**Motivation**

MUQ (pronounced "muck") is a collection of C++ and Python libraries for accelerating both the application of existing uncertainty quantification (UQ) algorithms and the development of new approaches. To facilitate these tasks, MUQ provides interfaces for defining and coupling models with UQ-related algorithms. Throughout our code, we use well-founded software engineering techniques and stable external libraries. Our goal is to provide an efficient and stable platform to help users "MUQ it" about—and much more—UQ.

**Inheritance and extendability**

A good software library will allow a user to complete complicated tasks with a minimal amount of code. In MUQ, we try to achieve this using abstract classes. Users implement a few member functions that define core functionality, and code from the parent class provides additional functionality. This type of object-oriented programming makes it easy to extend MUQ with new models and new algorithms.

**Example implementation: MCMC**

There are 3 key components to most MCMC algorithms:

1. The chain.
2. The kernel.
3. The proposal.

MUQ defines each component through an abstract base class.

**Combining model components**

Assume we want to create a Bayesian posterior density, \( p(f|\theta) \propto f_\theta V_\theta (f|\theta) \), where \( f(\theta) = N(f,0), d(\theta) = \cal{f} \), and \( c = N(0,1) \). The graphical model for the posterior and the corresponding code are given below.

**C++ Code:**

```c++
// Define the proposal density for a simple Gaussian random walk proposal.

// The exchangeability of MCMC kernels

// Extend the proposal variance from the parameters

// to tell the problem what information this proposal needs

// Code examples

// Separating models from algorithms

// It is challenging to provide a flexible interface between scientific models and a wide variety of algorithms. This is because different algorithms exploit different aspects of model structure, and models can provide varying levels of information (e.g., gradients, Hessians, block structure). In MUQ, we have generally adopted a three-component system to define the model-algorithm interface.

// Benefits of this approach:

// Models are independent of algorithms, which allows greater flexibility on both sides.

// Problems can extract algorithm-specific structure from models and provide meaningful defaults.

// Model/problem approximations can be employed without changing algorithms.

```

**Property tree parameters**

Nearly all algorithms have tunable parameters. In MUQ, these are defined through either a boost property tree (in C++) or a dictionary (in Python). In either case, parameter names are matched with parameter values. These pairs can be easily stored in structured text files.

**Use in research**

- Robust optimization of pumping rates
  - Idea: Use posterior samples to evaluate expected head.
  - Control pumping rates to restrict contaminant movement.
  - Optimization performed with sample average approximation (SAA) or stochastic approximation (SA).

- The same model implementation can be used for both optimization and inference (just in different graphs), reducing repeated code.

**Example:**

```
// This function generates a sample of the proposal

// This function evaluates the log proposal density for a pair of points

// to督查 Model

// Model Specific

```

**Where to get MUQ?**

Want more? Download MUQ and find links to our documentation at:

[bitbucket.mit.edu/mituq/muq](https://bitbucket.mit.edu/mituq/muq)

---

**MUQ (MIT Uncertainty Quantification): Flexible Software for Connecting Algorithms and Applications**

Matthew Parino, Andrew Davis, Patrick Conrad, and Yousef Marzouk

Center for Computational Engineering Massachusetts Institute of Technology

**ugroup.mit.edu**

Supported by the US Department of Energy, Office of Advanced Scientific Computing Research (ASCR), part of the SciDAC Institute for the Quantification of Uncertainty in Extreme Scale Computations (QUEST).