MIT Uncertainty Quantification (MUQ) Library

Andrew Davis [1] Matthew Parno [2] Youssef Marzouk [1]

[1]Massachusetts Institute of Technology

[2]Cold Regions Research and Engineering Lab (CRREL)

SIAM UQ 2016

MUQ: philosophy

"A small change in algorithm or application should only require a small change in code"

Goals:

- Provide a sandbox for playing with different models and algorithms
  - Facilitate constructive modeling
  - Extensible and easily hybridized algorithms
- Reduce distance and time between algorithms and applications

MUQ: philosophy

"A small change in algorithm or application should only require a small change in code"

Goals:

- Provide a sandbox for playing with different models and algorithms
  - Facilitate constructive modeling
  - Extensible and easily hybridized algorithms
- Reduce distance and time between algorithms and applications

Target audiences:

- Algorithm developers
- Application scientists and engineers

MUQ: outline

1. **Modeling framework**
   - Constructing combined (physical + statistical) models with high-level structure
   - Graphical modeling technique to easily combine sub models

2. **Algorithm framework**
   - *Example:* Markov chain Monte Carlo stack — using mathematical structure in software design

3. **MUQ development update**
   - Use of MUQ in our research

MUQ: modeling

Unique modeling approach:

1. **Graphically** building physical and statistical models
   - Connect models and algorithms to *expose underlying structure*

MUQ: modeling

Unique modeling approach:

1. **Graphically** building physical and statistical models
   - Connect models and algorithms to *expose underlying structure*

2. Easy to **swap and extend** models
   - Apply MCMC (and other algorithms) to a *wide variety of applications*

MUQ: modeling

Unique modeling approach:

1. **Graphically** building physical and statistical models
   - Connect models and algorithms to *expose underlying structure*
2. Easy to **swap and extend** models
   - Apply MCMC (and other algorithms) to a *wide variety of applications*
3. "Grey-box" approach: MUQ **wraps around existing models**, but uses any new or available problem structure

MUQ: modeling

Unique modeling approach:

1. **Graphically** building physical and statistical models
   - Connect models and algorithms to *expose underlying structure*
2. Easy to **swap and extend** models
   - Apply MCMC (and other algorithms) to a *wide variety of applications*
3. "Grey-box" approach: MUQ **wraps around existing models**, but uses any new or available problem structure
4. Built-in tools to **implement physical models**

Graphical models in MUQ

- Physical & statistical models are implemented separately (`ModPieces`)

```
# create submodels
theta = VectorPassthroughModel()
prior = GaussianDensity(mu, sig)
f = ForwardModel()
like = GaussianDensity()
post = DensityProduct()
```

Graphical models in MUQ

- Physical & statistical models are implemented separately (`ModPieces`)
- Each component represents a piece of the overall model

  (e.g., prior, likelihood, forward model)

```
# create submodels
theta = VectorPassthroughModel()
prior = GaussianDensity(mu, sig)
f = ForwardModel()
like = GaussianDensity()
post = DensityProduct()
```

Graphical models in MUQ

- Physical & statistical models are implemented separately (`ModPieces`)
- Each component represents a piece of the overall model

  (e.g., prior, likelihood, forward model)

Separately implement submodels:

```
# create submodels
theta = VectorPassthroughModel()
prior = GaussianDensity(mu, sig)
f = ForwardModel()
like = GaussianDensity()
post = DensityProduct()
```

Parameters: θ

Forward model:
$f(\theta)$

Likelihood:
$\pi(d|f(\theta))$

Prior:
$\pi(\theta)$

Posterior:
$\pi(\theta|d) \propto \pi(d|f(\theta))\pi(\theta)$

Graphical models in MUQ

- Submodels are connected — becoming **graphs** (`ModGraphs`)

```
# Add graph nodes
G = ModGraph()
G.AddNode(theta, 'Parameter')
G.AddNode(prior, 'Prior')
G.AddNode(f, 'Forward Model')
G.AddNode(like, 'Likelihood')
G.AddNode(post, 'Posterior')
```

Parameters: θ

Forward model:
$f(\theta)$

Likelihood:
$\pi(d|f(\theta))$

Prior:
$\pi(\theta)$

Posterior:
$\pi(\theta|d) \propto \pi(d|f(\theta))\pi(\theta)$

Graphical models in MUQ

- Submodels are connected — becoming **graphs** (`ModGraphs`)
  - *Node:* A submodel (`ModPiece`)

```
# Add graph nodes
G = ModGraph()
G.AddNode(theta, 'Parameter')
G.AddNode(prior, 'Prior')
G.AddNode(f, 'Forward Model')
G.AddNode(like, 'Likelihood')
G.AddNode(post, 'Posterior')
```

Parameters: θ

Forward model:
$f(\theta)$

Likelihood:
$\pi(d|f(\theta))$

Prior:
$\pi(\theta)$

Posterior:
$\pi(\theta|d) \propto \pi(d|f(\theta))\pi(\theta)$

Graphical models in MUQ

- Submodels are connected — becoming **graphs** (`ModGraphs`)
  - *Node:* A submodel (`ModPiece`)
  - *Directed edge:* out/input of each submodel

```
# Add graph nodes
G = ModGraph()
G.AddNode(theta, 'Parameter')
G.AddNode(prior, 'Prior')
G.AddNode(f, 'Forward Model')
G.AddNode(like, 'Likelihood')
G.AddNode(post, 'Posterior')
```

Parameters: θ

Forward model:
$f(\theta)$

Likelihood:
$\pi(d|f(\theta))$

Prior:
$\pi(\theta)$

Posterior:
$\pi(\theta|d) \propto \pi(d|f(\theta))\pi(\theta)$

Graphical models in MUQ

- Submodels are connected — becoming **graphs** (`ModGraphs`)
  - *Node:* A submodel (`ModPiece`)
  - *Directed edge:* out/input of each submodel
- Exposes underlying structure

```
# Add graph nodes
G = ModGraph()
G.AddNode(theta, 'Parameter')
G.AddNode(prior, 'Prior')
G.AddNode(f, 'Forward Model')
G.AddNode(like, 'Likelihood')
G.AddNode(post, 'Posterior')
```

Parameters: θ

Forward model:
$f(\theta)$

Likelihood:
$\pi(d|f(\theta))$

Prior:
$\pi(\theta)$

Posterior:
$\pi(\theta|d) \propto \pi(d|f(\theta))\pi(\theta)$

Graphical models in MUQ

- Submodels are connected — becoming **graphs** (`ModGraphs`)
  - *Node:* A submodel (`ModPiece`)
  - *Directed edge:* out/input of each submodel
- Exposes underlying structure

Add submodels to the graph:

```
# Add graph nodes
G = ModGraph()
G.AddNode(theta, 'Parameter')
G.AddNode(prior, 'Prior')
G.AddNode(f, 'Forward Model')
G.AddNode(like, 'Likelihood')
G.AddNode(post, 'Posterior')
```

Parameters: θ

Forward model: $f(\theta)$

Prior: $\pi(\theta)$

Likelihood: $\pi(d|f(\theta))$

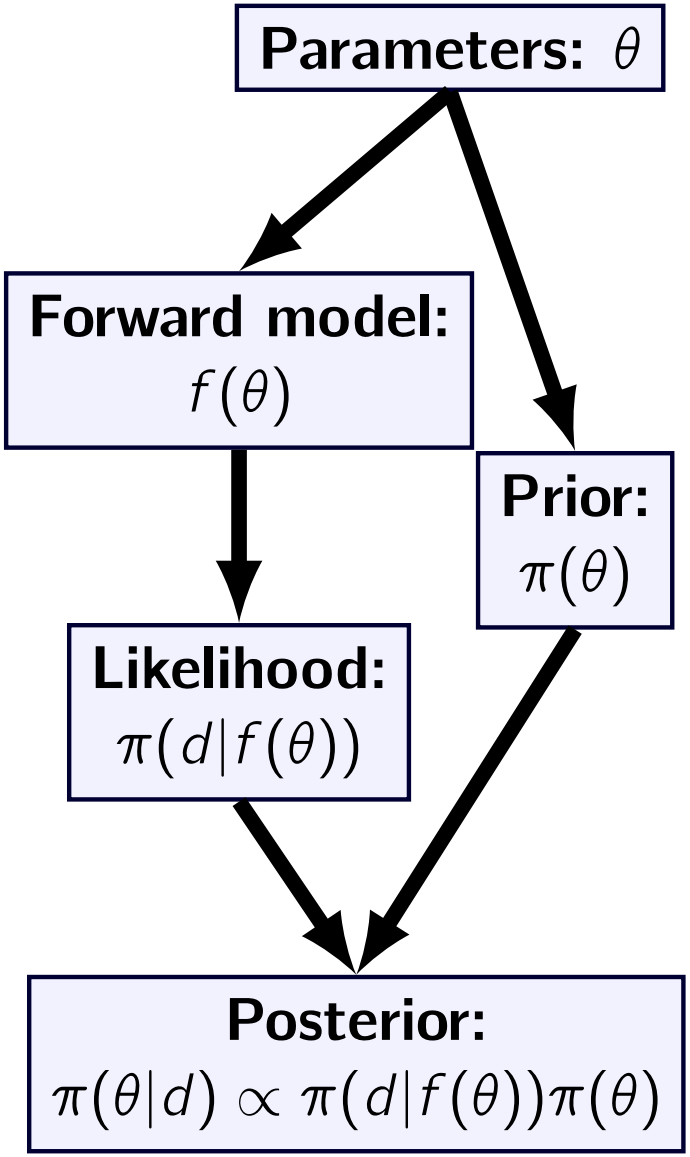
Posterior: $\pi(\theta|d) \propto \pi(d|f(\theta))\pi(\theta)$

Graphical models in MUQ

- Submodels are connected — becoming **graphs** (`ModGraphs`)
  - *Node:* A submodel (`ModPiece`)
  - *Directed edge:* out/input of each submodel
- Exposes underlying structure

Connect nodes with edges:

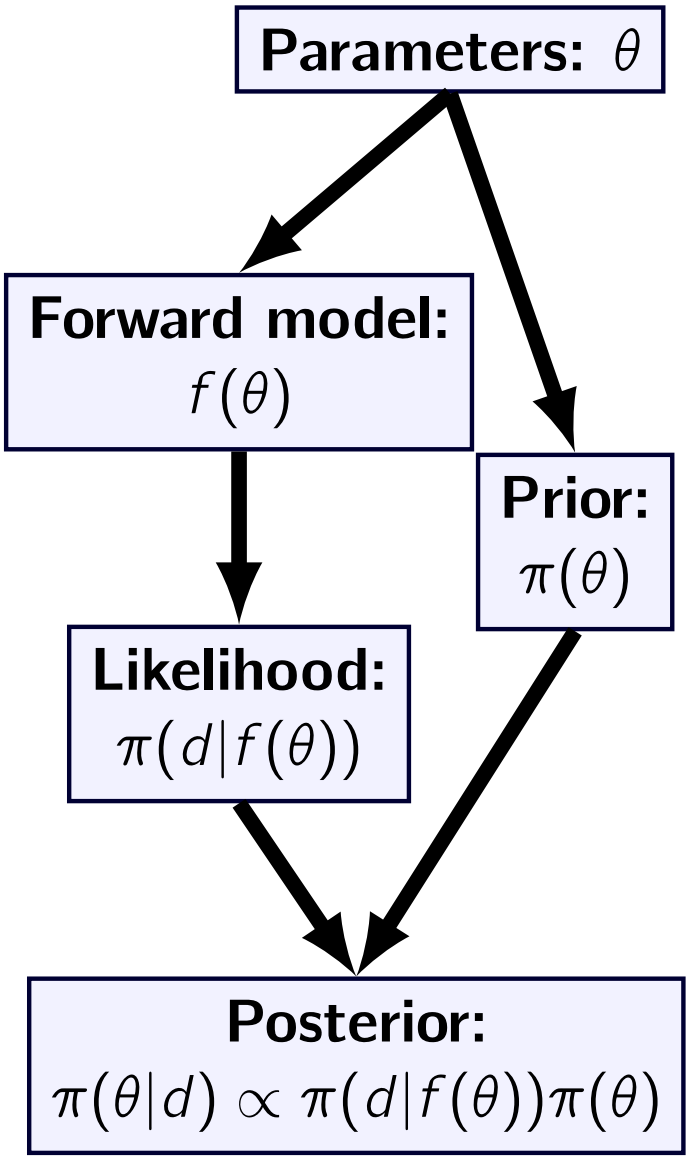
```
# Add graph edges
G.AddEdge('Parameters', 'Forward Model', 0)
G.AddEdge('Forward Model', 'Likelihood', 0)
G.AddEdge('Likelihood', 'Posterior', 0)

G.AddEdge('Parameters', 'Prior', 0)
G.AddEdge('Prior', 'Posterior', 1)
```



Advantages of graph-based modeling

1. Relates *implementation structure* to *intuitive interpretation*

```
# use a different forward model
g = AnotherForwardModel()
G.SwapNode('Forward Model', g)
```

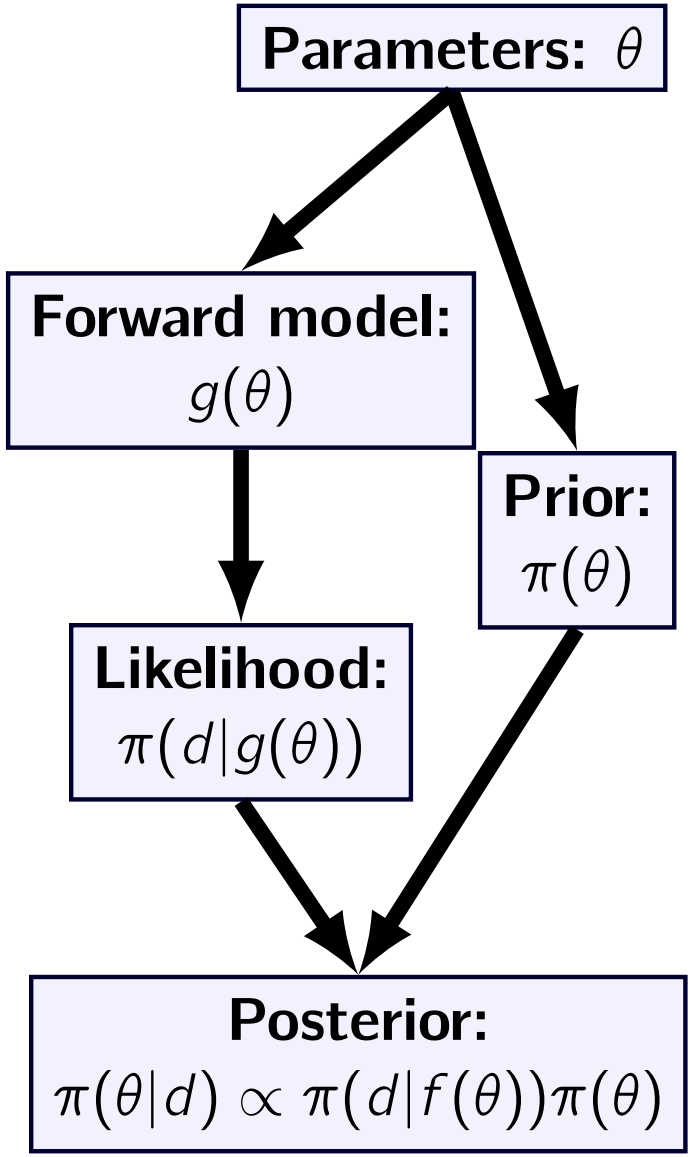


Advantages of graph-based modeling

1. Relates *implementation structure* to *intuitive interpretation*

2. Easy to **swap** *pieces of the problem formulation*

```
# use a different forward model
g = AnotherForwardModel()
G.SwapNode('Forward Model', g)
```



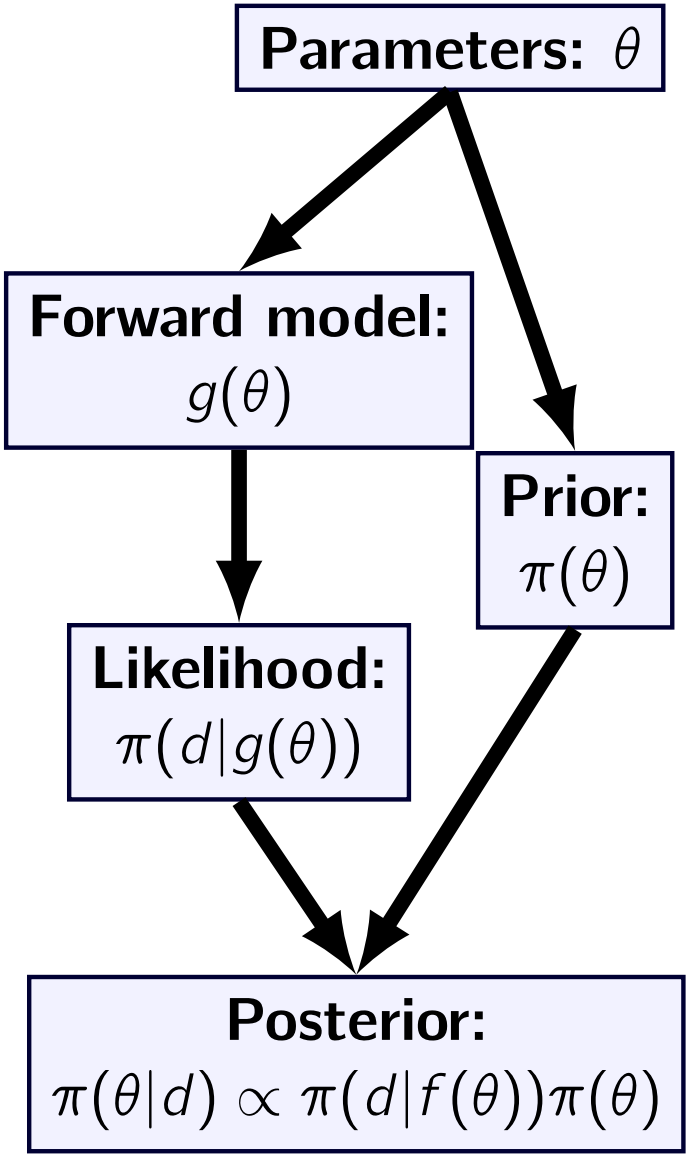
Advantages of graph-based modeling

1. Relates *implementation structure* to *intuitive interpretation*

2. Easy to **swap** *pieces of the problem formulation*

```
# use a different forward model
g = AnotherForwardModel()
G.SwapNode('Forward Model', g)
```

3. "Grey-box" wrapper around existing models



Grey-box wrapper: Using existing models (C++)

- Existing implementation becomes a submodel
- The user only needs to write a **MUQ/existing solver interface:**

```
class ModelWrapper : public muq::Modeling::ModPiece {
  public:
    // A way to create the submodel
    ModelWrapper(inputSizes, outputSize) :
       ModPiece(inputSizes, outputSize)

  private:
    // Grey-box implementation
    Eigen::VectorXd EvaluateImpl(std::vector<Eigen::VectorXd>
        const& inputs) {
      Eigen::VectorXd output(outputSize);
      /* ***** CALL EXISTING CODE HERE ***** */
      return output;
    }
};
```

Grey-box wrapper: Using existing models (Python)

- Existing implementation becomes a submodel
- The user only needs to write a **MUQ/existing solver interface:**

```
class ModelWrapper(libmuqModelling.ModPiece):
    # a way to create the submodel
    def __init__(self, inputSizes, outputSize):
        libmuqModelling.ModPiece.__init__(inputSizes, outputSize)

    # Grey-box implementation
    def EvaluateImpl(self, inputs):
        # #### CALL EXISTING CODE HERE #### #
        # existing code returns output (list type)
        return output
```

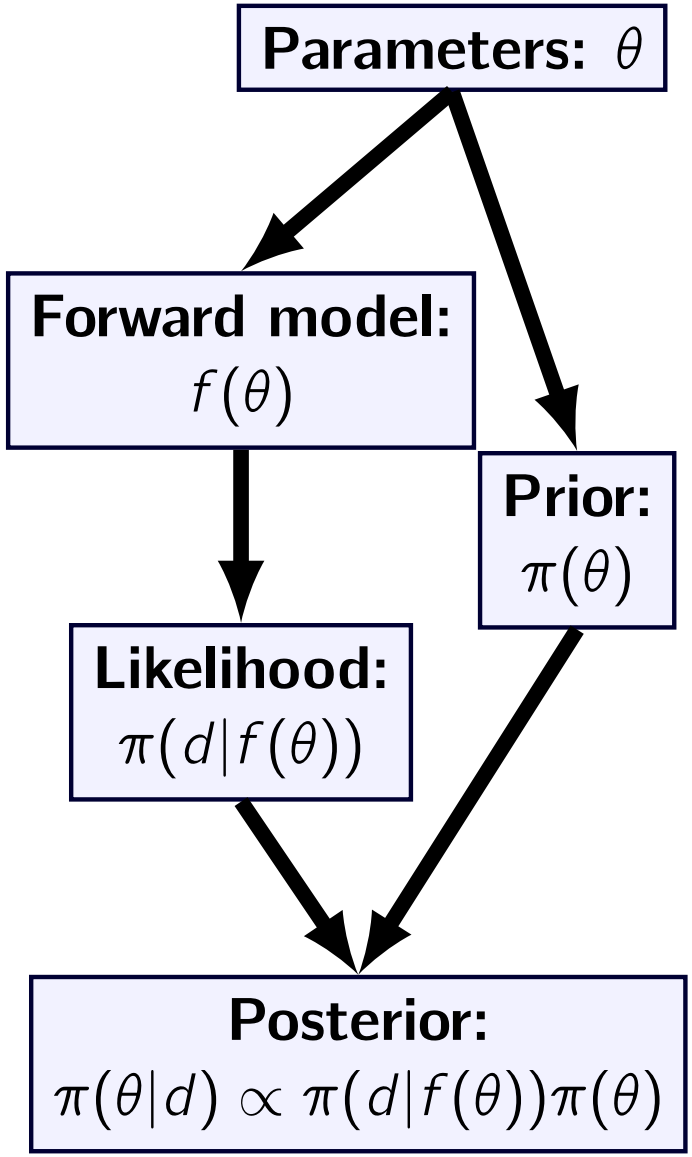
Derivative implementation (C++)

- User may optionally implement derivative information
  Gradient, Jacobian, Jacobian's action, and/or Hessian
- The user only needs to write a **MUQ/existing solver interface:**

```
class ModelWrapper : public muq::Modeling::ModPiece {
    // Model implementation
    Eigen::VectorXd EvaluateImpl(std::vector<Eigen::VectorXd>
        const& inputs) { return output; }
    // Derivative implementation

    virtual Eigen::VectorXd
        GradientImpl(std::vector<Eigen::VectorXd> const& inputs,
        Eigen::VectorXd const& sensitivity,
        int const inputDimWrt) override {
            return gradient;
        }
};
```

Model derivatives

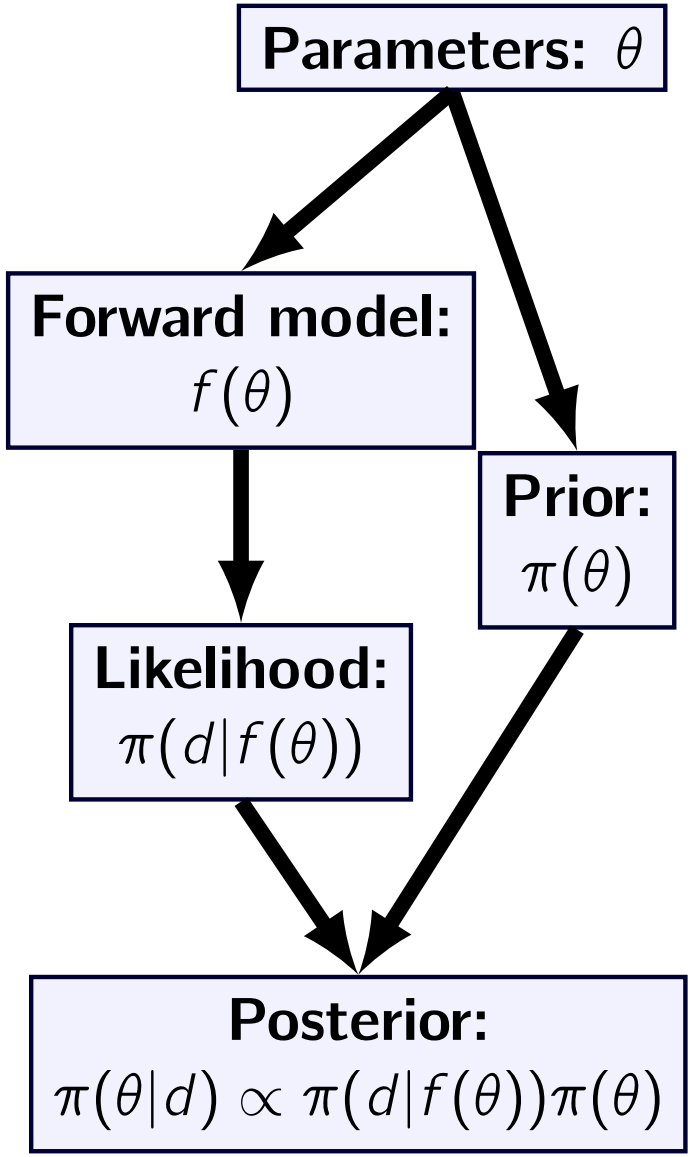
1. Option to include user implemented derivatives



Model derivatives

1. Option to include user implemented derivatives

2. Defaults to finite difference if analytic derivatives are not implemented

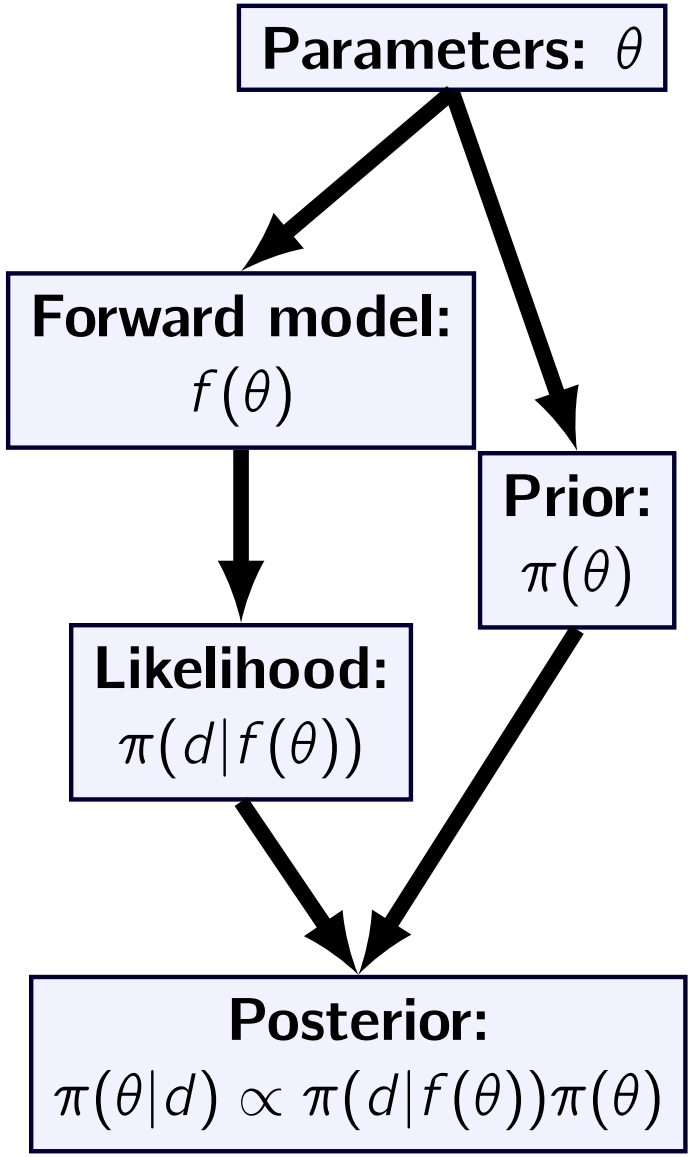


Model derivatives

1. Option to include user implemented derivatives

2. Defaults to finite difference if analytic derivatives are not implemented

3. Computes derivative of the graph's output via the chain rule



Built-in modeling tools

1. Extensible submodel class (`ModPiece`)

Built-in modeling tools

1. Extensible submodel class (`ModPiece`)
2. Implemented common models:
   - Component-wise functions (`exp`, `sin`, ...)
   - Component-wise sum
   - Linear solver $\mathbf{Ax} = \mathbf{b}$ (using `Eigen`)
   - *and many more* ...

Built-in modeling tools

1. Extensible submodel class (`ModPiece`)
2. Implemented common models:
   - Component-wise functions (`exp`, `sin`, ...)
   - Component-wise sum
   - Linear solver $\mathbf{Ax} = \mathbf{b}$ (using `Eigen`)
   - *and many more* ...
3. `Sundials` interface
   - Numerical integration of ODEs, state-discretized PDEs
   - Root-finding algorithms

Built-in modeling tools

1. Extensible submodel class (`ModPiece`)
2. Implemented common models:
   - Component-wise functions (`exp`, `sin`, ...)
   - Component-wise sum
   - Linear solver $\mathbf{Ax} = \mathbf{b}$ (using `Eigen`)
   - *and many more* ...
3. `Sundials` interface
   - Numerical integration of ODEs, state-discretized PDEs
   - Root-finding algorithms
4. Transient and steady PDE solver
   - User must implement the *weak form*
   - Derivative information computed via automatic differentiation (`Sacado`)
   - *Finite element method* spatial discretization (`LibMesh`)

MUQ: outline

1. **Modeling framework**
   - Constructing combined (physical + statistical) models with high-level structure
   - Graphical modeling technique to easily combine sub models

2. **Algorithm framework**
   - *Example:* Markov chain Monte Carlo stack — using mathematical structure in software design

3. **MUQ development update**
   - Use of MUQ in our research

MUQ: MCMC

Question

What are the fundamental components of an MCMC algorithm?

Metropolis-Hastings MCMC:

1. Save current state
$$\theta^{(k)} = \theta$$
2. Propose new state
$$\theta' \sim q(\theta'|\theta)$$
3. Accept or reject
$$\alpha = \min\left\{1, \frac{\pi(\theta')}{\pi(\theta)}\frac{q(\theta|\theta')}{q(\theta'|\theta)}\right\}$$

MUQ: MCMC

Question

What are the fundamental components of an MCMC algorithm?

Metropolis-Hastings MCMC:

1. Save current state

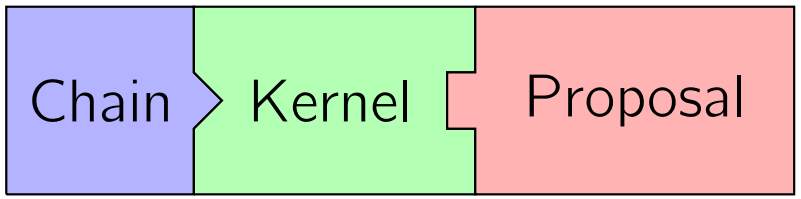
$$\theta^{(k)} = \theta$$

2. Propose new state

$$\theta' \sim q(\theta'|\theta)$$

3. Accept or reject

$$\alpha = \min\left\{1, \frac{\pi(\theta')}{\pi(\theta)}\frac{q(\theta|\theta')}{q(\theta'|\theta)}\right\}$$



MUQ: MCMC

Question

What are the fundamental components of an MCMC algorithm?

Metropolis-Hastings MCMC:

1. Save current state

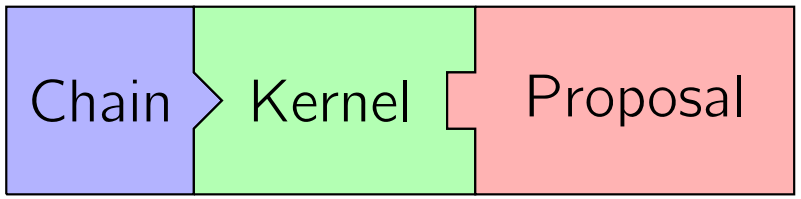
$$\theta^{(k)} = \theta$$

2. Propose new state

$$\theta' \sim q(\theta'|\theta)$$

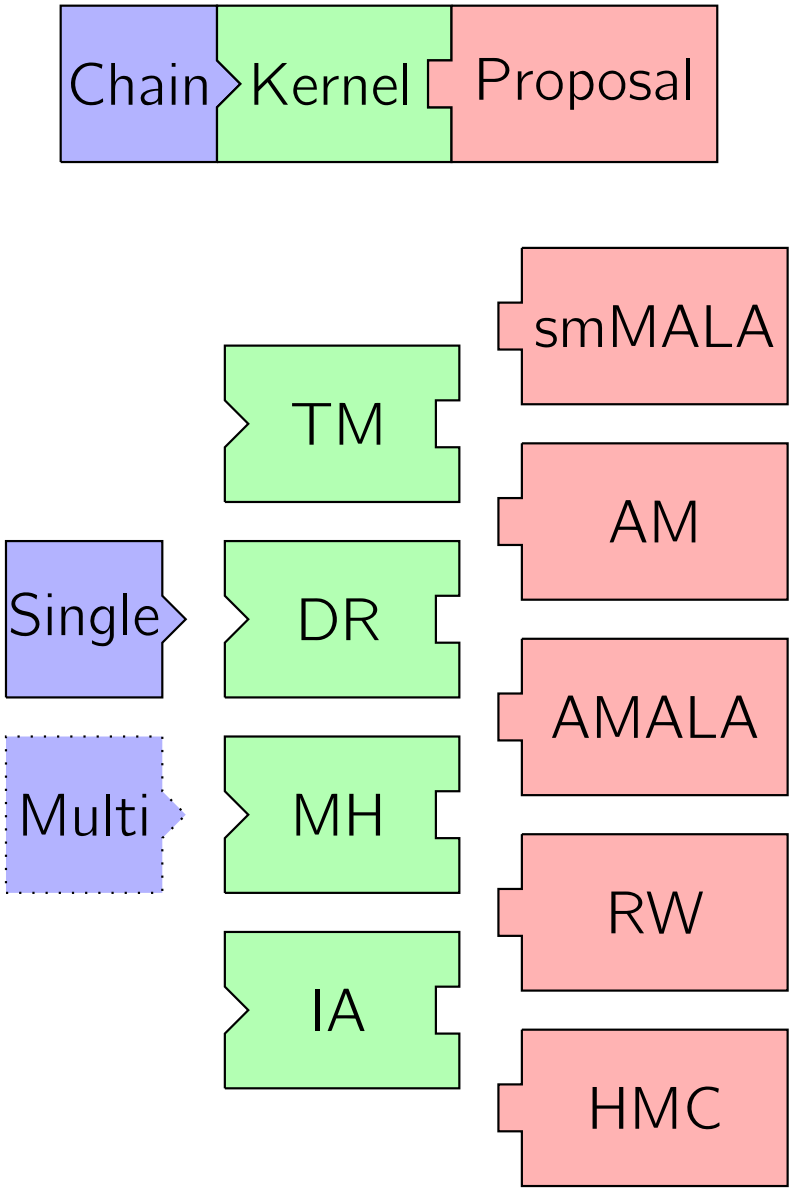
3. Accept or reject

$$\alpha = \min\left\{1, \frac{\pi(\theta')}{\pi(\theta)}\frac{q(\theta|\theta')}{q(\theta'|\theta)}\right\}$$



- We copy this structure with C++ classes
- Easily extendable
- Many possible configurations (> 30)

MUQ: full MCMC framework



Advantages:

- New proposals used with any kernel
- Focus efforts on *new* features
- Changing components is trivial
- Extensive performance comparisons are straightforward

```
// MH + AMALA:
ps.put("MCMC.Kernel","MH");
ps.put("MCMC.Proposal","AMALA");

// TM + RW:
ps.put("MCMC.Kernel","TransportMap");
ps.put("MCMC.Proposal","MHProposal");
```

MCMC and models

Observation:

Many MCMC algorithms rely on specific problem structure.

Solution:

Use problem classes that can exploit graphical representations.

MCMC and models

Observation:
Many MCMC algorithms rely on specific problem structure.

Solution:
Use problem classes that can exploit graphical representations.

Problem iterates through graph and extracts important structure:

- Gaussian prior and error model
- Linear models
- Gauss-Newton Hessians
- etc. . .

MCMC and models

Observation:
Many MCMC algorithms rely on specific problem structure.

Solution:
Use problem classes that can exploit graphical representations.

Problem iterates through graph and extracts important structure:

- Gaussian prior and error model
- Linear models
- Gauss-Newton Hessians
- etc. . .

No extra effort: **Structure automatically defined in "grey-box" model.**

Beyond MCMC

Lessons learned:

- Design code to mimic the mathematics
  - Allows us easily add new components without duplicate code
  - Makes algorithms more configurable and intuitive

Beyond MCMC

Lessons learned:

- Design code to mimic the mathematics
  - Allows us easily add new components without duplicate code
  - Makes algorithms more configurable and intuitive
- Use gray-box for automated structure extraction
  - Enables algorithms to exploit high-level structure
  - Still allows arbitrary model implementations (e.g., legacy code)

Beyond MCMC

Lessons learned:

- Design code to mimic the mathematics
  - Allows us easily add new components without duplicate code
  - Makes algorithms more configurable and intuitive
- Use gray-box for automated structure extraction
  - Enables algorithms to exploit high-level structure
  - Still allows arbitrary model implementations (e.g., legacy code)

Notes:

- Similar structure exists in our optimization algorithms
- More complicated chains being tested (DILI, DILI+maps, etc.)

MUQ: outline

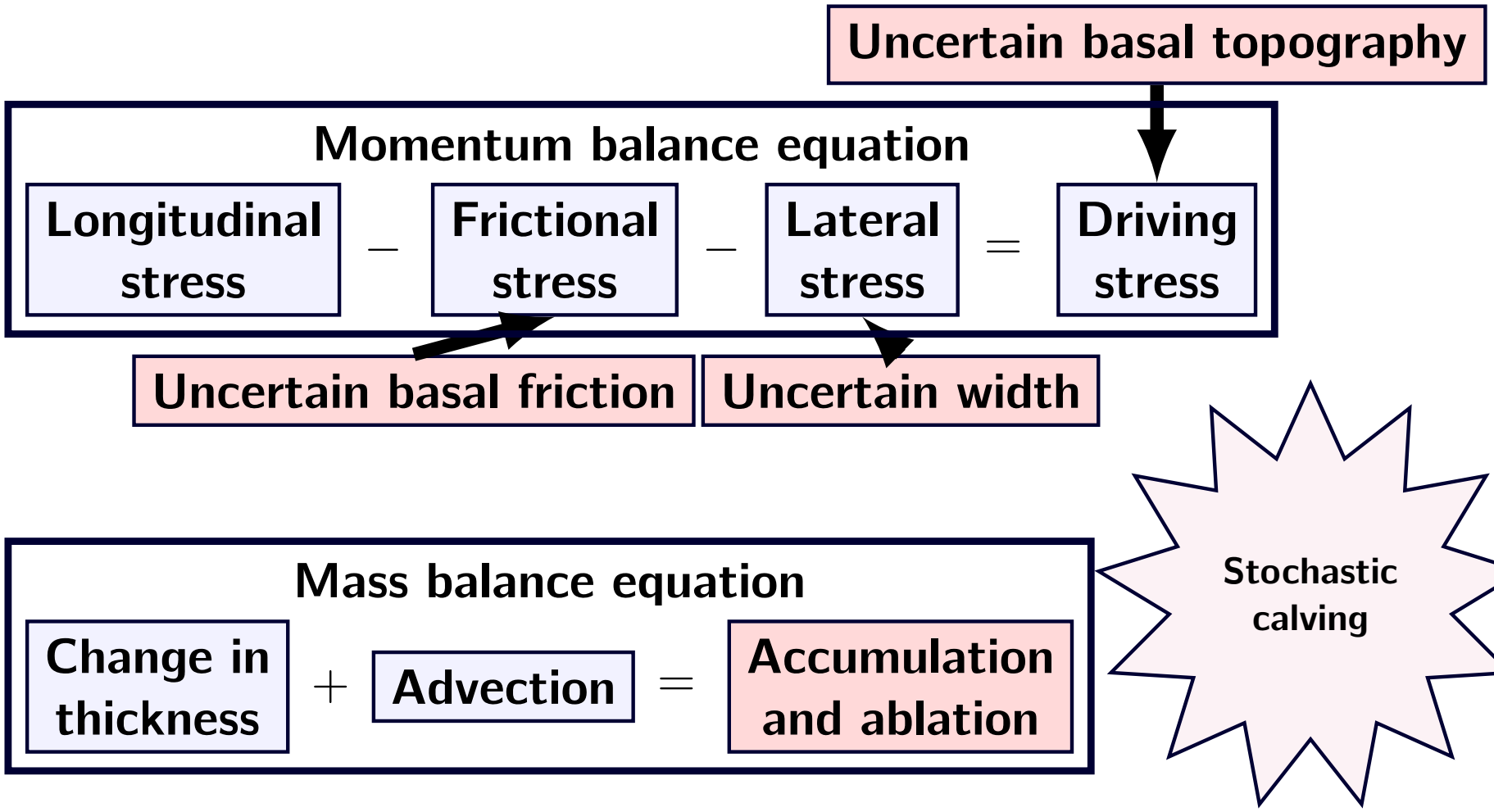
1. **Modeling framework**
   - Constructing combined (physical + statistical) models with high-level structure
   - Graphical modeling technique to easily combine sub models

2. **Algorithm framework**
   - *Example:* Markov chain Monte Carlo stack — using mathematical structure in software design

3. **MUQ development update**
   - Use of MUQ in our research

MUQ usage: ice sheet modeling

Solve conservation equations with *uncertain input parameters* . . .



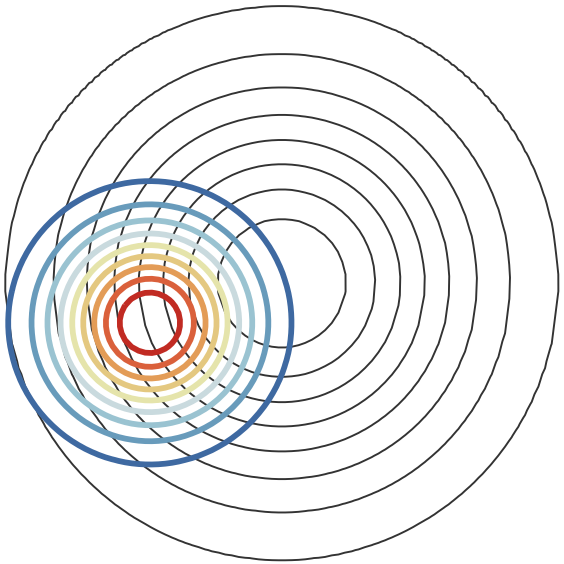
. . . computed field are used within UQ algorithms (e.g., Monte Carlo, optimization, MCMC, importance sampling ect. . . .)

MUQ usage: map-accelerated MCMC

Use nonlinear variable transformation to "precondition" target density.

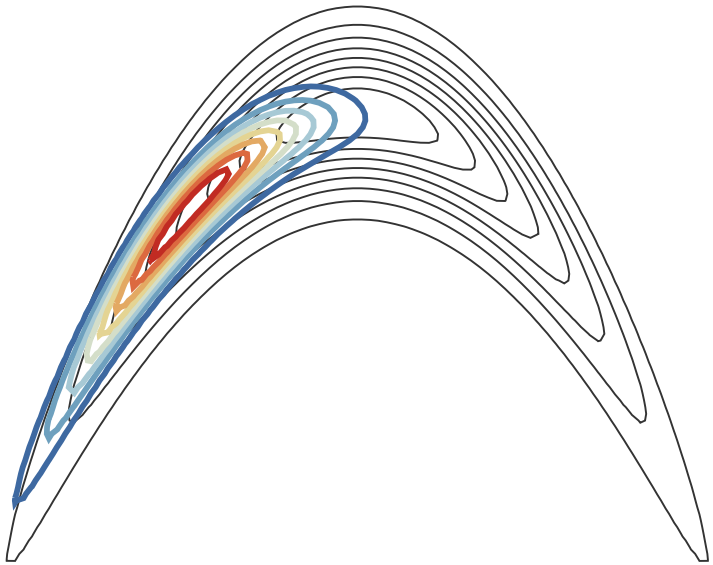
reference proposal

$q_r(r'|r) = N(r, \sigma^2 I)$



mapped proposal

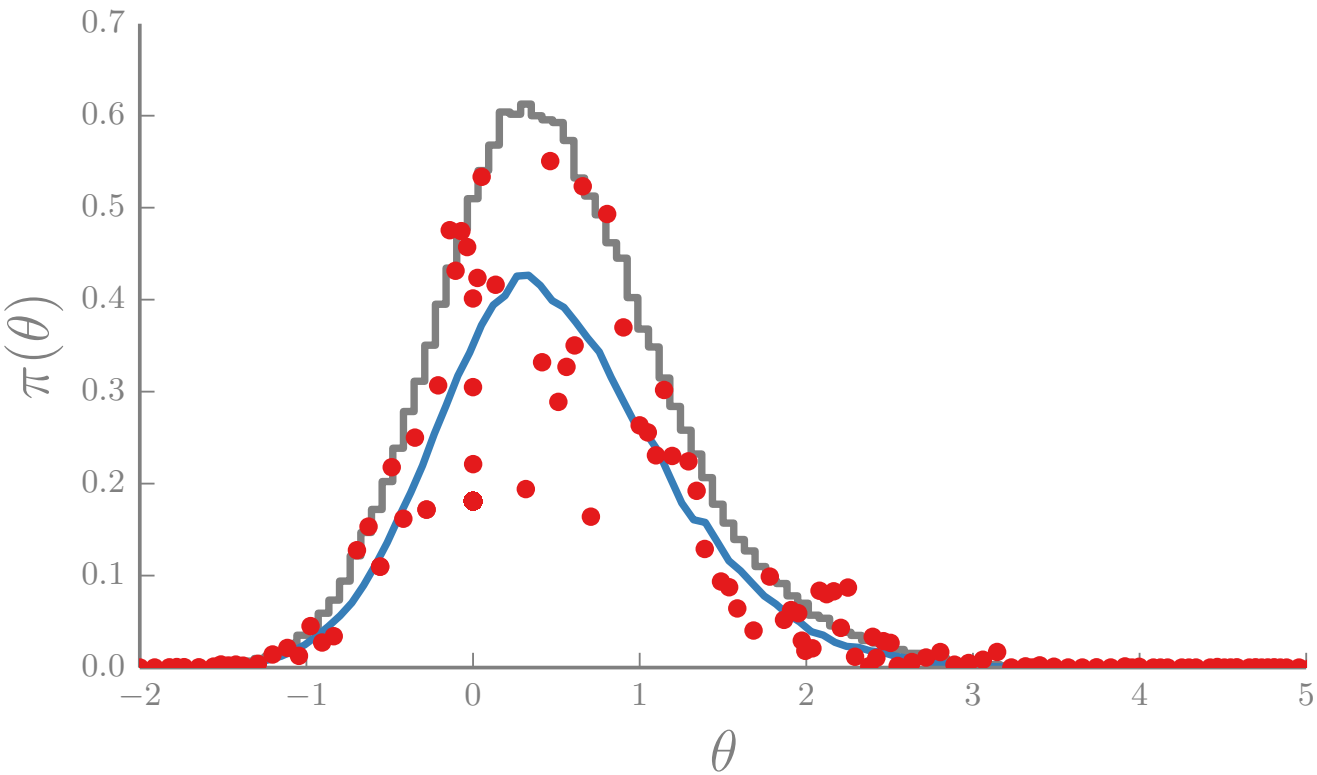
$q_\theta(\theta'|\theta) = q_r\left(\tilde{T}(\theta')|\tilde{T}(\theta)\right)\left|\det D\tilde{T}(\theta')\right|$



The exchangeability of MCMC components makes testing many reference proposals easy

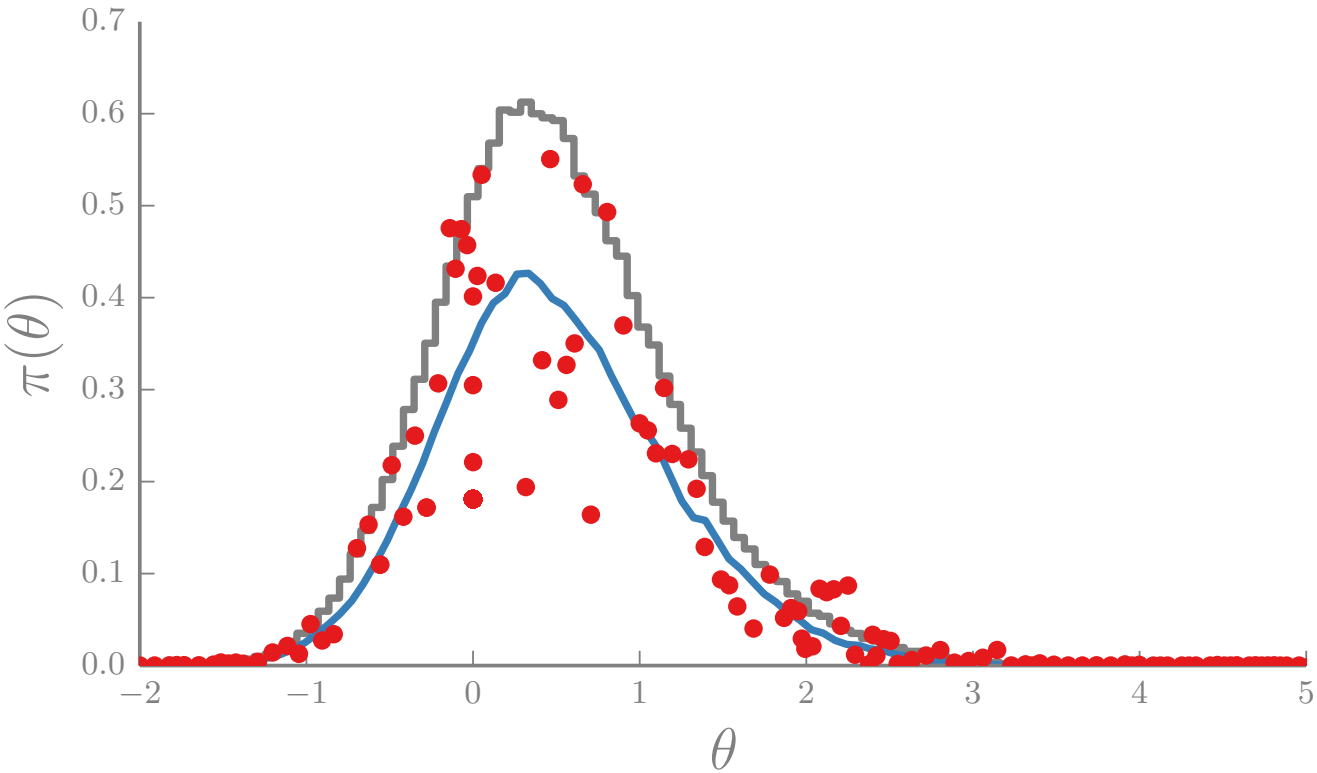
MUQ usage: characterizing marginal distributions

- Build a surrogate model of the marginal using stochastic evaluations



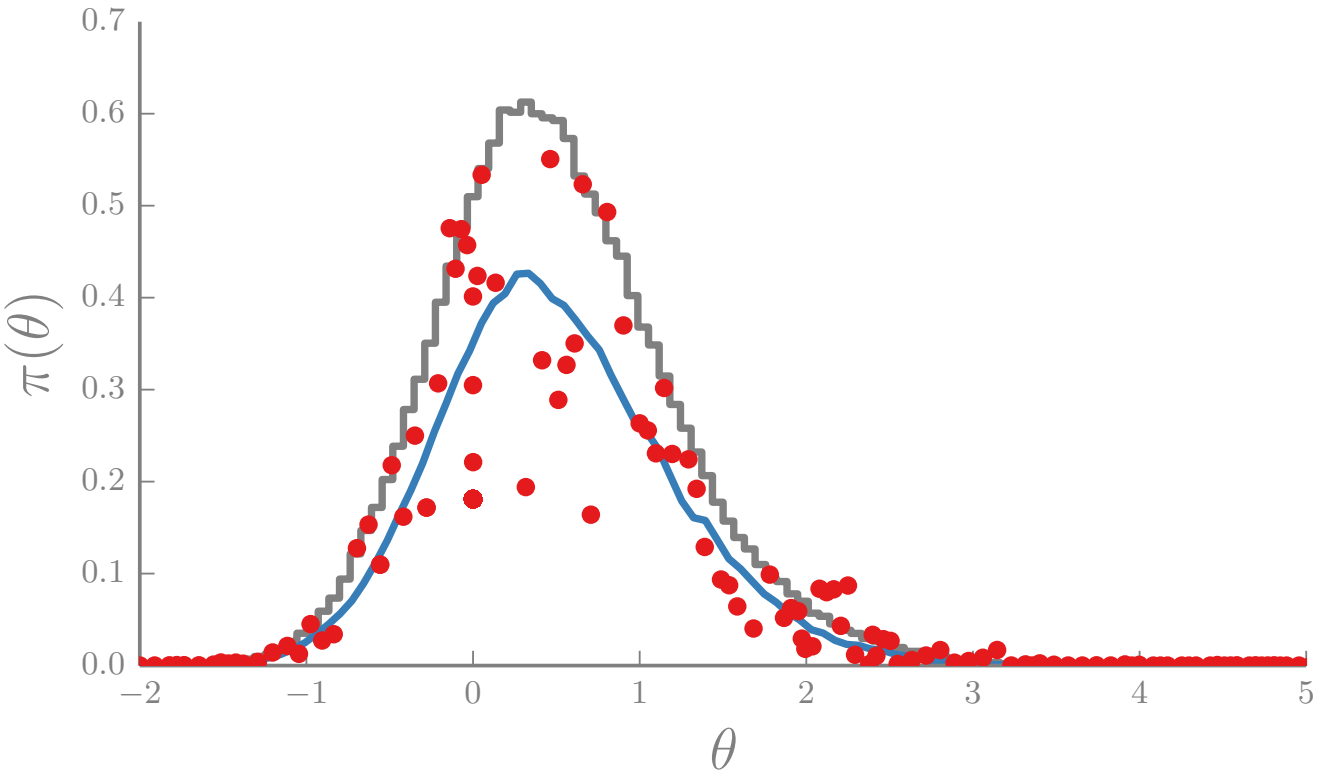
MUQ usage: characterizing marginal distributions

- Build a surrogate model of the marginal using stochastic evaluations
- Using the surrogate to evaluate the acceptance ratio does **not** affect the proposal



MUQ usage: characterizing marginal distributions

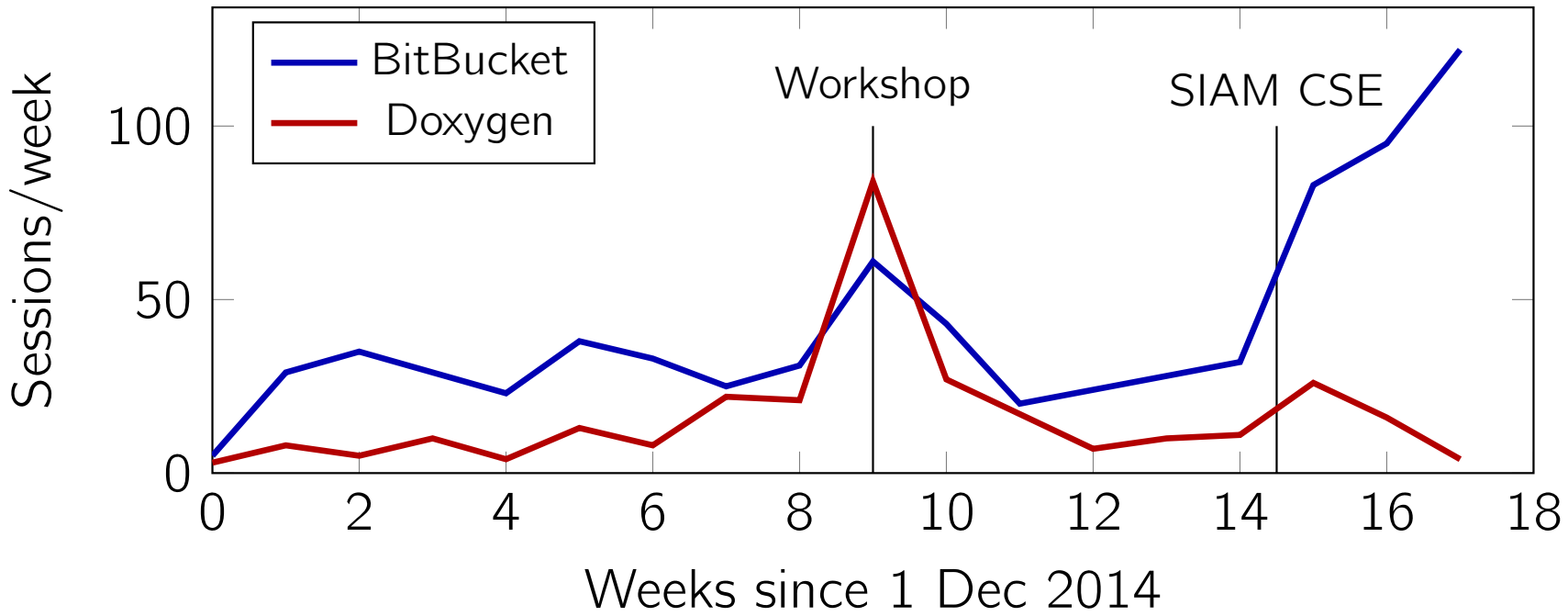
- Build a surrogate model of the marginal using stochastic evaluations
- Using the surrogate to evaluate the acceptance ratio does **not** affect the proposal
- We can easily combine many proposal methods (e.g., MH, AM, DRAM, . . .) with surrogate modeling



MUQ: development notes

Development improvements:

Google analytics:

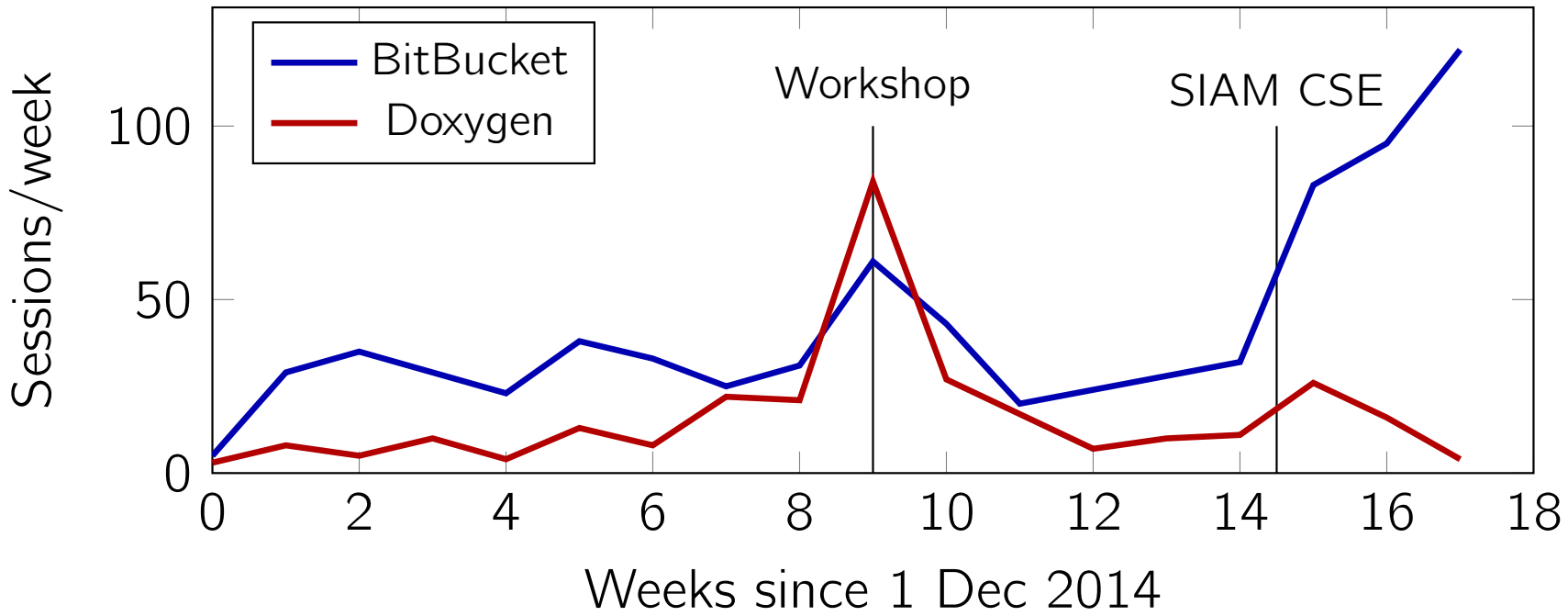


MUQ: development notes

Development improvements:

- Improved build system
  - `cmake` automatically downloads and installs dependencies

Google analytics:

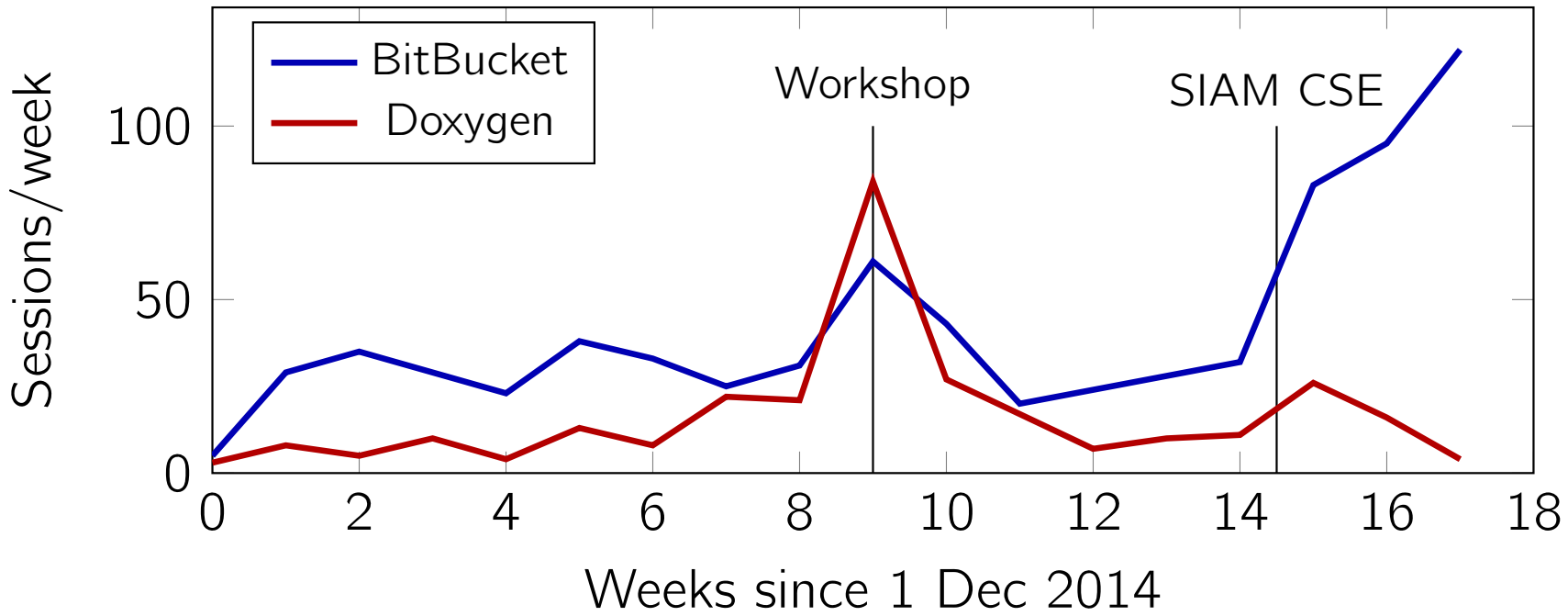


MUQ: development notes

Development improvements:

- Improved build system
  - `cmake` automatically downloads and installs dependencies
- Extensive testing for corrent and/or consistent perfermance
  - Automated code testing for verification with Jenkins

Google analytics:



muq.mit.edu