QUESO tutorial

Damon McDougall

Centre for Predictive Engineering and Computational Sciences
Institute for Computational Engineering and Sciences
The University of Texas at Austin

25th of March, 2016
Outline

1. History, design, and other software
2. Vectors and matrices
3. Generating random numbers
4. Defining a distribution
5. Implementing your likelihood
6. Defining the posterior
7. Analysing output
8. Assessing convergence
9. Contributing to QUESO
10. The future of QUESO
**QUESO**

Nutshell: QUESO gives samples from $\mathbb{P}(\theta|y)$ (called MCMC)

- Library for Quantifying Uncertainty in Estimation, Simulation and Optimisation
- Born in 2008 as part of PECOS PSAAP programme
- Provides robust and scalable sampling algorithms for UQ in computational models
- Open source
- C++
- MPI for communication
- Parallel chains, each chain can house several processes
- Dependencies are MPI, Boost and GSL. Other optional features exist
- [http://libqueso.com](http://libqueso.com)
Contributors (four new since v0.50.0)

Brian Adams
Paul Bauman
Morgan Bruns
Sai Hung Cheung
Kemelli Estacio-Hiroms
Nicholas Malaya
André Maurente
Damon McDougall
Kenji Miki
Rebecca Morrison

Todd Oliver
Sylvain Plessis
Teresa Portone
Ernesto Prudencio
Karl Schulz
Roy Stogner
Gabriel Terejanu
Rhys Ulerich
Rochan Upadhyay
Eric Wright
Why use QUESO?

Other solutions are available, e.g. R, PyMC, emcee, MICA, Stan, MUQ.

QUESO solves the same problem, but:

- Has been designed to be used with large forward problems
- Has been used successfully with 5000+ cores
- Leverages parallel MCMC algorithms
- Supports for finite and infinite dimensional problems
Preliminaries

- Instructions on how to build and install QUESO: http://libqueso.com
- All QUESO docs are here: http://libqueso.com/queso/html/.
- Links on these slides are clickable
- I’m British, so I spell things weird. QUESO uses American English.
  - It’s a hard habit for me to break, so be cognizant of potential spelling differences in the code on these slides
  - Try to compile stuff as we go along
- These slides will not take four hours to deliver:
  - There are hands-on tasks you can do. Please do them.
  - I encourage interruptions for questions or comments.
- There are some areas of QUESO I am not familiar with. I will only talk about the areas I am familiar with.
The environment

- All classes are available in the QUESO namespace.
- Most of the objects in QUESO need an *environment*.
- Some objects can infer it from other objects.
- Lives in `queso/Environment.h` and called `FullEnvironment`.

```cpp
#include <queso/Environment.h>

int main(int argc, char ** argv)
{
    MPI_Init(&argc, &argv);
    QUESO::FullEnvironment env(MPI_COMM_WORLD, argv[1], "", NULL);
    MPI_Finalize();
    return 0;
}
```
The environment

QUESO::FullEnvironment env(MPI_COMM_WORLD, argv[1], "", NULL);

Arguments:

1. MPI communicator.
2. Path to QUESO input file.
3. Prefix string for entries in the input file.
4. Optional programmatically provided input options.
Task 1

- Create a source file and instantiate a FullEnvironment.
- Compile it, link it against QUESO and run it.
- Make sure there are no errors.
- You might need an input file. You can find one here: https://github.com/libqueso/queso/blob/dev/examples/template_example/template_example_input.txt
- We’ll talk about the input file later. For now you can take it for granted.
- This task should take less than five minutes.
Vectors

- Vectors live in a vector space.
- Most classes in QUESO are templated around vector implementation.
- The default vector/matrix implementation is GSL.

```
QUESO::VectorSpace<> paramSpace(env, "param_", 1, NULL);
```

Arguments:

- The environment.
- The prefix for input file items.
- The dimension of the vector space.
- A pointer to a `std::vector` of `std::strings` for naming components of the vectors. Good for ‘naming’ parameters.
Vectors

In QUESO v0.53.x, the following classes are equivalent:

QUESO::VectorSpace<>
QUESO::VectorSpace<QUESO::GslVector, QUESO::GslMatrix>

• The default template type is QUESO::GslVector/QUESO::GslMatrix.

• The QUESO development team are working closely with the DAKOTA team at Sandia to provide a wider range of high-performance vector/matrix implementations in the 12–15 month timeline.

• If you really don’t like QUESO:: everywhere, you can insert
using namespace QUESO;

in your code and remove all instances of QUESO:: elsewhere.
The zeroVector method returns a GslVector object belonging to the vector space with every component set to zero.

```cpp
paramSpace.zeroVector();
```

Use the vector space to eke out the zero vector, then copy it.

```cpp
QUESO::GslVector myVector(paramSpace.zeroVector());
```

Argument is another GslVector object.
Vectors

Set element 3:

myVector[2] = 1.0;

Get element 2:

double myElement = myVector[1];

Set every element to 5.0:

myVector.cwSet(5.0);

For other methods, see
**Matrices**

- All matrices are square.
- Obviously not true, but most of the time we only care about square (covariance) matrices.
- You can create nonsquare matrices, but I doubt you will ever need one in QUESO unless your forward problem needs it.
- Matrices don’t *live* in the vector space, but they are defined over it.
- Use an existing `GslVector` to construct a square matrix with compatible dimension.

**QUESO:**

```cpp
GslMatrix myMatrix(myVector);
```

- Creates diagonal matrix with element \((i, i)\) equal to `myVector[i]`.
- Argument is a `GslVector` object but could be another `GslMatrix`, in which case the whole matrix is copied.
Matrices

Set element (3, 4):

\[ \text{myMatrix}(2, 3) = 1.0; \]

Get element (2, 3):

\[ \text{double myElement} = \text{myMatrix}(1, 2); \]

- QUESO has methods that can factorise matrices in order to solve linear systems.
- Linear algebra all relies on GSL.
- See above comment about ongoing work for high-performance vector/matrix implementations.
- For other methods, see
  \[ \text{http://libqueso.com/queso/html/a00127.html} \]
Task 2

- Create a GslVector.
- Create a GslMatrix.
- Multiply the vector by the matrix.
- Verify the result of the multiplication is correct.
- If you complete this quickly, look at the documentation and play with some of the other methods.
- This task should take less than ten minutes.
Parameter domains

- Random variables are defined on a ‘state space’ or ‘domain’.
- State space may be a proper subset of the whole vector space `paramSpace`.
- Most domains are boxes.
- More exotic domains are not supported out of the box.
- Creating your own will not be covered here.

```cpp
QUESO::GslVector paramMins(paramSpace.zeroVector());
QUESO::GslVector paramMaxs(paramSpace.zeroVector());
paramMins.cwSet(0.0);
paramMaxs.cwSet(1.0);

QUESO::BoxSubset<> paramDomain("param_", paramSpace, paramMins, paramMaxs);
```
Parameter domains

QUESO::BoxSubset<> paramDomain("param_", paramSpace,
    paramMins, paramMaxs);

Arguments are:

1. Prefix for input file entries.
2. The underlying vector space that elements of the domain belong to.
3. A vector of the minimum values in each dimension.
4. A vector of the maximum values in each dimension.

**Note:** Passing \texttt{INFINITY} (\texttt{-INFINITY}) as the maximum (minimum) of the domain limits is allowed.
Random variables

- **QUESO** supports the following random variables:
  - Inverse Gamma
  - Jeffreys
  - Log Normal
  - Beta
  - Gamma
  - Gaussian
  - Uniform
  - Wigner

- **QUESO** also supports creating custom random variables.
Random variables

- Random variables need the domain, and possibly other parameters
- Uniform:
  ```cpp
  QUESO::UniformVectorRV<> priorRv("prior_", paramDomain);
  ```
- Beta needs a vector of $\alpha$ and $\beta$ parameters as well:
  ```cpp
  QUESO::BetaVectorRV<> betaRv("prior_", paramDomain, alphas, betas);
  ```
- Gaussians need a mean and covariance:
  ```cpp
  QUESO::GaussianVectorRV<> gaussianRv("prior_", paramDomain, mean, cov);
  ```
- Unless it's obvious how to correlated elements of the vector, they are assumed to be independent.
Random variables

• *VectorRV classes have two major components:
  ▶ A *JointPdf object to evaluate the underlying density.
  ▶ A *VectorRealizer component to draw realisations.

• You can access these components with the pdf() method...

  QUESO::GslVector point(paramSpace.zeroVector());
betaRv.pdf().actualValue(point, NULL, NULL, NULL, NULL);

• ...or the realizer() method

  QUESO::GslVector draw(paramSpace.zeroVector());
betaRv.realizer().realization(draw);
Task 3

1. Create a Gaussian random variable over some domain.

2. Think about your domain for a few minutes.

3. Compute (and print) the empirical mean and variance (see point 2.) from $10^6$ realisations.

4. Are they (approximately) what you expect them to be? If not, why not?

5. This task should take less than fifteen minutes.
Likelihoods

• We now know enough to define our own prior random variable.

• For standard MCMC, we don’t need realisations from the prior. Only need pdf evaluations.

• Since all random variables in QUESO have a pdf() method, that’s what QUESO will use to evaluate the prior pdf when sampling.

• Up next is the likelihood.

• There are no out-of-the-box likelihoods in QUESO, because they need the forward problem!

• We will write an example forward problem first, and then create a likelihood function in QUESO to evaluate the forward problem.

• Then we’ll give the prior and likelihood to QUESO.

• QUESO will then solve the Bayesian problem.
General framework

Model (usually a PDE): $G(\theta)$ where $\theta$ are model parameters.

Observations:

$$y = G(\theta) + \eta, \quad \eta \sim \mathcal{N}(0, R)$$

Want:

$$p(\theta|y) \propto p(y|\theta)p(\theta)$$

$$\propto \exp \left( -\frac{1}{2} \| G(\theta) - y \|_R^2 \right) \exp \left( -\frac{1}{2} \| \theta - \bar{\theta} \|_{}^2 \right)$$

**Note:** The last proportionality is only true for Gaussian noise and Gaussian prior.
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?
What does MCMC look like?

\[ E(\theta|y) \approx \frac{1}{N} \sum_{k=1}^{N} \theta_k \]
How to do MCMC? Sampling $p(\theta | y)$

- Idea: Construct $\{\theta_k\}_{k=1}^{\infty}$ cleverly such that $\{\theta_k\}_{k=1}^{\infty} \sim p(\theta | y)$

1. Let $\theta_j$ be the ‘current’ state in the sequence and construct a proposal, $z \sim q(\theta_j, \cdot)$
How to do MCMC? Sampling $p(\theta|y)$

- Idea: Construct $\{\theta_k\}_{k=1}^{\infty}$ cleverly such that $\{\theta_k\}_{k=1}^{\infty} \sim p(\theta|y)$

1. Let $\theta_j$ be the ‘current’ state in the sequence and construct a proposal, $z \sim q(\theta_j, \cdot)$

2. Compute $\alpha(\theta_j, z) = 1 \land \frac{p(z|y)q(z, \theta_j)}{p(\theta_j|y)q(\theta_j, z)}$
How to do MCMC? Sampling \( p(\theta | y) \)

- Idea: Construct \( \{ \theta_k \}_{k=1}^{\infty} \) cleverly such that \( \{ \theta_k \}_{k=1}^{\infty} \sim p(\theta | y) \)
  
1. Let \( \theta_j \) be the ‘current’ state in the sequence and construct a proposal, \( z \sim q(\theta_j, \cdot) \)

2. Compute \( \alpha(\theta_j, z) = 1 \wedge \frac{p(z|y)q(z,\theta_j)}{p(\theta_j|y)q(\theta_j,z)} \)

3. Let

\[
\theta_{j+1} = \begin{cases} 
\theta & \text{with probability } \alpha(\theta_j, z) \\
\theta_j & \text{with probability } 1 - \alpha(\theta_j, z)
\end{cases}
\]
How to do MCMC? Sampling $p(\theta | y)$

- Idea: Construct $\{\theta_k\}_{k=1}^{\infty}$ cleverly such that $\{\theta_k\}_{k=1}^{\infty} \sim p(\theta | y)$

1. Let $\theta_j$ be the ‘current’ state in the sequence and construct a proposal, $z \sim q(\theta_j, \cdot)$

2. Compute $\alpha(\theta_j, z) = 1 \wedge \frac{p(z | y)q(z, \theta_j)}{p(\theta_j | y)q(\theta_j, z)}$

3. Let

$$\theta_{j+1} = \begin{cases} 
\theta & \text{with probability } \alpha(\theta_j, z) \\
\theta_j & \text{with probability } 1 - \alpha(\theta_j, z)
\end{cases}$$

- We can take $\theta_1$ to be a draw from $p(\theta)$
To create a custom likelihood, we will subclass BaseScalarFunction.

We will implement lnValue and actualValue.

Sublass like so:

```cpp
template<class V, class M>
class Likelihood : public QUESO::BaseScalarFunction<V, M>
{
```

You can call your class whatever you want, but Likelihood seemed like a good name.
Likelihoods

• Implement the *constructor*:

  ```cpp
  Likelihood(const char * prefix,
              const QUESO::VectorSet<V, M> & domain)
      : QUESO::BaseScalarFunction<V, M>(prefix, domain)
  {
    // Setup here
  }
  ```

• The constructor is called when you create an object of type `Likelihood`.

D. McDougall
QUESO tutorial
Likelihoods

- Implement the *destructor*:

  ```
  virtual ~Likelihood(
  {
    // Deconstruct here
  }
  )
  ```

- The destructor is called when your creates object of type `Likelihood` goes out of scope.

- Do all cleanup in the destructor.
Likelihoods

• Implement the `lnValue` method:

```cpp
virtual double lnValue(const V & domainVector,
    const V * domainDirection, V * gradVector,
    M * hessianMatrix, V * hessianEffect) const
{
    double diff = G(domainVector[0]) - m_observations[0];
    return -0.5 * diff * diff / (sigma * sigma);
}
```

• `lnValue` should return log of $p(\theta|y)$ at the point $\theta = \text{domainVector}$.

• You also need to implement `actualValue` but you can just return `std::exp(lnValue(...))`. 
Task 4

• We’ll use an example forward model in Chemistry called the Massman model:

\[ G(D, \beta) = DT^\beta \]

• Uncertain parameters are \( \theta = (D, \beta) \). Observations are taken at \( T = 313.7, 314.9, 375.2, 474.7, 481.0, 573.5, 671.1 \)

• Observation vector is \( y = \)

\((4603.50, 4638.15, 6302.27, 9505.89, 9755.11, 13239.08, 17431.02)\)

• Observational error is Gaussian and standard deviation is 10.0.

• Create your own likelihood for this forward problem.

• Instantiate it.

• Make sure you can evaluate it at a point in parameter space.

• This task should talk less than thirty minutes.
Statistical inverse problem

- So far, have created a prior
- Have created a likelihood
- We will now from the statistical inverse problem
- **QUESO** will solve the statistical inverse problem
Statistical inverse problem

Create a posterior random variable (on the same domain):

```cpp
QUESO::GenericVectorRV<> postRv("post_", paramSpace);
```

Create a statistical inverse problem with the prior, likelihood and posterior:

```cpp
QUESO::StatisticalInverseProblem<> ip("", NULL,

   priorRv,
   lhood, postRv);
```
Statistical inverse problem

Before we solve the problem we have to tell QUESO what the first sample in the chain is:

```cpp
QUESO::GslVector paramInitials(paramSpace.zeroVector());
paramInitials[0] = 1.0;
paramInitials[1] = 1.0;
```

And we also have to tell QUESO the variance of the proposal distribution:

```cpp
QUESO::GslMatrix propCovMatrix(paramSpace.zeroVector());
propCovMatrix(0, 0) = 0.0001;
propCovMatrix(1, 1) = 0.0001;
```

These are algorithmic knobs, and there are efforts to remove this necessity in future and start with sensible defaults. For now, though, we have to set these.
Now solve and tidy up:

```c
    ip.solveWithBayesMetropolisHastings(NULL, paramInitials,
                                          &propCovMatrix);
    MPI_Finalize();
    return 0;
```

Task 5

• Compile and run the code. Output will be in the outputData directory.
• Samples are stored in the ip_raw_chain_.m file as a matlab vector.
• The first column are samples of the first parameter (domainVector[0]). The second column are samples of the second parameter (domainVector[1]).
• Matlab output is a hinderance (not open source). There are efforts to output an HDF5 file in future. Initial support exists but is not usable at present.
• Use script to plot the samples to a PDF file.
• The truth is \((D, \beta) = (0.197, 1.75)\).
• Plot the running mean and variance of the samples. What do you get?
• This task should take less than twenty minutes.
**QUESO output**

- At this point, you should be able to see your samples.
- Are they correct?
- Should look like white noise
- Steps or large meanderings is considered transient behaviour
- An initial transient period is ok (see later) but it should settle down
- If there is an initial transient period, do not include this period when calculating mean, variance, other moments
- There are more quantitative convergence metrics: autocorrelation, Geweke, effective sample size, Raftery and Lewis, Heidelberger and Welch, Gelman and Rubin.
- See here for details: https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug_introbayes_sect008.htm
QUESTO output: Autocorrelation

Autocorrelation of chain:

![Graphs showing autocorrelation for D and β parameters.](image)
**QUESO output: Autocorrelation**

Autocorrelation of white noise:

Time taken for autocorrelation to decay is a measure of Markov chain mixing efficiency. 'Mixing' is how well the Markov chain explores the state space.
**QUESO output: Geweke diagnostic**

Take averages of chunk of samples from beginning and end separately:

\[
\bar{D}_1 = \frac{1}{n_1} \sum_{i=1}^{n_1} D_i \\
\bar{D}_2 = \frac{1}{n_2} \sum_{i=n-n_2+1}^{n} D_i
\]

The distribution of the random variable \( \bar{D}_1 - \bar{D}_2 \) converges to a normal distribution with mean zero.
QUESO output: Geweke diagnostic
Other QUESO output

- Other useful output is produced by QUESO.
- Should be reasonably obviously named, but:
  
  - log-likelihood values at each sample in the chain: `ip_raw_chain_loglikelihood.m`
  
  - log-target values at each sample in the chain: `ip_raw_chain_logtarget.m`
  
  - General diagnostic information and logging information that QUESO encounters: `display_sub0.txt`
  
  - Algorithmic information about the inverse problem: `sipOutput.m`

  - Proportion of rejected samples is written here
Parallel QUESO

- Have been running with ./binary inputfile.txt
- This is equivalent to mpirun -np 1 ./binary inputfile.txt
- Want to do mpirun -np N ./binary inputfile.txt
- \( N = 2 \) will tell QUESO to give the sampler two processes
- But still only one chain produced! Why?
- The usecase for this would be if the forward problem needed two processes per evaluation
Parallel QUESO

Chain 1

Node 1
Node 2

MCMC

Node 1
Node 2

Chain 2

Node 3
Node 4

MCMC

Node 3
Node 4

Chain 3

Node 5
Node 6

MCMC

Node 5
Node 6

D. McDougall
QUESO tutorial
25/03/2016
Parallel QUESO

- The number of chains can be increased from the input file. Change:
  
  ```python
  env_numSubEnvironments = 1
  ```
  
  to
  
  ```python
  env_numSubEnvironments = 4
  ```
  
  for four chains.

- For one process per chain, run with `mpirun -np 4`
- For two processes per chain, run with `mpirun -np 2`
- QUESO will check if `env_numSubEnvironments` divides the size of `MPI_COMM_WORLD`
- If not, an exception is thrown
Parallel QUESO

- When multiple subEnvironments are asked for, multiple output files are produced.
- The output files for subEnvironment $N + 1$ are prefixed with _subN before the file extension.
- Example: raw chain samples from process 3 are in ip_raw_chain_sub2.m.
- Example: log-likelihood values at each raw chain sample from process 2 are in ip_raw_chain_loglikelihood_sub1.m.
- QUESO automatically concatenates, in order of increasing subEnvironment index, each raw chain and saves it to ip_raw_chain.m so this file will contain the number of samples per subEnvironment multiplied by the number of subEnvironments.
QUESO input file: Environment options

- Already seen how to change number of chains with input file

- Prefix stuff is here. env is the default prefix is empty for the FullEnvironment. You can change it and the input file options will need to be prefixed with the passed user prefix.

- Some environment options:
  - env_subDisplayFileName is the place to QUESO will put general diagnostic information
  - env_subDisplayAllowAll toggles whether or not all process can write output related to environment
  - env_subDisplayAllowedSet if the above option is false, this option allows you to specify a subset of processes that can write output
  - env_seed is a flag for setting random number generator seed. Set this to something negative if you want multiple concurrent chains to produce distinct samples. If nonzero, this is the seed that is used for all chains.
Inverse problem options

- ip_computeSolution is a boolean. If false, no computation of the inverse problem is done. Useful for testing setup.

- ip_dataOutputFileName is the output file for inverse problem related information.

- ip_dataOutputAllowedSet similar to environment option, but specific to inverse problem
The QUESO input file

QUESO input file: Algorithm options

- All DRAM options are documented as part of the MhOptionsValues class
- Software documentation and usability is assessed by users
- If there are any options (anywhere, not just DRAM) that are unclear or undocumented feel free to open a new GitHub ticket
- Better yet, write a patch to fix it and you can be a QUESO contributor!
Programmatically setting options

- You can set options programmatically instead of via the input file.
- Objects whose behaviour is tweakable will accept a pointer to an instance of *OptionsValues in the constructor.
  
  ▶ FullEnvironment takes an EnvOptionsValues pointer:
  
  http://libqueso.com/queso/html/a00081.html#a77081a9fd8cb7b90ee3c0da289d91815

  ▶ StatisticalInverseProblem takes an SipOptionsValues pointer:
  
  http://libqueso.com/queso/html/a00207.html#a98ad98a7030b2c6577a0e840506bf74d

  ▶ Options object member names are similar to input file option names.

- You needn’t create a MetropolisHastingsSG object; the inverse problem will create one for you.
- You can pass options via solveWithBayesMetropolisHastings:
  
  ▶ http://libqueso.com/queso/html/a00207.html#a924189e647110129682308b9bfffc3a0d
Thinning

- A common way to try and improve decorrelation time is by thinning
- Thinning is simply saving every $m$-th sample where $m > 1$
- The \texttt{ip\_raw\_chain.m} contains the unthinned chain
- To enable thinning, turn on this input file option:
  - \texttt{ip\_mh\_filteredChain\_generate} = 1
- With this option on, you must also set the value for $m$:
  - \texttt{ip\_mh\_filteredChain\_lag} = 2
- Somewhere to store the output:
  - \texttt{ip\_mh\_filteredChain\_dataOutputFileName} = \texttt{outputData/ip\_filtered\_chain}
- Let all processes write their output
  - \texttt{ip\_mh\_filteredChain\_dataOutputAllowAll} = 1
Task 6

- Enable thinning
- Perhaps remove (or backup) the existing outputData directory
- Run the code (perhaps with multiple chains in parallel)
- Verify the length of the filtered chain is what you expect
- Compute the running mean of the samples. Does it converge faster or slower?
- This task should take less than twenty minutes.
Some useful DRAM options

- *ip_mh_putOutOfBoundsInChain*: If true, can be wasteful in high-dimensional bounded domains.

- *ip_mh_doLogitTransform*: Transforms bounded domain to unbounded domain for efficiency in high dimensions.


- *ip_mh_dr_listOfScalesForExtraStages*: A list of inverse scale factors to scale the proposal covariance matrix by in each DR stage.

- *ip_mh_am_initialNonAdaptInterval*: Number of iterations to not adapt.

- *ip_mh_am_adaptInterval*: The frequency at which to adapt proposal covariance matrix (after initial non-adapt period is over).
Optimisation

- **QUESO** can optimise, using deterministic optimisation tools in GSL, to find the MAP estimator before sampling.

- Better than starting from a user-defined initial sample, or from a realisation from the prior.

- Caveat: not all problems are easily optimised.

- If no derivative provided, then finite differences are used.

- The default tweakable optimisation parameters are hardcoded. There is an open ticket to change this, but it needs work: [https://github.com/libqueso/queso/pull/310](https://github.com/libqueso/queso/pull/310)

- To enable optimisation, call `seedWithMAPEstimator()` before calling `solveWithBayesMetropolisHastings(...)`.

Contributing

• All software has bugs

• QUESO is open source software, so it is free

• Free means you don’t have to pay for it (beer)

• Free also means you are free to play with the source code yourself (speech)

• If you use QUESO and you need help, please email the users mailing list

  ▶ queso-users@googlegroups.com

• To stay apprised of new releases or discuss future directions email the developers list

  ▶ queso-dev@googlegroups.com

• You can also help others that ask questions on these lists

• Community contribution
Contributing

• If you use QUESO and you encounter a bug, open a ticket
  ▶ https://github.com/libqueso/queso/issues/new

• If you fix a bug and want to contribute to the QUESO codebase
  ▶ Awesome
  ▶ Review contribution guidelines here:
    https://github.com/libqueso/queso#contributing
  ▶ Write tests for your patches
  ▶ And if you add a new feature, make sure to write documentation
The future of QUESO

- QUESO works closely with the DAKOTA team at Sandia
  - They provide feedback on our design
  - They request features
  - They find bugs
  - We learn from their (software and optimisation) expertise
  - DAKOTA can link to QUESO for inference
  - DAKOTA has a large userbase

- We have funding from Sandia to:
  - Improve the sampler interface to improve and ease maintainability for new and existing MCMC algorithms
  - Implement a generic vector/matrix interface so we can have higher performance vector implementations
  - Usability, user-friendliness, and licensing improvements