, Λ, it is found that rather few training points are required to reproduce the full model and provide a good estimate of its uncertainty. An accuracy metric is defined based on an average of the expected uncertainty over the prior before training data are actually gathered. Analytic expressions for the expected accuracy are provided based solely on an expected value of Λ and the location of the training points. This enables a tenable optimized placement of training points, which is shown to be significantly superior to random placement or that from Latin hyper-cube sampling. Examples of optimized placement are provided, and the accuracy is studied as a function of various variables, such as the number of model parameters, the number of training points, and the convergence parameter.

I. INTRODUCTION

In the quest to provide rigorous quantitative constraints of high-dimension model-parameter spaces by comparing sophisticated models to experimental results, one must sample the model-parameter space by running the full-model at large numbers of points. For example, sampling a 12-parameter model with only three points in each direction requires $3^{12} \approx 500,000$ runs. Thus, Markov Chain Monte Carlo (MCMC) searches, weighted by the likelihood that the model reproduces the experiment, often involve millions of full-model evaluations. If a single run of the full-model at one point in parameter space requires days of CPU time, the analysis becomes untenable. Model emulation provides a solution to this problem. If one can train an emulator to reproduce the full-model by training it on only a few hundred full-model runs or fewer, and if the emulator can provide a reasonable estimate of its uncertainty, the emulator can take the place of the full model in the MCMC search, and thus enable the analysis.

Most of the authors of this paper are practitioners in the field of heavy ion physics. In this field, emulators have been applied to several large-scale analyses, often involving more than a dozen model parameters with models that often require days of CPU time [1–43]. For example, in a global analysis of 14 model parameters [34], Gaussian-process (GP) emulators [44, 45] were able to emulate output from hybrid hydrodynamic/Boltzmann approaches with accuracies near one percent after being trained on 1200 points. However, one might ask whether one can do as well or better with fewer points. For models in this field the relation between the model parameters and the output tends to be approximately linear. For example, in high-energy Pb-Pb collisions at the LHC, the mean momentum of outgoing protons rises monotonically with the stiffness of the equation of state. That dependence should not be perfectly linear, but one would expect that within a reasonable prior that the dependence should be well described by linear or quadratic functions, and that higher-order terms should become increasingly negligible. This expectation should also apply to most any other observable with respect to nearly any other model parameter. For lower-energy collisions, where phase transitions are perhaps more relevant, one might imagine a dependence that has significant structure, e.g. a clear threshold w.r.t. some parameter, but certainly not at LHC energies.

If one were to expect that the model, with $N_{\rm pars}$ parameters, was nearly perfectly linear, a first-order polynomial would provide an excellent fit. One could then run the full model only $N_{\rm pars}+1$ times at points fairly far apart to constrain the $N_{\rm pars}$ slopes and the intercept. For a 12-parameter model, this would require only 13 full model runs. If one thought that a quadratic fit would well describe the data, one would need only $N_{\rm pars}(N_{\rm pars}+1)/2$ additional full-model runs to constrain all the quadratic contributions. For the 12-parameter case, this would be 91 full model runs, significantly fewer than the number of full-model runs often used in recent analyses.

Given the rather continuous and smooth response of most models applied to heavy-ion physics, it seems prudent to revisit such fits, and investigate the degree to which they might, or might not, improve upon the GP approaches cited above. The first goal of this paper is to exploit this expectation of convergence systematically by presenting an expansion where the importance of various terms in the polynomial decreases with increasing order. Once such an expectation is quantified in terms of a convergence parameter, Λ, one can find the optimum set of expansion coefficients along with their uncertainty. Here, the emulator is built on a polynomial expansion with the terms contributing as θ/Λ, where the model parameters $\vec{\theta}$ are all scaled so that they vary of order unity. Thus, higher-order terms contribute inversely with the convergence parameter Λ.

Polynomial fits based on chi-square minimization algorithms might seem like a good candidate for such problems. More modern approaches improve on such fits by providing estimates that exactly reproduce the training values when evaluated at the training points, and provide an estimate of the emulator's uncertainty as a function of $\vec{\theta}$. These approaches typically are based on GP assumptions, where the predictions are all based on an assumed form of the correlation between values Y_1 and Y_2 of the model evaluated at two points $\vec{\theta}_1$ and $\vec{\theta}_2$. That correlation kernel might be any function of $\vec{\theta}_1$ and $\vec{\theta}_2$, but a particularly simple and popular form is a Gaussian, $\langle Y_1 Y_2\rangle_0 = \sigma_A^2 e^{-|\vec{\theta}_1-\vec{\theta}_2|^2/2\Lambda^2}$. Here, the parameters σ_A and Λ are referred to as hyper parameters, to differentiate them from model parameters. They apply only to the emulator and can be optimized based on training data. The averaging $\langle\cdots\rangle_0$ refers to the averaging over all possible functions with those hyper parameters. It has been shown that polynomial expansions, or any expansion of some basis functions, where the terms have random coefficients can be optimized to reproduce training points. These can then be equated to GP forms once one defines the appropriate correlation function [44]. Previous approaches have typically been constructed from sets of orthogonal basis functions, e.g. [46], but this requirement is not necessary and is not enforced here. The sum over polynomial terms with random coefficients is sometimes referred to as "polynomial chaos" [47]. Once trained, one can solve for all the optimized coefficients for the polynomial expansion. One can then calculate the emulator values at any $\vec{\theta}$ using those optimum coefficients. However, there are many more coefficients than training constraints, and if one sums to all orders, the number of coefficients is infinite. Fortunately, once one has written an equivalent form for the correlations, the emulator can much more efficiently be expressed in terms of the correlation kernel and one can avoid performing sums over all polynomial terms. This is sometimes referred to as the *kernel trick* [44].

In this work, we consider an expansion of polynomials with random coefficients, and then constrain the form so that the behavior is independent of rotation and translation. We find that the equivalent correlation kernel must then have a simple Gaussian form. Thus, if one believes that the model is equally sensitive to all parameters and that there is no reason to expand the polynomials about one point than another point, then the equivalent GP correlation kernel must be Gaussian, and described by the two hyper-parameters σ_A and Λ mentioned above. If the user believes that the sensitivity w.r.t. some model parameter should be less than that of another parameter, rather than applying different convergence parameters, one can scale the parameters so that the expected sensitivity is isotropic in the scaled parameter basis. Thus, the assumptions of the behavior being independent to rotations or translations can reasonably apply to many problems.

The convergence parameter, Λ, which sets the relative importance of terms in the polynomial expansion of different order, is found to be precisely the same as the correlation length in the GP case with Gaussian correlation kernels. For the polynomial expansion to converge within a few orders, one must have the convergence parameter Λ larger than the characteristic scale of the prior. If the model parameters are scaled to vary roughly between ± 1, then Λ should be significantly larger than unity if the expansion is to be well defined by a few terms, or equivalently, to be constrained by a few training points. After equating to the GP paradigm, this is equivalently stated as having the GP correlation length, also Λ, being larger than the size of the prior.

Conversely, if one assumes a Gaussian correlation kernel, the relations derived here identify the form of the corresponding polynomial expansion. Thus, one can use this for to generate random functions that are consistent with the Gaussian kernel, albeit with some cutoff in the order of the expansion. This is useful for testing emulation software, as it allows one to build models that correspond to the desired correlation kernel and should, by construction, have uncertainties that are understood.

Our second thrust is to understand how in the limit of rapidly converging expansions one should optimally choose the points at which the full-model is run to train the emulator. A metric is presented that represents the potential accuracy of the emulator as a function of the convergence parameter, Λ, and the positions of the training points. For an assumed Λ, optimized arrangements of training points are then generated for various cases involving either Gaussian or uniform priors. For test cases, this procedure is found to be significantly superior to either random placement of the training points throughout the prior, or to Latin hyper-cube sampling [48]. In this limit of rapidly converging expansions, one might expect that the number of training points might be chosen to match that needed for linear, quadratic or cubic fits. For the 6-parameter example, that would be 7, 28 or 84 full-model runs respectively. Here, we show that the marginal improvement to the emulation decreases at each of these thresholds, assuming that the training points are optimally chosen. The optimization technique presented here is found to provide a significant improvement relative to random placement, or to Latin hyper-cube sampling.

By accomplishing the goals above, a user should be able to gain a better understanding of how many model runs might be necessary to design an emulator to the desired accuracy. This will still depend on the some pre-estimate of the convergence parameter, but the actual placement of the training points is not strongly influenced by that estimate, even though the accuracy is strongly sensitive to the estimate. If the user knows ahead of time that some of the model parameters are more likely to drive the functional behavior than others, a seamless method to exploit that expectation is presented. This assists with overcoming the challenges of high dimensionality, which are found to be critical in determining how confidently one might ultimately emulate models.

The next section presents the mathematical form of such an emulator and a derivation of the optimum set of coefficients given training data. Closed expressions for the emulator uncertainty are also found. The form also accounts for noisy models, i.e. those with point-by-point uncertainty. The physical arguments for choosing the specific form are discussed. The equivalence with a GP emulator is demonstrated. The subsequent section presents a tractable method optimizing the location of training points in the $N_{\rm pars}-$dimensional parameter space. For the case of Gaussian priors with $N_{\rm pars}+1$ training points, the placement follows a geometric $N_{\rm pars}-$dimensional simplex, but for most cases the optimized solution does not have a simple symmetry and is found numerically. Section IV tests the procedure by applying to some models with random polynomial expansions or randomized trigonometric or exponential forms to gain a feel for how well one might not only emulate a function, but also to understand how well one might trust the statement of the emulator's uncertainty. The faithfulness will be studied as a function of the dimensionality of the model space and of the convergence parameter.

II. THEORETICAL FOUNDATION OF EMULATING CONVERGENT FUNCTIONS

Here, a detailed presentation of the form and properties of an emulator is presented. The form is that of a polynomial expansion in terms of the model parameters, where the coefficient for each term in the polynomial is given an independent Gaussian weight. The form assumes that terms of higher order should contribute less. For example, in the absence of training data, quadratic terms should be expected to contribute less than linear terms, and cubic terms should contribute less than quadratic terms. What separates the form presented here compared to previous studies is that the expansion is constrained to behave independently of rotation and translation. Thus, moving the origin about which the polynomial expansion is constructed, or rotating the coordinate system about any such origin, should not affect the emulator. In fact, in the next subsection it is shown that this constraint forces a unique form to the emulator's polynomial expansion. That form is defined by two parameters, σ_A, which sets the strength of the fluctuation of the pre-trained emulator at any point, and Λ, which drives the convergence of the polynomial expansion, which expands in terms of powers of θ/Λ. It is well known that expansions, including polynomial expansions, that are linear in random coefficients, lead to emulators whose behavior can easily be expressed in terms of the pre-trained correlation kernel, $C_0(\vec{\theta}_1, \vec{\theta}_2)$. Both the emulator and its error are easily expressed in terms C_0, which allows one to actually avoid calculations where one must sum over the actual terms of the polynomial expansion. This result is known as the *kernel trick* [44]. What is new here, is that given the assumption that each term in the polynomial expansion is independent, the constraints of rotational and translational invariance lead to a unique form for the expansion, and thus to a unique form for the kernel. The unique form for the kernel is found to be that of a simple Gaussian, with Λ playing the role of the correlation length.

Throughout this section the model parameters, $\vec{\theta}$, are assumed to have been translated, rotated and scaled so that their expected sensitivities to a given step size $\Delta\Theta$ are similar. For example, if a model parameter x_i, has an initial prior, $a_i < x_i < b_i$, one defines a new variable

$$\theta_i = \frac{2\theta_{\max,i}}{(b_i - a_i)}(x_i - (a_i + b_i)/2). \tag{1}$$

The priors are then centered at zero and extend a range defined by $\pm\theta_{\max,i}$. One would define $\theta_{\max,i}$ for each i so that one would not expect the observable, y, to be any more sensitive to steps of size $\Delta\Theta$ in the direction of θ_i than to steps of the same size in the direction of θ_j. Thus, if the sensitivity to θ_1 was expected to be higher than the sensitivity to θ_2, one would reduce the ratio of $\theta_{\max,2}$ to $\theta_{\max,1}$ until the expected sensitivities were similar. Further, if one expected the $y(\vec{\theta})$ to be largely sensitive to a given direction in parameter space, e.g., $\theta_1 + \theta_2$, compared to some orthogonal direction, $\theta_1 - \theta_2$, one could rotate the coordinate system by 45 degrees, then scale the two dimensions accordingly. More generally, one could define a matrix, S_{ij}, describing th expected variance of y with respect to $\Delta\theta_i\Delta\theta_j$, then redefine a coordinate system where the new parameters were eigenvalues of S, then scale the new parameters so that the eigenvalues of S were similar. One would then perform the model emulation in this new basis where the functions was not expected to vary more in any specific direction. For the case where one had no expectation that the model were more likely to vary across the prior in one direction than in another, all the values of $\theta_{\max,i}$ would all be set to unity.

For uniform priors, it will be assumed that the parameter θ_i will be distributed according to

$$P_i(\theta_i) = \begin{cases} \frac{1}{2\theta_{\max,i}}, & -\theta_{\max,i} < \theta_i < \theta_{\max,i}, \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

As discussed in the previous paragraph, if all parameters are considered to be of equal importance, $\theta_{\max,i}$ is set to unity for each parameter. If the function is expected vary most with a specific parameters, that parameter is scaled with $\theta_{\max,i} = 1$, and other parameters are scaled with smaller values of $\theta_{\max,i}$.

If the parameters have Gaussian priors the parameters can be rotated, scaled and translated so that the Gaussian priors have the form

$$P_i(\theta_i) = \frac{1}{(2\pi/\beta_i)^{1/2}} e^{-\beta_i\theta_i^2/2}. \tag{3}$$

Again, if all the parameters are put on equal footing, the value of β_i is set to three. With this choice the Gaussian and uniform priors have the same variance. If some given parameter is expected to drive most of the variability that parameter is assigned $\beta_i = 3$ and other parameters are assigned lower values of β_i.

The polynomials will be expanded in term of θ_i/Λ, where Λ is the convergence parameter. Thus, higher order polynomial terms are assumed to be less important in describing the function. As discussed above, it may be the case that the function to be emulated is known to be less sensitive to some model parameters than it is to others. Rather than assigning different Λ for different model parameters to reflect the difference in sensitivities, the prior scales $\theta_{\max,i}$ and $\beta_i^{-1/2}$ are reduced from the values of unity and $1/\sqrt{3}$. This choice makes the mathematical description of the emulator simpler and maintains rotational invariance, while not forfeiting the ability to assume some parameters contribute in a more convergent manner than others. Because the emulator is a function of θ_i/Λ, scaling θ_i is mathematically identical to scaling Λ. One can assume different prior forms, e.g. half Gaussians, and it does not affect any of the derivations or lessons from this section. However, the choice of the prior is critical in Sec. III, where methods to optimize the training-point placement are considered.

This section includes several subsections briefly reviewing the methods needed to tune and train the emulator. Training is accomplished by calculating the full function $y(\vec{\theta})$ at $N_{\rm train}$ locations in model-parameter space. The training data is thus comprised of knowing $y_a = y(\vec{\theta}_a)$ for $a = 1 \cdots N_{\rm train}$. Expressions for the optimized coefficients are found in terms of the training data. After applying the kernel trick, generating these coefficients is mostly unnecessary given that the emulator and its uncertainty can readily be expressed in terms of the training data and the correlation kernel, without ever performing sums over all polynomial terms, of which there are an infinite number. Given that the resulting emulator is identical to one used for many GP treatments, this review of training and tuning might seem somewhat unnecessary. Nonetheless, it is insightful to understand what set of basis functions results in a GP emulator with a Gaussian correlation kernel.

One part of the review of training and tuning which does not appear in many GP treatments is the inclusion of the emulator uncertainty due to the uncertainty in choosing the hyper parameter Λ. Although this term is not usually significant once one has a reasonable distribution of training points, it plays an important role in optimizing training-point placement, a subject considered in detail in Sec. III.

A. General Form and the Assumption of Analyticity

The $N_{\rm pars}$-dimensional model parameters can be considered a vector $\vec{\theta}$. It is assumed that the original model parameters have been shifted and scaled so that the prior ranges are defined as described above. The first assumption for choosing the functional form is that it is analytic, so that it can be expanded as a polynomial in the range of the prior. Here, we assume the following form for the emulator, $Y(\vec{\theta})$,

$$Y(\vec{\theta}) = \sum_{\vec{n}} d_{\vec{n}}(|\vec{\theta}|^2) f_{K(\vec{n})}(|\vec{\theta}|^2) A_{\vec{n}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_{N_{\rm pars}}}. \tag{4}$$

Throughout the paper, the uppercase function $Y(\vec{\theta})$ will refer to the emulator whereas the lowercase function $y(\vec{\theta})$ refers to the evaluation of the full model. Here, $K(\vec{n})$ is the order of the polynomial, $K = n_1 + n_2 + \cdots n_{N_{\rm pars}}$. The vector $\vec{n}$ refers to the $N_{\rm pars}$-dimensional vector comprised of non-negative integers, $(n_1, n_2 \cdots)$.

Because the coefficients $A_{\vec{n}}$ are considered to be independent in the absence of training data, each polynomial term can be considered as a basis function. This set of basis functions is clearly non-orthogonal. The normalization and orthogonality properties are discussed in Sec. A. The form appears rather general, in that all terms for a given K are independent. To be analytic, the functions f_K and $d_{\vec{n}}$ must be polynomial expansions of the norm $|\vec{\theta}|^2$. Thus, the current form completely encapsulates all possible polynomials of analytic functions, i.e. functions that can be written in terms of polynomials of $\vec{\theta}$. For example, this precludes terms such as $\theta_1/|\vec{\theta}|$.

The second assumption is that we assume that the independent Gaussian coefficients $A_{\vec{n}}$ all have the same Gaussian weights and are independent of one another,

$$\begin{aligned} W(A_{\vec{n}}) &= \frac{1}{\sqrt{2\pi\sigma_A^2}} e^{-A_{\vec{n}}^2/2\sigma_A^2}, \\ \langle A_{\vec{m}} A_{\vec{n}} \rangle_0 &= \sigma_A^2 \delta_{\vec{m},\vec{n}}. \end{aligned} \tag{5}$$

The averaging $\langle ... \rangle_0$ refers to the averaging without any constraints from matching training data. The particular choice of a Gaussian weight does not affect any of the derivations of this section unless one is considering correlations to higher order than quadratic, e.g a correlation of order E^4 such as $\langle Y(\vec{\theta}_a) Y(\vec{\theta}_b) Y(\vec{\theta}_c) Y(\vec{\theta}_d) \rangle_0$. In Sec. II C it will be shown that $W(A_{\vec{n}})$ must have a Gaussian distribution if the higher-order correlations are to all be independent of translation and rotation. Any distribution that averages to zero and has some variance σ_A^2 would give the same behavior for correlations involving only quadratic terms of E. The choice that each coefficient's distribution is characterized by the same σ_A does not reduce the generality because $d_{\vec{n}}$ is still arbitrary at this point. However, the choice that these coefficients are all independent of one another does reduce the generality of the expression by eliminating the possibility that different terms are correlated. The fact that $d_{\vec{n}}$ can be any analytic function of $|\vec{\theta}|^2$ does represent allowing correlations between polynomials of different order that differ only by powers of $|\vec{\theta}|^2$, but correlations between terms that have different angular behavior are forbidden. Some other treatments might instead expand in a basis of orthonormal basis functions, e.g. [46]. The basis functions are effectively the functions which are multiplied by the random coefficients to construct the emulator. Those functions here are simple products of various parameter components multiplied by functions of $|\vec{\theta}|^2$, and are clearly not orthonormal. However, even with orthonormal terms one would be making some assumption that the terms are independent. It is not clear that orthonormality plays an important role in this application, but it is important to understand completeness. The basis functions will become better defined below. The appendix, Sec. A, reviews the orthonormality and completeness properties of the basis functions used here.

B. Rotational and Translational Invariance

The third assumption will be that the fluctuation at some point $\vec{\theta}$ is independent of the direction of $\vec{\theta}$. The fluctuation is

$$\langle Y^2(\vec{\theta})\rangle_0 = \sum_{\vec{n}} f_K^2(|\vec{\theta}|^2)\sigma_A^2 d_{\vec{n}}^2 \left(\frac{\theta_1^{2n_1}}{\Lambda^2}\right)\left(\frac{\theta_2^{2n_2}}{\Lambda^2}\right)\cdots\left(\frac{\theta_N^{2n_N}}{\Lambda^2}\right). \tag{6}$$

To be independent of direction, for a fixed norm the θ dependence must be proportional to $|\vec{\theta}|^{2K}$. Using the fact that

$$|\vec{\theta}|^{2K} = \sum_{\vec{n},s.t.\ \sum_i n_i=K} \frac{K!}{n_1!\cdots n_N!}(\theta_1^2)^{n_1}(\theta_2^2)^{n_2}\cdots(\theta_{N\text{pars}}^2)^{n_{N\text{pars}}}, \tag{7}$$

one can recognize that

$$d_{\vec{n}}^2(|\vec{\theta}|^2) \propto \frac{1}{n_1!n_2!\cdots n_{N_\text{pars}}!}. \tag{8}$$

It can also be some arbitrary function of K and $|\vec{\theta}|^2$, but that arbitrariness can be captured by $f_K(|\vec{\theta}|^2)$. Thus, with complete generality, once the independence of direction is enforced one can set

$$d_{\vec{n}}(|\vec{\theta}|^2) = \frac{1}{\sqrt{n_1!n_2!\cdots n_{N_\text{pars}}!}}. \tag{9}$$

The number of ways one can arrange the K elements for a given $\vec{n}$ is $K!d_{\vec{n}}^2$. The general form for the emulator, given the constraint that it is agnostic in regards to rotation in $\vec{\theta}$ space, is now

$$Y(\vec{\theta}) = \sum_{\vec{n}} f_{K(\vec{n})}(|\vec{\theta}|^2)\frac{A_{\vec{n}}}{\sqrt{n_1!n_2!\cdots n_{N\text{pars}}!}}\left(\frac{\theta_1}{\Lambda}\right)^{n_1}\left(\frac{\theta_2}{\Lambda}\right)^{n_2}\cdots\left(\frac{\theta_N}{\Lambda}\right)^{n_{N\text{pars}}}. \tag{10}$$

The fourth assumption is that the correlation for two different points should depend only on $|\vec{\theta}_1-\vec{\theta}_2|^2$. This will fix the form for $f_K(|\vec{\theta}|^2)$. The correlation is

$$\begin{aligned}\langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)\rangle_0 &= \sigma_A^2\sum_{\vec{n}}\frac{1}{\Lambda^{2K}}\frac{f_K(|\vec{\theta}_a|^2)f_K(|\vec{\theta}_b|^2)}{n_1!n_2!\cdots n_{N\text{pars}}!}(\theta_{a,1}\theta_{b,1})^{n_1}(\theta_{a,2}\theta_{b,2})^{n_2}\cdots(\theta_{a,N_\text{pars}}\theta_{b,N_\text{pars}})^{n_{N\text{pars}}} \\ &= \sigma_A^2\sum_{\vec{n}}\frac{f_K(|\vec{\theta}_a|^2)f_K(|\vec{\theta}_b)|^2}{K!}(\vec{\theta}_a\cdot\vec{\theta}_b/\Lambda^2)^K.\end{aligned} \tag{11}$$

The last step used the identity,

$$|\vec{\theta}_a\cdot\vec{\theta}_b|^K = \sum_{\vec{n},s.t.\ \sum_i n_i=K}\frac{K!}{n_1!\cdots n_N!}(\theta_{a,1}\theta_{b,1})^{n_1}(\theta_{a,2}\theta_{b,2})^{n_2}\cdots(\theta_{a,N\text{pars}}\theta_{b,N\text{pars}})^{n_{N\text{pars}}}, \tag{12}$$

If one were to choose

$$f_K(|\vec{\theta}|^2) = e^{-|\vec{\theta}|^2/2\Lambda^2}, \tag{13}$$

the resulting correlation would be

$$\langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)\rangle_0 = \sigma_A^2 e^{-|\vec{\theta}_a-\vec{\theta}_b|^2/2\Lambda^2}, \tag{14}$$

which would satisfy the constraint of being a function of $\vec{\theta}_a-\vec{\theta}_b$. But, one can go further and show that this is the only such solution. To prove this, one can write

$$f_K(|\vec{\theta}|^2) = e^{-\frac{|\vec{\theta}|^2}{2\Lambda^2}} + \delta f_K(|\vec{\theta}|^2). \tag{15}$$

Further, one can expand δf_K,

$$\delta f_K(|\vec{\theta}|^2) = \alpha_{K,2}|\vec{\theta}|^4 + \alpha_{K,3}|\vec{\theta}|^6 + \alpha_{K,4}|\vec{\theta}|^8 + \cdots \tag{16}$$

The series begins for quartic terms because adding a quadratic term, beyond what is in the Gaussian in Eq. (15), would violate the constraint that overall the quadratic term in Eq. (11) must behave as $|\vec{\theta}_a - \vec{\theta}_b|^2$. The corrections due to δf_K of quartic behavior are

$$\delta\langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)\rangle_0 = \alpha_{0,2}|\vec{\theta}_a|^4 + \alpha_{0,2}|\vec{\theta}_b|^4 + \alpha_{2,0}(\vec{\theta}_a \cdot \vec{\theta}_b)^2. \tag{17}$$

One can set $\alpha_{0,2} = -\alpha_{2,0}/2$ to make this disappear when $\vec{\theta}_1 = \vec{\theta}_2$, but this correction is not proportional to $|\vec{\theta}_1 - \vec{\theta}_2|^4$. The same can be said for higher-order terms in θ. Thus, δf must be zero and the form for $f_K(|\vec{\theta}|^2)$ is completely determined to match that of Eq. (15) if one insists that the correlation be a function of $|\vec{\theta}_1 - \vec{\theta}_2|^2$.

After enforcing the four criteria, the form for the emulator, correlation and fluctuations are now fixed to

$$\begin{aligned} Y(\vec{\theta}) &= e^{-|\vec{\theta}|^2/2\Lambda^2} \sum_{\vec{n}} \frac{A_{\vec{n}}}{\sqrt{n_1! n_2! \cdots n_{N\text{pars}}!}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_{N\text{pars}}}, \\ \langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)\rangle_0 &= \sigma_A^2 e^{-|\vec{\theta}_a - \vec{\theta}_b|^2/2\Lambda^2}, \\ \langle Y^2(\vec{\theta})\rangle_0 &= \sigma_A^2. \end{aligned} \tag{18}$$

In fact, for the four criteria mentioned above, satisfying the third criteria is automatically satisfied if the fourth is satisfied, because the fluctuation is simply the correlation evaluated at $\vec{\theta}_1 = \vec{\theta}_2$. Thus, only three criteria are required to constrain the form to that of Eq. (18). The form for the emulator, for fluctuations and for correlations is fixed to that of Eq. (18) if the following choices or assumptions are made:

1. The function should be analytic throughout the prior.
2. At any point the function should fluctuate according to a Gaussian distribution.
3. The prior-to-constraint correlation, $\langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)\rangle_0$, should depend only on $|\vec{\theta}_1 - \vec{\theta}_2|^2$.

From here on, the correlation will be referred through the function $C_0(\vec{\theta}_a, \vec{\theta}_b)$, defined as

$$C_0(\vec{\theta}_a, \vec{\theta}_b) \equiv \frac{1}{\sigma_A^2}\langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)\rangle_0 = e^{-|\vec{\theta}_a - \vec{\theta}_b|^2/2\Lambda^2}. \tag{19}$$

The subscript zero again refers to the fact that this averaging is performed before one has applied any constraints from the training data.

C. Higher Moments and the Assumption that Coefficients have Gaussian Distributions

One might ask whether all other moments, e.g. $\langle Y^4(\vec{\theta})\rangle_0$, are also independent of $\vec{\theta}$. The answer is yes, and derives from the initial choice that the weights for the coefficients are Gaussian. For Gaussians centered at zero, all cumulants are zero except for the second cumulant, which is σ_A^2. Thus, when averaging over the initial coefficients, the averaged fourth moment of the coefficients becomes

$$\langle A_i A_j A_m A_n\rangle_0 = \langle A_i A_j\rangle_0 \langle A_m A_n\rangle_0 + \langle A_i A_m\rangle_0 \langle A_j A_n\rangle_0 + \langle A_i A_n\rangle_0 \langle A_j A_m\rangle_0. \tag{20}$$

For any distribution where $\langle A_i\rangle_0 = 0$, the independent coefficients, Eq. (20) would be zero if the indices were not all the same, e.g. if $i = j$ and $m = n$, but $i \neq m$. Gaussian distributions are special in that this holds true even if $i = j = m = n$. If one expands $Y(\vec{\theta})$ in terms of the coefficients and polynomials, one readily sees that this implies that

$$\begin{aligned} \langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)Y(\vec{\theta}_c)Y(\vec{\theta}_c)\rangle_0 &= \langle Y(\vec{\theta}_a)Y(\vec{\theta}_b)\rangle_0 \langle Y(\vec{\theta}_c)Y(\vec{\theta}_d)\rangle_0 \\ &+ \langle Y(\vec{\theta}_a)Y(\vec{\theta}_c)\rangle_0 \langle Y(\vec{\theta}_b)Y(\vec{\theta}_d)\rangle_0 + \langle Y(\vec{\theta}_a)Y(\vec{\theta}_d)\rangle_0 \langle Y(\vec{\theta}_b)Y(\vec{\theta}_c)\rangle_0. \end{aligned} \tag{21}$$

All correlations and fluctuations to order E^4 can therefore be expressed in terms of correlations of Y^2. If the correlations of order Y^2 are independent of translations and rotations, the correlations of order Y^4 must then also satisfy this requirement. If all the coordinates of a correlation are rotated or translated, the correlation is unchanged. For the particular case where one considers all four angles to be equal to one another this gives

$$\langle Y^4(\vec{\theta})\rangle_0 = 3\sigma_A^4, \tag{22}$$

which is independent of $\vec{\theta}$. In fact, because all cumulants disappear, correlations to any order, $\langle Y(\vec{\theta}_1 \cdots Y(\vec{\theta}_n)\rangle_0$, can be expressed in terms of $\langle Y(\vec{\theta}_1)Y(\vec{\theta}_2)\rangle_0 = \sigma_A^2 e^{-|\vec{\theta}_1-\vec{\theta}_2|^2/2\Lambda^2}$.

If one replaces the choice of Gaussian distributions for the coefficients with another distribution with the same second moments, with all the odd moments being zero, but with different cumulants for the higher even moments, the property that the correlations will only depend on relative correlations will be broken. For example, if one considers the fourth moment,

$$\langle A_i^4\rangle_0 = 3\sigma_A^4 + \kappa. \tag{23}$$

Here, $\kappa = 0$ for a Gaussian distribution. If one were to calculate the fourth moment of E with all four positions being equal, and with $\vec{\theta}$ being along the first axis, $\vec{\theta} = [\theta_1, 0, 0\cdots]$,

$$\langle Y(\vec{\theta})^4\rangle_0 = 3\sigma_A^2 + \kappa e^{-2\theta_1^2|^2/\Lambda^2} \sum_n \frac{1}{(n!)^2}\theta_1^{4n}. \tag{24}$$

Clearly, this is not independent of θ_1, and is clearly not a function of relative coordinates, as all the relative coordinates are zero. The correlations are then clearly not invariant under translation and rotation. Thus, if one wishes all the higher-order correlations to depend only on relative coordinates and be independent of translation and rotation, the distribution for the coefficients must have zero cumulants. The distribution $W(A_{\vec{n}})$ in Eq. (5) must then be Gaussian.

D. Comparison with Gaussian Process Emulators

Ultimately, an emulator constructed from the form proposed above depends only on the unconstrained correlations, $C_0(\vec{\theta}_i, \vec{\theta}_j)$, a result known as the kernel trick. The emulators then map perfectly to Gaussian process emulators, which are also perfectly defined once one knows $C_0(\vec{\theta}_i, \vec{\theta}_j)$. Here, the convergence parameter Λ is introduced to set the relative strength of order $K + 1$ terms to order K terms, whereas in Gaussian process emulators the same parameter is introduced to set the correlation of the unconstrained correlation. Despite these differences in starting points and perspective, the resulting emulators are identical. This equivalence demonstrates how Gaussian process emulators can be equivalently viewed as polynomial expansions if one uses the basis assumen in Eq. (4).

Gaussian process emulators often assume the correlation has a more general form,

$$C_0(\vec{\theta}_1, \vec{\theta}_2) \sim \exp\left\{\left(-\frac{|\vec{\theta}_1 - \vec{\theta}_2|^2}{2\Lambda^2}\right)^\alpha, \right\}, \tag{25}$$

where α is an additionally adjustable hyper parameter. For $\alpha = 1$ this corresponds precisely to the form discussed above. If the Taylor expansion for the emulator is analytic, one should expect that the kernel, $C_0(\vec{\theta}_1, \vec{\theta}_2)$, should also be analytic in terms of $\vec{\theta}$. The form expressed in Eq. (25) will only be analytic if α is an integer. Thus, the Gaussian form for the emulator in Eq. (18) should not come as a surprise. What is perhaps surprising is that one α cannot be some other integer.

Despite the fact that the emulators are identical in practice to GP forms, there is insight into viewing the emulator from the perspective of polynomial expansions. Once one considers the emulator form in terms of terms of a polynomial expansion, one also understands how many training points are needed to constrain all terms of a specific order. For example, for a linear fit, $N_{\rm pars}+1$ training points are needed to determine the intercept and slopes. For a quadratic fit, one would need $(N_{\rm pars}+1)(N_{\rm pars}+2)/2$ terms, and for a cubic fit one would need $(N_{\rm pars}+1)(N_{\rm pars}+2)(N_{\rm pars}+3)/6$ training points. Thus, it might motivate one to choose a number of training points to constrain all coefficients to a given order.

E. Tuning the Emulator

Due to the Gaussian weights, the most probable set of expansion coefficients would be when all $A_{\vec{n}}$ are zero. However, once one fits to $N_{\rm train}$ training points one needs to find the set of coefficients that both maximizes the weight function, $W(A_{\vec{n}})$, while reproducing the $N_{\rm train}$ training points. Whereas there might be dozens or a few hundred training points, there aren and infinite number of coefficients. Thus, one needs to maximize the weight with $N_{\rm train}$ constraints. Here, it is shown how one can analytically solve for the set of most probable coefficients, $A_{\vec{n}}$, given the the training values, $y_a = y(\vec{\theta}_a)$, where $\vec{\theta}_a$ are the points at which the emulator was trained. Again, we consider $N_{\rm train}$ to be the number of training points and $N_{\rm coef}$ to be the number of coefficients, which is much larger than $N_{\rm train}$. For shorthand, we define the function $T_{\vec{n}}(\vec{\theta})$ to rewrite the emulator in Eq. (18),

$$\begin{aligned} Y(\vec{\theta}) &= \sum_{\vec{n}}^{N_{\rm coeff}} A_{\vec{n}} \mathcal{T}_{\vec{n}}(\vec{\theta}), \\ \mathcal{T}_{\vec{n}}(\vec{\theta}) &= \frac{1}{\sqrt{n_1! n_2! \cdots n_{N{\rm pars}}!}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_{N{\rm pars}}} e^{-|\vec{\theta}|^2/2\Lambda^2}. \end{aligned} \tag{26}$$

Here, $T_{\vec{n}}(\vec{\theta})$ is shorthand for the basis functions used to represent the emulator. None of the following derivations are dependent on this particular set of basis functions.

Evaluated at the $N_{\rm train}$ training points, one can define the $N_{\rm train} \times N_{\rm coef}$ matrix,

$$\sum_{\vec{n}} T_{a,\vec{n}} \equiv \mathcal{T}_{\vec{n}}(\vec{\theta}_a), \tag{27}$$

where $\vec{\theta}_a$ labels the training points, $0 < a \leq N_{\rm train}$, and $N_{\rm train} << N_{\rm coef}$.

The goal is to find the coefficients, $A_{\vec{n}}$, that maximize the probability given the constraints of matching the training data. If the training data are to be exactly matched, the differential probability of a given set of coefficients

$$dP(\vec{A}, \sigma_A, \Lambda) \sim \prod_{\vec{n}}^{N_{\rm coef}} \frac{dA_{\vec{n}}}{(2\pi\sigma_A^2)^{1/2}} e^{-A_{\vec{n}}^2/2\sigma_A^2} \prod_{a=1}^{N_{\rm train}} \delta[y(\vec{\theta}_a) - Y(\vec{\theta}_a)] \tag{28}$$

If the training constraints are not exact, the delta function should be replaced with some other form, perhaps a Gaussian. The Gaussian would be reasonable if the training had some sort of point-by-point noise, such as one would have if the full-model calculation involved some sort of sampling or Monte Carlo evaluations. However, more generally, one could also consider the fact that the training calculation might have some sort of error, e.g. that of missing physics. In that cast the error might be correlated between points. Noise in the full-model calculations at points $\vec{\theta}_a$ and $\vec{\theta}_b$ can be characterized by some error matrix,

$$\langle \delta y_a \delta y_b \rangle = \sigma_A^2 \Delta_{ab}. \tag{29}$$

Here, $y_a = y(\vec{\theta}_a)$, the full model value calculated at the training point a, and δy_a is the uncertainty of the full model due to point-by-point noise in the full model or any other source of error. The choice to scale out the factor of σ_A^2 in the definition is so that Δ_{ab} is dimensionless. If the random noise is completely uncorrelated, Δ_{ab} is diagonal. For example, if one were to recalculate the full model at exactly the same value of $\vec{\theta}_a$, the new value of y_a could be different. This is often the case for models involving Monte Carlo calculation or if numerical noise is non-negligible. Given that the characteristic size of the observable is σ_A, the elements of the diagonal matrix would then be chosen according to the percentage random error. For example, if the random error were 1%, one would chose $\Delta_{ab} = \delta_{ab} 10^{-4}$.

To account for the uncertainty represented by Δ, one can relax the constraint imposed by the delta function by replacing it,

$$\prod_a \delta[y_a - Y(\vec{\theta}_a)] \rightarrow \frac{|\Delta|^{-1/2}}{(2\pi)^{N_{\rm train}/2}} \exp\left\{ \frac{-1}{2}(y_a - Y(\vec{\theta}_a))\Delta_{ab}^{-1}(y_b - Y(\vec{\theta}_b)) \right\}. \tag{30}$$

The second step above exploits the fact that the Fourier transform of a Gaussian is a Gaussian. The r.h.s. above behaves like a delta function in the limit that the eigenvalues of Δ approach infinity. Otherwise, in the basis where Δ is diagonal this can be thought of as a finite-width modification to a delta function. Next, one can replace $Y(\vec{\theta})$ in Eq. (30) using Eq. (26). The altered δ functions then becomes

$$\prod_a \delta[y_a - Y(\vec{\theta}_a)] \rightarrow \int \prod_a \left(\frac{dk_a}{2\pi}\right) \exp\left\{ ik_a(y_a - \sum_{\vec{m}} A_{\vec{m}} T_{\vec{m}}(\vec{\theta}_a)) - \sum_b \frac{\sigma_A^2}{2} k_a \Delta_{ab} k_b \right\}. \tag{31}$$

Defining the net probability after integrating over the coefficients above, but not over σ_A^2 or Λ,

$$\begin{aligned}
\mathcal{Z}(\sigma_A^2,\Lambda) &\equiv \int\left(\prod_{\vec{n}} dA_{\vec{n}}\right)\ \frac{dP(\vec{A},\sigma_A,\Lambda)}{\prod dA_{\vec{n}}} \qquad (32)\\
&= \int\left(\prod_{\vec{n}} \frac{dA_{\vec{n}}}{\sqrt{2\pi\sigma_A^2}}\right)\ \exp\left[-\frac{\sum_i A_{\vec{i}}^2}{2\sigma_A^2}\right]\int\frac{dk}{2\pi}\exp\left[i\sum_b k_b(y_b-\sum_\ell A_\ell T_\ell(\vec{\theta}_b))-\frac{\sigma_A^2}{2}\sum_{ab}k_a\Delta_{ab}k_b\right].
\end{aligned}$$

The function $Z(\sigma_A^2,\Lambda)$ is the conditional probability of observing the training data given σ_A and Λ given the training data. From Eq. (28) one can see that $Z(\sigma_A^2,\Lambda)$ is normalized when integrated over all possible training data,

$$\int\left(\prod_a dy_a)Z(\sigma_A^2,\Lambda\right)=1.(33)$$

The average coefficients, and correlation of coefficients, for a given σ_A and Λ, become

$$\begin{aligned}
\langle A_{\vec{n}}\rangle &= \mathcal{Z}^{-1}\int\left(\prod_{\vec{i}}\frac{dA_{\vec{i}}}{\sqrt{2\pi\sigma_A^2}}\right)\ \exp\left[-\sum_{\vec{j}}\frac{A_{\vec{j}}^2}{2\sigma_A^2}\right]A_{\vec{n}} \qquad (34)\\
&\quad \int\left(\prod_a\frac{dk_a}{2\pi}\right)\exp\left[i\sum_b k_b(y_b-\sum_{\vec{\ell}}A_{\vec{\ell}}T_{\vec{\ell}}(\vec{\theta}_b))-\frac{\sigma_A^2}{2}\sum_{ab}k_a\Delta_{ab}k_b\right],\\
\langle A_{\vec{m}}A_{\vec{n}}\rangle &= \mathcal{Z}^{-1}\int\left(\prod_{\vec{i}}\frac{dA_{\vec{i}}}{\sqrt{2\pi\sigma_A^2}}\right)\ \exp\left[-\sum_j\frac{A_{\vec{j}}^2}{2\sigma_A^2}-\frac{\sigma_A^2}{2}\sum_{ab}k_a\Delta_{ab}k_b\right]\\
&\quad A_{\vec{m}}A_{\vec{n}}\int\prod_a\frac{dk_a}{2\pi}\exp\left[i\sum_b k_b(y_b-\sum_{\vec{\ell}}A_{\vec{\ell}}T_{\vec{\ell}}(\vec{\theta}_b))\right],
\end{aligned}$$

After completing the square, the correlation can be written as

$$\begin{aligned}
\langle A_{\vec{m}}A_{\vec{n}}\rangle &= \mathcal{Z}^{-1}\int\left(\prod_a\frac{dk_a}{2\pi}\right)e^{i\sum_b k_by_b-\frac{\sigma_A^2}{2}\sum_{ab}k_a\Delta_{ab}k_b} \qquad (35)\\
&\quad\int\left(\prod_{\vec{i}}dA_{\vec{i}}\right)\ A_{\vec{m}}A_{\vec{n}}\exp\left[-\frac{1}{2\sigma_A^2}\left(A_{\vec{i}}+i\sigma_A^2\sum_b T_{\vec{i}}](\vec{\theta}_b)k_b\right)^2-\frac{\sigma_A^2}{2}\left(\sum_{b,\vec{\ell}}T_{\vec{\ell}}(\vec{\theta}_b)k_b\right)^2\right]\\
&= \mathcal{Z}^{-1}\left(\int\prod_a\frac{dk_a}{2\pi}\right)\ \exp\left[-\frac{\sigma_A^2}{2}(T_{\vec{m}}(\vec{\theta}_b)k_b)^2+ik_by_b-\frac{\sigma_A^2}{2}k_a\Delta_{ab}k_b\right]\\
&\quad\left(\int\prod_{\vec{i}}dA_{\vec{i}}\right)\ (A_{\vec{n}}-i\sigma_A^2\sum_b T_k(\vec{\theta}_b)k_b)(A_{\vec{m}}-i\sigma_A^2T_{\vec{m}}(\vec{\theta}_c)k_c)\exp\left[-\sum_{\vec{\ell}}\frac{A_{\vec{\ell}}^2}{2\sigma_A^2}\right]
\end{aligned}$$

Similarly,

$$\begin{aligned}
\langle A_{\vec{m}}\rangle &= -\mathcal{Z}^{-1}\int\left(\prod_a\frac{dk_a}{2\pi}\right)\ \exp\left[-\sum_b\frac{\sigma_A^2}{2}(T_{\vec{m}}(\vec{\theta}_b)k_b)^2+i\sum_b k_by_b-\frac{\sigma_A^2}{2}\sum_{ab}k_a\Delta_{ab}k_b\right] \qquad (36)\\
&\quad\int\left(\prod_{\vec{i}}\frac{dA_{\vec{i}}}{\sqrt{2\pi\sigma_A^2}}\right)\ i\sigma_A^2\left(\sum_c T_{\vec{m}}(\vec{\theta}_c)k_c\right)\exp\left[-\sum_{\vec{\ell}}\frac{A_{\vec{\ell}}^2}{2\sigma_A^2}\right], \qquad (37)
\end{aligned}$$

It is convenient to define the function,

$$\begin{aligned}
C_0(\vec{\theta}_1,\vec{\theta}_2) &\equiv \sum_{\vec{i}}T_{\vec{i}}(\vec{\theta}_1)T_{\vec{i}}(\vec{\theta}_2) \qquad (38)\\
&= \frac{\langle Y(\vec{\theta}_1)Y(\vec{\theta}_2)\rangle_0}{\sigma_A^2}. \qquad (39)
\end{aligned}$$

The subscript zero, denotes that this is the average in the absence of training.

It is convenient to define the $N_{\rm train}\times N_{\rm train}$ matrix,

$$B_{ab}\equiv C_0(\vec{\theta}_a,\vec{\theta}_b)+\Delta_{ab}, \tag{40}$$

where $\vec{\theta}_a$ are the locations of the training points. Performing the integrals for $\mathcal{Z}(\sigma_A^2,\Lambda)$ in Eq. (32),

$$\mathcal{Z}(\sigma_A^2,\Lambda)=\sigma_A^{-N_{\rm train}}\left(\det B\right)^{-1/2}\exp\left[-\frac{1}{2\sigma_A^2}\sum_{ab}y_aB^{-1}_{ab}y_b\right]. \tag{41}$$

One can then find the moments of the integral over k,

$$\begin{aligned}\langle k_a\rangle&=-i\partial_{y_a}\ln(\mathcal{Z})=\frac{i}{\sigma_A^2}\sum_b B^{-1}_{ab}y_b,\\ \langle\delta k_a\delta k_b\rangle&=-\partial_{y_a}\partial_{y_b}\ln(\mathcal{Z})=\frac{1}{\sigma_A^2}B^{-1}_{ab}.\end{aligned} \tag{42}$$

The optimum coefficients are then:

$$\begin{aligned}\langle A_{\vec{n}}\rangle&=\sum_{ab}T_{\vec{n}}(\vec{\theta}_a)B^{-1}_{ab}y_b,\\ \langle\delta A_{\vec{m}}\delta A_{\vec{n}}\rangle&=\delta_{\vec{m},\vec{n}}\sigma_A^2-\sigma_A^2\sum_{ab}T_{\vec{m}}(\vec{\theta}_a)B^{-1}_{ab}T_{\vec{n}}(\vec{\theta}_b).\end{aligned} \tag{43}$$

The observables and their fluctuations due to the fluctuation in the coefficient are then:

$$\begin{aligned}\langle Y(\vec{\theta})\rangle&=\sum_{\vec{n}}\langle A_{\vec{n}}\rangle T_{\vec{n}}(\vec{\theta}),\\ &=\sum_{ab}y_aB^{-1}_{ab}C_0(\vec{\theta}_b,\vec{\theta}_a),\\ \langle\delta Y(\vec{\theta})^2\rangle&=\sum_{\vec{m}\vec{n}}\langle\delta A_{\vec{m}}\delta A_{\vec{n}}\rangle T_{\vec{m}}(\vec{\theta})T_{\vec{n}}(\vec{\theta})\\ &=\sigma_A^2\left[C_0(\vec{\theta},\vec{\theta})-\sum_{ab}C_0(\vec{\theta},\vec{\theta}_a)B^{-1}_{ab}C_0(\vec{\theta}_b,\vec{\theta})\right].\end{aligned} \tag{44}$$

The fact that the prediction for $Y(\vec{\theta})$ and its uncertainty can be expressed so simply in terms of the correlation kernel C_0 without involving the coefficients is the well known kernel trick [44]. This enables one to build the emulator without every bothering to calculate the individual expansion coefficients $A_{\vec{n}}$

After defining

$$\chi_a\equiv\textstyle\sum_b B^{-1}_{ab}y_b, \tag{45}$$

the emulated value and emulator uncertainty at some point $\vec{\theta}$ are

$$\begin{aligned}Y(\vec{\theta})&=\sum_a\chi_aC_0(\vec{\theta}_a,\vec{\theta}),\\ \langle\delta Y(\vec{\theta})^2\rangle&=\sigma_Y^2(\vec{\theta})=\sigma_A^2C_0(\vec{\theta},\vec{\theta})-\sigma_A^2\sum_{ab}C_0(\vec{\theta},\vec{\theta}_a)B^{-1}_{ab}C_0(\vec{\theta}_b,\vec{\theta}).\end{aligned} \tag{46}$$

The optimum value $\langle Y(\vec{\theta})\rangle$ depends linearly on the training values, $y_{a=1,N_{\rm train}}$. The emulator's uncertainty depends on $\vec{\theta}$ and vanishes at the training points. It is linear in σ_A and has a more complicated dependence on Λ. It does not depend on the model values at those training points, y_a, although after tuning the choice of σ_A will depend on the training values. Not surprisingly, this is the exactly the same expression used for Gaussian process emulators, with the requirement that the form for the GP correlation kernel should be Gaussian.

Even though the procedure outlined above allows one to solve for all the coefficients, $A_{\vec{n}}$, solving for the individual coefficients is not necessary because both the emulator and its uncertainty can be expressed, as shown in Eq. (46), in terms of the bare correlation kernel, $C_0(\vec{\theta}_1,\vec{\theta}_2)$. The simple form is the same for some GP emulators, but rather than being arbitrarily assigned as is the case for the GP paradigm, in this perspective the form is motivated by the three simple assumptions: analyticity, independent Gaussian fluctuation of the polynomial terms, and invariance to translations and rotations. The fact that the emulator can be expressed simply in terms of the kernel, $C_0(\vec{\theta}_1, \vec{\theta}_2)$, is not unique to the polynomial expansion used here. Any form where $Y(\vec{\theta})$ is expressed as a sum of terms, where each term is linear in some coefficient, A_i, that has some Gaussian distribution, can be expressed in the form of Eq. (46). This is known as the *kernel trick* [44] and has been exploited in many previous studies involving polynomial expansions. For example, if one had chosen the form $f_K(|\vec{\theta}|^2) = 1$, rather than being a Gaussian, the fluctuation would then not be independent of the norm of $\vec{\theta}$, and the correlation would no longer depend only on $|\vec{\theta}_1 - \vec{\theta}_2|^2$, but one could still optimize the expansion. The resulting correlation kernel would change to $C_0(\vec{\theta}_1, \vec{\theta}_2) = e^{\vec{\theta}_1 \cdot \vec{\theta}_2/\Lambda^2}$, but the form of Eq. (46) would not change.

Despite the fact that the emulator's behavior is completely determined by the kernel, knowing the form for the polynomial expansion that satisfies the three aforementioned constraints is nonetheless useful for testing. In Sec. IV random test functions will be created using this form. Each function is characterized by σ_A, Λ and the set of random Gaussian coefficients. If one trains an emulator as is done above, the emulator's value of σ_A and Λ should be close to those used to generate the function. Such tests are presented in Sec. IV.

The fluctuation described in Eq. (46) were only those due to the fluctuation of the coefficients. This fluctuation was calculated ignoring the fact that the hyper-parameters σ_A and Λ are not precisely determined. An additional contribution to $\langle \delta Y(\vec{\theta})^2 \rangle$ arises if one accounts for the variance in Λ. This is the subject of the next subsection.

F. Finding the Optimized Hyper-Parameters

This approach to emulation has two hyper parameters, the variance of the coefficients, σ_A, and the convergence parameter Λ. If one has sufficient training data, one can find their optimum values. The next subsection considers choosing σ_A for some fixed Λ and the following subsection considers choosing Λ while fixing σ_A.

Both σ_A and Λ are unknown, and the probability distribution for them is determined by the function $Z(\sigma_A^2, \Lambda)$, which can be considered as the conditional probability of observing the training data given σ_A and Λ. Restating Eq. (41),

$$\begin{aligned} \mathcal{Z}(\sigma_A^2, \Lambda) &= \int \prod_{\vec{i}} dA_{\vec{i}} P(\vec{A}, \sigma_A^2, \Lambda) \\ &= \sigma_A^{-N_{\rm train}} (\det B)^{-1/2} \exp\left[-\frac{1}{2\sigma_A^2} y_a B_{ab}^{-1} y_b\right]. \end{aligned} \tag{47}$$

By Bayes theorem the probability of σ_A and Λ is proportional to the product of Z and $Q(\sigma_A^2, \Lambda)$, the prior probability of σ_A and Λ, i.e. $Q(\sigma_A^2, \Lambda)$ is the probability for σ_A and Λ in the absence of training data. The most appealing method to chose σ_A and Λ, would be to choose their average over the weight ZQ,

$$\begin{aligned} \bar{\sigma}_A &= \frac{1}{\mathcal{N}} \int d\sigma_A d\Lambda\ \sigma_A Z(\sigma_A^2, \Lambda) Q(\sigma_A^2, \Lambda), \\ \bar{\Lambda} &= \frac{1}{\mathcal{N}} \int d\sigma_A d\Lambda\ \Lambda\ Z(\sigma_A^2, \Lambda) Q(\sigma_A^2, \Lambda), \\ \mathcal{N} &= \int d\sigma_A d\Lambda Z(\sigma_A^2, \Lambda) Q(\sigma_A^2, \Lambda). \end{aligned} \tag{48}$$

Similarly, one can find expressions for the uncertainties of the hyper parameters by calculating higher-moment integrals. We define the covariance matrix $W_{\alpha,\beta}$,

$$\begin{aligned} W_{\sigma_A \sigma_A} &= \langle (\sigma_A - \bar{\sigma_A})^2 \rangle, \\ W_{\sigma_A \Lambda} = W_{\Lambda \sigma_A} &= \langle (\sigma_A - \bar{\sigma_A})(\Lambda - \bar{\Lambda}) \rangle, \\ W_{\Lambda\Lambda} &= \langle (\Lambda - \bar{\Lambda})^2 \rangle. \end{aligned} \tag{49}$$

Here, the notation $\langle \mathcal{O} \rangle$ denote the average of some quantity $\mathcal{O}$ given the weight $Z(\sigma_a^2, \Lambda) Q(\sigma_A^2, \Lambda)$. Unfortunately, the functions $Z(\sigma_A, \Lambda)$ approaches a constant as $\Lambda \to \infty$. This means that all moments, even the normalization, are strongly affected by the range of the prior, $Q(\sigma_A^2, \Lambda)$. In fact, if the prior has some range $\Lambda_{\rm max}$, the $n^{\rm th}$ moment would scale as $\Lambda_{\rm max}^n$ for large $\Lambda_{\rm max}$. Thus, this weight gives one pause in using moments of the distribution to represent best values or uncertainties.

As an alternative to integrating the expressions above over Λ in order to find the optimized values of σ_A and Λ, one can solve for the values of σ_A and Λ that maximize $Z(\sigma_A^2, \Lambda)$ as alternatives to $\bar{\Lambda}$ and $\bar{\sigma}_A$. Further, one can assign the uncertainty by calculating the curvatures, i.e. the second derivatives of $Z(\sigma_A^2, \Lambda)$ evaluated at the values of σ_A and Λ for which the first derivatives vanish.

$$W_{\alpha\beta} = -\frac{\partial^2 \ln(Z)}{\partial\alpha\partial\beta}. \tag{50}$$

Essentially, this corresponds to assuming that $\ln(Z)$ is quadratic in the parameters. Normally, choosing the point of the maximum and its curvature is a reasonable assumption whenever Z is a sharply peaked function. If Z were a Gaussian function of the parameters, which it is not, one could assume an arbitrarily wide prior, and the averages of parameters and their covariances would be equivalent to the position of the maximum and the curvature at the maximum.

Another advantage of using the alternative expressions for σ_A and Λ and their uncertainties are that the quantities are calculated quickly. A Newtons method algorithm required $\lesssim 10$ evaluations of $Z(\sigma_A^2, \Lambda)$ and its derivatives. Both Z and its derivatives are easily calculated and finding these points at which Z is maximized is robust. The calculations shown in this paper apply this alternative. As a test, in Sec. IV functions $y(\vec{\theta})$ are built with polynomial coefficients chosen consistent with the form assumed by the emulator with specific values of σ_A and Λ. Emulators, $Y(\vec{\theta})$ are then constructed with σ_A and Λ chosen according to the method described above. The chosen values of σ_A and Λ well match those used to construct $y(\vec{\theta})$ for larger numbers training points.

1. *Choosing the Characteristic Coefficient Width*

First, we consider fixing the convergence parameter Λ and finding the optimum value of σ_A. This involves maximizing the probability $Z(\sigma_A^2, \Lambda)$, defined in Eq. (32) for a fixed Λ. The goal here is to find the value of σ_A that maximizes $\mathcal{Z}$. For the case where $\Delta = 0$ or if Δ is independent of σ_A, one can find σ_A rather easily because B_{ab} does not depend on σ_A. In that case, setting the derivative w.r.t. σ_A^2 equal to zero,

$$0 = \frac{N_{\rm train}}{\sigma_A^2}\mathcal{Z} - \frac{\mathcal{Z}}{\sigma_A^4} y_a B^{-1}_{ab} y_b, \tag{51}$$

$$\sigma_A^2 = \frac{1}{N_{\rm train}} y_a B^{-1}_{ab} y_b. \tag{52}$$

2. *Choosing the Convergence Parameter*

Eq. (52) provides the means for choosing σ_A for a given value Λ. One can then plot $Z(\sigma_A^2, \Lambda)$, as given in Eq. (47), as a function of Λ and choose the value of Λ that maximizes Z. For some value of Λ the expressions for B_{ab} and σ_A are easily found. One can also find analytic expressions for the derivatives of B with respect to Λ. This allows one to use a Newtons method to find the optimized values of σ_A and Λ with a few interactions. The uncertainties defined by Eq. (50) are then found analytically.

One should be careful in using smaller values of the convergence parameter, especially when there are a large number of parameters. The contribution to the variance of the emulator for a specific term in the polynomial expansion scales as $1/\Lambda^{2n}$, if n is the order of the term. The number of terms of a given order scales as $N_{\rm pars}^n$. Thus, the importance of a given order relative to the previous order scales as $N_{\rm pars}/\Lambda^2$. If $\Lambda \lesssim N_{\rm pars}^{1/2}$, the emulator uncertainty does not converge well for increasing order. For example, for four model parameters, one might find that any model characterized by $\Lambda \gtrsim 2$ can be reasonably well emulated by a Taylor expansion, a model with 16 parameters might need $\Lambda \gtrsim 4$ to converge similarly as function of the order K. The challenge of handling emulation with large numbers of parameters is not confined to emulators based on polynomial expansions, but can be a problem for all emulators. A discussion of these issues is given in Sec. III.

G. Contribution to the Emulator Uncertainty from the Uncertainty of the Hyper Parameters

The emulator uncertainty $\sigma_Y(\vec{\theta})$ has contributions from the variance of the expansion coefficients for given values of σ_A and Λ, and also contributions from the fact that σ_A and Λ are not known. Here, we present a general discussion of how one might approximate the latter contribution. We consider a set of hyper parameters, $\vec{h}$, which in this case would refer to σ_A and Λ.

In this derivation, $\langle\mathcal{O}\rangle(\vec{h})$ refers to the the average of $\mathcal{O}$ over all the possible expansion coefficients given the training data for a specific set of hyper parameters $\vec{h}$. The double brackets $\langle\langle ...\rangle\rangle$ refer to the average once one has also averaged over $\vec{h}$. Here $\vec{h}_0$ is the hyper parameter vector at which the probability $P(\vec{h})$ is maximized, $\bar{Y}(\vec{h}) = \langle Y\rangle(\vec{h})$, and $\bar{Y}_0 = \langle Y\rangle(\vec{h}_0)$. The goal is to estimate the variance of the emulated value at some point in parameter space, $\vec{\theta}$.

$$\begin{aligned}
\langle\langle\delta Y^2\rangle\rangle &= \int d\vec{h}\; P(\vec{h})\langle[Y-\bar{Y}_0]^2\rangle(\vec{h}) \qquad (53)\\
&= \int d\vec{h}\; P(\vec{h})\langle[(Y-\bar{Y}(\vec{h})+(\bar{Y}(\vec{h})-\bar{Y}_0)]^2\rangle(\vec{h})\\
&= \int d\vec{h}\; P(\vec{h})\left\{\langle(Y-\bar{Y}(\vec{h})^2\rangle(\vec{h})+\langle(\bar{Y}(\vec{h})-\bar{Y}_0)^2\rangle\right\}\\
&= \int d\vec{h}\; P(\vec{h})\left\{\langle(Y-\bar{Y}_0)^2\rangle_0+\frac{(h_i-h_{0i})(h_j-h_{0j})}{2}\frac{\partial^2}{\partial h_i\partial h_j}\langle(Y-\bar{Y}(\vec{h}))^2\rangle\right.\\
&\qquad\qquad \left.+(h_i-h_{0i})(h_j-h_{0j})\left(\frac{\partial\bar{Y}(\vec{h})}{\partial h_i}\right)\left(\frac{\partial\bar{Y}(\vec{h})}{\partial h_j}\right)+\mathcal{O}(h-h_0)^3\right\}\\
&\approx \langle(Y-\bar{Y}_0)^2\rangle_0+\left[\frac{\partial\bar{Y}(h)}{\partial h_i}\frac{\partial\bar{Y}(h)}{\partial h_j}\right]W_{ij}, \qquad (54)\\
W_{ij} &= \int d\vec{h}\; P(\vec{h})(h_i-h_{0i})(h_j-h_{0j}).
\end{aligned}$$

The approximation of the last step is built on the assumption that sufficient training data has been provided so that

1. $P(\vec{h})$ is sufficiently narrow about $\vec{h}_0$ that higher order terms in $\vec{h}-\vec{h}_0$ can be ignored.
2. $\langle(Y-\bar{Y})^2\rangle_0$ is small so that terms that scale both as $\langle(Y-\bar{Y})^2\rangle_0$ and $(\vec{h}-\vec{h}_0)^2$ can be ignored.

One of the remaining terms is small because $\langle(Y-\bar{Y})^2\rangle_0$ and the second is small because $\vec{h}-\vec{h}_0$ is small. The neglected terms should all be increasingly smaller relative to these terms as the number of training points is increased.

Applying the approximation of Eq. (54 to the calculations of the uncertainty of the emulator,

$$\begin{aligned}
\sigma_Y^2(\vec{\theta}) &\approx \sigma_A^2\left[C_0(\vec{\theta},\vec{\theta})-C_0(\vec{\theta},\vec{\theta}_a)B^{-1}_{ab}C_0(\vec{\theta}_b,\vec{\theta})\right] \qquad (55)\\
&\quad + W_{\sigma_A\sigma_A}\left(\frac{\partial Y}{\partial\sigma_A}\right)^2+2W_{\sigma_A\Lambda}\left(\frac{\partial Y}{\partial\sigma_A}\right)\left(\frac{\partial Y}{\partial\Lambda}\right)+W_{\Lambda\Lambda}\left(\frac{\partial Y}{\partial\Lambda}\right)^2.
\end{aligned}$$

The r.h.s. of Eq. (55) is evaluated with σ_A and Λ chosen to maximize $Z(\sigma_a,\Lambda)$, and the covariance of the hyper-parameters, W, is approximated from the curvature of $Z(\sigma_A^2,\Lambda)$, as described in Eq. (50). The first term represents the emulator's uncertainty due to the uncertainty of the coefficients $A_{\vec{n}}$ for fixed σ_A and Λ, and the terms proportional to W provide the uncertainty due to the fact that σ_A and Λ are themselves uncertain. In the limit of large numbers of training points the range of Λ is well constrained around the optimum value and this becomes an excellent approximation, but the approximation becomes dubious when Λ is poorly constrained. Often the contributions to the uncertainty from the terms proportional to $W_{\alpha\beta}$ are ignored. In our studies it was found these corrections were relatively small when one chooses a reasonable set of training-point locations. However, these terms were found to be critical for optimizing the choice of locations of training points, $\vec{\theta}_{a=1\cdots N_{\rm train}}$. For example, if all the training points were placed very close to the origin, the terms were found to be significant, but for optimized placement the terms were very small. In contrast the strength of the other terms was observed to be nearly independent of the training-point placement. Thus, these additional terms played an outsized role when optimizing training-point placement.

Restating Eq. (46), the emulator depends on the training values as,

$$Y(\vec{\theta}) = C_0(\vec{\theta},\vec{\theta}_a)B^{-1}_{ab}y_b, \qquad (56)$$

and is independent of σ_A because C_0 and B depend only on Λ. Thus, $\partial_{\sigma_A}Y(\vec{\theta})=0$, and one need only worry about $W_{\Lambda\Lambda}$ and can ignore $W_{\sigma_A\sigma_A}$ or $W_{\sigma_A,\Lambda}$, even though one needs to calculate all the terms in W^{-1} before finding W.

The probability Z depends on σ_A, Λ, the location of the training points and the training values.

$$\ln(Z) = -N_{\rm train}\ln(\sigma_A)-\frac{1}{2}\ln(|B|)-\frac{1}{2\sigma_A^2}y_aB^{-1}_{ab}y_b. \qquad (57)$$

Because B is independent of σ_A,

$$\begin{aligned}
W^{-1}_{\sigma_A\sigma_A} &= \frac{-N_{\rm train}}{\sigma_A^2} + \frac{3}{\sigma_A^4} y_a B^{-1}_{ab} y_b, \\
W^{-1}_{\sigma_A\Lambda} &\approx \frac{1}{\sigma_A^3} y_a \left(\frac{d}{d\Lambda} B^{-1}\right)_{ab} y_b, \\
W^{-1}_{\Lambda\Lambda} &\approx \frac{1}{2}\frac{d^2}{d\Lambda^2}\ln(|B|) - \frac{1}{2\sigma_A^2} y_a \left(\frac{d}{d\Lambda}\right)^2 B^{-1}_{ab} y_b.
\end{aligned} \tag{58}$$

These expressions assume that near the optimum values, $Z(\sigma_A,\Lambda)$ can be treated as if it were a Gaussian, so that the widths are determined by the curvature of $Z(\sigma_A,\Lambda)$ around the optimum. To take the derivatives w.r.t. B^{-1}, one can use the identity

$$\frac{d}{d\Lambda} B^{-1}_{ab} = -B^{-1}_{ac}\left(\frac{d}{d\Lambda} B_{cd}\right) B^{-1}_{db}. \tag{59}$$

After defining

$$\begin{aligned}
B'_{ab} &= \Lambda^3 \frac{d}{d\Lambda} B_{ab} = (|\vec{\theta}_a - \vec{\theta}_b|^2) B_{ab}, \\
B''_{ab} &= \Lambda^3 \frac{d}{d\Lambda} B'_{ab} = (|\vec{\theta}_a - \vec{\theta}_b|^4) B_{ab}.
\end{aligned} \tag{60}$$

one finds

$$\begin{aligned}
W^{-1}_{\sigma_A\sigma_A} &= N_{\rm train}\frac{1}{\sigma_A^2} - \frac{3}{\sigma_A^4} y_a B^{-1}_{ab} y_b, \\
W^{-1}_{\sigma_A\Lambda} &= \frac{-1}{\Lambda^3\sigma_A^3} y_a \left(B^{-1}B'B^{-1}\right)_{ab} y_b, \\
W^{-1}_{\Lambda\Lambda} &= \frac{1}{2}\frac{d^2}{d\Lambda^2}\ln(|B|) + \frac{1}{2\sigma_A^2\Lambda^6} y_a \left(2B^{-1}B'B^{-1}B'B^{-1} - B^{-1}B''B^{-1}\right)_{ab} y_b \\
&\quad + \frac{3}{2\sigma_A^2\Lambda^4} y_a \left(B^{-1}B'B^{-1}\right)_{ab} y_b.
\end{aligned} \tag{61}$$

To calculate $(d^2/d\Lambda^2)\ln(|B|)$, one can use the identity

$$\frac{d}{d\Lambda}|B| = |B| \mathrm{tr}\left(B^{-1}\frac{dB}{d\Lambda}\right). \tag{62}$$

This leads to

$$\frac{d^2}{d\Lambda^2}\ln(|B|) = -\mathrm{tr}\left(B^{-1}\frac{dB}{d\Lambda}B^{-1}\frac{dB}{d\Lambda}\right) + \mathrm{tr}\left(B^{-1}\frac{d^2B}{d\Lambda^2}\right). \tag{63}$$

The last piece required to calculate the emulator uncertainty in Eq. (55) is $\partial_\Lambda Y(\vec{\theta})$.

$$\begin{aligned}
\partial_\Lambda Y(\vec{\theta}) &= \partial_\Lambda C_0(\vec{\theta},\vec{\theta}_a) B^{-1}_{ab} y_b \\
&= \frac{1}{\Lambda^3}\left[|\vec{\theta}-\vec{\theta}_a|^2 C_0(\vec{\theta},\vec{\theta}_a) B^{-1}_{ab} - C_0(\vec{\theta},\vec{\theta}_a)(B^{-1}B'B^{-1})_{ab}\right] y_b.
\end{aligned} \tag{64}$$

Thus, to estimate the contribution to the emulator uncertainty due to the uncertainties in determining the hyper parameters one should follow the following steps.

1. First tune the emulator and find the optimum values of σ_A and Λ.
2. Find the 2×2 matrix W^{-1} by using Eq.s (61) with the matrices B, B' and B'' calculated using Λ from step 1. This also involves finding $(d^2/d\Lambda^2)\ln(|B|)$.
3. Invert W^{-1} to find W. Only $W_{\Lambda\Lambda}$ will be needed because $\partial Y/\partial\sigma_A = 0$.
4. Find $dE/d\Lambda$ from Eq. (64) and using $W_{\Lambda\Lambda}$ calculate the contribution to σ_Y^2 in Eq. (55).

H. The Emulator Uncertainty at Training Points

If one sets $\vec{\theta}=\vec{\theta}_c$, one of the training points, and if $\Delta=0$, then

$$\begin{aligned} C_0(\vec{\theta}_a,\vec{\theta}_c)&=B_{ac}, \\ T_{\vec{m}}(\vec{\theta}_c)T_{\vec{m}}(\vec{\theta}_c)&=B_{cc},\ \text{no sum over } c, \end{aligned} \tag{65}$$

and

$$\begin{aligned} Y(\vec{\theta}_c)&=\chi_a C_0(\vec{\theta}_a,\vec{\theta}_c)=B^{-1}_{ab}y_bB_{ac}=y_c, \\ \sigma_Y^2(\vec{\theta}_c)|_{\Delta=0}&=\sigma_A^2B_{cc}-\sigma_A^2B_{ca}B^{-1}_{ab}B_{bc}+W_{\Lambda\Lambda}\left(\frac{\partial Y}{\partial\Lambda}\right)^2 \\ &=W_{\Lambda\Lambda}\left(\frac{\partial Y}{\partial\Lambda}\right)^2 \end{aligned} \tag{66}$$

Next, one can see that evaluated at a training point,

$$\begin{aligned} \frac{\partial Y}{\partial\Lambda}&=\frac{1}{\Lambda^3}\left[|\vec{\theta}-\vec{\theta}_a|^2B_{ca}B^{-1}_{ab}-B_{ca}(B^{-1}B'B^{-1})_{ab}\right]y_b \\ &=0. \end{aligned} \tag{67}$$

Thus, if $\Delta=0$, there is no uncertainty when evaluated at the training points and the optimum value for the emulator exactly reproduces the training values.

When $\Delta\neq 0$, the optimum is no longer constrained to exactly reproduce the full model at the training points, and the uncertainty at a training point, $\vec{\theta}_c$ is no longer zero,

$$\langle\sigma_Y(\vec{\theta}_c)^2\rangle=\sigma_A^2\left(\Delta_{cc}-\Delta_{ca}B^{-1}_{ab}\Delta_{bc}\right),\quad \text{(no sum over } c). \tag{68}$$

If Δ is small, the uncertainty of the emulator is simply given by $\sigma_A\Delta$, however it is somewhat reduced for larger Δ.

Typically, the emulator will be applied thousands, or perhaps millions of times, all for different $\vec{\theta}$. Before evaluating the emulator, it would be efficient to first calculate and store both B^{-1}_{ab} and χ_a. Calculating the matrix B involves calculating $N_{\rm train}(N_{\rm train}+1)/2$ exponentials, which is extremely fast. One must then invert B, an $N_{\rm train}\times N_{\rm train}$ matrix.

When considering a specific $\vec{\theta}$, one calculates $C_0(\vec{\theta}_a,\vec{\theta})$ for each a according to Eq. (19). Once these are stored, calculating $\langle\delta E^2\rangle$ involves calculating a double sum with $N^2_{\rm train}$ elements.

The derivations above could have been performed finding the coefficients $A_{\vec{n}}$ that maximize the probability, $Z(\vec{A})$, rather than solving for the mean coefficients $\langle A_{\vec{n}}\rangle$, but the expressions for $Y(\vec{\theta})$ and $\sigma_Y(\vec{\theta})$ end up identical to those provided above.

I. Summarizing the Emulation

Despite the fact that the emulator was built in terms of optimized expansion coefficients $A_{\vec{n}}$, the entire emulator construction and application avoids any mention of the coefficients. Instead, through the kernel trick, the emulator need only know the form for the unconstrained correlation, $C_0(\vec{\theta}_1,\vec{\theta}_2)$. Using the short hand definitions of the $N_{\rm train}\times N_{\rm train}$ matrix and $N_{\rm train}$-dimensional vector χ_a,

$$\begin{aligned} B_{ab}&\equiv C_0(\vec{\theta}_a,\vec{\theta}_b)+\Delta_{ab}, \\ \chi_a&\equiv B^{-1}_{ab}y_b, \end{aligned} \tag{69}$$

one finds expressions for the following quantities,

$$\begin{aligned} \sigma_A^2&=\frac{1}{N_{\rm train}}\chi_ay_a, \\ Z(\sigma_A^2,\Lambda)&=\sigma_A^{-N_{\rm train}}|B|^{-1/2}\exp\left[-\frac{1}{2\sigma_A^2}y_aB^{-1}_{ab}y_b\right] \\ &=\sigma_A^{-N_{\rm train}}|B|^{-1/2}e^{-1/(2N_{\rm train})}\ \text{ when } \sigma_a \text{ is optimized}, \\ Y(\vec{\theta})&=\chi_aC_0(\vec{\theta}_a,\vec{\theta}), \\ \sigma_Y^2(\vec{\theta})&=\sigma_A^2\left[C_0(\vec{\theta},\vec{\theta})-C_0(\vec{\theta},\vec{\theta}_a)B^{-1}_{ab}C_0(\vec{\theta}_b,\vec{\theta})\right]+W_{\Lambda\Lambda}\left(\frac{\partial Y}{\partial\Lambda}\right)^2. \end{aligned} \tag{70}$$

Aside from the last term, proportional to $W_{\Lambda\Lambda}$, the other pieces are no different than what finds with GP emulators with Gaussian correlation kernels.

Outlining the procedure for tuning,

1. Choose the locations of the $N_{\rm train}$ training points, $\vec{\theta}_a$.
2. Choose a value of Λ.
3. Evaluate the full model at the training points, i.e. determine $y_{a=1,N_{\rm train}}$.
4. Define the noise matrix Δ_{ab}. For example, if the noise is uncorrelated, and is approximately one percent of the signal, one would choose $\Delta_{ab} = \alpha^2\delta_{ab}$, with $\alpha = 0.01$.
5. Calculate and store B, $B_{\rm inv}$, χ.
6. Calculate σ_A^2 according to Eq. (52).
7. Calculate $Z(\sigma_A^2, \Lambda)$.
8. Repeat steps 2-7 for different Λ and keep the value of Λ, and its corresponding σ_A, that maximized Z.
9. Once both Λ and σ_A are determined, calculate and store $W_{\Lambda\Lambda}$.

To this point, the most numerically taxing part of the procedure is inverting B and calculating its determinant. One might need to repeat the steps, as mentioned in (8) roughly a dozen times to best find Λ.

Once this is finished, one can evaluate the emulator and its uncertainty at any $\vec{\theta}$. At this point one needs to merely calculate a vector $C_0(\vec{\theta}_a, \vec{\theta})$ for $a = 1 - N_{\rm train}$, then contract the vector with χ to obtain $Y(\vec{\theta})$ or contract with B^{-1} to find the squared emulator uncertainty.

III. CHOOSING THE OPTIMUM TRAINING POINTS

For emulating a convergent function, optimizing one's choice of training points involves choosing points that allow accurate determination of the lowest-order expansion coefficients. If one has $N_{\rm pars}$ model parameters, one needs at least $N_{\rm train} = N_{\rm pars} + 1$ points to fit all the up-to-order-one coefficients, i.e. the slope in $N_{\rm pars}$ dimensions and the intercept. To obtain the coefficients up to second order, one would need at least $(N_{\rm pars} + 1)(N_{\rm pars} + 2)/2$ training values. Thus, it might seem natural to choose a set of either $N_{\rm pars} + 1$ or $(N_{\rm pars} + 1)(N_{\rm pars} + 2)/2$ training points with optimized positions.

One would like to find an algorithm than optimizes the constraining power of the training points for arbitrary priors, for an arbitrary numbers of model parameters, and for an arbitrary number of training points. This needs to be performed before the full model is run, and thus before any of the training values, y_a, are known. This means that the hyper-parameters, Λ and σ_A, are also as yet undetermined. In this section, algorithms are presented that accomplish this goal, but as will be shown, depend on an estimate for Λ. Given Λ, an optimized placement of the training points is found. Fortunately, the placement will not be strongly dependent on Λ.

An accuracy metric is defined based on the expected squared uncertainty averaged over the prior. If the priors are all some combination of Gaussian or uniform forms, analytic expressions are found for this metric. Although these expressions are rather messy, they enable a rapid numerical optimization of the position of the training points. The next subsection presents the analytic expressions for the accuracy metric. The analytic forms require that prior can be expressed in terms of a product of either uniform or Gaussian priors for each model parameter. The next subsection provides a case study of how the accuracy behaves with the number of training points, with the number of parameters, and as a function of the estimated convergence parameter. For context, results are compared to Latin hyper-cube sampling [48].

Certain numbers of training points allow the placement to satisfy symmetries of the prior, if they exist. Such configurations might be organizing the points at the corners of a cube if the $N_{\rm train} = 2^{N_{\rm pars}}$, or perhaps adding an additional point at the center. For spherically symmetric priors with $N_{\rm train} = N_{\rm pars} + 1$, one might place the points in a simplex. The additional improvement for these specific values is rather modest, and is discussed in Sec. III C.

A. Expressing the Overall Emulator Accuracy

To optimize the location of the training points, we define the accuracy metric as being proportional to the emulator's uncertainty squared, $\sigma_Y^2(\vec{\theta})$, averaged over the prior, $P(\vec{\theta})$, then divided by σ_A^2. One might, or might not, be able to make solid arguments that averaging σ_Y to some other power might be more appropriate. This choice has the benefit of allowing much of the averaging to be performed analytically, and seems to provide a reasonable balance of weighing areas with larger uncertainty relative to areas with lower uncertainty.

The goal is thus to calculate $\langle\sigma_Y^2\rangle$, with the averaging referring to the average over the prior. Because the hyper-parameters are not yet determined at the time when the training points are chosen, the goal will be to minimize $\langle\sigma_Y^2\rangle$ for some assumed values of σ_A and Λ.

$$\begin{aligned}\langle\sigma_Y^2\rangle &\equiv \int d\vec{\theta}P(\vec{\theta})\sigma_Y^2(\vec{\theta}) \\ &= \int d\vec{\theta}P(\vec{\theta})\left\{\sigma_A^2\left[1 - C_0(\vec{\theta},\vec{\theta}_a)B_{ab}^{-1}C_0(\vec{\theta}_b,\vec{\theta})\right] + \left(\frac{\partial Y}{\partial\Lambda}\right)^2 W_{\Lambda\Lambda}\right\}.\end{aligned} \tag{71}$$

The expressions from Sec. (II) for $W_{\Lambda\Lambda}$ and $(\partial Y/\partial\Lambda)^2$ in Eqs. (61) and Eq. (64) depend on the training values through the product $y_a y_b$. Given that the positions of the training points are not yet known, the quantities $\langle y_a y_b\rangle/\sigma_a^2$ must be replaced with their unconstrained expectation,

$$\frac{\langle y_a y_b\rangle_0}{\sigma_A^2} \rightarrow B_{ab} = \left[C_0(\vec{\theta}_a,\vec{\theta}_b) + \alpha^2\delta_{ab}\right]. \tag{72}$$

The expressions for W, which are independent of $\vec{\theta}$, then become

$$\begin{aligned}W_{\sigma_A\sigma_A}^{-1} &= \frac{2N_{\rm train}}{\sigma_A^2}, \\ W_{\sigma_A\Lambda}^{-1} &= \frac{-1}{\Lambda^3\sigma_A}{\rm Tr}(B'B^{-1}), \\ W_{\Lambda\Lambda}^{-1} &= +\frac{1}{2\Lambda^6}{\rm Tr}(B'B^{-1}B'B^{-1}).\end{aligned} \tag{73}$$

One needs to invert W^{-1} to find $W_{\Lambda\Lambda}$, which will be independent of σ_A.

The factor $(\partial_\Lambda Y)^2$ depends on $\vec{\theta}$ and becomes

$$\begin{aligned}(\partial_\Lambda Y)^2(\vec{\theta}) = &\frac{\sigma_A^2}{\Lambda^6}\sum_{ab}\left\{C_0(\vec{\theta}_b,\vec{\theta})|\vec{\theta}_b-\vec{\theta}|^2|\vec{\theta}-\vec{\theta}_a|^2 C_0(\vec{\theta},\vec{\theta}_a)B_{ab}^{-1}\right\} \\ &- \frac{\sigma_A^2}{\Lambda^6}\sum_{ab}\left\{C_0(\vec{\theta}_b,\vec{\theta})|\vec{\theta}-\vec{\theta}_a|^2 C_0(\vec{\theta},\vec{\theta}_a)(B^{-1}B'B^{-1})_{ab}\right\} \\ &- \frac{\sigma_A^2}{\Lambda^6}\sum_{ab}\left\{C_0(\vec{\theta}_b,\vec{\theta})|\vec{\theta}-\vec{\theta}_b|^2 C_0(\vec{\theta},\vec{\theta}_a)(B^{-1}B'B^{-1})_{ab}\right\} \\ &+ \frac{\sigma_A^2}{\Lambda^6}\sum_{ab}\left\{C_0(\vec{\theta}_b,\vec{\theta})C_0(\vec{\theta},\vec{\theta}_a)(B^{-1}B'B^{-1}B'B^{-1})_{ab}\right\}.\end{aligned} \tag{74}$$

The expression for σ_Y^2 above has a simple dependence w.r.t σ_A. Every term is proportional to σ_A^2. In contrast, the Λ dependence is complicated. To optimize the positions of the training points, $\vec{\theta}_{a=1\cdots N_{\rm train}}$, one must minimize the r.h.s. of Eq. (71). Unfortunately, without any input from full-model runs, one can only guess at Λ. Before training all values of σ_A^2 are considered equally likely. Thus, because of the simple proportionality to σ_A^2, one can choose any σ_A and it will not affect the choice of training-point locations to minimize $\langle\sigma_Y^2\rangle$. Thus, as a metric for the expected accuracy, we will choose

$$\text{Accuracy metric} = \frac{\langle\sigma_Y^2\rangle}{\sigma_A^2}, \tag{75}$$

which is independent of σ_A and depends only on Λ and on the location of the training points. The goal will be to find locations of the training points that minimize the metric after guessing at a value for Λ.

Once Λ is chosen, one can integrate Eq. (71) over the prior to calculate the accuracy. Because the matrices, B, B' and B'', defined in Eq.s (40) and (60), depend only on Λ and on the locations of the training points, one can express the metric in terms of the integrals, I_{ab}, J_{ab} and K_{ab} defined as:

$$\begin{aligned}
\frac{\langle\sigma_Y^2\rangle}{\sigma_A^2} &= \mathrm{Tr}\left(\mathbb{I} - IB^{-1}\right) + \frac{W_{\Lambda\Lambda}}{\Lambda^6}\mathrm{Tr}\left\{KB^{-1} + JB^{-1}B'B^{-1} + IB^{-1}B'B^{-1}B'B^{-1}\right\}, \quad (76)\\
I_{ab} &\equiv \int d\vec{\theta}\; P(\vec{\theta})C_0(\vec{\theta}_a,\vec{\theta})C_0(\vec{\theta},\vec{\theta}_b),\\
J_{ab} &\equiv J_{ab;a} + J_{ab;b},\\
J_{ab;a} &\equiv \int d\vec{\theta}\; P(\vec{\theta})C_0(\vec{\theta}_a,\vec{\theta})|\vec{\theta}_a - \vec{\theta}|^2 C_0(\vec{\theta},\vec{\theta}_b),\\
K_{ab} &\equiv \int d\vec{\theta}\; P(\vec{\theta})C_0(\vec{\theta}_a,\vec{\theta})|\vec{\theta}_a - \vec{\theta}|^2|\vec{\theta} - \vec{\theta}_b|^2 C_0(\vec{\theta},\vec{\theta}_b).
\end{aligned}$$

Here, a and b run from 1 to $N_{\rm train}$ The $N_{\rm train}\times N_{\rm train}$ matrices I, J and K carry all the relevant information about the prior.

If the prior $P(\vec{\theta})$ can be written as product of priors for individual model parameters,

$$P(\vec{\theta}) = \prod_i P_i(\theta_i), \quad (77)$$

one can express the integrals over the prior in terms of integrals over individual parameters.

$$\begin{aligned}
I_{ab} &= \prod_i I_{ab;i}, \quad (78)\\
J_{ab} &= I_{ab}\sum_i \frac{J_{ab;ai} + J_{ab;bi}}{I_{ab;i}},\\
K_{ab} &= \frac{J_{ab;a}J_{ab;b}}{I_{ab}} + I_{ab}\sum_i\left\{\frac{K_{ab;i}}{I_{ab;i}} - \frac{J_{ab;ai}J_{ab;bi}}{I^2_{ab;i}}\right\}.
\end{aligned}$$

Here, the integrals over single parameters are:

$$\begin{aligned}
I_{ab;i} &= \int d\theta_i\; P_i(\theta_i)\exp\left\{-\frac{(\theta_i-\theta_{ai})^2}{2\Lambda^2} - \frac{(\theta_i-\theta_{bi})^2}{2\Lambda^2}\right\}, \quad (79)\\
J_{ab;ai} &= \int d\theta_i\; P_i(\theta_i)(\theta_i-\theta_{ai})^2\exp\left\{-\frac{(\theta_i-\theta_{ai})^2}{2\Lambda^2} - \frac{(\theta_i-\theta_{bi})^2}{2\Lambda^2}\right\},\\
K_{ab;i} &= \int d\theta_i\; P_i(\theta_i)(\theta_i-\theta_{ai})^2(\theta_i-\theta_{bi})^2\exp\left\{-\frac{(\theta_i-\theta_{ai})^2}{2\Lambda^2} - \frac{(\theta_i-\theta_{bi})^2}{2\Lambda^2}\right\}.
\end{aligned}$$

After calculating the integrals $I_{ab;i}$ analytically, the integrals $J_{ab;ai}, J_{ab;bi}$ and $K_{ab;i}$ can all be generated by taking derivatives of $I_{ab;i}$ w.r.t. $\gamma = 1/\Lambda^2$.

One also needs an expression for W. This is found in the same way as for Eq.s (61). In those expressions $W_{\rm inv}$ was defined in terms of B_{ab}, where B_{ab} was expressed in terms of the training values through the matrix $y_a y_b$, whereas in this case B_{ab} is estimated from the pre-constrained correlation, defined in Eq. (72).

1. Analytic expressions for Gaussian and uniform priors

If a specific model parameter, θ_i, has a Gaussian prior,

$$P_i(\theta = \theta_i) = \frac{1}{(2\pi/\beta_i)^{1/2}}e^{-\beta_i\theta^2/2}, \quad (80)$$

one can integrate Eq.s (79) and find

$$
\begin{aligned}
I_{ab;i} &= \sqrt{\frac{\beta_i}{\lambda_i}} \exp\left\{-\frac{1}{2\lambda_i}[\gamma^2(\theta_a-\theta_b)^2 + \beta_i\gamma(\theta_a^2+\theta_b^2)]\right\}, \\
J_{ab;ai} &= I_{ab;i}\left\{\frac{1}{\lambda_i} + |\vec{\theta}_a - \vec{\theta}_b|^2 \frac{\gamma^2+\beta_i\gamma}{\lambda_i^2} + (\theta_a^2+\theta_b^2)\frac{\beta_i^2}{2\lambda_i^2} + \frac{\beta_i}{2\lambda_i}(\theta_a^2-\theta_b^2)\right\}, \\
J_{ab;bi} &= I_{ab;i}\left\{\frac{1}{\lambda_i} + |\vec{\theta}_a - \vec{\theta}_b|^2 \frac{\gamma^2+\beta_i\gamma}{\lambda_i^2} + (\theta_a^2+\theta_b^2)\frac{\beta_i^2}{2\lambda_i^2} - \frac{\beta_i}{2\lambda_i}(\theta_a^2-\theta_b^2)\right\}, \\
K_{ab;i} &= \frac{J_{ab;ai}J_{ab;bi}}{I_{ab;i}} + I_{ab;i}\left\{\frac{2}{\lambda_i^2} + 2\beta_i^2(\theta_a^2+\theta_b^2)/\lambda_i^3 - (4\gamma^2+2\beta_i\lambda_i)(\theta_a-\theta_b)^2/\lambda_i^3\right\},
\end{aligned}
\tag{81}
$$

where $\lambda_i \equiv 2\gamma + \beta_i$ and $\gamma = 1/\Lambda^2$.

Similarly, one can perform the integrals for uniform priors. In this case the prior is

$$
P_i(\theta = \theta_i) = \begin{cases} \frac{1}{2\theta_{\max,i}}, & -\theta_{\max,i} < \theta < \theta_{\max,i} \\ 0, & \text{otherwise} \end{cases}
\tag{82}
$$

After making the definitions

$$
\begin{aligned}
\bar{\theta} &\equiv \frac{\theta_a+\theta_b}{2}, \qquad (83) \\
\Delta\theta &\equiv \frac{\theta_a-\theta_b}{2}, \\
\theta_+ &\equiv \theta_{\max,i} - \bar{\theta}, \\
\theta_- &\equiv -\theta_{\max,i} - \bar{\theta}, \\
p &\equiv \frac{1}{2\theta_{\max,i}} e^{-\gamma\Delta\theta^2}, \qquad (84) \\
q &\equiv \frac{1}{2}\sqrt{\pi\gamma}\left[\mathrm{erf}(\sqrt{\gamma}\theta_+) - \mathrm{erf}(\sqrt{\gamma}\theta_-)\right] \\
r &\equiv \theta_+ e^{-\gamma\theta_+^2} - \theta_- e^{-\gamma\theta_-^2},
\end{aligned}
$$

the integrals of interest are

$$
\begin{aligned}
I_{ab;i} &= pq, \\
J_{ab;ai} &= (\frac{1}{2\gamma} - \Delta\theta^2)I_{ab;i} - \frac{1}{2\gamma}pr + pe^{-\gamma\theta_+^2}\frac{\Delta\theta}{\gamma} - Pe^{-\gamma\theta_-^2}\frac{\Delta\theta}{\gamma}, \\
J_{ab;bi} &= (\frac{1}{2\gamma} - \Delta\theta^2)I_{ab;i} - \frac{1}{2\gamma}pr - pe^{-\gamma\theta_+^2}\frac{\Delta\theta}{\gamma} + Pe^{-\gamma\theta_-^2}\frac{\Delta\theta}{\gamma}, \\
K_{ab;i} &= \frac{1}{2}(J_{ab;ai} + J_{ab;bi})\left(\frac{1}{2\gamma} + \Delta\theta^2\right) \\
&\quad + I_{ab;i}\left(\frac{1}{2\gamma^2} - \frac{2\Delta\theta^2}{\gamma}\right) \\
&\quad - pe^{-\gamma\theta_+^2}\theta_+\left(\frac{1}{2\gamma^2} - \frac{3\Delta\theta^2}{2\gamma} + \frac{\theta_+^2}{2\gamma}\right) \\
&\quad + pe^{-\gamma\theta_-^2}\theta_-\left(\frac{1}{2\gamma^2} - \frac{3\Delta\theta^2}{2\gamma} + \frac{\theta_-^2}{2\gamma}\right).
\end{aligned}
\tag{85}
$$

For most of the results presented in this section Gaussian priors were considered.

B. Dependence on the number of parameters, the number of training points and the convergence parameter

The accuracy metric, defined in Eq. (75), was optimized by performing a simple search through the parameter space. For some configuration of training points $\langle\sigma_Y{}^2\rangle/\sigma_A^2$ was calculated. A new set of training points was then chosen by taking a small random step in parameter space for each training points. If the new uncertainty was reduced,

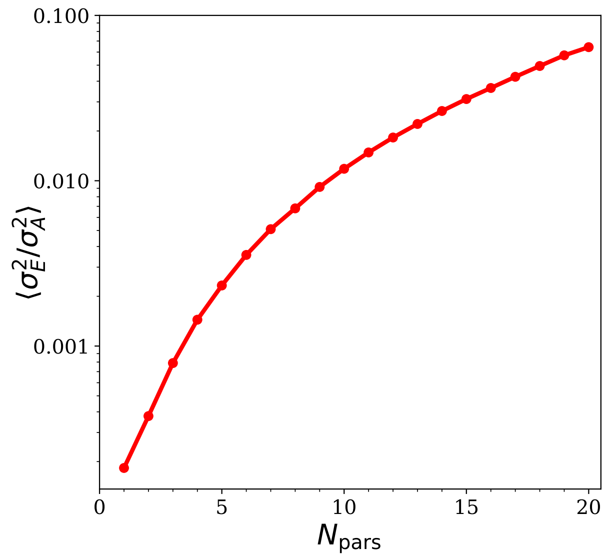

FIG. 1. The accuracy metric as a function of the number of model parameters assuming Gaussian priors. The convergence parameter was set to $\Lambda = 3$, the point-by-point noise was set to $\alpha = 0.01$ and the number of training points was set to match the number of expansion coefficient of quadratic order or less. This illustrates the difficulty to accurately emulate higher dimensional functions even if the number of training points increases as $N_{\rm train} = (N_{\rm pars}+1)(N_{\rm pars}+2)/2$.

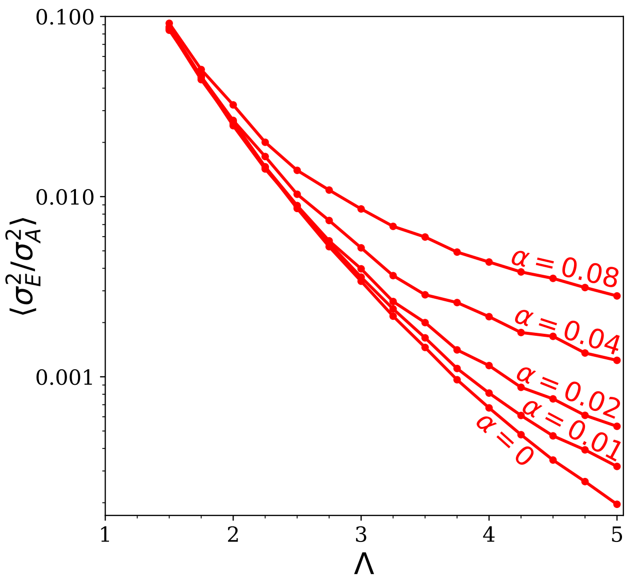

FIG. 2. The expected accuracy of the emulator falls rapidly with the convergence parameter Λ. For this illustration $N_{\rm pars} = 6$ and the number of training points was set to 28, the number of expansion coefficients of order $n = 2$ or less. Calculations were repeated for different values of the point-by-point noise, α. As expected, the impact of the point-by-point noise sets in as the accuracy of the emulator improves.

the step was accepted. This is by no means a particularly efficient method, but the method was sufficiently fast for $N_{\rm pars} \lesssim 10$, as the optimized values were found within a few minutes or less. Larger numbers of parameters required increasing amounts of patience.

The difficulty in emulating high-dimension parameter spaces increases more than linearly with the number of model parameters. This is because the number of expansion coefficients of order K is

$$\begin{aligned}
N_{\rm coefficients}(K=0) &= 1,\\
N_{\rm coefficients}(K=1) &= N_{\rm pars},\\
N_{\rm coefficients}(K=2) &= N_{\rm pars}(N_{\rm pars}+1)/2,\\
N_{\rm coefficients}(K=3) &= N_{\rm pars}(N_{\rm pars}+1)(N_{\rm pars}+2)/6,\\
&\vdots
\end{aligned} \tag{86}$$

Further, the number of coefficients of order less than or equal to K, is

$$\begin{aligned}
N_{\rm coefficients}(K\le 0) &= 1,\\
N_{\rm coefficients}(K\le 1) &= N_{\rm pars}+1,\\
N_{\rm coefficients}(K\le 2) &= (N_{\rm pars}+1)(N_{\rm pars}+2)/2,\\
N_{\rm coefficients}(K\le 3) &= (N_{\rm pars}+1)((N_{\rm pars}+2)(N_{\rm pars}+3)/6,\\
&\vdots
\end{aligned} \tag{87}$$

Thus, the number of terms of order $K+1$ is larger than the number of terms of order K by a factor of $(N_{\rm pars}+K)/K$. If the number of parameters is large, there are large numbers of parameters of higher order. For example for $N_{\rm pars} = 12$ there are 364 cubic terms, and for $N_{\rm pars} = 20$ there are 1540 cubic terms. Because the importance of higher order terms falls as $1/\Lambda^{2n}$, emulating high-dimensional functions requires larger values of the convergence parameter to arrive at the same accuracy. Figure 1 shows the accuracy metric as a function of the number of parameters for fixed $\Lambda = 3$ and for a number of training points set to correspond to the number of expansion parameters of order two or less, $N_{\rm train} = (N_{\rm pars}+1)(N_{\rm pars}+2)/2$.

It should be emphasized that this measure of the uncertainty is the proportional to $\langle \sigma_Y^2 \rangle$, not to $\langle \sigma \rangle$. Thus, to get to 10% accuracy, one might want $\langle \sigma_Y^2 \rangle / \sigma_A^2$ to be close to 0.01.

If Λ is higher, it is easier to emulate functions because the contribution to $\langle\sigma^2\rangle$ of terms of order n falls roughly as $1/\Lambda^2$. Figure 2 shows how the expected accuracy falls as a function of Λ. For this illustration the number of parameters was set to 6 and the number of training points was again set to correspond to the number of expansion coefficients of order $n = 2$ or less. Figure 2 also illustrates the dependence on the point-by-point noise α. The effect of α only sets in when the accuracy metric falls to near α^2. However, given that there are multiple training points are considered the expected uncertainty can fall below α^2.

The accuracy should increase with more training points. This is illustrated for the case of six model parameters with $\Lambda = 3.0$ and $\alpha = 0.01$ in Fig. 3. The behavior underscores the value of choosing a number of training points corresponding to the number of independent coefficients for a linear, quadratic or cubic expansion. For six model parameters, those numbers of coefficients are $N_{\rm train} = 7$, 28 and 84 respectively. Once one of these thresholds is passed, the marginal value of an additional training point decreases. Then, as one approaches the next threshold, the marginal value increases again, though not to the degree of those points below the previous threshold.

The principal source of structure, or kinks, in Fig. 3 is due to whether the number of training points passes one of the thresholds mentioned above. However, there is a second less prominent source of structure that occurs when the the number of training points corresponds to a symmetry. Given that this example was for Gaussian priors for each parameter, with the same widths (relative to Λ) for each prior, the prior had a spherical symmetry. Examples of arrangements with symmetries are simplexes and cubes. These might also include an extra point at the origin. Such structures vary with dimensionality. For this case, $N_{\rm pars} = 6$, the uncertainty for $N_{\rm train} = 8$ appears noticeably less. On inspection, this turned out to be a simplex with an extra point at the center. Thus, one might well choose $N_{\rm train} = 8$, even though it is one above the threshold. A smaller kink is seen at $N_{\rm train} = 44$. But this structure is not so prominent as to drive one's decision to run more points.

For many cases, one finds that the minimization procedure might settle on a false minimum. The searches were initialized by either randomly placing the training points according to the prior weights, or using Latin hyper-cube samples [48]. For Latin hyper-cube sampling each dimension of parameter space was divided into $N_{\rm train}$ equally probable slices. For each parameter, the value for each training point was set randomly forced to lie in a different slice. Figure 4 shows how the accuracy metric improved with the number of random steps. Several chains are displayed to illustrate the progress of the search, and its dependence on the initial configuration. Settling on a false minimum can affect the estimated uncertainty by at most a few percent, and can lead to a bit of jitter for the optimized values. One can find the true minimum by repeating the minimization search several times, then choosing the best configuration. The number of random steps required to reach a desired accuracy increases with dimensionality of the model parameter space. The simple method applied here seems fine for cases where $N_{\rm pars}$ is near or less than a dozen, but if one had two dozen parameters, it might warrant a more sophisticated search algorithm.

Also shown in Figs. 3 and 4 are the accuracy metrics for latin hyper-cube sampling. As this is not an optimized choice, there is more noticeable jitter in the curves. One can see that the optimization procedure reduces the expected uncertainty by nearly a factor of two at the quadratic threshold and by nearly a factor of five at the cubic threshold. Depending on the desired accuracy, one might need 50% more runs with Latin hyper-cube sampling to achieve the same accuracy as would be obtained with an optimized placement. Figure 4 also illustrates the improvement compared to random placement of the training points. In that case the training points were placed independently, according to the prior, with no correlation between points.

While performing the calculations of $\langle\sigma_Y^2\rangle/\sigma_A^2$ considered here, the contributions to the uncertainty can be split into two pieces as expressed in Eq. (76). Whereas the first term in Eq. (76) represented the uncertainty assuming Λ is fixed at its most probable value, the second term represents the emulator uncertainty coming from Λ varying, with the weight Z. Remarkably, this second contribution was never more than a few percent of the total expected uncertainty for the optimized placement, which makes it appear largely negligible. However, that fraction was often much larger when the optimization procedure was first started, perhaps a third of the uncertainty. Then as the training-point placement approached the optimal arrangement, this second contribution shrunk. It was observed that the first contribution, the part assuming Λ was fixed, varied rather little with the training point placement. Thus, the second contribution was more instrumental in driving the placement of the training points, despite the fact that it is the lesser contribution. This would suggest that while this contribution is crucial for choosing the location of the training points, but that once that optimization is completed, and one has built an emulator, that contribution can be largely neglected when estimating the emulator's uncertainty.

C. Fine Structure of Point Placement

Here, fine structure refers to the fact that certain numbers of training points allow the placement to satisfy symmetries of the prior. If the prior is Gaussian in all directions, and is rotationally symmetric, then one might expect the optimum choice to exploit the symmetry. For the case of $N_{\rm train} = N_{\rm pars} + 1$, one can place the training points in

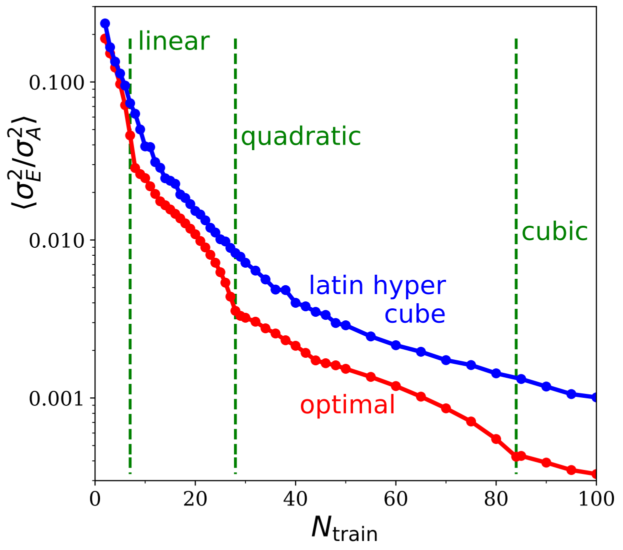

FIG. 3. For all Gaussian priors, with $N_{\rm pars}=6$, $\Lambda=3$, and $\alpha=0.01$, the expected mean accuracy falls with increasing the number of training points. The minimum number of training points required for linear, quadratic and cubic fits are 7, 28 and 82. After passing one of these threhsolds the marginal improvement for additional training points decreases. For $N=8$, one can place the points in a simplex with an additional point at the center, and the symmetry of this configuration provides an especially large improvement. This behavior might inspire the modeler to choose a number of training points equal to one of the thresholds. The optimized placement significantly improves compared to Latin hyper-cube sampling.

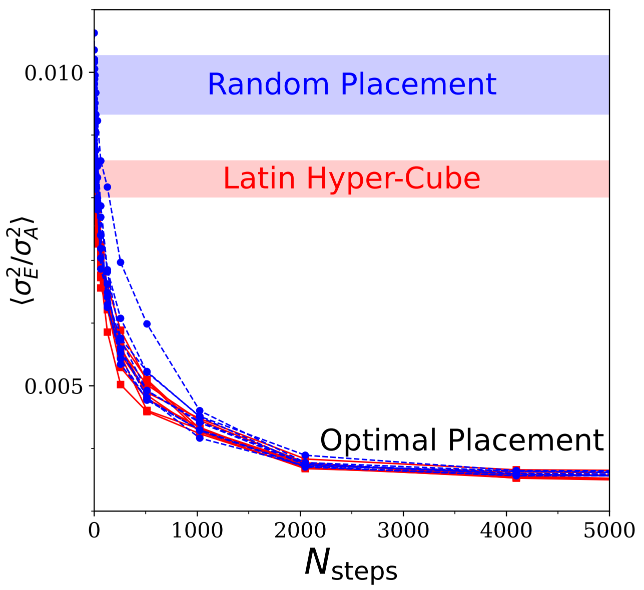

FIG. 4. The search procedure provides most of the improvement for the first few thousand steps. For this example, $N_{\rm pars}=6$, $\Lambda=3$ and $\alpha=0.01$ the number of training points was set to 28. Several initial configurations were considered. Those where the points were placed randomly (blue circles) tended to start at roughly three times the optimized value, whereas those that were initialized with Latin hyper-cube sampling tended to start at a bit less than double the optimized uncertainty. The bands roughly represent the spread of initial accuracies, and the fall from the bands to the optimal values represents the improvement one expects to gain through the optimization procedure.

an $N_{\rm pars}$ dimensional simplex, a simplex being the placement of all points at the same radius, with the same distance between each pair. In two dimensions a simplex is an equilateral triangle, while for three dimensions the simplex is a tetrahedron. With this placement, there are a set of rotations under which each point can be rotated to another point in the structure without changing the structure. A few other arrangements may have similar symmetries. For example, one could have $N_{\rm pars}+1$ training points placed in a tetrahedron, while one additional point might be placed in the center. For three dimensions, a buckyball configuration with $N_{\rm train}=60$ has a symmetry. But these symmetries are hard to come by, so in general it is not at all obvious how to place training points in an arbitrary number of dimensions, unless the number of training points happens to match some specific value. Further, even if the number of training points were to match that of a simplex, or of a simplex plus one extra point, one would have to determine the optimum radius of the simplex, which would necessitate a numerical search. Finally, in most cases, not all the model parameters have Gaussian priors, and if they were Gaussian, they might not have identical spreads. Another such symmetry is that where points are placed at the corners of an $N_{\rm pars}-$dimensional hyper-cube. This would require $2^{N_{\rm pars}}$ points. Again, another point might be added to the center. This arrangement would seem relevant for either the case where the prior was purely Gaussian with a spherical symmetry, or if all the priors were uniform with the same widths.

Thus, unless one has a symmetry, and additionally if the number of training points happens to allow placement according to that symmetry, the placement of the points is difficult to predict. The radius of a training point is simply the point's distance from the origin,

$$|\vec{\theta}_a|=\left(\sum_{i<N_{\rm pars}}\theta_{a,i}^2\right)^{1/2}. \tag{88}$$

For the case with $N_{\rm pars}=12$ parameters, Fig. 5 shows the radii of the various positions. For two cases, all the priors were Gaussian, with an r.m.s. radius of $1/\sqrt{3}$. The number of training points was set to $N_{\rm train}=91$, which corresponds to the quadratic-fit threshold for $N_{\rm pars}=12$. As can be seen the optimized placement is for most of the

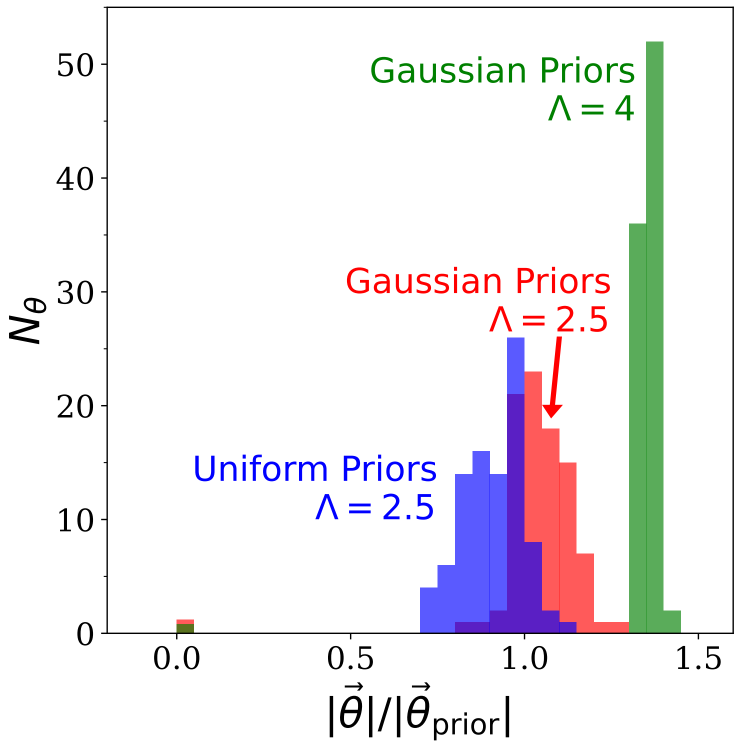

FIG. 5. The distribution of the 91 optimized radii, $|\vec{\theta}_a|$, for the optimized placement of 91 training points in the 12-dimensional model-parameter space. For this case $N_{\rm train} = 91$. If all model parameters had Gaussian priors with the same width relative to Λ, the radii tend to organize with one point in the center while the others have similar radius spread over different directions. For larger Λ (Λ = 4,green) larger radii were preferred compared to smaller Λ (Λ=2.5,red). For uniform priors (Λ = 2.5,blue) radii were smaller with a similarly broad distribution, and there was not training point in the middle.

points to be a nearly the same radius, whereas one point is placed near the origin. When this same example was worked out for $N_{\rm pars} = 6$, one point was at the center, while the others were all positioned at exactly the same radius, but for $N_{\rm pars} = 12$ there was some small variation in the radii of the points away from the center. Two values of Λ were considered in Fig. 5, $\Lambda = 2.5$ and $\Lambda = 4.0$. One can see that the radii moved outward for larger Λ. Figure 5 also shows the radii for 12 model parameters where each parameter had a uniform prior with minimum and maximum values of -1 and $+1$. That distribution is at somewhat smaller radius, and without a point placed at the origin.

D. Overcoming the Pernicious Nature of High Dimensions

The overarching goal of this manuscript is to understand how to best emulate functions that can be described well with polynomials not extending to high order, or equivalently, are well described by Taylor expansions. In Sec. II it was described how this can equivalently be posed in terms of Gaussian process approaches by requiring that the correlation length to be larger than the size of the prior. Either way, emulation depended on defining a characteristic scale, Λ. For smaller Λ, more training points are needed. In high dimensionality parameter spaces, it becomes more difficult to effectively cover the prior. In Fig. 1, it was shown that even if the number of training points increased with number of quadratic coefficients, the accuracy of the emulator still fell as a function of $N_{\rm pars}$. The discussion here aims to provide several perspectives from which can appreciate the challenge of building high-dimensional emulators.

As discussed earlier and shown with Eq.s (86) and (87), the number of polynomial terms of order $n+1$ exceeds the number of terms of order n by a factor proportional to the dimensionality of the parameter space, $N_{\rm pars}$. The variance due to the terms of order $n+1$ compared to those of order n roughly grows a $N_{\rm pars}/\Lambda^2$. One can roughly state that the value of Λ that is necessary to give a certain accuracy should increase as $\sqrt{N_{\rm pars}}$. Thus, accurate emulation will be more difficult in higher dimensions as illustrated in Fig. 1. Another perspective from which one can gain the same conclusion is to consider the characteristic size of the parameter space. The characteristic r.m.s. sizes increase as $N_{\rm pars}^{1/2}$. The unconstrained correlation behaves as $e^{-(|\vec{\theta}_i-\vec{\theta}_j|)^2/2\Lambda^2}$, so to maintain the same characteristic correlations, Λ^2 must be increased proportional to $N_{\rm pars}$, or again, the Λ must increase as $\sqrt{N_{\rm pars}}$.

In higher dimensions it is increasingly critical that the convergence parameter be longer than the system. Otherwise, one would need to sample the function in each corner of the model-parameter phase space. For uniform priors, there are $2^{N_{\rm pars}}$ corners to the phase space. For 12 dimensions this requires 4096 training points, and for 18 dimensions, 262,144 training points. Thus, it can be untenable to place training points in each corner of the space.

One can also see the problem with placing the training points in a spherically symmetric fashion. The hyper-volume of the parameter-space hyper-cube is 2^N, whereas the volume of an $N-$dimensional hyper-sphere of radius $R=1$ is

$$V_{\rm sphere} = \Omega_N \int_0^R dr\ r^{N-1} = \Omega_N \frac{R^N}{N}. \tag{89}$$

The solid angle, Ω_N in N dimensions is

$$\Omega_N = \frac{2\pi^{N/2}}{\Gamma(N/2)}, \tag{90}$$

and after putting this together, the fraction of the hyper-cube's volume that is within the hyper-sphere is

$$\frac{V_{\rm sphere}}{V_{\rm cube}} = \begin{cases} \frac{(\pi/2)^{N/2}}{N!!}, & N=2,4,6,8\cdots \\ \frac{(\pi/2)^{(N-1)/2}}{N!!}, & N=3,5,7,\cdots \end{cases} \tag{91}$$

In two dimensions, the ratio is $\pi/4$, and in three dimensions it is $\pi/6$. In 10 dimensions it is 2.5×10^{-3}. For high dimensions only a small fraction of the parameter space can ever lie inside a sphere used to place points. And, if the model is expensive, it may not be tenable to run the full model inside every corner.

All the perspectives above emphasize the same conclusion. The only way a high dimensional space can be well emulated is if the behavior is highly convergent. The parameter Λ must be large compared to the characteristic size, which is $\sqrt{N_{\rm pars}/3}$. For 12 parameters this suggests $\Lambda > 2$, and for 18 parameters this suggests $\Lambda > 2.45$. If Λ is near or below this limit, emulation may require a large number of training points.

For large numbers of model parameters it may behoove the user to consider whether some model parameters are more or less important. As stated in the beginning of Sec. **??** this can be accomplished by scaling the model parameters. The model parameters with uniform priors can be scaled such that their priors have defined by $\theta_{{\rm max},i}=s_i$, where $s_i<1$ can be thought of as a suppression factor. The r.m.s. of the prior is then $s_i/\sqrt{3}$. The parameters with Gaussian priors can be scaled so the r.m.s. of their variance is $s_i/\sqrt{3}$. Scaling the range of the priors by s_i is mathematically equivalent to scaling the convergence parameters, i.e. setting $\Lambda_i=\Lambda/s_i$. Scaling the priors, rather than scaling the convergence parameters, allows emulation to be simpler, as there is then only one convergence parameter. One could scale the parameters so that the most important parameters had $\theta_{{\rm max},i}=1$, or $\beta_i=3$. One would set Λ to a value appropriate for those parameters, then scale $\theta_{{\rm max},i}$ and β_i for the remaining parameters to account for the suppression of the expected sensitivity of $y(\vec{\theta})$ to the other parameters.

If warranted, assigning suppression factors could be an effective means of dealing with high dimensionality. For example if $\Lambda=2$, it would be difficult to emulate a 12-dimensional parameter space. However, if all but a couple of parameters were suppressed with $s_i=1/2$, the less critical parameters would not contribute much beyond their linear contribution. For example, if only three parameters were given full strength, then training-point placement might optimize so that the $2^3=8$ corners of the three-dimensional space were covered. This is much easier than attempting to cover $2^{12}=4096$ corners of parameter space. The software developed for this study was specifically designed to allow the user to assign suppression factors to each model parameter.

One might ask whether certain parameters might be deemed less important after the training data is created. That is certainly possible, and the emulator's uncertainty would fall whenever one would reduce s_i. However, assigning $s_i\neq1$ also affects the optimized choice of training-point location. Thus, if the reduced importance is understood before the training data is generated it can be exploited in the optimization of the training-point placement.

IV. TESTING THE EMULATOR AND TRAINING POINT OPTIMIZATION

A. Overview of Testing

For this study software was developed according to the methods described in the previous sections. The first piece of software generated a list of optimized training points as described in Sec. III. The input for this code is a list of model parameters and their priors, plus an estimate of the convergence parameter, Λ, and the point-by-point noise parameter α. Each model parameter was assumed to have an independent prior, either Gaussian with a given centroid and width, or uniform distribution between some minimum and maximum value. These parameters were then translated and scaled so that their priors were centered around zero. By default, the widths were set according to the definitions in Eqs. (2) and (3) so that the priors would have the same variance as a uniform distribution with $-1 < \theta < 1$ even if the prior distribution is Gaussian. An option was added to allow the width for a given parameter, θ_i, to be scaled down by a factor s_i if one believed a certain direction in parameter space was relatively less important. The software then provides a list of training points, $\vec{\theta}_a$, with $a = 1 - N_{\rm train}$. Some tests of this procedure were provided in the previous section.

The second piece of software was the emulator discussed in Sec. II. The emulator assumed the forms for $Y(\vec{\theta})$ and its uncertainty as described in Eq. (70). This includes the contribution to the emulator uncertainty from the uncertainty in knowing Λ. The goal of this section is to study and test the emulator, and the theoretical foundations of both the emulator and the training-point optimization. The emulator's mean prediction, $Y(\vec{\theta})$, is exactly that of a typical GP emulator with a Gaussian kernel, despite the fact that the form was generated from a different perspective than the usual presentation of GP emulators. Thus, the tests here will focus on the following questions.

1. If a model is built using a polynomial expansion based on specific values of Λ and σ_A, how well will the tuned emulator reproduce those values?
2. In the limits of $\Lambda \gtrsim 2$, can one not only reproduce a model, but can it also provide a meaningful estimate of its uncertainty?
3. Is the contribution to the emulator uncertainty from the uncertainty in Λ significant?
4. Did the estimate of the emulator's mean uncertainty, $\langle\sigma_Y^2\rangle/\sigma_A^2$, from the training point selection before any model runs were processed, match that of the tuned emulator?

A subsection is devoted to each of the four questions above.

In order to test the models, two model types were created. The first type, Type-A, is based on creating polynomials with random coefficients, and the second, Type-B, is a sum of randomized trigonometric and exponential functions. Test functions were thus generated of both types, and for each function an emulator was constructed. By creating thousands of such models, and comparing emulators trained upon them, one can test how accurately the emulator represents its uncertainty. One can also test whether the estimates of accuracy used for the optimization of training-point placement are consistent with the accuracy actually exhibited by the trained emulator.

1. Type-A Model

The form of the expansion is the same as what was assumed when designing the emulator:

$$F(\vec{\theta}) = e^{-|\vec{\theta}|^2/2\Lambda^2} \sum_{\vec{n}} \frac{A_{\vec{n}}}{\sqrt{n_1! n_2! \cdots n_{N_{\rm pars}}!}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_{N_{\rm pars}}}{\Lambda}\right)^{n_{N_{\rm pars}}}. \tag{92}$$

Here, the coefficients $A_{\vec{n}}$ are assumed to have a Gaussian distribution with variance σ_A^2. To create the model, one chooses the dimensionality, $N_{\rm pars}$, the variance σ_A^2 and the convergence parameter Λ. The coefficients $A_{\vec{n}}$ are then randomly set according to a Gaussian distribution, with the maximum order set to $K_{\rm max} = 5$. Because this is the same form assumed by the emulator, aside from limiting $K \leq K_{\rm max}$, one can test whether the emulator, $Y(\vec{\theta})$, will choose optimum values of the hyper-parameters, σ_A and Λ, which match those used to create the full model, $F(\vec{\theta})$. A different choice for the coefficients represented a different model. For this study thousands of Type-A models were considered.

2. Type-B Model

The second functional form was constructed by summing over randomly generated sums of trigonometric and hyperbolic functions. The observable was a sum of six functions, each constructed from six model parameters.

$$F(\vec{\theta}) = D_0\cos(t_0/\Lambda_0) + D_1\sin(t_1/\Lambda_1) + D_2\tan^{-1}(t_2/\Lambda_2) + D_3\cosh(t_3/\Lambda_3) + D_4\sinh(t_3/\Lambda_4) + D_5\tanh^{-1}(s_5), \tag{93}$$

$$s_5 \equiv \frac{t_5}{\sqrt{\Lambda_5^2 + t_5^2}},$$

$$t_i = \sum_j r_{ij}\theta_j/\sqrt{6} + r_{ijk}\theta_j\theta_k/\lambda_{jk}. \tag{94}$$

Here, r_i and r_{ij} and r_{ijk} are random numbers taken from a uniform distribution between ± 1. The values Λ_i and λ_{jk} were random numbers between 2.0 and 5.0. Each of the coefficients D_i were randomly chosen according to a Gaussian distribution with the same variance. Again, thousands of such models were generated to perform tests.

B. Choosing the Hyper-Parameters

In order to test the algorithms from Sec. (II) for choosing the hyper parameters, σ_A and Λ, 10^5 Type-A models were constructed. Each model was constructed using $\sigma_A = 100$ and $\Lambda = 3$. Because the type-A models use the form assumed by the emulator, one might expect the algorithm to choose values of σ_A and Λ that match the values used to generate the models. There should be some fluctuation due to the coefficients for the models having been chosen with random distributions. The distribution of extracted hyper parameters for the 10^5 models is displayed in Fig. 6.

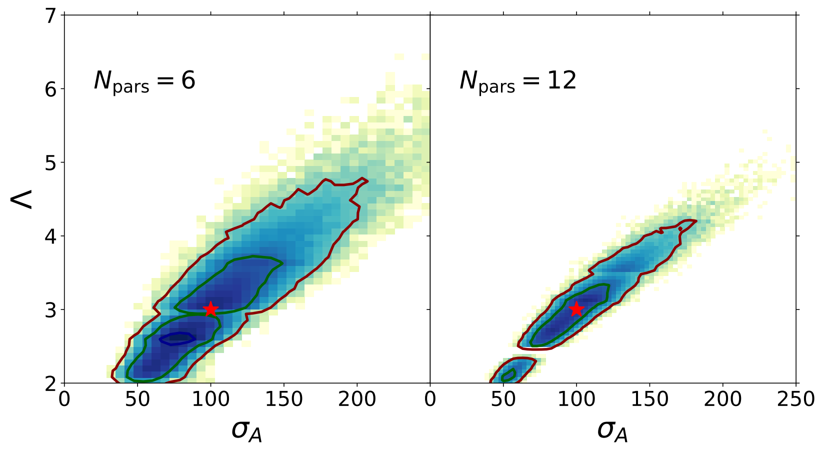

FIG. 6. 10^5 models were constructed based on a expansion with random coefficients chosen according to the form assumed by the emulator using $\Lambda = 3$ and $\sigma_A = 100$. The hyper parameters chosen by the tuning methods from Sec. II F are compared to the values used to build the model (red stars). For each model the extracted hyper-parameters varied significantly, but if one averages the extracted hyper-parameters over the 10^5 models, the average values come within a percent of $\Lambda = 3$ and $\sigma_A = 100$. A strong band structure was found in that if the extracted value of σ_A was low, the value of Λ also tended to be low. Both panels exhibit some bimodal behavior in the distribution.

Distributions were found for two cases, one with 6-parameter models, and one with 12-parameter models. For the 6-parameter models, the emulators were trained at 28 points in parameter space, and for the 12-parameter model tuning involved 91 training points. These numbers correspond to the number coefficients that have $K_{\rm max} < 2$. On average, the hyper parameters well matched those of the model (shown by the red stars), $\sigma_A = 100$ and $\Lambda = 3$. Averaging over the distribution, the extracted hyper parameters matched those used to generate the model to within 1%. However, the distribution was rather wide, with widths of a few tens of percent. A strong correlation was observed between the extracted values of σ_A and Λ, as one can see a strong band structure. Larger Λ is highly correlated with larger σ_A. For both cases, the distributions exhibit some bimodal behavior. Being that the emulator uncertainty is proportional to σ_A and falls with increasing Λ, the overall uncertainty did not vary greatly from one instance to another despite the rather broad range of the distribution. The variance of the extracted hyper parameters decreased for the case with larger numbers of model parameters.

If one extracts small values of Λ, the expansion is poorly converging. For that reason, whenever the extracted value fell below $\Lambda = \sqrt{N_{\rm pars}/3}$, the emulator would set Λ to that value. This was a rare occurrence for this case. A second issue can arise for large Λ, especially for larger numbers of parameters. The tuning procedure involves inverting the matrix B and finding its determinant. The form for B is

$$\begin{aligned} B_{ab} &= C_0(\vec{\theta}_a, \vec{\theta}_b) + \Delta_{ab}, \\ \Delta_{ab} &= \alpha^2 \delta_{ab}. \end{aligned} \tag{95}$$

The term with $\vec{n} = 0$ provides a contribution of unity to each element of the $N_{\rm pars} \times N_{\rm pars}$ matrix. Thus, ignoring the other contributions, one would have $B_{ab} = 1$, and B would be highly singular. Contributions from other $\vec{n}$ behave as $\Lambda^{-|\vec{n}|}$, and keep the matrix from being singular, but numerically, overcoming the singularity is more difficult for large Λ or for higher dimensionality of the $N_{\rm pars} \times N_{\rm pars}$ matrix. Further, some choices of training points can accidentally lead to singular, or nearly singular, matrices. Fortunately, adding point-by-point noise, $\alpha \neq 0$, also eliminates the singularity. The same issues can plague GP emulators. In that case the noise term above is referred to as a "nugget" [44].

C. Predicting The Uncertainty

To address questions (2) and (3) from the beginning of this section, both Type-A and Type-B models were applied. For each model emulators were trained from model values at the training points. The training points were chosen according to method outlined in Sec. III. For this study, the parameter space was 12-dimensional with Gaussian priors. The training involved evaluating the model at 91 optimized points, 91 being the threshold for a quadratic fit. Figure 7 presents this comparison for 500 of these points from a variety of different models, and for a variety of random points in model-parameter space. The emulator's uncertainty for each point is also displayed. The percentage of emulations within one unit of its stated uncertainty should be approximately 68.3% for Gaussian errors. For the Type-A models the success rate was indeed 68%, while for the random assorted functions, the success rate was 88%.

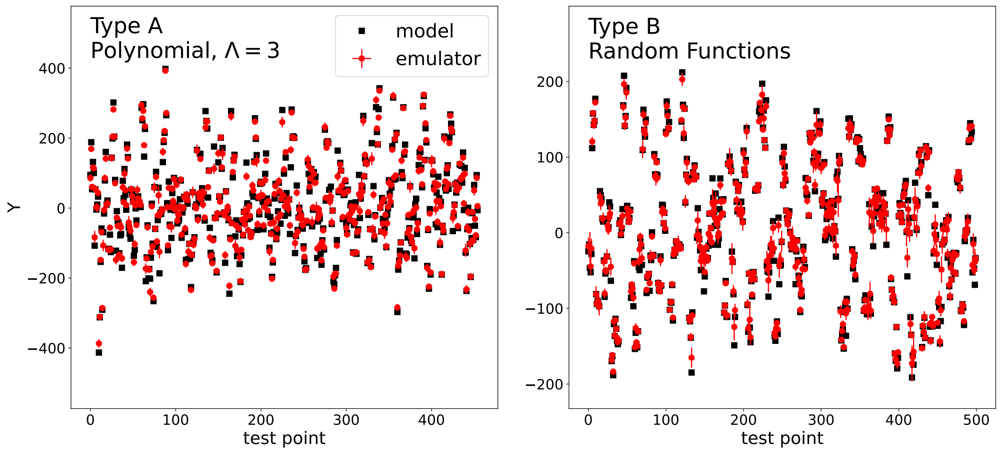

FIG. 7. Emulator predictions and their uncertainties (red circles) are compared to model calculations (black squares) at 500 random points in parameter space. Models involved 12 parameters and were trained at 91 points. For the left-side panel, the model was a polynomial with random coefficients, type-A, chosen with the emulator's assumed form with $\Lambda = 3$, while in the right-side panel the model is a sum of random assorted trigonometric and exponential functions, type-B as described by Eq. (93).

When calculating the uncertainties at the 500 random points used to generate Fig. 7, the contribution to the uncertainty from the uncertainty in determining Λ was included. That contribution, related to term proportional to $W_{\Lambda\Lambda}$ in Eq. (70) was rarely more than a few percent of the total uncertainty. This was consistent with the expectations for that contribution, which were calculated when the choice of training points was being optimized.

D. Testing the Preconstrained Estimates of the Uncertainty

While choosing the training points, and before any full-model runs were performed, estimates were made of the expected emulator's uncertainty averaged over the prior. Figure 9 illustrates the degree to which that prediction was successful. Ten thousand models of type-A were randomly generated using $N_{\rm pars} = 12$ with Gaussian priors. The coefficients were random but chosen with weights assuming $\Lambda = 3$. The distribution's average indeed matched the predicted value, but the width of the distribution is remarkably wide. The bimodality seen in Fig. 6 is also manifest here. An island of fits with smaller σ_A and smaller Λ in Fig. 6 is seen in Fig. 9 as an additional island of values of larger $(\sigma_Y/\sigma_A)^2$. Even though these models tended to have larger values of σ_Y/σ_A, there overall uncertainties were similar to those from the bulk of the distribution because σ_A tended to be smaller. Hence, the distribution of accurate predictions in Fig. 8 was not particularly wide despite the widths of the distributions in Fig. 6 and 9.

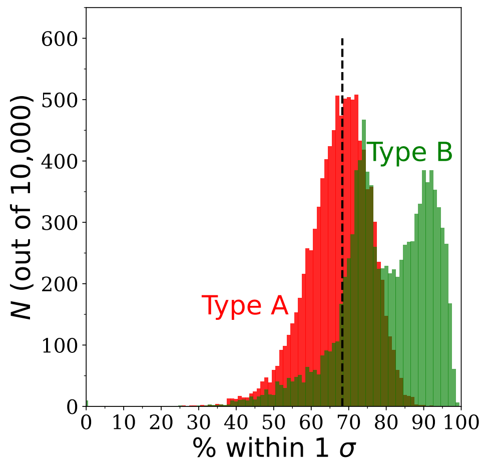

FIG. 8. Ten thousand models were created, each a polynomial expansion with random coefficients chosen to match the form assumed by the emulator. After the emulator was tuned, each model was evaluated at 1000 random points within the prior and compared to the emulator prediction. For each model the percentage was found for the prediction being within $\pm\sigma_Y$. If the emulator were to perfectly estimate its uncertainty, the distribution would be peaked around 68.3%. For the models of type A the emulator tended to predict its uncertainty rather well, whereas for type B the emulator tended to modestly overpredict its uncertainty.

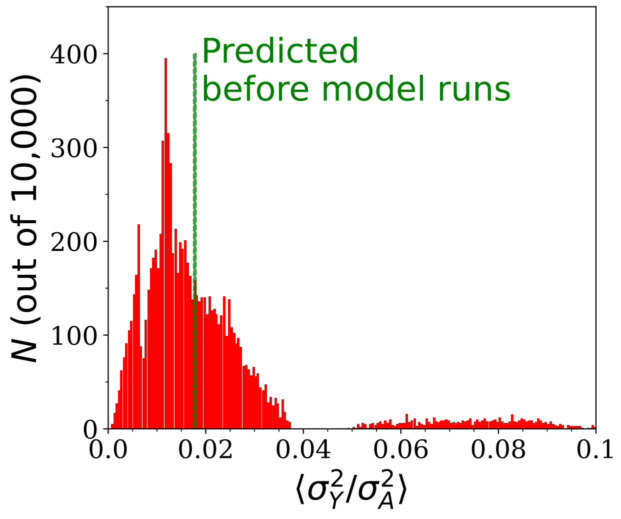

FIG. 9. When optimizing the choice of training points, the procedure produced estimates of the prior-averaged uncertainty relative to σ_A^2. That value is represented by the green dashed line. From tuning 10,000 randomly generated type-A models with $N_{\rm pars} = 12$, a distribution of the ratio, $\langle\sigma_Y^2/\sigma_A^2\rangle$, is displayed here. The distribution is remarkably wide, but the average indeed matches the description. The large width mainly comes from the fact that the value of Λ varies, and that the estimated uncertainty is strongly sensitive to Λ.

V. SUMMARY

The goal of this study was to consider the emulation of functions which are expected to be well described by polynomials with a few orders in the expansion. From this perspective, questions were addressed about what forms of the emulator one should use, how one should choose the training points, and whether one can trust the emulator's statement of its uncertainty. In particular, the motivation is driven by the need to consider models with high-dimensional parameter spaces that, due to their numerical cost, should be trained with optimum efficiency.

In Sec. II it was found that if the following criteria were set, then the form of the emulator would be that of a Gaussian process with a Gaussian kernel. The criteria were:

1. The function should be analytic throughout the prior.

2. At any point the function should fluctuate according to a Gaussian distribution.
3. The prior-to-constraint variance should depend only on $|\vec{\theta}_1 - \vec{\theta}_2|^2$.

The polynomial form corresponding to these criteria was found to be:

$$E(\vec{\theta}) = \sum_{\vec{n}}^{N_{\rm coeff}} A_{\vec{n}} \mathcal{T}_{\vec{n}}(\vec{\theta}), \tag{96}$$

$$\mathcal{T}_{\vec{n}}(\vec{\theta}) = \frac{1}{\sqrt{n_1! n_2! \cdots n_{N_{\rm pars}}!}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_{N_{\rm pars}}} e^{-|\vec{\theta}|^2/2\Lambda^2}.$$

Expressions were found for all expansion coefficients in terms of the training data. The actual coefficients were not needed given that the emulator predictions and uncertainties could all be expressed in terms of the unconstrained correlation,

$$\langle Y(\vec{\theta}_1) Y(\vec{\theta}_2)\rangle_0 = \sigma_A^2 e^{-|\vec{\theta}_1 - \vec{\theta}_2|^2/2\Lambda^2}. \tag{97}$$

In practice, the emulator behaves exactly as a GP emulator with a Gaussian kernel, and with the convergence parameter Λ being the same as the correlation-length hyper-parameter in the GP paradigm. Whereas a large convergence parameter, Λ, relative to the prior size, forces the polynomial to converge rapidly, within the GP paradigm one can equivalently state that the correlation length, Λ, should well exceed the size of the prior.

The emulator's uncertainty was shown to have two contributions. The first comes from the fact that the expansion coefficients, $A_{\vec{n}}$, are constrained by the training but are still very much unknown given that there are typically many more expansion coefficients than training constraints. This first contribution is identical to that which is typically used for GP emulators. The second contribution comes from the fact that the hyper-parameters, σ_A and Λ, are not precisely known. Although there are expressions for the most likely values, the likelihood for varying σ and Λ is characterized by widths. Because the width $W_{\Lambda\Lambda} \neq 0$ and because the prediction can be sensitive to Λ, this leads to a second contribution to the emulator uncertainty. For cases with reasonably large numbers of training points, this second contribution turned out to be relatively small but played an outsized role in the optimization of training-point placement.

In Sec. III, a procedure was presented for optimizing the placement of training points. The metric for accuracy was the squared uncertainty averaged over the prior, $\langle\sigma_E^2/\sigma_A^2\rangle$. Aside from the location of the training points, the accuracy metric also depended on a guess for what value of Λ might be found once the actual training was performed. The measure was expressed analytically for Gaussian or uniform priors. Even with a rather inefficient search procedure, optimized configurations of training points were readily found, although for very high dimensions the procedure might take hours. The optimization was driven mainly by the second source of uncertainty, that due to Λ varying. However, once an optimized placement was chosen, this second source of uncertainty was found to be rather negligible.

It was found that the optimized configuration would typically improve on simple Latin hyper-cube sampling by a factor of two or more. This approach might thus reduce the number of full-model training runs by perhaps a third. The marginal improvement of an additional full-model run was found to fall when the thresholds were passed related to the number of expansion coefficients of a given order. The lesson gained here is that it could be advantageous to choose a number of full-model runs that exactly correspond to the number of coefficients up to a certain order. For example, for $N_{\rm pars} = 12$, there are 91 coefficients of quadratic order or less. The expected accuracy improved significantly more while increasing the number of runs to 91, and improved less with additional runs.

The measure of expected accuracy, $\langle\sigma_E^2/\sigma_A^2\rangle$, was found to be highly sensitive to the convergence parameter, Λ and to the dimensionality of the model-parameter space. Sensitivity to the point-by-point noise was small as long as the noise was small compared to the other variation of the model. Although the estimate of the accuracy depended strongly on Λ, the actual optimum locations of the training points changed only modestly with Λ. The sensitivity to the dimensionality is a prime lesson to be taken from this study. Even if the number of training points is increased to match the increased number of coefficient of a given order as $N_{\rm pars}$ increases, the expected accuracy falls. The challenge of emulating high-dimension spaces was discussed in detail at the end of Sec. III.

To test the methods discussed above, two simple models were considered. The first was a model based on the expansion in Eq. (96), using random expansion coefficients. This precisely coincides with the form assumed by the emulator. The second form was a sum of random trigonometric and exponential forms. Thousands of models were constructed for each class. The models were each then used to train and build emulators. Emulator predictions were then compared to model values at randomly selected points throughout the prior. For the first class of models the emulator provided an outstanding estimate of its accuracy. This was expected given that the model's form matched that assumed by the emulator. For the case of randomized trigonometric and exponential functions, the emulator modestly understated its accuracy.

For the first type of model, one would expect the emulator's value of σ_A and Λ to match those used to define the model. On average the extracted values matched well, but the distribution of extracted values was large. Despite the fact that σ_A and Λ might stray far from the values used to generate the functional data, and despite the fact that the ratio σ_E/σ_A often strayed far from the predictions provided by the training-point-placement procedure, the emulators largely provided good estimates of their uncertainty. This was driven by the fact that extracted σ_A and Λ values were highly correlated. Thus, if the extracted σ_A was much higher than the value used to generate the training data, the extracted values of Λ were also quite high. The two taken together, then led to rather modestly varying uncertainties.

An important lesson to taken from this study is that high-dimension parameter spaces are difficult to emulate unless the functions are highly convergent. However, in most instances one expects many of the model parameters to contribute weakly. A strategy was provided in Sec. III D to exploit this fact by simply assigning suppression factors to some subset of model parameters. The suppression factors would allow the procedure to account for the relatively weaker sensitivity to some parameters by scaling their priors to narrower ranges, while assuming all parameters contributed according to a single convergence parameter.

Aside from the lessons gained from this study, the work points to places where the procedures could be improved or expanded. The main opportunity for improvement would be in coding a more efficient procedure for optimizing the training points. Steepest descent algorithms might accelerate the search by multiple factors of two. A primary opportunity for expanding the functionality would be to consider more classes of priors when optimizing the training-point placement. The current software considered Gaussian and uniform priors. One could consider half Gaussians, or even or odd functions.

Finally, one needs to ask the question as to whether one should perform full model runs simultaneously or sequentially. Here, by doing a significant number simultaneously, one can maximize the accuracy for that number of runs as measured over the entire prior. However, once the emulator is tuned well enough, one could perform a Markov Chain Mote Carlo (MCMC) exploration of space and identify those regions of the parameter space which are likely. One could then augment the training data with model runs specifically aimed to best constrain predictions for regions that are relevant to the goal of determining the model parameters. The additional points would likely be located in the likely regions of parameter space. Given that one has little concrete knowledge of where the likely region of parameter space lies before the training, it might make sense to simultaneously perform a first set of predictions to constrain the linear or quadratic behavior without comparing to experiment. Then, after determining the likely regions of parameter space by comparing to data, one might perform additional runs in a more sequential manner. Unfortunately, it is difficulty to say whether the first set of runs should be small, e.g. only enough for a linear fit, or whether one might opt for enough for a quadratic fit. The decision might rest on numerous factors. This might include how convergent the functions were expected to be, their numerical cost, and whether one can easily perform some number of full-model runs simultaneously. Unfortunately, their convergence would not be well understood until after the first round of model runs. Choosing an optimized strategy for most cases should be quite beneficial and allow one to avoid performing unnecessary full-model runs.

Appendix A: Properties of the Polynomial Basis Functions

The polynomial expansion presented in Sec. II can be considered as an expansion in basis functions. The purpose of this appendix is to examine the orthonormality and completeness properties of those basis functions.

The polynomial expansion, for $N = N_{\rm pars}$, used for the studies here is of the form

$$Y(\vec{\theta}) = e^{-|\vec{\theta}|^2/2\Lambda^2} \sum_{\vec{n}} \frac{A_{\vec{n}}}{\sqrt{n_1!n_2!\cdots n_N!}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_N}, \tag{A1}$$

where $A_{\vec{n}}$ were independent coefficients, each obeying a Gaussian distribution,

$$\langle A_{\vec{m}} A_{\vec{n}} \rangle_0 = \sigma_A^2 \delta_{\vec{m}\vec{n}}. \tag{A2}$$

This can be considered as an expansion in terms of basis functions, $\phi_{\vec{n}}$ defined as

$$\phi_{\vec{n}}(\vec{\theta}) \equiv \frac{1}{(2\pi\Lambda^2)^{N/4}} e^{-|\vec{\theta}|^2/2\Lambda^2} \frac{1}{\sqrt{n_1!n_2!\cdots n_N!}} \left(\frac{\theta_1}{\Lambda}\right)^{n_1} \left(\frac{\theta_2}{\Lambda}\right)^{n_2} \cdots \left(\frac{\theta_N}{\Lambda}\right)^{n_N}. \tag{A3}$$

Here, the basis functions are labeled by $\vec{n}$, which is a list of any N non-negative integers. With this definition the emulator is

$$Y(\vec{\theta}) = (2\pi\Lambda^2)^{N/4} \sum_{\vec{n}} A_{\vec{n}} \phi_{\vec{n}}(\vec{\theta}). \tag{A4}$$

The prefactor, $1/(2\pi\Lambda^2)^{N/4}$, is defined for convenience.

First, the orthogonality and normalization properties are:

$$\begin{aligned}
Z_{\vec{m}\vec{n}} &= \int d\theta_1 \cdots d\theta_N \; \phi_{\vec{m}}(\vec{\theta})\phi_{\vec{n}}(\vec{\theta}) \qquad \text{(A5)}\\
&= \prod_{j=1}^{N} z_{m_j n_j},\\
z_{m_j n_j} &= \frac{1}{(2\pi\Lambda^2)^{1/2}} \frac{1}{\sqrt{m_j! n_j!}} \int d\theta \; e^{-\theta^2/\Lambda^2} \left(\frac{\theta}{\Lambda}\right)^{m_j+n_j}\\
&= \begin{cases} \frac{1}{(2\pi)^{1/2}} \frac{1}{\sqrt{m_j! n_j!}} \Gamma([m_j+n_j+1]/2), & m_j+n_j \text{ is even} \\ 0, & m_j+n_j \text{ is odd} \end{cases}.
\end{aligned}$$

The basis functions are neither normalized nor orthogonal, although the normalizations are of order unity. For the purposes of emulation it is not clear that orthonormality properties play an important role.

On the other hand, understanding completeness determines whether the basis can capture all facets of the function it is attempting to represent. The completeness properties are defined by

$$\begin{aligned}
\sum_{\vec{n}} \; \phi_{\vec{n}}(\vec{\theta}_a)\phi_{\vec{n}}(\vec{\theta}_b) &= \frac{1}{(2\pi\Lambda^2)^{N/2}} e^{-|\vec{\theta}_a|^2/2\Lambda^2} e^{-|\vec{\theta}_b|^2/2\Lambda^2} \qquad \text{(A6)}\\
&\quad \sum_{\vec{n}} \frac{1}{n_1! \cdots n_{N!}} \left(\frac{(\theta_{a1}\theta_{b1})^{n_1}}{\Lambda^{2n_1}}\right) \cdots \left(\frac{(\theta_{aN}\theta_{bN})^{n_N}}{\Lambda^{2n_N}}\right)\\
&= \frac{1}{(2\pi\Lambda^2)^{N/2}} e^{-|\vec{\theta}_a - \vec{\theta}_b|^2/2\Lambda^2}.
\end{aligned}$$

For a complete basis the right-hand side would have been a delta function. Instead, it is a smeared delta function. It integrates to unity but has a non-zero width, Λ. This would satisfy the completeness relation if Λ were allowed to approach zero, but in that limit one would likely require an infinite number of terms in the expansion.

This basis can then be considered as being incomplete in that it cannot well represent small scale features, i.e. those with characteristic scales below Λ. For perfect completeness to hold over an infinite volume one would expect to require a continuous set of coefficients that cover an infinite extent. A similar example would be a Fourier transform, where instead of the infinite sets of integers, $\vec{n}$, one has the vector $\vec{k}$ which is distributed continuously. If one confined the vectors $\vec{k}$ to integer multiples of some step vectors, Δk_i, one would be unable to represent features below the inverse size of the step vectors, and if one cut off the expansion at some maximum, $k_{\rm max}$, one would be unable to describe structures of sizes below $1/k_{\rm max}$. For the problem here, Λ sets the minimum scale at which one would expect features for the full function, and the maximum scale at which one can represent the function is determined by whether the sum over $\vec{n}$ is cut off at some maximum order.

Above, for a given rank K, the basis functions include independent terms for all $n_1 + n_2 + \cdots n_{N_{\rm pars}} = K$. One might have instead chosen to express the polynomials as all terms i_1, i_2, i_K, so that $Y(\vec{\theta})$ would then be:

$$Y(\vec{\theta}) = \sum_{K, i_1 \cdots i_K} A_{i_1 \cdots i_K} \frac{1}{K!} \theta_{i_1} \theta_{i_d} \cdots \theta_{i_K}. \qquad \text{(A7)}$$

Here, $1 \leq i_j \leq N_{\rm pars}$. In this form, there are more coefficients. In the first form there was a term proportional to $\theta_1^2\theta_2$, and in this form there are three separate terms with the same product, $\theta_1\theta_1\theta_2$, $\theta_2\theta_1\theta_1$ and $\theta_1\theta_2\theta_1$. This alternate form has redundant coefficients, but those coefficients have less strength. In the previous form $N!d_{\vec{n}}^2$ was set to $N!/(n_1!n_2!n_3...)$, which equals the number of identical terms contributing in the new form. Because having d independent gaussians is the same as having a single gaussian of strength $\sqrt{d}$, the end result is the same correlation. Thus, the alternate form in Eq. (A7) leads to all the same results as that presented in Sec. (II).

ACKNOWLEDGMENTS

This work was supported by the Department of Energy Office of Science through grant number DE-FG02-03ER41259 and by the National Science Foundation's Cyberinfrastructure for Sustained Scientific Innovation program through grant number OAC-2004601. The authors express gratitude to Simon Mak and Daniel Phillips for generously sharing their insight into the subject.

[1] T. S. Domingues, R. Krupczak, J. Noronha, T. N. da Silva, J. F. Paquet and M. Luzum, Phys. Rev. C **110**, no.6, 064904 (2024).
[2] R. Ehlers *et al.* [JETSCAPE], [arXiv:2408.08247 [hep-ph]].
[3] R. Ehlers *et al.* [JETSCAPE], EPJ Web Conf. **296**, 15009 (2024).
[4] J. F. Paquet, J. Phys. G **51**, no.10, 103001 (2024).
[5] W. Fan *et al.* [JETSCAPE], Phys. Rev. C **109**, no.6, 064903 (2024).
[6] M. R. Heffernan, C. Gale, S. Jeon and J. F. Paquet, Phys. Rev. Lett. **132**, no.25, 252301 (2024).
[7] M. R. Heffernan, C. Gale, S. Jeon and J. F. Paquet, Phys. Rev. C **109**, no.6, 065207 (2024).
[8] B. Weiss, J. F. Paquet and S. A. Bass, J. Phys. G **50**, no.6, 065104 (2023).
[9] Y. Ji, H. S. Yuchi, D. Soeder, J. F. Paquet, S. A. Bass, V. R. Joseph, C. F. J. Wu and S. Mak, [arXiv:2209.13748 [stat.ME]].
[10] M. Heffernan, C. Gale, S. Jeon and J. F. Paquet, Acta Phys. Polon. Supp. **16**, no.1, 1-A151 (2023).
[11] D. Liyanage, Y. Ji, D. Everett, M. Heffernan, U. Heinz, S. Mak and J. F. Paquet, Phys. Rev. C **105**, no.3, 034910 (2022).
[12] J. Mulligan *et al.* [JETSCAPE], [arXiv:2106.11348 [nucl-th]].
[13] S. Cao *et al.* [JETSCAPE], Phys. Rev. C **104**, no.2, 024905 (2021).
[14] J. E. Bernhard, J. S. Moreland and S. A. Bass, Nature Phys. **15**, no.11, 1113-1117 (2019).
[15] J. S. Moreland, J. E. Bernhard and S. A. Bass, Phys. Rev. C **101**, no.2, 024911 (2020).
[16] J. S. Moreland, J. E. Bernhard and S. A. Bass, Nucl. Phys. A **982**, 503-506 (2019).
[17] Y. Xu, J. E. Bernhard, S. A. Bass, M. Nahrgang and S. Cao, Phys. Rev. C **97**, no.1, 014907 (2018).
[18] W. Ke, J. S. Moreland, J. E. Bernhard and S. A. Bass, Nucl. Part. Phys. Proc. **289-290**, 483-486 (2017).
[19] Y. Xu, S. Cao, M. Nahrgang, J. E. Bernhard and S. A. Bass, Nucl. Part. Phys. Proc. **289-290**, 257-260 (2017).
[20] J. Auvinen, J. E. Bernhard, S. A. Bass and I. Karpenko, Phys. Rev. C **97**, no.4, 044905 (2018).
[21] Y. Xu, M. Nahrgang, J. E. Bernhard, S. Cao and S. A. Bass, Nucl. Phys. A **967**, 668-671 (2017).
[22] J. Auvinen, I. Karpenko, J. E. Bernhard and S. A. Bass, Nucl. Phys. A **967**, 784-787 (2017).
[23] J. E. Bernhard, J. S. Moreland and S. A. Bass, Nucl. Phys. A **967**, 293-296 (2017).
[24] W. Ke, J. S. Moreland, J. E. Bernhard and S. A. Bass, Phys. Rev. C **96**, no.4, 044912 (2017).
[25] J. Auvinen, J. E. Bernhard and S. A. Bass, Acta Phys. Polon. Supp. **10**, 455 (2017).
[26] J. E. Bernhard, P. W. Marcy, C. E. Coleman-Smith, S. Huzurbazar, R. L. Wolpert and S. A. Bass, Phys. Rev. C **91**, no.5, 054910 (2015).
[27] S. A. Jahan, H. Roch and C. Shen, Phys. Rev. C **110**, no.5, 054905 (2024).
[28] A. Mankolli *et al.* [JETSCAPE], EPJ Web Conf. **296**, 05010 (2024).
[29] C. Shen, B. Schenke and W. Zhao, EPJ Web Conf. **296**, 14001 (2024).
[30] H. Mäntysaari, B. Schenke, C. Shen and W. Zhao, Acta Phys. Polon. Supp. **16**, no.1, 1-A33 (2023).
[31] H. Mäntysaari, B. Schenke, C. Shen and W. Zhao, Phys. Lett. B **833**, 137348 (2022).
[32] D. Everett *et al.* [JETSCAPE], Phys. Rev. C **103**, no.5, 054904 (2021).
[33] J. F. Paquet *et al.* [JETSCAPE], Nucl. Phys. A **1005**, 121749 (2021).
[34] S. Pratt, E. Sangaline, P. Sorensen and H. Wang, Phys. Rev. Lett. **114**, 202301 (2015).
[35] E. Sangaline and S. Pratt, Phys. Rev. C **93**, no.2, 024908 (2016).
[36] J. Novak, K. Novak, S. Pratt, J. Vredevoogd, C. Coleman-Smith and R. Wolpert, Phys. Rev. C **89**, no.3, 034917 (2014).
[37] B. A. Li and W. J. Xie, [arXiv:2501.02579 [nucl-th]].
[38] M. Alqahtani, R. S. Bhalerao, G. Giacalone, A. Kirchner and J. Y. Ollitrault, Phys. Rev. C **110**, no.6, 064906 (2024).
[39] J. M. Wang, X. G. Deng, W. J. Xie, B. A. Li and Y. G. Ma, [arXiv:2406.07051 [nucl-th]].
[40] B. A. Li and W. J. Xie, Nucl. Phys. A **1039**, 122726 (2023).
[41] A. V. Giannini *et al.* [ExTrEMe], Phys. Rev. C **107**, no.4, 044907 (2023).
[42] J. Jia and S. Mohapatra, Phys. Rev. C **88**, no.1, 014907 (2013).
[43] P. Morfouace et al., Phys. Lett. B **799**, 135045 (2019).
[44] C.E. Rasmussen and C.K.I. Williams, Gaussian Processes for Machine Learning, the MIT Press, 2006, ISBN 026218253X.
[45] Williams, C.K.I. and Rasmussen, C.E., *Gaussian Processes for Regression* in *Advances in Neural Information Processing Systems* **8**, ed. by D.S. Touretzky, M.C. Mozer and M.E. Hasselmo, MIT Press (1996).
[46] T. Tang, S. Mak and D. Dunson, *Hierarchical Shrinkage Gaussian Processes: Applications to Computer Code Emulation and Dynamical System Recovery* SIAM/ASA J. on Uncertainty Quantification **12**, no. 4, 10.1137 (2024).
[47] D. Xiu, *Generalized Polynomial Chaos* in *Numerical Methods for Stochastic Computations*, Princeton University Press (2010)
[48] M.D. McKay, R.J. Beckman, R.J. and W.J. Conover. Technometrics. 21 (2). American Statistical Association: 239–245. (1979).