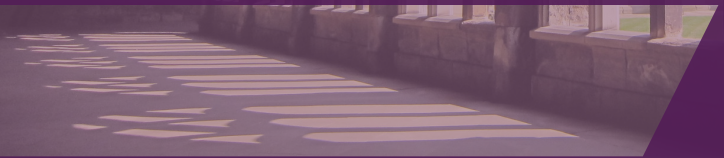
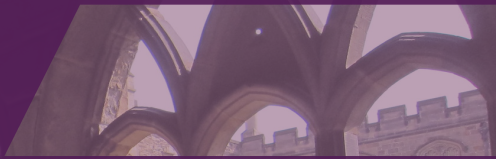
Session 3

Practical Bayesian Inference using MCMC





Anne Reinarz

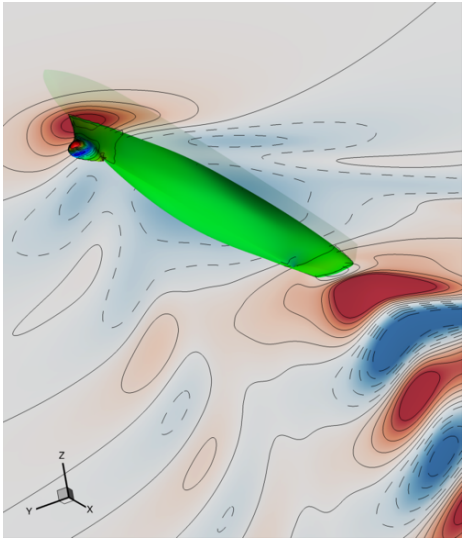
Durham University

Overview

- Session 1: Introduction
- Session 2: Markov Chain Monte-Carlo
- **Session 3:** Scaling up and hierarchical models
  - How to scale up models
  - Hands-on exercise
  - Multilevel MCMC
  - Application Example
  - Contributing

Scalability

Scalability: Starting with an example



L2 Sea application

As an example we use a naval engineering application

Goal: compute the PDF of the resistance to advancement R_T of a boat (naval equivalent of the drag force for airplanes).

The boat is advancing in calm water, under uncertain Froude number and draft.

Thread parallel UQ

- QMCPy's `UMBridgeWrapper(..., parallel=N)` uses a only allows for thread parallelism and is as such restricted to a single machine.
- For many integrand calls, the wrapper creates N workers (processes or threads).
- Each worker independently calls the UM-Bridge model via HTTP:

$$\texttt{model.evaluate}(\theta)$$

- Each call becomes an independent HTTP request to the server.
- The UM-Bridge server handles each HTTP request in parallel (if resources allow).
- QMCPy still "thinks" it is running with thread parallelism.

UM-Bridge and HPC

Goal

Automatically scale up UM-Bridge models for challenging UQ problems; keep UQ client simple

UM-Bridge and HPC

Goal

Automatically scale up UM-Bridge models for challenging UQ problems; keep UQ client simple

Wish list:

- Avoid machine specific model setup
- Many instances of (in turn parallel) UM-Bridge models
- Single point of entry for UQ,
- automatic load balancing

HPC vs. Cloud

UM-Bridge support for cloud and classical HPC

Cloud

Use kubernetes to orchestrate containers and use their provided loadbalancer.
UM-Bridge currently provides two kubernetes configurations:

- Parallel model instances, single node models
- multinode MPI

HPC systems

Instead of docker use apptainer (prev. singularity) containers or install directly and either

- Start individual slurm jobs from the login node or laptop, or
- make use of loadbalancing via. hyperqueue

MPI Support in the cloud

- MPI within containers trivial, but restricted to one hardware node
- MPI across containers/nodes:
  Separate kubernetes configuration Requires `mpioperator` base image for model container

MPI Support in the cloud

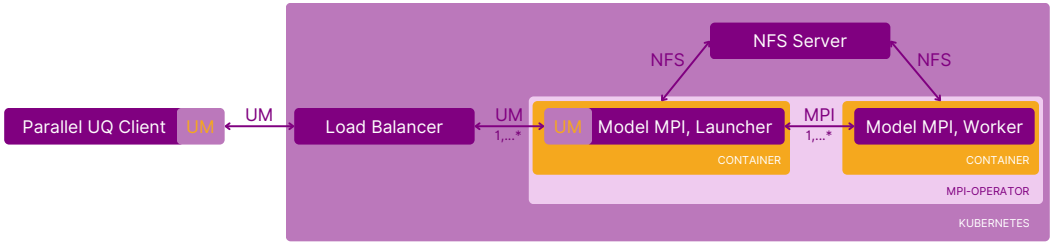


Figure: Kubernetes configuration provides a pre-built reference configuration with loadbalancer. Support for openmpi, mpich and intel MPI.

UM-Bridge: Load-balancer backends

Two load-balancer backends

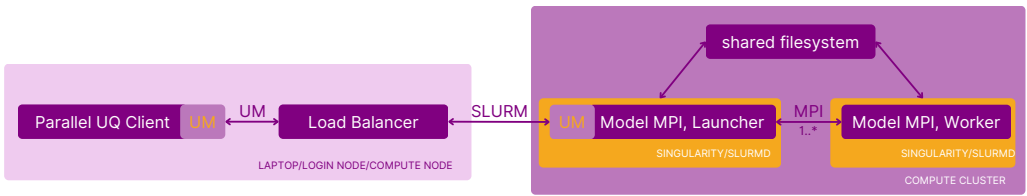
- SLURM
  - Easier to setup
  - Higher job overhead
- HyperQueue
  - More difficult to setup
  - Lower job overhead
  - More flexibility per job

Compile load-balancer with `make` (Makefile included)
Execute load-balancer binary with

```
./loadbalancer --scheduler=<scheduler> --port=<port>
```

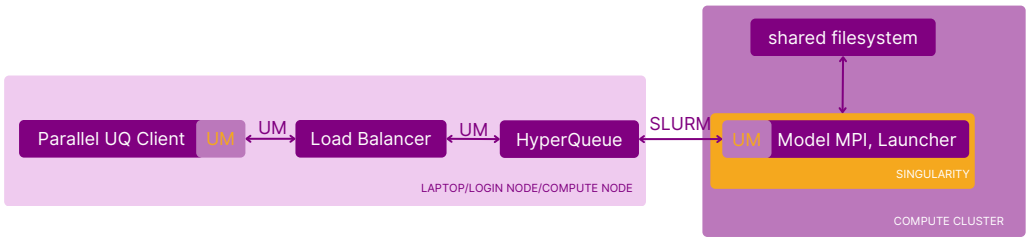
UM-Bridge: HPC Load-balancer

- Sends resources request to scheduler
- Distribute parallel client requests to model servers
- Client sees it as a server, server sees it as a client
- Can run on login node or compute node



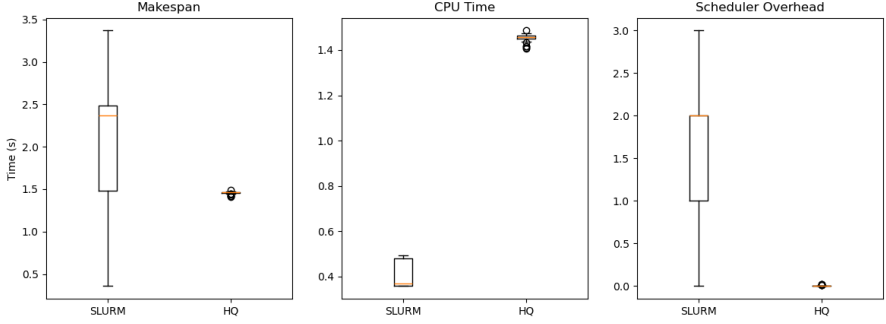
UM-Bridge: HyperQueue Load-balancer

- Sends resources request to scheduler
- Distribute parallel client requests to model servers
- Client sees it as a server, server sees it as a client
- Can run on login node or compute node



Loadbalancing for quick jobs

If each model evaluation is fast (e.g. coarse models or surrogate models), the overhead from starting a SLURM job for each becomes high. UM-Bridge provides an interface for running HyperQueue (`https://it4innovations.github.io/hyperqueue/stable/`) to ameliorate this issue.



Hands-on

How to run UM-Bridge on olivia

Start a terminal following the same instructions provided:
`https://md.sigma2.no/l_Y-tYGfQVy5CNYorOiTtQ?view#Exercises-for-lecture-on-EESSI`

Set up a virtual environment to install python packages:

```
python -m venv venv_umbridge
. venv_umbridge/bin/activate
pip install umbridge qmcpy pymc
```

Now we are ready to run something!

Setting up the loadbalancer

The following steps will deploy a simple load balancer using the SLURM backend.

Firstly, you'll need the C++ source code for the load balancer, which can be found in the hpc directory of the UM-Bridge repository.

A Makefile is provided to compile the executable (by typing make in the terminal).

Setting up the loadbalancer

We are following the instructions of section 6 of the tutorial `https://um-bridge-benchmarks.readthedocs.io/en/docs/tutorial.html`

In the tutorial section of the git repository you can find the modified jobscript. It contains the right partition, account and reservation for the winterschool. By default it will run the simple testmodel, we can replace this with a more complex model later.

Setting up the loadbalancer

Instead of copying the existing job_sigma2.sh from the repository you can also use the base job and set

```
#SBATCH --account=nn14000k
#SBATCH --partition=normal
#SBATCH --reservation=geilo_winter_school
```

then modify this check (you are changing the -I to -i)

```
host=$(hostname -i | awk '{print $1}')
```

Running the load balancer

For the purpose of this tutorial we will keep `./testmodel` as the model server since the Makefile will have compile this dummy model automatically; this server is exactly the same as the minimal-server.py in earlier sections.

The load balancer can be launched with

```
./load-balancer --scheduler=slurm
```

Running the load balancer

The terminal should show something like:

```
Argument --port not set! Using port 4242 as default.
Waiting for URL file: urls/url-15053556.txt
==============================MODEL INFO====================
Available models and corresponding job-scripts:
* Model 'forward' --> 'slurm_scripts/job.sh'


================================================================
Listening on port 4242...
```

Once the Listening on port line is displayed, you can connect any parallel client to it.

Hands-on

Follow the instructions to run the load balancer on olivia

Connect a parallel client to it, e.g. the qmcpy client in the tutorial folder or the pymc client from yesterday.

Monitor the running slurm jobs with

```
squeue -u YOUR_USERNAME
```

I don’t want to overload you with new software, but tmux is a very useful tool for splitting the terminal into multiple screens.

Scalability results

Scalability on classical HPC systems

Parallel MCMC on a simple diffusion model

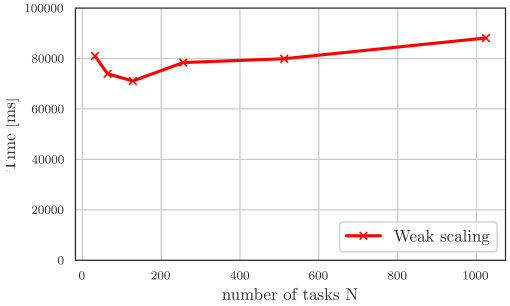


Figure: Weak scaling (more samples, more cores)

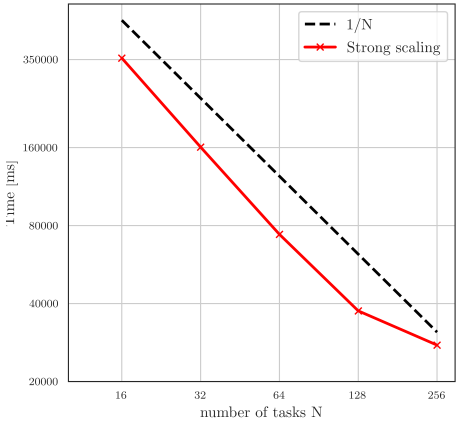
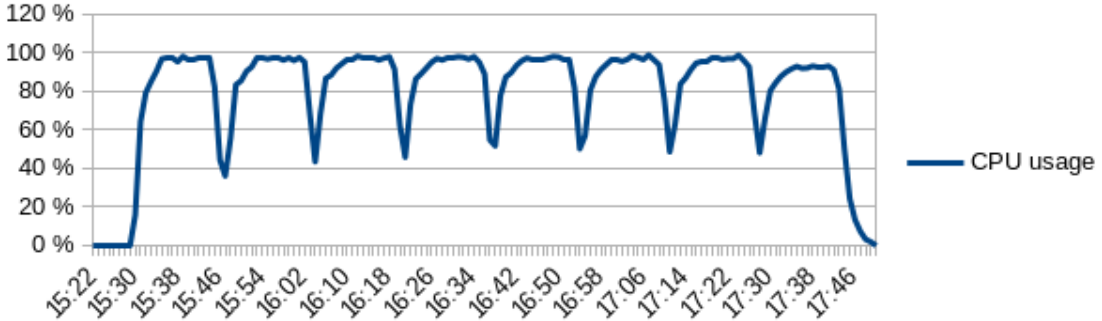


Figure: Strong scaling (constant

Performance on GKE

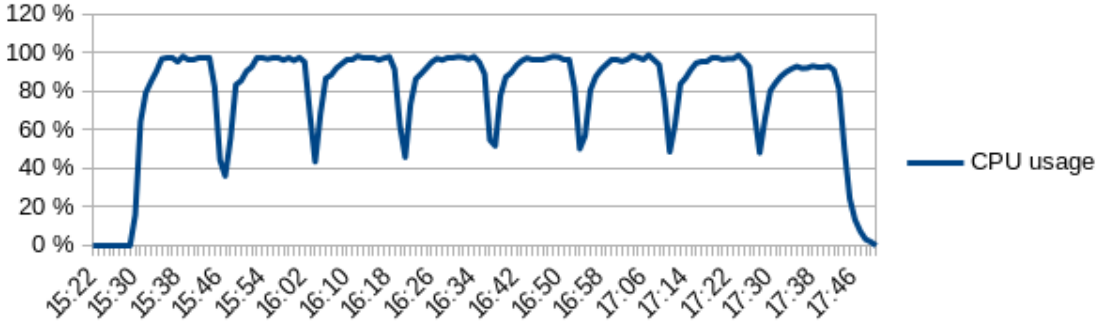
- Run using GKE with 100 `c2d-highcpu-56` nodes running 100 instances of the model container.
- *Total:* 5600 vCPUs including SMT threads on GKE
- Each model instance was run on one node using Intel TBB parallelization within ExaHyPE
- MLDA was run on a workstation, connecting to the cluster above via UM-Bridge
- Gaussian process evaluations were conducted directly on the workstation due to their low computational cost
- Parallelization of tinyDA using multiple samplers (enabled by the python library Ray)

Performance



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

Performance



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

- *Overall run time:* 2h20min
- Parallel speedup of 96.38 for 100 MLDA chains.

MLMCMC

Multilevel MCMC

Telescoping sum of QOI like MLMC:

$$\mathbb{E}_{\pi^L}[Q_L] = \underbrace{\mathbb{E}_{\pi^0}[Q_0]}_{\text{Coarse approx.}} + \sum_{l=1}^{L} \underbrace{\left(\mathbb{E}_{\pi^l}[Q_l] - \mathbb{E}_{\pi^{l-1}}[Q_{l-1}]\right)}_{\text{Corrections}}.$$

Multilevel MCMC

Telescoping sum of QOI like MLMC:

$$\mathbb{E}_{\pi_L}[Q_L] = \underbrace{\mathbb{E}_{\pi_0}[Q_0]}_{\text{Coarse approx.}} + \sum_{l=1}^{L} \underbrace{\left(\mathbb{E}_{\pi_l}[Q_l] - \mathbb{E}_{\pi_{l-1}}[Q_{l-1}]\right)}_{\text{Corrections}}.$$

- Coarser chains can be used as cheap-to-compute, well-informed and nearly uncorrelated proposals for finer chains.

Multilevel MCMC

Telescoping sum of QOI like MLMC:

$$\mathbb{E}_{\pi^L}[Q_L] = \underbrace{\mathbb{E}_{\pi^0}[Q_0]}_{\text{Coarse approx.}} + \sum_{l=1}^{L} \underbrace{(\mathbb{E}_{\pi^l}[Q_l] - \mathbb{E}_{\pi^{l-1}}[Q_{l-1}])}_{\text{Corrections}}.$$

- Coarser chains can be used as cheap-to-compute, well-informed and nearly uncorrelated proposals for finer chains.
- fine corrections are more expensive per sample, but the variance in those corrections is reduced and thus only few samples are needed

Multilevel MCMC

- Efficiency depends on strong correlation between levels k and $k-1$

Multilevel MCMC

- Efficiency depends on strong correlation between levels k and $k-1$
- MLMCMC constructs *coupled Markov chains*:
  - Coarse and fine chains share randomness
  - Proposals reuse coarse-level information

Multilevel MCMC

- Efficiency depends on strong correlation between levels k and $k-1$
- MLMCMC constructs *coupled Markov chains*:
  - Coarse and fine chains share randomness
  - Proposals reuse coarse-level information
- Each level targets its own posterior π^k

Multilevel MCMC

- Efficiency depends on strong correlation between levels k and $k-1$
- MLMCMC constructs *coupled Markov chains*:
  - Coarse and fine chains share randomness
  - Proposals reuse coarse-level information
- Each level targets its own posterior π^k
- Acceptance probability is modified to preserve:
  - Correct marginal distribution at level k
  - Detailed balance

MLMCMC Acceptance Probability

- The acceptance probability has the form

$$\alpha(\theta'_k, \theta_k) = \min\left\{1,\ \frac{\pi^k(\theta'_k)}{\pi^k(\theta_k)} \frac{g_k(\theta_k \mid \theta'_k)}{g_k(\theta'_k \mid \theta_k)} \frac{\pi^{k-1}(\theta_{k,C})}{\pi^{k-1}(\theta'_{k,C})}\right\}$$

MLMCMC Acceptance Probability

- The acceptance probability has the form

$$\alpha(\theta'_k, \theta_k) = \min\left\{1,\ \frac{\pi^k(\theta'_k)}{\pi^k(\theta_k)} \frac{g_k(\theta_k \mid \theta'_k)}{g_k(\theta'_k \mid \theta_k)} \frac{\pi^{k-1}(\theta_{k,C})}{\pi^{k-1}(\theta'_{k,C})}\right\}$$

  - First ratio: standard Metropolis–Hastings correction at level k
  - Second ratio: proposal symmetry correction (often cancels)
  - Third ratio: compensates for using samples from level $k-1$

MLMCMC Algorithm

Result: Markov chains $\{\theta_k^i\}_{i=0}^{N_k}$ for all levels $k \in \{0, \ldots, L\}$.

On level 0, run a conventional MCMC chain, delivering samples θ_0^i.

for *level* $k = 1, \ldots, L-1$ **do**

  Choose starting point θ_k^0 with coarse component from next coarser starting point θ_{k-1}^0.

  for *sample* $j = 1, \ldots, N_k$ **do**

    Given θ_k^j, generate proposal $\theta_k' = \left\{ \begin{array}{c} \theta_{k,C}' \\ \theta_{k,F} \end{array} \right\}$ where

    - $\theta_{k,C}'$ is drawn from a coarser chain and
    - $\theta_{k,F}'$ from proposal density $q_k(\theta_{k,F}' | \theta_{k,F}^j)$.

    Compute acceptance probability

$$\alpha(\theta_k', \theta_k^j) = \min \left\{ 1, \frac{\pi^k(\theta_k') g_k(\theta_k^j | \theta_k')}{\pi^k(\theta_k^j) g_k(\theta_k' | \theta_k^j)} \cdot \frac{\pi^{k-1}(\theta_{k,C}^j)}{\pi^{k-1}(\theta_{k,C}')} \right\}.$$

    Draw a random number $r \in [0, 1]$.

    if $r < \alpha(\theta_k', \theta_k^j)$ **then**

      Accept proposal: θ_k' as θ_k^{j+1}

    else

      Reject proposal: $\theta_k^{j+1} = \theta_k^j$

    end

  end

end

MLMCMC Efficiency

MLMCMC cost

For the Poisson equation: The computational cost $\mathcal{C}_\epsilon$ required to achieve a mean square error below ϵ is then bounded by

$$\mathcal{C}_\epsilon \lesssim \epsilon^{-(d+2)-\delta} \qquad \textit{and} \qquad \mathcal{C}_\epsilon \lesssim \epsilon^{-(d+1)-\delta}$$

respectively, where δ is a model specific constant.

- the cost for the multilevel method is one order below the single-level one.

[Tim Dodwell, Chris Ketelsen, Robert Scheichl, and Aretha Teckentrup. 2015]

Application Example

The Bayesian inverse problem

- We want to infer the initial displacements most likely to lead to given data at buoy points.
- Our parameters θ are the position in x and y of the initial seafloor displacement.
- The likelihood $\mathcal{L}(y|\theta)$ is given by a normal distribution $\mathcal{N}(\mu, \Sigma)$ with mean μ given by maximum waveheight $\max h$ and the time t at which it is reached for the the two DART buoys 21418 and 21419. The covariance matrix Σ depends only on the level.
- Seelinger, et al. *High Performance Uncertainty Quantification with Parallelized Multilevel Markov Chain Monte Carlo*, 2021.

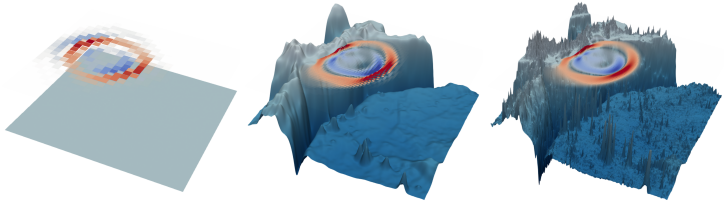
The forward model

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ hu \\ hv \\ b \end{pmatrix} + \nabla \cdot \begin{pmatrix} hu & hv \\ hu^2 & huv \\ huv & hv^2 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 \\ hg\,\partial_x(b+h) \\ hg\,\partial_y(b+h) \\ 0 \end{pmatrix} = 0, \qquad (1)$$

where h denotes the height of the water column, (u, v) the horizontal flow velocity, g gravity and b denotes the bathymetry.

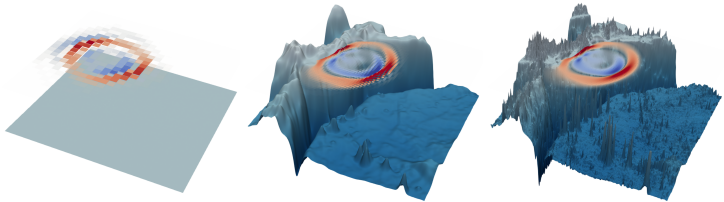
- The numerical solver is well-balanced handles wetting and drying through use of an HLLE limiter in coastal regions.
- Rannabauer, et al. *ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework*, 2018.

The level hierarchy



1. Coarsest level has constant bathymetry (59,049 P0 elements) and runs on 48 cores in aprox. 8 sec.
2. Smoothed bathmetry model (59,049 P3 elements + limiting), approx. 2 minutes
3. Finest model (531,441 P3 elements + limiting), approx. 10 minutes

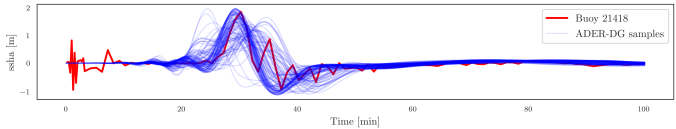
The level hierarchy



The MLMCMC method with three levels computes

- 400 samples on level 0 (subsampling rate 25, burn-in 50), required 60,000 computed samples
- 300 on level 1 (subsampling rate 5), required 2000 computed samples
- 100 on level 2 (no subsampling)

The Tohoku Scenario



level l	t_l	ρ_l	$\mathbb{E}[Q_0]$ or $\mathbb{E}[Q_l - Q_{l-1}]$		$\mathbb{V}[Q_0]$ or $\mathbb{V}[Q_l - Q_{l-1}]$	
0	7.38	25	1.33	29.35	1687.38	1016.16
1	97.3	5	-16.15	-7.33	947.59	585.19
2	438.1	0	9.81	17.41	414.83	1052.54

Table: Computational cost t_l, chosen subsampling rate ρ_l, mean and variance for both components of Q.

The Tohoku Scenario

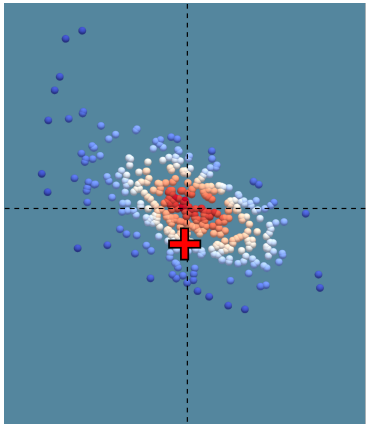


Figure: level 0

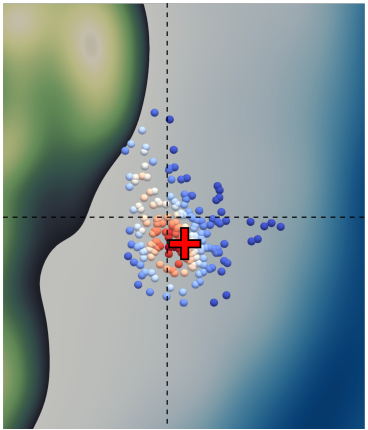


Figure: level 1

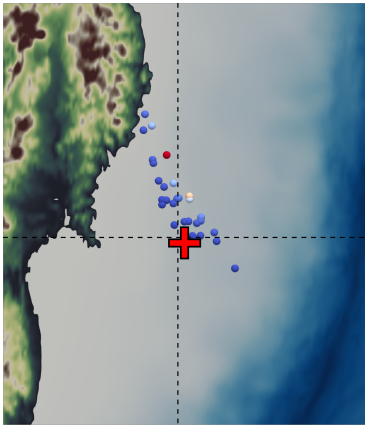
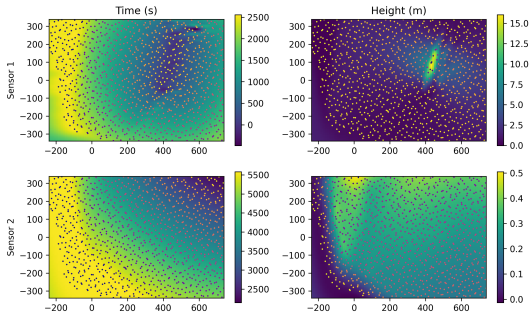


Figure: level 2

Surrogate for the Coarsest level

- trained on 1024 low-discrepancy samples from the smoothed (l=1) model
- constant mean function, a Matérn-$\frac{5}{2}$ covariance function with Automatic Relevance Determination (ARD) and a noise-free Gaussian likelihood
- hyperparameters optimized using Type-II Maximum Likelihood Estimation



GP training points (dots) and GP predictions (surfaces)

Contributing

Contributing

We are always happy for more people to get involved.

If you have a language or framework or feature you would like to see added get in touch.

We have a slack channel you can join via the documentation and a weekly standup if you have any questions.

If you run into a problem/bug: open an issue of github

UM-Bridge Models & Benchmarks Library

Goal

Maintain a curated set of ready-to-run benchmark problems and models which can be used to reproducibly compare UQ methods.

UM-Bridge Models & Benchmarks Library

Goal

Maintain a curated set of ready-to-run benchmark problems and models which can be used to reproducibly compare UQ methods.

- **Automated testing** and building via Github actions.
- Partially-automated **documentation**.
- **Ease of use:** Each model or benchmark can be accessed from any supported language through a simple function call

UM-Bridge Models

Mathematically, an UM-Bridge model is a function mapping parameter vectors onto model output vectors, supporting some of the following: evaluation, gradient evaluation, Jacobian action, or Hessian action.

UM-Bridge models may be used in multiple benchmarks, e.g. there may be an infererence problem and a propagation problem in the benchmarks library using the same model.

UM-Bridge Benchmarks

Uncertainty quantification benchmarks are identical to models regarding implementation of both server and client. They only differ in that they fully define a UQ problem rather than just a forward model.

They are based on forward models of the previous section.

- The inference benchmarks define a complete Bayesian posterior density, including prior, likelihood and observations. The goal is then to determine the posterior distribution or quantities derived from it.

UM-Bridge Benchmarks

Uncertainty quantification benchmarks are identical to models regarding implementation of both server and client. They only differ in that they fully define a UQ problem rather than just a forward model.

They are based on forward models of the previous section.

- The propagation benchmarks that are defined by a model and a specific distribution probability distribution. The goal is then to find the corresponding distribution of the model output, or a quantity derived therefrom.

UM-Bridge Benchmarks

Uncertainty quantification benchmarks are identical to models regarding implementation of both server and client. They only differ in that they fully define a UQ problem rather than just a forward model.

They are based on forward models of the previous section.

- The optimization benchmark defines an optimization problem with a range of parameters under certain constraints.

Benchmark examples

Name	Short description
Analytic densities	Infers the PDF of various analytic function.
Membrane model	Infers the PDF of stiffness values of a membrane from measured deformation data.
Disease transmission model	Agent based model of disease transmission in an entirely susceptible population with correlation between disease acquisition and transmission. Provided problem is to infer the disease parameters that result in 40% of the population infected at the end of the outbreak.
Computed Tomography	Defines a posterior distribution for a 2D X-ray CT image reconstruction problem problem, with a Gaussian noise distribution.

Take-aways

- Multilevel MCMC (MLMCMC) reduces computational cost by reducing variance
- Coupling inference to simulators is difficult without abstraction

$\Rightarrow$ Naïve integration takes longer than expected, is hard to maintain, and not reusable

- UM-Bridge provides a general interface to plug in models and UQ methods

$\Rightarrow$ Promotes modularity, reuse, and separation of concerns

Contribute

UM-Bridge Github `https://github.com/UM-Bridge/`

The documentation is at `https://um-bridge-benchmarks.readthedocs.io/en/docs/`

You can join our slack channel

We run an online workshop every December: `https://um-bridge.github.io/workshop/`

In case there is still time

Advanced tasks

Go to `https://um-bridge-benchmarks.readthedocs.io/en/docs/tutorial.html`

Follow the steps in section 5

or look at the alternative load balancers such as hyperqueue at the bottom of the page

References

- L. Seelinger, A. Reinarz, L. Rannabauer, M. Bader, P. Bastian, R. Scheichl. High performance uncertainty quantification with parallelized multilevel Markov chain Monte Carlo. *Proc. of the Int. Conf. for High Performance Computing*, 2021.
- L. Seelinger, A. Reinarz et al. Democratizing uncertainty quantification. *Journal of Computational Physics*, 521:113542, 2024.
- J. Dubois, S. Imperiale, A. Mangeney, F. Bouchut, J. Sainte-Marie. Acoustic and gravity waves in the ocean: a new derivation of a linear model from the compressible Euler equation. *Geophysical Journal International*, 233(1):171–195, 2023.