Structured Covariance Gaussian Networks for Orion Crew Module Aerodynamic Uncertainty Quantification

Tenavi Nakamura-Zimmerer,* Mary T. Stringer,†, Brendon K. Colbert‡, and James B. Scoggins§
NASA Langley Research Center, Hampton, VA 23681

In this paper we propose a new approach for nonlinear regression and uncertainty quantification. The method is based on a pair of neural networks which parameterize mean and dense covariance functions of a multivariate Gaussian process, trained together to maximize the log-likelihood of observing the given data. The covariance matrix is made positive definite at every input by construction. We also propose a sampling approach that produces viable surrogate function realizations from the Gaussian process. We call the proposed model a Structured Covariance Gaussian Network (SCGN). We illustrate the use of SCGNs for learning an aerodynamic response surface with built-in uncertainty for the Orion crew module. We find that SCGN provides an efficient and systematic way to learn nonlinear functional relationships and dense covariances. We compare results to a baseline Gaussian process regressor and observe that the SCGN provides comparable uncertainty descriptions with improved scalability to dataset size. The sample functions generated by SCGN are fast to evaluate online and are therefore convenient for use in trajectory simulations. These results suggest that SCGN may be a viable computational method for aerodynamic uncertainty quantification.

I. Introduction

For many regression tasks we are interested not only in learning a single best-fit input-output relationship, but also capturing uncertainty in the data and sometimes the model parameters. Numerous computational approaches have been proposed for such *uncertainty quantification* (UQ) tasks. Some well-studied examples include Gaussian process regression (GPR) [1–3], generalized polynomial chaos [4], conditional deep generative models [5–7], and Bayesian neural networks [8]. Each of the aforementioned UQ methods comes with its own advantages and disadvantages, and there is no one-size-fits-all which works for all problems. Therefore the development of UQ methods is the subject of ongoing research.

In this paper we propose a new approach for UQ and nonlinear regression. We use neural networks (NNs) to parameterize mean and dense covariance functions of a multivariate *Gaussian process* (GP). To ensure that the covariance matrix is positive definite we learn its Cholesky decomposition and force the eigenvalues to be positive. The NNs for the mean and covariance are trained together to maximize the log-likelihood of observing the given data. This method for constructing the covariance and NN training are adapted from [9], who introduced this architecture for UQ of generative (i.e. not regression) models. In the context of regression we are also interested in learning input-output *functions*, which introduces additional challenges compared to learning pointwise statistics. Specifically, we want to learn a *stochastic process* to generate statistically and physically reasonable surrogate functions. To this end we propose a method for treating the learned distribution as a GP and sampling continuous surrogate functions. We call the proposed UQ model a *Structured Covariance Gaussian Network* (SCGN).

The SCGN method proposed in this paper, while more generally applicable, was developed in the context of aerodynamic modeling and UQ. Aerodynamic models are typically based on force and moment data from computational fluid dynamics (CFD) simulations and wind tunnel tests (WTT). Despite the increasing capabilities of CFD and WTT, or perhaps *because of* the increasing availability and complexity of the data, determination of an aerodynamic model for a novel aerospace vehicle can be challenging. In practice, aerodynamic models are often developed based on piecewise polynomials and ad hoc root sum of squares UQ [10, 11]. This requires thorough bookkeeping and manual analysis of many possible sources of uncertainty, as well as considerable use of engineering judgement at each step of the process. That is to say that at present time, there is no standard and systematic methodology for constructing aerodynamic models

*Research Aerospace Engineer, Flight Dynamics Branch.
†Airworthiness Lead, X-59 Low Boom Flight Demonstrator.
‡Research Aerospace Engineer, Dynamic Systems and Controls Branch.
§Research Aerospace Engineer, Aerothermodynamics Branch.

with integrated UQ. As a result, aerodynamic models take longer to develop and can sometimes have uncertainties which are too conservative. This in turn leads to additional expense and inefficiencies in the vehicle and control system design. As NASA develops new aerospace vehicles for future missions in space and on earth, it has become clear that the current methodology is insufficient. This motivates the development of a systematic framework for building aerodynamic models which rigorously capture uncertainties. Developing such a framework is the goal of the AeroFusion project at NASA [12], under which SCGN has been developed.

As a concrete example, we apply SCGN to model force and moment coefficients of the Orion crew module [10, 11] as a function of velocity and angle of attack. We compare the proposed method to a GPR [1], which often serves as baseline standard for evaluating novel UQ methods (see e.g. [7]) and has been proposed as a method of UQ in aerospace applications [2]. We find that SCGN provides an efficient way to learn nonlinear relationships and dense covariances. The results indicate that SCGN provide competitive uncertainty descriptions with improved scalability to dataset size and fast evaluation time. Such fast evaluation times are important when aerodynamic models are queried repeatedly in Monte Carlo trajectory simulations, for example. These results suggest that SCGN may be a viable computational method for UQ and a potentially useful tool for developing aerodynamic models of new aerospace vehicles.

The rest of this paper is organized as follows. In Section I.A we describe the mathematical setting for the UQ problem at hand and introduce some key notation. In Section I.B we discuss the work of [9] on which we base the SCGN model architecture, as well as some recent work on UQ in aerodynamics. In Section II we present the model architecture and training process, then in Section III we discuss options for making predictions and generating sample functions with SCGN. Here we also show how to interpret SCGN as a GP and derive the covariance kernel for each sampling approach. Finally in Section IV we apply SCGN to learn the aerodynamics of the Orion crew module and compare the results to GPR. A concluding discussion and directions for future work are given in Section V.

A. Mathematical problem setting

Suppose that we have a dataset

$$\mathcal{D} = \left\{\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right\}_{i=1}^{N}, \tag{1}$$

which consists of N input-output sample pairs $\mathbf{x}^{(i)} \in \mathbb{R}^{n_x}, \mathbf{y}^{(i)} \in \mathbb{R}^{n_y}$. We can think of these pairs as being drawn from a conditional distribution

$$\mathbf{y}^{(i)} \sim \rho\left(\mathbf{y}|\mathbf{x}^{(i)}\right), \qquad i = 1, \ldots, N. \tag{2}$$

There are two basic ways to think about modeling Eq. (2). The most straightforward way is through a conditional distribution $\widehat{\rho}(\mathbf{y}|\mathbf{x}) \approx \rho(\mathbf{y}|\mathbf{x})$. However, in the context of aerodynamic modeling we are not just interested in pointwise predictions but rather entire response surfaces $\widehat{\mathbf{y}} = \widehat{\mathbf{f}}(\mathbf{x})$. Each such response surface should be a physically and statistically plausible surrogate function describing the aerodynamics.

Because of this, it is reasonable to frame the modeling problem in terms of learning a stochastic process, which provides a mechanism for randomly generating such surrogate functions. Intuitively, a stochastic process can be thought of as a random variable in a function space, so sampling this random variable yields a function. A formal statement is given by Definition 1 below.

Remark 1 *For the Orion crew module, uncertainty has historically been been applied by taking a random (constant) offset from the nominal response surface [10, 11]. While intuitive and simple to implement, this method does not express the full variability of plausible surrogate functions. By contrast, stochastic processes provide a convenient mathematical framework for sampling the function space.*

Definition 1 (Stochastic processes) *Let ω be a random variable defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, see e.g. [4]. A stochastic process is a collection of random variables $\mathbf{y} \in \mathbb{R}^{n_y}$ associated with input variables $\mathbf{x} \in \mathbb{R}^{n_x}$ by a map $\mathbf{y} = \mathbf{f}(\mathbf{x}; \omega)$ denoted as*

$$\{\mathbf{f}(\mathbf{x}; \omega) : \mathbb{R}^{n_x} \times \Omega \to \mathbb{R}^{n_y}\}. \tag{3}$$

A realization, or sample function, $\mathbf{f}(\cdot; \omega_0) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, of a stochastic process is defined for any realization of $\omega_0 \in \Omega$. We use the following notation to show that a function is a realization of a particular stochastic process:

$$\mathbf{f}(\cdot; \omega_0) \sim \{\mathbf{f}(\mathbf{x}; \omega) : \mathbb{R}^{n_x} \times \Omega \to \mathbb{R}^{n_y}\}. \tag{4}$$

The use of lower case letters to denote random variables and stochastic process is non-standard, but is meant to be consistent with the notation used to differentiate scalars (lower case), vectors (lower case bold), and matrices (upper case bold). When a variable is random this will be stated where it is defined.

The challenge of course is how to parameterize and fit a stochastic process model. Many modeling approaches exist; a partial review will be given in Section I.B. One popular choice is to use GPs, which serve as the basis for GPR and the SCGN approach introduced in this paper. We give a brief definition of Gaussian processes below; further details, including an introduction to GPR, can be found in e.g. [1].

Definition 2 (Gaussian process) *Let ω be a random variable defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. A stochastic process $\{\mathbf{f}(\mathbf{x};\omega) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}\}$ is a (multivariate) GP if, given any finite set $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\} \subset \mathbb{R}^{n_x}$, the joint distribution of $\left\{\mathbf{f}\left(\mathbf{x}^{(1)};\omega\right), \ldots, \mathbf{f}\left(\mathbf{x}^{(N)};\omega\right)\right\} \in \mathbb{R}^{n_y}$ is multivariate Gaussian. That is,*

$$\begin{pmatrix} \mathbf{f}\left(\mathbf{x}^{(1)};\omega\right) \\ \vdots \\ \mathbf{f}\left(\mathbf{x}^{(N)};\omega\right) \end{pmatrix} \sim \mathcal{N}\left(\mathbf{M}, \mathbf{\Sigma}\right), \tag{5}$$

where

$$\mathbf{M} = \begin{pmatrix} \boldsymbol{\mu}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \boldsymbol{\mu}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} \in \mathbb{R}^{N \cdot n_y}, \quad \mathbf{\Sigma} = \begin{pmatrix} \boldsymbol{\kappa}\left(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}\right) & \cdots & \boldsymbol{\kappa}\left(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}\right) \\ \vdots & \ddots & \vdots \\ \boldsymbol{\kappa}\left(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}\right) & \cdots & \boldsymbol{\kappa}\left(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}\right), \end{pmatrix} \in \mathbb{R}^{\left(N \cdot n_y\right) \times \left(N \cdot n_y\right)}. \tag{6}$$

Here $\boldsymbol{\mu} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ is called the mean function and $\boldsymbol{\kappa} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_y \times n_y}$ is the covariance kernel. We require that the kernel be defined in such a way that $\mathbf{\Sigma} \succeq \mathbf{0}$ (positive semi-definite) for all finite sets of inputs $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\} \subset \mathbb{R}^{n_x}$. We denote GPs and their realizations by

$$\mathbf{f}(\cdot) \sim \mathcal{GP}\left(\boldsymbol{\mu}(\cdot), \boldsymbol{\kappa}(\cdot,\cdot)\right). \tag{7}$$

B. Related work

The SCGN approach is adapted from [9], who originally proposed the covariance NN structure to model uncertainty for generative models. In this context we seek to model a dataset

$$\mathcal{D} = \left\{\mathbf{y}^{(i)}\right\}_{i=1}^{N}, \qquad \mathbf{y}^{(i)} \sim \rho(\mathbf{y}).$$

Notice that there is no input variable, $\mathbf{x}$. Suppose we are given some generative model (such as a NN) which approximates $\mathbf{y} \approx \boldsymbol{\mu}(\mathbf{z})$, where $\mathbf{z}$ is a *latent* random variable. Then sampling $\mathbf{z}^*$ from the latent distribution generates a (hopefully) statistically representative sample, $\mathbf{y}^* = \boldsymbol{\mu}\left(\mathbf{z}^*\right)$. [9] proposes quantifying the error in the learned distribution by a multivariate Gaussian distribution, with mean and covariance depending on the latent variable $\mathbf{z}$:

$$\rho(\mathbf{y}) \approx \mathcal{N}\left(\boldsymbol{\mu}(\mathbf{z}), \mathbf{\Sigma}(\mathbf{z})\right).$$

Here the covariance $\mathbf{\Sigma}(\cdot)$ is represented by a second NN constructed as in Section II.A.

In this paper we apply the NN architecture from [9] to learning conditional distributions (2) for aerodynamic models. This comes with an additional consideration: the learned model must be interpreted as a GP and realizations of the GP must all be physically reasonable surrogate functions. In addition, we observe that the sampling scheme used by [9] introduces unintended statistical dependence on the *order* of the target variables. If the objective is just pointwise UQ applied to image generation then this may not be as critical, but for the intended application of aerodynamic modeling this is important. For these reasons we extend [9] to include an efficient sampling approach which respects the learned pointwise distributions but is order-independent and generates useful surrogate functions.

II. Structured Covariance Gaussian Networks

In this section we describe the proposed SCGN approach for UQ. The SCGN approach is a conceptually simple idea: use NNs to learn a spatially-varying Gaussian distribution. But it turns out that a few technical details have significant effects on the learned GP, notably in the construction of the covariance and the sampling methodology. In Section II.A

we introduce the model architecture, which consists of a pair of NNs which learn mean and covariance functions. Next in Section II.B we discuss how to train these NNs by log-likelihood maximization. In Section III we discuss how to generate random samples and function realizations from the model. Therein lies our main contribution: from among several possible sampling approaches we suggest one which yields physically-reasonable surrogate functions.

A. Model architecture

We model the conditional distribution (2) using a spatially-varying multivariate Gaussian:

$$\widehat{\mathbf{y}} \sim \mathcal{N}\left(\boldsymbol{\mu}\left(\mathbf{x};\boldsymbol{\theta}_\mu\right), \boldsymbol{\Sigma}\left(\mathbf{x};\boldsymbol{\theta}_U\right)\right) \approx \rho\left(\mathbf{y}\,|\mathbf{x}\right). \tag{8}$$

Here $\boldsymbol{\mu} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ is a NN parameterized by weights $\boldsymbol{\theta}_\mu \in \mathbb{R}^{n_{\theta_\mu}}$ which models the mean:

$$\boldsymbol{\mu}\left(\mathbf{x};\boldsymbol{\theta}_\mu\right) \approx \mathbb{E}_\rho\left\{\mathbf{y}\,|\mathbf{x}\right\}. \tag{9}$$

We construct the covariance via a second NN which parameterizes the *Cholesky decomposition* of the *inverse covariance* matrix (also called the *sensitivity* matrix):

$$\boldsymbol{\Sigma}\left(\mathbf{x};\boldsymbol{\theta}_U\right) = \left(\left[\mathbf{U}\left(\mathbf{x};\boldsymbol{\theta}_U\right)\right]^T \mathbf{U}\left(\mathbf{x};\boldsymbol{\theta}_U\right)\right)^{-1} \approx \mathbb{E}_\rho\left\{\left(\mathbf{y}-\mathbb{E}_\rho\left\{\mathbf{y}\,|\mathbf{x}\right\}\right)\left(\mathbf{y}-\mathbb{E}_\rho\left\{\mathbf{y}\,|\mathbf{x}\right\}\right)^T \middle| \mathbf{x}\right\}. \tag{10}$$

Here $\mathbf{U} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y \times n_y}$ is an *upper-triangular matrix-valued* NN parameterized by $\boldsymbol{\theta}_U \in \mathbb{R}^{n_{\theta_U}}$. The diagonal entries are made positive by exponentiating the corresponding outputs of the NN. This ensures that $\mathbf{U}(\cdot)$ is invertible. As discussed, $\mathbf{U}(\cdot)$ is the Cholesky decomposition of the sensitivity matrix:

$$\left[\boldsymbol{\Sigma}\left(\mathbf{x};\boldsymbol{\theta}_U\right)\right]^{-1} = \left[\mathbf{U}\left(\mathbf{x};\boldsymbol{\theta}_U\right)\right]^T \mathbf{U}\left(\mathbf{x};\boldsymbol{\theta}_U\right). \tag{11}$$

This construction of the sensitivity matrix using a NN with exponentiated diagonal is due to [9], though in this paper we apply it to learn conditional distributions instead of UQ for generative models. Note that because $\mathbf{U}(\cdot)$ is always invertible, by construction $\boldsymbol{\Sigma}(\cdot) \succ \mathbf{0}$ (positive definite) everywhere, making the probability density well-defined for every input. In addition, as we see in Section II.B learning the Cholesky decomposition of the inverse covariance makes it very convenient to train the model using log-likelihood maximization.

B. Model training

In [9] the mean function $\boldsymbol{\mu}(\cdot)$ is taken as fixed and the covariance is trained by maximizing the log-likelihood of observing the data. For our application we want to learn both the mean and covariance. In this setting we find that results are noticeably more consistent if we pre-train the mean by minimizing, for example, a mean square error (MSE) loss:

$$\boldsymbol{\theta}_\mu = \underset{\boldsymbol{\theta}_\mu}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{y}^{(i)} - \boldsymbol{\mu}\left(\mathbf{x}^{(i)};\boldsymbol{\theta}_\mu\right)\right\|_2^2. \tag{12}$$

To limit overfitting at this stage, we also employ ℓ^2 weight regularization, so that we now solve a modified optimization problem:

$$\boldsymbol{\theta}_\mu = \underset{\boldsymbol{\theta}_\mu}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{y}^{(i)} - \boldsymbol{\mu}\left(\mathbf{x}^{(i)};\boldsymbol{\theta}_\mu\right)\right\|_2^2 + \frac{\lambda_\mu}{n_{\theta_\mu}}\left\|\boldsymbol{\theta}_\mu\right\|^2, \tag{13}$$

where $\lambda_\mu \geq 0$ is a hyperparameter governing the strength of the regularization.

After initializing the mean function we train both networks simultaneously by maximizing the average log-likelihood:

$$\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_U = \underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmax}} \frac{1}{N}\sum_{i=1}^{N}\log\widehat{\rho}\left(\mathbf{y}^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U\right). \tag{14}$$

Since we specify $\widehat{\rho}(\mathbf{y}|\mathbf{x})$ as Gaussian, maximizing the average log-likelihood (14) is equivalent to the following

minimization problem, where we write $\boldsymbol{\mu}^{(i)} \coloneqq \boldsymbol{\mu}\left(\mathbf{x}^{(i)};\boldsymbol{\theta}_\mu\right)$, $\mathbf{U}^{(i)} \coloneqq \mathbf{U}\left(\mathbf{x}^{(i)};\boldsymbol{\theta}_U\right)$, and $\boldsymbol{\Sigma}^{(i)} \coloneqq \boldsymbol{\Sigma}\left(\mathbf{x}^{(i)};\boldsymbol{\theta}_U\right)$ for brevity:

$$
\begin{aligned}
&\underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmax}} \frac{1}{N}\sum_{i=1}^{N} \log \widehat{\rho}\left(\mathbf{y}^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U\right) \\
=&\underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{\exp\left(-\frac{1}{2}\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]^T\left[\boldsymbol{\Sigma}^{(i)}\right]^{-1}\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]\right)}{\sqrt{(2\pi)^{n_y}\det\boldsymbol{\Sigma}^{(i)}}}\right) \\
=&\underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]^T\left[\boldsymbol{\Sigma}^{(i)}\right]^{-1}\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]+\frac{1}{2}\left[\log\det\boldsymbol{\Sigma}^{(i)}+\log(2\pi)^{n_y}\right]\right) \\
=&\underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\left\|\left[\mathbf{U}^{(i)}\right]\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]\right\|_2^2+\frac{1}{2}\log\frac{1}{\det\left(\left[\mathbf{U}^{(i)}\right]^T\mathbf{U}^{(i)}\right)}+\frac{n_y}{2}\log 2\pi\right) \\
=&\underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\left\|\left[\mathbf{U}^{(i)}\right]\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]\right\|_2^2-\log\det\mathbf{U}^{(i)}+\frac{n_y}{2}\log 2\pi\right).
\end{aligned}
$$

Dropping the constant term and using the fact that $\mathbf{U}^{(i)}$ is upper triangular with diagonal entries $u_{jj}^{(i)} \coloneqq \left[\mathbf{U}\left(\mathbf{x}^{(i)};\boldsymbol{\theta}_U\right)\right]_{jj}$, this becomes

$$
\underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmax}} \frac{1}{N}\sum_{i=1}^{N} \log \widehat{\rho}\left(\mathbf{y}^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U\right) = \underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\left\|\left[\mathbf{U}^{(i)}\right]\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]\right\|_2^2-\sum_{j=1}^{n_y}\log u_{jj}^{(i)}\right). \tag{15}
$$

Here we also use ℓ^2 weight regularization for both the mean and covariance NN, so that the optimization problem becomes

$$
\underset{\boldsymbol{\theta}_\mu,\boldsymbol{\theta}_U}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}\left\|\left[\mathbf{U}^{(i)}\right]\left[\mathbf{y}^{(i)}-\boldsymbol{\mu}^{(i)}\right]\right\|_2^2-\sum_{j=1}^{n_y}\log u_{jj}^{(i)}\right)+\frac{\lambda_\mu}{n_{\boldsymbol{\theta}_\mu}}\left\|\boldsymbol{\theta}_\mu\right\|^2+\frac{\lambda_U}{n_{\boldsymbol{\theta}_U}}\left\|\boldsymbol{\theta}_U\right\|^2, \tag{16}
$$

for loss function weights $\lambda_\mu, \lambda_U \geq 0$.

Here we can clearly see the advantage of learning a decomposition of the inverse covariance. Since no matrix inversion is required, the first term easy to implement and cheap to compute. The second term is also convenient to calculate because the log diagonals, $\log u_{jj}^{(i)}$, are obtained naturally as the raw outputs of the NN before they are exponentiated. A further benefit of this log-likelihood formulation is that, for moderately-sized datasets and NNs, we can perform *full-batch optimization* of Eq. (16). This allows us to use second order optimizers like L-BFGS [13], which has built-in early stopping criteria and a faster convergence rate than popular stochastic gradient descent-based methods [14].

III. Making predictions with SCGNs

We now turn to the problem of sampling the learned Gaussian distribution. When interpreting SCGN as a GP, we realize that the choice of sampling method has a significant effect on the structure of the covariance kernel.

Suppose for the moment that we are only interested in making sample predictions at a single point, say $\mathbf{x}^* \in \mathbb{R}^{n_x}$. If we have learned a good surrogate distribution (8) then we know $\boldsymbol{\mu} = \boldsymbol{\mu}\left(\mathbf{x}^*;\boldsymbol{\theta}_\mu\right)$ and $\boldsymbol{\Sigma} = \left(\mathbf{U}^T\mathbf{U}\right)^{-1}$, where $\mathbf{U} = \mathbf{U}\left(\mathbf{x}^*;\boldsymbol{\theta}_U\right)$. Then all we need to do is to sample $\widehat{\mathbf{y}} \sim \mathcal{N}\left(\boldsymbol{\mu},\boldsymbol{\Sigma}\right)$. To this end we recall that if $\mathbf{V} \in \mathbb{R}^{n_y\times m}$ is any matrix such that $\boldsymbol{\Sigma} = \mathbf{V}\mathbf{V}^T$ then

$$
\widehat{\mathbf{y}} \sim \mathcal{N}\left(\boldsymbol{\mu},\boldsymbol{\Sigma}\right) \iff \widehat{\mathbf{y}} = \boldsymbol{\mu} + \mathbf{V}\mathbf{z}, \qquad \mathbf{z} = \begin{pmatrix} z_1 & \dots & z_m \end{pmatrix}^T, \quad z_i \sim \mathcal{N}\left(0,1\right). \tag{17}
$$

Different from [9] we are ultimately interested in sample *functions*. Thus we must consider how to extend this sampling approach to work for multiple inputs $\mathbf{x}^{(1)},\dots,\mathbf{x}^{(N)} \in \mathbb{R}^{n_x}$. These inputs might be queried simultaneously (e.g. when visualizing sample functions on a mesh) or sequentially (e.g. in a Monte Carlo simulation). In any case we need a function $\widehat{\mathbf{f}}(\cdot;\omega) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$, which is a realization of a stochastic process. The random variable ω, its probability space $(\Omega,\mathcal{F},\mathbb{P})$, and the properties of the function are determined by

1) the choice of the matrix $\mathbf{V}$ in Eq. (17), and
2) the sampling strategy for $\mathbf{z}$.

Multiple approaches can generate the correct *pointwise* statistics, but have different implications for the resulting stochastic process. Three possible choices and their advantages and disadvantages are discussed in the following sections. Ultimately we propose a method based on a spectral decomposition of the learned covariance, which yields smooth function realizations retaining the learned cross-covariance at each point.

In Figure 1 we compare function realizations produced with each of the possible sampling schemes discussed in this section. The SCGN used to generate these sample functions models coefficients of pitching moment $C_{m_{cg}}$, drag C_D, and lift C_L, as functions of angle of attack α and velocity FMV^*, for the Orion crew module.

A. Sampling directly with the Cholesky decomposition

Let us start with the first question, how to choose the matrix $\mathbf{V}$ in Eq. (17). If we continue to follow [9] then we would take the most straightforward option, $\mathbf{V} = \mathbf{U}^{-1}$. That is, for a given $\mathbf{z} \in \mathbb{R}^{n_y}$ we sample

$$\widehat{\mathbf{y}} = \boldsymbol{\mu}\left(\mathbf{x}; \boldsymbol{\theta}_\mu\right) + \left[\mathbf{U}\left(\mathbf{x}; \boldsymbol{\theta}_U\right)\right]^{-1} \mathbf{z}. \tag{18}$$

In practice Eq. (18) would be computed by solving a triangular linear system:

$$\left[\mathbf{U}\left(\mathbf{x}; \boldsymbol{\theta}_U\right)\right]\left[\widehat{\mathbf{y}} - \boldsymbol{\mu}\left(\mathbf{x}; \boldsymbol{\theta}_\mu\right)\right] = \mathbf{z}. \tag{19}$$

Now suppose we want to make predictions for multiple points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \in \mathbb{R}^{n_x}$. The most obvious choice is to generate new independent samples $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(N)} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}_{n_y \times n_y}\right)$ for each input. From this we obtain independent linear systems of the form (19), which can be solved in parallel if multiple points are queried simultaneously.

The disadvantage of sampling independent $\mathbf{z}^{(i)}$ is seen immediately in Figure 1. Although this approach captures the learned distribution at each point in space, sample *functions* are nonsmooth (actually discontinuous) with the nonsmoothness only getting worse as we increase the grid resolution. Indeed, the collection of independent normals $\mathbf{z}^{(i)}$ is actually a realization of Gaussian white noise. To see this, let $\{\mathcal{W}(\mathbf{x}; \omega) : \mathbb{R}^{n_x} \times \Omega \to \mathbb{R}^{n_y}\}$ be a Gaussian white noise process and define the stochastic process $\left\{\widehat{\mathbf{f}}(\mathbf{x}; \omega) : \mathbb{R}^{n_x} \times \Omega \to \mathbb{R}^{n_y}\right\}$ by

$$\widehat{\mathbf{f}}\left(\mathbf{x}; \omega\right) = \boldsymbol{\mu}(\mathbf{x}) + \left[\mathbf{U}\left(\mathbf{x}\right)\right]^{-1} \mathbf{z}, \qquad \mathbf{z} = \mathcal{W}(\mathbf{x}; \omega), \tag{20}$$

where we have dropped the dependence on parameters $\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_U$ for brevity. Now for any realization $\mathcal{W}\left(\cdot; \omega_0\right) \sim \{\mathcal{W}(\mathbf{x}; \omega) : \mathbb{R}^{n_x} \times \Omega \to \mathbb{R}^{n_y}\}$ we have

$$\mathbf{z}^{(i)} = \mathcal{W}\left(\mathbf{x}^{(i)}; \omega_0\right) \implies \mathbf{z}^{(i)} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}_{n_y \times n_y}\right), \qquad i = 1, \ldots, N. \tag{21}$$

Evaluating Eq. (20) at each input we obtain

$$\widehat{\mathbf{f}}\left(\mathbf{x}^{(i)}; \omega_0\right) = \boldsymbol{\mu}\left(\mathbf{x}^{(i)}\right) + \left[\mathbf{U}\left(\mathbf{x}^{(i)}\right)\right]^{-1} \mathbf{z}^{(i)}, \qquad i = 1, \ldots, N. \tag{22}$$

From Eqs. (20–22) it is clear that functions generated by sampling independent standard normals for each new input are realizations of a stochastic process driven by white noise. This is undesirable as such functions will not look like candidates for the true physics. Furthermore, if we do not keep track of which $\mathbf{x}^{(i)}$ have been previously observed, then new predictions at the same inputs will be inconsistent with old predictions, hence the resulting function realization would still be stochastic.

B. Pre-sampling standard normal vectors

To alleviate the problem with white noise we can instead generate surrogate function realizations based on single samples $\mathbf{z}_0 \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}_{n_y \times n_y}\right)$. Each sampled $\mathbf{z}_0$ yields a single realization $\widehat{\mathbf{f}}\left(\cdot; \mathbf{z}_0\right) \sim \left\{\widehat{\mathbf{f}}(\mathbf{x}; \mathbf{z}) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_y}\right\}$, where

$$\widehat{\mathbf{f}}\left(\mathbf{x}; \mathbf{z}_0\right) = \boldsymbol{\mu}\left(\mathbf{x}\right) + \left[\mathbf{U}\left(\mathbf{x}\right)\right]^{-1} \mathbf{z}_0. \tag{23}$$

*We use a blended function of Mach and velocity, with velocity in units of 1000 feet per second. See [10, 11, Eq. (1)].

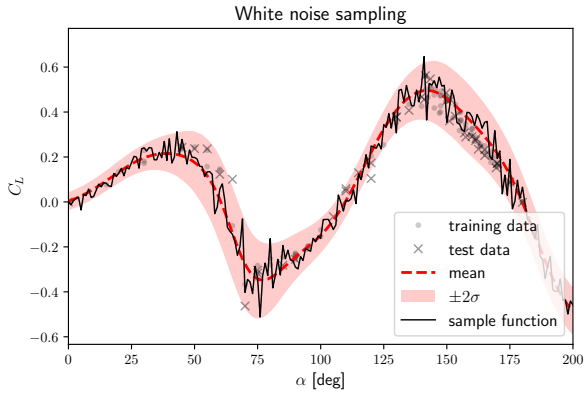
As seen in Figure 1, sample functions generated in this way are continuous and still capture the learned pointwise distribution. Correlation between nearby points will be quantified mathematically in Section III.D. Sampling in this way is analogous to the approach in [9, Figure 9] in our conditional distribution context.

However, this approach reveals a problem with directly using the Cholesky matrix $\mathbf{U}(\cdot)$ for sampling: one of the output variables will always be above or below the mean, and not all variables will be directly cross-correlated. To illustrate this consider the two dimensional case $n_y = 2$ and assume without loss of generality $\boldsymbol{\mu}(\mathbf{x}; \boldsymbol{\theta}_\mu) \equiv \mathbf{0}$. For fixed $\mathbf{z}$, dropping the dependence on parameters $\boldsymbol{\theta}_U$ for ease of notation, Eq. (19) becomes

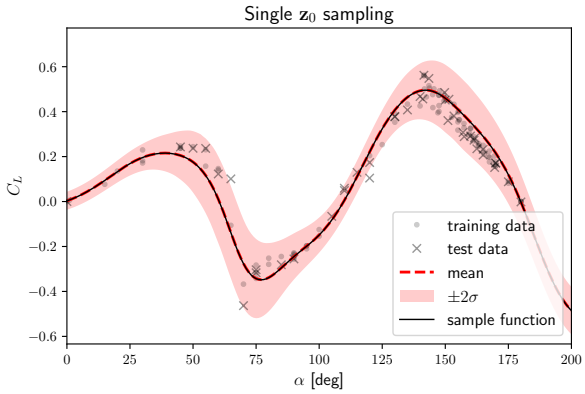
$$\begin{pmatrix} u_{11}(\mathbf{x}) & u_{12}(\mathbf{x}) \\ 0 & u_{22}(\mathbf{x}) \end{pmatrix} \begin{pmatrix} \widehat{y}_1(\mathbf{x}) \\ \widehat{y}_2(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \quad \Longrightarrow \quad \begin{pmatrix} \widehat{y}_1(\mathbf{x}) \\ \widehat{y}_2(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \frac{1}{u_{11}(\mathbf{x})} \left(z_1 - \frac{u_{12}(\mathbf{x})}{u_{22}(\mathbf{x})} z_2 \right) \\ \frac{z_2}{u_{22}(\mathbf{x})} \end{pmatrix}. \tag{24}$$

By construction, $u_{11}(\mathbf{x}), u_{22}(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathbb{R}^{n_x}$, so while the sign of $\widehat{y}_1(\mathbf{x})$ can vary with $\mathbf{x}$ depending on the relative size of $z_1, z_2, u_{12}(\mathbf{x})$, and $u_{22}(\mathbf{x})$, the sign of $\widehat{y}_2(\mathbf{x})$ is fixed and equal to the sign of z_2. Furthermore, the value of $\widehat{y}_2(\mathbf{x})$ also does not depend on the cross-covariance term $u_{12}(\mathbf{x})$. The same pattern holds for higher n_y.

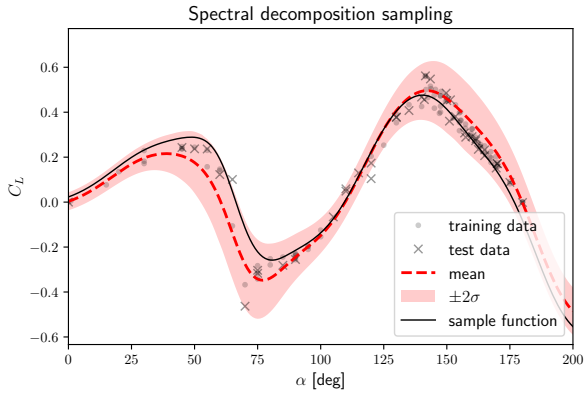
This property implies that the arbitrary ordering of the variables $y_i, i = 1, \ldots, n_y$ affects how the model behaves. Obviously we do not want such an arbitrary choice to affect the modeling results, so another approach is needed.



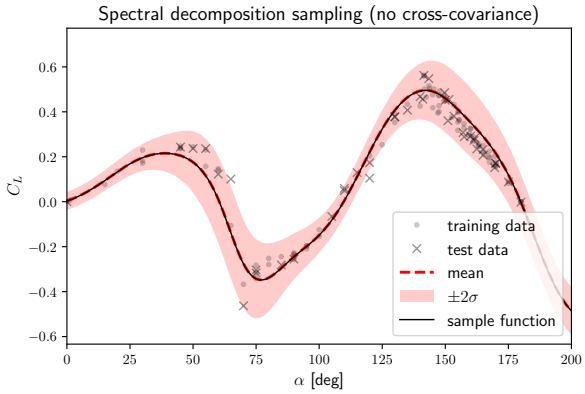
(a) A function sampled using Eq. (22).



(b) A function sampled using Eq. (23). Due to the (arbitrary) order of output variables, namely $C_{m_{cg}}, C_D, C_L$, predictions for C_L with this sample method will not include $C_{m_{cg}}$ and C_D cross-covariance terms.



(c) A function sampled using Eq. (27), with the same random seed as in Figure 1b. Observe that cross-covariance allows sample functions to vary above and below the mean.



(d) A function sampled using Eq. (27), with the first two components of $\mathbf{z}_0$ set to zero to eliminate cross-covariance terms from $C_{m_{cg}}$ and C_D. Note the equivalence to Figure 1b.

Fig. 1 SCGN predictions of $C_L(\alpha, FMV)$ viewed at the slice $FMV = 0.5$. The same SCGN model is used with different function sampling methods in each subfigure.

C. Spectral decomposition sampling

While the order-dependent sampling property (24) discussed above is a result of directly using the Cholesky decomposition for sampling, the learned covariance (10) is not affected by this problem. We recall from Eq. (17) that we can sample $\widehat{\mathbf{y}} = \boldsymbol{\mu} + \mathbf{V}\mathbf{z}$, where $\mathbf{V}$ is *any* matrix satisfying $\mathbf{V}\mathbf{V}^T = \boldsymbol{\Sigma}$. With this in mind we propose a *spectral decomposition* sampling method. This relies on the singular value decomposition (SVD) of $\mathbf{U}(\cdot)$, and can be implemented without changing the NN structure or training algorithm.

For a given pair $(\mathbf{x}, \mathbf{z})$ we can evaluate $\widehat{\mathbf{f}}(\mathbf{x}; \mathbf{z})$ by first computing the SVD

$$\mathbf{U}(\mathbf{x}; \boldsymbol{\theta}_U) = [\mathbf{P}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]\,[\mathbf{Q}(\mathbf{x})]^T, \tag{25}$$

where $\boldsymbol{\Lambda}(\mathbf{x}) = \operatorname{diag}\begin{pmatrix}\lambda_1 & \dots & \lambda_{n_y}\end{pmatrix}$ is the matrix of singular values of $\mathbf{U}(\mathbf{x}; \boldsymbol{\theta}_U)$, and $\mathbf{Q}(\mathbf{x}), \mathbf{P}(\mathbf{x}) \in \mathbb{R}^{n_y \times n_y}$ are orthonormal. Since $\mathbf{U}(\mathbf{x}; \boldsymbol{\theta}_U)$ has n_y positive real eigenvalues by construction, we must have $\lambda_i > 0$ for all $i = 1, \dots, n_y$. Now since $\mathbf{P}(\mathbf{x}), \mathbf{Q}(\mathbf{x})$ are orthonormal we can reconstruct the covariance matrix as

$$\begin{aligned}
\boldsymbol{\Sigma}(\mathbf{x}; \boldsymbol{\theta}_U) &= \left([\mathbf{U}(\mathbf{x}; \boldsymbol{\theta}_U)]^T\, \mathbf{U}(\mathbf{x}; \boldsymbol{\theta}_U)\right)^{-1} \\
&= \left([\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]\,[\mathbf{P}(\mathbf{x})]^T\,[\mathbf{P}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]\,[\mathbf{Q}(\mathbf{x})]^T\right)^{-1} \\
&= [\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}\,[\mathbf{Q}(\mathbf{x})]^T \\
&= [\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}\,[\mathbf{Q}(\mathbf{x})]^T\,[\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}\,[\mathbf{Q}(\mathbf{x})]^T \\
&= \left([\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}\,[\mathbf{Q}(\mathbf{x})]^T\right)\left([\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}\,[\mathbf{Q}(\mathbf{x})]^T\right)^T.
\end{aligned} \tag{26}$$

Observe that this is in fact the *spectral decomposition* of $\boldsymbol{\Sigma}(\mathbf{x}; \boldsymbol{\theta}_U)$, but it is computed without forming $\boldsymbol{\Sigma}(\mathbf{x}; \boldsymbol{\theta}_U)$ itself. Note that $\boldsymbol{\Lambda}(\mathbf{x})$ is diagonal with positive entries, and hence its inverse is trivial to compute. Now we can take

$$\widehat{\mathbf{f}}(\mathbf{x}; \mathbf{z}) = \boldsymbol{\mu}(\mathbf{x}; \boldsymbol{\theta}_\mu) + [\mathbf{V}(\mathbf{x})]\,\mathbf{z}, \qquad \mathbf{V}(\mathbf{x}) = [\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}\,[\mathbf{Q}(\mathbf{x})]^T. \tag{27}$$

As in Section III.B the value of $\mathbf{z}$ must be fixed for each sample function to ensure continuity. Algorithm 1 summarizes our recommended approach.

Algorithm 1 SCGN GP sampling with covariance spectral decomposition

1: **Input:** NN parameters $\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_U$.
2: Sample $\mathbf{z}_0 \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}_{n_y \times n_y}\right)$.
3: **Input:** new test point $\mathbf{x}^*$.
4: Evaluate $\boldsymbol{\mu}\left(\mathbf{x}^*; \boldsymbol{\theta}_\mu\right)$ and $\mathbf{U}\left(\mathbf{x}^*; \boldsymbol{\theta}_U\right)$.
5: Compute the SVD, $\mathbf{U}\left(\mathbf{x}^*; \boldsymbol{\theta}_U\right) = \mathbf{P}\boldsymbol{\Lambda}\mathbf{Q}^T$.
6: $\mathbf{V} = \mathbf{Q}\boldsymbol{\Lambda}^{-1}\mathbf{Q}^T$.
7: $\widehat{\mathbf{y}} = \boldsymbol{\mu}\left(\mathbf{x}^*; \boldsymbol{\theta}_\mu\right) + \mathbf{V}\mathbf{z}_0$.

Evaluating $\widehat{\mathbf{f}}(\cdot)$ in this way preserves the learned covariance. At the same time the order of the variables will not affect what the function realizations look like because $\mathbf{V}(\cdot)$ is not triangular. Computing the spectral decomposition (25) is not prohibitively expensive since in aerodynamic modeling we are dealing with relatively small matrices, say $n_y = O(10)$. In addition, parallel implementations of SVD are available in popular machine learning software packages like *TensorFlow* [15], allowing multiple test predictions to be made simultaneously. Thus this method provides an efficient way to generate continuous, order-independent sample functions which preserve the learned covariance.

Figure 1 illustrates the impact of sampling with Algorithm 1. In particular, if we sample with Algorithm 1 but set the first $n_y - 1$ components of $\mathbf{z}_0$ to zero, we recover the predictions made with (23). In other words, the proposed spectral decomposition sampling approach incorporates the learned cross-covariance which is lost if one directly uses the Cholesky decomposition.

Remark 2 *In principle we could also choose* $\mathbf{V}(\mathbf{x}) = [\mathbf{Q}(\mathbf{x})]\,[\boldsymbol{\Lambda}(\mathbf{x})]^{-1}$ *and recover the correct pointwise covariance. In practice, however, the sampled functions will become discontinuous at points where the SVD solver (arbitrarily) outputs singular vectors* $\mathbf{Q}(\mathbf{x})$ *with signs that are different from neighboring points.*

D. Interpreting SCGN as a Gaussian process

In this section we formally interpret SCGNs as GPs, with the structure of the covariance kernel depending on the choice of sampling approach, Eqs. (22), (23), or (27). The respective covariance kernels are derived in Propositions 1 to 3 below. Throughout this section we consider functions $\boldsymbol{\mu} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ and $\mathbf{U} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y \times n_y}$, where $\mathbf{U}(\mathbf{x})$ is upper triangular with positive diagonal entries for all $\mathbf{x}$. Dependence on parameters $\boldsymbol{\theta}_\mu$ and $\boldsymbol{\theta}_U$ is not needed.

As expected, we find that all of these kernels recover the conditional Gaussian covariance $\boldsymbol{\kappa}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(i)}\right) = \boldsymbol{\Sigma}\left(\mathbf{x}^{(i)}\right)$, but the spatial cross-covariances $\boldsymbol{\kappa}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$ for $\mathbf{x}^{(i)} \neq \mathbf{x}^{(j)}$ are different. Independent sampling (22) yields a block-diagonal covariance kernel, which corresponds to spatially uncorrelated predictions, while the other two methods (23) and (27) yield non-stationary outer product type kernels. Note that these kernels are very different from the "NN kernel" of Bayesian NNs [1, 8].

Proposition 1 (SCGN sampled with Eq. (22) is a GP) *Let $\{\mathcal{W}(\mathbf{x}; \omega) : \mathbb{R}^{n_x} \times \Omega \to \mathbb{R}^{n_y}\}$ be a Gaussian white noise process and define $\widehat{\mathbf{f}}(\mathbf{x}; \omega)$ according to Eq.* (20)*. Then $\widehat{\mathbf{f}}(\cdot) \sim \mathcal{GP}\left(\boldsymbol{\mu}(\cdot), \boldsymbol{\kappa}(\cdot, \cdot)\right)$ with covariance kernel*

$$\boldsymbol{\kappa}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \begin{cases} \left(\left[\mathbf{U}\left(\mathbf{x}^{(i)}\right)\right]^T \mathbf{U}\left(\mathbf{x}^{(i)}\right)\right)^{-1}, & \text{if } i = j, \\ \mathbf{0}_{n_y \times n_y}, & \text{if } i \neq j. \end{cases} \tag{28}$$

Proof: Consider $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \in \mathbb{R}^{n_x}$ and let $\mathcal{W}(\cdot; \omega_0) \sim \{\mathcal{W}(\mathbf{x}; \omega) : \mathbb{R}^{n_x} \times \Omega \to \mathbb{R}^{n_y}\}$. From Eq. (22) we get

$$\widehat{\mathbf{Y}} \coloneqq \begin{pmatrix} \widehat{\mathbf{y}}^{(1)} \\ \vdots \\ \widehat{\mathbf{y}}^{(N)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \boldsymbol{\mu}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} + \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} \mathbf{z}^{(1)} \\ \vdots \\ \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \mathbf{z}^{(N)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \boldsymbol{\mu}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} + \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} & \ldots & \mathbf{0}_{n_y \times n_y} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n_y \times n_y} & \ldots & \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \end{pmatrix} \mathbf{Z},$$

where $\mathbf{Z} \coloneqq \begin{pmatrix} \left[\mathbf{z}^{(1)}\right]^T & \ldots & \left[\mathbf{z}^{(N)}\right]^T \end{pmatrix}^T$. It is clear that $\mathbf{Z}$ is multivariate standard normal, and consequently $\widehat{\mathbf{Y}}$ is multivariate normal with covariance matrix

$$\begin{aligned} \boldsymbol{\Sigma} &= \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} & \ldots & \mathbf{0}_{n_y \times n_y} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n_y \times n_y} & \ldots & \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \end{pmatrix} \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} & \ldots & \mathbf{0}_{n_y \times n_y} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n_y \times n_y} & \ldots & \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \end{pmatrix}^T \\ &= \begin{pmatrix} \left(\left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^T \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]\right)^{-1} & \ldots & \mathbf{0}_{n_y \times n_y} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n_y \times n_y} & \ldots & \left(\left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^T \mathbf{U}\left(\mathbf{x}^{(N)}\right)\right)^{-1} \end{pmatrix}. \end{aligned}$$

Therefore $\widehat{\mathbf{f}}(\cdot) \sim \mathcal{GP}\left(\boldsymbol{\mu}(\cdot), \boldsymbol{\kappa}(\cdot)\right)$ following Definition 2. □

Proposition 2 (SCGN sampled with Eq. (23) is a GP) *For any $\mathbf{z} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}_{n_y \times n_y}\right)$ define $\widehat{\mathbf{f}}(\mathbf{x}; \mathbf{z})$ according to Eq.* (23)*. Then $\widehat{\mathbf{f}}(\cdot) \sim \mathcal{GP}\left(\boldsymbol{\mu}(\cdot), \boldsymbol{\kappa}(\cdot, \cdot)\right)$ with covariance kernel*

$$\boldsymbol{\kappa}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \left(\left[\mathbf{U}\left(\mathbf{x}^{(j)}\right)\right]^T \mathbf{U}\left(\mathbf{x}^{(i)}\right)\right)^{-1}. \tag{29}$$

Proof: Evaluating Eq. (23) at inputs $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)} \in \mathbb{R}^{n_x}$ gives us

$$\widehat{\mathbf{y}}^{(i)} \coloneqq \widehat{\mathbf{f}}\left(\mathbf{x}^{(i)}; \mathbf{z}\right) = \boldsymbol{\mu}\left(\mathbf{x}^{(i)}\right) + \left[\mathbf{U}\left(\mathbf{x}^{(i)}\right)\right]^{-1} \mathbf{z}, \qquad i = 1, \ldots, N.$$

We have

$$\widehat{\mathbf{Y}} \coloneqq \begin{pmatrix} \widehat{\mathbf{y}}^{(1)} \\ \vdots \\ \widehat{\mathbf{y}}^{(N)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \boldsymbol{\mu}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} + \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} \mathbf{z} \\ \vdots \\ \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \mathbf{z} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \boldsymbol{\mu}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} + \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} \\ \vdots \\ \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \end{pmatrix} \mathbf{z}.$$

Since $\mathbf{z}$ is multivariate standard normal it follows that $\widehat{\mathbf{Y}}$ is multivariate normal with covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} \\ \vdots \\ \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \end{pmatrix} \begin{pmatrix} \left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^{-1} \\ \vdots \\ \left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^{-1} \end{pmatrix}^T = \begin{pmatrix} \left(\left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^T \mathbf{U}\left(\mathbf{x}^{(1)}\right)\right)^{-1} & \dots & \left(\left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^T \mathbf{U}\left(\mathbf{x}^{(1)}\right)\right)^{-1} \\ \vdots & \ddots & \vdots \\ \left(\left[\mathbf{U}\left(\mathbf{x}^{(1)}\right)\right]^T \mathbf{U}\left(\mathbf{x}^{(N)}\right)\right)^{-1} & \dots & \left(\left[\mathbf{U}\left(\mathbf{x}^{(N)}\right)\right]^T \mathbf{U}\left(\mathbf{x}^{(N)}\right)\right)^{-1} \end{pmatrix}.$$

Therefore $\widehat{\mathbf{f}}(\cdot) \sim \mathcal{GP}(\boldsymbol{\mu}(\cdot), \boldsymbol{\kappa}(\cdot))$ following Definition 2. □

Proposition 3 (SCGN sampled with Eq. (27) is a GP) *For any* $\mathbf{z} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}_{n_y \times n_y}\right)$ *define* $\widehat{\mathbf{f}}(\mathbf{x}; \mathbf{z})$ *according to Eq.* (27). *Then* $\widehat{\mathbf{f}}(\cdot) \sim \mathcal{GP}(\boldsymbol{\mu}(\cdot), \boldsymbol{\kappa}(\cdot, \cdot))$ *with covariance kernel*

$$\boldsymbol{\kappa}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \left[\mathbf{V}\left(\mathbf{x}^{(i)}\right)\right]\left[\mathbf{V}\left(\mathbf{x}^{(j)}\right)\right]^T, \tag{30}$$

where $\mathbf{V}(\mathbf{x})$ *is constructed from the spectral decomposition of* $\mathbf{U}(\mathbf{x})$ *using Eqs.* (25) *and* (27).

Proof: Evaluating Eq. (27) at inputs $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)} \in \mathbb{R}^{n_x}$ gives us

$$\widehat{\mathbf{y}}^{(i)} := \widehat{\mathbf{f}}\left(\mathbf{x}^{(i)}; \mathbf{z}\right) = \boldsymbol{\mu}\left(\mathbf{x}^{(i)}\right) + \left[\mathbf{V}\left(\mathbf{x}^{(i)}\right)\right]\mathbf{z}, \qquad i = 1, \dots, N.$$

We have

$$\widehat{\mathbf{Y}} := \begin{pmatrix} \widehat{\mathbf{y}}^{(1)} \\ \vdots \\ \widehat{\mathbf{y}}^{(N)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \boldsymbol{\mu}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} + \begin{pmatrix} \left[\mathbf{V}\left(\mathbf{x}^{(1)}\right)\right]\mathbf{z} \\ \vdots \\ \left[\mathbf{V}\left(\mathbf{x}^{(N)}\right)\right]\mathbf{z} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\mu}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \boldsymbol{\mu}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} + \begin{pmatrix} \mathbf{V}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \mathbf{V}\left(\mathbf{x}^{(N)}\right) \end{pmatrix}\mathbf{z}.$$

Since $\mathbf{z}$ is multivariate standard normal it follows that $\widehat{\mathbf{Y}}$ is multivariate normal with covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \mathbf{V}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \mathbf{V}\left(\mathbf{x}^{(N)}\right) \end{pmatrix} \begin{pmatrix} \mathbf{V}\left(\mathbf{x}^{(1)}\right) \\ \vdots \\ \mathbf{V}\left(\mathbf{x}^{(N)}\right) \end{pmatrix}^T = \begin{pmatrix} \left[\mathbf{V}\left(\mathbf{x}^{(1)}\right)\right]\left[\mathbf{V}\left(\mathbf{x}^{(1)}\right)\right]^T & \dots & \left[\mathbf{V}\left(\mathbf{x}^{(1)}\right)\right]\left[\mathbf{V}\left(\mathbf{x}^{(N)}\right)\right]^T \\ \vdots & \ddots & \vdots \\ \left[\mathbf{V}\left(\mathbf{x}^{(N)}\right)\right]\left[\mathbf{V}\left(\mathbf{x}^{(1)}\right)\right]^T & \dots & \left[\mathbf{V}\left(\mathbf{x}^{(N)}\right)\right]\left[\mathbf{V}\left(\mathbf{x}^{(N)}\right)\right]^T \end{pmatrix}.$$

Therefore $\widehat{\mathbf{f}}(\cdot) \sim \mathcal{GP}(\boldsymbol{\mu}(\cdot), \boldsymbol{\kappa}(\cdot))$ following Definition 2. □

IV. Learning aerodynamics for the Orion crew module

In this section we illustrate the application of the SCGN to learn an aerodynamic model for the Orion crew module [10, 11]. We compare the results to a GPR model, which serves as a benchmark UQ method. We demonstrate that SCGN provides competitive nominal accuracy and uncertainty descriptions, while computation time is significantly reduced and scales better with dataset size.

We model coefficients of drag C_D, lift C_L, and pitching moment $C_{m_{cg}}$ as functions of angle of attack α and velocity FMV. We use the same dataset which was used to construct the version 0.60 Orion aerodatabase [10, 11], except that we ignore data with non-zero sideslip angle to focus on the longitudinal aerodynamics. The total dataset contains $N = 5057$ data points with inputs in the range $FMV \in [0.2, 37.7]$ and $\alpha \in [-5°, 200°]$. As we discuss later, some of this data will be used to train the models while the remainder will be reserved for evaluating performance at unseen flight conditions.

Note that the data are obtained from different WTT and CFD sources, correspond to multiple similar vehicle geometries, and are generally of varying quality [10, 11]. Some approaches to multi-fidelity modeling are proposed in [3, 7] and references therein. Since this is not the focus of the present work, in this paper we treat all data as equally valid, with any spread in the data contributing to uncertainty which we would like to capture.

A. Model implementation descriptions

We implement SCGN in *TensorFlow* [15] with the following hyperparameters:

- $\boldsymbol{\mu}\left(\cdot;\boldsymbol{\theta}_\mu\right)$ is a standard feedforward NN with 4 hidden layers, 16 neurons per layer, and tanh activation functions;
- $\mathbf{U}\left(\cdot;\boldsymbol{\theta}_U\right)$ is a standard feedforward NN with 8 hidden layers, 32 neurons per layer, and tanh activation functions;
- ℓ^2 weight regularization loss function weights $\lambda_\mu = \lambda_U = 25 \cdot 10^{-N_{\text{train}}/1000}$; as is common practice, more weight regularization is used to reduce overfitting on the smaller datasets;
- optimization of (13) then (16) using L-BFGS [13].

We compare results to a set of GPRs which independently approximate each variable of interest, $y_d \in \left\{C_D, C_L, C_{m_{cg}}\right\}$. Using independent univariate GPRs in the multivariate setting is common practice [1]. While it is possible to learn cross-covariances with GPR [1], SCGN provides a natural and efficient way to do so. We implement the GPRs in *JAX* [16] and set the following hyperparameters:

- anisotropic squared exponential covariance kernels of the form

$$\kappa\left(\mathbf{x}, \mathbf{x}'\right) = \sigma^2 \exp\left[-\frac{1}{2}\sum_{j=1}^{n_x}\left(\frac{x_j - x'_j}{\ell_j}\right)^2\right], \tag{31}$$

  where $\sigma, \ell_1, \ldots, \ell_{n_x} > 0$ are trainable parameters;
- observation noise hyperparameter $\sigma_n = 0.1$, see [1, 3, 17] for details;
- optimization of $\sigma > 0$, $\ell_1, \ldots, \ell_{n_x} \in [0.1, 10]$ by maximizing the log marginal likelihood [1, Eq. (2.30)] using L-BFGS [13].

This is a typical setup for GPR and provides a reasonable baseline for UQ accuracy. Further improved approximation accuracy might be obtained by finding specialized covariance kernels [2], but this would not change the computational advantage provided by a NN.

Since the data are densest in the low FMV regime (subsonic, transonic), and this is where the aerodynamics change most rapidly, we improve predictive performance for both GPR and SCGN by working with $\log FMV$ instead of FMV. For both methods we also apply affine scaling to the input and output variables, which is standard practice for improving numerical optimization results. Input variables are scaled such that training data are mapped to $[-1, 1] \times [1, 1]$, and output variables are scaled such that training data have zero mean and unit variance. Learning and prediction is done in the scaled domain, and predictions are mapped back to the original domain for plotting and test metric evaluation.

Finally, to make a fair comparison with SCGN, we have the GPR output *prediction intervals* which aim to cover the observed data. In practice this means that for computing percentiles we add σ_n^2 to the diagonal of the GPR kernel covariance matrix [1]. To sample smooth functions from this prediction interval we generate function realizations in the usual way [1], but then perturb each sample function by sampling a Gaussian random variable with variance σ_n^2.

B. Evaluation metrics

We consider the following test accuracy metrics for model evaluation. Test metrics are evaluated on a separate test dataset which the model does not observe during training. This provides an unbiased measure of model performance.

For the nominal prediction we report the mean absolute error (MAE) for each variable of interest, $y_d \in \left\{C_D, C_L, C_{m_{cg}}\right\}$:

$$\text{MAE} \coloneqq \frac{1}{N_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}\left|y_d^{(i)} - \mathbb{E}\left\{\widehat{y}_d\left(\mathbf{x}^{(i)}\right)\right\}\right|. \tag{32}$$

Here N_{test} denotes the number of test data.

Evaluating uncertainty predictions is less straightforward; see e.g. [18, 19] for a review of some possible metrics. In this paper we simply report the percentile score (also called "pinball loss" [17]),

$$k\text{th pctl. score} \coloneqq \frac{1}{N_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}\left[\frac{k}{100}\max\left\{y_d^{(i)} - \widehat{y}_{d,k}^{(i)}, 0\right\} + \left(1 - \frac{k}{100}\right)\max\left\{\widehat{y}_{d,k}^{(i)} - y_d^{(i)}, 0\right\}\right], \tag{33}$$

where $\widehat{y}_{d,k}^{(i)}$ denotes the predicted kth percentile of y_d at the test point $\mathbf{x}^{(i)}$. A smaller percentile score indicates a better estimation of the uncertainty. Note that the 50th pctl. score corresponds to $1/2$ times the MAE.

Lastly we report the log-likelihood ($\ell\ell$) of the test data, normalized by the number of data:

$$\ell\ell \coloneqq -\frac{1}{2N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left(\left[\mathbf{y}^{(i)} - \boldsymbol{\mu}\left(\mathbf{x}^{(i)}\right) \right]^T \left[\boldsymbol{\Sigma}\left(\mathbf{x}^{(i)}\right) \right]^{-1} \left[\mathbf{y}^{(i)} - \boldsymbol{\mu}\left(\mathbf{x}^{(i)}\right) \right] + \log\det \boldsymbol{\Sigma}\left(\mathbf{x}^{(i)}\right) + n_y \log 2\pi \right). \tag{34}$$

This metric is straightforward to evaluate for conditional Gaussian distributions and provides a combined assessment of the nominal prediction accuracy and UQ quality, with a higher $\ell\ell$ indicating better performance.

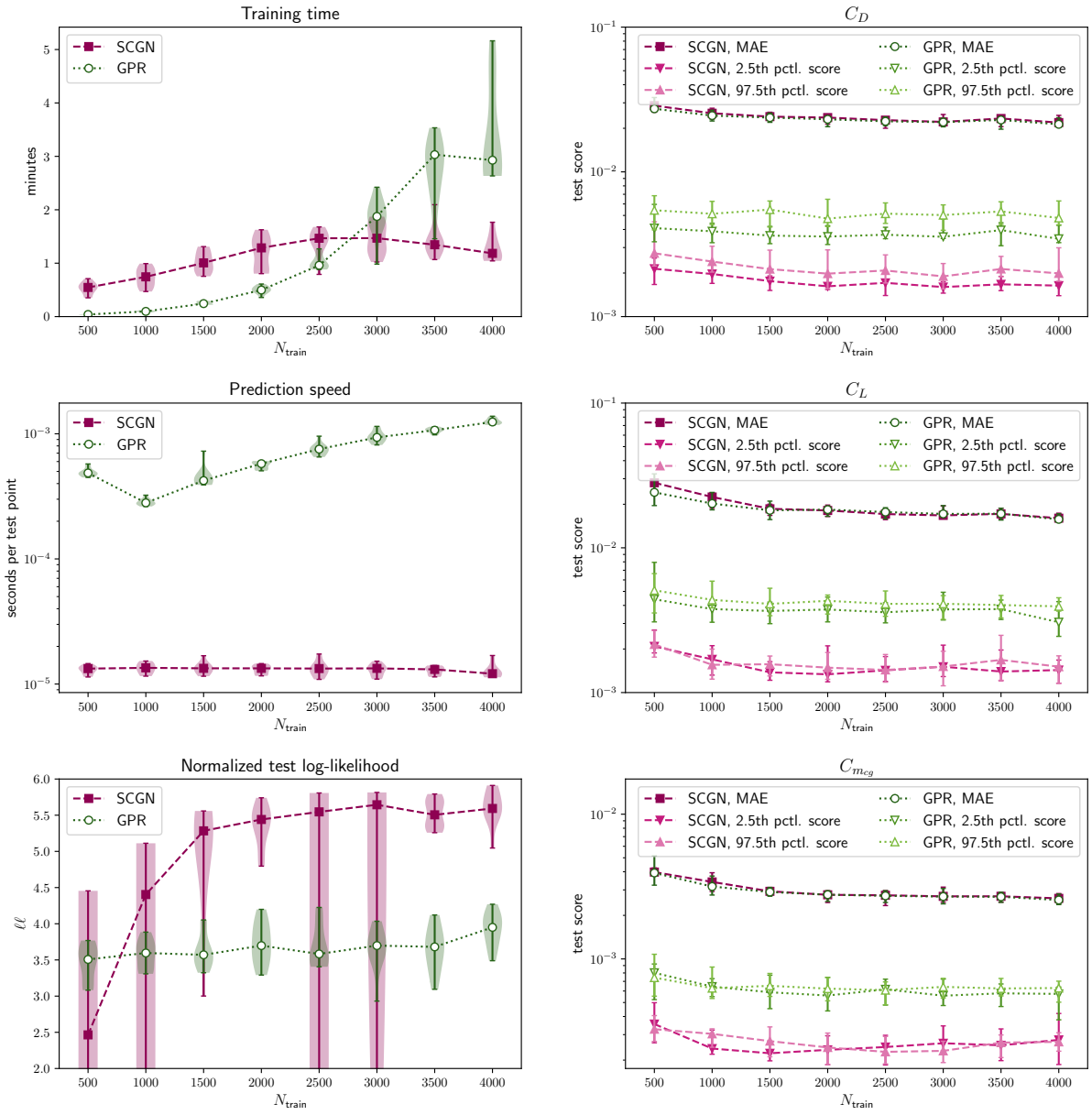


Fig. 2 Comparison of computational efficiency and accuracy of SCGN and GPR models of the Orion crew module aerodynamics, depending on the number of training data, N_{train}. Markers indicate the median and error bars are the minima and maxima over 10 repeated trials. Violin plots are not shown in figures on the right to limit clutter. Training and prediction time are for models are run on a 2019 Macbook Pro.

C. Numerical results

We now present the training time, prediction time on the test data set, and model accuracy (based on the metrics defined in Section IV.B) for SCGN and GPR, implemented as described in Section IV.A. We study the sensitivity of nominal accuracy, UQ quality, and computational effort to the number of training data, N_{train}. For this purpose we reserve $N_{\text{test}} = 1057$ data for testing and vary N_{train} from 500 to 4000. We account for random variations in the train/test data split and model parameter initializations by cross-validation. Specifically, we repeat the experiment 10 times for each N_{train}, each time with different random data splits and parameter initializations. Results are shown in Figure 2.

We find that the nominal accuracy (as measured by the MAE) is virtually identical between SCGN and the baseline GPR. SCGN tends to provide better UQ (as measured by the $\ell\ell$ and percentile scores), with results improving as N_{train} increases, whereas GPR is more robust to overfitting (based on the spread of test $\ell\ell$ scores) when N_{train} is small. *The main advantage of SCGN is that training and prediction times scale well with the size of the dataset*, while GPR scales poorly since it requires inverting a covariance matrix of the entire dataset during training and each time it is evaluated. This is a well-known limitation of GPR [1]. On the other hand, $N_{\text{train}} = 4000$ is a very small dataset in the context of deep learning. Furthermore, NN training takes place offline so prediction time is unaffected by the dataset size and can thus be order of magnitudes faster than GPR. This makes SCGN suitable for use in trajectory simulations where sample functions are called many times.

To visualize the impact of accuracy and UQ quality on the predicted variables we plot predicted distributions and sample functions for SCGN and GPR models trained on $N_{\text{train}} = 4000$ data. Figure 3 shows 2.5th to 97.5th percentile prediction intervals for C_D, C_L, and $C_{m_{cg}}$ in the subsonic and transonic region. A closer look is given in Figure 4 for a slice of the distribution at $FMV = 0.5$, where sample function realizations have been generated with the spectral decomposition approach (Algorithm 1). These plots illustrate that SCGN effectively learns nonlinear input-output relationships in the data while capturing the spread.

V. Discussion

In this paper we have proposed SCGN as a simple and potentially effective approach to perform UQ. The model is constructed as a GP parameterized by two NNs; one NN learns the mean and the other learns a Cholesky decomposition of the inverse covariance. Numerical results suggest that the method is able to accurately capture nonlinear features in the data and produce well-calibrated uncertainty. Nominal prediction accuracy and UQ performance are shown to be competitive with and sometimes superior to widely-used GPR models.

We have also introduced an algorithm to efficiently generate sample function realizations. These sample functions are continuous and recover the full, dense covariance at each point. These properties are important in the context of aerodynamic modeling, where random aerodynamic response surfaces may be employed in trajectory simulations. These surfaces need to be physically reasonable and fast to evaluate at different flight conditions. By comparison, GPR is slow to evaluate with large training sets and does not provide a simple way to evaluate consistent function realizations online at new points.

The purpose of these comparisons is not claim that SCGN is superior to GPR or other UQ methods. Our aim is simply to demonstrate that SCGN is competitive with state-of-the-art methods and can be useful for aerodynamic modeling. Ultimately, each method comes with important advantages and tradeoffs which should be weighed for the application at hand.

In future work we plan to study possible methods for incorporating prior knowledge of data fidelity and measurement noise into model training. We will also more thoroughly test SCGN and compare the results with other UQ algorithms like Bayesian NNs [8] and GPR with more sophisticated kernel selection [2]. Such tests should illuminate some advantages and disadvantages of SCGN relative to other UQ methods, and thereby allow us to further improve the methodology. We expect that SCGN will continue to prove competitive and could become a useful tool for many UQ tasks.

Acknowledgments

We would like to thank Karen L. Bibb for providing data for the Orion crew module and many helpful discussions. We would like to thank Steven Snyder and T. J. Wignall for their feedback during the development of this work. Finally we would like to thank the Space Technology Mission Directorate Early Career Initiative program for funding this work.

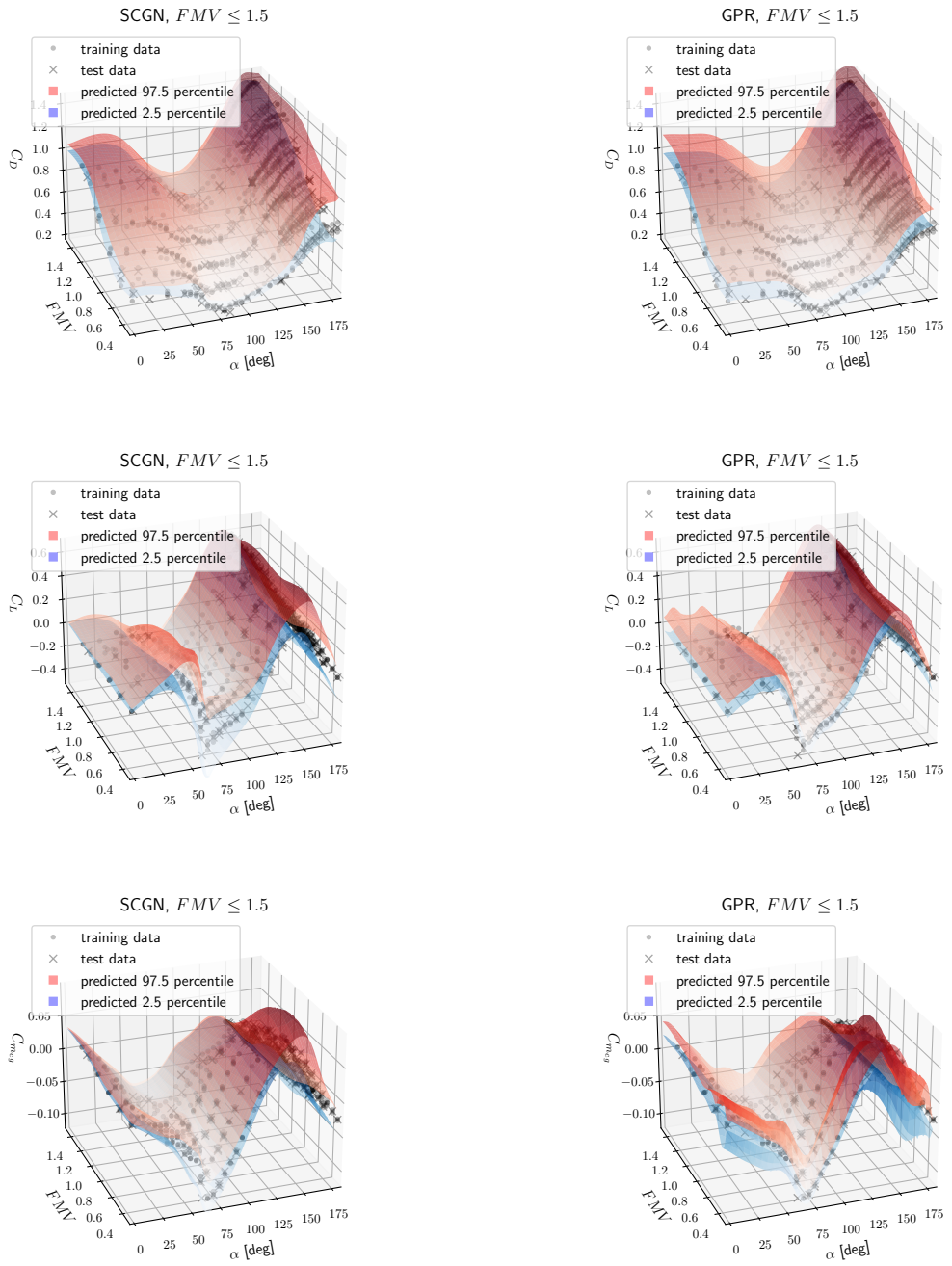


Fig. 3 2.5th to 97.5th percentile prediction intervals for sub and transonic speeds.

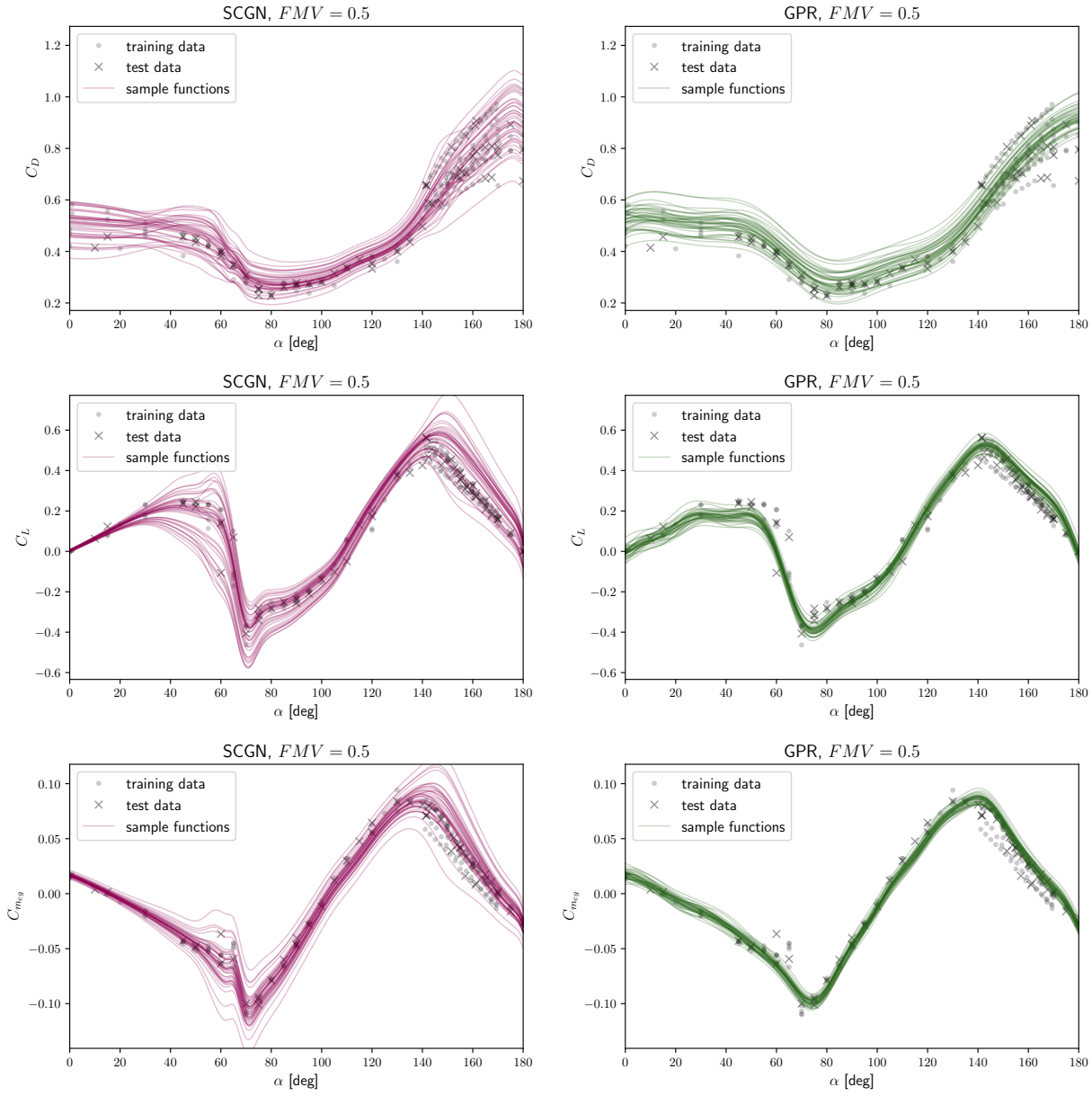


Fig. 4 Sample function realizations evaluated at a slice $FMV = 0.5$.

References

[1] Rasmussen, C. E., and Williams, C. K. I., *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, Massachusetts, 2006.

[2] Palar, P. S., Zakaria, K., Zuhal, L. R., Shimoyama, K., and Liem, R. P., "Gaussian processes and support vector regression for uncertainty quantification in aerodynamics," *AIAA SciTech Forum*, 2021, pp. 1–12. https://doi.org/10.2514/6.2021-0181.

[3] Scoggins, J. B., Wignall, T. J., Nakamura-Zimmerer, T., and Bibb, K., "Multihierarchy Gaussian process models for probabilistic aerodynamic databases using uncertain nominal and off-nominal configuration data," *AIAA SciTech Forum (to appear)*, 2023.

[4] Xiu, D., *Numerical Methods for Stochastic Computations: A Spectral Method Approach*, Princeton University Press, Princeton, NJ, 2010.

[5] Mirza, M., and Osindero, S., "Conditional generative adversarial nets," *arXiv:1411.1784 [cs.LG]*, 2014. https://doi.org/10.48550/ARXIV.1411.1784.

[6] Sohn, K., Lee, H., and Yan, X., "Learning structured output representation using deep conditional generative models," *Advances in Neural Information Processing Systems*, Vol. 28, 2015.

[7] Yang, Y., and Perdikaris, P., "Conditional deep surrogate models for stochastic, high-dimensional, and multi-fidelity systems," *Computational Mechanics*, Vol. 64, No. 2, 2019, pp. 417–434. https://doi.org/10.1007/s00466-019-01718-y.

[8] Neal, R. M., *Bayesian Learning for Neural Networks*, Springer, New York, NY, 1996. https://doi.org/10.1007/978-1-4612-0745-0.

[9] Dorta, G., Vicente, S., Agapito, L., Campbell, N. D., and Simpson, I., "Structured uncertainty prediction networks," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5477–5485. https://doi.org/10.1109/CVPR.2018.00574.

[10] Bibb, K., Brauckmann, G., Walker, E., and Robinson, P., "Development of the Orion crew module static aerodynamic database, part I: Hypersonic," *29th AIAA Applied Aerodynamics Conference*, 2011, pp. 1–23. https://doi.org/10.2514/6.2011-3506.

[11] Bibb, K., Walker, E., Brauckmann, G., and Robinson, P., "Development of the Orion crew module static aerodynamic database, part II: Subsonic/Supersonic," *29th AIAA Applied Aerodynamics Conference*, 2011, pp. 1–34. https://doi.org/10.2514/6.2011-3507.

[12] Snyder, S., Wignall, T. J., Green, J. S., Kumar, S. G., Lee, M. W., Nakamura-Zimmerer, T., Scoggins, J. B., Stringer, M. T., and Williams, R. A., "AeroFusion: Data fusion and uncertainty quantification for lander vehicles," *AIAA SciTech Forum (to appear)*, 2023.

[13] Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C., "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientific Computing*, Vol. 16, No. 5, 1995, pp. 1190–1208. https://doi.org/10.1137/0916069.

[14] Bottou, L., Curtis, F. E., and Nocedal, J., "Optimization methods for large-scale machine learning," *SIAM Review*, Vol. 60, No. 2, 2018, pp. 223–311. https://doi.org/10.1137/16M1080173.

[15] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-scale machine learning on heterogeneous systems," Tech. rep., Google Research, 2015. URL https://www.tensorflow.org/.

[16] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q., "JAX: Composable transformations of Python+NumPy programs," http://github.com/google/jax, 2018.

[17] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-Learn: machine learning in Python," *Journal of Machine Learning Research*, Vol. 12, No. 85, 2011, pp. 2825–2830.

[18] Tran, K., Neiswanger, W., Yoon, J., Zhang, Q., Xing, E., and Ulissi, Z. W., "Methods for comparing uncertainty quantifications for material property predictions," *Machine Learning: Science and Technology*, Vol. 1, No. 2, 2020, pp. 1–13. https://doi.org/10.1088/2632-2153/ab7e1a.

[19] Wignall, T. J., Nakamura-Zimmerer, T., Scoggins, J. B., Snyder, S., Kumar, S. G., and Colbert, B. K., "Comparisons of performance metrics and machine learning methods on an entry descent and landing database," *AIAA SciTech Forum (to appear)*, 2023.