Estimating parameters and discrepancy of computer models with graphs and neural networks

Yigit A. Yucesan*, Andre Von Zuben†, Felipe A. C. Viana‡
University of Central Florida, Orlando, FL, 32816, USA

Jarir Mahfoud§
Université de Lyon, INSA-Lyon, CNRS UMR5259, F-69621, France

Computer models are typically used to describe the input-output relationships in physical systems. These models require a set of known inputs besides parameters that need to be specified. Often, these unknown parameters are calibrated using a limited amount of field data. The seminal paper by Kennedy and O’Hagan [1] introduced a statistical approach based on the Gaussian process model that allows for quantification of uncertainties due to: a) model parameters (calibration), b) limited number of simulations, c) discrepancy between the simulation code and the actual physical system, and d) observation process. The approach is very robust and has been widely applied to cases where physical systems are described by computationally expensive simulations. In this paper, we propose a framework for bias estimation and calibration of computer models using graph theory and neural networks. Similarly to the Kennedy and O’Hagan approach, we use observed data to simultaneously calibrate model parameters and estimate the discrepancy term through a metamodel. While Kennedy and O’Hagan based their approach on the Gaussian process model, we choose neural networks. Another difference is that we represent the hybrid model that merges the original computer model with the neural network through a graph. This allows us to easily estimate the discrepancy term even for hidden nodes of the graph. We studied the performance of our framework with the aid of numerical experiments and state-of-the-art machine learning software packages. The preliminary results are promising and demonstrate the ability to perform simultaneous estimation of discrepancy and model parameters at reasonable computational cost.

I. Nomenclature

\[ y(x) \] = output of the physical system  
\[ x \] = controllable inputs  
\[ w \text{ and } b \] = adjustable parameters of the discrepancy corrector model  
\[ \eta(x, \theta) \] = computer simulation model  
\[ \delta(x, w, b) \] = discrepancy corrector model  
\[ \theta \] = adjustable parameters of the computer simulation model

II. Introduction

The seminal work by Kennedy and O’Hagan [1] introduced a statistical approach for Bayesian calibration of computer simulations based on the Gaussian process model. They considered prediction and uncertainty analysis for systems described by “complex mathematical models” coded in the form of simulations. Observed data is used to learn unknown model parameters (fitting model to data, a.k.a. calibration). The main innovations of the Kennedy and O’Hagan approach are the ability to simultaneously a) handle multiple sources of uncertainty (model parameter, limited number of simulations, and observation process), and b) correct for discrepancy between observed data and model

---

*Graduate Research Assistant, Department of Mechanical and Aerospace Engineering, yucesan@knights.ucf.edu.
†Graduate Research Assistant, Department of Mechanical and Aerospace Engineering, avzuben@knights.ucf.edu.
‡Assistant Professor, Department of Mechanical and Aerospace Engineering, viana@ucf.edu, Sr. AIAA Member.
§Maître de Conférences, Dynamics and Control of Structures, jarir.mahfoud@insa-lyon.fr.
predictions. The basic formulation assumes that the physical system can be observed at various inputs settings:

\[ y(x_i) = \zeta(x_i) + \epsilon_i, \]

where \( x_i \) are the input settings; \( y \) are the output observations; \( \zeta(x_i) \) is the actual system output (not observable); and \( \epsilon_i \) represents the variability in observation. The physical system is modeled through computer simulations such that:

\[ y(x_i) = \eta(x_i; \Theta) + \delta(x_i; \mu, \beta, \lambda) + \epsilon_i, \]

where \( \eta(x_i; \Theta) \) is the simulator; \( \Theta \) is the set of calibration parameters; \( \delta(x_i; \mu, \beta, \lambda) \) is the Gaussian process model that describes the discrepancy; and \( \mu, \beta, \) and \( \lambda \) are the hyper-parameters of the discrepancy model. The approach is Bayesian and allows for incorporation of prior information about parameters for both the simulator and the Gaussian process models. In other words, all parameters in Eq. (2) are estimated by solving the posterior distribution given observed data

\[ p(\Theta, \mu, \beta, \lambda | y) \propto L(y|\Theta, \mu, \beta, \lambda, \Sigma_y) p_0(\Theta) p_0(\mu) p_0(\beta) p_0(\lambda). \]

Further details about the likelihood formulation and how the approach can be extended to cases in which simulations are computationally expensive to obtain can be found in [1,2]. For now, it is enough to say that the complexity of Eq. (3) is such that estimation is carried out using a Markov chain Monte Carlo method [3,4].

Since the Kennedy and O’Hagan was first introduced, the approach was further developed and also found many applications. For example, Higdon et al. [2] revisited the approach describing it in the context of engineering applications. They applied it a charged particle accelerator and a spot welding process. Higdon et al. [5] extended the basic framework to be used in cases which the field data and the simulator output are highly multivariate (e.g., the output may be an image or may describe the shape of a physical object). The methodology was tested on historical implosion experiments. Arendt et al. [6] discussed an important problem one might encounter when using the Bayesian calibration framework. The issue of distinguishing between the effects of calibration parameters versus model discrepancy. They illustrated this identifiability problem with several examples and explained the mechanisms behind it. Kumar et al. [7] employed the framework for calibration when there are multiple time-dependent (transient) outputs. They highlight their formulation with a turbomachinery compressor dynamic response application. Viana et al. [8] used the framework for composite load (in electrical power systems) model selection and calibration. The composite load model is first selected from a library of models, then it is calibrated to better match the field data. Mullins and Mahadevan [9] showed how the approach can be expanded to a comprehensive tool for approach to prediction under uncertainty. They applied it to the Sandia National Laboratories verification and validation challenge [10], in which legacy data and experimental measurements of different levels of fidelity and complexity are used simultaneously.

As robust and widely used as the Kennedy and O’Hagan approach is, it hinges on the Gaussian process model. With that, the framework inherits limitations such as difficulty to handle large number of data points (due to repetitive inversion of large covariance matrices) and consequent scalability to applications with large number of input variables. At least in its basic form, as the one illustrated by Eqs. (1) to (3) the calibration and discrepancy estimation is limited to simulators of directly observable outputs. We propose a strategy for estimation of discrepancy and calibration of computer models based on the graph models and neural networks [11]. The overall system is represented by a directed graph. The nodes can represent as many sub-models as needed for the system-wide simulation. The edges coming into the nodes are inputs and the edges coming out of nodes are outputs. In the graph, outputs do not need to be observed and can also act as inputs for subsequent nodes. Within this framework, the discrepancy models are simply a collection of nodes in the graph. Given its flexibility and convenient numerical properties, we use neural networks to act as discrepancy models. Graph models have recently been utilized to construct physics-informed machine learning models, where both physics and neural networks layers are modeled as nodes [12–14]. We do not aim to outperform the Kennedy and O’Hagan approach; instead, we expect to offer a straightforward alternative for the cases in which there is benefit in modeling the system through a graph.

The remaining of the paper is organized as follows. Section III details our proposed formulation for estimation of discrepancy and calibration of computer models. Section IV presents the numerical experiments and discusses the results obtained in our study. Finally, section V closes the paper recapitulating salient points and presenting concluding remarks and future work.
### III. Proposed Framework for Estimation of Discrepancy and Model Parameters

**A. Implementation through Graph Models**

Before we present our framework, we use Figure 1 to introduce the notation and elements used in graph representation of the model. The graph denotes the operators on top of tensors that represent the inputs, outputs, and model parameters. The basic tensor operators include tensor transfer (which takes the tensor from one node of the graph to another), concatenation, copy, as well as algebraic operators, such as sum and multiplication. We use boxes to denote the models (nodes in the graph), in which case we use \( \eta(x, \theta) \) and \( \delta(x, w, b) \) to represent the computer model and the discrepancy corrector, respectively. In this paper, we propose using neural networks as a surrogate model.

![Graph Model Elements](image)

---

**Fig. 1 Elements used in the graph representation of the hybrid model. To help with visualization, we use green for the computer model and blue for the discrepancy corrector.**

We use the flexibility of graphs to define how the different models elements interact with one another. This way, Fig. 2a is the graph model representing

\[
y(x) = \zeta(x) = \eta(x, \theta) + \delta(x; w, b),
\]

where \( \zeta(x) \) is fully observable, as there is no sample-to-sample variability; \( \eta(x, \theta) \) is the simulator; and \( \delta(x; w, b) \) is the neural network functions as a discrepancy corrector; and \( w \) and \( b \) are the weights and biases of the neural network.

When making predictions using the Fig. 2a graph, one has to set values for \( \theta, w, b \). An intuitive choice for the loss function is the mean squared error

\[
\Lambda(y_{OBS}, y) = \frac{e^T e}{n_{OBS}}, \quad e = y_{OBS} - y
\]

where \( n_{OBS} \) is the number of observations and \( y = \eta(x; \theta) + \delta(x; w, b) \). As a consequence, minimizing \( \Lambda(y_{OBS}, y) \) implies in simultaneous model calibration by adjusting \( \theta \) and discrepancy estimation by adjusting \( w \) and \( b \).

Equation 4 and Fig. 2a represent a rather monolithic model. The simulator \( \eta(x; \theta) \) is computed at once and then corrected with \( \delta(x; w, b) \). There are no intermediate steps with auxiliary models being called before the final output is available. This is not always the case when modeling complex engineering systems. For example, consider a hypothetical system described by the following equations:

\[
\begin{align*}
\eta_4(x, \eta_1, \eta_2, \eta_3; \theta_4), \\
\eta_2 &= \eta_2(\eta_1; \theta_2), \\
\eta_3 &= \eta_3(x, \eta_1, \eta_2; \theta_3), \quad \text{and} \\
\eta_1 &= \eta_1(x; \theta_1).
\end{align*}
\]

Additionally, the discrepancy correction do not necessarily have to be applied at the predicted \( \zeta(x) \). For example, through engineering analysis, it might be decided that only \( \eta_2(\eta_1; \theta_2) \) and \( \eta_3(x, \eta_1, \eta_2; \theta_3) \) need to be corrected. Therefore

\[
\begin{align*}
\eta_2^*(\eta_1; \theta_2) &= \eta_2(\eta_1; \theta_2) + \delta_1(x; w_1, b_1) \quad \text{and} \\
\eta_3^*(x, \eta_1; \theta_3) &= \eta_3(x, \eta_1, \eta_2^*; \theta_3) + \delta_2(x; w_2, b_2).
\end{align*}
\]
Fig. 2 Graph representation of Eq. 4. In prediction, both the simulator and the discrepancy corrector are used with parameters defined in training. In training, a loss function is used to help matching $\eta(X_{OBS}, \theta) + \delta(X_{OBS}, w, b)$ to observed outputs $y_{OBS}$ by adjusting parameters $\theta, w$ and $b$.

Fig. 3a illustrates the graph representation of such system. Similarly to the previous example, the model parameters in the graph model are adjusted using observed data. However, some of the computer model nodes might not need to have their parameters calibrated. Figure 3b shows the graph that returns a loss function $\Lambda(y_{OBS}, y)$ to be optimized by adjusting the trainable parameters amongst $\theta_2, w_{1,2}$ and $b_{1,2}$. Equation 5 can be used here as well.

Besides complexity, the graphs of Figs. 2a and 3a differ in one important feature. In Fig. 2a, even though $X(x; w, b)$ is not observed, it feeds into Eq. 5 directly through $y = \eta(x; \theta) + \delta(x, w, b)$. This is not the case of Fig. 3a, $\delta_1(x; w_1, b_1)$ and $\delta_2(x; w_2, b_2)$ are neither observed nor feed into Eq. 5 directly. As a matter of fact, they are both used to compensate for missing physics in the outputs of hidden nodes in the graph.

Fig. 3 Graph representation of Eq. 6. Set of simulator and discrepancy corrector models do not need to share the same input variables.

While graphs offer flexibility to modeling, it is not clear whether parameter estimation through minimization of the loss function is computationally feasible. Any realistic implementation of the graphs such as those shown in Figs. 2a and 3a would end up in a large number of parameters (mainly because $w$ and $b$ can easily reach the hundreds to several...
Therefore, the loss function minimization is likely to be performed by gradient-based methods. Here we believe there are two main aspects contributing to the success of our proposed approach.

First, we propose using neural networks as discrepancy corrector models (i.e., the set of $\delta(x; w, b)$). On top of being flexible as a surrogate model, neural network training does not require tedious operations such as inversion of large matrices, computing distance between points, etc. Implementations of neural networks rely on backpropagation for hyper-parameter optimization. As summarized in Fig. 4, training data is fed forward, generating the corresponding outputs (Fig. 4a), prediction error, and finally the loss function. Then, the loss function adjoint is propagated backward (through the chain rule) giving the gradient with respect to the weights. For the example, in Fig. 4b that means

$$
\frac{\partial \Lambda}{\partial w_1} = \frac{\partial \Lambda}{\partial \delta} \frac{\partial \delta}{\partial u_1} \frac{\partial u_1}{\partial w_1}, \quad \frac{\partial \Lambda}{\partial w_2} = \frac{\partial \Lambda}{\partial \delta} \frac{\partial \delta}{\partial u_1} \frac{\partial u_1}{\partial w_2}, \\
\frac{\partial \Lambda}{\partial w_3} = \frac{\partial \Lambda}{\partial \delta} \frac{\partial \delta}{\partial u_2} \frac{\partial u_2}{\partial w_3}, \quad \frac{\partial \Lambda}{\partial w_4} = \frac{\partial \Lambda}{\partial \delta} \frac{\partial \delta}{\partial u_2} \frac{\partial u_2}{\partial w_4}, \\
\frac{\partial \Lambda}{\partial w_5} = \frac{\partial \Lambda}{\partial \delta} \frac{\partial \delta}{\partial w_5}, \quad \text{and} \\
\frac{\partial \Lambda}{\partial w_6} = \frac{\partial \Lambda}{\partial \delta} \frac{\partial \delta}{\partial w_6}. 
$$

(8)

Figure 4 is the graphical representation of the multi-layer perceptron. Even though the figure only shows the weight hyper-parameters ($w$), the formulation can be extended for the case where each perceptron also has a bias term ($b$).

Fig. 4 Backpropagation overview. In prediction, inputs are fed forward, generating the activations of hidden layers $u_i$ and output layer $\delta$. In training, partial derivatives are fed backward, generating the gradient of the loss function with respect to the weights, $\nabla \Lambda = \left[ \frac{\partial \Lambda}{\partial w_1} \cdots \frac{\partial \Lambda}{\partial w_6} \right]^T$.

The second aspect contributing to the success of our proposed approach is the focus on computer models of reduced order (i.e., the set of $\eta(x; \theta)$). More specifically, we assume these computer models (a) are cheap to evaluate, and (b) have their gradient information with respect to model parameters available. Computationally efficient kernels are important as these hybrid models are called many times both during training and in prediction (in applications including uncertainty quantification, design optimization, inverse problems and model identification, etc.). We recognize not all computer models are cheap to evaluate, even though some could benefit from projection-based techniques to improve computational speed [15-17]. With regards to model-parameter gradient computation, we also recognize that not all computer models would lead to easy-to-get gradients. For the cases in which the gradients can not be implemented directly, we (a) disapprove the implementation through finite differences and alike, as these tend to have undesirable errors and do not scale well with dimensionality, and (b) recommend implementation considering automatic differentiation (a.k.a. algorithmic differentiation, dynamic computational graphs or differentiable programming). For an accessible review of what automatic differentiation is and what it is not, the interested reader is referred to [18].

B. Limitations of Proposed Framework

The main concern in the proposed framework is the implementation of the computer models (i.e., the set of $\eta(x; \theta)$). As we mentioned, the typical usage and training of the proposed hybrid models requires a large number of function evaluations. Therefore, the nodes representing the computer model (a) have to be cheap to evaluate, and (b) have their gradient information with respect to model parameters available.

We can see how these two limitations can reduce the practical application of our proposed approach. However, as we already mentioned, we believe there are ways to circumvent these limitations. For example, one could either reduce
the fidelity of models to the point that computational speed is not an issue anymore, or use projection-based techniques to improve computational speed [15, 17]. With regards to the gradients with respect to model parameters, depending on the computational tools chosen for implementation of our framework, one can also benefit from native automatic differentiation capabilities. As a matter of fact, automatic differentiation is available in most modern machine learning frameworks, such as TensorFlow [19, 20] (www.tensorflow.org) and PyTorch [21] (pytorch.org).

### IV. Numerical Experiments and Discussion

In this section, we present the results of two analytical examples with different types of input-output relationships. The first example illustrates the scenario where the output of the neural networks is fully observable. The second example presents a case where the output which will be corrected by the neural networks is hidden (not observed directly). Table 1 provides the configurations of optimization parameters for both examples. The loss function used in all cases is the mean squared error (see Eq. 5).

**Table 1  Configuration of optimization algorithm.** In all case studies, we used the “Adam” [22] optimizer. We used the TensorFlow implementation and through its callback functions, we configure it to reduce the learning rate as the loss function plateaus and to terminate upon “not-a-number” loss.

<table>
<thead>
<tr>
<th>Case study</th>
<th>Epochs</th>
<th>Learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>200</td>
<td>0.1</td>
</tr>
<tr>
<td>#2</td>
<td>1000</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**A. Case study #1: observable output, classical calibration and discrepancy estimation**

In this example, we modified a set of equations popularized by Forrester et al. [23] in their study of multi-fidelity modeling. The output of interest is given by:

\[
y(x) = \zeta(x) = (6x - 2)^2 \sin(12x - 4) + 10(x - 0.5)^3 + 10(x - 0.5) - 5.0,
\]

and that the simulator is defined as:

\[
\eta(x, \theta) = (6x - 2)^2 \sin(\theta_0x - 4) + \theta_1(x - 0.5) + \theta_2
\]

where \( \theta = [\theta_0, \theta_1, \theta_2]^T \) are the calibration parameters.

As we optimize the graph model illustrated in Fig. 2, we considered the cases where 5 and 11 samples are available. Table 2 details the three multi-layer perceptrons tested as discrepancy correction models. To average out the influence of the initial calibration parameter, we repeated the experiments with 100 different points using Latin hypercube sampling [24]. The samples for the calibration parameters are drawn within the following intervals:

\[
\theta_0 \in [6.0, 18.0], \theta_1 \in [5.0, 15.0], \text{ and } \theta_2 \in [-7.5, -2.5].
\]

In this case study, calibration and discrepancy estimation clearly carry a physical meaning. Upon examination of Eqs. 9 and 10, the unknown “actual” values of the calibration parameters are \( \theta_0 = 12.0, \theta_1 = 10.0, \text{ and } \theta_2 = -5.0 \), which makes the discrepancy correction term \( \delta(x) = 10(x - 0.5)^3 \). Figure 5 illustrates how the output can be decomposed into \( \eta(x, \theta_{\text{actual}}) \) and \( \delta(x) \). Interestingly, the discrepancy is rather small compared to the output range and it deviates only slightly from zero when \( x < 0.2 \) or \( x > 0.8 \). Building a perfect model for such discrepancy is actually not trivial.

Table 3 shows the computational cost associated with the calibration and discrepancy estimation and Fig. 6 illustrates the convergence of the loss function throughout the model optimization. Given the problem simplicity, it is no surprise how fast the task runs and converges. As expected, increasing the complexity of neural networks (hence the parameters to be trained) as well as the number of training data points increases the computational cost.

Figure 7 shows the spread of calibration parameters after optimization is done (initialization is done at random within intervals specified in Eq. 11). The multi-layer perceptron complexity seems to marginally impact the final result...
Table 2  Details of multi-layer perceptron architecture details for case studies #1 and #2 (number of neurons per layer and respective activation function).

<table>
<thead>
<tr>
<th>Layer</th>
<th>Deep</th>
<th>Medium</th>
<th>Shallow</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>20/elu</td>
<td>10/elu</td>
<td>5/elu</td>
</tr>
<tr>
<td>#2</td>
<td>10/elu</td>
<td>5/elu</td>
<td>1/linear</td>
</tr>
<tr>
<td>#3</td>
<td>5/elu</td>
<td>1/linear</td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>1/linear</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Number of parameters 311 81 16

Fig. 5  Decomposing $\zeta(x)$ from Eq. 9 such that $\zeta(x) = \eta(x, \theta_{actual}) + \delta(x)$, where $\delta(x) = 10(x - 0.5)^3$.

Table 3  Approximated wall clock time in seconds associated with one run of the calibration and discrepancy estimation for case study #1.

<table>
<thead>
<tr>
<th>Observations</th>
<th>Deep</th>
<th>Medium</th>
<th>Shallow</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3 (s)</td>
<td>1 (s)</td>
<td>0.8 (s)</td>
</tr>
<tr>
<td>11</td>
<td>6 (s)</td>
<td>4 (s)</td>
<td>3 (s)</td>
</tr>
</tbody>
</table>

Fig. 6  Example of loss function history for case study #1.

(a) 5 observations.  (b) 11 observations.

of the optimization. The shallow multi-layer perceptrons resulted in consistent spreads for the optimum calibration parameters with both 5 and 11 points. On the other hand, both the medium and the deep multi-layer perceptron resulted in tighter spreads for $\theta_1$ when compared to the shallow architecture for with both 5 and 11 points.
(a) 5 observations.

(b) 11 observations.

Fig. 7  Spread of calibrated parameters for case study #1 (true values $\theta_0 = 12.0$, $\theta_1 = 10.0$, $\theta_2 = -5.0$).

Figure 8 shows the predictions for the different multi-layer perceptrons detailed in Table 2 after training with 5 and 11 observations. With the shallow multi-layer perceptrons, the calibration seems to be poor, as $\eta(x, \theta_{actual})$ diverges from $\zeta(x)$ when $x > 0.5$. Then the discrepancy correction model compensates for it and brings the predictions of $\eta(x, \theta_{actual}) + \delta(x)$ close to $\zeta(x)$ again. Figure 9 illustrates this by showing the median predictions and the prediction ranges for the models relative to Fig. 8a. As a disclaimer, the prediction ranges are not uncertainty estimators as these reflect variations on the parameters $(\theta, w, b)$ found during optimization. The medium and deep multi-layer perceptrons seem to be robust to the number of points and calibration is relatively effective. Then, the discrepancy correction is small for the most part (as expected from Fig. 5). Figure 10 illustrates this by showing the median predictions and the prediction ranges for the models relative to Fig. 8b.

B. Case study #2: hidden output, calibration parameter and discrepancy terms physically meaningful

In this case study, we consider an example in which the multi-layer perceptron compensates for discrepancy in a hidden node of the graph. The output of interest is given by:

$$y(x, \xi_2, \xi_3) = \zeta_4(x, \xi_2, \xi_3) = \frac{(2\pi x_1)^2}{\sqrt{\left(\xi_2 - 0.03x_1^2\right)^2 + \left(2\pi x_1\xi_3\right)^2}},$$

$$\zeta_3(x, \xi_1, \xi_2) = \frac{\sqrt{0.03\xi_2} \left(6 \times 10^6x_1^{0.01} + 5 \times 10^6\right)x_2}{\xi_1} + \frac{x_1}{250} - \frac{x_2}{50} + \frac{x_1}{10x_2},$$

$$\zeta_2(x) = 0.002667\pi \xi_1 + x_1 - 10x_2 + \frac{x_1}{x_2},$$

$$\zeta_1(x) = 5 \times 10^6x_1^{0.01}x_2.$$
The simulator is built with a discrepancy (missing physics) in both $Z_1(x)$ and $Z_2(x), Z_3(x, z_1, z_2)$. The model is defined as:
\[
\eta_4(x, \eta_2, \eta_3) = \frac{(2\pi x_1)^2}{\sqrt{(\eta_2 - 0.03 x_1^2)^2 + (2\pi x_1 \eta_3)^2}},
\]
\[
\eta_3(x, \eta_1, \eta_2) = \frac{\sqrt{0.03\eta_2 (6 \times 10^6 x_1^{0.01} + 5 \times 10^6) x_2}}{\eta_1},
\]
\[
\eta_2(x; \theta) = \theta \pi \eta_1, \text{ and}
\]
\[
\eta_1(x; \alpha) = 5 \times 10^6 x_1^{0.01} x_2.
\]

where \(\theta\) is the only calibration parameter.

As we optimize the graph model illustrated in Fig. 3, we considered the cases where 5, 20, 50, and 100 observations are available. Table 4 details the three multi-layer perceptron used as discrepancy correction model. To average out the influence of the initial calibration parameter, we repeated the experiments with 100 different points drawn at random within the following interval \(\theta \in [0.002667, 0.003333]\).

Table 4 Details of multi-layer perceptron architecture details for case studies #2 (number of neurons per layer and respective activation function).

<table>
<thead>
<tr>
<th>Layer</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>10/sigmoid</td>
</tr>
<tr>
<td>#2</td>
<td>5/elu</td>
</tr>
<tr>
<td>#3</td>
<td>1/linear</td>
</tr>
<tr>
<td>Number of parameters</td>
<td>91</td>
</tr>
</tbody>
</table>

Upon examination of Eqs. 12 and 13, the unknown ‘actual’ value of the calibration parameter is \(\theta = 0.002667\). The discrepancy correction terms are \(\tilde{\phi}_1(x) = \frac{x_1}{250} - \frac{x_2}{50} + \frac{x_1}{10x_2}\) (associated with \(\eta_3\)) and \(\tilde{\phi}_2(x) = x_1 - 10x_2 + \frac{\Delta_1}{x_2}\) (associated with \(\eta_2\)). Figure 11 illustrates how the output can be decomposed into \(\eta(x, \theta_{\text{actual}})\) and a discrepancy term that will be injected in a hidden node of the graph.

![Decomposing \(\zeta_1(x, \zeta_2, \zeta_3)\) from Eq. 12](image1)

![\(\eta_4(x, \eta_2, \eta_3)\) from Eq. 13 with \(\theta_{\text{actual}}\)](image2)

![Discrepancy term.](image3)

**Fig. 11** Decomposing \(\zeta_1(x, \zeta_2, \zeta_3)\) such that \(\zeta_1(x, \zeta_2, \zeta_3) = \eta_4(x, \eta_2, \eta_3) + \text{discrepancy} \). The discrepancy term is due to the missing physics in the hidden nodes (for the sake of the plot, \(\zeta_1(x, \zeta_2, \zeta_3) - \eta_4(x, \eta_2, \eta_3)\)).

We first look at the optimization of model parameters (calibration and hyper-parameters of discrepancy compensator model). Table 5 shows the computational cost associated with the calibration and discrepancy estimation and Fig. 12 illustrates the convergence of the loss function throughout the model optimization. This is a simple problem and task has runs and converges fast. As expected, increasing the number of training data points increases the computational cost.
Approximated wall clock time in seconds associated with one run of the calibration and discrepancy estimation for case study #3.

<table>
<thead>
<tr>
<th>Observations</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
</tr>
</tbody>
</table>

Fig. 12 Example of loss function history for case study #2.

Fig. 13 Spread of calibrated parameters for case study #2 (true value $\theta = 0.002667$).

Fig. 13 shows the spread of calibration parameters after optimization is done. Except for the case with 5 observations, there seems to be only marginal differences across the other cases. We expect that the fast convergence to $\theta = 0.002667$ frees up the discrepancy model to compensate for the missing physics.

Now let us focus on the predictions of the calibrated and discrepancy-adjusted models. Figure 14 shows the model predictions for the simulator and discrepancy-adjusted simulator after training with 5, 20, 50, and 100 observations against the actual output values. Calibrated model is $\eta_4(x, \eta_2, \eta_3)$ from Eq. 13 with $\theta$ coming from optimization. As shown in Fig. 13, optimization brought $\theta$ close to the actual output values regardless the number of observations. Therefore, the model predictions are equally close to the actual output values. Nevertheless, there is still residual epistemic uncertainty due to the missing physics in the hidden nodes of the graph. The adjusted model includes the calibrated model with the hidden nodes having missing physics being adjusted by multi-layer perceptron. In such case, with only 5 observations, the discrepancy-adjusted model ends up overly compensated and its predictions are even further from the actual output values when compared to the simply calibrated model. With increased number of observations that is no longer a problem and models tend to improve their agreement with actual output values.

Finally, Fig. 15 illustrates how the model predictions look across the input space. Fig. 15a shows the actual $\zeta_1(x, \zeta_2, \zeta_3)$ from Eq. 12 as well the 100 observations used in the model optimization. Figs. 15b and 15c show the calibrated model and discrepancy-adjusted models, respectively. The discrepancy corrector model significantly reduces
V. Summary and Future Work

In this work, we proposed an approach for calibration of computer models assisted by graph models and neural networks. We tested our approach with the help of analytical examples where we could test (a) discrepancy estimation of observable vs. hidden nodes in the graph, (b) the number of available observations; and (c) the neural network architecture (number of layers and neurons, activation functions, etc.).

Our proposed framework was able to account for lack of knowledge regarding value of model parameters, and discrepancy between the simulation code and the actual physical system. For observable output example studied in this paper, we found that the complexity of multi-layer perceptron affects the optimization results. While having a shallow architecture for discrepancy corrector results in a wider spread for calibration parameters, using medium or deep multi-layer perceptron yields to tighter spreads, regardless of the number of observations. In addition, prediction results showed that shallow multi-layer perceptron performs poorly during calibration, whereas medium and deep networks perform relatively well. In the hidden output example studied in this paper, we examined the ability of the approach to handle discrepancies within the unseen nodes and parameters. We showed that the model can be calibrated and the discrepancy corrector can be trained simultaneously and successfully, given the adequate number of observations.

Overall, our approach can be utilized to estimate parameters and discrepancy of computer models with the help of graphs and neural networks. Nevertheless, in order to improve our the framework, and as topic for future research, we suggest studying uncertainty in observations and hyperparameters (for the physics-based and data-driven nodes).

Acknowledgments

This work was supported by the University of Central Florida (UCF). Nevertheless, any view or recommendations expressed in this material are those of the authors alone. Therefore, UCF does not accept any liability in regard thereto.
References


