Author version

Published as:

Wang, B., Sperry, M., Gandarillas, V. E., & Hwang, J. T. (2022). Efficient uncertainty propagation through computational graph modification and automatic code generation. In AIAA Aviation 2022 Forum. https://lsdolab.github.io/lsdo-publications/publication/wang2022efficient/

BibTeX for this reference:

https://lsdolab.github.io/lsdo-publications/publication/wang2022efficient/

```
@inproceedings{wang2022efficient,
  author = {Bingran Wang and Mark Sperry and Victor E. Gandarillas and John T. Hwang},
  title = {Efficient Uncertainty Propagation Through Computational Graph Modification and
    Automatic Code Generation},
  booktitle = {AIAA Aviation 2022 Forum},
  year = {2022}
}
```

Efficient uncertainty propagation through computational graph modification and automatic code generation

Bingran Wang*, Mark Sperry†, Victor E. Gandarillas‡, and John T. Hwang§
University of California, San Diego, La Jolla, CA, 92093

Uncertainty quantification (UQ) methods such as integration-based non-intrusive polynomial chaos (NIPC) and stochastic collocation often use a tensor-product grid to sample the space of random inputs. In these situations, the number of quadrature points increases exponentially with the number of uncertain inputs. If we view the model as a computational graph with only elementary functions and operations, the current framework evaluates each operation node in the graph for all of the quadrature points. However, each node in the model only has distinct values on the quadrature points from the uncertain inputs on which the operation's output depends. This means that the current framework could create many unnecessary repeated evaluations on certain operation nodes. In this paper, we propose a new method that uses a computational graph representation of the model to algorithmically remove these unnecessary repeated evaluations so that the evaluation cost on full-grid quadrature points can be significantly reduced. This method is implemented in the Computational System Design Language, an algebraic modeling language, to achieve the cost reduction on UQ problems in a general way. The process of detecting and achieving savings from the model structure is abstracted by the modeling language and its back-end. This method is applied to a multi-point aircraft mission analysis problem with two uncertain inputs. Our method reduces the computing time of the integration-based NIPC method by an order of magnitude. For a wide range of forward UQ problems, we expect this method to significantly improve time scaling with the number of the uncertain inputs.

I. Introduction

Engineers use numerical models to simulate, analyze and design engineering systems. In practice, numerical models are never perfect representations of reality, and they are always subject to uncertainties. Incorporating the uncertainties into the numerical models and determining their effect is the process of uncertainty quantification (UQ). Applications of UQ can be found in many disciplines, including biology [1], chemistry [2]; hydrology [3, 4]; fluid [5, 6] and aerodynamics [7]. There are mainly two types of problems in uncertainty quantification: forward problems and inverse problems. In forward problems, we are given the uncertainties in the model, and the problem is to determine their effect on the output. In inverse problems, we are given some experimental measures of a system, and the problem is to estimate the parameter uncertainties in the numerical models.

Here, we consider a forward uncertainty quantification problem with a small number of uncertain inputs. For such a problem, non-intrusive polynomial chaos (NIPC) and stochastic collocation (SC) methods are the most used non-intrusive methods. The NIPC method approximates the output of the model using orthogonal polynomials of the uncertain inputs, and the weights are calculated by using either integration or regression methods. The integration problems are solved using Gauss quadrature rules, requiring that we evaluate the model at (in the worst case) the full tensor-product grid of quadrature points of the uncertain inputs. The SC method forms Lagrangian interpolation functions to approximate the model output from a set of collocation points. The collocation points are typically chosen the same way as the full tensor-product quadrature points in the integration-based NIPC method. This means both integration-based NIPC and SC methods could require the evaluation results of the output on the full-grid quadrature points of the uncertain inputs.

In the current framework of NIPC and SC, the model is treated as a black-box function, and is repeatedly evaluated at each of the input sample points. The computational cost would therefore increase exponentially with the number of

*Ph.D Student, Department of Mechanical and Aerospace Engineering, AIAA Student Member.
†Ph.D Student, Department of Mechanical and Aerospace Engineering, AIAA Student Member.
‡Ph.D Candidate, Department of Mechanical and Aerospace Engineering, AIAA Member.
§Assistant Professor, Department of Mechanical and Aerospace Engineering, AIAA Member.

uncertain inputs. If we view the numerical model as a computational graph with only elementary function and operation, in the current framework, each function and operation is evaluated for the same number of times at the tensor-product quadrature points of all the uncertain inputs. However, for each function/operation, the output data only has distinct values on the quadrature points of the uncertain inputs it is dependent on. This means the current framework of NIPC and SC could create a lot of wasteful evaluations on certain operations in the computational graph.

In this paper, we propose a graph modification method that automatically modifies the computational graph to reduce the unnecessary evaluations incurred by the current framework of integration-based NIPC and SC. This method reduces the model evaluation cost on full-grid quadrature points, by controlling the size of the input data that pass into each operation node and uses tensor-algebra operations to modify the input data size when necessary. The proposed method is implemented in a new algebraic modeling language, Computational System Design Language (CSDL). We achieve cost reduction on UQ problems in a general and automatic way by taking advantage of the modeling language and its back-end algorithm.

This method is applied to an UQ problem of the multi-point mission analysis of an eVTOL aircraft. Applying the graph modification method reduces the computing time by more than an order of magnitude, and allow the integration-based NIPC to have the best performance among sparse-grid NIPC and Monte Carlo methods.

This paper is organized as follows. In Section II, we give some background on the UQ problems, PCE-related UQ methods; computational graph along with the motivation of our method. In Section III, we present the outline of the graph modification algorithm and the how it is implemented in CSDL. In Section IV, we show the numerical results multi-point flight mission UQ problem for an eVTOL aircraft. The conclusion is made in Section V.

II. Background

A. Uncertainty quantification

The standard forward problem in uncertainty quantification (UQ) aims to determine how an output of a deterministic model is affected by the uncertainty in the inputs. The uncertainties can be broadly divided into aleatoric uncertainty and epistemic uncertainty. Aleatoric uncertainty is inherent to the problem and often defined in a probabilistic framework, e.g., random variations in operating conditions or in model parameters. Epistemic uncertainty is reducible uncertainty, which often arises from the lack of knowledge and from model simplifications. For example, the errors associated with the computational model and round-off errors are epistemic uncertainties, and they are difficult to define in a probabilistic framework. In this paper, we consider a forward UQ problem in which the uncertain inputs are characterized as continuous random variables with known probability density distribution. For the output, which also becomes continuous random variables, if the integral convergence is defined, it is possible to characterize them using their statistical moments: e.g., expected value, $\mathbb{E}[\cdot]$; variance, $\mathbb{V}[\cdot]$; or standard deviation, $\mathbb{S}[\cdot]$. Other risk measures include probability of failure, value at risk at level β, $\text{VaR}_\beta[\cdot]$; and conditional value at risk at level β, $\text{CVaR}_\beta[\cdot]$.

This type of UQ problem is also an essential sub-problem in optimization under uncertainty (OUU) problems. In OUU problems, we solve the forward UQ problem in each optimization iteration, and we optimize the design of a system to have minimal risk. There are two types of OUU problems: robust design optimization (RDO) and reliability-based design optimization (RBDO) [8] . The difference between these two methods lies in the treatment of constraints. RBDO adds probabilistic constraints based on the probability of failure. On the other hand, RDO considers the effects of uncertainty in the objective and constraints through statistical moments, e.g., mean plus variance.

Uncertainty quantification is a broad research field, and many approaches have been developed or are the subject of ongoing investigation. There are four standard non-intrusive UQ methods: method of moments, Monte Carlo, kriging and polynomial chaos. Method of moments is based on the Taylor series expansion of the quantity of interest (QoI). It avoids sampling and only uses one model evaluation and first-derivative information to compute the statistical moments of the output. If derivatives are computed using the adjoint method, the cost of UQ using the method of moments can be as cheap as the cost of two function evaluations [9]. However, it can only compute the output's statistical moments and lacks accuracy when the variance of the uncertain inputs is significant. The Monte Carlo method samples the space of uncertain inputs and performs many model evaluations to compute the corresponding outputs. Due to the law of large numbers, the convergence rate is independent of the dimension of the uncertain inputs. Thus, the Monte Carlo method is typically used for large-scale UQ, and many efforts have been made to improve its efficiency. In [10] and [11], Ng and Willcox reuse information from previous optimization iterations to reduce the number of Monte Carlo evaluations required in the current iteration. In [12], Peherstorfer et al. find an optimal way to perform a multi-fidelity Monte Carlo estimation. These methods can significantly reduce the overall computational cost of the Monte Carlo method. However,

the Monte Carlo method still requires a large number of sampling points and is more computationally expensive than the other UQ methods when the dimension of the random space is small. Kriging [13] constructs a surrogate model using Gaussian process regression, while polynomial chaos represents the QoI using orthogonal polynomials of input variables. Readers interested in more details on kriging are referred to [14]. These two methods are the most efficient and frequently used for UQ problems with small to medium number of uncertain inputs. However, both methods still suffer from the curse of dimensionality; the computational cost dramatically increases as the number of uncertain inputs increases. In this paper, we will focus on the polynomial-chaos-based non-intrusive UQ methods, including stochastic collocation (SC) [15, 16] and non-intrusive polynomial chaos (NIPC) [17].

1. Polynomial chaos theory

Wiener first proposed polynomial chaos or polynomial chaos expansion (PCE) using Hermite polynomials to represent the model response with Gaussian uncertain inputs [18]. This idea was extended by Xiu and Karniadakis in [17] by proposing the generalized polynomial chaos (gPC) method. In gPC, model responses are selected from the space spanned by orthogonal polynomials of the uncertain inputs. For each of a number of standard input distributions, the corresponding orthogonal polynomials are derived to ensure an exponential convergence rate.

Consider a general UQ problem,

$$f = \mathcal{F}(U), \tag{1}$$

where $U = (U_1, \ldots, U_n) \in \mathbb{R}^n$ are a set of mutually independent random variables with support range I_U, $f \in \mathbb{R}$ is the model output which is also a random variable and $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}$ is the model evaluation function. In gPC, the model output, f, is defined as a linear combination of orthogonal polynomials of the uncertain input variables:

$$f(U) = \sum_{i=0}^{\infty} \alpha_i \Phi_i(U), \tag{2}$$

where Φ_i are the PCE basis functions, and α_i are the corresponding weights. In practice, one often truncates it to polynomial order p, resulting in $d + 1$ terms:

$$f(U) \approx \sum_{i=0}^{d} \alpha_i \Phi_i(U), \tag{3}$$

with

$$d + 1 = \frac{(n + p)!}{n!p!}. \tag{4}$$

The PCE basis is chosen to satisfy the orthogonal property, i.e.

$$\left\langle \Phi_i, \Phi_j \right\rangle = \delta_{ij}, \tag{5}$$

where δ_{ij} is the Kronecker delta. In the multi-dimensional case, the PCE basis functions are tensor products of one-dimensional orthogonal polynomials. For example, the first six PCE basis functions in the two-dimensional case are

$$\begin{aligned}
\Phi_0(U) &= \phi_0(U_1)\phi_0(U_2) \\
\Phi_1(U) &= \phi_1(U_1)\phi_0(U_2) \\
\Phi_2(U) &= \phi_0(U_1)\phi_1(U_2) \\
\Phi_3(U) &= \phi_2(U_1)\phi_0(U_2) \\
\Phi_4(U) &= \phi_1(U_1)\phi_1(U_2) \\
\Phi_5(U) &= \phi_0(U_1)\phi_2(U_2).
\end{aligned}$$

The one-dimensional orthogonal polynomials $\phi_i(U_j)$ are chosen based on the type of the input random variable U_j. The corresponding orthogonal polynomials for some common types of uncertainty can be found in Table 1. Once the PCE coefficients are determined, statistical moments such as the mean and standard deviation of the model output can be easily computed as

$$\mu_f = \alpha_0, \tag{6}$$

$$\sigma_f = \sum_{i=0}^{d} \alpha_i^2 \left\langle \Phi_i^2 \right\rangle. \tag{7}$$

Table 1 Orthogonal polynomials and weight functions for common continuous probability distributions

Distribution	Polynomials	Weight function	Support range
Normal	Hermite	$e^{-\frac{x^2}{2}}$	$(-\infty, \infty)$
Uniform	Legendre	1	$[-1, 1]$
Exponential	Laguerre	e^{-x}	$[0, \infty)$
Beta	Jacobi	$(1-x)^{\alpha}(1+x)^{\beta}$	$(-1, 1)$
Gamma	Generalized Laguerre	$x^{\alpha}e^{-x}$	$[0, \infty)$

2. Non-intrusive polynomial chaos method

The non-intrusive polynomial chaos (NIPC) method solves the PCE coefficients α_i in (3) by either integration or regression. The integration approach uses the orthogonal property in (5) and projects the model output onto each basis function:

$$\alpha_i = \frac{\left\langle f(U), \Phi_j \right\rangle}{\left\langle \Phi_i^2 \right\rangle} = \frac{1}{\left\langle \Phi_i^2 \right\rangle} \int_U f(U)\Phi_i(U)\rho(U)dU. \tag{8}$$

This requires solving a multi-dimensional integration problem, which is often done by using the Gauss quadrature approach:

$$\begin{aligned} \alpha_i &= \frac{1}{\left\langle \Phi_i^2 \right\rangle} \int_{U_1} \dots \int_{U_n} f(U_1, \dots, U_n)\Phi_k(U_1, \dots, U_n)p(U_1)\dots p(U_n)du_1 \dots dU_n \\ &\approx \frac{1}{\left\langle \Phi_i^2 \right\rangle} \sum_{r_1=0}^{k} \dots \sum_{r_n=0}^{k} w_1^{r_1} \dots w_n^{r_n} f(U_1^{r_1}, \dots, U_n^{r_n})\Phi_i(U_1^{r_1}, \dots, U_n^{r_n})\rho(U_1^{r_1}, \dots, U_n^{r_n}), \end{aligned} \tag{9}$$

where k is the number of quadrature points used in each dimension. This requires us to evaluate the model output at the full tensor product quadrature points

$$\Theta = \Theta_1^k \times \dots \times \Theta_n^k, \tag{10}$$

where

$$\Theta_j^k := \{U_j^{r_j}\}_{r_j=1}^k \tag{11}$$

is the set of quadrature points at U_j dimension. The total number of quadrature points is thus k^n, and this number increases exponentially with respect to the number of random input variables. For high-dimensional problems, one way to mitigate the exponential growth of quadrature points is to use the Smolyak sparse grid approach [19], which drops the higher-order cross terms in the quadrature points with minimal loss on accuracy. The sparse grid quadrature points can be expressed as:

$$\Theta = \bigcup_{\ell-n+1 \le |\mathbf{i}| \le \ell} \left(\Theta_1^{i_1} \times \dots \times \Theta_n^{i_n} \right), \tag{12}$$

where ℓ is the level of construction.

The regression approach first generates m sample points for U and evaluates the model for each sample point. The coefficients are determined by solving a linear least-squares problem:

$$\begin{bmatrix} \Phi_1(U^1) & \dots & \Phi_d(U^1) \\ \vdots & & \vdots \\ \Phi_1(U^m) & \dots & \Phi_d(U^m) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix} = \begin{bmatrix} F(U^1) \\ \vdots \\ F(U^m) \end{bmatrix}.$$

The rule of thumb is that the number of samples m needs to be 2-3 times the number of coefficients $d+1$. This means the cost of the regression-based NIPC method can be reduced by using the sparse PCE basis, resulting in fewer coefficients to be estimated. A recent review paper for sparse PCE is presented in [20].

3. Stochastic collocation

Stochastic collocation (SC) is another PCE-related UQ method. It forms Lagrange interpolation functions from a set of collocation points as follows:

$$L_j(U) = \prod_{i=1, i\neq j}^{M} \frac{U - U^i}{U^j - U^i}, \tag{13}$$

where $L_j = 1$ at $U = U_j$ and $L_j = 0$ for all other collocation points. The model output is approximated as the sum of the Lagrange interpolation functions with the weights equal to the model output evaluations at each collocation point:

$$f(U) \approx \sum_j f(U^j) L_j(U). \tag{14}$$

SC typically chooses the set of collocation points the same way as quadrature points are chosen in NIPC. The full-grid collocation points are in (10), and sparse grid collocation points are in (12). This means SC evaluates the model in the same way as the integration-based NIPC. The difference is that NIPC computes the PCE coefficients by solving the integration problem, while SC forms Lagrangian interpolation functions to approximate the model output. In [21], Eldred and Burkardt compared the standard NIPC and SC methods and found similar performance in most of the cases, although SC had a small edge over NIPC for problems with a dimension greater than two.

B. Model representation using a computational graph

Any computer program that describes a computational model is a computational process that executes a sequence of elementary functions and operations. A computational process can be described by a computational graph. A computational graph is a directed acyclic graph in which the nodes represent the operations and variables, and the directed edges represent how operations and variables are connected to each other. Such a graph is also bipartite because each operation is connected to an output variable, and likewise, variables are connected to operations (as arguments). For example, the computer program that evaluates a function $f = cos(u_1) + exp(-u_2)$ can be decomposed into the following set of fundamental operations:

$$\begin{aligned} \xi_1 &= cos(u_1); \\ \xi_2 &= -u_2; \\ \xi_3 &= exp(\xi_2); \\ f &= \xi_3 + \xi_2. \end{aligned} \tag{15}$$

The computational graph for this model is shown in Fig. 1 with the blue nodes representing the variables and the orange nodes representing the operations. A common use of computational graphs is in automatic differentiation (AD) [22] to automatically compute the total derivatives. The AD method views the computer program as a sequence of operations and functions and repeatedly applies the chain rule to compute the total derivative. The Autograd software package [23] uses the AD method and computes the derivative for any model written in Python/Numpy. Another way to use the computational graph is to modify the computational graph to save the evaluation cost of the model. In Tensorflow [24], a graph optimization method is implemented to reduce the memory usage and reduce the evaluation cost for neural network models. It modifies the computational graph through constant folding, arithmetic simplification and optimization [25].

C. Motivation

The current framework for NIPC and SC methods includes three steps:

1) Generate sample points using the distribution of uncertain inputs.
2) Evaluate the model for all of the sample points.
3) Post-process the output points to compute the risk measure of the output.

Step one generates the sample points either randomly (for regression-based NIPC) or using a full/sparse grid sampling strategy (for integration-based NIPC and SC). In step two, the model is repeatedly evaluated at each of the input sample points. In step three, the model output samples are used to either estimate the PCE coefficients for NIPC, or form Lagrangian interpolation functions for SC, before computing the risk measure of the output. Step two always requires the most computation time for large-scale problems as it requires repeated evaluations on the model. If we view the model as a computational graph, in step two, each operation and function is evaluated for the same number of times as the number of input sample points. However this could create a lot of unnecessary evaluations when we want to

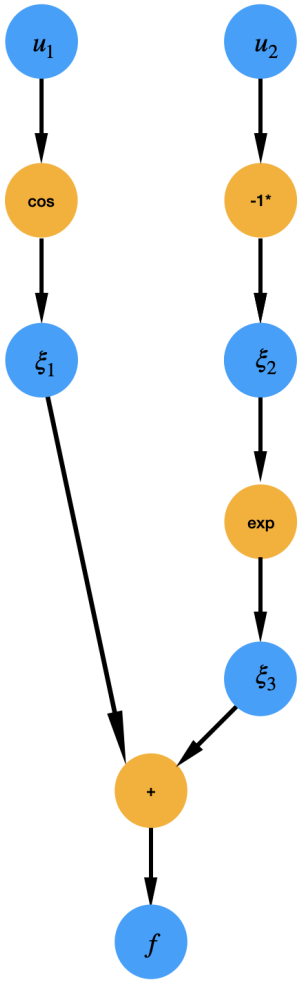


Fig. 1 Computational graph of function $f = cos(u_1) + exp(-u_2)$

evaluate the model at full/sparse grid quadrature points. For example, consider a UQ problem involving the simple function we showed before, given by

$$f = cos(u_1) + exp(-u_2). \tag{16}$$

In this problem, u_1 and u_2 are independent uncertain inputs, and we aim to compute the random variable f. The computational process that describes this function is in (15). If we choose the full-grid quadrature points with k quadrature points in each dimension, we evaluate the function at total of k^2 quadrature points which are

$$\Theta = \begin{Bmatrix} (u_1^1, u_2^1) & \dots & (u_1^k, u_2^1) \\ \vdots & \ddots & \vdots \\ (u_1^1, u_2^k) & \dots & (u_1^k, u_2^k) \end{Bmatrix}. \tag{17}$$

If we pass into a vector for input u_1 and u_2 to evaluate all of quadrature points at once, the input vectors are

$$
\begin{aligned}
u_1 &= \text{vec}\left([u_1^1, \dots u_1^k] \otimes [1, \dots, 1]\right) = \text{vec}\left(\begin{bmatrix} u_1^1 & \dots & u_1^k \\ \vdots & \ddots & \vdots \\ u_1^1 & \dots & u_1^k \end{bmatrix}\right), \\
u_2 &= \text{vec}\left([1, \dots, 1] \otimes [u_2^1, \dots u_2^k]\right) = \text{vec}\left(\begin{bmatrix} u_2^1 & \dots & u_2^1 \\ \vdots & \ddots & \vdots \\ u_2^k & \dots & u_2^k \end{bmatrix}\right).
\end{aligned} \tag{18}
$$

In the current NIPC framework, we evaluate $\xi_1 = cos(u_1)$ in (15) for k^2 points. However, ξ_1 is only a function of u_1. In the full-grid quadrature points, there are only k distinct values for u_1, meaning there are $k^2 - k$ redundant and wasteful evaluations in this case. Similarly for ξ_2 and ξ_3, there are only k distinct values to be evaluated for both of the operation nodes as they are related to only u_2. In contrast, for f, there are k^2 distinct values to be evaluated as f is affected by both ξ_1 and ξ_2.

In practical UQ problems, we could have a large model with a number of uncertain parameters, but the parameter uncertainties could each affect one part of model, making the most of the data nodes in the computational graph relatively sparse in terms of the uncertain inputs to which they are related. In this case, we could reduce the total computational cost of UQ by reducing the repeated evaluation cost for each node but the output data still carry the necessary results to compute the results of the output. The logic here is simple and straightforward, for a model that takes in the vectorized input, the engineers can manually modify the size of the input and state variables in the model to achieve this reduction if the model has simple graph structure. However, for a complicated model with large number of uncertainties, it becomes impractical to manually change the code to perform only necessary evaluations and match the shape of input for each node in the computational process.

III. Methodology

The goal of our method is to enable automatic generation of a modified computational graph, given a computer program, that eliminates all wasteful evaluations due to tensor-product sampling within an NIPC or SC algorithm. The modified computational graph leverages the graph structure of the numerical model and enables performing the minimal number of evaluations for each operation. To achieve this, we want to modify the size of input data that are passed into the operations.

In the computational graph of a model, each data node can be seen as a state variable in the model. In the stochastic case, these state variables become random variables, and we want to store the necessary information to describe them. In our method, we store in each data node, a tensor with the first dimension describing its evaluations on the quadrature points in its random space, and the rest of the dimensions describing the dimensions of the data in the deterministic case. For example, in the deterministic case, if we have $\xi_1 \in \mathbb{R}^{n \times n}$, the state variable ξ_1 is a matrix (deterministic rank of two) with the shape, $n \times n$. In the stochastic case, if ξ_1 is dependent on two uncertain inputs u_1 and u_2, it has the stochastic rank of two. The random space of ξ_1 can be described by the evaluations of ξ_1 on the two-dimensional full grid quadrature points from u_1 and u_2. Thus, the computational graph stores ξ_1 as a tensor (rank three), with the shape $(k_{u_1} k_{u_2}, n, n)$. To decide the size of a data node, we first perform a topological sort on the computational graph using the Kahn's algorithm [26] to generate the correct order for the operations to be evaluated without unnecessary feedback loops. Then we iterate through the operations in order and find the ancestor uncertain inputs that have a path to the node. The ancestor information is used to decide on the size of the tensor to store for each data node.

Since we store only the necessary information for the data node, we need to ensure we get the right size for the output data when passing the input data into the operation node. We have two types of operation nodes based on the number of inputs: univariate operations and multivariate operations. For the univariate operations (e.g. power functions, trigonometric functions), the input data and the output data are always related to the same uncertain inputs, which means we can pass the input data into the operation and get the output with the correct size for the random dimension. The multivariate operations include the bivariate arithmetic operations (e.g. addition, multiplication) and implicit functions. For multivariate operations, the output data will be dependent on all of the uncertain inputs the inputs are dependent on, and we may need to modify the shape of the input data to ensure we get the output data with the correct size. Fortunately,

we can use tensor-algebra operations to modify the size of the data. For example, suppose we have an addition operation:

$$\xi_3 = \xi_1 + \xi_2$$

with $\xi_1(u_1) \in \mathbb{R}^{k_{u_1} \times n}$ and $\xi_2(u_2) \in \mathbb{R}^{k_{u_2} \times n}$. The output data ξ_3 is related to both u_1 and u_2 and should have the size of $\mathbb{R}^{k_{u_1} k_{u_2} \times n}$. In this case, we need to transform the input into the same size as the output. For ξ_1, in its random dimension, it only k_{u_1} points represent its evaluations on the quadrature points in u_1 dimension. Although ξ_1 is not related to u_2, we still need to include the information on the u_2 dimension to match the size of the output. Similarly, we need to include the information on u_1 dimension for ξ_2. In this case, we can use NumPy's Einstein summation function followed by a reshape function. The specific code in Python would be

```
import numpy as np
original_shape_xi_1 = (k_u1,n1,n2) #Original size for xi_1
original_shape_xi_2 = (k_u2,n1,n2) #Original size for xi_2
modified_shape = (k_u1*k_u2, n1, n2) #Target size for xi_1 and xi_2
xi_1 = np.einsum('i...,p...->pi...', xi_1, np.ones((k_u2, 1)) #(k_u1,n1,n2) -> (k_u1,k_u2,n1,n2)
xi_2 = np.einsum('i...,p...->ip...', xi_2, np.ones((k_u1, 1)) #(k_u2,n1,n2) -> (k_u1,k_u2,n1,n2)
xi_1 = np.reshape(xi_1, modified_shape) #(k_u1,k_u2,n1,n2) ->(k_u1*k_u2, n1, n2)
xi_2 = np.reshape(xi_2, modified_shape) #(k_u1,k_u2,n1,n2) ->(k_u1*k_u2, n1, n2).
```

The code above transforms both inputs into the same size of the output with the correct order in the random dimension. In fact, for any dimension and size of the input, we can always use the Einstein summation functions to transform the input data to the correct size we need. In our method, we insert the *einsum* operation node with the correct arguments whenever the size of the input and output data do not match. The outline for our graph modifying algorithm is shown in Alg. 1. To demonstrate how our algorithm modifies the graph and reduces the evaluation costs, we show a comparison of computational graphs on the function (16) in Fig. 5. In Fig. 5, we show the computational graph for the current UQ method when we evaluate the model at full-grid quadrature points of the uncertain inputs, and the computational graph after we apply the graph modifying method. The size of the data flow is also labelled in the graphs. In this case, with the graph modification method, we pass u_1 and u_2 with only the distinct values, and thus reduce the repeated evaluations on computing ξ_1, ξ_2 and ξ_3 and two *einsum* nodes are added to make sure we compute f at the correct size.

Algorithm 1 Graph modification algorithm for full-grid quadrature points evaluations

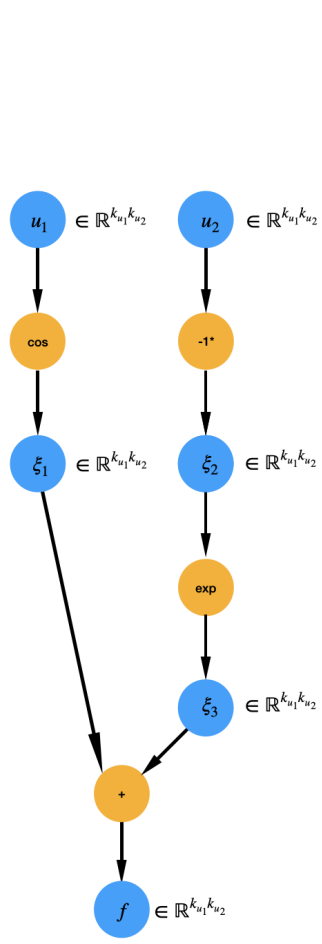
```
Specify a computational graph for a numerical model where operations can be vectorized
Run Kahn's algorithm to determine the correct order for the operations to be evaluated.
Visit each operation node in order and record the ancestor random inputs that have a path to it.
for each operation node do
    for each input data node do
        if the ancestor information of the input data node does not match the output data node then
            Insert an einsum operation node and modify the shape of the input node
        end if
    end for
end for
```

A. Implementation

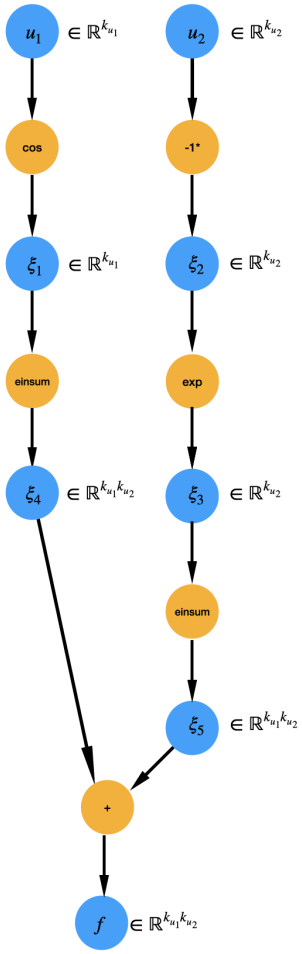
The graph modification algorithm can only be implemented from the data access layer of a software package where we have access to the computational graph and data structure of a computer program. This is possible thanks to the Computational System Design Language (CSDL) package*. CSDL is an embedded domain-specific language targeting large-scale multidisciplinary design analysis and optimization problems. With CSDL, the user defines the model as a sequence of operations by using a software interface that is highly expressive and natural (difficult to differentiate from regular Python).

The CSDL model development involves three layers, front-end, middle-end and back-end. Using the front-end, the user defines the model and specifies the problem that needs to be solved. Next, the middle-end constructs a

*CSDL software repository: `https://lsdolab.github.io/csdl/`



(a) Computational graph for the current method



(b) Computational graph after applying the graph modification method

Fig. 2 Computational graphs with data size for full-grid quadrature points evaluation on $f = cos(u_1) + exp(-u_2)$

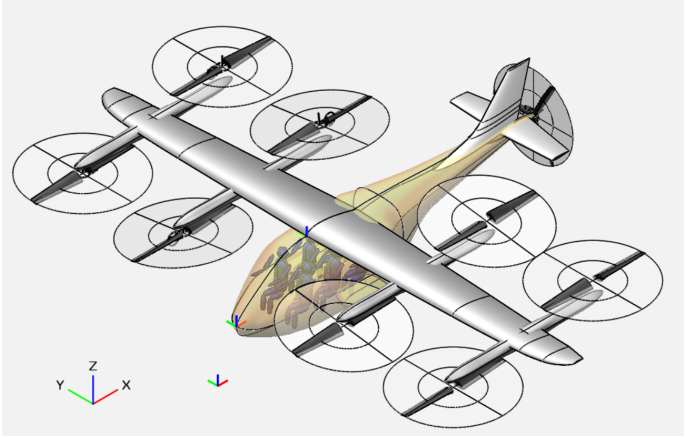


Fig. 3 Figure, courtesy of NASA. Representation of the eVTOL aircraft[27]

computational graph, which can be operated on and transformed, as our method does here. Finally, using an *automatic code generation* approach, an executable back-end is created.

Below is a coarse outline of how CSDL and our proposed method interact:

1) User defines the model and identifies the uncertain inputs.
2) The algorithm generates the quadrature points according to the distributions of the input variables.
3) The algorithm generates the computational graph from the numerical model the user defines.
4) The algorithm transforms the computational graph as in Alg. 1.
5) An executable back-end is generated.
6) Post-processing is done to calculate the risk measure of the output.

IV. Numerical Results

To test our uncertainty propagation method, we use a multi-point mission analysis problem involving an electric vertical takeoff and landing (eVTOL) aircraft. The problem is adapted from [27]. We aim to predict the total energy consumption of a lift-plus-cruise eVTOL aircraft concept (Fig.3) in a two-segment mission including climb and cruise. The mission has a total range of $R = 37.5$ nmi and includes a climb segment in which the aircraft climbs from 6,000 ft altitude to 10,000 ft altitude with a flight path angle of $\gamma = 20$ deg, and a cruise segment at 10,000 ft with $\gamma = 0$ deg until the end of the mission. The climb speed and cruise speed stay constant through the stages. The UQ problem is to find the mean and standard deviations of the total energy consumption in this mission where the climb velocity and cruise velocity are random variables with $V_{climb} \sim N(0.3, 0.03)$ Mach and $V_{cruise} \sim N(0.5, 0.05)$ Mach. In the model, we use the vortex-lattice method (VLM), a low-fidelity incompressible and inviscid aerodynamic analysis method, to compute the lift and induced drag on the wing. The description of the VLM model can be found in [28]. At each stage, we apply a small angle approximation for the angle of attack and perform a single point analysis halfway through the stage to compute the lift and drag forces. Then the energy consumption at each stage is calculated by:

$$E = \frac{(sin(\gamma)W + D)\,R}{\eta}, \tag{19}$$

which is derived from a simple force balance. The properties for the wing are presented in Table 2 and properties for the climb and cruise segment are presented in Table 3.

The UQ problem is solved using the following four methods: integration-based NIPC method using the full-grid quadrature points (full-grid NIPC), full-grid NIPC with the graph modification method (graph-modified full-grid NIPC); integration-based NIPC method using the Smolyak sparse-grid quadrature points (sparse-grid NIPC) and Monte Carlo method (MC). In our numerical experiments, the full-grid NIPC and the graph-modified full-grid NIPC always generate the same output results. Figure 4 shows the computing time in terms of the number of equivalent model evaluations for the full-grid NIPC and graph-modified full-grid NIPC. We observe a significant reduction in computation time when applying the graph modification method, and the reduction becomes more significant when we increase the number of quadrature points, as expected. In Figure 5, the convergence plots of these four methods are compared for

Table 2 Properties of the wing

Airfoil	NASA/LANGLEY LS(1)-0417 (GA(W)-1) AIRFOIL
Span (ft)	50.53
Chord (ft)	3.38
Area (ft^2)	210.27
Aspect Ratio	12.13

Table 3 Properties for climb and cruise segments of the flight mission

	Climb segment	Cruise segment
Initial altitude (ft)	6,000	10,000
Final altitude (ft)	10,000	10,000
Flight path angle γ (deg)	20	0
Range R (nmi)	R_{climb}	$37.5 - R_{climb}$
Speed V (Mach)	$N(0.3, 0.03)$	$N(0.5, 0.05)$
Aircraft weight W (lbs)	8,000	8,000
Combined efficiency of the motor and powertrain η	0.85	0.85

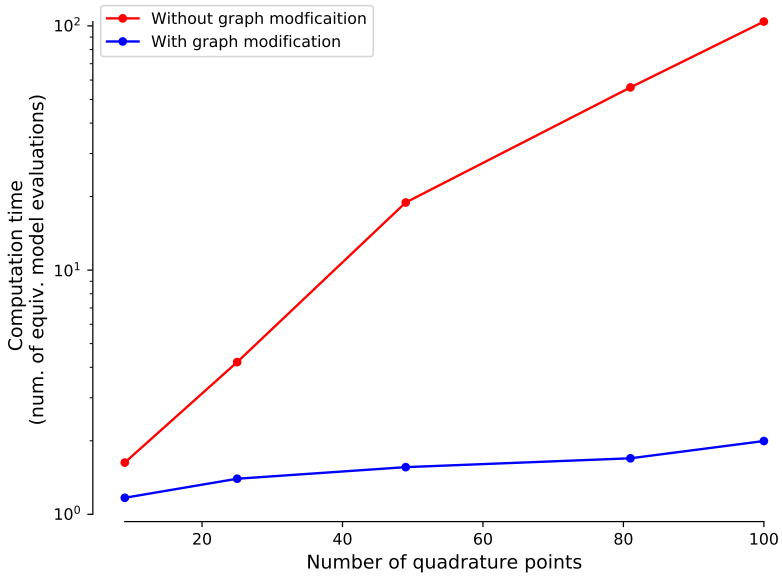
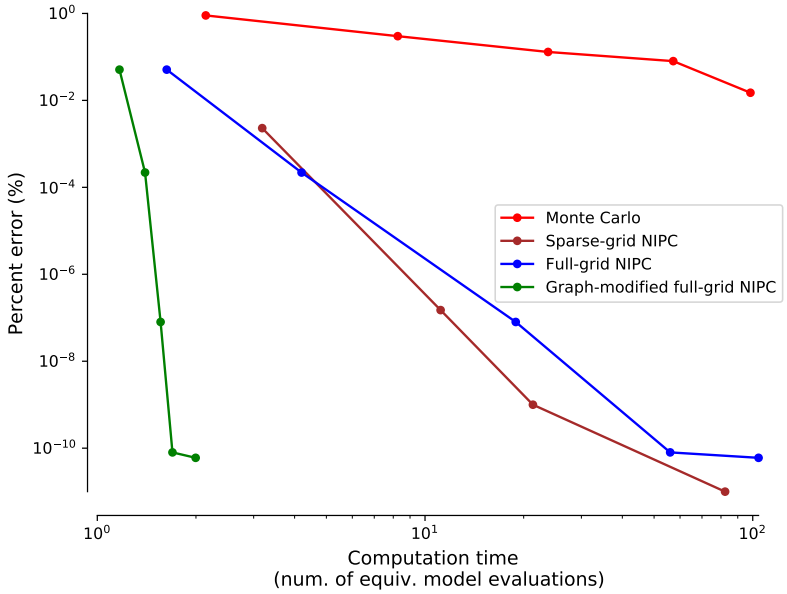
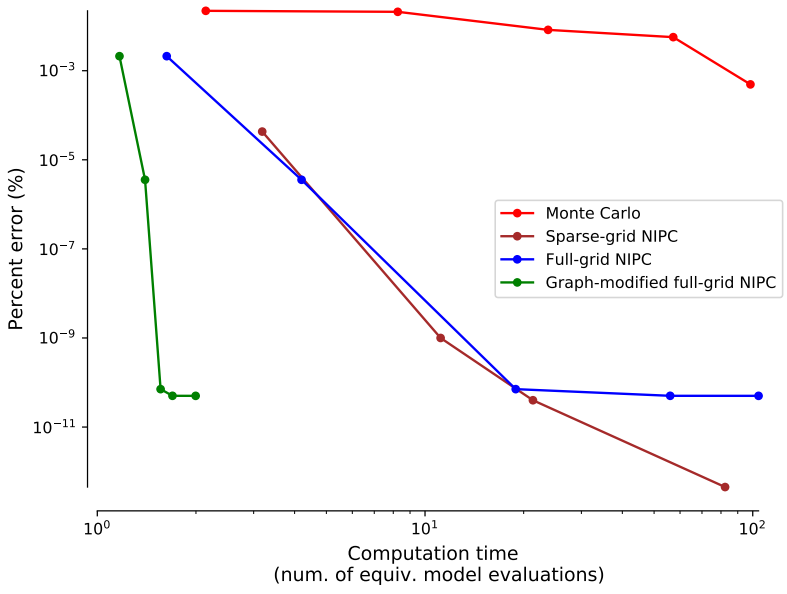


Fig. 4 Comparison of the full-grid NIPC methods with and without applying the graph modification method

the expectation and standard deviation of the total energy, namely μ_E and σ_E, respectively. The percentage errors are calculated with respect to the results of the full-grid NIPC using 400 quadrature points. From the convergence plots, we observe similar trends for the mean and standard deviation of the output in this UQ problem. The Monte Carlo has the slowest convergence rate among other methods. This is because we only have two uncertain inputs in this UQ problem, and the Monte Carlo method only outperforms other methods when the random space is significantly high-dimensional. PCE-related methods are often the most efficient choices for small-dimensional UQ problems like this. Sparse-grid



(a) μ_E



(b) σ_E

Fig. 5 Convergence of the UQ results for the four methods

NIPC and full-grid NIPC have similar performance, with sparse-grid NIPC converging better when the percent error is below 10^{-10}. However, we see a dramatic speed up after applying the graph modification method to the full-grid NIPC; the total computing time is reduced by an order of magnitude, making the graph-modified full-grid NIPC the most efficient method among other methods.

It is not surprising that we see a major speed-up after we apply the graph modification method to the full-grid NIPC method in this problem. This test problem is a multi-point problem, with two random variables each affecting the aerodynamic analysis at one point, and the output of the model is based on the results from both aerodynamic analyses. The model has a two-dimensional random space, but for each aerodynamic analysis at a point, only one random variable is affecting it. In this case, after we apply the graph modification method, each aerodynamic analysis is performed on the one-dimensional quadrature points of the input random variable it is dependent on, but we still get the evaluation results of the final output on the two-dimensional quadrature points. The reduction of input points reduces the evaluation time on each aerodynamic analysis, and this causes a significant speed-up for the full-grid NIPC method.

The results are so favorable for our method because of the particular structure of the UQ problem we test it on, and we do not expect such favorable results on all UQ problems. In fact, the performance of the graph modification method is case-dependent. In the worst-case scenario, if all the data nodes are dependent on all of the uncertain inputs, there are no repeated evaluations to reduce. Applying the graph modification problem to this problem will not make a difference. However, it is common that most of the variables in a practical engineering model are not affected by all of the input variables (i.e. in the multi-point problems). In fact, practical engineering problems can involve many more than two mission segments or operating conditions, which is a reason to expect larger improvements in those cases with, say, 10 or more random inputs that each affect only one or a subset of operating conditions. In these cases, applying our method could save significant computing time without requiring manual effort, and in some problems, achieve more savings than would be possible purely manually because the algorithm has access to the entire computational graph.

V. Conclusion

In this paper, we propose a new method that can significantly evaluation cost for integration-based non-intrusive polynomial chaos and stochastic collocation methods. This method achieves that by using the computational graph of the model and algorithmically remove the repeated evaluations on each data node so that it generates the model evaluations on the full-grid quadrature points in an efficient way. This method is implemented in the Computational System Design Language, so that the modeling language and its back-end automates the process of detecting and capitalizing the model structure, achieving the UQ cost reduction in a general way.

We use a multi-point mission analysis problem for an eVTOL aircraft to test on our UQ method. For this UQ problem, we observe that applying the graph modification method reduces the computing time of the full-grid integration-based NIPC method by an order of magnitude and makes it the best method among Monte Carlo and sparse-grid NIPC methods. The results look better than we expect for the average problems, but the results have the potential to look even better in problems with more uncertain inputs. The integration-based NIPC and SC suffer the curse of dimensionality because of the exponential growth of the computation time caused by the exponential growth of the quadrature points. However, our methods breaks this relationship on certain problems, and the computation time does grow at the same rate as the number of quadrature points. For example, for a multi-point problems with a large number of points, e.g. ten uncertain velocities for ten different mission segments, the cost of our method would be roughly the same to solving this problem with only one uncertain input using the current NIPC and SC methods.

A limitation of our method is that its performance is case-dependent. Significant effectiveness is only expected when most operations are not dependent on all of the uncertain inputs. However, for a wide range of UQ problems, we expect this method to significantly improve the scalability of the NIPC and SC methods.

For future work,the graph modification method can be applied to other problems that involve multi-dimensional integration. Also, the idea here can be used to reduce the derivative computation cost for optimization under uncertainty problems.

VI. Acknowledgments

The material presented in this paper is, in part, based upon work supported by NASA under award No. 80NSSC21M0070.

References

[1] Mitra, E. D., and Hlavacek, W. S., "Parameter estimation and uncertainty quantification for systems biology models," *Current opinion in systems biology*, Vol. 18, 2019, pp. 9–18.

[2] Najm, H. N., "Uncertainty quantification and polynomial chaos techniques in computational fluid dynamics," *Annual review of fluid mechanics*, Vol. 41, 2009, pp. 35–52.

[3] Lall, U., Josset, L., and Russo, T., "A snapshot of the world's groundwater challenges," *Annual Review of Environment and Resources*, Vol. 45, 2020, pp. 171–194.

[4] Zhang, J., Man, J., Lin, G., Wu, L., and Zeng, L., "Inverse modeling of hydrologic systems with adaptive multifidelity Markov chain Monte Carlo simulations," *Water Resources Research*, Vol. 54, No. 7, 2018, pp. 4867–4886.

[5] Le Maître, O., and Knio, O. M., *Spectral methods for uncertainty quantification: with applications to computational fluid dynamics*, Springer Science & Business Media, 2010.

[6] Montomoli, F., Carnevale, M., D'Ammaro, A., Massini, M., and Salvadori, S., *Uncertainty quantification in computational fluid dynamics and aircraft engines*, Springer, 2015.

[7] Beran, P., Stanford, B., and Schrock, C., "Uncertainty quantification in aeroelasticity," *Annual review of fluid mechanics*, Vol. 49, 2017, pp. 361–386.

[8] Tu, J., Choi, K. K., and Park, Y. H., "A new study on reliability-based design optimization," 1999.

[9] Fragkos, K., Papoutsis-Kiachagias, E., and Giannakoglou, K., "pFOSM: An efficient algorithm for aerodynamic robust design based on continuous adjoint and matrix-vector products," *Computers & Fluids*, Vol. 181, 2019, pp. 57–66.

[10] Ng, L. W.-T., "Multidelity approaches for design under uncertainty," Ph.D. thesis, Massachusetts Institute of Technology, 2013.

[11] Ng, L. W., and Willcox, K. E., "Monte Carlo information-reuse approach to aircraft conceptual design optimization under uncertainty," *Journal of Aircraft*, Vol. 53, No. 2, 2016, pp. 427–438.

[12] Peherstorfer, B., Willcox, K., and Gunzburger, M., "Optimal model management for multifidelity Monte Carlo estimation," *SIAM Journal on Scientific Computing*, Vol. 38, No. 5, 2016, pp. A3163–A3194.

[13] Cressie, N., "The origins of kriging," *Mathematical geology*, Vol. 22, No. 3, 1990, pp. 239–252.

[14] Fuhg, J. N., Fau, A., and Nackenhorst, U., "State-of-the-art and comparative review of adaptive sampling methods for kriging," *Archives of Computational Methods in Engineering*, Vol. 28, No. 4, 2021, pp. 2689–2747.

[15] Xiu, D., and Hesthaven, J. S., "High-order collocation methods for differential equations with random inputs," *SIAM Journal on Scientific Computing*, Vol. 27, No. 3, 2005, pp. 1118–1139.

[16] Babuška, I., Nobile, F., and Tempone, R., "A stochastic collocation method for elliptic partial differential equations with random input data," *SIAM Journal on Numerical Analysis*, Vol. 45, No. 3, 2007, pp. 1005–1034.

[17] Xiu, D., and Karniadakis, G. E., "The Wiener–Askey polynomial chaos for stochastic differential equations," *SIAM journal on scientific computing*, Vol. 24, No. 2, 2002, pp. 619–644.

[18] Wiener, N., "The homogeneous chaos," *American Journal of Mathematics*, Vol. 60, No. 4, 1938, pp. 897–936.

[19] Smolyak, S. A., "Quadrature and interpolation formulas for tensor products of certain classes of functions," *Doklady Akademii Nauk*, Vol. 148, Russian Academy of Sciences, 1963, pp. 1042–1045.

[20] Luthen, N., Marelli, S., and Sudret, B., "Sparse polynomial chaos expansions: Literature survey and benchmark," *SIAM/ASA Journal on Uncertainty Quantification*, Vol. 9, No. 2, 2021, pp. 593–649.

[21] Eldred, M., and Burkardt, J., "Comparison of non-intrusive polynomial chaos and stochastic collocation methods for uncertainty quantification," *47th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, 2009, p. 976.

[22] Griewank, A., et al., "On automatic differentiation," *Mathematical Programming: recent developments and applications*, Vol. 6, No. 6, 1989, pp. 83–107.

[23] Maclaurin, D., "Modeling, inference and optimization with composable differentiable procedures," Ph.D. thesis, 2016.

[24] Abadi, M., "TensorFlow: learning functions at scale," *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016, pp. 1–1.

[25] Larsen, R. M., and Shpeisman, T., "Tensorflow graph optimizations," 2019.

[26] Kahn, A. B., "Topological sorting of large networks," *Communications of the ACM*, Vol. 5, No. 11, 1962, pp. 558–562.

[27] Silva, C., Johnson, W. R., Solis, E., Patterson, M. D., and Antcliff, K. R., "VTOL urban air mobility concept vehicles for technology development," *2018 Aviation Technology, Integration, and Operations Conference*, 2018, p. 3847.

[28] Jasa, J. P., Hwang, J. T., and Martins, J. R., "Open-source coupled aerostructural optimization using Python," *Structural and Multidisciplinary Optimization*, Vol. 57, No. 4, 2018, pp. 1815–1827.