Journal of Computational Physics 521 (2025) 113542

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Democratizing uncertainty quantification

Linus Seelinger[a,*], Anne Reinarz[b], Mikkel B. Lykkegaard[c], Robert Akers[d], Amal M.A. Alghamdi[e], David Aristoff[f], Wolfgang Bangerth[g], Jean Bénézech[h], Matteo Diez[i], Kurt Frey[j], John D. Jakeman[k], Jakob S. Jørgensen[e], Ki-Tae Kim[l], Benjamin M. Kent[m], Massimiliano Martinelli[m], Matthew Parno[n], Riccardo Pellegrini[i], Noemi Petra[l], Nicolai A.B. Riis[o,e], Katherine Rosenfeld[j], Andrea Serani[i], Lorenzo Tamellini[m], Umberto Villa[p], Tim J. Dodwell[c], Robert Scheichl[q]

[a] *Scientific Computing Center, Karlsruhe Institute of Technology, Karlsruhe, Germany*
[b] *Department of Computer Science, Durham University, Durham, United Kingdom*
[c] *digiLab, Exeter, United Kingdom*
[d] *UK Atomic Energy Authority, Oxford, United Kingdom*
[e] *Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Lyngby, Denmark*
[f] *Department of Mathematics, Colorado State University, Fort Collins, CO, USA*
[g] *Department of Mathematics, Department of Geosciences, Colorado State University, Fort Collins, CO, USA*
[h] *Centre for Integrated Materials, Processes and Structures, Department of Mechanical Engineering, University of Bath, Bath, United Kingdom*
[i] *National Research Council-Institute of Marine Engineering, Rome, Italy*
[j] *Institute for Disease Modeling, Global Health Division, Bill & Melinda Gates Foundation, USA*
[k] *Optimization and Uncertainty Quantification, Sandia National Laboratories, Albuquerque, NM, USA*
[l] *University of California, Merced, USA*
[m] *National Research Council-Institute for Applied Mathematics and Information Technologies "E. Magenes", Pavia, Italy*
[n] *Solea Energy, Thetford, VT, USA*
[o] *Copenhagen Imaging ApS, Herlev, Denmark*
[p] *The University of Texas at Austin, USA*
[q] *Institute for Mathematics / Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Germany*

ARTICLE INFO

Keywords:
Uncertainty quantification
Numerical simulation
Benchmarks
Scientific software
High-performance computing

ABSTRACT

Uncertainty Quantification (UQ) is vital to safety-critical model-based analyses, but the widespread adoption of sophisticated UQ methods is limited by technical complexity. In this paper, we introduce UM-Bridge (the UQ and Modeling Bridge), a high-level abstraction and software protocol that facilitates universal interoperability of UQ software with simulation codes. It breaks down the technical complexity of advanced UQ applications and enables separation of concerns between experts. UM-Bridge democratizes UQ by allowing effective interdisciplinary collaboration, accelerating the development of advanced UQ methods, and making it easy to perform UQ analyses from prototype to High Performance Computing (HPC) scale.
In addition, we present a library of ready-to-run UQ benchmark problems, all easily accessible through UM-Bridge. These benchmarks support UQ methodology research, enabling reproducible performance comparisons. We demonstrate UM-Bridge with several scientific applications, harnessing HPC resources even using UQ codes not designed with HPC support.

* Corresponding author.
E-mail address: linus.seelinger@kit.edu (L. Seelinger).

https://doi.org/10.1016/j.jcp.2024.113542
Received 18 March 2024; Received in revised form 9 September 2024; Accepted 25 October 2024

Available online 30 October 2024

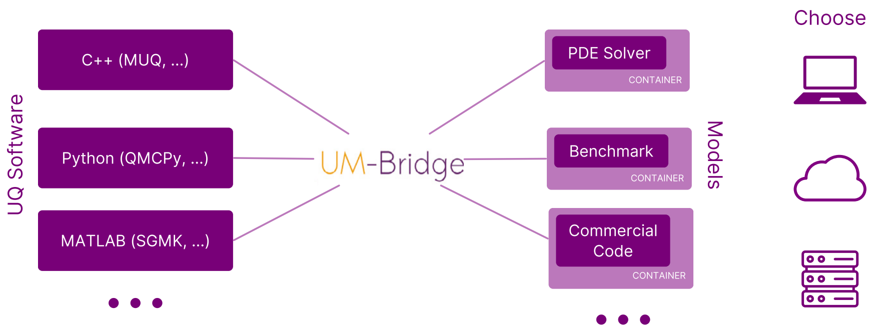

Fig. 1. UM-Bridge connecting UQ and numerical models through a universal interface.

1. Introduction

The quantification of uncertainties is crucial for safety-critical applications where an understanding of the associated risks is integral to a broader analysis, such as in engineering design, clinical trials, weather and extreme event forecasting, environmental risk assessment, and infectious disease control, to name but a few. Such risk analyses are typically model-driven since they involve extrapolating complex processes into the future. Accurate quantitative information on uncertainties in predictions underlying decisions can lead to deeper insight into the problem and, consequently, better decisions.

The last couple of decades have seen remarkable growth in the use of UQ, including uncertainty propagation and Bayesian inference [1]. This is partly due to efficient and robust techniques such as Latin hypercube, quasi-Monte Carlo, ADVI, HMC, and NUTS [2–4], and partly due to flexible, open-source implementations of those algorithms, e.g. SciPy stats [5], Stan [6], PyMC [7], and Pyro [8].

The success of these software packages can be attributed to the high level of abstraction with which they allow a user to define their statistical model. The subtle details of the underlying algorithms are not exposed to the user, who can simply specify their statistical model using high-level representations of the relevant components (prior, likelihood, observation model, etc.) and solve it using existing general-purpose algorithms. This arrangement is a form of *separation of concerns*: The algorithm developer does not require detailed knowledge of the domain expert's application and vice versa.

Notwithstanding those successes, advanced statistical inference tools are not yet employed extensively in conjunction with computationally demanding models. Such models appear, for example, in engineering design, in geophysics, or in medical imaging. We believe that this is in part due to a chicken-and-egg problem: UQ has not been demonstrated to be useful in those areas, but demonstration requires advanced knowledge of both application and UQ techniques that few research groups have. By developing the first generic interface between UQ and application, our contribution addresses both of these issues.

We rely on an existing and largely universal theoretical vocabulary for the interaction between models and UQ: The application model (or data-generating process) is represented as a *forward operator* $F(\theta)$, which is a map from parameter space to data space. In forward UQ, the parameter θ is uncertain, and we want to infer the probability distribution of the outputs $F(\theta)$; in inverse problems, we specify a likelihood model for the observed data based on $F(\theta)$ and aim to infer the probability distribution of θ. Many UQ algorithms exploit knowledge of derivatives of the forward map to increase computational efficiency; as a consequence, application models needs to be able to expose the computation of gradients (Jacobians) or Hessians of F, provided they are available within a particular application. This short list of shared components – evaluating $F(\theta)$ and, perhaps, derivatives – can be understood as a form of weak coupling between the statistical and the application model: Only a small part of the machinery inside the application model needs to be exposed to the statistical model, while the application model does not need to see the statistical model at all. This simple fact implies that it is – in theory – relatively straightforward to apply a wide variety of UQ methods to arbitrary application models.

Despite that, in practice, the software packages mentioned above are typically only used to solve problems with relatively simple data-generating processes, such as basic numerical integration or hierarchical Bayesian regression. This dichotomy can at least in part be attributed to the high technical complexity of combining advanced software packages for statistical inference with state-of-the-art implementations of sophisticated application models. The technical complexity arises since there is no equally universal counterpart to the mathematical interface in software, and becomes particularly challenging in HPC applications.

To address these issues, we introduce UM-Bridge (Fig. 1): The first universal link between *any* UQ package and *any* forward model, breaking down complex software stacks into manageable components. Conceptually, we introduce a high-level abstraction that enables separation of concerns between the statistical and physical modeling domains. On a practical level, language-specific integrations make UM-Bridge models appear as native entities (classes, function calls etc.) in the respective programming language or even as native model classes in specific UQ packages. Additionally, model agnostic load balancers provide easy access to parallellism.

Our unified interface allows linking arbitrary tools and makes it easy to swap out components on either side, reducing user lock-in. UM-Bridge further offers portable models through optional containerization, which offers a high degree of separation of concerns between UQ and model experts. In order to strengthen rigorous performance comparisons in UQ, we present what we believe to be the first library of UQ benchmark problems, developed by the authors of this paper and provided as part of an open-source software repository. UM-Bridge support makes the benchmarks available to virtually any UQ software and ensures portability and reproducibility via containerization. We describe these UQ benchmarks in detail in Appendix A.

In addition, UM-Bridge offers a universal approach to scale-up UQ applications on parallel compute clusters that is fully model agnostic. In a black-box fashion, UM-Bridge allows for HPC-scale computations with UQ codes that had never been intended for that purpose. In Section 5, we describe in detail the workflow for three complex UQ applications, where UQ software is successfully coupled with HPC codes.

Thus, by separating concerns of UQ, model and HPC experts, UM-Bridge democratizes UQ. It paves the way to its widespread adoption and to scientific innovation across a wide range of application fields, provides transparent scalability and facilitates reproducible benchmarking.

2. UM-Bridge – a new paradigm for UQ software integration

The aim of UM-Bridge is to establish a new paradigm for UQ software integration, enabling the analysis of previously intractable problems, development of better methods, and streamlining the workflow for every stakeholder. At its core, UM-Bridge links arbitrary UQ and model applications through a network protocol, which leads to an entire range of new opportunities for UQ. Stakeholders include applied scientists and engineers who employ UQ methodologies to interrogate challenging problems, developers of novel UQ methods who require objective and reproducible benchmarks to validate their methods, and application modelers who wish to integrate their model into a UQ workflow. Here, we outline the benefits of the UM-Bridge paradigm to the various stakeholders.

- **Benefits for scientists and engineers.** While UQ techniques are critical for making trustworthy predictions, particularly in safety-critical contexts, rigorous UQ is often held back by perceived complexity and cost of setting up a UQ workflow. UM-Bridge contributes to the establishment of "UQ-as-a-Service" by providing an environment to quantify uncertainties through a streamlined interface for all stages of the UQ workflow. This allows researchers with varying levels of expertise to engage in the rigorous analysis of uncertainties.
- **Benefits for UQ method developers.** The development of efficient algorithms for UQ requires access to objective and challenging problems that can be used to adequately benchmark the algorithms in question. UM-Bridge enables UQ method developers to separate the development of UQ methods, such as the design of experiments, surrogate model construction, and uncertainty propagation, from the model or application being studied. Each UM-Bridge model and benchmark can be accessed from any UQ platform through the respective UM-Bridge integration, promoting flexibility in UQ workflows and enabling method developers to easily test their methods on new problems.
- **Benefits for model code developers.** The coupling of UQ software with an application model currently requires intricate knowledge of both software packages. The UM-Bridge model interaction interface (Section 3.1) allows for application modelers to simply equip their model with UM-Bridge support and otherwise focus on the performance of the application model itself. They can thus create performant models that are both fully compatible with virtually any UQ software and easy to share, enabling effective collaboration with their UQ counterparts.

In the following, we outline some of the mechanisms by which UM-Bridge accelerates development and simplifies adoption of advanced UQ methods, contributing to the democratization of UQ.

2.1. Making tools accessible

UM-Bridge makes advanced UQ tools accessible to a wide range of applications by providing a unified and language-agnostic interface. This is especially important since each community has specific requirements and therefore tends to use specific languages and software packages. For example, efficient numerical solvers for partial differential equation (PDE) models are often limited by computational performance and consequently need fine-grained control over memory operations, sometimes use low-level hardware features, and often interact with many other software libraries providing specialized algorithms (see, for example, [19]); they are therefore often implemented in lower-level languages like C++ or Fortran. On the other hand, UQ methods themselves are often far less compute-intensive, are relatively self-contained, and so allow for fast development in high-level languages like Python or Matlab. The UM-Bridge interface allows these communities to continue using their preferred tools, while still being able to collaborate.

Native integrations for various languages and UQ packages (see Table 1) implement the UM-Bridge network protocol both for client side (e.g. UQ software) and server side (numerical models). Many UQ packages such as PyMC [7] or QMCPy [14] have a specific model interface, which the respective integrations implement. UM-Bridge thus acts as a cross-language and cross-platform translation layer.

More generally, any language supporting HTTP communication can implement UM-Bridge support. Models that are written in unsupported languages or that cannot be embedded in a higher-level code may still reap most benefits of UM-Bridge when called from a simple UM-Bridge wrapper. As an example that demonstrates how previously incompatible applications are now readily available to high-level UQ tools, in Section 5.1 we seamlessly apply a Matlab UQ code to a Fortran-based model of a ship hull [20], as shown in Fig. 2 (left).

2.2. Separating concerns

Building UQ applications requires expertise in both UQ and numerical modeling. In addition to a theoretical understanding, it also calls for experience with software packages in both domains. It is often time-consuming or outright infeasible to achieve such an overarching expertise within the constraints of a project.

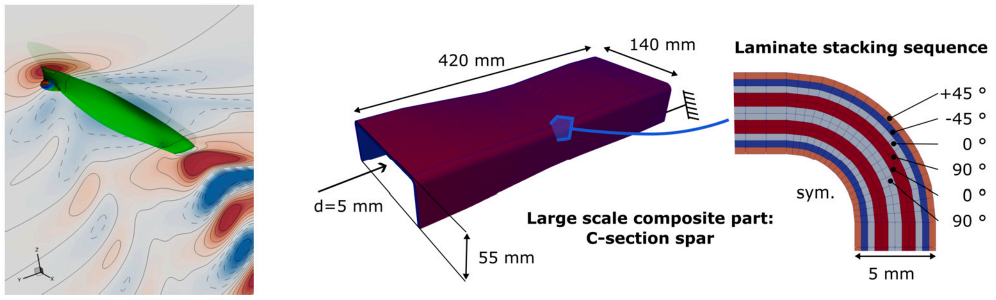

Fig. 2. Fortran code for modeling the pressure distribution and the wave elevation generated by a vessel. UM-Bridge support is achieved through a simple wrapper, enabling parallelized UQ from, e.g., Matlab (left figure). FE model of a carbon fiber composite aerospace component implemented in a complex PDE solver in C++. Through UM-Bridge and containerization, a UQ expert could solve uncertainty propagation without technical knowledge of the model code (center & right figures). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 1
Currently available UM-Bridge integrations. Ticks indicate support for UM-Bridge clients (UQ algorithms) and UM-Bridge servers (models) in the respective language or UQ package.

Language	Client	Server
Python	✓	✓
C++	✓	✓
Matlab	✓	✗
R	✓	✗
Julia	✓	✓

UQ package	Client	Server
CUQIpy [9,10]	✓	✓
emcee [11]	✓	✗
MUQ [12]	✓	✓
PyApprox [13]	✓	✗
PyMC [7]	✓	✗
QMCPy [14]	✓	✗
SGMK [15]	✓	✗
tinyDA [16]	✓	✗
TT-Toolbox [17]	✓	✗
UQPy [18]	✓	✗

Since UM-Bridge makes the mathematical model available through a universal software interface, it now becomes possible to properly separate concerns between the model and UQ experts. The model expert can focus on implementing the map F and (if available) its derivatives, while the UQ expert can focus on UQ aspects. This is especially important when applying state-of-the-art UQ methods to complex models.

In addition, since the UM-Bridge interface is network-based, it becomes straightforward to containerize models in UQ workflows. This allows sharing models among collaborators and running them on any machine or cluster without per-machine installation effort. The model expert may now pass their implementation of the model as a ready-to-use container to the UQ expert, which further contributes to separation of concerns.

A further benefit of containerization is reproducibility of the model outputs. Besides being integral to the scientific method, it is helpful for developing UQ applications across teams. Model container images can be published alongside the scientific results, enabling the exact reproducibility of scientific work. We exploit this feature in the benchmark library below.

Illustrating separation of concerns, in application Section 5.2 a simulation expert developed an advanced modeling application in C++, containerized it, and shared it with a UQ expert, see Fig. 2 (right). The UQ expert was able to quantify the parametric uncertainty using a high-level UQ package written in Python, without further integration work or any deeper technical knowledge of the model implementation.

2.3. *Providing transparent scalability*

A major roadblock to UQ on large-scale models is the need for HPC capability. To achieve that, both the underlying model and the UQ method need support for parallelization in theory and implementation. Additionally, the parallelization techniques involved have to be compatible and tightly linked.

In practice, many UQ packages do support parallelization in some capacity but are often limited to a single machine or a specific way of setting up a parallel simulation. On the other hand, many PDE solvers are able to use HPC resources, typically through Message Passing Interface (MPI). However, they often come with specific assumptions about the nature of the machine they are running on. As part of UM-Bridge, we introduce an architecture that separates parallelization of UQ and model: A cluster-side load balancer accepts parallel model evaluation requests from the UQ code, without making any assumptions on what parallelization technique the UQ code is employing. It then distributes those requests among independent instances of the model, each of which may now also employ any parallelization strategy.

With such an architecture, even thread-parallel UQ codes can offload costly numerical simulation runs to a (potentially remote) cluster. Many existing UQ codes do not require any modification to scale up via UM-Bridge, and neither do typical models. Both cloud-clusters and HPC systems are supported.

We illustrate these concepts with the application in Section 5.3. It shows how an existing UQ code can transparently offload costly model evaluations to a remote cluster of 2800 processor cores, running 100 instances of a sophisticated, parallel tsunami model.

2.4. *Facilitating reproducible benchmarking*

The current lack of available and reproducible UQ benchmarks is severely limiting quantitative comparisons of competing UQ methods. The few existing initiatives are typically limited to a single programming language (e.g., [13]). Support for more languages then requires reimplementation (e.g., [21]), but this is clearly not viable for complex models.

A universal interface like UM-Bridge can address this problem. Thus, we have collected a large number of benchmark problems (described in more detail in Section 4 below), with the goal of being:

- **Representative:** The benchmarks are a curated collection, aiming to cover all properties of models relevant to UQ. That includes different choices of parameter dimension, smoothness of model output, model run time etc. Testing methods across several benchmarks in our collection can be used to demonstrate their behavior on problems ranging from simple artificial tests all the way to realistic large-scale simulations.
- **Easy to use:** All benchmark problems are implemented with UM-Bridge support and have easily accessible documentation. They can readily be used to benchmark any UQ package or prototype code.
- **Reproducible:** Containerization ensures that benchmark problems are fully defined in portable software, avoiding potential ambiguities in definition or implementation details.

The library aims to establish a neutral foundation enabling subsequent studies. We intentionally do not include benchmark results of competing methods, since specific metrics may depend strongly on the goals of such a comparison, and results will change as future methods are developed.

Appendix A gives detailed documentation of all currently available models and benchmark problems. We intend for this library to be a growing collection and encourage future contributions to it. This collection can also serve related areas, like optimization, and an example in this direction can be found in Appendix A.4.

3. UM-Bridge architecture and usage

UQ problems typically revolve around a deterministic model, which is operationalized as a *forward operator* $F : \mathbb{R}^n \to \mathbb{R}^m$. The forward operator takes a parameter vector θ of dimensions n (containing uncertain forcing terms and/or coefficients in the model) to some model output, a vector of dimension m.

Uncertainty propagation problems are then defined by considering θ as a random vector following some distribution $\pi(\theta)$. The goal is to find properties of the distribution of $F(\theta)$, such as moments, quantiles, the probability density function (PDF) or the cumulative distribution function.

On the other hand, Bayesian inference defines the probability of a parameter θ given observed data y in terms of a prior distribution π_{prior} and a likelihood function L:

$$\pi_{\text{posterior}}(\theta|y) \propto \pi_{\text{prior}}(\theta)L(y|\theta).$$

The first factor on the right represents prior information about the parameter distribution we seek, while the likelihood $L(y|\theta)$ provides a measure of distance between model prediction $F(\theta)$ and data y. The goal is then to characterize $\pi_{\text{posterior}}$, for example through sampling or by estimating its moments.

These concepts constitute the typical theoretical context of UQ methods to which UM-Bridge lends itself. In the following, we explore how UM-Bridge implements these concepts.

3.1. *UM-Bridge architecture*

Many forward UQ methods (e.g. many Monte Carlo (MC) samplers, or stochastic collocation [22]) only require model evaluations $\{F(\theta_i)\}$ at a finite number of points $\theta_1, \dots, \theta_N$. Other methods may additionally need derivatives of F at θ_i. Similarly, inversion problems are often solved by pointwise evaluations of π_{post}, for which different methods can be employed (Metropolis-Hastings MCMC [23], Hamiltonian MCMC [3], NUTS [4], etc.). Again, these approaches require only forward model evaluations $F(\theta_i)$ and, in some cases, derivatives at a finite number of points $\theta_1, \dots, \theta_N$.

In general, across most UQ algorithms, the model is typically assumed to provide (some of) the following pointwise mathematical operations:

- Model evaluation $F(\theta)$,
- Gradient $v^\top J_F(\theta)$,
- Jacobian action $J_F(\theta)v$,
- Hessian action.

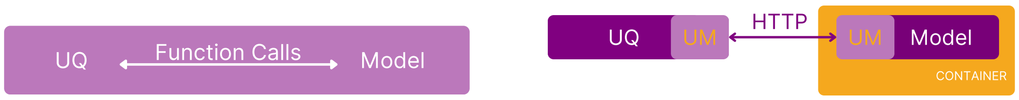

Fig. 3. Monolithic coupling of UQ and model software in a single application (left); UM-Bridge providing a universal interface between UQ and model applications (right). UM-Bridge integrations ("UM") handle network communication behind the scenes. Optionally, models may be containerized.

Any UQ method whose requirements are limited to this list could therefore – in principle – be applied to any model supporting the appropriate operations.

The key idea behind UM-Bridge is to implement the above mathematical interface in an equally universal software interface: We treat UQ algorithm and numerical model as stand-alone applications, linked only through an Hypertext Transfer Protocol (HTTP) based network protocol (see Fig. 3, [24]). In contrast to a monolithic approach, this microservice-inspired architecture enables linking across arbitrary languages and software packages, as well as separating concerns. It should be noted that multilevel, multi-index or multi-fidelity methods (e.g. [25–34]) such as the one in Section 5.3 operate on an entire hierarchy of models. Less accurate but cheaper-to-compute models are used to accelerate the UQ method, leading to considerable gains in efficiency both in forward and inverse UQ problems. While this necessitates multiple models, each individual model still fits in the framework described above.

3.2. *The UM-Bridge protocol*

At its core, the UM-Bridge protocol is built on HTTP and JSON for compatibility with many programming languages and easy implementation. All operations a model may provide are considered optional. For example, models may be implemented with gradients or not, and UQ clients can query and adapt to that. Importantly, this means that the protocol can be extended in the future without breaking compatibility.

UM-Bridge treats model inputs and outputs as lists of vectors, which is inspired the proven internal model interface in MUQ [12]. This enables automatic differentiation with respect to certain components of the input, even when chaining multiple models.

In all the aforementioned integrations of UM-Bridge (see Table 1), the protocol is implemented behind the scenes, making UM-Bridge models available either as simple function calls or fully integrated in the respective software package's model structure.

3.3. *Basic examples*

Let us illustrate how UM-Bridge works in practice with some simple examples. While intentionally basic, the same approach underlies the applications in Section 5.

3.3.1. *Clients - Interacting with models*

First, we need a running model server, providing a forward operator F. This could be a simulation code with UM-Bridge support running directly on your system, a custom one as in Section 3.3.2, or a containerized model. For example, we could download and run the containerized tsunami model from the benchmark library (see Appendix A.1.3) via `docker`:

```
docker run -it -p 4242:4242 \
   linusseelinger/model-exahype-tsunami
```

Once running, any UM-Bridge client can request evaluations of F from that model server. This could be any supported UQ package or a custom code in any supported language (see Table 1). For example, we can call the model from Python via the `umbridge` Python module:

```
url = "http://localhost:4242"
model = umbridge.HTTPModel(url, "forward")

print(model([[0.0, 10.0]]))
```

All that is needed is the URL of the model server (here `http://localhost:4242`), the name of the requested model F (here `forward`) and a parameter θ to evaluate F at. The actual evaluation is then a call to the model with a given parameter $\theta = (0, 10)$, which returns $F(\theta)$. Optionally, arbitrary model-specific configuration options can be passed, for example to set the discretization level of a multifidelity model:

```
print(model([[0.0, 10.0]], {"level": 0}))
```

Similar calls allow access to gradient, Jacobian action and Hessian action, as long as the model implements these operations.

Clients written in other languages follow a similar pattern. For example, code equivalent to the above would look like this in C++:

```
std::string url = "http://localhost:4242";
umbridge::HTTPModel model(url, "forward");

std::vector<std::vector<double>> outputs
  = model.Evaluate({{0.0, 10.0}});
```

3.3.2. *Servers - Defining models*

Defining a model conversely requires implementing the model map F, which could be anything from simple arithmetic all the way to advanced numerical simulations. In addition, model name as well as input and output dimensions are specified. The following minimal example in Python implements the multiplication of a single input value by two:

```
class TestModel(umbridge.Model):

  def __init__(self):
    super().__init__("forward")

  def get_input_sizes(self, config):
    return [1] # Input dimensions

  def get_output_sizes(self, config):
    return [1] # Output dimensions

  def __call__(self, parameters, config):
    output = parameters[0][0] * 2
    return [[output]]

  def supports_evaluate(self):
    return True

umbridge.serve_models([TestModel()], 4242)
```

The code sets `supports_evaluate` to indicate to UM-Bridge that model evaluations are supported. Likewise, gradients, Jacobian- or Hessian-action could be implemented and marked as supported.

The final line instructs UM-Bridge to make the model available to clients, acting as a server. UM-Bridge servers written in other languages follow the same principle.

3.3.3. *Containers - Making models portable*

UM-Bridge models ranging from simple test problems all the way to advanced simulation codes can be containerized for portability, and possibly be published in public repositories. Such a container can then be launched on any local machine or remote cluster without installation effort, and users can readily interact with it as in Section 3.3.1.

Defining such a container comes down to instructions for installing dependencies, compiling if needed, and finally running the desired UM-Bridge server code. For example, a docker container wrapping the example server above can be defined in the following simple Dockerfile:

```
FROM ubuntu:latest

COPY minimal-server.py /

RUN apt update && \
   DEBIAN_FRONTEND="noninteractive" apt install -y python3-pip && \
   pip install umbridge

CMD python3 minimal-server.py
```

Once built, `docker run` can launch an instance of the container as above in Section 3.3.1.

Some legacy tools are built for one-shot simulation runs controlled by an input file, and may be too rigid to be called from a UM-Bridge server code like the one in Section 3.3.2. In such cases, a UM-Bridge server can act as a small wrapper application that, on every evaluation request, creates an input file, launches the model software, and then evaluates the code's output files. Such file based interfaces can be error-prone when set up individually, but are reproducible and portable when containerized.

3.4. *Scaling up on clusters*

We enable scaling up UQ applications by providing universal configurations for cloud clusters and tooling for HPC systems. Fig. 4 illustrates how their key component, a cluster-side load balancer, takes concurrent evaluation requests from a UQ client and distributes them across many independent model instances.

As the client-side interface is entirely identical, a UQ software is completely oblivious to the model being executed on a cluster instead of the local machine. However, the UQ software may now send multiple concurrent evaluation requests to the cluster. Since the model evaluation is typically costly compared to the UQ algorithm itself, a UQ package spawning hundreds of threads on a laptop offloading work to a large cluster is perfectly viable. Likewise, since every model instance only receives sequential model evaluations via UM-Bridge, no modification to the model is needed compared to running on a single machine. In addition, containerized models can be run on cloud or HPC systems without additional setup cost as long as the HPC systems support containers.

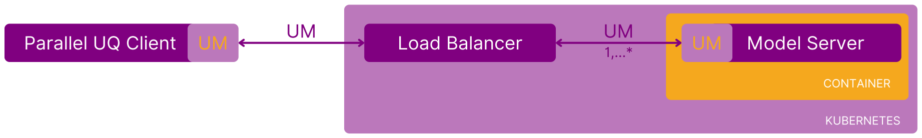

Fig. 4. UM-Bridge provides a general-purpose Kubernetes setup for scaling up UQ applications. It runs many parallel instances of containerized models and distributes concurrent requests from a UQ client among them. Any UM-Bridge supporting UQ or model code readily works with this setup.

Since the UQ software is not involved in distributing evaluations across the cluster, parallelism within many existing UQ packages works out of the box with this setup. For example, the Sparse Grids Matlab Kit (SGMK), MUQ, QMCPy, and PyMC have all been tested successfully across a number of different models. Some of these applications are discussed in Section 5.

Note that, due to the flexibility of the interface, more complex architectures are easy to construct. For example, as in Section 5.3, a multilevel UQ code may at the same time be pointed to a fast surrogate model running on the same workstation and a remote cluster for full-scale model runs, accommodating for heterogeneous resource demand across levels

3.4.1. *UM-Bridge on kubernetes clusters*

One implementation of this architecture is based on kubernetes for cloud systems. The reasoning behind using kubernetes is that (i) existing model containers can be run without modification, (ii) the entire setup can be specified in configuration files and is easily reproducible, and (iii) it can be run on systems ranging from single servers to large-scale clusters and is available on many public cloud systems.

Even though they do not offer the same level of support for containers, UM-Bridge also supports HPC systems managed by SLURM or PBS (Section 3.4.2), which may be interesting for low network latency. Further, UM-Bridge cloud support could be implemented for Docker swarm or Nomad. We prioritized Kubernetes over these due to good parallel scaling and deep integration with public cloud providers.

We employ HAProxy as a Kubernetes load balancer. HAProxy was originally intended for web services processing large numbers of requests per backend container. However, we set it up such that only a single evaluation request is ever sent to a model server at once, since we assume running multiple numerical model evaluations concurrently on a single model instance would lead to performance degradation.

Kubernetes can be fully controlled through configuration files. The UM-Bridge kubernetes setup can therefore be applied on any kubernetes cluster by cloning the UM-Bridge Git repository and executing

```
kubectl apply -f FILENAME
```

on the provided configuration files. This procedure is described in more detail in the UM-Bridge documentation, and takes only a few minutes.

The only changes needed for a custom application are docker image name, number of instances (`replicas`) and resource requirements in the provided `model.yaml` configuration. Below is an example of an adapted model configuration, namely the one used in the sparse grids application in Section 5.1:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: model-deployment
spec:
  replicas: 48
  template:
    metadata:
      labels:
        app: model
    spec:
      containers:
      - name: model
        image: linusseelinger/model-l2-sea:latest
        env:
        - name: OMP_NUM_THREADS
          value: "1"
        resources:
          requests:
            cpu: 1
            memory: 1Gi
          limits:
            cpu: 1
            memory: 1Gi
```

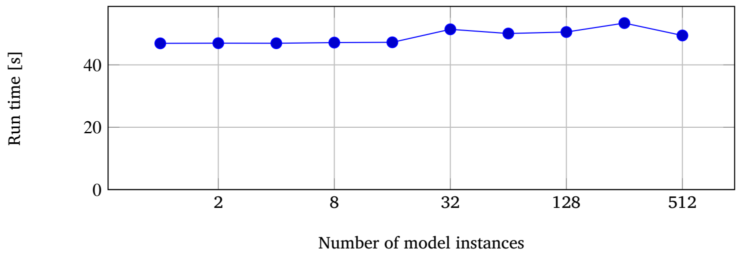

Fig. 5. Synthetic weak scaling test of the kubernetes setup on Google Kubernetes Engine (GKE) for various numbers of parallel model instances, requesting a constant number of evaluations per model instance.

In addition, we provide a kubernetes setup that allows for MPI parallelism across containers and therefore across compute nodes. It requires the model container to inherit from the `mpi-operator` base image, but behaves the same in any other regard.

Section 5.3 demonstrates the UM-Bridge kubernetes setup on 2,800 processor cores running 100 parallel model instances. We further performed synthetic weak scaling tests running up to $N = 512$ instances of the L2-Sea model (see Appendix A.1.2), configured to use 1 processor core each and taking around 2.5 s per evaluation. We then requested $20 \cdot N$ model evaluations from the cluster. As Fig. 5 shows, we did not observe a significant bottleneck, indicating that this setup could be scaled further. In addition, the model itself could be parallelized. With a model parallelized across 100 cores, we expect similar results to the 512 instance test while fully utilizing a 51,200 core cluster.

3.4.2. UM-Bridge on HPC systems

UM-Bridge also includes support for SLURM or PBS based HPC systems. In this case, a custom component takes UM-Bridge evaluation requests from the client and translates those into jobs managed by HyperQueue [35]. HyperQueue then acts as the load balancer, spawning model instances as HPC scheduler jobs and passing evaluation requests to them as needed.

Models may be local installations of the simulation software or containerized versions. While typical HPC systems lack Docker support, apptainer [36] is increasingly available, where the same abstraction of parallelization effort as in the kubernetes case is achieved, both regarding model and UQ.

4. UQ benchmark library

An important part of the work we present herein is a sizable library of UQ benchmark problems. This collection is a collaboration with more than 15 contributors (coauthors of this paper) from more than 10 international institutions. Each benchmark is implemented as a UM-Bridge-supporting software and provided as a ready-to-run container. We offer three types: “Models” implement a numerical (forward) model of a physical phenomenon. Many expose a number of configuration options for a high level of control of the details of the model to be solved. “Inference problems” in turn define a specific inverse UQ problem, usually in terms of a Bayesian posterior. “Propagation benchmarks” define a very specific forward UQ problem to be solved for the model they implement, and are much more restricted in order to ensure comparable and reproducible results.

Each model or benchmark is accompanied by a markdown file that documents how to run the published containers, input and output dimensions, configuration options, and a mathematical description of the map (or a reference to an existing publication). A documentation website presents the entire library in an accessible way, drawing from these markdown files behind the scenes. The Docker containers [37] for our benchmarks are built automatically in a CI system, ensuring validity through automated testing.

Appendix A.1 contains detailed descriptions of all models, Appendix A.2 describes inference benchmarks and Appendix A.3 describes propagation problems. Tables 2, 3, and 4 provide brief summaries.

4.1. Models

The models in our library range from simple analytic functions to complex numerical simulations (see Table 2). In contrast to the propagation benchmarks, models intentionally offer configuration options, allowing users to explore model variations and to adapt them to their needs. For example, the tsunami model can be set to operate on three different refinement levels. When developing algorithms, operating on fast approximate model levels may be more suitable for quick test runs.

Mathematically, following Section 3.1, we consider a model to be a function $F : \mathbb{R}^n \to \mathbb{R}^m$ taking a parameter vector θ to a model output $F(\theta)$. Some models additionally offer derivatives, for example in terms of Hessian actions.

4.2. Bayesian inference benchmarks

Inference benchmarks define a (potentially not normalized) probability density function $\pi : \mathbb{R}^n \to \mathbb{R}$ whose properties are to be estimated. They are therefore restricted to a one-dimensional output. For numerical stability, our definition of the interface requires implementations to provide the logarithm of π. See Table 3 for a list of inference problems that are currently part of the library.

Table 2
List of models.

Models	
Name	Short description
Euler-Bernoulli beam	Deformation of an Euler-Bernoulli beam with a spatially variable stiffness parameter.
L2-Sea	Total resistance estimation of the DTMB 5415 destroyer-type vessel at model scale by potential flow.
Tsunami	Propagation of the 2011 Tohoku tsunami modeled by solving the shallow water equations.
Composite material	Elastic deformation of L-shaped composite with random wrinkles.
Tritium desorption	Microscopic transport of tritium through fusion reactor materials using Foster-McNabb equations.
Agent-based disease transmission	Transmission of disease in a heterogeneous population using EMOD, a stochastic agent based disease transmission model.
Membrane model	Deformation of a membrane for a fixed right-hand side given the stiffness values on an 8×8 grid that makes up the membrane.
Cookies problem	Simplified thermal problem with uncertain conductivity coefficient in 8 circular regions (“cookies”), while it is known (and constant) elsewhere (“oven”).

Table 3
List of inference benchmarks.

Inference benchmarks	
Name	Short description
Analytic densities	Infers the PDF of various analytic functions.
Membrane model	Infer the PDF of stiffness values of a membrane from measured deformation data.
Tritium diffusion posterior	Compute the posterior density of the input parameters given by experimental data.
Disease transmission model	Agent-based model of disease transmission in an entirely susceptible population with correlation between disease acquisition and transmission. The aim is to infer the disease parameters that result in 40% of the population infected at the end of the outbreak.
Deconvolution problem	Infer the posterior distribution for a 1D deconvolution problem with Gaussian likelihood and four different prior distributions, configurable via a parameter.
Computed tomography	Compute a posterior distribution for a 2D X-ray CT image reconstruction problem, with a Gaussian noise distribution.
Inverse heat equation	Evaluate the posterior distribution for a 1D inverse heat equation with Gaussian likelihood and Karhunen-Loève parameterization of the uncertain coefficient.
Beam inference	Bayesian inverse problem for characterizing the stiffness in an Euler-Bernoulli beam given observations of the beam displacement with a prescribed load.
Tsunami Source	Infer parameters describing the initial displacements leading to the tsunami from the data of two available buoys located near the Japanese coast.
Poisson	Estimate a spatially varying diffusion coefficient in an elliptic PDE given limited noisy observations of the PDE solution.
p-Poisson	Estimate a two dimensional flux boundary condition for a nonlinear p-Poisson PDE in a three dimensional domain.

Some of the benchmarks define a Bayesian inference problem based on a model and fixed data y,

$$\pi_{\text{posterior}}(\theta) \propto \pi_{\text{prior}}(\theta) L(y|\theta),$$

where the likelihood $L(y|\theta)$ relates observed data to model output, and is itself based on the forward model $F : \mathbb{R}^n \to \mathbb{R}^m$ defined by the benchmark. π_{prior} denotes the prior probability distribution. Some UQ tools specialized in Bayesian inference may require access to the individual factors of Bayes' formula above, and possibly the model output itself. For this reason, our interface also defines functions to query those components individually.

4.3. Uncertainty propagation benchmarks

We construct benchmark problems like the models above but intentionally restrict them to the minimum necessary options in order to ensure reproducibility. Some benchmarks in the implementation are in fact more restrictive versions of the model codes above.

The goal of uncertainty propagation is to determine the effect of uncertainties in model parameters on model outputs. The parameter θ is specified to be a random variable with a distribution documented in the markdown file. The solution of a propagation benchmark is then an approximation to the distribution of $F(\theta)$, or an estimate of some derived quantity like $\mathbb{E}[G(F(\theta))]$ where G is an “output functional”. Table 4 shows the currently available propagation benchmarks.

Table 4
List of propagation benchmarks.

Propagation benchmarks	
Name	Short description
Euler-Bernoulli beam	Modeling the effect of uncertain material parameters on the displacement of an Euler-Bernoulli beam with a prescribed load.
Genz	Multi-dimensional functions for testing quadrature and surrogate methods.
L2-Sea UQ	Forward UQ of the total resistance in calm water conditional to operational and geometrical uncertain parameters.
Cookies problem	Forward UQ of average temperature in a corner of the “oven”.

Table 5
List of optimization benchmarks.

Optimization benchmarks	
Name	Short description
L2-Sea OPT	Constraint deterministic global optimization problem for the total resistance reduction in calm water at fixed speed.

4.4. *Optimization benchmarks*

UM-Bridge’s interface is fundamentally agnostic to what algorithms do with what the interface provides. This makes it possible to also use UM-Bridge for other fields like black-box optimization. We have included an initial example of an optimization benchmark in Table 5 and Appendix A.4.

5. Applications

The following sections present some applications of uncertainty quantification to real-world problems that utilize UM-Bridge and demonstrate at the same time aspects of the work presented in the main part of the paper. Specifically, Section 5.1 demonstrates programming language independence by connecting an existing Matlab UQ code to a number of model instances written in Fortran. Section 5.2 demonstrates parallel runs of a highly complex numerical solver in a UQ application, greatly benefiting from portable models and separation of concerns between model and UQ experts. Finally, Section 5.3 demonstrates a complex UQ application combining fast model approximations running on a workstation and transparently offloaded model runs on a cloud cluster of 2,800 processor cores. Models and UQ problems used in Section 5.1 and Section 5.3 are available as part of the benchmark library.

5.1. *Accelerating a naval engineering application*

5.1.1. *Problem description*

In this section we focus on a forward UQ application in the context of naval engineering, which we publish as part of the benchmark library (see Appendix A.1.2 for more details). The goal is to compute the PDF of the resistance R to advancement of a boat, i.e., the naval equivalent of the drag force for airplanes. The boat is advancing in calm water, under uncertain Froude number Fr (a dimensionless number proportional to the navigation speed) and draft T (immersed portion of the hull, directly proportional to the payload). That is, the set of uncertain inputs is $\theta = (Fr, T)$, which are modeled as follows: Froude is a unimodal triangular random variable with support over $[Fr_{\mathrm{a}}, Fr_{\mathrm{b}}] = [0.25, 0.41]$, i.e.,

$$\pi_{Fr}(t) = \frac{2}{(Fr_{\mathrm{b}} - Fr_{\mathrm{a}})^2}\left(Fr_{\mathrm{b}} - t\right), \tag{1}$$

while draft is a beta random variable with support over $[T_{\mathrm{a}}, T_{\mathrm{b}}] = [-6.776, -5.544]$ and shape parameters $\alpha = 10, \beta = 10$, i.e.,

$$\pi_T(t) = \frac{\Gamma(\alpha+\beta+2)}{\Gamma(\alpha+1)\Gamma(\beta+1)} \times \dots \\ (T_{\mathrm{b}} - T_{\mathrm{a}})^{\alpha+\beta+1}(t - T_{\mathrm{a}})^{\alpha}(T_{\mathrm{b}} - t)^{\beta}. \tag{2}$$

The computation of R for fixed Fr, T, – i.e., the evaluation of the response function $R = R(\theta)$ – is performed by the L2-Sea model [20] which is written in Fortran and is available as a container from the UM-Bridge benchmark library, having being wrapped in an UM-Bridge Python server.

In order to showcase the ease of parallelizing an existing application by using UM-Bridge, we will show the required command and code snippets in this section. This model can be run locally via the following docker command:

```
docker run -it -p 4242:4242 linusseelinger/model-l2-sea
```

5.1.2. *UQ workflow*

To compute the probability density function of R we proceed in two steps:

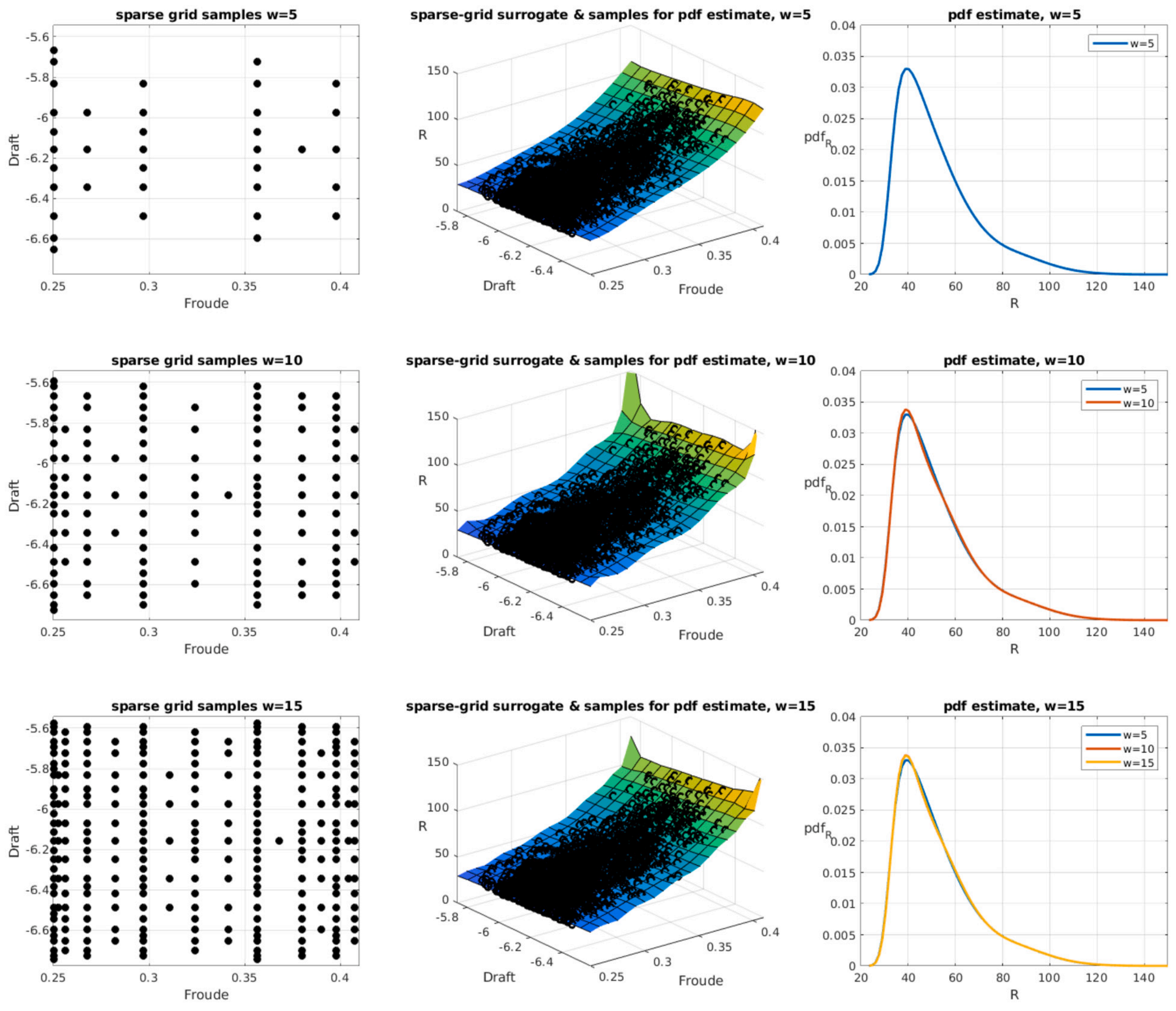

Fig. 6. Results for the UQ workflow on the L2-Sea model for three sparse-grid levels w (one row for each w). w controls the number of evaluation points in a sparse grid. Left column: Sparse grids covering the space of possible values of $\theta = [Fr, T]$. Center column: Sparse-grid surrogate models and their evaluations at the random sample points used for computing the PDF of R. Right column: Resulting PDF of R for all w up to the current one.

1. We create a surrogate model for the response function $R = R(\theta)$, i.e., an approximation of the actual function based on a limited number of (judiciously chosen) evaluations of $R = R(\theta)$.
2. We generate a large sample of values $\theta_i = (Fr_i, T_i)$ according to their PDF. We then evaluate the surrogate model for each element of the sample (which is considerably cheaper than evaluating the full model $R(\theta)$, but yields only approximate results), and then use the corresponding values R_i to compute an approximation of the PDF of R by a standard kernel density method [38] (specifically, in this application we use Gaussian kernels and automatic selection of bandwidth, [39]).

To build the surrogate model, we use the sparse-grids method, and in particular the implementation provided by SGMK. This method requires sampling the space of feasible values of $\theta = (Fr, T)$ with a structured non-Cartesian strategy, depending on the PDF of the two uncertain parameters, see the left column of Fig. 6. The surrogate model is constructed as a sum of certain interpolating polynomials, each based on a subset of samples. The surrogate models are reported in the center column Fig. 6. The overshoots in the corners of the domain happen in regions of the parameter space of zero probability; neither full model evaluations nor samples to approximate the PDF of R are placed there.

SGMK provides ready-to-use functions to generate sparse grids according to several types of random variables (uniform, normal, exponential, beta, gamma, triangular); once the grid is created, it is a simple matter of looping through its points and calling the L2-Sea solver for each one using the point coordinates as values for the inputs θ. In SGMK this is as easy as the following snippet:

```
uri  = 'http://104.199.68.148';
model = HTTPModel(uri,'benchmark_UQ');
```

```
fid = 3;
config = struct('fidelity',fid);

R = @(theta) model.evaluate(theta(:)',config);

Fr_a=0.25; Fr_b=0.41;
knots_Fr = @(n) knots_triangular_leja(n,Fr_a,Fr_b);
lev2knots_Fr = @lev2knots_lin;

T_a=-6.776; T_b=-5.544; alpha=10; beta=10;
knots_T = @(n) knots_beta_leja(n,alpha,beta,T_a,T_b,'sym_line','on_file');
lev2knots_T = @lev2knots_2step;

N=2; w=5;
S = create_sparse_grid(N,w,{knots_Fr,knots_T},{lev2knots_Fr,lev2knots_T});
Sr = reduce_sparse_grid(S);

R_values = evaluate_on_sparse_grid(f,Sr,...);
```

Once we have obtained a characterization of the surrogate model in the form of `R_values`, we generate a random sample of values of $\theta_i = (Fr_i, T_i)$ according to their PDF by rejection sampling [40], and evaluate the surrogate model on such values with a straightforward call to a SGMK function:

```
surr_evals=interpolate_on_sparse_grid(S,Sr,R_values,random_sample);
```

These evaluations are shown as black dots in the center column of Fig. 6. These surrogate model evaluations are used as input to the `ksdensity` estimator provided by Matlab to compute an approximation of the PDF of R.

Results are reported in the right column of Fig. 6, and as expected, the estimated PDF stabilizes as we add more points to the sparse grid and the surrogate model becomes more reliable. The three rows are obtained repeating the snippets above three times, increasing the so-called *sparse grid level* `w`, which is an integer value that controls how many points are to be used in the sparse grid procedure, specifically $w = 5, 10, 15$, corresponding to $36, 121, 256$ points. Note that the three sparse grids produced are nested, i.e., they are a sub-set of one another, so that in total only 256 calls to L2-Sea are needed. SGMK is able to take advantage of this and only evaluate at the *new* sparse-grid points in each grid level.

5.1.3. Computational aspects

In order to allow parallel model evaluations, 48 instances of the model were run on `c2d-high-cpu` nodes of GKE using the kubernetes configuration in Section 3.4. For the test case presented here, one evaluation of the L2-Sea model takes about 30-35 s on a single physical CPU core. We disable Simultaneous Multithreading (SMT) on GKE to ensure stability in L2-Sea run times.

On the UQ side, SGMK was run on a regular laptop, connecting to the cluster. Due to UM-Bridge, the SGMK Matlab code can directly be coupled to the Fortran model code. Further, no modification to SGMK is necessary in order to issue parallel evaluation requests: The call `R_values = evaluate_on_sparse_grid(f,Sr,...)` in SGMK is internally already using Matlab's `parfor` in order to loop over parameters to be evaluated. By opening a parallel session in Matlab with 48 workers (via the `parpool` command), the very same code executes the requests in parallel, and the cluster transparently distributes requests across model instances.

Using this approach, we obtained a total run time of around 290 seconds instead of the sequentially expected $30 \times 256 = 7680$ seconds, corresponding to a speedup of more than 26. Three factors affect the speedup: The evaluations are split in three batches, none of the batches having cardinality equal to a multiple of the 48 parallel model instances; the CPU-time for evaluating the L2-Sea model at different values of θ_i is not identical; and finally a minor overhead in the UQ method itself.

Apart from the limited number of evaluations needed for the specific application, the main limit in scalability we observe here is Matlab's memory consumption: approximately 500 MB allocated per worker, regardless of the actual workload. This could be improved by sending batches of requests per worker, but not without modifying the SGMK code.

5.2. Investigating material defects in composite aero-structures

5.2.1. Problem description

In this section we examine the effect of random material defects on structural aerospace components made out of composite laminated material. These may span several meters in length, but are still governed by a mechanical response to processes on smaller, sub-millimeter length scales. Such a multi-scale material displays an intricate link between meso-scale (ply scale) and macro-scale behavior (global geometric features) that cannot be captured with a model describing only the macro-scale response. This link between material scales is even stronger as abnormalities or defects, such as wrinkles (out-of-plane fiber waviness) Fig. 7 (c), may occur during the fabrication process. To simulate such objects, we use a multi-scale spectral generalized finite element method (MS-GFEM) [41]. The approach constructs a coarse approximation space (i.e., a reduced order model) of the full-scale underlying composite problem by solving local generalized eigenproblems in an A-harmonic subspace.

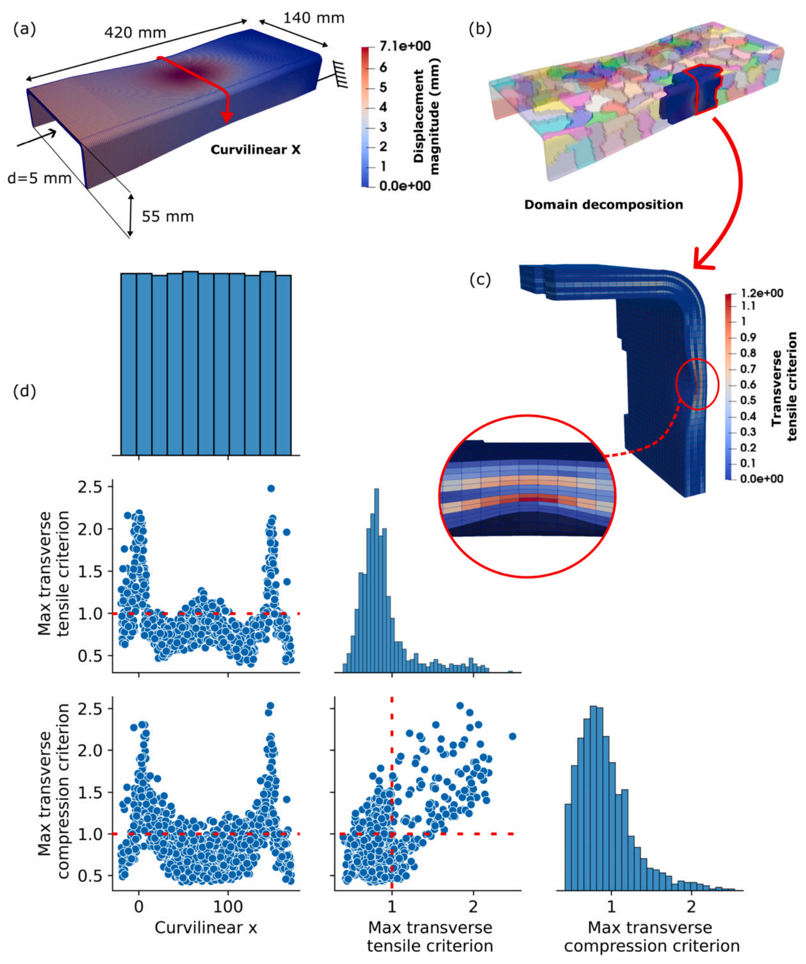

Fig. 7. Offline-Online framework and results: Geometric description at macro-scale and displacement field of the pristine (offline) model (a). The domain decomposition and the targeted subdomains for a set of defect parameters (b) show the fraction of the domain that needs to be recomputed online. The update of the solution for a wrinkle defect (c) shows the local variation of the transverse tensile failure criterion at the defect location. The pairwise relationships of the defect location along the curvilinear axis X (model input) (depicted on (a)) and two damage criteria (model output): the transverse compressive and tensile failure criteria are plotted as a scatter plots (off-diagonal) and histograms (diagonal) in (d).

By operating on subdomains, this MS-GFEM approach allows for reusing the solutions of local problems in places not affected by localized changes (such as a localized defect in an aerospace part). To exploit this structure, we use an offline-online framework in which the MS-GFEM is applied to a pristine model offline (see Fig. 7(a)), and only the eigenproblems in subdomains intersecting local defects are recomputed online to update the approximation space (see Fig. 7(b)). Online recomputations can be performed by a single processor.

For this application, we simulate the behavior of an aerospace component (namely a C-shaped laminated composite spar, see Fig. 7) under compression. To model the wrinkling defect, a geometric transformation is applied to the grid as depicted on Fig. 7(c). To assess the severity of the defect, we use two damage criteria, representing the ability of the ply to fail in the direction transverse to the fiber orientation in tension and compression (see Fig. 7(d)). The full-scale model consists of 1.4 million degrees of freedom. The MS-GFEM approximation reduces it to 12, 819, which corresponds to a model order reduction factor of around 106. The method is implemented in C++ and part of dune-composites, a Distributed and Unified Numerics Environment (DUNE) [42] module.

5.2.2. UQ workflow

We define a forward UQ problem by assuming a random defect parameter with a uniform distribution $\theta \sim \mathcal{U}([-20, 171] \times [150, 270] \times [0, 90] \times [3.5, 4.5])$. The first two components are the position of the defect along the curvilinear axes of the width and length of the part, the third is the orientation of the wrinkle, and the fourth is the size of the wrinkle. Position and size values are given in mm and the angle in degrees.

To solve the UQ problem, we employ the Quasi-Monte Carlo method implemented in QMCPy [14], drawing 1024 samples from QMCPy's `DigitalNetB2` generator. We observed that the main influence on the defect is the position along X (maxima located at the corners of the C section) for both criterion, as shown in Fig. 7(d). We consider the material to be capable of initiating failure mechanisms under load when any criterion reaches 1 (the threshold is represented as a dashed red line in the figure). We observed a different distribution for both criterion with a probability of failure of 20.41% for the tensile criterion and 32.42% for the compressive criterion. This is due to the complex interaction between the global geometric feature and the local material behavior (the laminated stacking sequence is shown on the right-hand side of Fig. 2). This interaction is not trivial and cannot be anticipated by looking only at the pristine output fields. This methodology is ideal and necessary to conduct an efficient parametric study of defects in large composite parts.

5.2.3. Computational aspects

We conducted the numerical experiment by running the model container on a 12-core workstation, which was outfitted with a simple k3s installation in order to support the kubernetes setup of Section 3.4. QMCPy was run on the same system, with QMCPy's UM-Bridge integration set to 12 parallel evaluations. QMCPy's UM-Bridge integration then issues parallel requests through Python's `multiprocessing` framework, which the kubernetes setup in turn processes.

Online model evaluations on a single core average at around 57 minutes of run time, with most runs between 30 and 90 minutes. The main reason for this variability is that the number of subdomains that need to be recomputed varies depending on defect location. We observe a total run time of 81.2 hours for 1024 Quasi-Monte Carlo (QMC) samples on the 12-core workstation.

The preparatory offline run computing the low-dimensional basis on the entire domain was conducted on the Hamilton HPC Service of Durham University, UK. On 128 processor cores it took 8 min 04 s. Compared to computing such a full MS-GFEM solution each time, the online method contributes a speedup of roughly 18 in addition to the UQ method's parallelization.

Containerization turned out crucial in this application: The MS-GFEM method is complex to set up and multiple instances running on the same system might lead to conflicts, which is alleviated by containerization. In addition, sharing model containers between collaborators greatly accelerated development.

5.3. Multilevel Delayed Acceptance for tsunami source inversion

5.3.1. Problem description

We solve a Bayesian inference problem based on a tsunami model, which we make available in the benchmark library (see Appendix A.2.9). It models the propagation of the 2011 Tohoku tsunami by solving the shallow water equations with wetting and drying. For the numerical solution of the PDE, we apply an ADER-DG method implemented in the ExaHyPE framework [43,44]. Details on the model and its discretization can be found in [45], and include a model with a smoothed bathymetry data incorporating wetting and drying through the use of a finite volume subcell limiter with polynomial order 2 and a total of $1.7 \cdot 10^5$ degrees of freedom (2187 spatial); and a model using fully resolved bathymetry data and a limited ADER DG scheme of polynomial degree 2 with a total of $1.7 \cdot 10^7$ degrees of freedom (6561 spatial). Bathymetry data was obtained from GEBCO https://www.gebco.net/data_and_products/gridded_bathymetry_data/.

5.3.2. UQ workflow

The aim is to obtain information about the parameters describing the initial displacements in bathymetry leading to the tsunami from the data of two DART buoys located near the Japanese coast. (The data for DART buoys 21418 and 21419 was obtained from NDBC https://www.ndbc.noaa.gov/.) The posterior distribution of model inputs, i.e. the source location of the tsunami, was sampled with the Multilevel Delayed Acceptance (MLDA) MCMC algorithm [27,30] using the open-source Python package tinyDA [16]. MLDA is a scalable MCMC algorithm, which can broadly be understood as a hybrid of Delayed Acceptance (DA) MCMC [46] and Multilevel MCMC [47]. The MLDA algorithm works by recursively applying DA to a model hierarchy of arbitrary depth, see [30].

We used a three-level model hierarchy consisting of the fully resolved model on the finest level, the smoothed model on the intermediate level and a Gaussian Process (GP) emulator on the coarsest level. A GP is a probabilistic model which provides both the predictive mean and variance, the latter indicating the credibility of the prediction. Both the mean and the variance were leveraged for computing the likelihood of the GP prediction as outlined in e.g. [48]. This resulted in a state-dependent tempering of the likelihood functional on the coarsest level, which was informed by the predictive uncertainty of the GP.

The GP emulator was trained on 1024 low-discrepancy samples of the model response from the smoothed model. The GP emulator employed a constant mean function, a Matérn-$\frac{5}{2}$ covariance function with Automatic Relevance Determination (ARD) and a noise-free Gaussian likelihood. The GP hyperparameters were optimized using Type-II Maximum Likelihood Estimation [49].

5.3.3. Computational aspects

The model was run using the kubernetes setup of Section 3.4 on GKE with 100 `c2d-highcpu-56` nodes running 100 instances of the model container. In total, 2800 physical CPU cores were used. Each model instance was run on one `c2d-highcpu-56` node exploiting the Intel TBB parallelization capabilities of ExaHyPE [43].

Parallelization of the UQ method was achieved through 100 independent MLDA samplers. tinyDA allows running multiple samplers by internally making use of Ray [50], a Python library for distributed computing. The UQ software was run on a workstation, connecting to the cluster via tinyDA's UM-Bridge integration. Gaussian process evaluations were conducted directly on the workstation due to their low computational cost, while model evaluations were offloaded to the cluster.

Overall run time was 2 h 20 min, with very effective use of the cluster as indicated by its CPU utilization in Fig. 8. For parameter $(0, 0)$, the smoothed model takes 1m1s to evaluate, while the fully resolved model takes 15m4s. We computed a total of 800 evaluations of the fully resolved model and 1400 of the smoothed model. Disregarding the additional inexpensive GP evaluations, we obtain a parallel speedup of 96.38, giving a close to perfect speedup for 100 MLDA chains.

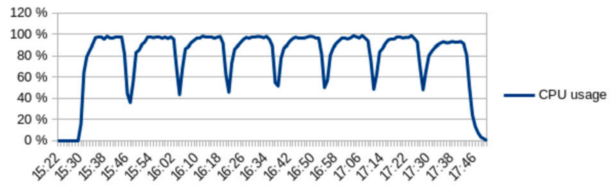

Fig. 8. CPU usage on GKE kubernetes cluster during MLDA run for the tsunami model. Intermittent drops are caused by runs of the intermediate model, which require fewer resources.

Table 6
Exemplar list of UQ software, describing which type of connection between UQ and model is supported (multiplicity): one particular UQ package to one model (one-to-one), one particular UQ package to M different models (one-to-many), or N UQ codes to M models (many-to many). Further, the table describes whether the model can be implemented in arbitrary languages (model language agnostic), how the interface is realized, e.g. via reading/writing files or in software (technology) and which forms of HPC support the package provides (scalability).

Package	Link			Scalability
	Multiplicity	Model language agnostic	Technology	
hIPPYlib-MUQ [51]	one-to-one	No	C++, Python	Single node
MUQ [12]	one-to-many	No	C++, Python	HPC (MPI)
PyMC [7], QMCPy [14]	one-to-many	No	Python	Single node
Dakota [52]	one-to-many	Yes	C++, Files	HPC (SLURM)
lagun [53]	one-to-many	Yes	Files	HPC (SLURM)
UQLab [54]	one-to-many	Yes	Matlab, files	HPC (SLURM)
UM-Bridge [24]	many-to-many	Yes	C++, Julia, Matlab, Python, R	HPC (SLURM, PBS), Kubernetes

6. Discussion

UM-Bridge fills a major gap in the UQ software ecosystem: There is currently no common platform linking arbitrary UQ algorithms and model software, let alone general-purpose cloud and HPC support or language-independent benchmarks.

In practice, most publications that couple advanced UQ methods and complex models do so in one of two ways: Either they directly embed the model in the UQ code, which requires compatible programming languages and entails unwieldy software stacks. Or, the UQ component passes parameters to the modeling component via some kind of input file, executes the model software, and then processes the outcomes.

Both approaches do not lend themselves to containerized models, and the parallelization infrastructure for HPC support remains up to each individual UQ code. Since the interfaces are incompatible across UQ packages, they lead to user lock-in: It is impractical for models to implement interfaces to multiple UQ packages and it is often difficult to translate data and data structures between programming languages. All of this further precludes systematic comparisons between UQ methods, which are often only demonstrated on trivial test cases (say a superposition of Gaussians) and a single complex model that is inaccessible to the broader community.

UM-Bridge addresses those problems by providing a uniform and language-independent set of interfaces. As a consequence, it is able to couple N UQ codes and M model applications using only $N + M$ implementations of the coupled interface, rather than the $N \cdot M$ modifications otherwise necessary. In addition, UM-Bridge provides generic parallelization platforms that serve *any* UQ code and *any* model, and it addresses the issue of different languages and complex, perhaps conflicting dependencies, build systems, or parallelization schemes. Table 6 gives an overview of how commonly used UQ packages realize the interface between UQ and model and how that compares to UM-Bridge.

In building UM-Bridge, we were inspired by projects in other domains that aim at coupling with modeling software. For example, the Functional Mockup Interface (FMI) [55], prominent in commercial codes like ANSYS and Matlab, standardizes this process by encapsulating models as Functional Mockup Units (FMUs), promoting interoperability between different simulation tools. As another example, preCICE (Precise Code Interaction Coupling Environment) [56] facilitates multi-physics simulations by coupling grid interfaces between different specialized simulation codes through the preCICE interface.

In the end, a UQ project's success hinges on whether its benefits can be demonstrated using practical examples of relevance to the community. To this end, we have built the UM-Bridge benchmarks library that provides a ready made set of benchmarks that can be used to test the performance of new methods, and which can be used as a reference for comparison with existing methods. In other areas such as numerical linear algebra [57], geophysics [58–60], HPC [61], model order reduction [62], etc., such benchmark collections are widely used. To the best of our knowledge no such collection currently exists for UQ.

Beyond its initial focus on UQ, the protocol can adapt to the needs of other fields through extensions. Ongoing work, e.g., adds an optional "fast path" through shared memory whenever client and server run on the same machine, enabling transfer of large amounts of data like entire random fields.

Limitations Algorithms requiring information beyond the current UM-Bridge interface can not be integrated. Also, for large data transfers or extremely fast models, the network overhead may become significant. Both limitations are somewhat mitigated by support for future protocol extensions. At the time of writing, smooth handling of model code crashes is planned but not yet supported.

7. Outlook

We aim to establish UM-Bridge across the UQ community, serving as a catalyst for highly efficient UQ methods applied to challenging problems, and acting as a link between UQ and applications as well as between academia and industry. With growing computational resources and the advance of science, we expect growing complexity in UQ applications, making the need for separation of concerns and advanced UQ tools even more prominent.

CRediT authorship contribution statement

Linus Seelinger: Writing – original draft, Software, Investigation, Conceptualization. **Anne Reinarz:** Writing – original draft, Software, Investigation, Conceptualization. **Mikkel B. Lykkegaard:** Writing – original draft, Software, Investigation, Conceptualization. **Robert Akers:** Resources, Software. **Amal M.A. Alghamdi:** Writing – original draft, Software, Investigation. **David Aristoff:** Writing – original draft, Software, Investigation. **Wolfgang Bangerth:** Writing – original draft, Software, Investigation. **Jean Bénézech:** Writing – original draft, Software, Investigation. **Matteo Diez:** Writing – original draft, Software, Investigation. **Kurt Frey:** Writing – original draft, Software, Investigation. **John D. Jakeman:** Writing – original draft, Software, Investigation. **Jakob S. Jørgensen:** Writing – original draft, Software, Investigation. **Ki-Tae Kim:** Writing – original draft, Software, Investigation. **Benjamin M. Kent:** Investigation, Writing – review & editing. **Massimiliano Martinelli:** Writing – original draft, Software, Investigation. **Matthew Parno:** Writing – original draft, Software, Investigation. **Riccardo Pellegrini:** Writing – original draft, Software, Investigation. **Noemi Petra:** Writing – original draft, Software, Investigation. **Nicolai A.B. Riis:** Writing – original draft, Software, Investigation. **Katherine Rosenfeld:** Writing – original draft, Software, Investigation. **Andrea Serani:** Writing – original draft, Software, Investigation. **Lorenzo Tamellini:** Writing – original draft, Software, Investigation. **Umberto Villa:** Writing – original draft, Software, Investigation. **Tim J. Dodwell:** Conceptualization. **Robert Scheichl:** Writing – original draft, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

D. Aristoff gratefully acknowledges support from the National Science Foundation via awards DMS-1818726 and DMS-2111277.

W. Bangerth was partially supported by the National Science Foundation under award OAC-1835673 as part of the Cyberinfrastructure for Sustained Scientific Innovation (CSSI) program; by award DMS-1821210; and by award EAR-1925595.

J. Bénézech research was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through the Programme Grant EP/S017038/1 “Certification of Design: Reshaping the Testing Pyramid” and made use of the Hamilton HPC Service of Durham University. The code is developed and integrated in the Distributed and Unified Numerics Environment (DUNE) https://www.dune-project.org/.

J. D. Jakeman was supported by the US Department of Energy’s Office of Advanced Scientific Computing Research program. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan.

A. Reinarz and L. Seelinger gratefully acknowledge support by Cristian Mezzanotte (Google). This material is based upon work supported by the Google Cloud Research Credits program with the award GCP211124249.

The work of N. A. B. Riis, J. S. Jørgensen and A.M.A. Alghamdi was supported by The Villum Foundation (grant no. 25893).

R. Scheichl and L. Seelinger gratefully acknowledge support by the state of Baden-Württemberg through bwHPC, as well as the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, as well as its Excellence Strategy EXC 2181/1 - 390900948 (the Heidelberg STRUCTURES Excellence Cluster).

L. Tamellini has been partially supported by the project 202222PACR “Numerical approximation of uncertainty quantification problems for PDEs by multi-fidelity methods (UQ-FLY)”, funded by European Union—NextGenerationEU. Lorenzo Tamellini and Massimiliano Martinelli are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM), and have been partially supported by the ICSC—Centro Nazionale di Ricerca in High Performance Computing, Big Data, and Quantum Computing, funded by European Union—NextGenerationEU.

M. Diez, R. Pellegrini, and A. Serani are partially supported by The Office of Naval Research through NICOP grant N62909-21-1-2042, administered by Woei-Min Lin, Elena McCarthy, and Salahuddin Ahmed of the Office of Naval Research and Office of Naval Research Global, and their work has been conducted in collaboration with the NATO task group AVT-331 on "Goal-driven, multi-fidelity approaches for military vehicle system-level design".

Appendix A. Benchmark library

This appendix documents all models and benchmark problems currently part of the UM-Bridge benchmark library, using the three categories defined in Section 4.

A.1. Models

In this section we describe the models contained in the benchmark library, these implement a numerical (forward) model of a physical phenomenon. Each model is accompanied by a markdown file that documents how to run the published container, input and output dimensions, configuration options, and a mathematical description of the map (or a reference to an existing publication).

A.1.1. Euler-Bernoulli beam

This model describes the deformation of a beam with a spatially variable stiffness parameter. Let $u(x)$ denote the vertical deflection of the beam and $f(x)$ denote the vertical force acting on the beam at point x (positive for upwards, negative for downwards). We assume that the displacement can be well approximated using Euler-Bernoulli beam theory and thus satisfies the fourth-order PDE

$$\frac{\partial^2}{\partial x^2}\left[rE(x)\frac{\partial^2 u}{\partial x^2}\right] = f(x), \tag{A.1}$$

where $E(x)$ is an effective stiffness and r is the beam radius. For a beam of length L, the cantilever boundary conditions take the form

$$u(x=0)=0, \quad \left.\frac{\partial u}{\partial x}\right|_{x=0} = 0$$

and

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=L} = 0, \quad \left.\frac{\partial^3 u}{\partial x^3}\right|_{x=L} = 0.$$

This PDE is solved with a finite difference method and the beam stiffness is defined at each of the (typically $N = 31$) nodes in the discretization. The displacement of the beam is also returned at these N points. The beam radius is set to $r = 0.1$ and the value of $f(x)$ is fixed.

This model takes in $E(x)$ at N finite difference nodes and returns the value of $u(x)$ at those nodes.

A.1.2. L2-Sea model

This model describes the calm-water resistance R of a destroyer-type vessel by potential flow. Specifically, the vessel under investigation is the DTMB 5415 (at model scale), which is a widely used benchmark for towing tank experiments [63], CFD studies [64], and hull-form optimization [65], considering both deterministic [66] and stochastic [67] formulations.

A potential flow solver is used to evaluate the hydrodynamic loads, based on the Laplace equation

$$\nabla^2\phi = 0 \tag{A.2}$$

where ϕ is the velocity scalar potential, from which the velocity is computed through $\mathbf{u} = \nabla\phi$. ϕ is computed numerically through the Dawson linearization [68] of the potential flow equations, using the boundary element method [69] (see Fig. 2). Finally, the total resistance R is estimated as the sum of the wave and the frictional resistance: The wave resistance component is estimated by integrating the pressure distribution over the hull surface, obtained using Bernoulli's theorem

$$\frac{p}{\rho} + \frac{(\nabla\phi)^2}{2} - gz = const; \tag{A.3}$$

the frictional resistance component is estimated using a flat-plate approximation based on the local Reynolds number [70].

The steady equilibrium involving two degrees of freedom (sinkage and trim) is computed iteratively through the coupling between the hydrodynamic loads and the rigid-body equation of motion.

The model takes as inputs 16 parameters, i.e., $\theta \in \mathbb{R}^{16}$: the Froude number Fr, the ship draft T (we have already introduced these quantities in Section 5.1), and $N = 14$ shape modification parameters $\mathbf{s} = [s_1, \dots, s_{14}]$ that change the shape of the hull of the vessel, i.e., $\theta = (Fr, T, \mathbf{s})$. More specifically, the shape of the hull is described by a function $\mathbf{g}$, depending on the Cartesian coordinates ξ and on the shape parameters $\mathbf{s}$ as follows:

$$\mathbf{g}(\xi, \mathbf{s}) = \mathbf{g}_0(\xi) + \gamma(\xi, \mathbf{s}) \tag{A.4}$$

where $\mathbf{g}_0$ is the original geometry and $\boldsymbol{\gamma}$ is a shape modification vector obtained by a physics-informed design-space dimensionality reduction [71]

$$\boldsymbol{\gamma}(\boldsymbol{\xi},\mathbf{s}) = \sum_{k=1}^{N} s_k \boldsymbol{\psi}_k(\boldsymbol{\xi})$$

with $\boldsymbol{\psi}$ a set of orthonormal functions. Note that the shape parameters and the associated shape modifications are organized in a hierarchical order, meaning that the first parameters produce larger design modifications than the last ones [67].

We now discuss bounds for the parameters collected in θ. Concerning Fr, although in principle it can be any number between 0 (excluded) and the maximum velocity of the ship (corresponding to $Fr = 0.41$, in practice the mesh is suitable for values in the range $[0.25, 0.41]$. T must be negative (otherwise the ship would be above water), but not too much (or the ship would be completely under water); for meshing reasons again, the suitable range for T is $[-6.776, -5.544]$. Finally, the shape modification parameters s_i must be chosen in the range [-1,1].

Moreover, three configuration parameters can be set to control the behavior for this model: *fidelity* is an integer value that controls the mesh refinement, ranging from to 1 (highest resolution, default) to 7 (lowest resolution), with a refinement ratio of $2^{0.25}$, see [20] for more details; *sinkoff* is a character that can be set to `'y'` (default) / `'n'` to respectively disable/enable sinkage secondary movements (up/down translation on the vertical axis); similarly, *trimoff* can be set to `'y'` (default) / `'n'` to respectively disable/enable trim secondary movement (up/down rotation about its transverse axis, i.e. bow to stern).

The model returns the total resistance R as well as 4 geometrical quantities (related to the beam, draft, and sonar dome dimensions) whose value changes when playing with the shape parameters $\mathbf{s}$; we postpone their definition to Appendix A.4.1.

A.1.3. Tsunami model

This model describes the propagation of the 2011 Tohoku tsunami by solving the shallow water equations, see also Section 5.3. For the numerical solution of the PDE, we apply an ADER-DG method implemented in the ExaHyPE framework [43].

This benchmark creates a sequence of three models:

1. In the first model the bathymetry is approximated only by a depth average over the entire domain and the discretization is a DG discretization of order 2.
2. In the second model the bathymetry is smoothed using a Gaussian filter and the DG discretization is supplemented with a finite volume subcell limiter allowing for wetting and drying.
3. In the third model the full bathymetry data is used and the same DG discretization is used as in the second model.

The model takes two inputs giving the location of the displacement initiating the tsunami. The model outputs four values giving the maximal wave heights and time at two buoys located off the coast.

A.1.4. Composite material

This benchmark implements the 3D anisotropic linear elasticity equations for a composite part with randomized wrinkle. The maximum deflection of the part is estimated by embedding a wrinkle into a high fidelity finite element (FE) simulation using the high performance toolbox dune-composites [72]. A simplified model for a 3D bending test was used. The curved composite parts were modeled with shortened limbs of length 10 mm. A unit moment was applied to the end of one limb using a multi-point constraint, with homogeneous Dirichlet conditions applied at the end of the opposite limb. This gives the same stress field towards the apex of the curved section as a full 3D bending test. The analysis assumes standard anisotropic 3D linear elasticity and further details on the numerical model and discretization can be found in [72].

The wrinkle defect is defined by a deformation field $W : \Omega \to \mathbb{R}^3$ mapping a composite component from a pristine state to the defected state.

The wrinkles are defined by the wrinkle functions

$$W(x,\xi) = g_1(x_1) g_3(x_3) \sum_{i=1}^{N_w} a_i f_i(x_1,\lambda),$$

where $g_i(x_i)$ are decay functions, $f_i(x_1,\lambda)$ are the first N_w Karhunen–Loève (KL) modes parameterized by the length scale λ and a_i the amplitudes. The wrinkles are prismatic in x_2, i.e., the wrinkle function is assumed to have no x_2 dependency. For more details on the wrinkle representation see [73].

The amplitude modes and the length scale can be taken as random variables, so that the stochastic vector is defined by $\theta = [a_1, a_2, \ldots, a_{N_w}, \lambda]^T$.

The model outputs a single scalar value, the maximum deflection of the composite part under an applied pressure of 50 Atmospheres.

A.1.5. Tritium desorption

In this model, we consider macroscopic tritium transport processes through fusion materials using the Foster-McNabb equations, as described in [74,75] using the Achlys [76] software package, which is built on top of the FE library MOOSE [77]. Specifically, the following equations are solved:

$$\frac{\partial C_m}{\partial t} = \nabla \cdot \left(D(T) \nabla C_m \right) - \sum_i \frac{\partial C_{t,i}}{\partial t} + S_{\text{ext}},$$
$$\frac{\partial C_{t,i}}{\partial t} = \nu_m(T) C_m \left(n_i - C_{t,i} \right) - \nu_i(T) C_{t,i},$$
$$\rho_m C_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T).$$

Here, C_m represents the mobile concentration of Hydrogen Isotopes (HI), $C_{t,i}$ represents the trapped concentration of HI at the ith trap and T is the temperature of the material. Furthermore, the evolution of the extrinsic trap density n_3 is modeled as described in [75], taking into account additional trapping sites that are created as the material is damaged during implantation.

The simulation encompasses three distinct phases, namely an *implantation* phase, where the material is exposed to a particle source, a *resting* phase, and a *desorption* phase, where the material is heated and the desorption rate is computed.

The input parameters of the forward model consist of the detrapping energy of the HI traps, E_1, E_2, and E_3, and the densities of the intrinsic traps, n_1 and n_2. All other model parameters are kept fixed. The output of the model is the desorption rate in atomic fraction as a function of the temperature $T \in [300, 800]$ during the desorption phase, discretized on a grid of 500 interpolation points. This model supports the evaluation of the forward map.

A.1.6. Agent-based disease transmission model

This model simulates the transmission of disease in a heterogeneous population using EMOD [78], a stochastic agent based disease transmission model.

Our simulation consists of 100,000 individuals who are susceptible to a disease that is introduced into the population with a probabilistic rate of 1 infection per day for the first 5 days. This disease has a latent period that follows a Gaussian distribution with mean 3 days and standard deviation of 0.8 days. The infectious period P is assumed to also follow a Gaussian distribution with a mean of 8 days and a standard deviation of 0.8. The infectivity of the disease (I_t; how likely it is for an infectious individual to infect another) is assumed to follow a log-normal distribution and is determined by the simulation's R_0 and its variance: $\mu_{I_t} = \log\left(\frac{R_0}{\mu_P}\right) - 0.5\sigma_{I_t}^2$ where $\sigma_{I_t} = \log\left(\frac{\sigma_{R_0}^2}{2R_0^2} + 0.5\right)$.

The likelihood of an individual to acquire and then transmit the disease is correlated. There is no waning immunity. The benchmark is fitting to an attack fraction of 0.40 with a standard deviation of 0.025. There are bounds on $R_0 > 0$; the variance of $R_0 > 0$; and the correlation between acquisition and transmission must lie between 0 and 1 (inclusive).

The input parameters are the reproductive number, its logarithmic standard deviation, and correlation between acquisition and transmission. Optional configuration parameters are a flag to fix the simulation random seed, the number of infected individuals introduced into the simulation for the first 5 days, and the epsilon parameter utilized by scipy.optimize.approx_fprime (used to estimate the gradient). The simulation terminates when there are no longer any infected individuals (minimum run time of 50 days) and returns daily timeseries 'New Infections', 'Infected', 'Infectious Population', 'Susceptible Population', 'Symptomatic Population', 'Recovered Population', and 'Exposed Population'.

A.1.7. The membrane model

The "membrane model" describes the vertical deformation of a membrane for a fixed right hand side, as a function of the membrane's stiffness properties. This forward model is described in great detail in [21] as a building block for a Bayesian inverse problem.

More specifically, the model considers the vertical deflection $u(\mathbf{x})$ of a membrane given the spatially variable stiffness coefficient $a(\mathbf{x})$ for a known force $f(\mathbf{x}) = 10$. The model uses a specific, prescribed discretization of the following partial differential equation to obtain $u(\mathbf{x})$:

$$-\nabla \cdot [a(\mathbf{x}) \nabla u(\mathbf{x})] = f(\mathbf{x}),$$

augmented by zero boundary conditions. The forward model considers as inputs 64 values that define $a(\mathbf{x})$ on an 8×8 subdivision of the domain, computes the discretized deformation $u(\mathbf{x})$, and then computes an output vector of size 169 by evaluating the deflection $u(\mathbf{x})$ on a 13×13 grid of points.

The detailed description in [21] allows the implementation of this model in a number of programming languages. Indeed, [21] provides links to a C++ implementation based on the DEAL.II library [79] that is used for the containers described herein, as well as Matlab and Python implementations.

A.1.8. The cookies problem

This model implements the so-called "cookies problem" [80–82], i.e., a simplified thermal equation in which the conductivity coefficient is uncertain in 8 circular subdomains ("the cookies"), whereas it is known (and constant) in the remaining of the domain ("the oven"), see Fig. A.9. More specifically, the temperature solves the stationary thermal equation

$$-\nabla \cdot [a(\mathbf{x}, \theta) \nabla u(\mathbf{x}, \theta)] = f(\mathbf{x}),$$

with homogeneous Dirichlet boundary conditions and forcing term defined as

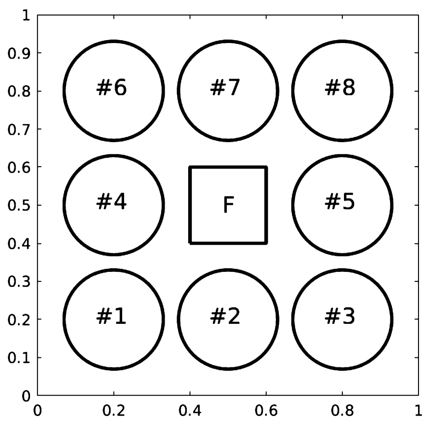

Fig. A.9. Computational domain for the cookies problem, Appendix A.1.8.

Table A.7
Centers of subdomains for the cookies problem, Appendix A.1.8.

Cookie	1	2	3	4	5	6	7	8
x	0.2	0.5	0.8	0.2	0.8	0.2	0.5	0.8
y	0.2	0.2	0.2	0.5	0.5	0.8	0.8	0.8

$$f(\mathbf{x}) = \begin{cases} 100 & \text{if } \mathbf{x} \in F \\ 0 & \text{otherwise} \end{cases}$$

where F is the square $[0.4, 0.6]^2$ in the center of the domain, again see Fig. A.9. The 8 subdomains with uncertain diffusion coefficient (the cookies) are circles with radius 0.13 and center coordinates reported in Table A.7. The uncertain diffusion coefficient is defined as

$$a = a_0 + \sum_{n=1}^{8} \theta_n \chi_n(\mathbf{x}) \tag{A.5}$$

where $a_0 = 1, \theta_n > -1$ and

$$\chi_n(\mathbf{x}) = \begin{cases} 1 & \text{inside the n-th cookie} \\ 0 & \text{otherwise.} \end{cases}$$

The output of the model is the integral of the solution over F, i.e.

$$\Psi(\theta) = \int_F u(\mathbf{x}, \theta) d\mathbf{x}. \tag{A.6}$$

The PDE is solved by a classical finite element method with standard Lagrangian polynomial bases on quadrilateral elements of degree p (tunable by the user, default $p = 1$). The mesh is composed of $N_{el} \times N_{el}$ elements, with $N_{el} = 100 \times k$ where k is an integer value ("fidelity") that can be set by the user (default $k = 4$). The solver is provided by a legacy version of the software FEniCs [83,84], via the Python interface. Further model configurations allow: changing the value of a_0, adding an advection term to the equation, changing the quadrature rule used to build the finite element matrix, or changing the solver of the linear system - see the UM-Bridge documentation for further details. Furthermore, the model provides a time-dependent version of the equation,

$$\frac{\partial u(\mathbf{x}, t, \theta)}{\partial t} - \nabla \cdot [a(\mathbf{x}, \theta) \nabla u(\mathbf{x}, t, \theta)] = f(\mathbf{x}),$$

with initial condition $u(\mathbf{x}, t, \theta) = 0$. The output of the model is the integral of the solution over F at final time T (default $T = 10$). All configuration options carry to the parabolic variant of the test case. The time-stepping is adaptive with local error control, specifically an implementation of the TR-AB2 scheme, see, e.g., [85].

A.2. Inference benchmarks

In this section we describe the inference benchmarks contained in the benchmark library, these implement a specific inverse UQ problem, usually in terms of a Bayesian posterior.

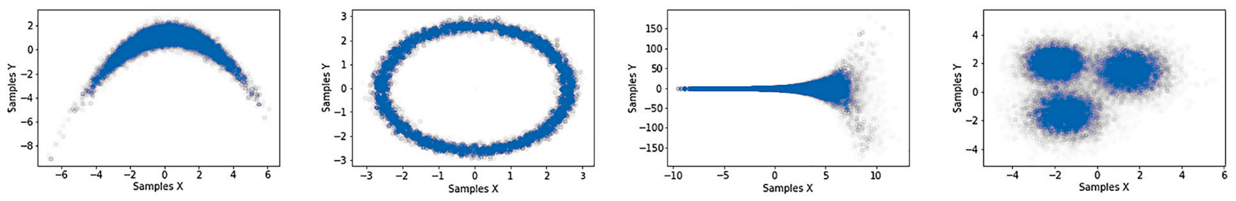

Fig. A.10. Samples drawn from the distribution defined through the PDF π for the analytic banana, donut, funnel and Gaussian mixture respectively.

Table A.8
Configuration parameters the for Analytic Banana benchmark.

Name	Type	Default	Description
a	double	2.0	Transformation parameter
b	double	0.2	Transformation parameter
scale	double	1.0	Scaling factor applied to the underlying normal distribution's variance

A.2.1. Analytic functions

The benchmarks in this series of four benchmarks consist of an analytically defined PDF $\pi : \mathbb{R}^2 \to \mathbb{R}$ resembling the shape of a banana, donut, funnel, and a Gaussian mixture, see Fig. A.10. They are based on transformed normal distributions; the variance may be adjusted for all benchmarks.

Analytic banana benchmark We begin with a normally distributed random variable

$$Z \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 4 \end{pmatrix}, \text{scale}\begin{pmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{pmatrix}\right),$$

and denote its PDF by f_Z.

In order to reshape the normal distribution, define a transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$

$$T(\theta) := \begin{pmatrix} \theta_1 / a \\ a\theta_2 + ab(\theta_1^2 + a^2) \end{pmatrix}.$$

Finally, the benchmark PDF is defined as

$$\pi(\theta) := f_Z(T(\theta)).$$

The parameters a, b, and 'scale' that appear in the formulas above are configurable, see Table A.8. This benchmark supports only evaluation of the forward map, but not its derivatives.

Analytic donut benchmark The PDF π is defined as

$$\pi(\theta) := -\frac{(\|\theta\| - r)^2}{\sigma^2},$$

where $r = 2.6$ and $\sigma^2 = 0.033$. This benchmark supports the 'Evaluate', 'Gradient', and 'ApplyJacobian' operations.

Analytic funnel benchmark First, define a helper function

$$f(\theta, m, s) := -\frac{1}{2}\log(2\pi) - \log(s) - \frac{1}{2}((\theta - m)/s)^2,$$

where here $\pi = 3.141\ldots$. The logarithm of the output PDF is then defined as

$$\log(\pi(\theta)) := f(\theta_1, 0, 3) + f\left(\theta_2, 0, \exp\left(\frac{1}{2}\theta_1\right)\right).$$

This benchmark supports the 'Evaluate', 'Gradient', and 'ApplyJacobian' operations.

Analytic Gaussian mixture benchmark Let

$$X_1 \sim \mathcal{N}\left(\begin{pmatrix} -1.5 \\ -1.5 \end{pmatrix}, 0.8I\right),$$

$$X_2 \sim \mathcal{N}\left(\begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}, 0.8I\right),$$

Table A.9
Prior distributions for the Tritium desorption model.

Parameter	Prior distribution
E_1	$\mathcal{U}(0.7, 1.0)$
E_2	$\mathcal{U}(0.9, 1.3)$
E_3	$\mathcal{U}(1.1, 1.75)$
n_1	$\mathcal{U}(5 \cdot 10^{-4}, 5 \cdot 10^{-3})$
n_2	$\mathcal{U}(10^{-4}, 10^{-3})$

$$X_3 \sim \mathcal{N}\left(\begin{pmatrix}-2\\2\end{pmatrix}, 0.5I\right).$$

Denote by $f_{X_1}, f_{X_2}, f_{X_3}$ the corresponding PDFs. The PDF π is then defined as

$$\pi(\theta) := \sum_{i=1}^{3} f_{X_i}(\theta).$$

This benchmark supports the 'Evaluate', 'Gradient', and 'ApplyJacobian' operations.

A.2.2. Membrane inverse benchmark

This benchmark for Bayesian inversion is, like the forward model in Section A.1.7, described in substantial detail in [21]. In short, whereas the forward model maps the 64 coefficients $\theta \in \mathbb{R}^{64}$ that describe $a(\mathbf{x}) = a^{\theta}(\mathbf{x})$ to a vector of measurements of $u(\mathbf{x})$ at 139 distinct locations, denoted by $z^{\theta} \in \mathbb{R}^{139}$, the inverse problem consists of inferring θ from a specific set of measurements $\hat{z}$. To this end, we define the probability that a given set of coefficients θ is correct as

$$\pi(\theta|\hat{z}) = L(\hat{z}|\theta)\pi_{\text{pr}}(\theta), \tag{A.7}$$

where the likelihood L and prior probability are defined as

$$L(z|\theta) = \exp\left(-\frac{\|z - z^{\theta}\|^2}{2\sigma^2}\right) \tag{A.8}$$

$$= \prod_{k=0}^{168} \exp\left(-\frac{(z_k - z_k^{\theta})^2}{2\sigma^2}\right), \tag{A.9}$$

$$\pi_{\text{pr}}(\theta) = \prod_{i=0}^{63} \exp\left(-\frac{(\ln(\theta_i) - \ln(1))^2}{2\sigma_{\text{pr}}^2}\right), \tag{A.10}$$

with $\sigma = 0.05, \sigma_{\text{pr}} = 2$, and the values of $\hat{z}$ as tabulated in [21]. The benchmark only supports evaluation of the forward map.

The description of the benchmark in [21] also contains very detailed statistics for mean and covariance matrix of the posterior distribution for these 64 values, along with estimates for the accuracy for these mean values. As a consequence, any inference scheme using the implementation of this benchmark provided through the containers described in Sections 3.1 can compare their results with those provided in [21]; this includes the workload-accuracy trade-off of inference algorithms for this benchmark.

A.2.3. Bayesian calibration of tritium desorption

This benchmark performs Bayesian model calibration of the tritium desorption model outlined in Section A.1.5. The benchmark defines a probability density $\pi : \mathbb{R}^5 \to \mathbb{R}$ where the inputs θ are E_1, E_2, E_3 (the detrapping energies used in the model description), and n_1, n_2 (the density of intrinsic traps).

The prior distributions over model parameters are all assumed to be uniform as shown in Table A.9. This prior distribution is then conditioned on the experimental data of [86], where the desorption rate of HI was measured for a tungsten lattice, to yield the posterior distribution $\pi(\theta|d)$. More formally, the (unnormalized) posterior distribution of parameters θ given data $d = \{\phi_i\}_{i=0}^{N_d-1}$ with $N_d = 14$ measurements of the desorption rate ϕ is computed using Bayes Theorem $\pi(\theta|d) \propto \mathcal{L}(d|\theta)\pi_{\text{pr}}(\theta)$. The measurement errors ε are assumed to follow a Gaussian distribution, so that

$$\mathcal{L}(d|\theta) \propto \exp\left(-\frac{1}{2\sigma_{\varepsilon}^2}\sum_{i=0}^{13}(\mathcal{F}_i(\theta) - d_i)^2\right),$$

with $\sigma_{\varepsilon} = 10^{17}$. Fig. A.11 shows $N = 1000$ posterior samples of the Tritium desorption curve along with the experimental data from [86].

A.2.4. Agent-based disease transmission benchmark

This benchmark uses EMOD [78], a stochastic agent-based disease transmission model, to look at how the reproductive number, R_0, and the correlation between an individual's acquisition and transmission correlation can affect the ultimate attack fraction and

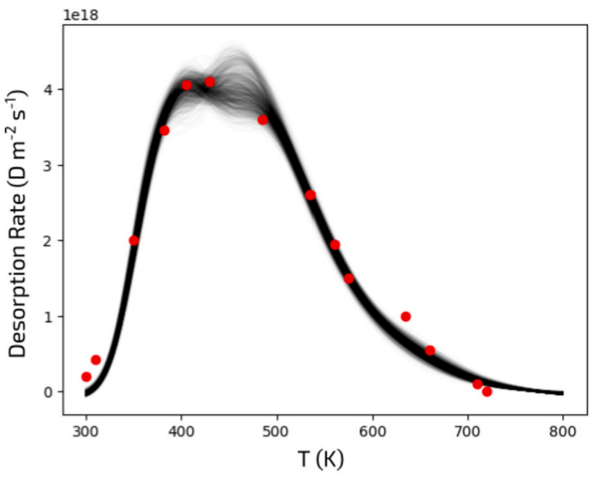

Fig. A.11. Posterior samples ($N = 1000$) of the Tritium desortion rate (black curves) and the experimental data from [86] (red dots). Both are plotted against the temperature T.

Table A.10
Configuration parameters for the disease model benchmark.

Config	Type	Default	Description
refresh_seed	int (bool)	1	Change random number seed
daily_import _pressure	double	1.0	Average daily importations for the first 5 simulated days
log_level	string	ERROR	Level of logging
epsilon	list	[0.001, 0.001, 0.001]	Increment used in scipy optimize approx_fprime to estimate the gradient

timing of the outbreak peak. This correlation can greatly affect the size and timing of the outbreak for a fixed reproductive number. The model underlying this benchmark is described in Section A.1.6.

The benchmark computes a probability distribution $\pi : \mathbb{R}^3 \to \mathbb{R}$ from three inputs that correspond to R_0, the variance of R_0, and the correlation between acquisition and transmission. The benchmark is fitting to an attack fraction of 0.40 with a standard deviation of 0.05 and an outbreak peak at 175 days from the start of the simulation with standard deviation of 10 days. The benchmark has additional configuration parameters, described in Table A.10. The benchmark only supports evaluation of the forward map.

One of the challenges of this benchmark is that the computation of $\pi(\theta)$ is not deterministic because it evaluates a stochastic model whose outputs (even for a fixed θ) depends on random numbers: Even for the same parameters the total number of infections will differ between simulations. Furthermore, it is not guaranteed that there will always be an outbreak: sometimes there are not enough initial infections to cause an outbreak. This challenge can be decreased by increasing the number of initial infections via the `daily_import_pressures` configuration parameter (e.g., from 1 infection per day to 10 infections per day for the first 5 days).

A.2.5. 1D deconvolution benchmark

This benchmark is based on a 1D deconvolution test problem from the library CUQIpy[1] [10,9]. It defines a posterior distribution $\pi : \mathbb{R}^{128} \to \mathbb{R}$ with a Gaussian likelihood and four different choices of prior distributions with configurable parameters. Supported features of this benchmark are: ‘Evaluate’ and ‘Gradient’.

The 1D periodic deconvolution problem is defined by the inverse problem

$$\mathbf{b} = \mathbf{A}\mathbf{x} + \mathbf{e},$$

where $\mathbf{b}$ is an m-dimensional random vector representing the observed data, $\mathbf{A}$ is an $m \times n$ matrix representing the convolution operator, $\mathbf{x}$ is an n-dimensional random vector representing the unknown signal we would like to recover, and $\mathbf{e}$ is an m-dimensional random vector representing the noise. The posterior distribution over $\mathbf{x}$ given $\mathbf{b}$ is defined as

$$\pi(\mathbf{x} \mid \mathbf{b}) \propto L(\mathbf{b} \mid \mathbf{x})\pi_{\mathrm{pr}}(\mathbf{x}),$$

where $L(\mathbf{b}|\mathbf{x})$ is a likelihood function and $\pi_{\mathrm{pr}}(\mathbf{x})$ is a prior distribution. The noise is assumed to be Gaussian with a known noise level, and thus the likelihood is defined via

[1] https://cuqi-dtu.github.io/CUQIpy/.

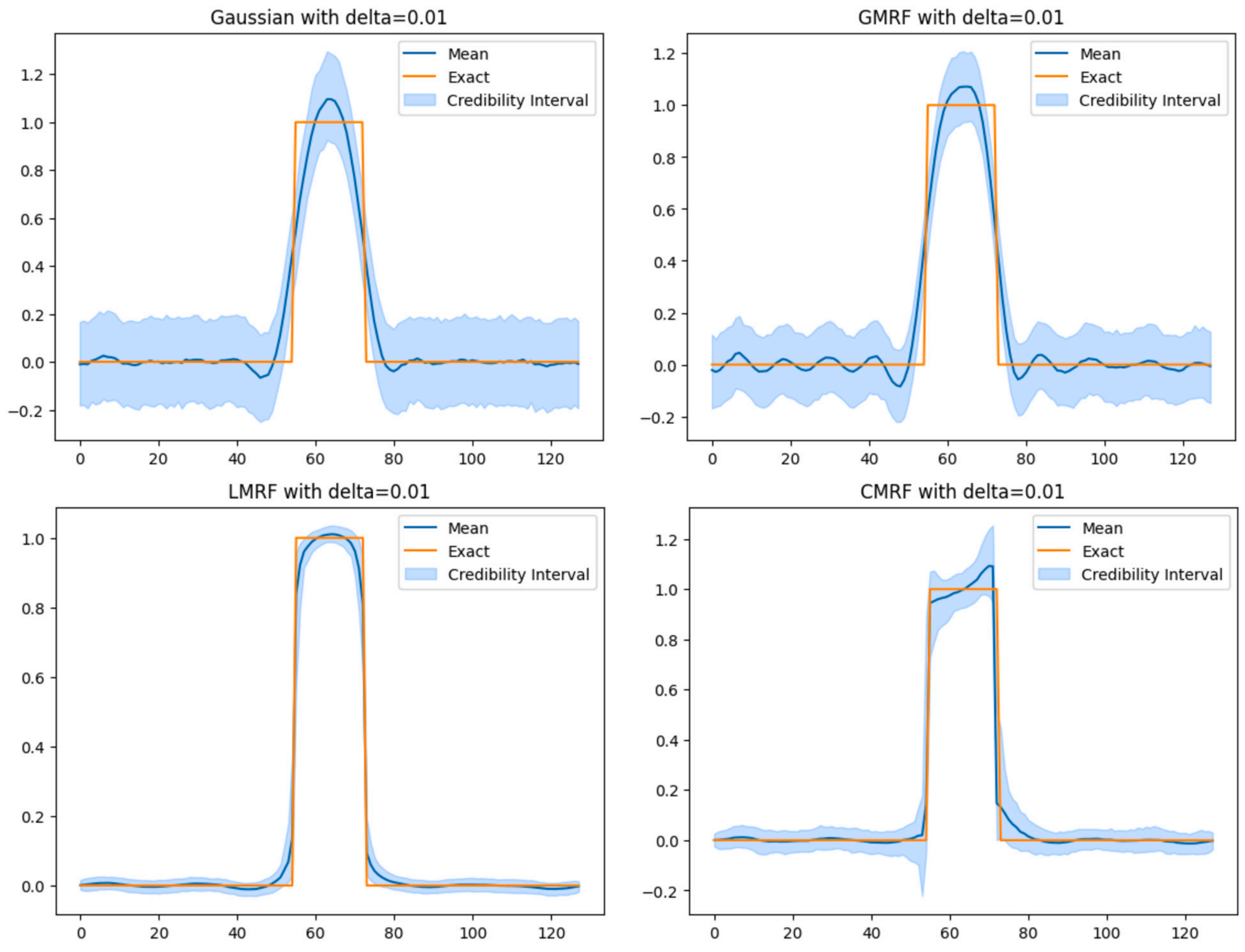

Fig. A.12. Sample mean and 95% credibility intervals of the posteriors defined for the deconvolution 1D benchmark.

$$\mathbf{b} \mid \mathbf{x} \sim \mathcal{N}(\mathbf{A}\mathbf{x}, \sigma^2 \mathbf{I}_m),$$

where $\mathbf{I}_m$ is the $m \times m$ identity matrix and $\sigma = 0.01$. The prior can be configured by choosing from the following assumptions about $\mathbf{x}$:

- 'Gaussian': Gaussian (normal) distribution:
  $x_i \sim \mathcal{N}(0, \delta)$,
- 'GMRF' Gaussian Markov Random Field:
  $x_i - x_{i-1} \sim \mathcal{N}(0, \delta)$,
- 'LMRF' Laplace Markov Random Field:
  $x_i - x_{i-1} \sim \mathcal{L}(0, \delta)$,
- 'CMRF' Cauchy Markov Random Field:
  $x_i - x_{i-1} \sim \mathcal{C}(0, \delta)$,

where $\mathcal{L}$ is the Laplace distribution and $\mathcal{C}$ is the Cauchy distribution. The parameter δ is the prior parameter and is configurable (as an additional parameter that can be provided via UM-Bridge; its default is $\delta = 0.01$.

In Fig. A.12 we show examples of posterior samples obtained for the four different choices of prior. The posterior mean and 95% credibility intervals are shown. The samples are obtained using the built-in automatic sampler selection in CUQIpy [10,9].

A.2.6. Computed tomography benchmark

This benchmark focuses on 2D image reconstruction in X-ray computed tomography (CT) [87]. In CT, X-rays are passed through an object of interest and projection images are recorded at a number of orientations. Materials of different density absorb X-rays by different amounts quantified by their linear attenuation coefficients. A linear forward model describes how the 2D linear attenuation image of an object results in a set of projections measured, typically known as a sinogram.

The inverse problem of CT takes the same form as in the previous section, with $\mathbf{b}$ now an m-dimensional random vector representing the observed (vectorized) CT sinogram data, $\mathbf{A}$ is an $m \times n$ matrix representing the CT projection operator or "system matrix", $\mathbf{x}$ is

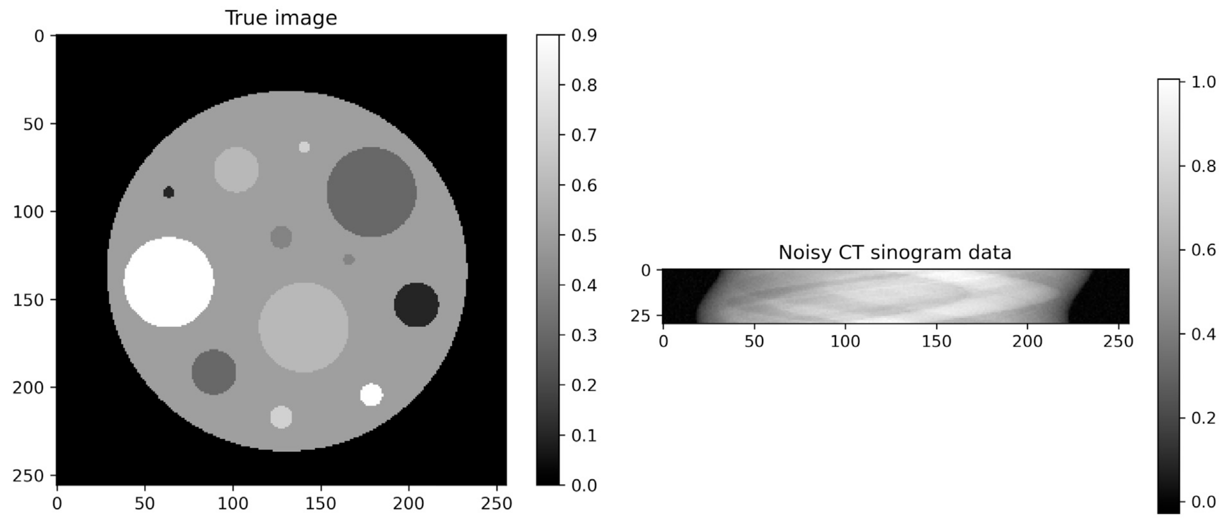

Fig. A.13. Left: Exact solution image. Right: noisy CT sinogram data.

an n-dimensional random vector representing the unknown (vectorized) square image, and $\mathbf{e}$ is an m-dimensional random vector representing the noise. The underlying true image and the noisy CT sinogram data are shown in Fig. A.13.

The matrix $\mathbf{A}$ is generated by using the MATLAB package AIR Tools II[2] [88]. The forward model matrix corresponds to an image of 256×256 pixels, hence $n = 256^2$, scanned in a parallel-beam geometry with 30 projections taking over 180 degrees with angular steps of 6 degrees, each projection consisting of 256 X-ray lines, i.e., $m = 30 \cdot 256 = 7680$. See the above links for more details.

This benchmark follows the same structure and provides the same features as the 1D Deconvolution benchmark in Appendix A.2.5, providing a posterior distribution of $\mathbf{x}$ given $\mathbf{b}$. It also employs CUQIpy[3] [10,9] to specify the linear forward model representing CT and uses the same Gaussian noise model.

The same prior configurations for $\mathbf{x}$ as in the 1D deconvolution benchmark are available, but in 2D versions where distributions stated above are applied to both vertical and horizontal finite differences, $x_{i,j} - x_{i-1,j}$ and $x_{i,j} - x_{i,j-1}$ for $x_{i,j}$ for $i, j = 1, \dots, 256$.

In Fig. A.14 we show examples of posterior samples obtained for the four different choices of prior. The posterior mean and standard deviation images are shown. MCMC samples are obtained using the built-in automatic sampler selection provided by CUQIpy.

A.2.7. One dimensional heat Bayesian inverse problem benchmark

This benchmark is built using the library CUQIpy [10,9]. It defines a posterior distribution of a Bayesian inverse problem governed by a 1D heat equation, with a Gaussian likelihood and KL parameterization of the uncertain parameters. Specifically, the inputs $\mathbf{x} \in \mathbb{R}^{20}$ are 20 KL coefficients, and the output is the posterior probability $\pi(\mathbf{x} \mid \mathbf{b})$ where $\mathbf{b}$ are given data. Posteriors for two cases are available: In the first case, the data is available everywhere in the domain (the interval $[0, 1]$) with a noise level of 0.1%; in the other case, the data is available only on the left half of the domain and with a noise level of 5%. For more details about this Bayesian inverse problem, see [9]. The two cases are accessible via UM-Bridge via the names `Heat1DSmallNoise` and `Heat1DLargeNoise`, respectively. These benchmarks only support evaluation of the forward map.

The underlying inverse problem of this benchmark is to infer an initial temperature profile $g(\xi)$ at time $\tau = 0$ on the unit interval $\xi \in [0, 1]$ from measurements of temperature $u(\xi, \tau)$ at time $\tau = \tau^{\max}$.

We parameterize the discretized initial condition $\mathbf{g}$ using a truncated KL expansion to impose some regularity and spatial correlation, and reduce the dimension of the discretized unknown parameter from n to n_{KL}, where $n_{\mathrm{KL}} \ll n$ and $n_{\mathrm{KL}} = 20$ is the number of the truncated KL expansion coefficients. For a given (discretized) expansion basis $\mathbf{a}_1, \dots, \mathbf{a}_n$, the parameterization of $\mathbf{g}$ in terms of the KL expansion coefficients $x_1, \dots, x_n$ can be written as [9]

$$\mathbf{g} = \sum_{i=1}^{n} x_i \mathbf{a}_i \approx \sum_{i=1}^{n_{\mathrm{KL}}} x_i \mathbf{a}_i .$$

In the Bayesian setting, we consider $\mathbf{x} = [x_1, \dots, x_{n_{\mathrm{KL}}}]^{\mathsf{T}}$ a random vector representing the unknown truncated KL expansion coefficients. Consequently, the corresponding $\mathbf{g}$ is a random vectors as well. We define the inverse problem as

$$\mathbf{b} = \mathbf{A}(\mathbf{x}) + \mathbf{e},$$

[2] https://github.com/jakobsj/AIRToolsII.

[3] https://cuqi-dtu.github.io/CUQIpy/.

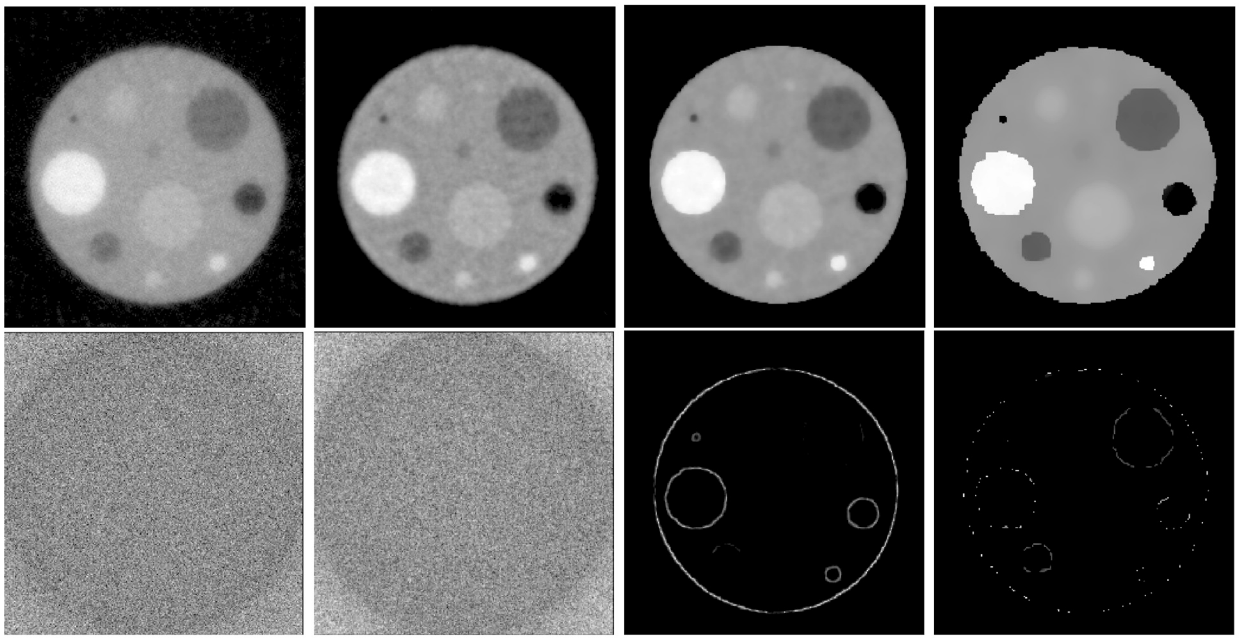

Fig. A.14. Top: Posterior mean images for Gaussian, GMRF, LMRF, CMRF priors, with $\delta = 0.01$, except LMRF where $\delta = 0.1$. Bottom: Posterior standard deviation images for the same cases.

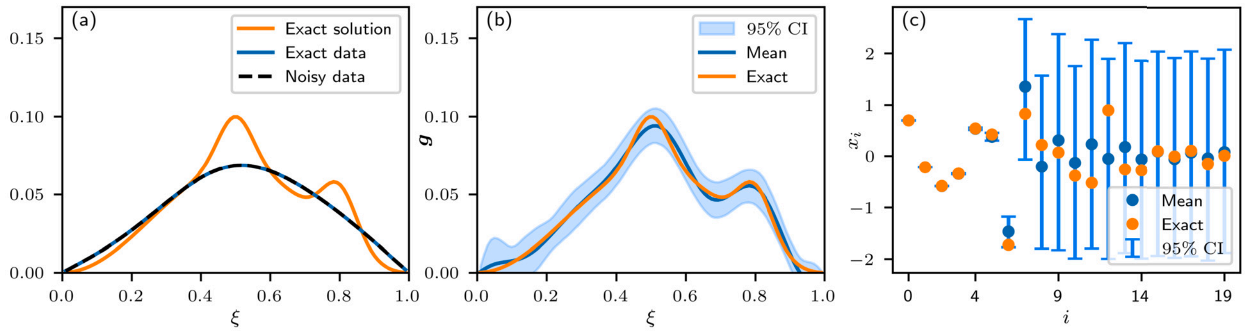

Fig. A.15. The small noise case of the 1d inverse heat benchmark. (a) Noisy data, exact data (solution at $\tau = \tau^{\max}$), and exact solution (solution at $\tau = 0$). Note that for the small noise case, noisy and exact data are visually indistinguishable. (b) Samples' 95% credible interval computed after mapping the samples of KL coefficients $\mathbf{x}$ to samples of the corresponding function $\mathbf{g}$. (c) KL coefficients samples' 95% credible intervals.

where $\mathbf{b}$ is an m-dimensional random vector representing the observed data, measured temperature profile at time $\tau^{\max} = 0.01$ in this case, and $\mathbf{e}$ is an m-dimensional random vector representing the noise. $\mathbf{A}$ is the forward operator that maps $\mathbf{x}$ to the exact data $\mathbf{A}(\mathbf{x})$. Applying $\mathbf{A}$ involves solving the heat PDE above for a given realization of $\mathbf{x}$ and extracting the observations from the space-time solution $\mathbf{u}$.

This benchmark defines a posterior distribution over $\mathbf{x}$ given $\mathbf{b}$ as

$$\pi(\mathbf{x} \mid \mathbf{b}) \propto L(\mathbf{b} \mid \mathbf{x})\pi_{\mathrm{pr}}(\mathbf{x}),$$

where $L(\mathbf{b}|\mathbf{x})$ is a likelihood function and $\pi_{\mathrm{pr}}(\mathbf{x})$ is a prior distribution. The noise is assumed to be Gaussian with a known noise level, and so the likelihood is defined via

$$\mathbf{b} \mid \mathbf{x} \sim \mathcal{N}(\mathbf{A}(\mathbf{x}), \sigma^2 \mathbf{I}_m),$$

where $\mathbf{I}_m$ is the $m \times m$ identity matrix and σ defines the noise level. The prior $\pi_{\mathrm{pr}}(\mathbf{x})$ on the KL coefficients is assumed to be a standard multivariate Gaussian.

Fig. A.15a shows the exact solution (the true initial condition), the exact data $\mathbf{A}(\mathbf{x})$, and a synthetic noisy data $\mathbf{b}^{\mathrm{obs}}$ for the small noise case. We generate 50,000 samples of the posterior using the component-wise Metropolis–Hastings (CWMH) sampler [10,9]. We show in Fig. A.15b the samples' 95% credible interval computed after mapping the KL coefficients $\mathbf{x}$ samples to the corresponding function $\mathbf{g}$ samples; and we show the KL coefficients $\mathbf{x}$ samples 95% credible intervals in Fig. A.15c.

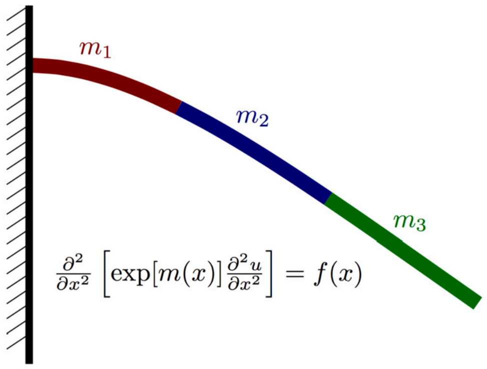
m_1
m_2
m_3
$\frac{\partial^2}{\partial x^2}\left[\exp[m(x)]\frac{\partial^2 u}{\partial x^2}\right] = f(x)$

Fig. A.16. Schematic of the Euler-Bernoulli beam with the material properties m_1, m_2 and m_3.

A.2.8. Beam inference

This benchmark is a Bayesian inverse problem for characterizing the stiffness of an Euler-Bernoulli beam given observations of the beam displacement under a prescribed load. The Young's modulus is piecewise constant over three regions as shown in Fig. A.16. Synthetic data is used. A realization of a Gaussian process is used to define the *true* material properties. Additive noise is added to the output of the model to create the synthetic data. The benchmark is then defined by a probability density function $\pi : \mathbb{R}^3 \to \mathbb{R}$ where the three inputs correspond to Young's modulus on the three regions. The benchmark supports the 'Evaluate', 'Gradient', 'ApplyJacobian', and 'ApplyHessian' operations. The latter three are evaluated using finite differences.

The output probability is defined as usual as the product of a likelihood and a prior:

$$\pi(m \mid y) = L(y \mid m)\pi_{\text{pr}}(m).$$

In order to define the likelihood, let N_x denote the number of finite difference nodes used to discretize the Euler-Bernoulli PDE above. For this problem, we will have observations of the solution $u(x)$ at N_y of the finite difference nodes. Let $\hat{u} \in \mathbb{R}^{N_x}$ denote a vector containing the finite difference solution and let $y \in \mathbb{R}^{N_y}$ denote the observable random variable, which is the solution u at N_y nodes plus some noise ϵ, i.e.

$$y = B\hat{u} + \epsilon,$$

where $\epsilon \sim N(0, \sigma_y)$ and B is the finite difference mass matrix. The solution vector is given by

$$\hat{u} = [K(m)]^{-1}\hat{f},$$

where $K(m)$ is the stiffness matrix of the problem.

Combining this with the definition of y, we have the complete forward model

$$y = B[K(m)]^{-1}\hat{f} + \epsilon.$$

The likelihood function then takes the form

$$L(y|m) = N\left(B[K(m)]^{-1}f,\ \Sigma_y\right).$$

For the prior $\pi_{\text{pr}}(m)$, we assume each value is an independent normal random variable

$$m_i \sim N(\mu_i, \sigma_i^2).$$

A.2.9. Tsunami source inference

The aim of this benchmark is to obtain the parameters describing the initial displacements leading to the Tohoku tsunami from the data of two available buoys located near the Japanese coast. It is based on the model described in Section A.1.3, which also describes the different levels available. The likelihood of a set of parameters given the simulation results is computed using a weighted average of the maximal wave height and the time at which it is reached. The likelihood is given by a normal distribution with mean given by maximum waveheight and the time at which it is reached for the two DART buoys 21418 and 21419 (This data can be obtained from NDBC). The covariance matrix shown in Table A.11 depends on the level, but not the probe point.

The prior cuts off all parameters which would lead to an initial displacement which is too close to the domain boundary. Some parameters may lead to unstable models, e.g., a parameter which initializes the tsunami on dry land; in this case we treat the parameter as unphysical and assign an almost zero likelihood. The description of this benchmark in [45] also contains details on a solution using parallel MLMCMC implemented in the MUQ library [12].

Table A.11
Mean and variances for the Tsunami test case.

μ	$\Sigma\ (l=0)$	$\Sigma\ (l=1)$	$\Sigma\ (l=2)$
1.85232	0.15	0.1	0.1
0.6368	0.15	0.1	0.1
30.23	2.5	1.5	0.75
87.98	2.5	1.5	0.75

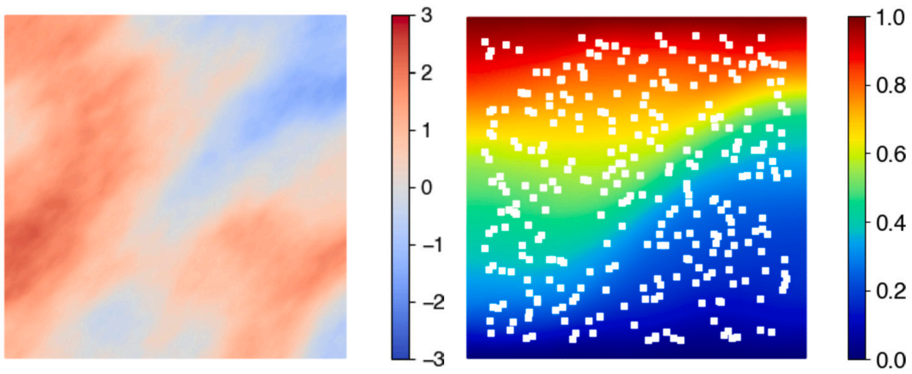

Fig. A.17. True parameter field (left) and the corresponding state field (right) for the Poisson problem. The locations of the observation points are marked as white squares in the right figure.

The inputs of this benchmark are the x and y coordinates of a proposed tsunami origin; the output is a probability given the arrival time and maximum water height at two buoy points of the model compared to measured data provided by NDBC. The benchmark only supports evaluation of the forward map.

A.2.10. Coefficient field inversion in a two-dimensional Poisson linear PDE

This is an implementation of the Bayesian inverse problem presented in [51]. The inverse problem estimates a spatially varying diffusion coefficient in an elliptic PDE given limited noisy observations of the PDE solution – the benchmark is thus related to the one discussed in Section A.2.2, but uses a different parameterization. The coefficient in the equation is discretized with finite elements and efficient derivative information is available through the adjoint techniques implemented in hIPPYlib and hIPPYlib-MUQ.

Specifically, the benchmark is based on the following model:

$$\begin{aligned} -\nabla\cdot\left[\exp(m)\nabla u\right] &= 0 \quad &&\text{in } \Omega,\\ u &= g \quad &&\text{on } \partial\Omega_D,\\ \exp(m)\nabla u\cdot\hat{n} &= 0 \quad &&\text{on } \partial\Omega_N. \end{aligned}$$

Here, Ω is the domain, $u(x)$ is the solution, g are prescribed Dirichlet conditions on the boundary $\partial\Omega_D \subseteq \partial\Omega$, and the last equation represents homogeneous Neumann conditions on the remainder of the boundary $\partial\Omega_N = \partial\Omega\setminus\partial\Omega_D$. The field $m(x)$ is discretized with linear finite elements on a uniform triangular mesh of the unit square $\Omega = [0,1]^2$ with 32 cells in each coordinate direction, and is thus described by 1,089 values that form the input to the problem. The state is discretized with quadratic elements.

The prior distribution on the field $m(x)$ is a Gaussian process defined through a stochastic differential equation. The likelihood is formed from observations of $u(x)$ at 300 points x^i drawn uniformly over $[0.05,0.95]^2$, as shown in Fig. A.17. These observations are obtained by solving the forward problem on the finest mesh with the true parameter field shown on the left in Fig. A.17 and then adding a random Gaussian noise with a relative standard deviation of 0.5% to the resulting state. Model, prior, likelihood, and sampling of the posterior, as well as convergence diagnostics are implemented with the hIPPYlib-MUQ[4] package, which enables efficient calculation of gradients, Jacobian actions, and Hessian actions using adjoint and tangent linear techniques.

Based on these definitions, the benchmarks then return the Laplace approximation of the posterior probability that maps the inputs $m\in\mathbb{R}^{1089}$ to $\mathbb{R}$. The benchmark supports the 'Evaluate', 'Gradient', 'ApplyJacobian', and 'ApplyHessian' operations.

A.2.11. Coefficient field inversion in a Robin boundary condition for a three-dimensional p-Poisson nonlinear PDE

This benchmark focuses on a three-dimensional nonlinear Bayesian inverse problem presented in [51]. The inverse problem estimates a two-dimensional flux boundary condition on the bottom of a thin three-dimensional domain with nonlinear p-Poisson PDE. Observations of the PDE solution at the top of the domain are used, as shown in Fig. A.18.

[4] https://hippylib.github.io/muq-hippylib/.

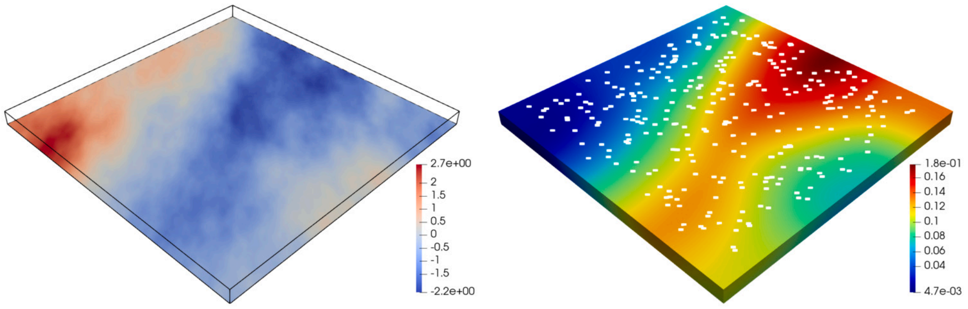

Fig. A.18. Left: True parameter field on the bottom surface. Right: Corresponding state field and $l = 300$ observation points (white square marks) on the top surface.

Table A.12
Properties of the Euler-Bernoulli beam propagation benchmark.

Mapping	Dimensions	Description
input	[3]	The value of the beam stiffness in each lumped region.
output	[31]	The resulting beam displacement at 31 equidistant grid nodes.

This benchmark then defines a Bayesian posterior density over a spatially-variable boundary condition $m(x)$ given noisy observations of the solution of the nonlinear partial differential equation

$$\begin{aligned} -\nabla \cdot \left[|\nabla u|_\epsilon^{p-2} \nabla u \right] &= f && \text{in } \Omega, \\ u &= g && \text{on } \partial\Omega_D, \\ |\nabla u|_\epsilon^{p-2} \nabla u \cdot \hat{n} &= m && \text{on } \partial\Omega_N, \end{aligned}$$

where Ω is the domain, $u(x)$ is the solution and g are prescribed Dirichlet conditions on the boundary $\Omega_D \subseteq \partial\Omega$. In this benchmark we set $p = 3$ and $f = 0$. The domain, illustrated in the figure above, is $\Omega = [0,1]^2 \times [0, 0.05]$.

The output of the benchmark is a probability that maps a discretized version of $m(x)$ at 233,389 nodes to the product of a prior and a likelihood. The prior distribution on the boundary condition $m(x)$ is a Gaussian process defined through a stochastic differential equation; see [51, Section 5.1.2] for details. The likelihood is formed from observations of $u(x)$ at 300 points x^i drawn uniformly over the top surface of the domain. Gaussian noise with a standard deviation of 0.005 is assumed. Like the benchmark of the previous section, model, prior, likelihood, and sampling of the posterior, as well as convergence diagnostics are implemented with the hIPPYlib-MUQ package.

This benchmark supports the 'Evaluate', 'Gradient', 'ApplyJacobian', and 'ApplyHessian' operations.

A.3. Propagation benchmarks

In this section we describe the propagation benchmarks contained in the benchmark library, these implement a specific forward UQ problem.

A.3.1. Euler-Bernoulli beam

This is an uncertainty propagation problem modeling the effect of uncertain material parameters on the displacement of an Euler-Bernoulli beam with a prescribed load. Refer to Appendix A.2.8 for details.

The inputs and outputs of this model for this propagation benchmark can be found in Table A.12. Supported features of this benchmark are the 'Evaluate', 'Gradient', 'ApplyJacobian', and 'ApplyHessian' operations. The latter three are evaluated using finite differences.

For the prior, we assume the material parameter to be a uniformly distributed random variable

$$M \sim U_{[1,1.05]^3}.$$

The goal is to identify the distribution of the resulting quantity of interest, the deflection of the beam at finite difference node 10 and 25 of the discretization. Samples from the model output $\hat{u}$ are illustrated in Fig. A.19.

A.3.2. Genz integration and surrogate benchmark

This benchmark implements a suite of six families of test functions first proposed in [89] to elucidate the performance of quadrature algorithms for computing $\mathbb{E}[f(\theta)]$ which can be computed analytically if θ is an uncertain parameter with specific distribution, and f is a specific forward model. It is also useful for benchmarking methods for constructing surrogate models, for example see [90].

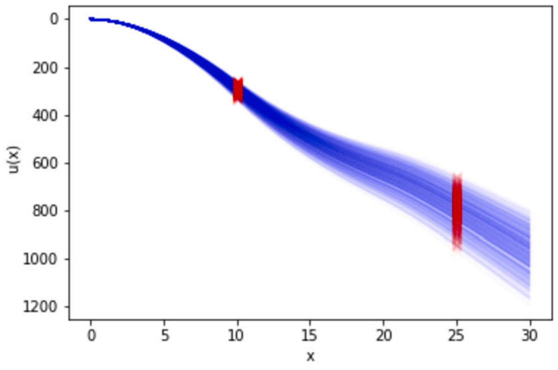

Fig. A.19. Samples of the Euler-Bernoulli beam model output $F(M)$ with the quantity of interest (solution at nodes 10 and 25) marked with crosses.

The six function families are parameterized by a scalable number $n \geq 1$ of independent and identically distributed uniform variables with non-zero joint density defined on the unit hypercube $[0,1]^n$.

Specifically, the six families of functions are as follows:

$$\text{Oscillatory} \qquad f(\theta) = \cos\left(2\pi w_1 + \sum_{i=1}^{n} c_i\theta_i\right)$$

$$\text{ProductPeak} \qquad f(\theta) = \prod_{i=1}^{n} \left(c_i^{-2} + (\theta_i - w_i)^2\right)^{-1}$$

$$\text{CornerPeak} \qquad f(\theta) = \left(1 + \sum_{i=1}^{n} c_i\theta_i\right)^{-(n+1)}$$

$$\text{GaussianPeak} \qquad f(\theta) = \exp\left(-\sum_{i=1}^{n} c_i^2(\theta_i - w_i)^2\right)$$

$$C_0 - \text{Continuous} \qquad f(\theta) = \exp\left(-\sum_{i=1}^{n} c_i|\theta_i - w_i|\right)$$

$$\text{Discontinuous} \qquad f(\theta) = \begin{cases} 0, & \theta_1 > w_1 \text{ or} \\ & \theta_2 > w_2 \\ \exp\left(\sum_{i=1}^{n} c_i\theta_i\right), & \text{otherwise} \end{cases}$$

The relative magnitude of the coefficients $c_i > 0$ impacts the difficulty of integrating or approximating these families of functions. This benchmark implements various types of decay ofthe coefficients as a function of the index i. This introduces anisotropy into the functions (i.e., it varies each variable's importance) that can be exploited by adaptive methods. The types of coefficient decay are:

$$\text{No decay} \qquad \hat{c}_i = \frac{i + 0.5}{n}$$

$$\text{Quadratic decay} \qquad \hat{c}_i = \frac{1}{(i+1)^2}$$

$$\text{Quartic decay} \qquad \hat{c}_i = \frac{1}{(i+1)^4}$$

$$\text{Exponential decay} \qquad \hat{c}_i = \exp\left(\log(c_{\min})\frac{i+1}{n}\right)$$

$$\text{Squared-exponential decay} \qquad \hat{c}_i = 10^{\left(\log_{10}(c_{\min})\frac{(i+1)^2}{n^2}\right)}$$

Here $c_{\min} = 5 \times 10^{-6}$. With the decay formulas above, the final benchmark coefficients are obtained by normalizing such that

$$c_i = C\frac{\hat{c}_i}{\sum_{i=1}^{n} \hat{c}_i},$$

where $C > 0$ is set by the user and defaults to 1.0 if not provided. Generally, increasing the size of coefficients $\hat{c}_i$ will make integration or constructing surrogate models of these functions more difficult. Finally, while the $0 \leq w_i \leq 1$ parameters were varied in [89], they do not affect the difficulty of the integration problem and so we set $w_1 = w_2 = \ldots = W_n = W$ to a user specified constant $W \in \mathbb{R}$.

Table A.13
Configuration variables for the Genz benchmark.

Config	Type	Default	Description
n	integer	1	Function dimension
C	float	1	Normalizing constant
W	float	0.5	Shift parameter
T	string	sqexp	coefficient decay type
N	string	oscillatory	integrand family

With these definitions, each of the six families of functions maps an uncertain parameter $\theta \in \mathbb{R}^D$ to $\mathbb{R}$. This benchmark only supports evaluation of the forward map, with configuration options in Table A.13.

A.3.3. L2-Sea propagation

This benchmark has been presented and discussed in Section 5.1. Here we only specify that the optional configuration parameters presented when discussing the model in Section A.1.2 are set in the benchmark as follows: *sinkoff* = `'y'`, *trimoff* = `'y'`. As a reference value, we also report an approximation of the expected value of the resistance: more specifically, following the script reported in Section 5.1 (note in particular that such example is run with *fidelity* = 3) and setting the sparse grid to $w = 15$, we obtain the sparse grid with 256 points shown at the bottom row of Fig. 6, and $\mathbb{E}[R] \approx 51.664730931098383$ N.

We also provide a reference value for a second setting, namely assuming that *Fr* and T are uniformly distributed in their intervals, rather than having respectively a triangular and a beta random distribution; we also lower the *fidelity* config parameter to *fidelity* = 6. We then adjust the following lines of the script reported in Section 5.1

```
fid = 6;
knots_Fr = @(n) knots_leja(n,Fr_a,Fr_b,'sym_line');
lev2knots_Fr = @lev2knots_2step;
knots_T = @(n) knots_leja(n,T_a,T_b,'sym_line');
lev2knots_T = @lev2knots_2step;
w = 9;
S = create_sparse_grid(N,w,{knots_Fr,knots_T},{@lev2knots_Fr,@lev2knots_T});
```

and obtain a grid with 181 points that yields $\mathbb{E}[R] \approx 68.398655639653271$ N.

A.3.4. The cookies problem

This benchmark runs a forward UQ problem for the elliptic variant of the cookies model described in Appendix A.1.8. The results have been obtained using the Sparse Grids Matlab Kit [15] and the Matlab interface to UM-Bridge. More specifically, we assume that the uncertain parameters θ_n appearing in the definition of the diffusion coefficient Eq. (A.5) are uniform i.i.d. random variables on the range $[-0.99, -0.2]$ and we aim at computing the expected value of the quantity of interest Ψ in Eq. (A.6). The structure of this benchmark therefore identical to the test discussed in [80]; however, raw numbers are different since in that work the mesh and PDE solver employed were different (standard FEM with piecewise linear basis on a triangular mesh); in particular, for this benchmark, we fix the finite element degree at $p = 1$ and the fidelity config parameter to $k = 4$.

As a reference value, the UM-Bridge documentation provides the approximation of the expected value $\mathbb{E}[\Psi]$ computed with a standard Smolyak sparse grid, based on Clenshaw–Curtis points for levels $w = 0, 1, \dots, 5$ (the finest grid having around 15000 collocation points); see e.g. [15] for more details on the sparse grid construction, as well as the benchmark script available on the UM-Bridge documentation.

A.4. Optimization benchmarks

UM-Bridge's interface is sufficiently generic to allow more than just UQ problems to be solved. As a proof of concept we provide an example optimization benchmark in this section.

A.4.1. L2-Sea optimization

This optimization benchmark employs the L2-sea model described in Appendix A.1.2, and was developed within the activities of the NATO-AVT-331 Research Task Group on “Goal-driven, multifidelity approaches for military vehicle system-level design” [91]. The goal is to minimize the resistance R of the DTMB 5415 in calm water at fixed speed (i.e. fixed Froude number, namely $Fr = 0.28$), and nominal draft $T = -6.16$, by suitably modifying the shape of the hull of the vessel. In other words, the first two components of the parameters vector θ are fixed, and we optimize on the $N = 14$ shape parameters, $\mathbf{s}$, only. The optimization problem thus reads

$$\begin{array}{lll} \text{minimize} & f(\mathbf{s}) & \\ \text{subject to} & h_i(\mathbf{s}) = 0 & i = 1,2 \\ & g_k(\mathbf{s}) \le 0 & k = 1,\dots,4 \\ & -1 \le s_i \le 1 & i = 1,\dots,N \end{array}$$

where the objective function f is the total resistance R, the equality constraints h_i (automatically satisfied by the shape modification method) include (h_1) fixed length between perpendiculars of the vessel and (h_2) fixed vessel displacement, whereas the inequality constraints $g_1, \dots, g_4$ encode geometrical constrains on the modified hull in relation to the original one. In particular, g_1 relates to the beam, g_2 to the draft (both allowing $\pm 5\%$ of maximum variation), and g_3, g_4 relate to the minimum cylindrical dimensions to contain the sonar in the bow dome, see [66] for details: these are the four additional outputs of the L2-sea model that were hinted at the end of Appendix A.1.2.

A reference optimum has been found in [20] with a multi-fidelity method (i.e., the config parameter *fidelity* is not fixed a-priori but chosen adaptively by the optimizer), where an objective improvement of 12.5% is achieved with an NCC = 628. Specifically, the minimum of the resistance is at $\mathbf{s} = (-1.000000, -0.906250, -0.781250, -0.242188, 0.533203, -0.906250, 0.128418, 0.812500, -0.070313, -0.878296, 0.422363, 0.031250, 0.365234, -0.175659)$ and the corresponding value computed at *fidelity* = 1 is 36.3116 N. Note that NCC is the normalized computational cost and it is evaluated as the ratio between the CPU time needed for the generic j-th fidelity and the highest-fidelity. To allow comparison between methods, the standard NCC to be used as the cost of function evaluation for each fidelity level is {1.00, 0.46, 0.20, 0.11, 0.07, 0.04, 0.03}, from highest to lowest. These results have been obtained by fixing the config parameters *sinkoff* and *trimoff* to `'y'`.

Data availability

Simulation codes and relevant data for inference problems are published as container images in the online benchmark library.

References

[1] A.M. Stuart, Inverse problems: a Bayesian perspective, Acta Numer. 19 (2010) 451–559, https://doi.org/10.1017/S0962492910000061.

[2] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, D.M. Blei, Automatic differentiation variational inference, J. Mach. Learn. Res. (2017).

[3] S. Duane, A. Kennedy, B.J. Pendleton, D. Roweth, Hybrid Monte Carlo, Phys. Lett. B 195 (2) (1987) 216–222, https://doi.org/10.1016/0370-2693(87)91197-X.

[4] M.D. Hoffman, A. Gelman, et al., The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo, J. Mach. Learn. Res. 15 (1) (2014) 1593–1623.

[5] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 contributors, SciPy 1.0: fundamental algorithms for scientific computing in Python, Nat. Methods 17 (2020) 261–272, https://doi.org/10.1038/s41592-019-0686-2.

[6] B. Carpenter, A. Gelman, M.D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, A. Riddell, Stan: a probabilistic programming language, J. Stat. Softw. 76 (1) (2017) 1–32, https://doi.org/10.18637/jss.v076.i01.

[7] J. Salvatier, T.V. Wiecki, C. Fonnesbeck, Probabilistic programming in Python using PyMC3, PeerJ Comput. Sci. 2 (2016) e55.

[8] E. Bingham, J.P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, N.D. Goodman, Pyro: deep universal probabilistic programming, J. Mach. Learn. Res. (2018).

[9] A.M.A. Alghamdi, N.A.B. Riis, B.M. Afkham, F. Uribe, S.L. Christensen, P.C. Hansen, J.S. Jørgensen, CUQIpy – part II: computational uncertainty quantification for PDE-based inverse problems in Python, arXiv:2305.16951, 2023.

[10] N.A.B. Riis, A.M.A. Alghamdi, F. Uribe, S.L. Christensen, B.M. Afkham, P.C. Hansen, J.S. Jørgensen, CUQIpy – part I: computational uncertainty quantification for inverse problems in Python, arXiv:2305.16949, 2023.

[11] D. Foreman-Mackey, D.W. Hogg, D. Lang, J. Goodman, Emcee: the MCMC hammer, pasp 125 (925) (2013) 306, https://doi.org/10.1086/670067, arXiv:1202.3665.

[12] M. Parno, A. Davis, L. Seelinger, MUQ: the MIT uncertainty quantification library, J. Open Sour. Softw. 6 (68) (2021) 3076, https://doi.org/10.21105/joss.03076.

[13] J. Jakeman, PyApprox: a software package for sensitivity analysis, Bayesian inference, optimal experimental design, and multi-fidelity uncertainty quantification and surrogate modeling, Environ. Model. Softw. 170 (2023) 105825, https://doi.org/10.1016/j.envsoft.2023.105825.

[14] S.-C.T. Choi, F.J. Hickernell, M. McCourt, A. Sorokin, QMCPy: a quasi-Monte Carlo Python library, github.com/QMCSoftware/QMCSoftware, 2020.

[15] C. Piazzola, L. Tamellini, Algorithm 1040: the sparse grids Matlab kit - a Matlab implementation of sparse grids for high-dimensional function approximation and uncertainty quantification, ACM Trans. Math. Softw. (Mar 2024), https://doi.org/10.1145/3630023.

[16] M.B. Lykkegaard, tinyDA, github.com/mikkelbue/tinyDA, 2023.

[17] I.V. Oseledets, S. Dolgov, V. Kazeev, D. Savostyanov, O. Lebedeva, P. Zhlobich, T. Mach, L. Song, TT-toolbox, github.com/oseledets/TT-Toolbox, 2011.

[18] D. Tsapetis, M.D. Shields, D.G. Giovanis, A. Olivier, L. Novak, P. Chakroborty, H. Sharma, M. Chauhan, K. Kontolati, L. Vandanapu, D. Loukrezis, M. Gardner, UQpy v4.1: uncertainty quantification with Python, SoftwareX 24 (2023) 101561, https://doi.org/10.1016/j.softx.2023.101561.

[19] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, D. Wells, The deal. II finite element library: design, features, and insights, Comput. Math. Appl. 81 (2021) 407–422, https://doi.org/10.1016/j.camwa.2020.02.022.

[20] R. Pellegrini, A. Serani, G. Liuzzi, F. Rinaldi, S. Lucidi, M. Diez, A derivative-free line-search algorithm for simulation-driven design optimization using multi-fidelity computations, Mathematics 10 (3) (2022) 481.

[21] D. Aristoff, W. Bangerth, A benchmark for the Bayesian inversion of coefficients in partial differential equations, SIAM Rev. 65 (4) (2023) 1074–1105, https://doi.org/10.1137/21M1399464.

[22] D. Xiu, Stochastic Collocation Methods: A Survey, 2015, pp. 1–18.

[23] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092, https://doi.org/10.1063/1.1699114.

[24] L. Seelinger, V. Cheng-Seelinger, A. Davis, M. Parno, A. Reinarz, UM-Bridge: uncertainty quantification and modeling bridge, J. Open Sour. Softw. 8 (83) (2023) 4748, https://doi.org/10.21105/joss.04748.

[25] M.B. Giles, Multilevel Monte Carlo path simulation, Oper. Res. 56 (3) (2008) 607–617, https://doi.org/10.1287/opre.1070.0496.

[26] T. Dodwell, C. Ketelsen, R. Scheichl, A. Teckentrup, A hierarchical multilevel Markov chain Monte Carlo algorithm with applications to uncertainty quantification in subsurface flow, https://doi.org/10.1137/130915005, 2015.

[27] M.B. Lykkegaard, G. Mingas, R. Scheichl, C. Fox, T.J. Dodwell, Multilevel delayed acceptance MCMC with an adaptive error model in PyMC3, arXiv:2012.05668, 2020.

[28] V. Hoang, C. Schwab, A. Stuart, Complexity analysis of accelerated MCMC methods for Bayesian inversion, Inverse Probl. 29 (2012), https://doi.org/10.1088/0266-5611/29/8/085010.

[29] T. Cui, G. Detommaso, R. Scheichl, Multilevel Dimension-Independent Likelihood-Informed MCMC for Large-Scale Inverse Problems, 2019.

[30] M.B. Lykkegaard, T.J. Dodwell, C. Fox, G. Mingas, R. Scheichl, Multilevel delayed acceptance MCMC, SIAM/ASA J. Uncertain. Quantificat. 11 (1) (2023) 1–30, https://doi.org/10.1137/22M1476770.

[31] A.L. Teckentrup, P. Jantsch, C.G. Webster, M. Gunzburger, A multilevel stochastic collocation method for partial differential equations with random input data, SIAM/ASA J. Uncertain. Quantificat. 3 (1) (2015) 1046–1074, https://doi.org/10.1137/140969002.
[32] A. Jasra, K. Kamatani, K. Law, Y. Zhou, A multi-index Markov chain Monte Carlo method, Int. J. Uncertain. Quantificat. 8 (2017), https://doi.org/10.1615/Int.J.UncertaintyQuantification.2018021551.
[33] C. Piazzola, L. Tamellini, R. Pellegrini, R. Broglia, A. Serani, M. Diez, Comparing multi-index stochastic collocation and multi-fidelity stochastic radial basis functions for forward uncertainty quantification of ship resistance, Eng. Comput. 39 (2023) 2209–2237, https://doi.org/10.1007/s00366-021-01588-0.
[34] J.D. Jakeman, M.S. Eldred, G. Geraci, A. Gorodetsky, Adaptive multi-index collocation for uncertainty quantification and sensitivity analysis, Int. J. Numer. Methods Eng. 121 (6) (2020) 1314–1343, https://doi.org/10.1002/nme.6268, https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.6268.
[35] J. Beránek, A. Böhm, R. Machacek, Vyomkesh, J. Klinkovský, B. Decker, Vanessasaurus, It4innovations/hyperqueue: v0.16.0, https://doi.org/10.5281/zenodo.8140508, Jul 2023.
[36] G.M. Kurtzer, V. Sochat, M.W. Bauer, Singularity: scientific containers for mobility of compute, PLoS ONE 12 (5) (2017) 1–20, https://doi.org/10.1371/journal.pone.0177459.
[37] D. Merkel, Docker: lightweight Linux containers for consistent development and deployment, Linux J. 2014 (239) (2014) 2.
[38] M. Rosenblatt, Remarks on some nonparametric estimates of a density function, Ann. Math. Stat. 27 (3) (1956) 832–837.
[39] A. Bowman, A. Azzalini, Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations, Oxford Statistical Science Series, OUP, Oxford, 1997.
[40] G. Casella, C.P. Robert, M.T. Wells, Generalized accept-reject sampling schemes, Lect. Notes Monogr. Ser. (2004) 342–347.
[41] J. Bénézech, L. Seelinger, P. Bastian, R. Butler, T. Dodwell, C. Ma, R. Scheichl, Scalable multiscale-spectral GFEM with an application to composite aero-structures, arXiv:2211.13893, 2023.
[42] P. Bastian, M. Blatt, A. Dedner, N.-A. Dreier, C. Engwer, R. Fritze, C. Gräser, C. Grüninger, D. Kempf, R. Klöfkorn, M. Ohlberger, O. Sander, The Dune framework: basic concepts and recent developments, Comput. Math. Appl. 81 (2021) 75–112, https://doi.org/10.1016/j.camwa.2020.06.007.
[43] A. Reinarz, D.E. Charrier, M. Bader, L. Bovard, M. Dumbser, K. Duru, F. Fambri, A.-A. Gabriel, J.-M. Gallard, S. Köppel, L. Krenz, L. Rannabauer, L. Rezzolla, P. Samfass, M. Tavelli, T. Weinzierl, ExaHyPE: an engine for parallel dynamically adaptive simulations of wave problems, Comput. Phys. Commun. 254 (2020) 107251, https://doi.org/10.1016/j.cpc.2020.107251.
[44] L. Rannabauer, M. Dumbser, M. Bader, ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework, Comput. Fluids 173 (2018) 299–306, https://doi.org/10.1016/j.compfluid.2018.01.031.
[45] L. Seelinger, A. Reinarz, L. Rannabauer, M. Bader, P. Bastian, R. Scheichl, High performance uncertainty quantification with parallelized multilevel Markov chain Monte Carlo, in: Proc. of the Int. Conf. for HPC, Netw., Stor. and Ana., SC'21, Association for Computing Machinery, New York, NY, USA, 2021.
[46] J.A. Christen, C. Fox, Markov chain Monte Carlo using an approximation, J. Comput. Graph. Stat. 14 (4) (2005) 795–810, https://doi.org/10.1198/106186005X76983.
[47] T.J. Dodwell, C. Ketelsen, R. Scheichl, A.L. Teckentrup, Multilevel Markov chain Monte Carlo, SIAM Rev. 61 (3) (2015) 509–545, https://doi.org/10.1137/19M126966X.
[48] T. Bai, A.L. Teckentrup, K.C. Zygalakis, Gaussian processes for Bayesian inverse problems associated with linear partial differential equations, Stat. Comput. 34 (4) (2024) 139, https://doi.org/10.1007/s11222-024-10452-2.
[49] C.E. Rasmussen, C.K.I. Williams, Gaussian Processes for Machine Learning, Adapt. Comp. and Mach. Learn., MIT Press, Cambridge, Mass, 2006, oCLC: ocm61285753.
[50] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J.E. Gonzalez, I. Stoica, Tune: a research platform for distributed model selection and training, arXiv preprint, arXiv:1807.05118, 2018.
[51] K.-T. Kim, U. Villa, M. Parno, Y. Marzouk, O. Ghattas, N. Petra, HIPPYlib-MUQ: a Bayesian inference software framework for integration of data with complex predictive models under uncertainty, arXiv:2112.00713, 2022.
[52] B. Adams, W. Bohnhoff, K. Dalbey, M. Ebeida, J. Eddy, M. Eldred, R. Hooper, P. Hough, K. Hu, J. Jakeman, et al., Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification and sensitivity analysis: version 6.12 user's manual, https://doi.org/10.2172/1630694, 2020.
[53] D. Sinoquet, Lagun, gitlab.com/drti/lagun, 2023.
[54] S. Marelli, B. Sudret, UQLab: a framework for uncertainty quantification in Matlab, pp. 2554–2563, https://doi.org/10.1061/9780784413609.257.
[55] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauss, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, S. Wolf, The functional mockup interface for tool independent exchange of simulation models, pp. 105–114, https://doi.org/10.3384/ecp11063105, 2011.
[56] G. Chourdakis, K. Davis, B. Rodenberg, M. Schulte, F. Simonis, B. Uekermann, G. Abrams, H. Bungartz, L. Cheung Yau, I. Desai, K. Eder, R. Hertrich, F. Lindner, A. Rusch, D. Sashko, D. Schneider, A. Totounferoush, D. Volland, P. Vollmer, O. Koseomur, preCICE v2: a sustainable and user-friendly coupling library [version 2; peer review: 2 approved], Open Res. Europe 2 (51) (2022), https://doi.org/10.12688/openreseurope.14445.2.
[57] J.J. Dongarra, The LINPACK benchmark: an explanation, in: E.N. Houstis, T.S. Papatheodorou, C.D. Polychronopoulos (Eds.), Supercomputing, Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, pp. 456–474.
[58] R.A. Harris, M. Barall, R. Archuleta, E. Dunham, B. Aagaard, J.P. Ampuero, H. Bhat, V. Cruz-Atienza, L. Dalguer, P. Dawson, S. Day, B. Duan, G. Ely, Y. Kaneko, Y. Kase, N. Lapusta, Y. Liu, S. Ma, D. Oglesby, K. Olsen, A. Pitarka, S. Song, E. Templeton, The SCEC/USGS dynamic earthquake rupture code verification exercise, Seismol. Res. Lett. 80 (1) (2009) 119–126, https://doi.org/10.1785/gssrl.80.1.119, https://pubs.geoscienceworld.org/ssa/srl/article-pdf/80/1/119/2760745/119.pdf.
[59] R.A. Harris, M. Barall, B. Aagaard, S. Ma, D. Roten, K. Olsen, B. Duan, D. Liu, B. Luo, K. Bai, J. Ampuero, Y. Kaneko, A. Gabriel, K. Duru, T. Ulrich, S. Wollherr, Z. Shi, E. Dunham, S. Bydlon, Z. Zhang, X. Chen, S.N. Somala, C. Pelties, J. Tago, V.M. Cruz-Atienza, J. Kozdon, E. Daub, K. Aslam, Y. Kase, K. Withers, L. Dalguer, A suite of exercises for verifying dynamic earthquake rupture codes, Seismol. Res. Lett. 89 (3) (2018) 1146–1162, https://doi.org/10.1785/0220170222, https://pubs.geoscienceworld.org/ssa/srl/article-pdf/89/3/1146/4127122/srl-2017222.1.pdf.
[60] L. Xing, N. Salman, V. Aarre, T. Theoharis, E. Tjaland, A3mark: a publicly available online web service for seismic attribute benchmarking, pp. 1929–1933, https://doi.org/10.1190/segam2016-13841990.1, 2016.
[61] P. Fischer, M. Min, T. Rathnayake, S. Dutta, T. Kolev, V. Dobrev, J.-S. Camier, M. Kronbichler, T. Warburton, K. Świrydowicz, J. Brown, Scalability of high-performance pde solvers, Int. J. High Perform. Comput. Appl. 34 (5) (2020) 562–586, https://doi.org/10.1177/1094342020915762.
[62] The MORwiki community, MORwiki - model order reduction wiki, modelreduction.org.
[63] A. Olivieri, F. Pistani, A. Avanzini, F. Stern, R. Penna, Towing tank, sinkage and trim, boundary layer, wake, and free surface flow around a naval combatant INSEAN 2340 model, Tech. Rep., DTIC, 2001.
[64] H. Sadat-Hosseini, D.H. Kim, S. Toxopeus, M. Diez, F. Stern, CFD and potential flow simulations of fully appended free running 5415M in irregular waves, in: World Marit, Tech. Conf, Providence, RI, Nov 2015, pp. 3–7.
[65] A. Serani, G. Fasano, G. Liuzzi, S. Lucidi, U. Iemma, E.F. Campana, F. Stern, M. Diez, Ship hydrodynamic optimization by local hybridization of deterministic derivative-free global algorithms, Appl. Ocean Res. 59 (2016) 115–128.
[66] G. Grigoropoulos, E.F. Campana, M. Diez, A. Serani, O. Goren, K. Sarioz, D. Danisman, M. Visonneau, P. Queutey, M. Abdel-Maksoud, S. Frederick, Mission-based hull-form and propeller optimization of a transom Stern destroyer for best performance in the sea environment, in: Proc. of the VII Int. Cong. on Comp. Met. in Marine Eng.-MARINE, 2017.

[67] A. Serani, F. Stern, E.F. Campana, M. Diez, Hull-form stochastic optimization via computational-cost reduction methods, Eng. Comput. 38 (3) (2022) 2245–2269.
[68] C.W. Dawson, A practical computer method for solving ship-wave problems, in: Proc. of the 2nd Int. Conf. on Num. Ship Hydrodyn., Berkeley, 1977, pp. 30–38.
[69] M. Landrini, E. Campana, Steady waves and forces about a yawing flat plate, J. Ship Res. 40 (03) (1996) 179–192.
[70] H. Schlichting, K. Gersten, Boundary-Layer Theory, Springer-Verlag, Berlin, 2000.
[71] A. Serani, M. Diez, J. Wackers, M. Visonneau, F. Stern, Stochastic shape optimization via design-space augmented dimensionality reduction and RANS computations, in: 57th AIAA Aero. Sci. Meet., SciTech 2019, 2019, p. 2218.
[72] A. Reinarz, T. Dodwell, T. Fletcher, L. Seelinger, R. Butler, R. Scheichl, Dune-composites – a new framework for high-performance finite element modelling of laminates, Compos. Struct. 184 (2018) 269–278, https://doi.org/10.1016/j.compstruct.2017.09.104.
[73] A. Sandhu, A. Reinarz, T. Dodwell, A Bayesian framework for assessing the strength distribution of composite structures with random defects, Compos. Struct. 205 (2018) 58–68, https://doi.org/10.1016/j.compstruct.2018.08.074.
[74] E. Hodille, X. Bonnin, R. Bisson, T. Angot, C. Becquart, J. Layet, C. Grisolia, Macroscopic rate equation modeling of trapping/detrapping of hydrogen isotopes in tungsten materials, J. Nucl. Mater. 467 (2015) 424–431, https://doi.org/10.1016/j.jnucmat.2015.06.041.
[75] R. Delaporte-Mathurin, E.A. Hodille, J. Mougenot, Y. Charles, C. Grisolia, Finite element analysis of hydrogen retention in ITER plasma facing components using FESTIM, Nucl. Mater. Energy 21 (2019) 100709, https://doi.org/10.1016/j.nme.2019.100709.
[76] S. Dixon, Achlys: isotope self-diffusion, https://doi.org/10.5281/zenodo.6412090, Dec 2021.
[77] A.D. Lindsay, D.R. Gaston, C.J. Permann, J.M. Miller, D. Andrš, A.E. Slaughter, F. Kong, J. Hansel, R.W. Carlsen, C. Icenhour, L. Harbour, G.L. Giudicelli, R.H. Stogner, P. German, J. Badger, S. Biswas, L. Chapuis, C. Green, J. Hales, T. Hu, W. Jiang, Y.S. Jung, C. Matthews, Y. Miao, A. Novak, J.W. Peterson, Z.M. Prince, A. Rovinelli, S. Schunert, D. Schwen, B.W. Spencer, S. Veeraraghavan, A. Recuero, D. Yushu, Y. Wang, A. Wilkins, C. Wong, 2.0 - MOOSE: enabling massively parallel multiphysics simulation, SoftwareX 20 (2022) 101202, https://doi.org/10.1016/j.softx.2022.101202.
[78] A. Bershteyn, J. Gerardin, D. Bridenbecker, C.W. Lorton, J. Bloedow, R.S. Baker, G. Chabot-Couture, Y. Chen, T. Fischle, K. Frey, J.S. Gauld, H. Hu, A.S. Izzo, D.J. Klein, D. Lukacevic, K.A. McCarthy, J.C. Miller, A.L. Ouedraogo, T.A. Perkins, J. Steinkraus, Q.A. ten Bosch, H.-F. Ting, S. Titova, B.G. Wagner, P.A. Welkhoff, E.A. Wenger, C.N. Wiswell, Institute for Disease Modeling, Implementation and applications of EMOD, an individual-based multi-disease modeling platform, Pathog. Dis. 76 (5) (2018) fty059, https://doi.org/10.1093/femspd/fty059.
[79] D. Arndt, W. Bangerth, M. Feder, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, M. Kronbichler, M. Maier, P. Munch, J.-P. Pelteret, S. Sticko, B. Turcksin, D. Wells, The deal. II library, version 9.4 30, J. Numer. Math. (2022) 231–246, https://doi.org/10.1515/jnma-2022-0054.
[80] J. Bäck, F. Nobile, L. Tamellini, R. Tempone, Stochastic spectral Galerkin and collocation methods for PDEs with random coefficients: a numerical comparison, in: J. Hesthaven, E. Ronquist (Eds.), Spec. and High Ord. Met. for Part. Diff. Eq., in: Lecture Notes in Computational Science and Engineering, vol. 76, Springer, 2011, pp. 43–62.
[81] D. Kressner, C. Tobler, Low-rank tensor Krylov subspace methods for parametrized linear systems, SIAM J. Matrix Anal. Appl. 32 (4) (2011) 1288–1316, https://doi.org/10.1137/100799010.
[82] J. Ballani, L. Grasedyck, Hierarchical tensor approximation of output quantities of parameter-dependent PDEs, SIAM/ASA J. Uncertain. Quantificat. 3 (1) (2015) 852–872.
[83] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, The fenics project version 1.5, Arch. Numer. Softw. 3 (100) (2015), https://doi.org/10.11588/ans.2015.100.20553.
[84] A. Logg, K.-A. Mardal, G. Wells, Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book, vol. 84, Springer Science & Business Media, 2012.
[85] A. Iserles, A First Course in the Numerical Analysis of Differential Equations, 2nd edition, Cambridge Texts in Applied Mathematics, Cambridge University Press, 2008.
[86] O. Ogorodnikova, J. Roth, M. Mayer, Deuterium retention in tungsten in dependence of the surface conditions, J. Nucl. Mater. 313–316 (2003) 469–477, https://doi.org/10.1016/S0022-3115(02)01375-2.
[87] P.C. Hansen, J. S.J., William R.B. Lionheart (Eds.), Computed Tomography: Algorithms, Insight, and Just Enough Theory, SIAM, Philadelphia, 2021.
[88] P.C. Hansen, J.S. Jørgensen, AIR tools II: algebraic iterative reconstruction methods, improved implementation, Numer. Algorithms 79 (2018) 107–137, https://doi.org/10.1007/s11075-017-0430-x.
[89] A. Genz, Testing multidimensional integration routines, in: Proc. of International Conference on Tools, Methods and Languages for Scientific and Engineering Computation, Elsevier North-Holland, Inc., USA, 1984, pp. 81–94.
[90] J. Jakeman, M. Eldred, K. Sargsyan, Enhancing ℓ_1-minimization estimates of polynomial chaos expansions using basis selection, J. Comput. Phys. 289 (2015) 18–34, https://doi.org/10.1016/j.jcp.2015.02.025.
[91] P.S. Beran, D.E. Bryson, A.S. Thelen, M. Diez, A. Serani, Comparison of multi-fidelity approaches for military vehicle design, in: 21st AIAA/ISSMO Multidisc. Ana. and Opt. Conf. (MA&O), AVIATION 2020, Virtual Event, June 15-19, 2020.