The QUESO Library

User's Manual
Version 0.46.0

Quantification of Uncertainty for Estimation, Simulation, and Optimization (QUESO)

Editors:
Kemelli C. Estacio-Hiroms
Ernesto E. Prudencio

Center for Predictive Engineering and Computational Sciences (PECOS)
Institute for Computational and Engineering Sciences (ICES)
The University of Texas at Austin
Austin, TX 78712, USA

Abstract

QUESO stands for Quantification of Uncertainty for Estimation, Simulation and Optimization and consists of a collection of algorithms and C++ classes intended for research in uncertainty quantification, including the solution of statistical inverse and statistical forward problems, the validation of mathematical models under uncertainty, and the prediction of quantities of interest from such models along with the quantification of their uncertainties.

QUESO is designed for flexibility, portability, easy of use and easy of extension. Its software design follows an object-oriented approach and its code is written on C++ and over MPI. It can run over uniprocessor or multiprocessor environments.

QUESO contains two forms of documentation: a user's manual available in pdf format and a lower-level code documentation available in web based/html format.

This is the user's manual: it gives an overview of the QUESO capabilities, provides procedures for software execution, and includes example studies.

Disclaimer

Contents

Preface

The QUESO project started in 2008 as part of the efforts of the recently established Center for Predictive Engineering and Computational Sciences (PECOS) at the Institute for Computational and Engineering Sciences (ICES) at The University of Texas at Austin.

The PECOS Center was selected by the National Nuclear Security Administration (NNSA) as one of its new five centers of excellence under the Predictive Science Academic Alliance Program (PSAAP). The goal of the PECOS Center is to advance predictive science and to develop the next generation of advanced computational methods and tools for the calculation of reliable predictions on the behavior of complex phenomena and systems (multiscale, multidisciplinary). This objective demands a systematic, comprehensive treatment of the calibration and validation of the mathematical models involved, as well as the quantification of the uncertainties inherent in such models. The advancement of predictive science is essential for the application of Computational Science to the solution of realistic problems of national interest.

The QUESO library, since its first version, has been publicly released as open source under the GNU General Public License and is available for free download world-wide. See http://www.gnu.org/licenses/gpl.html for more information on the GPL software use agreement.

Contact Information:

Ernesto E. Prudencio
Institute for Computational and Engineering Sciences
1 University Station C0200
Austin, Texas 78712
email: prudenci@ices.utexas.edu
web: http://pecos.ices.utexas.edu

Referencing the QUESO Library

When referencing the QUESO library in a publication, please cite the following:

```
@incollection{QUESO,
  author    = "Ernesto Prudencio and Karl W. Schulz",
  title     = {The Parallel C++ Statistical Library ‘QUESO’:
               Quantification of Uncertainty for Estimation,
               Simulation and Optimization},
  booktitle = {Euro-Par 2011: Parallel Processing Workshops},
  series    = {Lecture Notes in Computer Science},
  publisher = {Springer Berlin / Heidelberg},
  isbn      = {978-3-642-29736-6},
  keyword   = {Computer Science},
  pages     = {398-407},
  volume    = {7155},
  url       = {http://dx.doi.org/10.1007/978-3-642-29737-3_44},
  year      = {2012}
}

@TechReport{queso-user-ref,
   Author      = {Kemelli C. Estacio-Hiroms and Ernesto E. Prudencio},
   Title       = {{T}he {QUESO} {L}ibrary, {U}ser’s {M}anual},
   Institution = {Center for Predictive Engineering and Computational Sciences
                  (PECOS), at the Institute for Computational and Engineering
                  Sciences (ICES), The University of Texas at Austin},
   Note        = {in preparation},
   Year        = {2013}
}
```

QUESO Development Team

The QUESO development team currently consists of Paul T. Bauman, Sai Hung Cheung, Kemelli C. Estacio-Hiroms, Damon McDougall, Kenji Miki, Todd A. Oliver, Ernesto E. Prudencio (lead developer), Karl W. Schulz, and Rhys Ulerich.

Acknowledgments

This work has been supported by the United States Department of Energy, under the National Nuclear Security Administration Predictive Science Academic Alliance Program (PSAAP) award number [DE-FC52-08NA28615], and under the Office of Science Scientific Discovery through Advanced Computing (SciDAC) award number [DE-SC0006656].

We would also like to thank James Martin, Roy Stogner and Lucas Wilcox for interesting discussions and constructive feedback.

Target Audience

QUESO is a collection of statistical algorithms and programming constructs supporting research into the uncertainty quantification (UQ) of models and their predictions. UQ may be a very complex and time consuming task, involving many steps: decide which physical model(s) to use; decide which reference or experimental data to use; decide which discrepancy models to use; decide which quantity(ies) of interest (QoI) to compute; decide which parameters to calibrate; perform computational runs and collect results; analyze computational results, and eventually reiterate; predict QoI(s) with uncertainty.

The purpose of this manual is <u>not</u> to teach UQ and its methods, but rather to introduce QUESO library so it can be used as a tool to assist and facilitate the uncertainty quantification of the user's application.

Thus, the target audience of this manual is researchers who have solid background in Bayesian methods, are comfortable with UNIX concepts and the command line, and have knowledge of a programming language, preferably C/C++. Bellow we suggest some useful literature:

1. Probability, statistics, random variables [6, 7, 20];
2. Bayes' formula [4, 9, 21, 32];
3. Markov chain Monte Carlo (MCMC) methods [3, 10, 13, 14, 15, 23, 25, 28, 29];
4. Monte Carlo methods [33];
5. Kernel density estimation [34];
6. C++ [22, 26];
7. Message Passing Interface (MPI) [39, 37];
8. UNIX/Linux (installation of packages, compilation, linking);
9. MATLAB/GNU Octave (for dealing with output files generated by QUESO); and
10. UQ issues in general [5].

CHAPTER 1

Introduction

QUESO is a parallel object-oriented statistical library dedicated to the research of statistically robust, scalable, load balanced, and fault-tolerant mathematical algorithms for the quantification of uncertainty (UQ) of mathematical models and their predictions.

The purpose of this chapter is to introduce relevant terminology, mathematical and statistical concepts, statistical algorithms, together with an overall description of how the user's application may be linked with the QUESO library.

1.1 Preliminaries

Statistical inverse theory reformulates inverse problems as problems of statistical inference by means of Bayesian statistics: all quantities are modeled as random variables, and probability distribution of the quantities encapsulates the uncertainty observed in their values. The solution to the inverse problem is then the probability distribution of the quantity of interest when all information available has been incorporated in the model. This (posterior) distribution describes the degree of confidence about the quantity after the measurement has been performed [23].

Thus, the solution to the statistical inverse problem may be given by Bayes' formula, which express the posterior distribution as a function of the prior distribution and the data represented through the likelihood function.

The likelihood function has an open form and its evaluation is highly computationally expensive. Moreover, simulation-based posterior inference requires a large number of forward calculations to be performed, therefore fast and efficient sampling techniques are required for posterior inference.

It is often not straightforward to obtain explicit posterior point estimates of the solution, since it usually involves the evaluation of a high-dimensional integral with respect to a possibly non-smooth posterior distribution. In such cases, an alternative integration technique is the Markov chain Monte Carlo method: posterior means may be estimated using the sample mean from a series of random draws from the posterior distribution.

QUESO is designed in an abstract way so that it can be used by any computational model, as long as a likelihood function (in the case of statistical inverse problems) and a quantity of interest (QoI) function (in the case of statistical forward problems) is provided by the user application.

QUESO provides tools for both sampling algorithms for statistical inverse problems, following Bayes' formula, and statistical forward problems. It contains Monte Carlo solvers (for autocorrelation, kernel density estimation and accuracy assessment), MCMC (e.g. Metropolis Hastings [28, 15]) as well as the DRAM [13] (for sampling from probability distributions); it also has the capacity to handle many chains or sequences in parallel, each chain or sequence itself demanding many computing nodes because of the computational model being statistically explored [31].

1.2 Key Statistical Concepts

A computational model is a combination of a mathematical model and a discretization that enables the approximate solution of the mathematical model using computer algorithms and might be used in two different types of problems: forward or inverse.

Any computational model is composed of a vector $\boldsymbol{\theta}$ of n *parameters*, *state variables* $\mathbf{u}$, and *state equations* $\mathbf{r}(\boldsymbol{\theta}, \mathbf{u}) = \mathbf{0}$. Once the solution $\mathbf{u}$ is available, the computational model also includes extra functions for e.g. the calculation of *model output data* $\mathbf{y} = \mathbf{y}(\boldsymbol{\theta}, \mathbf{u})$, and the *prediction* of a vector $\mathbf{q} = \mathbf{q}(\boldsymbol{\theta}, \mathbf{u})$ of m quantities of interest (QoI),

Parameters designate all model variables that are neither state variables nor further quantities computed by the model, such as: material properties, coefficients, constitutive parameters, boundary conditions, initial conditions, external forces, parameters for modeling the model error, characteristics of an experimental apparatus (collection of devices and procedures), discretization choices and numerical algorithm options.

In the case of a forward problem, the parameters $\boldsymbol{\theta}$ are given and one then needs to compute $\mathbf{u}$, $\mathbf{y}$ and/or $\mathbf{q}$. In the case of an inverse problem, however, experimental data $\mathbf{d}$ is given and one then needs to *estimate* the values of the parameters $\boldsymbol{\theta}$ that cause $\mathbf{y}$ to best fit $\mathbf{d}$.

Figure 1.2.1 represents general inverse and forward problems respectively.

There are many possible sources of uncertainty on a computational model. First, $\mathbf{d}$ need not be equal to the actual values of observables because of errors in the measurement process. Second, the values of the input parameters to the phenomenon might not be precisely known. Third, the appropriate set of equations governing the phenomenon might not be well understood.

Computational models can be classified as either deterministic or stochastic – which are the ones of interest here. In deterministic models, all parameters are assigned numbers, and

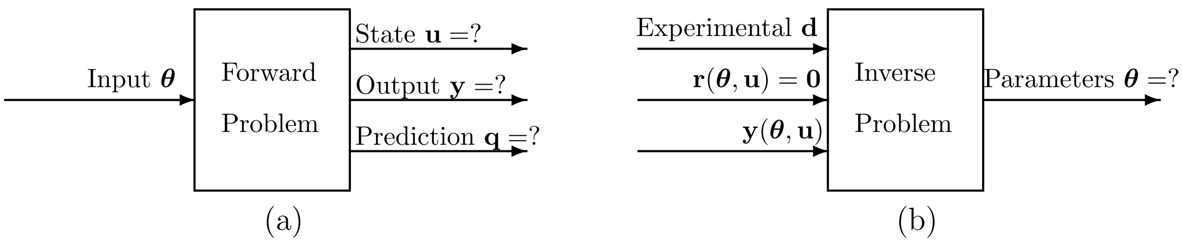


Figure 1.2.1: The representation of (a) a generic forward problem and (b) a generic inverse problem.

no parameter is related to the parametrization of a random variable (RV) or field. As a consequence, a deterministic model assigns a number to each of the components of quantities $\mathbf{u}$, $\mathbf{y}$ and $\mathbf{q}$. In stochastic models, however, at least one parameter is assigned a probability density function (PDF) or is related to the parametrization of a RV or field, causing $\mathbf{u}$, $\mathbf{y}$ and $\mathbf{q}$ to become random variables. Note that not all components of $\boldsymbol{\theta}$ need to be treated as random. As long as at least one component is random, $\boldsymbol{\theta}$ is a random vector, and the problem is stochastic.

In the case of forward problems, statistical forward problems can be represented very similarly to deterministic forward problems, as seen in Figure 1.2.2. In the case of inverse problems, as depicted in Figure 1.2.3, however, the conceptual connection between deterministic and statistical problems is not as straightforward.

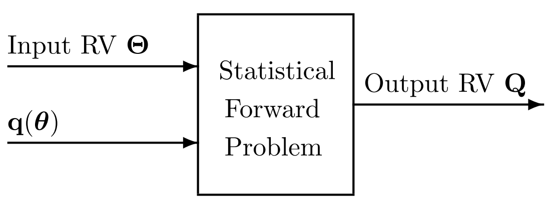


Figure 1.2.2: The representation of a statistical forward problem. $\boldsymbol{\Theta}$ denotes a random variable related to parameters, $\boldsymbol{\theta}$ denotes a realization of $\boldsymbol{\Theta}$ and $\mathbf{Q}$ denotes a random variable related to quantities of interest.

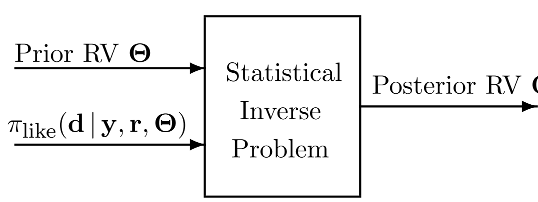


Figure 1.2.3: The representation of a statistical inverse problem. $\boldsymbol{\Theta}$ denotes a random variable related to parameters, $\boldsymbol{\theta}$ denotes a realization of $\boldsymbol{\Theta}$ and $\mathbf{r}$ denotes model equations, $\mathbf{y}$ denotes some model output data and $\mathbf{d}$ denotes experimental data.

QUESO adopts a Bayesian analysis [23, 32] for statistical inverse problems, interpreting the posterior PDF

$$\pi_{\text{posterior}}(\boldsymbol{\theta}|\mathbf{d}) = \frac{\pi_{\text{prior}}(\boldsymbol{\theta})\pi_{\text{likelihood}}(\mathbf{d}|\boldsymbol{\theta})}{\pi(\mathbf{d})} \tag{1.2.1}$$

as the solution. Such solutions combine the prior information $\pi_{\text{prior}}(\boldsymbol{\theta})$ of the parameters, the information $\pi(\mathbf{d})$ on the data, and the likelihood $\pi_{\text{likelihood}}(\mathbf{d}|\boldsymbol{\theta})$ that the model computes certain data values with a given set of input parameters.

This semantic interpretation of achieving a posterior knowledge on the parameters (on the model) after combining some prior model knowledge with experimental information provides an important mechanism for dealing with uncertainty. Although mathematically simple, is not computationally trivial.

1.3 The Software Stack of an Application Using QUESO

An application using QUESO falls into three categories: a statistical inverse problem (IP), a statistical forward problem (FP), or combinations of both. In each problem the user might deal with up to five vectors of potentially very different sizes: parameters $\boldsymbol{\theta}$, state $\mathbf{u}$, output $\mathbf{y}$, data $\mathbf{d}$ and QoIs $\mathbf{q}$.

Algorithms in the QUESO library require the supply of a likelihood routine $\pi_{\text{like}} : \mathbb{R}^n \to \mathbb{R}_+$ for statistical inverse problems and of a QoI routine $\mathbf{q} : \mathbb{R}^n \to \mathbb{R}^m$ for statistical forward problems. These routines exist at the application level and provide the necessary bridge between the statistical algorithms in QUESO, model knowledge in the model library and scenario and experimental data in the disk space. Figure 1.3.1 shows the software stack of a typical application that uses QUESO. In the figure, the symbol $\boldsymbol{\theta}$ represents a vector of $n \geqslant 1$ parameters.

Even though QUESO deals directly with $\boldsymbol{\theta}$ and $\mathbf{q}$ only, it is usually the case the one of the other three vectors ($\mathbf{u}$, $\mathbf{y}$ and $\mathbf{d}$) will have the biggest number of components and will therefore dictate the size of the minimum parallel environment to be used in a problem. So, for example, even though one processor might be sufficient for handling $\boldsymbol{\theta}$, $\mathbf{y}$, $\mathbf{d}$ and $\mathbf{q}$, eight processors at least might be necessary to solve for $\mathbf{u}$. QUESO currently only requires that the amounts n and m can be handled by the memory available to one processor, which allows the analysis of problems with thousands of parameters and QoIs, a large amount even for state of the art UQ algorithms.

QUESO currently supports three modes of parallel execution: an application user may simultaneously run:

(a) multiple instances of a problem where the physical model requires a single processor, or

(b) multiple instances of a problem where the physical model requires multiple processors, or

(c) independent sets of types (a) and (b).

For example, suppose an user wants to use the Metropolis-Hastings (MH) algorithm to solve a statistical IP, and that 1,024 processors are available. If the physical model is simple

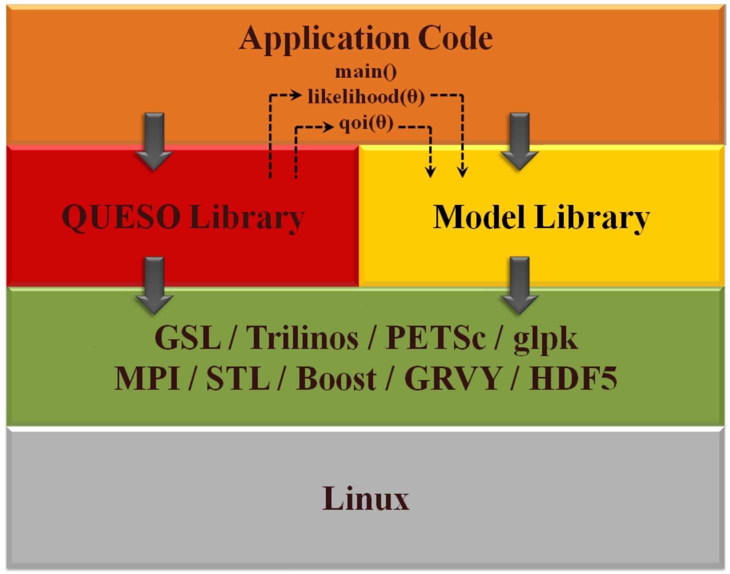


Figure 1.3.1: An application software stack. QUESO requires the input of a likelihood routine $\pi_{\text{like}} : \mathbb{R}^n \to \mathbb{R}_+$ for IPs and of a QoI routine $\mathbf{q} : \mathbb{R}^n \to \mathbb{R}^m$ for FPs. These application level routines provide the bridge between the statistical algorithms in QUESO, physics knowledge in the model library, and relevant experimental (calibration and validation) data.

enough to be handled efficiently by a single processor, then the user can run 1,024 chains simultaneously, as in case (a). If the model is more complex and requires, say, 16 processors, then the user can run 64 chains simultaneously, as in case (b), with 16 processors per chain. QUESO treats this situation by using only 1 of the 16 processors to handle the chain. When a likelihood evaluation is required, all 16 processors call the likelihood routine simultaneously. Once the likelihood returns its value, QUESO puts 15 processors into idle state until the routine is called again or the chain completes. Case (c) is useful, for instance, in the case of a computational procedure involving two models, where a group of processors can be split into two groups, each handling one model. Once the two-model analysis end, the combined model can use the full set of processors.[1]

[1]The parallel capabilities of QUESO have been exercised on the Ranger system of the TACC [2] with up to 16k processors.

CHAPTER 2

Installation

This chapter covers the basic steps that a user will need follow when beginning to use QUESO: how to obtain, configure, build, install, and test the library. It also presents both QUESO source and installed directory structure, some simple examples and finally, introduces the user on how to use QUESO together with his/her application.

This manual is current at the time of printing; however, QUESO library is under active development.

2.1 Pre-QUESO Installation Steps

Herein, suppose you want to install QUESO and its dependencies on the following directory:

```
$HOME/LIBRARIES/
```

so that you will not need root access rights. The directory above is referred to as the QUESO installation directory (tree).

There are two main steps to prepare your LINUX computing system for the QUESO Library: obtain and install QUESO dependencies, and define a number of environmental variables. These steps are discussed bellow.

2.1.1 Obtain and Install QUESO Dependencies

QUESO interfaces to a number of high-quality software packages to provide certain functionalities. While some of them are required for the successful installation of QUESO, other may be used for enhancing its performance. QUESO dependencies are:

1. **STL**: The Standard Template Library is a C++ library of container classes, algorithms, and iterators; it provides many of the basic algorithms and data structures of computer science [19].

2. **GSL**: The GNU Scientific Library is a numerical library for C and C++ programmers. It provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting [8]. The lowest version of GSL required by QUESO is GSL 1.10.

3. **Boost**: Boost provides free peer-reviewed portable C++ source libraries, which can be used with the C++ Standard Library [35]. QUESO requires Boost 1.35.0 or newer.

4. **MPI**: The Message Passing Interface is a standard for parallel programming using the message passing model. E.g. Open MPI [39] or MPICH [37]. QUESO requires MPI during the compilation step; however, you may run it in serial mode (e.g. in one single processor) if you wish.

QUESO also works with the following optional libraries:

1. **GRVY**: The Groovy Toolkit (GRVY) is a library used to house various support functions often required for application development of high-performance, scientific applications. The library is written in C++, but provides an API for development in C and Fortran [36]. QUESO requires GRVY 0.29 or newer.

2. **HDF5**: The Hierarchical Data Format 5 is a technology suite that makes possible the management of extremely large and complex data collections [11]. The lowest version required by QUESO is HDF5 1.8.0.

3. **GLPK**: The GNU Linear Programming Kit package is is a set of routines written in ANSI C and organized in the form of a callable library for solving large-scale linear programming, mixed integer programming, and other related problems [27]. QUESO works with GLPK versions newer than or equal to GLPK 4.35.

4. **Trilinos**: The Trilinos Project is an effort to develop and implement robust algorithms and enabling technologies using modern object-oriented software design, while still leveraging the value of established libraries. It emphasizes abstract interfaces for maximum flexibility of component interchanging, and provides a full-featured set of concrete classes that implement all abstract interfaces [17, 16]. QUESO requires Trilinos release to be newer than or equal to 8.0.7.

The majority of QUESO output files is MATLAB®/GNU Octave compatible [12, 38]. Thus, for results visualization purposes, it is recommended that the user have available either one of these tools.

2.1.2 Prepare your LINUX Environment

Before using QUESO, the user must first set a number of environmental variables, and indicate the full path of the QUESO's dependencies: GSL, Boost and GRVY.

For example, under the UNIX C shell (csh) a command of the form

```
$ setenv LD_LIBRARY_PATH \$LD_LIBRARY_PATH:
    $HOME/LIBRARIES/gsl-1.15/lib:
    $HOME/LIBRARIES/boost-1.53.0/lib:
    $HOME/LIBRARIES/grvy-0.32.0/lib
```

can be placed in the user's `.cshrc` or other startup file. Under UNIX bash shell, `.cshrc` is replaced with a startup file such as `.bashrc`, and the command `setenv` with `export`.

In addition, the user must set the following environmental variables:

```
$ setenv CC gcc
$ setenv CXX g++
$ setenv MPICC mpicc
$ setenv MPICXX mpic++
$ setenv F77 f77
$ setenv FC gfortran
```

2.2 Obtaining a Copy of QUESO

The QUESO library is available in the form of a tarball (tar format compressed with gzip), for research collaborators, upon request to Ernesto E. Prudencio (prudenci@ices.utexas.edu).

Suppose you have copied the file '`queso-0.46.0.tar.gz`' into `$HOME/queso_download/`. Then just follow these commands to expand the tarball:

```
$ cd $HOME/queso_download/
$ tar xvf queso-0.46.0.tar.gz
$ cd queso-0.46.0     #enter the directory
```

Naturally, for versions of QUESO other than 0.46.0, the file names in the above commands must be adjusted.

2.2.1 Recommended Build Directory Structure

Via Autoconf and Automake, QUESO configuration facilities provide a great deal of flexibility for configuring and building the existing QUESO packages. However, unless a user has prior experience with Autotools, we strongly recommend the following process to build and maintain local builds of QUESO (as an example, see note on Section 2.6). To start, we defined three useful terms:

Source tree - The directory structure where the QUESO source files are located. A source tree is is typically the result of expanding an QUESO distribution source code bundle, such as a tarball.

Build tree - The tree where QUESO is built. It is always related to a specific source tree, and it is the directory structure where object and library files are located. Specifically, this is the tree where you invoke `configure, make`, etc. to build and install QUESO.

Installation tree - The tree where QUESO is installed. It is typically the `prefix` argument given to QUESO's configure script; it is the directory from which you run installed QUESO executables.

Although it is possible to run `./configure` from the source tree (in the directory where the configure file is located), we recommend separate build trees. The greatest advantage to having a separate build tree is that multiple builds of the library can be maintained from the same source tree [17].

2.3 Configure QUESO Building Environment

QUESO uses the GNU Autoconf system for configuration, which detects various features of the host system and creates Makefiles. The configuration process can be controlled through environment variables, command-line switches, and host configuration files. For a complete list of switches type:

```
$ ./configure  --help
```

from the top level of the source tree.

This command will also display the help page for QUESO options. Many of the QUESO configure options are used to describe the details of the build. For instance, to include a package that is not currently built by default, HDF5, append `--with-hdf5=DIR`, where `DIR` is the root directory of HDF5 installation, to the configure invocation line.

QUESO default installation location is '`/usr/local`', which requires superuser privileges. To use a path other than '`/usr/local`', specify the path with the '`--prefix=PATH`' switch. For instance, '`--prefix=$HOME/LIBRARIES`'.

The basic steps to configure QUESO using Boost, GSL and HDF5 for installation at '`$HOME/LIBRARIES/QUESO-0.46.0`' are:

```
$ ./configure --prefix=$HOME/LIBRARIES/QUESO-0.46.0 \
  --with-boost=$HOME/LIBRARIES/boost-1.53.0 \
  --with-gsl=$HOME/LIBRARIES/gsl-1.15 \
  --with-hdf5=$HOME/LIBRARIES/hdf5-1.8.10
```

Note: the directory '`$HOME/LIBRARIES/QUESO-0.46.0`' does not need to exist in advance, since it will be created by the command `make install` described in Section 2.4.

2.4 Compile, Check and Install QUESO

In order to build, check and install the library, the user must enter the following three commands sequentially:

```
$ make
$ make check          # optional
$ make install
```

Here, `make` builds the library, confidence tests, and programs; `make check` conducts various test suites in order to check the compiled source; and `make install` installs QUESO library, include files, and support programs

The files are installed under the installation tree (refer to Section 2.2.1), e.g. the directory specified with '`--prefix=DIR`' in Section 2.3. The directory, if not existing, will be created automatically.

By running `make check`, several printouts appear in the screen and you should see messages such as:

```
--------------------------------------------------------------------
(rtest): PASSED: Test 1 (TGA Validation Cycle)
--------------------------------------------------------------------
```

The tests printed in the screen are tests under your QUESO build tree, i.e., they are located at the directory `$HOME/queso_download/queso-0.46.0/test` (see Section 2.7 for the complete list of the directories under QUESO build tree). These tests are used as part of the periodic QUESO regression tests, conducted to ensure that more recent program/code changes have not adversely affected existing features of the library.

2.5 QUESO Developer's Documentation

QUESO code documentation is written using Doxygen [40], and can be regenerated by typing in the build tree:

```
$ make docs
```

A directory named `docs` will be created in `$HOME/queso_download/queso-0.46.0` (the build tree; your current path) and you may access the code documentation in two different ways:

1. HyperText Markup Language (HTML) format: `docs/html/index.html`, and the browser of your choice can be used to walk through the HTML documentation.
2. Portable Document Format (PDF) format: `docs/queso.pdf`, which can be accessed thought any PDF viewer.

2.6 Summary of Installation Steps

Supposing you have downloaded the file `queso-0.46.0.tar.gz` into `$HOME/queso_download/`. The basic steps to configure QUESO using Boost, GSL and HDF5 for installation at '`$HOME/LIBRARIES/QUESO-0.46.0`' are:

```
$ cd $HOME/queso_download/                    #enter source dir
$ gunzip < queso-0.46.0.tar.gz  | tar xf -
$ cd $HOME/queso_download/queso-0.46.0  #enter the build dir
$ ./configure --prefix=$HOME/LIBRARIES/QUESO-0.46.0 \
  --with-boost=$HOME/LIBRARIES/boost-1.53.0 \
  --with-gsl=$HOME/LIBRARIES/gsl-1.15 \
  --with-hdf5=$HOME/LIBRARIES/hdf5-1.8.10
$ make
$ make check
$ make install
$ make docs
$ ls $HOME/LIBRARIES/QUESO-0.46.0 #listing QUESO installation dir
>>  bin  include  lib  examples
```

2.7 The Build Directory Structure

The QUESO build directory contains three main directories, `src`, `examples` and `test`. They are listed below and more specific information about them can be obtained with the Developer's documentation from Section 2.5 above:

1. `src`, with five subdirectories:
   (a) `src/basic/`: with `inc` and `src` subdirectories,
   (b) `src/core/`: with `inc` and `src` subdirectories,
   (c) `src/misc/`: with `inc` and `src` subdirectories,
   (d) `src/stats/`: with `inc` and `src` subdirectories, and
   (e) `src/contrib/`.

The `src` directory contains the QUESO library itself; thus it has an entire chapter dedicated to its description (see Chapter 3).

2. `examples` with six subdirectories:

    (a) `examples/gravity/`
    (b) `examples/infoTheoryProblem/`
    (c) `examples/statisticalForwardProblem/`
    (d) `examples/statisticalInverseProblem/`
    (e) `examples/validationCycle/`
    (f) `examples/validationCycle2/`

    The simplest examples under this directory are: `statisticalInverseProblem` and `statisticalForwardProblem`, which solve a SIP and a SFP, respectively (see Section 2.9). The examples `validationCycle` and `validationCycle2` present a combination of SIP and SFP to solve the same problem; however, the first has the majority of its code in `*.h` files, with templated routines, whereas the latter has the majority of its code in `*.C` files. The `gravity` example is also a combination of a SIP and a SFP, and it is presented in detail in Chapter 5.

    The build directory contains only the source codes. The executables are available under the QUESO installation directory, together with some Matlab files for visualization of the results.

3. `test` with ten subdirectories:

    (a) `test/gsl_tests`
    (b) `test/t01_valid_cycle/`
    (c) `test/t02_sip_sfp/`
    (d) `test/t03_sequence/`
    (e) `test/t04_bimodal/`
    (f) `test_Environment/`
    (g) `test_GaussianVectorRVClass/`
    (h) `test_GslMatrix/`
    (i) `test_GslVector/`
    (j) `test_uqEnvironmentOptions/`

    The executables under `tests` are used as part of the periodic QUESO regression tests, conduct to ensure that more recent program/code changes have not adversely affected existing features of the library, as described in Section 2.4. They can optionally be called during QUESO installation steps by entering the instruction: `make check`.

2.8 The Installed Directory Structure

After having successfully executed steps described in §2.1 through §2.4, the QUESO installed directory will contain four subdirectories:

1. `bin`: contains the executable `queso_version`, which provides information about the installed library. The code bellow presents the output of it:

```
kemelli@violeta:~/LIBRARIES/QUESO-0.46.0/bin$ ./queso_version
----------------------------------------------------------------
QUESO Library: Version = 0.46.0 (4600)

Development Build

Build Date   = 2013-04-29 17:05
Build Host   = violeta
Build User   = kemelli
Build Arch   = x86_64-unknown-linux-gnu
Build Rev    = 38998M

C++ Config   = mpic++ -g -O2 -Wall

Trilinos DIR =
GSL Libs     = -L/home/kemelli/LIBRARIES/gsl-1.15/lib -lgsl -
   lgslcblas -lm
GRVY DIR     =
GLPK DIR     =
HDF5 DIR     = /home/kemelli/LIBRARIES/hdf5-1.8.10
----------------------------------------------------------------
```

2. `lib`: contains the static and dynamic versions of the library. The full to path to this directory, e.g., `$HOME/LIBRARIES/QUESO-0.46.0/bin` should be added to the user's `LD_LIBRARY_PATH` environmental variable in order to use QUESO library with his/her application code.

3. `include`: contains the library `*.h` files.

4. `examples`: contains the same examples of QUESO build directory, and listed in Section 2.7, together with their executables and Matlab files that may be used for visualization purposes.

   The executables under `examples` are examples of application codes that use QUESO to solve either SIP or SFP, or both. Section 2.9 presents the steps for running two of them, `statisticalInverseProblem` and `statisticalForwardProblem`; the user is invited to run them and understand their purpose (the codes are well documented and self-explanatory). The examples `validationCycle` and `validationCycle2` present a combination of SIP and SFP to solve the same problem; the main difference between them is that the first has the majority of its code in `*.h` files, with templated routines, whereas the latter has the majority of its code in `*.C` files. The `gravity` example is also a combination of a SIP and a SFP, and it is presented in detail in Chapter 5.

2.9 Running QUESO Examples

This section assumes that you have successfully executed steps described in Sections 2.1 through 2.5 above. The codes listed in this section are quite self-explanatory and print messages during execution to make clearer which problem they are solving and how.

The two following subsections illustrate how to run the executables provided under the `examples/statisticalInverseProblem/` and `examples/statisticalForwardProblem/`. For the remaining three examples, the steps should be analogous, except for `infoTheoryProblem`, which requires QUESO to be compiled with the ANN library [30].

It is worth noting presence of an argument passed to the executable in the examples. The argument is a input file to be provided to QUESO with options for the solution of the SIP and/or SFP; and it is always required. Each option in the input file is related to one (or more) of the QUESO classes, and is presented throughout Chapter 3.

A complete example of an application that uses QUESO to solve a combination of a SIP and a SFP is in presented in Chapter 5. The chapter presents the mathematical models for both the SIP and SFP, the application code, the options input file and the Makefile to link the code with QUESO library.

2.9.1 A Simple Statistical Inverse Problem

This example is located at `examples/statisticalInverseProblem` and consists of a set of three files to illustrate the use of QUESO library to solve a simple inverse problem with four parameters.

To run the executable provided, enter the following commands:

```
$ cd $HOME/LIBRARIES/QUESO-0.46.0/
$ cd examples/statisticalInverseProblem
$ rm outputData/*
$ ./exStatisticalInverseProblem_gsl sip.inp
$ matlab
   $ sip_plot             # inside matlab
   # press the left button of the mouse at each picture displayed
   # by 'sip_plot.m', in order to display the next picture
   $ exit                 # inside matlab
$ ls -l outputData/*.png
>>  parameters_samples_plane.png  parameters_PDF.png
```

As a result, the user should have created a couple of PNG plots for both marginal posterior PDFs of all four parameters and samples of first two parameters on the plane.

2.9.2 A Simple Statistical Forward Problem

This example consists of a set of three files to illustrate the use of QUESO library to solve a simple forward problem and it is located at '`examples/statisticalForwardProblem/`'.

To run the executable provided, enter the following commands:

```
$ cd $HOME/LIBRARIES/QUESO-0.46.0/
$ cd examples/statisticalForwardProblem/
$ rm outputData/*
$ ./exStatisticalForwardProblem_gsl sfp.inp
$ matlab
   $ sfp_plot        # inside matlab
   # press the left button of the mouse at a picture displayed
   # by 'sfp_plot.m', in order to display the next picture
   $ exit            # inside matlab
$ ls -l outputData/*.png
>>  QoI_autocorrelation.png  QoI_CDF.png  QoI_PDF.png
```

In this case, the user should have created a few PNG plots for the QoI kernel density estimation, cumulative distribution function and autocorrelation.

2.10 Create your Application with the Installed QUESO

Prepare your environment by either running or saving the following command in your `.cshrc` file (or `.bashrc` file depending whether you have a C or a bash shell):

```
setenv LD_LIBRARY_PATH \$LD_LIBRARY_PATH:
       $HOME/LIBRARIES/QUESO-0.46.0/lib
```

Suppose your application code consists of the files: `example_main.C`, `example_qoi.C`, `example_likelihood.C, example_compute.C` and respective `.h` files. Your application code may be linked with QUESO library through a Makefile such as the one displayed as follows:

```
QUESO_DIR = $HOME/LIBRARIES/QUESO-0.46.0/
BOOST_DIR = $HOME/LIBRARIES/boost-1.53.0/
GSL_DIR = $HOME/LIBRARIES/gsl-1.15/
GRVY_DIR = $HOME/LIBRARIES/grvy-0.32.0/
HDF5_DIR = $HOME/LIBRARIES//hdf5-1.8.10
TRILINOS_DIR = $HOME/LIBRARIES/trilinos-10.12.2
```

```
INC_PATHS = \
  -I. \
  -I$(QUESO_DIR)/include \
  -I$(BOOST_DIR)/include \
  -I$(GSL_DIR)/include \
  -I$(GRVY_DIR)/include \
  -I$(HDF5_DIR)/include \
  -I$(TRILINOS_DIR)/include \

LIBS = \
  -L$(QUESO_DIR)/lib \
  -lqueso \
  -L$(BOOST_DIR)/lib \
  -lboost_program_options \
  -L$(GSL_DIR)/lib \
  -lgsl \
  -L$(GRVY_DIR)/lib \
  -lgrvy \
  -L$(HDF5_DIR)/lib \
  -lhdf5 \
  -L$(TRILINOS_DIR)/lib \
  -lepetra \
  -lteuchos

CXX = mpic++
CXXFLAGS += -O3 -Wall -c

default: all

.SUFFIXES: .o .C

all:  ex_gsl

clean:
  rm -f *~
  rm -f *.o
%   rm -f example_gsl

ex_gsl: example_main.o example_likelihood.o example_qoi.o
   example_compute.o
  $(CXX) example_main.o example_likelihood.o example_qoi.o \
         example_compute.o -o example_gsl $(LIBS)

%.o: %.C
  $(CXX) $(INC_PATHS) $(CXXFLAGS) $<
```

CHAPTER 3

C++ Classes in the Library

QUESO is is a parallel object-oriented statistical library dedicated to the research of statistically robust, scalable, load balanced, and fault-tolerant mathematical algorithms for the quantification of uncertainty in realistic computational models and predictions related to natural and engineering systems.

Classes in QUESO can be divided in four main groups: core, templated basic, templated statistical and miscellaneous. The classed that handle environment (and options), vector and matrix classes are considered *core* classes. Classes implementing vector sets and subsets, vector spaces, scalar functions, vector functions, scalar sequences and vector sequences are *templated basic* classes; they are necessary for the definition and description of other entities, such as RVs, Bayesian solutions of IPs, sampling algorithms and chains. Vector realizer, vector RV, statistical IP (and options), MH solver (and options), statistical FP (and options), MC solver (and options) and sequence statistical options are part of *templated statistical* classes. And finally, the *miscellaneous* classes consist of C and FORTRAN interfaces.

3.1 Core Classes

QUESO core classes are the classes responsible for handling the environment (and options), vector and matrix operations. They are described in the following sections.

3.1.1 Environment Class (and Options)

The `Environment` class sets up the environment underlying the use of the QUESO library by an executable. This class is virtual. It is inherited by `uqEmptyEnvironmentClass` and `uqFullEnvironmentClass`. The QUESO environment class is instantiated at the application level, right after `MPI_Init(&argc,&argv)`. The QUESO environment is required by reference by many constructors in the QUESO library, and is available by reference from many classes as well.

The constructor of the environment class requires a communicator, the name of an options input file, and the eventual prefix of the environment in order for the proper options to be read (multiple environments can coexist, as explained further below).

The environment class has four primary tasks:

1. Assigns rank numbers, other than the world rank, to nodes participating in a parallel job,
2. Provides MPI communicators for generating a sequence of vectors in a distributed way,
3. Provides functionality to read options from the options input file (whose name is passed in the constructor of this environment class), and
4. Opens output files for messages that would otherwise be written to the screen (one output file per allowed rank is opened and allowed ranks can be specified through the options input file).

Let $S \geqslant 1$ be the number of problems a QUESO environment will be handling at the same time, in parallel. S has default value of 1 and is an option read by QUESO from the input file provided by the user. The QUESO environment class manages five types of communicators, referred to as:

1. *world*: MPI_WORLD_COMM;
2. *full*: communicator passed to the environment constructor, of size F and usually equal to the world communicator;
3. *sub*: communicator of size F/S that contains the number of MPI nodes necessary to solve a statistical IP or a statistical FP;
4. *self*: MPI_SELF_COMM, of size 1; and
5. *inter0*: communicator of size S formed by all MPI nodes that have subrank 0 in their respective subcommunicators.

A *subenvironment* in QUESO is the smallest collection of processors necessary for the proper run of the model code. An *environment* in QUESO is the collection of all subenvironments, if there is more than one subenvironment. Each subenvironment is able to generate a statistical inverse problem and/or a statistical forward problem; that is, each subenvironment is able to handle a “sub” Markov chain (a sequence) of vectors and/or a “sub” Monte Carlo sequence of output vectors. The “sub” sequences can be seen as forming a “unified” sequence in a distributed way. Indeed, the virtual class `uqVectorSequenceClass` provides "sub" and "unified" statistical operations.

Thus, if the model code requires 16 processors to run and the user decides to run 64 Markov chains in parallel, then the environment will consist of a total of $F = 1024$ processors and $S = 64$ subenvironments, each subenvironment with $F/S = 16$ processors. Any given computing node in a QUESO run has potentially five different ranks. Each subenvironment is assigned a subid varying from 0 (zero) to $S-1$, and is able to handle a statistical IP and/or a statistical FP. That is, each subenvironment is able to handle a *sub* Markov chain (a sequence) of vectors and/or a *sub* MC sequence of output vectors. The *sub* sequences form an unified sequence in a distributed way. QUESO takes care of the unification of results for the application programming and for output files. Of course, if the user is solving just one statistical problem with just one MPI node, then all ranks are equal to zero.

A QUESO subenvironment eventually prints messages to its own output file. In order for that to happen, the requirements are:

1. option `m_subDisplayFileName`, a string, must be different than the default value `"."`;
2. option `m_subDisplayAllowedSet`, a set of sub ids, must contain the id of the sub environment wanting to write a message to the output file;
3. the previous requirement is automatically satisfied if the option `m_subDisplayAllowAll`, a boolean, is set to 1 (the default value is 0);
4. the processor wanting to write a message to the output file must have sub rank 0 (zero).

If all requirements are satisfied, then QUESO will generate a file with name `<m_subDisplayFileName>_sub<sub id>.txt`. For instance, if `m_subDisplayFileName` is '`pROblem_775_`' then a node of sub rank 0 in sub environment 17 will write a message to the file '`pROblem_775_sub17.txt`'.

Figure 3.1.1 depicts class diagram for the environment class; and Figure 3.1.2 displays environment options class. Finally, the input file options for a QUESO environment, i.e., the options the user may set in his/her input file when using QUESO together with the application of interest, is presented in Table 3.1.1.

3.1.2 Vector

The Vector class handles all the vector operations carried out in QUESO, and currently has two derived classes: `uqGslVectorClass` and `uqTrilinosVectorClass`. `uqGslVectorClass` is based on the GSL vector structure; whereas `uqTrilinosVectorClass` is based on Trilinos Epetra vector structure.

A class diagram for `uqVectorClass` is presented in Figure 3.1.3; the reader may notice that the diagram also presents an extra class, `uqPetscVectorClass`, in order to show QUESO flexibility to the inclusion of other classes – this class has yet to be implemented.

Figure 3.1.1: The class diagram for the environment class described in Section 3.1.1.

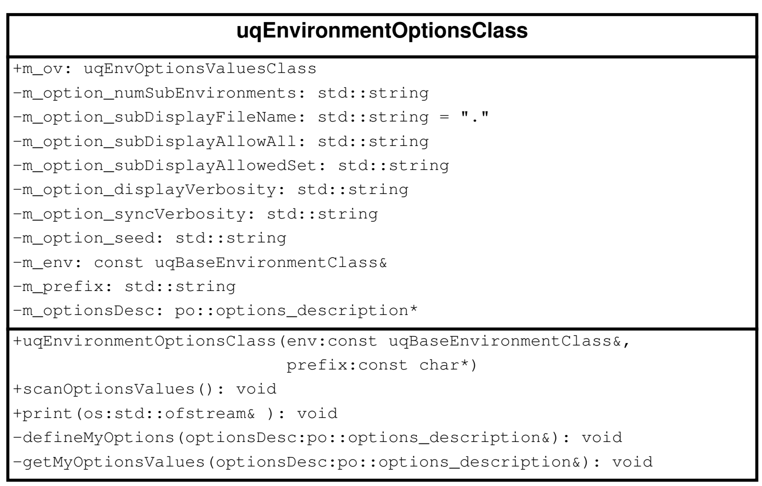


Figure 3.1.2: The environment options class.

Table 3.1.1: Input file options for a QUESO environment.

Option name	Default value	Description
⟨PREFIX⟩env_help		Produces help message for environment class
⟨PREFIX⟩env_numSubEnvironments	1	Number of subenvironments
⟨PREFIX⟩env_subDisplayFileName	"."	Output filename for sub-screen writing
⟨PREFIX⟩env_subDisplayAllowAll	0	Allows all subenvironments to write to output file
⟨PREFIX⟩env_subDisplayAllowedSet	""	Subenvironments that will write to output file
⟨PREFIX⟩env_displayVerbosity	0	Sets verbosity
⟨PREFIX⟩env_syncVerbosity	0	Sets syncronized verbosity
⟨PREFIX⟩env_seed	0	Set seed

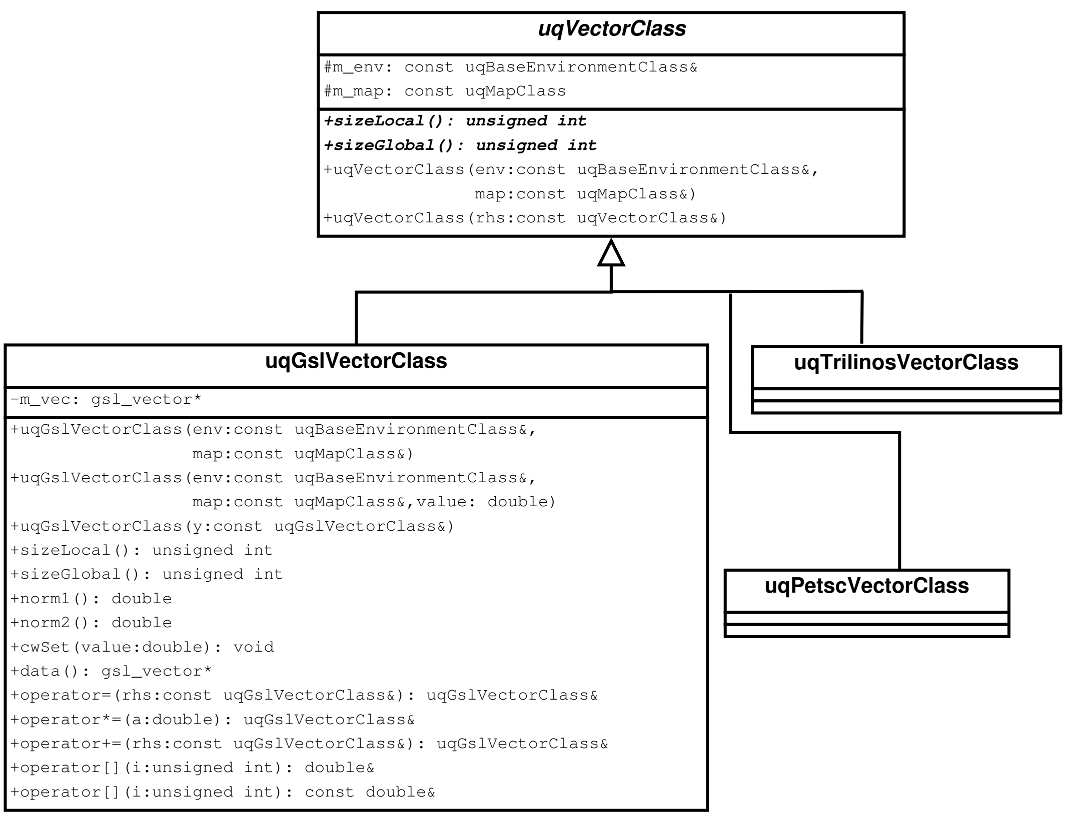


Figure 3.1.3: The class diagram for the vector class described in Section 3.1.2.

3.1.3 Matrix

The Matrix class handles all the matrix operations carried out in QUESO. Analogously to the vector class case described in the previous section, matrix class currently has two derived classes: `uqGslMatrixClass` and `uqTrilinosMatrixClass`. `uqGslMatrixClass` is based on the GSL matrix structure; whereas `uqTrilinosMatrixClass` is based on Trilinos Epetra matrix structure.

A class diagram for `uqMatrixClass` is presented in Figure 3.1.4; it displays some of its protected attributed together with some member functions. Again, the diagram displays in some detail the class `uqGslMatrixClass`, it shows without details the other inherited class, `uqTrilinosMatrixClass`, and indicated the possible inclusion of a third class, `uqPetscMatrixClass`.

Figure 3.1.4: The class diagram for the matrix class.

3.2 Templated Basic Classes

The classes in this group are: vector sets, subsets and spaces (Section 3.2.1), scalar and vector function classes (Section 3.2.2), and scalar and vector sequences (Section 3.2.3).

These classes constitute the core entities necessary for the formal mathematical definition and description of other entities, such as random variables, Bayesian solutions of inverse problems, sampling algorithms and chains.

3.2.1 Vector Set, Subset and Vector Space Classes

The vector set class is fundamental for the proper handling of many mathematical entities. Indeed, the definition of a scalar function such as $\pi : \mathbf{B} \subset \mathbb{R}^n \to \mathbb{R}$ requires the specification of the domain $\mathbf{B}$, which is a *subset* of the *vector space* $\mathbb{R}^n$, which is itself a *set*. Additionally, SIPs need a likelihood routine $\pi_{\text{like}} : \mathbb{R}^n \to \mathbb{R}_+$, and SFPs need a QoI routine $\mathbf{q} : \mathbb{R}^n \to \mathbb{R}^m$; the *sets* $\mathbb{R}^n$, $\mathbb{R}^m$, etc., are *vector spaces*.

The relationship amongst QUESO classes for handling sets, namely `uqVectorSetClass`; subsets, namely `uqVectorSubsetClass`; and vector spaces, namely `uqVectorSpaceClass` is sketched in Figure 3.2.1. An attribute of the *subset* class is the *vector space* which it belongs to, and in fact a reference to a vector space is required by the constructor of the subset class. An example of this case is the definition of a scalar function such as $\pi : \mathbf{B} \subset \mathbb{R}^n \to \mathbb{R}$ above.

The power of an object-oriented design is clearly featured here. The intersection subset derived class `uqIntersectionSubsetClass` is useful for handling a posterior PDF on Equation (1.2.1), since its domain is the intersection of the domain of the prior PDF with the domain of the likelihood function.

3.2.2 Scalar Function and Vector Function Classes

Joint PDF, marginal PDF, and CDF are all examples of scalar functions present in statistical problems. QUESO currently supports basic PDFs such as uniform and Gaussian and also more complex PDFs, such as the ones coming from a Bayesian analysis. They are implemented in the classes `uqUniformJointPdfClass`, `uqGaussianJointPdfClass`, and `uqBayesianJointPdfClass`, respectively. The posterior PDF may be represented within QUESO by `uqGenericJointPdfClass`. See Diagram 3.2.2 for the scalar function class.

The handling of vector functions within QUESO is also quite straightforward. Indeed, the definition of a vector function $\mathbf{q} : \mathbf{B} \subset \mathbb{R}^n \to \mathbb{R}^m$ requires only the extra specification of the image vector space $\mathbb{R}^m$. The classes representing the vector function class `uqGenericVectorFunctionClass` and `uqConstantVectorFunctionClass` are derived from `uqBaseVectorFunctionClass` and are presented in Diagram 3.2.3

3.2.3 Scalar Sequence and Vector Sequence Classes

The scalar sequence class contemplates *scalar* samples generated by an algorithm, as well as operations that can be done over them, e.g., calculation of means, variances, and convergence indices. Similarly, the vector sequence class contemplates *vector* samples and operations such as means, correlation matrices and covariance matrices.

Figures 3.2.4 and 3.2.5 display the class diagram for the scalar sequence and vector sequence classes, respectively.

Figure 3.2.1: The class diagram for vector set, vector subset and vector space classes, described in Section 3.2.1.

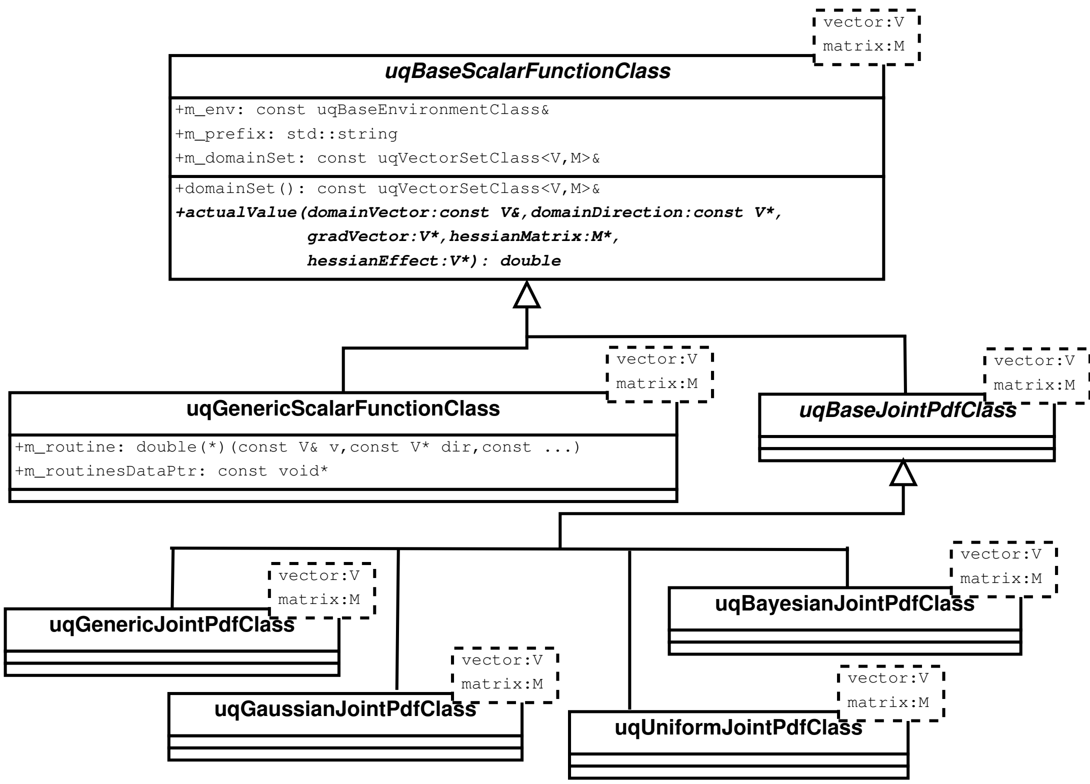


Figure 3.2.2: The class diagram for the scalar function class.

Figure 3.2.3: The class diagram for the vector function class described in Section 3.2.2.

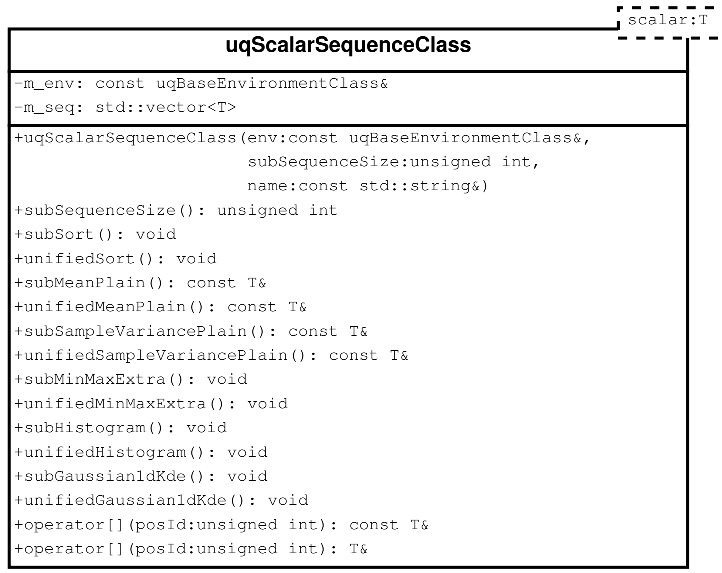


Figure 3.2.4: The class diagram for the scalar sequence class.

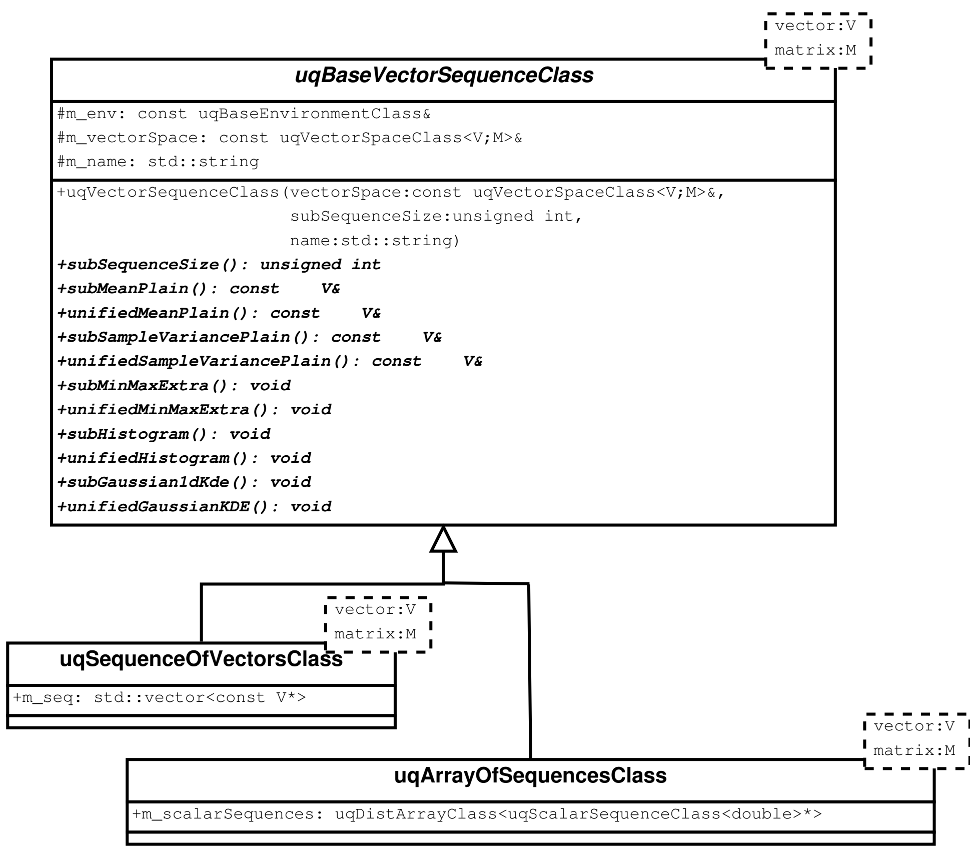


Figure 3.2.5: The class diagram for the vector sequence class.

3.3 Templated Statistical Classes

The classes in this group are: vector realizer, vector random variable, statistical inverse problem (and options), Metropolis-Hastings solver (and options), statistical forward problem (and options), Monte Carlo solver (and options), and Sequence statistical options.

For QUESO, a SIP has two input entities, a prior RV and a likelihood routine, and one output entity, the posterior RV, as shown in Chapter 1, Figure 1.2.3. Similarly, a SFP has two input entities, a input RV and a QoI routine, and one output entity, the output RV, as shown in Figure 1.2.2.

3.3.1 Vector Realizer Class

A *realizer* is an object that, simply put, contains a `realization()` operation that returns a sample of a vector RV. QUESO currently supports several realizers: uniform (implemented in `uqUniformVectorRealizerClass`), Gaussian (`uqGaussianVectorRealizerClass`), Log Normal (`uqLogNormalVectorRealizerClass`), Gamma (`uqGammaVectorRealizerClass`), Inverse Gamma (`uqInverseGammaVectorRealizerClass`) and Beta (`uqBetaVectorRealizerClass`), which are all derived from the base class `uqBaseVectorRealizerClass`.

QUESO conveniently provides the class `uqConcatenatedVectorRealizerClass`, which allows two distinct realizers to be concatenated. It also contains a *sequence realizer* class for storing samples of a MH algorithm.

3.3.2 Vector Random Variable Class

Vector RVs are expected to have two basic functionalities: compute the value of its PDF at a point, and generate realizations following such PDF. The joint PDF (`uqBaseJointPdfClass` and derived classes, see Section 3.2.2) and vector realizer (`uqBaseVectorRealizerClass` and derived classes, see Section 3.3.1) classes allow a straightforward definition and manipulation of vector RVs. Similarly to the vector realizer class above, QUESO also allows users to form new RVs through the concatenation of existing RVs (class `uqConcatenatedVectorRVClass`).

QUESO currently supports a few vector RVs such as uniform, Gaussian, Gamma and Beta, as depicted in Diagram 3.3.1. A derived class called *generic vector RV* allows QUESO to store the solution of an statistical IP: a *Bayesian joint PDF* becomes the PDF of the posterior RV, while a *sequence vector realizer* becomes the realizer of the same posterior RV.

3.3.3 Statistical Inverse Problem (and Options)

Similarly to its mathematical concepts, a SIP in QUESO also expects two input entities, a prior RV and a likelihood routine, and one output entity, the posterior RV. The SIP is represented in QUESO through the templated class `uqStatisticalInverseProblemClass<P_V,P_M>`, which is illustrated in Figure 3.3.2. One important characteristic of the QUESO design is that it

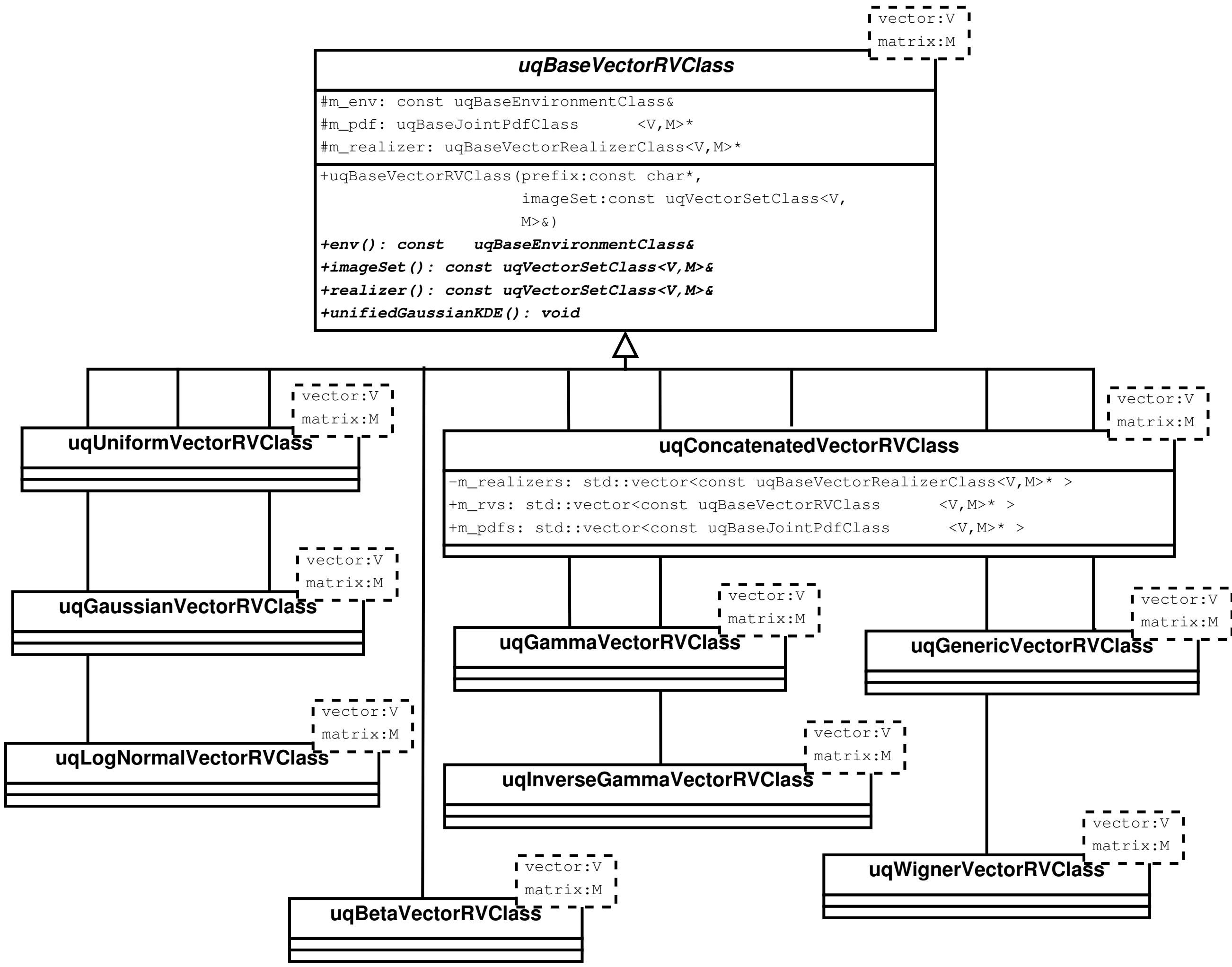


Figure 3.3.1: The class diagram for the vector random variable class.

separates 'what the problem is' from 'how the problem is solved'. The prior and the posterior RV are instances of the `uqBaseVectorRvClass<V,M>` class, while the likelihood function is an instance of the `uqBaseScalarFunctionClass<V,M>` class.

The solution of a SIP is computed by calling the `solveWithBayesMetropolisHastings()` member function of the `uqStatisticalInverseProblemClass<P_V,P_M>` class. Upon return from a solution operation, the posterior RV is available through the `postRv()` member function.

Figure 3.3.3 displays the statistical inverse problem options class, i.e. that class that handles a variety of options for solving the SIP. Such options may be provided to QUESO by the user's input file; and they are listed in Table 3.3.1.

```
                                                              param vector:P_V
                                                              param matrix:P_M
uqStatisticalInverseProblemClass
-----------------------------------------------------------------------------
-m_env: const uqBaseEnvironmentClass&
-m_optionsObj: uqStatisticalInverseProblemOptionsClass*
-m_priorRv: const uqBaseVectorRVClass<P_V,P_M>&
-m_likelihoodFunction: const uqBaseScalarFunctionClass <P_V,P_M>&
-m_postRv: uqGenericVectorRVClass<P_V,P_M>&
-m_solutionDomain: uqVectorSetClass<P_V,P_M>*
-m_solutionPdf: uqBaseJointPdfClass<P_V,P_M>*
-m_solutionRealizer: uqBaseVectorRealizerClass<P_V,P_M>*
-m_mhSeqGenerator<P_V,P_M>*: uqMetropolisHastingsSGClass
-m_chain: uqBaseVectorSequenceClass<P_V,P_M>*
-----------------------------------------------------------------------------
+uqStatisticalInverseProblemClass(prefix:const char*,
                                  alternativeOptionsValues:const uqSipOptionsValuesClass*,
                                  priorRv:const uqBaseVectorRVClass<P_V;P_M>&,
                                  likelihoodFunction:const uqBaseScalarFunctionClass<P_V;P_M>&,
                                  postRv:uqGenericVectorRVClass&)
+computeSolutionFlag(): bool
+solveWithBayesMetropolisHastings(alternativeOptionsValues:const uqMhOptionsValuesClass*,
                                  initialValues:const P_V&,
                                  initialProposalCovMatrix:const P_M*): void
+priorRv(): const uqBaseVectorRVClass   <P_V;P_M>&
+postRv(): const uqGenericVectorRVClass<P_V;P_M>&
```

Figure 3.3.2: The statistical inverse problem class. It implements the representation in Figure 1.2.3.

```
uqBaseStatisticalInverseProblemOptionsClass
-----------------------------------------------------------------------------
+m_computeSolution: bool = 1
+m_dataOutputFileName: std::string = "."
+m_dataOutputAllowSet: std::set<unsigned int> = empty
+m_prefix: std::string
-m_env: const uqBaseEnvironmentClass&
-m_optionsDesc: po::options_description*
-----------------------------------------------------------------------------
+uqStatisticalInverseProblemOptionsClass(env:const uqBaseEnvironmentClass&,
                                         prefix:const char*)
+scanOptionsValues(): void
+print(std::ostream& os): void
-defineMyOptions(optionsDesc:po::options_description&): void
-getMyOptionsValues(optionsDesc:po::options_description&): void
```

Figure 3.3.3: The statistical inverse problem options class.

Table 3.3.1: Input file options for a QUESO statistical inverse problem.

Option name	Default Value	Description
⟨PREFIX⟩`ip_help`		Produces help message for statistical inverse problem
⟨PREFIX⟩`ip_computeSolution`	1	Computes solution process
⟨PREFIX⟩`ip_dataOutputFileName`	"."	Name of data output file
⟨PREFIX⟩`ip_dataOutputAllowedSet`	""	Subenvironments that will write to data output file

3.3.4 Metropolis-Hastings Solver (and Options)

The templated class that represents a Metropolis-Hastings generator of samples in QUESO is `uqMetropolisHastingsSGClass<P_V,P_M>`, where SG stands for 'Sequence Generator'. This class implements the DRAM algorithm of Haario, Laine, Mira and Saksman [13] together with an operation based on the core routine at the MCMC toolbox for MATLAB [24]. In fact,, the reader may notice that the example available in the QUESO build tree directory `examples/statisticalInverseProblem` is closely related to the 'normal example' in the toolbox.

The Metropolis-Hastings sequence generator class is depicted in Figure 3.3.4; the Metropolis-Hastings sequence generator options class is depicted in Figure 3.3.5; whereas its options are presented in Table 3.3.2.

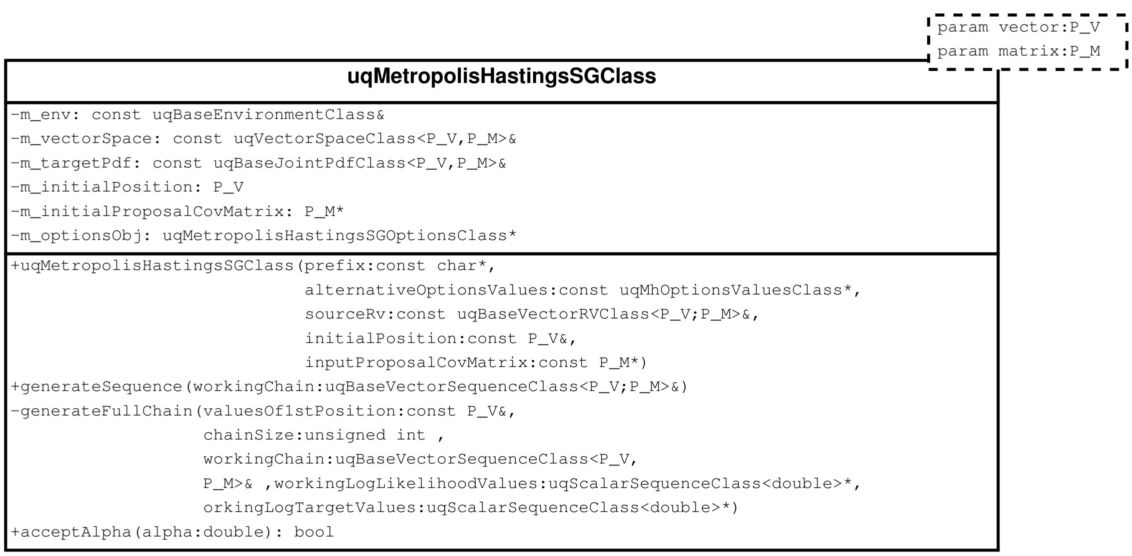


Figure 3.3.4: The Metropolis-Hastings sequence generator class.

Table 3.3.2: Input file options for a QUESO Metropolis-Hastings solver.

Option Name	Default Value
⟨PREFIX⟩mh_dataOutputFileName	"."
⟨PREFIX⟩mh_dataOutputAllowAll	0
⟨PREFIX⟩mh_initialPositionDataInputFileName	"."
⟨PREFIX⟩mh_initialPositionDataInputFileType	"m"
⟨PREFIX⟩mh_initialProposalCovMatrixDataInputFileName	"."
⟨PREFIX⟩mh_initialProposalCovMatrixDataInputFileType	"m"
⟨PREFIX⟩mh_rawChainDataInputFileName	"."
⟨PREFIX⟩mh_rawChainDataInputFileType	"m"
⟨PREFIX⟩mh_rawChainSize	100
⟨PREFIX⟩mh_rawChainGenerateExtra	0
⟨PREFIX⟩mh_rawChainDisplayPeriod	500
⟨PREFIX⟩mh_rawChainMeasureRunTimes	1
⟨PREFIX⟩mh_rawChainDataOutputPeriod	0
⟨PREFIX⟩mh_rawChainDataOutputFileName	"."
⟨PREFIX⟩mh_rawChainDataOutputFileType	"m"
⟨PREFIX⟩mh_rawChainDataOutputAllowAll	0
⟨PREFIX⟩mh_filteredChainGenerate	0
⟨PREFIX⟩mh_filteredChainDiscardedPortion	0.
⟨PREFIX⟩mh_filteredChainLag	1
⟨PREFIX⟩mh_filteredChainDataOutputFileName	"."
⟨PREFIX⟩mh_filteredChainDataOutputFileType	"m"
⟨PREFIX⟩mh_filteredChainDataOutputAllowAll	0
⟨PREFIX⟩mh_displayCandidates	0
⟨PREFIX⟩mh_putOutOfBoundsInChain	1
⟨PREFIX⟩mh_tkUseLocalHessian	0
⟨PREFIX⟩mh_tkUseNewtonComponent	1
⟨PREFIX⟩mh_drMaxNumExtraStages	0
⟨PREFIX⟩mh_drDuringAmNonAdaptiveInt	1
⟨PREFIX⟩mh_amKeepInitialMatrix	0
⟨PREFIX⟩mh_amInitialNonAdaptInterval	0
⟨PREFIX⟩mh_amAdaptInterval	0
⟨PREFIX⟩mh_amAdaptedMatricesDataOutputPeriod	0
⟨PREFIX⟩mh_amAdaptedMatricesDataOutputFileName	"."
⟨PREFIX⟩mh_amAdaptedMatricesDataOutputFileType	"m"
⟨PREFIX⟩mh_amAdaptedMatricesDataOutputAllowAll	0
⟨PREFIX⟩mh_amEta	1.
⟨PREFIX⟩mh_amEpsilon	1×10^{-5}
⟨PREFIX⟩mh_enableBrooksGelmanConvMonitor	0
⟨PREFIX⟩mh_BrooksGelmanLag	100

```
uqMetropolisHastingsSGOptionsClass
-m_env: const uqBaseEnvironmentClass&
-m_optionsDesc: po::options_description*
-m_option_dataOutputFileName: std::string = "."
-m_option_dataOutputAllowedSet: std::string = empty
-m_option_rawChain_size: unsigned int = 100
- m_option_rawChain_displayPeriod: unsigned int = 500
-m_option_rawChain_dataOutputFileName: std::string
-m_option_rawChain_dataOutputAllowedSet: std::string = empty
-m_filteredChainGenerate: std::string
-m_option_filteredChain_discardedPortion: std::string
-m_option_filteredChain_lag: std::string
-m_option_filteredChain_dataOutputFileName: std::string = "."
-m_option_filteredChain_dataOutputAllowedSet: std::string
-m_option_dr_maxNumExtraStages: std::string
-m_option_dr_listOfScalesForExtraStages: std::string
-m_option_am_initialNonAdaptInterval: std::string
-m_option_am_adaptInterval: std::string
-m_option_am_eta: std::string
-m_option_am_epsilon: std::string
+m_prefix: std::string
+m_ov: uqMhOptionsValuesClass
+uqMetropolisHastingsSGOptionsClass(env:const uqBaseEnvironmentClass&,
                                    prefix:const char*)
+scanOptionsValues(): void
+print(os:std::ofstream& ): void
-defineMyOptions(optionsDesc:po::options_description&): void
-getMyOptionsValues(optionsDesc:po::options_description&): void
```

Figure 3.3.5: The Metropolis-Hastings sequence generator options class.

3.3.5 Statistical Forward Problem (and Options)

A SFP in QUESO also has two input entities, the input (parameter) RV and a QoI function, and one output entity, the QoI RV. The SIP is represented through the templated class `uqStatisticalForwardProblemClass<P_V,P_M,Q_V,Q_M >`, which diagram is presented in Figure 3.3.6. Again, the types `P_V` and `Q_V` of vectors and types `P_M` and `Q_M` of matrices, where `P_` stands for 'parameter' and `Q_` stands for 'quantities of interest'.

The input RV and the output QoI RV are instances of the `uqBaseVectorRvClass<P_V,P_M>` class, while the QoI function is an instance of `uqBaseVectorFunctionClass<P_V,P_M,Q_V,Q_M>`. In the template parameters, the prefix `P_` refers to the parameters, whereas the prefix `Q_` refers to the QoIs.

In order to find the solution of a SFP, one must call the `solveWithMonteCarlo()` member function of the `uqStatisticalForwardProblemClass<P_V,P_M>` class. Upon return from a solution operation, the QoI RV is available through the `qoiRv()` member function. Such QoI RV is able to provide: a vector realizer through the operation `'qoiRv().realizer()'`, which returns an instance of the class `'uqBaseVectorRealizerClass<Q_V,Q_M>'`.

Figure 3.3.7 displays the statistical forward problem options class, i.e. that class that handles a variety of options for solving the SFP. Such options may be provided to QUESO at the user's input file; and they are listed in Table 3.3.3. In the table, `p-q` stands for parameter–quantity of interest.

```
param vector:P_V
param matrix:P_M
qoi vector:Q_V
qoi matrix:Q_M

uqStatisticalForwardProblemClass
-m_env: const uqBaseEnvironmentClass&
-m_optionsObj: uqStatisticalForwardProblemOptionsClass*
-m_paramRv: const uqBaseVectorRVClass<P_V,P_M>&
-m_qoiFunction: const uqBaseVectorFunctionClass <P_V,P_M,Q_V,Q_M>&
-m_qoiRv: uqGenericVectorRVClass<Q_V,Q_M>&
-m_solutionPdf: uqBaseJointPdfClass<P_V,P_M>*
-m_solutionRealizer: uqBaseVectorRealizerClass<Q_V,Q_M>*
-m_mcSeqGenerator: uqMonteCarloSGClass <P_V,P_M,Q_V,Q_M>*
-m_paramChain: uqBaseVectorSequenceClass<Q_V,Q_M>*
-m_qoiChain: uqBaseVectorSequenceClass<Q_V,Q_M>*

+uqStatisticalForwardProblemClass(prefix:const char*,
                                  alternativeOptionsValues:const uqSfpOptionsValuesClass*,
                                  paramRv:const uqBaseVectorRVClass<P_V;P_M>&,
                                  qoiFunction:const uqBaseVectorFunctionClass<P_V;P_M;Q_V;Q_M>&,
                                  qoiRv:uqGenericVectorRVClass<Q_V;Q_M>&)
+computeSolutionFlag(): bool
+olveWithMonteCarlo(alternativeOptionsValues:const uqMcOptionsValuesClass*): void
+qoiRv(): const uqGenericVectorRVClass<Q_V;Q_M>&
```

Figure 3.3.6: The statistical forward problem class. It implements the representation in Figure 1.2.2.

```
uqStatisticalForwardProblemOptionsClass
+m_prefix: std::string
+m_ov: uqSfpOptionsValuesClass
-m_env: const uqBaseEnvironmentClass&
-m_option_computeSolution: std::string
-m_option_dataOutputFileName: std::string = "."
-m_option_dataOutputAllowedSet: std::string
-m_optionsDesc: po::options_description*

+uqStatisticalForwardProblemOptionsClass(env:const uqBaseEnvironmentClass&,
                                         prefix:const char*)
+scanOptionsValues(): void
+print(os:std::ostream& ): void
-defineMyOptions(optionsDesc:po::options_description&): void
-getMyOptionsValues(optionsDesc:po::options_description&): void
```

Figure 3.3.7: The statistical forward problem options class.

Table 3.3.3: Input file options for a QUESO statistical forward problem.

Option Name	Default Value	Description
⟨PREFIX⟩`fp_computeSolution`	1	Computes the solution process
⟨PREFIX⟩`fp_computeCovariances`	1	Compute `p-q` covariances
⟨PREFIX⟩`fp_computeCorrelations`	1	Compute `p-q` correlations
⟨PREFIX⟩`fp_dataOutputFileName`	"."	Name of data output file
⟨PREFIX⟩`fp_dataOutputAllowedSet`	""	Subenvironments that will write to data output file

3.3.6 Monte Carlo Solver (and Options)

The templated class that implements a Monte Carlo generator of samples within QUESO is `uqMonteCarloSGClass<P_V,P_M,Q_V,Q_M>`, as illustrated in Figure 3.3.8. This class has the requirement that the image set of the vector random variable and the domain set of the QoI function belong to vector spaces of equal dimensions. If the requirements are satisfied, the class constructor reads input options that begin with the string ‘`<PREFIX>_mc_`’ (See Table 3.3.4). Options reading is handled by class `uqMonteCarloOptionsClass`, which is illustrated in Figure 3.3.9.

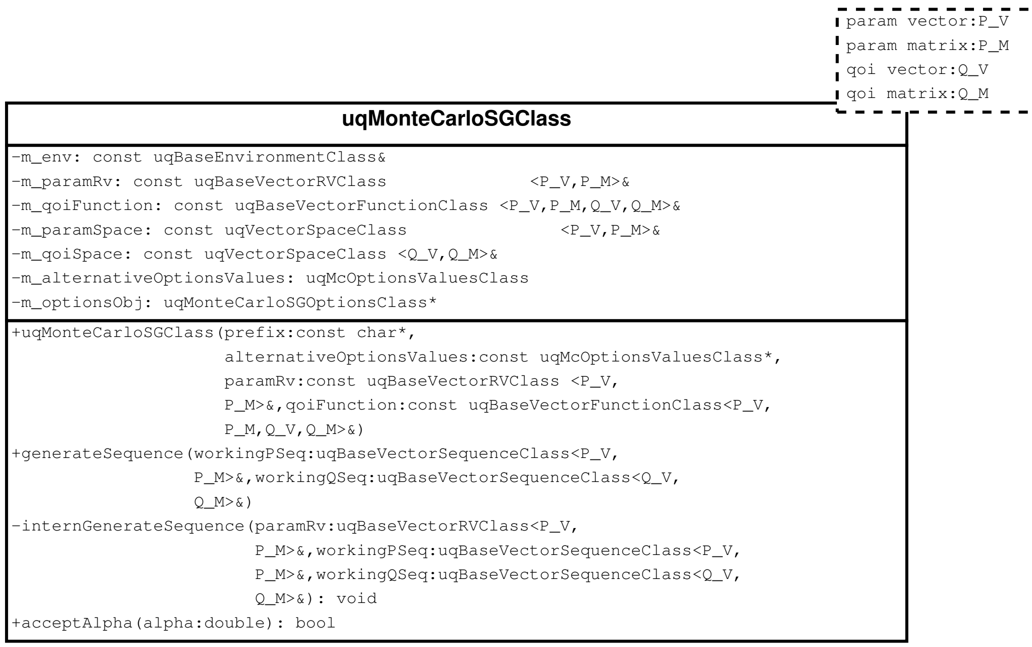


Figure 3.3.8: The Monte Carlo sequence generator class.

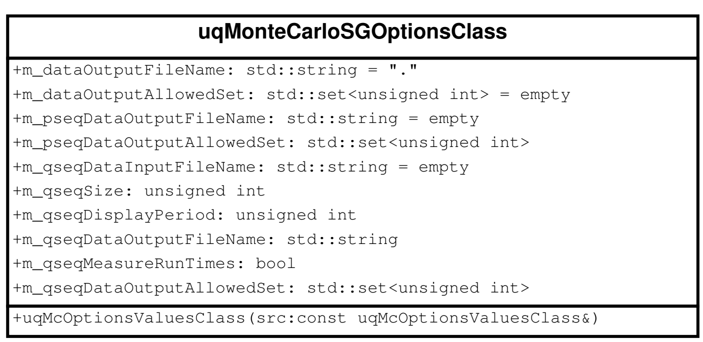


Figure 3.3.9: The Monte Carlo sequence generator options class.

Table 3.3.4: Input file options for a QUESO statistical forward problem solved via Monte Carlo algorithm.

Option Name	Default Value
⟨PREFIX⟩`mc_dataOutputFileName`	`"."`
⟨PREFIX⟩`mc_dataOutputAllowedSet`	
⟨PREFIX⟩`mc_pseq_dataOutputFileName`	`"."`
⟨PREFIX⟩`mc_pseq_dataOutputAllowedSet`	
⟨PREFIX⟩`mc_qseq_dataInputFileName`	`"."`
⟨PREFIX⟩`mc_qseq_size`	`100`
⟨PREFIX⟩`mc_qseq_displayPeriod`	`500`
⟨PREFIX⟩`mc_qseq_measureRunTimes`	`0`
⟨PREFIX⟩`mc_qseq_dataOutputFileName`	`"."`
⟨PREFIX⟩`mc_qseq_dataOutputAllowedSet`	

3.4 Miscellaneous Classes and Routines

As the name suggests, QUESO miscellaneous classes and routines have a variety of routines. For instance, the function `uqMiscReadDoublesFromString` is used for reading the options input files and assigning the values to the respective variables, in `uqMonteCarloSGOptionsClass::getMyOptionValues` and in `uqMetropolisHastingsSGOptionsClass::getMyOptionValues`.

QUESO class `uqBaseOneDGridClass` generates grids necessary for calculating the CDF of a RV; it is required by class `uqArrayOfOneDGridsClass`, which, in turn, is used in both classes: `uqStatisticalForwardProblemClass` and `uqStatisticalInverseProblemClass`.

CHAPTER 4

Important Remarks

At this point, the user may feel comfortable and ready to start his/her validation and calibration exercises using QUESO. There are, however, a few quite important remarks that will make the linkage of the QUESO Library with the user application code possible. They are addressed in the following sections.

4.1 Revisiting Input Options

Input options are read from the QUESO input file, whose name is required by the constructor of the QUESO environment class. Herein, suppose that no prefix is defined, i.e., nothing will precede the input variables names (`PREFIX = ""` in Tables 3.1.1 – 3.3.4). An example of the use of prefixes may be found in the input file `tgaCycle.inp` under the subdirectory `/examples/validationCycle/` of QUESO installation tree.

The first part of a input file commonly handles the environment options. he variable assignment `env_numSubEnvironments = 1` indicates to QUESO that only one subenvironment should be used. The variable assignment `env_subDisplayFileName = outputData/ display` create both the subdirectory `outputData/` and a file named `display_sub0.txt` that contains all the options listed in the input file together with more specific information, such as the chain run time and the number of delayed rejections. The existence of file `display_sub0.txt` allows, for instance, the user in verifying the actual parameters read by QUESO.

For an SIP, the user may set up variables related to the DRAM algorithm. Six important variables are: `ip_mh_dr_maxNumExtraStages` defines how many extra candidates will be generated; `ip_mh_dr_listOfScalesForExtraStages` defines the list s of scaling factors that will multiply the covariance matrix. The variable `ip_mh_am_initialNonAdaptInterval` defines the initial interval in which the proposal covariance matrix will not be changed; whereas `ip_mh_am_adaptInterval` defines the size of the interval in which each adapted proposal covariance matrix will be used. `ip_mh_am_eta` is a factor used to scale the proposal covariance matrix, usually set to be $2.4^2/d$, where d is the dimension of the problem [25, 13]. Finally, `ip_mh_am_epsilon` is the covariance regularization factor used in the DRAM algorithm.

For a SFP, the variable assignment `fp_computeSolution = 1` tells QUESO to compute the solution process; the assignment `fp_computeCovariances = 1`, instructs QUESO to compute parameter–QoI covariances, and analogously, `fp_computeCorrelations = 1` inform QUESO to compute parameter–QoI correlations. The name of the data output file can be set with variable `fp_dataOutputFileName arg`; and `fp_dataOutputAllowedSet` defines which subenvironments will write to data output file.

An example a complete input file used by QUESO to solve a SIP–SFP is presented in Section 5.4; however every application example included in QUESO build and installation directories `examples` has an options input file and the user is invited to familiarize him/herself with them.

4.2 Revisiting Priors

QUESO offers a variety of prior distributions: uniform, Gaussian, Beta, Gamma, Inverse Gamma, and Log Normal. Also, QUESO presents the option of concatenating any of those priors, through the Concatenated prior.

Concatenated priors are employed in problems with multiple random parameters. They allow one random parameter to have a different prior distribution then other; i.e., one variable may have a uniform prior distribution whereas other may have a Gaussian prior distribution.

It is important to notice that, in order to set a Gaussian prior, besides providing the mean, the user must also supply the variance, not the standard deviation.

4.3 Running with Multiple Chains or Monte Carlo Sequences

As presented in the previous section, the variable `env_numSubEnvironments` determines how many subenvironments QUESO will work with. Thus, if `env_numSubEnvironments=1`, then only one subenvironment will be used, and QUESO will use only one set on Monte Carlo chains of size defined by ones of the variables `ip_mh_rawChain_size` or `fp_mc_qseq_size`, depending either the user is solving a SIP or a SFP.

If the user wants to run QUESO with multiple chains or Monte Carlo sequences, then two variables have to be set in QUESO input file: `env_numSubEnvironments` = N_s, with $N_s > 1$ is the number of chains and/or Monte Carlo sequences of samples; and `env_seed` = $-z$, with $z \geqslant 1$, so that each processor sets the seed to value MPI_RANK+z. It is crucial that `env_seed` takes a negative value, otherwise all chain samples are going to be the same.

Also, the total number N_p of processors in the full communicator, usually named MPI_COMM_WORLD, needs to be a multiple of N_s.

4.4 Running with Models that Require Parallel Computing

It is possible to run QUESO with models that require parallel computing as long as total number of processors N_p is multiple of the number of subenvironments N_s. QUESO will internally create N_s subcommunicators, each of size N_p/N_s, and make sure that the likelihood and QoI routines are called for all processors in these subcommunicators – the likelihood/QoI routine will have available a communicator of size N_p/N_s. For instance, if $N_p = 2048$ and $N_s = 16$, then each likelihood/QoI will have available a communicator of size 128. s Each subcommunicator is accessible through `env.subComm()`. At the end of the simulation, there will be a total of N_s chains.

The user, however, must keep in mind the possible occurrence of race condition, especially in the case where the application is a black box and files are being accessed constantly (e.g. data is being written and read).

4.5 A Requirement for the DRAM Algorithm

Besides setting up the variables related to the DRAM algorithm in the input file, as described in Section 4.1 above, the user must also provide an initialized <u>covariance matrix</u> before calling the DRAM solver, `solveWithBayesMetropolisHastings(...)`, in his/her application code.

It is worth to note that this is rather a DRAM requirement [25], not a QUESO limitation. An example of the instantiation and initialization of a proposal covariance matrix and its subsequent use is presented in lines 145-147 of Listings 5.3, Section 5.3.

CHAPTER 5

An Application Example

This chapter presents an example of how to use QUESO in order to develop an application that solves a statistical inverse problem and a statistical forward problem, where the solution of the former serves as input to the later.

Section 5.1 gives the mathematical formulation of the statistical inverse problem, and presents the required tools and approach for solving it (experimental data, Bayesian approach, and a short overview of DRAM algorithm). Section 5.2 gives the mathematical formulation of the statistical forward problem, its input RV and QoI function, and indicates the statistical method for its solution, the Monte Carlo algorithm.

Section 5.3 presents the codes that translate the mathematical language into C++ using the QUESO classes and algorithms; Section 5.4 shows the input file that contains a list of options for the Markov chain algorithm (SIP) and the Monte Carlo algorithm (SFP) which will be used by QUESO classes and algorithms. Instructions on how to compile and run the code are presented in Section 5.5. Finally, Section 5.6 shows how to plot figures using Matlab and the output data generated by the application. All the program examples provided in this document are compatible with QUESO 0.46.0.

5.1 A Statistical Inverse Problem

The example described in this section consists of a statistical inverse problem which infers the acceleration due to gravity for an object in free fall near the surface of the Earth. Later, the inferred acceleration of gravity will be used in a statistical forward problem to propagate uncertainty in the calculation the motion of an object in projectile movement.

5.1.1 Mathematical Model for the Statistical Inverse Problem

A possible deterministic mathematical model for the vertical motion of an object in free fall near the surface of the Earth is given by

$$h(t) = -\frac{1}{2}gt^2 + v_0 t + h_0. \tag{5.1.1}$$

where v_0 $[m/s]$ is the initial velocity, h_0 $[m]$ is the initial altitude, $h(t)$ $[m]$ is the altitude with respect to time, t $[s]$ is the elapsed time, and g $[m/s^2]$ is the magnitude of the acceleration due to gravity (the parameter which cannot be directly measured and will be statistically inferred).

5.1.2 Experimental Data

We assume that the experiment of allowing an object to fall from different altitudes with zero initial velocity has been repeatedly conducted (See Figure 5.1.1). The data collected, e.g. $\mathbf{d}$, is displayed in Table 5.1.1; the standard deviations, σ's, refer to the uncertainties in the measured times during the experiment execution [1].

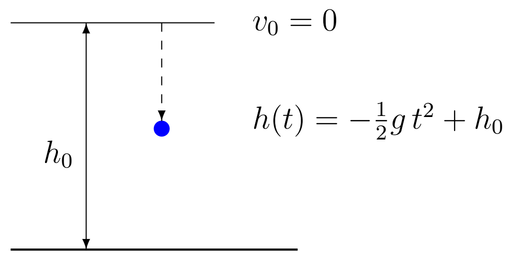


Figure 5.1.1: An object falls from altitude h_0 with zero initial velocity ($v_0 = 0$).

5.1.3 The Prior RV, Likelihood and Posterior RV

In a straightforward classical interpretation of Bayesian inference, the prior signifies the modeler's honest opinion about the unknown. For the gravity inference problem, let's assume that gravity varies uniformly in the interval [8,11], or, in other words, we chose uniform prior distribution in that interval:

$$\pi_{\text{prior}} = \mathcal{U}(8, 11). \tag{5.1.2}$$

We choose the usual likelihood function:

$$\pi_{\text{like}}(\mathbf{d}|\boldsymbol{\theta}) \propto \exp\left\{-\frac{1}{2}[\mathbf{y}(\boldsymbol{\theta}) - \mathbf{d}]^T [\mathbf{C}(\boldsymbol{\theta})]^{-1} [\mathbf{y}(\boldsymbol{\theta}) - \mathbf{d}]\right\}, \tag{5.1.3}$$

where $\mathbf{C}(\boldsymbol{\theta})$ is a given covariance matrix, $\mathbf{d}$ denotes experimental data, $\mathbf{y}(\boldsymbol{\theta})$ is the model output data.

Table 5.1.1: Measurement data $\mathbf{d}$ of size $n_d = 14$. The object falls from altitude h_0 in t seconds, with standard deviation of σ seconds in the time measurement [1].

altitude $[m]$	time $[s]$	Std. Dev. σ $[s]$
10	1.41	0.02
20	2.14	0.12
30	2.49	0.02
40	2.87	0.01
50	3.22	0.03
60	3.49	0.01
70	3.81	0.03
80	4.07	0.03
90	4.32	0.03
100	4.47	0.05
110	4.75	0.01
120	4.99	0.04
130	5.16	0.01
140	5.26	0.09

Recalling the deterministic model for the acceleration of gravity (5.1.1) with zero initial velocity, the information provided in Table 5.1.1, and Equation (5.1.3); and, additionally, invoking the nomenclature used in Section 1.2, we have:

$$\boldsymbol{\theta} \stackrel{\text{def.}}{=} g, \quad \mathbf{y}(\boldsymbol{\theta}) = \begin{bmatrix} \sqrt{\dfrac{2h_1}{g}} \\ \sqrt{\dfrac{2h_2}{g}} \\ \vdots \\ \sqrt{\dfrac{2h_{n_d}}{g}} \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_{n_d} \end{bmatrix}, \quad \mathbf{C}(\boldsymbol{\theta}) = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \sigma_{n_d}^2 \end{bmatrix}, \tag{5.1.4}$$

where $n_d = 14$ is the number of data points in Table 5.1.1.

Now we are ready to evoke Bayes' formula in order to obtain the posterior PDF $\pi_{\text{post}}(\boldsymbol{\theta})$:

$$\pi_{\text{post}}(\boldsymbol{\theta}|\mathbf{d}) \propto \pi_{\text{like}}(\mathbf{d}|\boldsymbol{\theta})\, \pi_{\text{prior}}(\boldsymbol{\theta}). \tag{5.1.5}$$

5.1.4 Algorithms for solving the Statistical Inverse Problem

The goal of inference is to characterize the posterior PDF, or to evaluate point or interval estimates based on the posterior [18]. Samples from posterior can be obtained using Markov chain Monte Carlo (MCMC) which require only pointwise evaluations of the unnormalized posterior. The resulting samples can then be used to either visually present the posterior or its marginals, or to construct sample estimates of posterior expectations. Examples of MCMC are: the Metropolis-Hastings (MH) algorithm [28, 15], the Delayed Rejection (DR) algorithm [10, 29], and Adaptive Metropolis (AM) [14] which are combined together in the DRAM algorithm [13].

DRAM, the Delayed Rejection Adaptive Metropolis algorithm, is implemented in QUESO and used to solve the gravity inference problem. There are six variables in the QUESO input file used to set available options for the DRAM algorithm; they are presented and explained in details in Section 5.4.

5.2 A Statistical Forward Problem

In this section we describe a statistical forward problem of predicting the distance traveled by a projectile launched at a given angle and altitude, and using a calibrated magnitude of the acceleration of gravity. This calibrated (inferred) gravity is the Bayesian solution (posterior PDF) of the inverse problem described in Section 5.1.

5.2.1 Mathematical Model for the Statistical Forward Problem

Projectile motion refers to the motion of an object projected into the air at an angle, e.g. a soccer ball being kicked, a baseball being thrown, or an athlete long jumping. Supposing the object does not have a propulsion system and neglecting air resistance, then the only force acting on the object is a constant gravitational acceleration g.

A possible deterministic two-dimensional mathematical model for the vertical motion of an object projected from near the surface of the Earth is given by

$$v_x = v_{0x} \tag{5.2.1}$$

$$v_y = v_{0y} - gt \tag{5.2.2}$$

$$x = v_{0x}t \tag{5.2.3}$$

$$h = h_0 + v_{0y}t - \frac{1}{2}gt^2 \tag{5.2.4}$$

where h_0 is the initial height, $x = x(t)$ is the distance traveled by the object, $\mathbf{v_0} = (v_{0x}, v_{0y})$ is the initial velocity, $v_{0x} = v_0 \cos(\alpha)$, $v_{0y} = v_0 \sin(\alpha)$, and $v_0 = \|\mathbf{v_0}\|^2$. Figure 5.2.1 displays the projectile motion of an object in these conditions.

For this example, we assume that $h_0 = 0$ m, $\alpha = \pi/4$ radians, $v_0 = 5$ m/s, all deterministic variables; and g is the solution of the SIP described in Section 5.1.

Since a PDF is assigned to parameter g; thus, the output of the mathematical model (5.2.1) becomes a random variable, thus we have a statistical forward problem.

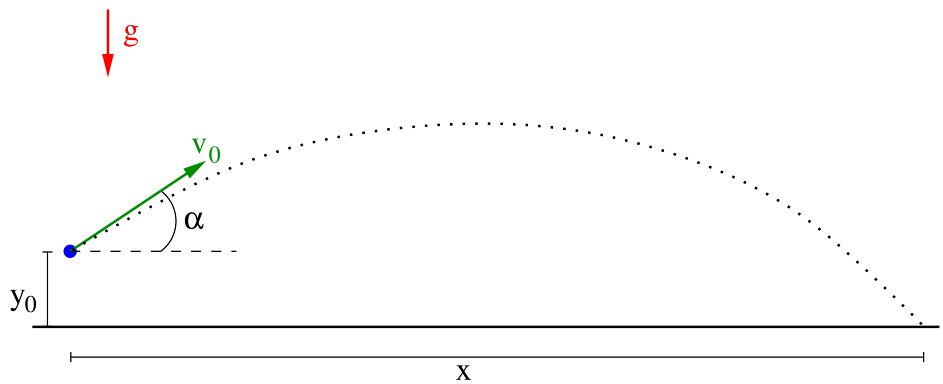


Figure 5.2.1: Object traveling with projectile motion.

5.2.2 The Input RV, QoI Function and Output RV

The input random variable for the statistical forward problem is the acceleration of gravity g, which is also the solution (posterior PDF) of the inverse problem described in Section 5.1. The output random variable for this example is the distance x traveled by an object in projectile motion. Note that, since there is uncertainty in the parameter g (g is given as a PDF), one can expect that this uncertainty will be propagated to x, which will also be given as a PDF.

Combining the expressions in Equation 5.2.1 and rearranging them, we have that QoI function for x is:

$$x = \frac{v_0 \cos\alpha}{g}\left(v_0 \sin\alpha + \sqrt{(v_0 \sin\alpha)^2 + 2g\, y_0}\right). \tag{5.2.5}$$

where y is the distance traveled and our quantity of interest (QoI).

5.2.3 Algorithms for solving the Statistical Forward Problem

The Monte Carlo method is commonly used for analyzing uncertainty propagation, where the goal is to determine how random variation, lack of knowledge, or error affects the sensitivity, performance, or reliability of the system that is being modeled [33].

Monte Carlo works by using random numbers to sample, according to a PDF, the 'solution space' of the problem to be solved. Then, it iteratively evaluates a deterministic model using such sets of random numbers as inputs.

Monte Carlo is implemented in QUESO and it is the chosen algorithm to compute a sample of the output RV (the QoI) of the SFP for each given sample of the input RV.

5.3 Application Code

The source code for the SIP and the SFP is composed of 7 files. Three of them are common for both problems: `gravity_main.C`, `gravity_compute.h` and `gravity_compute.C`,; they combine both problems and use the solution of the SIP (the posterior PDF for the gravity) as an input for the SFP and are presented, respectively, in Listings 5.1, 5.2 and 5.3. Two of files specifically handle the SIP: `gravity_likelihood.h`, and `gravity_likelihood.C`, and are displayed in Listings 5.4 and 5.5. Finally, the files specific for the SFP are `gravity_qoi.h` and `gravity_qoi.C`, and they are presented in Listings 5.6 and 5.7.

```
/*------------------------------------------------------------------
 * Brief description of this file:
 *
 * This is an example of how to use QUESO classes and algorithms in
 * order to define and solve a statistical inverse problem (SIP) and/
 * or a statistical forward problem (SFP).
 *
 * The SIP consists on calibrating the magnitude 'g' of acceleration
 * gravity using measurements of the time that it takes for an object
 * in free fall to reach the ground from a given height and zero
 * initial velocity. The solution of the SIP is the posterior
 * probability density function (PDF) of 'g'.
 *
 * The SFP consists of calculating the maximum distance traveled by
 * an object in projectile motion. The posterior PDF of 'g' from the
 * SIP might be used as input to the SFP.
 *
 * The code consists of 7 files:
 * - 'gravity_main.C' (this file)
 * - 'gravity_compute.C' (the driving application code)
 * - 'gravity_compute.h'
 * - 'gravity_likelihood.C' (necessary for the SIP)
 * - 'gravity_likelihood.h'
 * - 'gravity_qoi.C' (necessary for the SFP)
 * - 'gravity_qoi.h'
 *-----------------------------------------------------------------*/

#include <gravity_compute.h>

int main(int argc, char* argv[])
{
  // Initialize QUESO environment
  MPI_Init(&argc,&argv);
  uqFullEnvironmentClass* env =
    new uqFullEnvironmentClass(MPI_COMM_WORLD,argv[1],"",NULL);

  // Call application
  computeGravityAndTraveledDistance(*env);

  // Finalize QUESO environment
```

```
  delete env;
  MPI_Finalize();

  return 0;
}
```

Listing 5.1: File gravity_main.C.

```
/*------------------------------------------------------------------
 * Brief description of this file:
 *
 * This is the header file for 'gravity_compute.C'.
 *-----------------------------------------------------------------*/

#ifndef __EX_COMPUTE_H__
#define __EX_COMPUTE_H__

#include <uqEnvironment.h>

void computeGravityAndTraveledDistance(const uqFullEnvironmentClass&
   env);

#endif
```

Listing 5.2: File gravity_compute.h.

```
/*------------------------------------------------------------------
 * Brief description of this file:
 *
 * This file is divided in two parts:
 * - the first one handles the statistical inverse problem (SIP)
 * for estimating the magnitude 'g' of gravity acceleration; and
 * - the second part handles the statistical forward problem (SFP)
 * for predicting the maximum distance traveled by a projectile.
 *
 * The SIP definition requires a user defined likelihood function.
 * See files 'gravity_likelihood.h' and 'gravity_likelihood.C'.
 *
 * The SFP definition requires a user defined qoi function.
 * See files 'gravity_qoi.h' and 'gravity_qoi.C'.
 *-----------------------------------------------------------------*/

#include <gravity_compute.h>
#include <gravity_likelihood.h>
#include <gravity_qoi.h>
```

```
#include <uqGslMatrix.h>
#include <uqStatisticalInverseProblem.h>
#include <uqStatisticalForwardProblem.h>
#include <sys/time.h>
#include <cmath>

//================================================================
// If PRIOR_IS_GAUSSIAN is defined, then:
// --> Gaussian prior for gravity.
// Otherwise:
// --> Uniform prior for gravity.
//================================================================

//#define PRIOR_IS_GAUSSIAN

void computeGravityAndTraveledDistance(const uqFullEnvironmentClass& env) {
  struct timeval timevalNow;

  gettimeofday(&timevalNow, NULL);
  if (env.fullRank() == 0) {
    std::cout << "\nBeginning run of 'Gravity + Projectile motion' example at "
              << ctime(&timevalNow.tv_sec)
              << "\n my fullRank = "         << env.fullRank()
              << "\n my subEnvironmentId = " << env.subId()
              << "\n my subRank = "          << env.subRank()
              << "\n my interRank = "        << env.inter0Rank()
               << std::endl << std::endl;
  }

  // Just examples of possible calls
  if ((env.subDisplayFile()       ) &&
      (env.displayVerbosity() >= 2)) {
    *env.subDisplayFile() << "Beginning run of 'Gravity + Projectile motion' example at "
                          << ctime(&timevalNow.tv_sec)
                          << std::endl;
  }
  env.fullComm().Barrier();
  env.subComm().Barrier();  // Just an example of a possible call

  //================================================================
  // Statistical inverse problem (SIP): find posterior PDF for 'g'
  //================================================================
```

```
  gettimeofday(&timevalNow, NULL);
  if (env.fullRank() == 0) {
    std::cout << "Beginning 'SIP -> Gravity estimation' at "
              << ctime(&timevalNow.tv_sec)
              << std::endl;
  }

  //------------------------------------------------------
  // SIP Step 1 of 6: Instantiate the parameter space
  //------------------------------------------------------
  uqVectorSpaceClass<uqGslVectorClass,uqGslMatrixClass> paramSpace(
     env, "param_", 1, NULL);

  //------------------------------------------------------
  // SIP Step 2 of 6: Instantiate the parameter domain
  //------------------------------------------------------
  uqGslVectorClass paramMinValues(paramSpace.zeroVector());
  uqGslVectorClass paramMaxValues(paramSpace.zeroVector());

  paramMinValues[0] = 8.;
  paramMaxValues[0] = 11.;

  uqBoxSubsetClass<uqGslVectorClass,uqGslMatrixClass>
    paramDomain("param_",
      paramSpace,
      paramMinValues,
      paramMaxValues);

  //------------------------------------------------------
  // SIP Step 3 of 6: Instantiate the likelihood function
  // object to be used by QUESO.
  //------------------------------------------------------
  likelihoodRoutine_DataClass likelihoodRoutine_Data(env);
  uqGenericScalarFunctionClass<uqGslVectorClass,uqGslMatrixClass>
    likelihoodFunctionObj("like_",
        paramDomain,
        likelihoodRoutine,
        (void *) &likelihoodRoutine_Data,
        true); // the routine computes [ln(function)]

  //------------------------------------------------------
  // SIP Step 4 of 6: Define the prior RV
  //------------------------------------------------------

#ifdef PRIOR_IS_GAUSSIAN
```

```
  uqGslVectorClass meanVector( paramSpace.zeroVector() );
  meanVector[0] = 9;

  uqGslMatrixClass covMatrix = uqGslMatrixClass(paramSpace.zeroVector());
  covMatrix(0,0) = 1.;

  // Create a Gaussian prior RV
  uqGaussianVectorRVClass<uqGslVectorClass,uqGslMatrixClass> priorRv("prior_",paramDomain,meanVector,covMatrix);

#else
  // Create an uniform prior RV
  uqUniformVectorRVClass<uqGslVectorClass,uqGslMatrixClass> priorRv("prior_",paramDomain);

#endif

  //------------------------------------------------------
  // SIP Step 5 of 6: Instantiate the inverse problem
  //------------------------------------------------------
  uqGenericVectorRVClass<uqGslVectorClass,uqGslMatrixClass>
    postRv("post_",  // Extra prefix before the default "rv_" prefix
           paramSpace);

  uqStatisticalInverseProblemClass<uqGslVectorClass,uqGslMatrixClass>
    ip("",           // No extra prefix before the default "ip_" prefix
       NULL,
       priorRv,
       likelihoodFunctionObj,
       postRv);

  //------------------------------------------------------
  // SIP Step 6 of 6: Solve the inverse problem, that is,
  // set the 'pdf' and the 'realizer' of the posterior RV
  //------------------------------------------------------
  std::cout << "Solving the SIP with Metropolis Hastings"
      << std::endl << std::endl;

  uqGslVectorClass paramInitials(paramSpace.zeroVector());
  priorRv.realizer().realization(paramInitials);

  uqGslMatrixClass proposalCovMatrix(paramSpace.zeroVector());
```

```
proposalCovMatrix(0,0) = std::pow( fabs(paramInitials[0])/20. , 2.
   );

ip.solveWithBayesMetropolisHastings(NULL, paramInitials, &
   proposalCovMatrix);

//================================================================
// Statistical forward problem (SFP): find the max distance
// traveled by an object in projectile motion; input pdf for 'g'
// is the solution of the SIP above.
//================================================================
gettimeofday(&timevalNow, NULL);
std::cout << "Beginning 'SFP -> Projectile motion' at "
          << ctime(&timevalNow.tv_sec)
          << std::endl;

//------------------------------------------------------
// SFP Step 1 of 6: Instantiate the parameter *and* qoi spaces.
// SFP input RV = FIP posterior RV, so SFP parameter space
// has been already defined.
//------------------------------------------------------
uqVectorSpaceClass<uqGslVectorClass,uqGslMatrixClass> qoiSpace(env,
    "qoi_", 1, NULL);

//------------------------------------------------------
// SFP Step 2 of 6: Instantiate the parameter domain
//------------------------------------------------------

// Not necessary because input RV of the SFP = output RV of SIP.
// Thus, the parameter domain has been already defined.

//------------------------------------------------------
// SFP Step 3 of 6: Instantiate the qoi function object
// to be used by QUESO.
//------------------------------------------------------
qoiRoutine_DataClass qoiRoutine_Data;
qoiRoutine_Data.m_angle          = M_PI/4.0; //45 degrees (radians)
qoiRoutine_Data.m_initialVelocity= 5.;      //initial speed (m/s)
qoiRoutine_Data.m_initialHeight  = 0.;       //initial height (m)

uqGenericVectorFunctionClass<uqGslVectorClass,uqGslMatrixClass,
   uqGslVectorClass,uqGslMatrixClass>
  qoiFunctionObj("qoi_",
                 paramDomain,
                 qoiSpace,
```

```
                        qoiRoutine,
                        (void *) &qoiRoutine_Data);

  //--------------------------------------------------------
  // SFP Step 4 of 6: Define the input RV
  //--------------------------------------------------------

  // Not necessary because input RV of SFP = output RV of SIP
  // (postRv).

  //--------------------------------------------------------
  // SFP Step 5 of 6: Instantiate the forward problem
  //--------------------------------------------------------
  uqGenericVectorRVClass<uqGslVectorClass,uqGslMatrixClass> qoiRv("qoi_", qoiSpace);

  uqStatisticalForwardProblemClass<uqGslVectorClass,uqGslMatrixClass,
     uqGslVectorClass,uqGslMatrixClass>
    fp("",
       NULL,
       postRv,
       qoiFunctionObj,
       qoiRv);

  //--------------------------------------------------------
  // SFP Step 6 of 6: Solve the forward problem
  //--------------------------------------------------------
  std::cout << "Solving the SFP with Monte Carlo"
            << std::endl << std::endl;
  fp.solveWithMonteCarlo(NULL);

  //--------------------------------------------------------
  gettimeofday(&timevalNow, NULL);
  if ((env.subDisplayFile()       ) &&
      (env.displayVerbosity() >= 2)) {
    *env.subDisplayFile() << "Ending run of 'Gravity + Projectile motion' example at "
                          << ctime(&timevalNow.tv_sec)
                          << std::endl;
  }
  if (env.fullRank() == 0) {
    std::cout << "Ending run of 'Gravity + Projectile motion' example at "
              << ctime(&timevalNow.tv_sec)
              << std::endl;
```

```
  }

230   return;
}
```

Listing 5.3: File `gravity_compute.C`. The first part of the code (lines 37–113) handles the statistical forward problem, whereas the second part of the code (lines 115–163) handles the statistical forward problem.

```
 /*------------------------------------------------------------------
 * Brief description of this file:
 *
 * This is the header file for gravity_likelihood.C.
 *-----------------------------------------------------------------*/

#ifndef __GRAVITY_LIKELIHOOD_H__
#define __GRAVITY_LIKELIHOOD_H__

#include <uqGslMatrix.h>

struct likelihoodRoutine_DataClass // user defined class
{
  likelihoodRoutine_DataClass(const uqBaseEnvironmentClass& env);
 ~likelihoodRoutine_DataClass();

  std::vector<double> m_heights; // heights
  std::vector<double> m_times;   // times
  std::vector<double> m_stdDevs; // account for uncertainties in
         // time measurement: sigmas

  const uqBaseEnvironmentClass* m_env;
};

double likelihoodRoutine( // user defined routine
  const uqGslVectorClass& paramValues,
  const uqGslVectorClass* paramDirection,
  const void*             functionDataPtr,
  uqGslVectorClass*       gradVector,
  uqGslMatrixClass*       hessianMatrix,
  uqGslVectorClass*       hessianEffect);

#endif
```

Listing 5.4: File `gravity_likelihood.h`.

```
/*------------------------------------------------------------------
 * Brief description of this file:
 *
 * This file contains the code for the user defined likelihood data
 * class and the user defined likelihood routine.
 *------------------------------------------------------------------*/

#include <gravity_likelihood.h>
#include <cmath>
#include <stdio.h>
#include <fstream>

// Construtor
likelihoodRoutine_DataClass::likelihoodRoutine_DataClass(const
   uqBaseEnvironmentClass& env)
  :
  m_heights(0),
  m_times  (0),
  m_stdDevs(0),
  m_env    (&env)
{
  // Data available in /inputData/data02.dat
  double const heights[] =
     {10,20,30,40,50,60,70,80,90,100,110,120,130,140};
  double const times  [] =
     {1.41,2.14,2.49,2.87,3.22,3.49,3.81,4.07,4.32,4.47,
      4.75,4.99,5.16,5.26};
  double const stdDevs[] =
     {0.020,0.120,0.020,0.010,0.030,0.010,0.030,0.030,
      0.030,0.050,0.010,0.040,0.010,0.09};

  std::size_t const n = sizeof(heights)/sizeof(*heights);
  m_heights.assign(heights, heights + n);
  m_times.assign  (times,   times   + n);
  m_stdDevs.assign(stdDevs, stdDevs + n);
}

// Destructor
likelihoodRoutine_DataClass::~likelihoodRoutine_DataClass()
{
}

//------------------------------------------------------
// The user defined likelihood routine
//------------------------------------------------------
```

```
double likelihoodRoutine(
  const uqGslVectorClass& paramValues,
  const uqGslVectorClass* paramDirection,
  const void*             functionDataPtr,
  uqGslVectorClass*       gradVector,
  uqGslMatrixClass*       hessianMatrix,
  uqGslVectorClass*       hessianEffect)
{
  const uqBaseEnvironmentClass& env = *(((likelihoodRoutine_DataClass
     *) functionDataPtr)->m_env);

  if (paramDirection && functionDataPtr && gradVector &&
     hessianMatrix && hessianEffect)
  {
    // Just to eliminate INTEL compiler warnings
  }

  env.subComm().Barrier();

  // The user, at the application level, should have set
  // the vector 'paramValues' to have size 1.
  UQ_FATAL_TEST_MACRO(paramValues.sizeGlobal() != 1,
                      env.fullRank(),
                      "likelihoodRoutine()",
                      "paramValues vector does not have size 1");

  // Compute likelihood
  double g = paramValues[0];
  const std::vector<double>& heights=((likelihoodRoutine_DataClass*)
     functionDataPtr)->m_heights;
  const std::vector<double>& times  =((likelihoodRoutine_DataClass*)
     functionDataPtr)->m_times;
  const std::vector<double>& stdDevs=((likelihoodRoutine_DataClass*)
     functionDataPtr)->m_stdDevs;

  double misfitValue = 0.;
  for (unsigned int i = 0; i < heights.size(); ++i) {
    double modelTime = sqrt(2.0 * heights[i]/g);
    double ratio = (modelTime - times[i])/stdDevs[i];
    misfitValue += ratio*ratio;
  }
  return (-0.5*misfitValue);
}
```

Listing 5.5: File gravity_likelihood.C.

```
/*------------------------------------------------------------------
 * Brief description of this file:
 *
 * This is the header file from gravity_qoi.C.
 *-----------------------------------------------------------------*/

#ifndef __GRAVITY_QOI_H__
#define __GRAVITY_QOI_H__

#include <uqGslMatrix.h>
#include <uqDistArray.h>

struct
qoiRoutine_DataClass
{
  double m_angle;
  double m_initialVelocity;
  double m_initialHeight;
};

void
qoiRoutine(
  const uqGslVectorClass&                     paramValues,
  const uqGslVectorClass*                     paramDirection,
  const void*                                 functionDataPtr,
        uqGslVectorClass&                     qoiValues,
        uqDistArrayClass<uqGslVectorClass* >* gradVectors,
        uqDistArrayClass<uqGslMatrixClass* >* hessianMatrices,
        uqDistArrayClass<uqGslVectorClass* >* hessianEffects);

#endif
```

Listing 5.6: File gravity_qoi.h.

```
/*------------------------------------------------------------------
 * Brief description of this file:
 *
 * This file contains the code for the user defined qoi routine.
 *-----------------------------------------------------------------*/

#include <gravity_qoi.h>
#include <cmath>

//-----------------------------------------------------
```

```
/// The actual (user-defined) qoi routine
//------------------------------------------------------
void
qoiRoutine(
  const uqGslVectorClass&                          paramValues,
  const uqGslVectorClass*                          paramDirection,
  const void*                                      functionDataPtr,
        uqGslVectorClass&                          qoiValues,
        uqDistArrayClass<uqGslVectorClass*>* gradVectors,
        uqDistArrayClass<uqGslMatrixClass*>* hessianMatrices,
        uqDistArrayClass<uqGslVectorClass*>* hessianEffects)
{
  const uqBaseEnvironmentClass& env = paramValues.env();

  if (paramDirection &&
      gradVectors    &&
      hessianEffects &&
      hessianMatrices) {
    // Logic just to avoid warnings from INTEL compiler
  }

  // The user, at the application level, should have set
  // the vector 'paramValues' to have size 1 and
  // the vector 'qoiValues' to have size 1.
  UQ_FATAL_TEST_MACRO(paramValues.sizeGlobal() != 1,
                      env.fullRank(),
                      "qoiRoutine()",
                      "paramValues vector does not have size 1");

  UQ_FATAL_TEST_MACRO(qoiValues.sizeGlobal() != 1,
                      env.fullRank(),
                      "qoiRoutine()",
                      "qoiValues vector does not have size 1");

  // Compute qoi(s)
  double g = paramValues[0]; // Sample of the RV 'gravity
     acceleration'
  double distanceTraveled = 0.;
  if (env.subRank() == 0) {
    double velocity = ((qoiRoutine_DataClass *) functionDataPtr)->
       m_initialVelocity;
    double heights  = ((qoiRoutine_DataClass *) functionDataPtr)->
       m_initialHeight;
    double alpha    = ((qoiRoutine_DataClass *) functionDataPtr)->
       m_angle;
```

```
    double aux        = velocity * sin(alpha);
    distanceTraveled = (velocity * cos(alpha) / g) * ( aux + sqrt(pow
       (aux,2) + 2.*g*heights) );
  }

  qoiValues[0] = distanceTraveled;

  return;
}
```

Listing 5.7: File `gravity_qoi.C`.

5.4 Application Input File

QUESO reads an input file for solving statistical problems. In the case of a SIP, it expects a list of options for MCMC, while in case of SFP it expects a list of options for Monte Carlo.

The most relevant options of an input file for the solution of a SIP using QUESO are displayed in Listing 5.8. Note that the names of the variables have been designed to be informative:

env: refers to QUESO environment;

ip: refers to inverse problem;

mh: refers to Metropolis-Hastings;

dr: refers to delayed rejection;

am: refers to adaptive Metropolis;

rawChain: refers to the raw, entire chain;

filteredChain: refers to a filtered chain (related to a specified `lag`);

fp: refers to forward problem;

mc: refers to Monte Carlo;

pseq: refers to the parameter sequence; and

qseq: refers to the quantity of interest sequence.

```
###############################################
# UQ Environment
###############################################
env_numSubEnvironments   = 1 #or 8, if 8 processors can be used in a
   parallel run;  see Section 5.5.1
env_subDisplayFileName   = outputData/display_env
env_subDisplayAllowAll   = 0
env_subDisplayAllowedSet = 0 1 2 3 4 5 6 7
env_displayVerbosity     = 2
```

```
env_seed                  = -1

###############################################
# Statistical inverse problem (ip)
###############################################
ip_computeSolution       = 1
ip_dataOutputFileName    = outputData/sip_gravity
ip_dataOutputAllowedSet = 0 1

###############################################
# Information for Metropolis-Hastings algorithm
###############################################
ip_mh_dataOutputFileName    = outputData/sip_gravity
ip_mh_dataOutputAllowedSet = 0 1

ip_mh_rawChain_dataInputFileName    = .
ip_mh_rawChain_size                 = 20000
ip_mh_rawChain_generateExtra        = 0
ip_mh_rawChain_displayPeriod        = 2000
ip_mh_rawChain_measureRunTimes      = 1
ip_mh_rawChain_dataOutputFileName   = outputData/
   sip_gravity_raw_chain
ip_mh_rawChain_dataOutputAllowedSet = 0 1 2 3 4 5 6 7

ip_mh_displayCandidates             = 0
ip_mh_putOutOfBoundsInChain         = 0
ip_mh_dr_maxNumExtraStages          = 3
ip_mh_dr_listOfScalesForExtraStages = 5. 10. 20.
ip_mh_am_initialNonAdaptInterval    = 0
ip_mh_am_adaptInterval              = 100
ip_mh_am_eta                        = 1.98    #(2.4^2)/d, d is the
   dimension of the problem
ip_mh_am_epsilon                    = 1.e-5

ip_mh_filteredChain_generate             = 1
ip_mh_filteredChain_discardedPortion     = 0.
ip_mh_filteredChain_lag                  = 20
ip_mh_filteredChain_dataOutputFileName   = outputData/
   sip_gravity_filtered_chain
ip_mh_filteredChain_dataOutputAllowedSet = 0 1

###############################################
# Statistical forward problem (fp)
###############################################
fp_help                  = anything
```

```
fp_computeSolution        = 1
fp_computeCovariances     = 1
fp_computeCorrelations    = 1
fp_dataOutputFileName     = outputData/sfp_gravity
fp_dataOutputAllowedSet = 0 1

################################################
# 'fp_': information for Monte Carlo algorithm
################################################
fp_mc_help                   = anything
fp_mc_dataOutputFileName     = outputData/sfp_gravity
fp_mc_dataOutputAllowedSet = 0 1

fp_mc_pseq_dataOutputFileName    = outputData/sfp_gravity_p_seq
fp_mc_pseq_dataOutputAllowedSet = 0 1

fp_mc_qseq_dataInputFileName     = .
fp_mc_qseq_size                  = 16384
fp_mc_qseq_displayPeriod         = 20000
fp_mc_qseq_measureRunTimes       = 1
fp_mc_qseq_dataOutputFileName    = outputData/sfp_gravity_qoi_seq
fp_mc_qseq_dataOutputAllowedSet = 0 1
```

Listing 5.8: Some options for QUESO library used in application code (Listings 5.1-5.5).

Moreover, for the gravity inverse problem, one may notice that QUESO will use the Metropolis-Hastings algorithm to sample the posterior PDF (indicated by the prefix `mh_`in the variable names) without adaptive steps (indicated by the zero value assigned to the variable `ip_mh_am_initialNonAdaptInterval`, which can also be achieved by setting zero to `ip_mh_am_adaptInterval`) and with delayed rejection (indicated by the one-value assigned to the variable `ip_mh_dr_maxNumExtraStages`).

5.5 Application Compilation and Run

Makefiles are special format files that together with the make utility will help one to compile and automatically build and manage projects (programs). Listing 5.9 presents a Makefile, named `Makefile_example_violeta`, that may be used to compile the code and create the executable `gravity_gsl`. Naturally, it must be adapted to the user's settings, i.e., it has to have the correct paths for the user's libraries that were actually used to compile and install QUESO (see Sections 2.1–2.4).

```
############################################################
# Using installed QUESO 0.46.0, from tarball, in 'violeta'
############################################################
```

```
QUESO_DIR = /home/kemelli/LIBRARIES/QUESO-0.46.0
BOOST_DIR = /home/kemelli/LIBRARIES/boost-1.53.0
GSL_DIR  =  /home/kemelli/LIBRARIES/gsl-1.15
HDF5_DIR  = /home/kemelli/LIBRARIES/hdf5-1.8.10

INC_PATHS = \
  -I. \
  -I$(QUESO_DIR)/include \
  -I$(BOOST_DIR)/include \
  -I$(GSL_DIR)/include \
  -I$(HDF5_DIR)/include

LIBS = \
  -L$(QUESO_DIR)/lib \
  -lqueso \
  -L$(BOOST_DIR)/lib \
  -lboost_program_options \
  -L$(GSL_DIR)/lib \
  -lgsl \
  -L$(HDF5_DIR)/lib \
  -lhdf5

CXX = mpic++
CXXFLAGS += -g -Wall -c

default: all

.SUFFIXES: .o .C

all:  example_gravity_gsl

clean:
  rm -f *~
  rm -f *.o
  rm -f gravity_gsl

example_gravity_gsl: gravity_main.o gravity_likelihood.o
   gravity_compute.o gravity_qoi.o
  $(CXX) gravity_main.o \
         gravity_likelihood.o \
         gravity_compute.o \
         gravity_qoi.o \
         -o gravity_gsl $(LIBS)
```

```
%.o: %.C
  $(CXX) $(INC_PATHS) $(CXXFLAGS) $<
```

Listing 5.9: Makefile for the application code in Listings 5.1-5.5

Suppose the user either does not needto modify the Makefile presented in Listings 5.9 or has already done so. Then, in order to compile, build and run the application code together with its input file `gravity_inv_fwd.inp`, presented in Listing 5.8, the user simply needs to enter following commands, under QUESO build tree:

```
cd $HOME/queso_download/queso-0.46.0/examples/gravity/src
make -f Makefile_example_violeta
cd ../tests/test_2013_01_22
../../src/gravity_gsl gravity_inv_fwd.inp
```

Alternatively, if the user is not interested in re-compiling and and building the code, he/she can simply run the executable created during QUESO installation, and, thus, located at QUESO installation tree:

```
cd $HOME/LIBRARIES/QUESO-0.46.0/examples/gravity
./gravity_gsl gravity_inv_fwd.inp
```

In either case, the console output of the program is:

```
kemelli@violeta:~/LIBRARIES/QUESO-0.46.0/examples/gravity$ ./
   gravity_gsl gravity_inv_fwd.inp
--------------------------------------------------------------------
QUESO Library: Version = 0.46.0 (4600)

Development Build

Build Date   = 2013-04-29 17:05
Build Host   = violeta
Build User   = kemelli
Build Arch   = x86_64-unknown-linux-gnu
Build Rev    = 38998M

C++ Config   = mpic++ -g -O2 -Wall

Trilinos DIR =
GSL Libs     = -L/home/kemelli/LIBRARIES/gsl-1.15/lib -lgsl -
   lgslcblas -lm
GRVY DIR     =
GLPK DIR     =
HDF5 DIR     = /home/kemelli/LIBRARIES/hdf5-1.8.10
```

```
-----------------------------------------------------------------------------

Beginning run at Mon Apr 29 17:27:32 2013

MPI node of worldRank 0 has fullRank 0, belongs to subEnvironment of
   id 0, and has subRank 0
MPI node of worldRank 0 belongs to sub communicator with full ranks 0
MPI node of worldRank 0 also belongs to inter0 communicator with full
    ranks 0, and has inter0Rank 0


Beginning run of 'Gravity + Projectile motion' example at Mon Apr 29
   17:27:32 2013

 my fullRank = 0
 my subEnvironmentId = 0
 my subRank = 0
 my interRank = 0

Beginning 'SIP -> Gravity estimation' at Mon Apr 29 17:27:32 2013

Solving the SIP with Metropolis Hastings

Beginning 'SFP -> Projectile motion' at Mon Apr 29 17:27:33 2013

Solving the SFP with Monte Carlo

Ending run of 'Gravity + Projectile motion' example at Mon Apr 29
   17:27:33 2013

Ending run at Mon Apr 29 17:27:33 2013
Total run time = 1 seconds
kemelli@violeta:~/LIBRARIES/QUESO-0.46.0/examples/gravity$
```

Listing 5.10: Console output of program `gravity_gsl`

5.5.1 Application Run with Several Processors

Even though the application described in Section 5.3 is a serial code, it is possible to run it using more than one processor, i.e., in parallel mode. Supposing the user's workstation has $N_p = 8$ processors, then, the user my choose to have $N_s = 8$, 4 or 2 subenvironments. This complies with the requirement that the total number of processors in the environment must be a multiple of the specified number of subenvironments.

Thus, to build and run the application code with $N_p = 8$, and $N_s = 8$ subenvironments, the must set the variable `env_numSubEnvironments = 8` in the input file (Listing 5.8) and enter the following command lines:

```
cd $HOME/LIBRARIES/QUESO-0.46.0/examples/gravity/
mpirun -np 8 ./gravity_gsl gravity_inv_fwd.inp
```

The steps above will create a total number of 8 raw chains, of size defined by the variable `ip_mh_rawChain_size`. QUESO internally combines these 8 chains into a single chain of size 8 $\times$ `ip_mh_rawChain_size` and saves it in a file named according to the variable `ip_mh_rawChain_dataOutputFileName`. QUESO also provides the user with the option of writing each chain – handled by its corresponding processor – in a separate file, which is accomplished by setting the variable `ip_mh_rawChain_dataOutputAllowedSet = 0 1 ... Ns-1`.

Note: Although the discussion in the previous paragraph refers to the raw chain of a SIP, the analogous is true for the filtered chains (SIP), and for the samples employed in the SFP (`ip_mh_filteredChain_size`, `fp_mc_qseq_size` and `fp_mc_qseq_size`, respectively).

5.6 Application Results

There are a few Matlab-ready commands that are very helpful tools for post-processing the data generated by QUESO when solving statistical inverse problems. This section discusses the results computed by QUESO with the code of Section 5.3, and shows how to use Matlab for the post-processing of such results.

According to the specifications of the input file in Listing 5.8, both a folder named `outputData` and a the following files should be generated:

```
display_env_sub0.txt
sfp_gravity_p_seq.m
sfp_gravity_p_seq_sub0.m
sfp_gravity_qoi_seq.m
sfp_gravity_qoi_seq_sub0.m
sfp_gravity_sub0.m
sip_gravity_filtered_chain.m
sip_gravity_filtered_chain_sub0.m
sip_gravity_raw_chain.m
sip_gravity_raw_chain_sub0.m
sip_gravity_sub0.m
```

In this section, a convenient capability of QUESO of internally handling possible conflicts in chain size is presented. Recalling the input file `gravity_inv_fwd.inp` presented in Listing 5.8, one may notice that the raw chain size for the SIP is chosen to have 20000 positions (`ip_mh_rawChain_size = 20000`); the lag of the filtered chain is chosen to be 20 (`ip_mh_filteredChain_lag = 20`) and the chain size for the SFP has 16384 positions (`fp_mc_qseq_size = 16384`). Because the solution of the SIP, ie, the posterior PDF, is used as input PDF for the SFP, QUESO internally sets `fp_mc_qseq_size = 20000`, as can be seen in the file `display_env_sub0.txt`. The file `display_env_sub0.txt` contains information from the subenvironment '0' that was generated during the run of the application code.

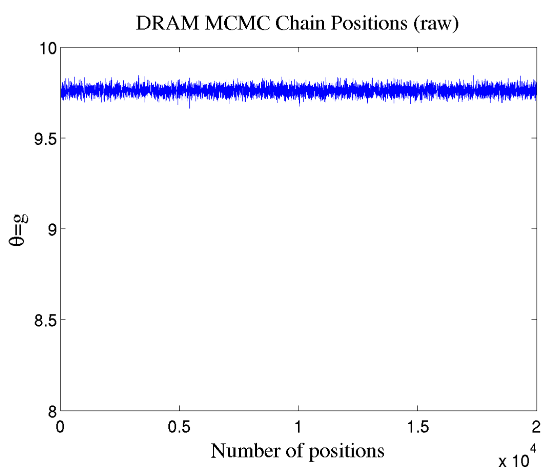
5.6.1 Statistical Inverse Problem

5.6.1.1 Chain Plots

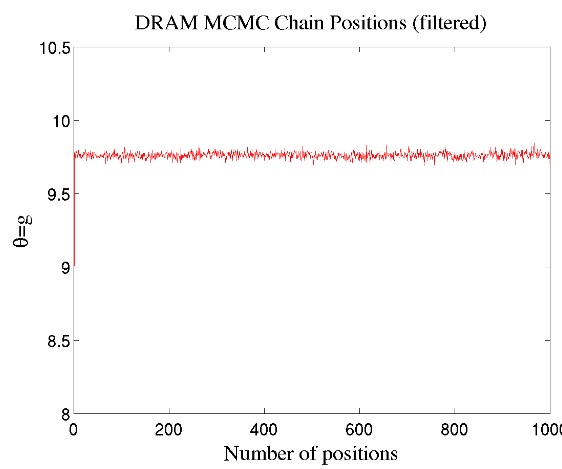
It is quite simple to plot, using Matlab, the chain of positions used in the DRAM algorithm implemented within QUESO. The sequence of Matlab commands presented in Listing 5.11 generates the graphic depicted in Figure 5.1(a). Figure 5.1(b) is obtained analogously.

```
% inside Matlab
>> sip_gravity_raw_chain
>> plot(ip_mh_rawChain_unified)
>> ylabel('\theta=g','fontsize',20);
>> xlabel('Number of positions','fontsize',20);
>> title('DRAM Chain Positions (raw)','fontsize',20);
```

Listing 5.11: Matlab code for the chain plot.



(a) Raw chain



(b) Filtered chain

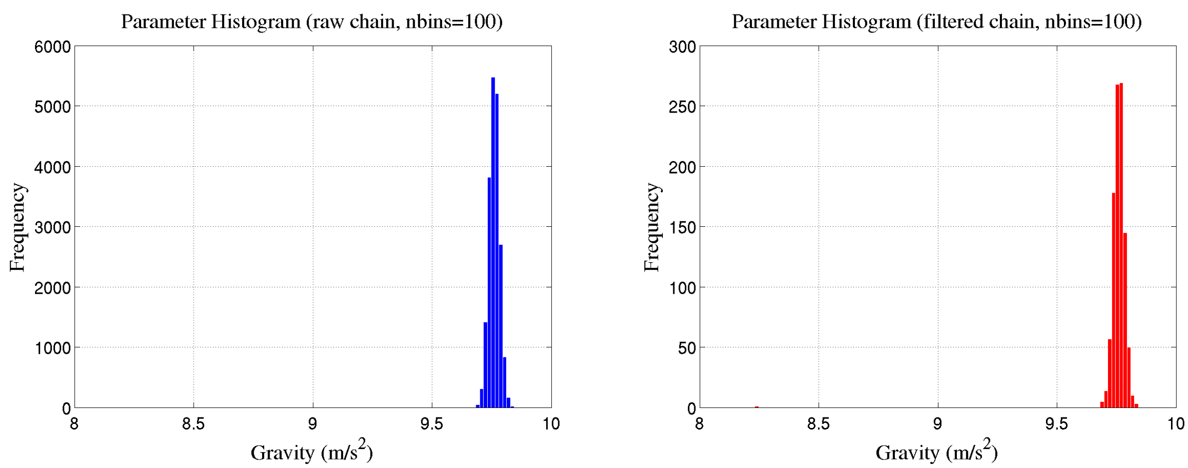
Figure 5.6.1: MCMC raw chain with 20000 positions and a filtered chain with lag of 20 positions.

5.6.1.2 Histogram Plots

In order to plot histograms of the parameter using either the raw chain or the filtered chain, you simply have to use the pre-defined Matlab function `hist`.

```
% inside Matlab
>> sip_gravity_raw_chain
>> nbins=100;
>> hist(ip_mh_rawChain_unified,nbins)
>> title('Parameter Histogram (raw chain)','fontsize',20);
>> xlabel('Gravity (m/s^2)','fontsize',20);
>> ylabel('Frequency','fontsize',20);
>> grid on;
```

Listing 5.12: Matlab code for the histogram plot.



(a) Raw chain (b) Filtered chain

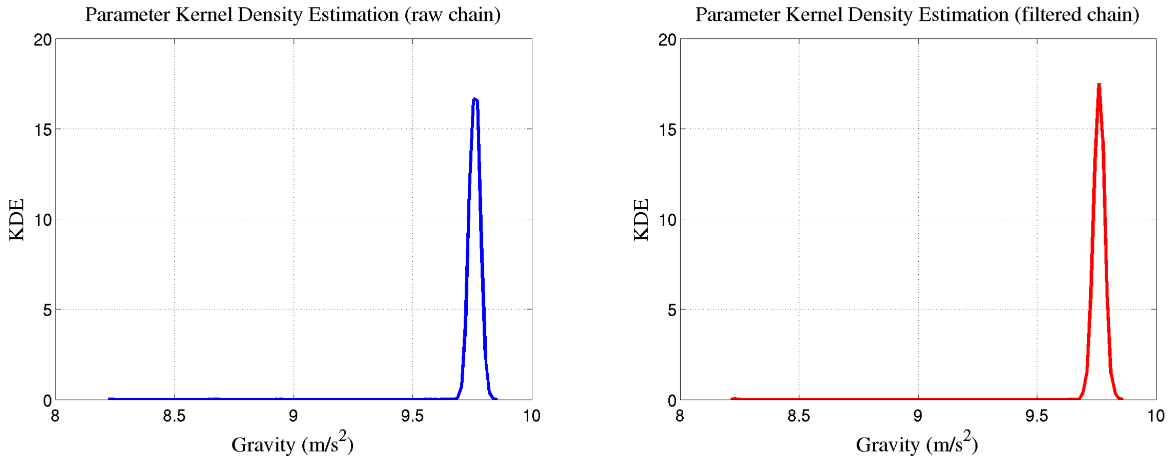
Figure 5.6.2: Histograms of parameter $\theta = g$.

5.6.1.3 KDE Plots

Matlab function `ksdensity` (Kernel smoothing density estimate) together with the option `'pdf'` may be used for plotting the KDE of the parameter.

```
% inside Matlab
>> sip_gravity_raw_chain
>> [f,xi] = ksdensity(ip_mh_rawChain_unified,'function','pdf');
>> plot(xi,f,'-b','linewidth',3)
>> title('Parameter Kernel Density Estimation','fontsize',20);
>> xlabel('Gravity (m/s^2)','fontsize',20);
>> ylabel('KDE','fontsize',20);
>> grid on;
```

Listing 5.13: Matlab code for the KDE plot.



(a) Raw chain

(b) Filtered chain

Figure 5.6.3: Kernel Density Estimation.

5.6.1.4 CDF Plots

Matlab function **ksdensity** (Kernel smoothing density estimate) with **'cdf'** option may also be used for plotting the Cumulative Distribution Function of the parameter.

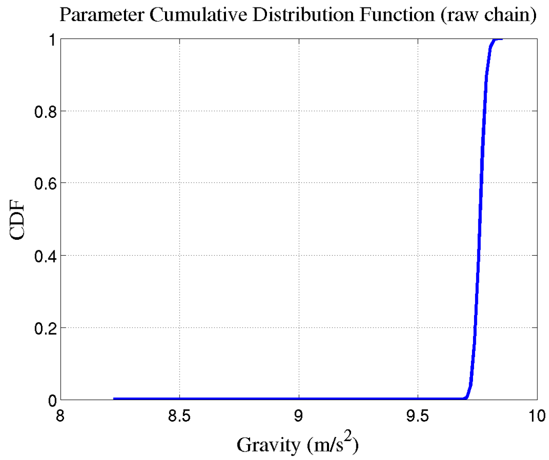
```
% inside Matlab
>> sip_gravity_raw_chain
>> [f,xi] = ksdensity(ip_mh_rawChain_unified,'function','cdf');
>> plot(xi,f,'-b','linewidth',3)
>> title('Parameter Cumulative Distribution Function','fontsize',20);
>> xlabel('Gravity (m/s^2)','fontsize',20);
>> ylabel('CDF','fontsize',20);
>> grid on;
```

Listing 5.14: Matlab code for the CDF plot.

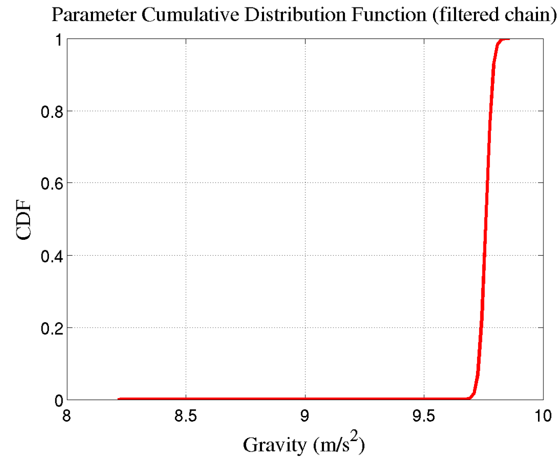
5.6.1.5 Autocorrelation Plots

The code presented in Listing 5.15 uses matlab function **autocorr** to generate Figure 5.6.5 which presents the autocorrelation of the parameter g in both cases: raw and filtered chain.

```
% inside Matlab
>> sip_gravity_raw_chain
>> sip_gravity_filtered_chain
>> nlags=10;
>> [ACF_raw,lags,bounds]= autocorr(ip_mh_rawChain_unified, nlags, 0);
>> [ACF_filt,lags,bounds]=autocorr(ip_mh_filtChain_unified,nlags, 0);
>> plot(lags,ACF_raw,'bo-',lags,ACF_filt,'r*-','linewidth',3);
```



(a) Raw chain



(b) Filtered chain

Figure 5.6.4: Cumulative Distribution Function.

```
>> ylabel('Autocorrelation for \theta=g','fontsize',20);
>> xlabel('Lag','fontsize',20);
>> title('Parameter Autocorrelation','fontsize',20);
>> grid on;
>> h=legend('raw chain','filtered chain','location','northeast');
>> set(h,'fontsize',16);
```

Listing 5.15: Matlab code for the autocorrelation plots.

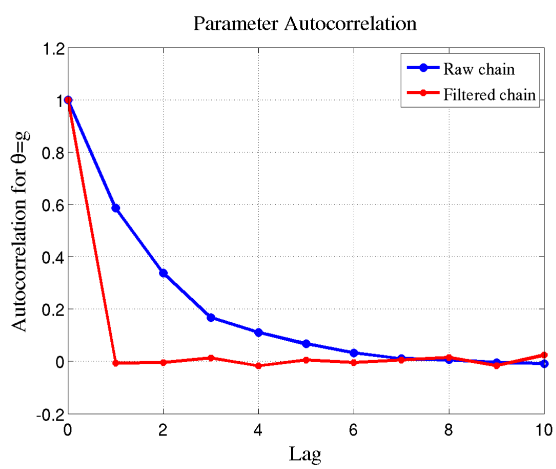


Figure 5.6.5: Autocorrelation plots.

5.6.1.6 Covariance and Correlation Matrices

Matlab function `cov` calculates the covariance matrix for a data matrix (where each column represents a separate quantity), and `corr` calculates the correlation matrix. Since our statistical inverse problem has only one parameter (the acceleration g due to gravity), both covariance and correlation matrices have dimension 1×1, i.e., they are scalars.

```
% inside Matlab
>> sip_gravity_raw_chain;
>> cov_matrix_g = cov(ip_mh_rawChain_unified)

cov_matrix_g =

   6.8709e-04
>> corr_matrix_g = corr(ip_mh_rawChain_unified)

corr_matrix_g =

     1
>>
```

Listing 5.16: Matlab code for finding the covariance matrix.

5.6.2 Statistical Forward Problem

5.6.2.1 Chain Plots

It is quite simple to plot, using Matlab, the chain of positions generated by the Monte Carlo algorithm implemented within QUESO and called during the solution of the statistical forward problem.

```
% inside Matlab
>> sfp_gravity_qoi_seq.m
>> plot(fp_mc_QoiSeq_unified);
>> ylabel('QoI','fontsize',20);
>> xlabel('Number of positions','fontsize',20);
>> title('MC Chain Positions','fontsize',20);
```

Listing 5.17: Matlab code for the chain plot.

5.6.2.2 Histogram Plots

In order to plot a histogram of the QoI, you may use the pre-defined Matlab function `hist`.

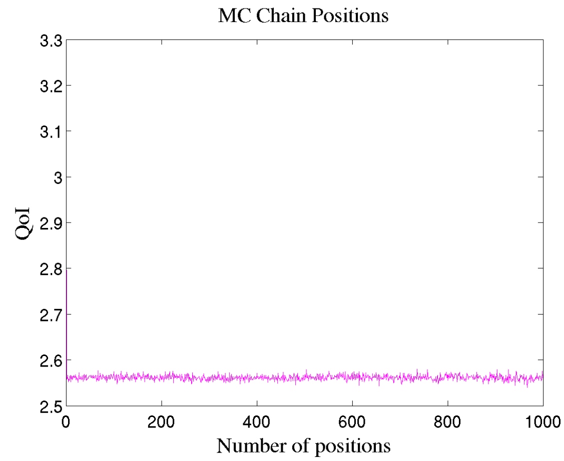


Figure 5.6.6: MC chain positions for the QoI.

```
>> sfp_gravity_qoi_seq.m
>> nbins=100;
>> hist(fp_mc_QoiSeq_unified);
>> title('QoI Histogram','fontsize',20);
>> xlabel('Distance traveled (m)','fontsize',20);
>> ylabel('Frequency','fontsize',20);
>> grid on;
```

Listing 5.18: Matlab code for the QoI histogram plot.

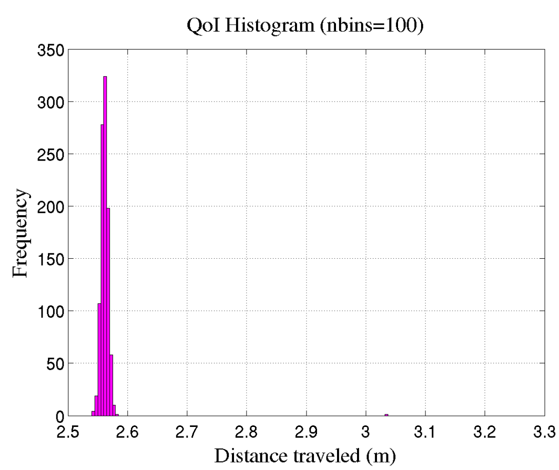


Figure 5.6.7: Histogram of QoI $= d$.

5.6.2.3 KDE Plots

Matlab function `ksdensity` (Kernel smoothing density estimate) together with the option `'pdf'` may be used for plotting the KDE of the he QoI.

```
% inside Matlab
>> sfp_gravity_qoi_seq.m
>> [f,xi] = ksdensity(fp_mc_QoiSeq_unified,'function','pdf');
>> plot(xi,f,'-b','linewidth',3)
>> title('QoI Kernel Density Estimation ','fontsize',20);
>> xlabel('Distance traveled (m)','fontsize',20);
>> ylabel('KDE','fontsize',20);
>> grid on;
```

Listing 5.19: Matlab code for the QoI KDE plot.

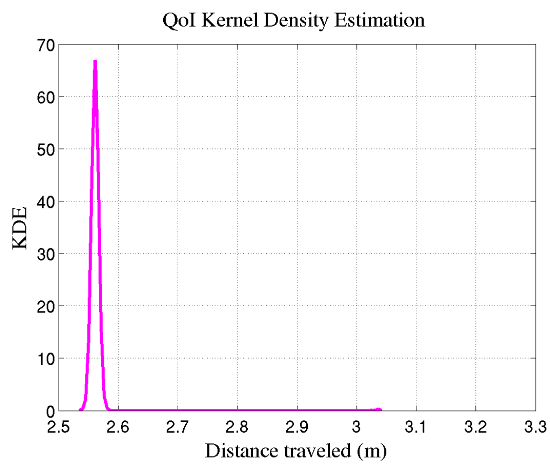


Figure 5.6.8: Kernel Density Estimation.

5.6.2.4 CDF Plots

Matlab function `ksdensity` (Kernel smoothing density estimate) with `'cdf'` option may also be used for plotting the Cumulative Distribution Function of the QoI.

```
% inside Matlab
>> sfp_gravity_qoi_seq.m
>> [f,xi] = ksdensity(fp_mc_QoiSeq_unified,'function','cdf');
>> plot(xi,f,'-b','linewidth',3)
>> title('QoI Cumulative Distribution Function ','fontsize',20);
>> xlabel('Distance traveled (m)','fontsize',20);
>> ylabel('CDF','fontsize',20);
>> grid on;
```

Listing 5.20: Matlab code for the QoI CDF plot.

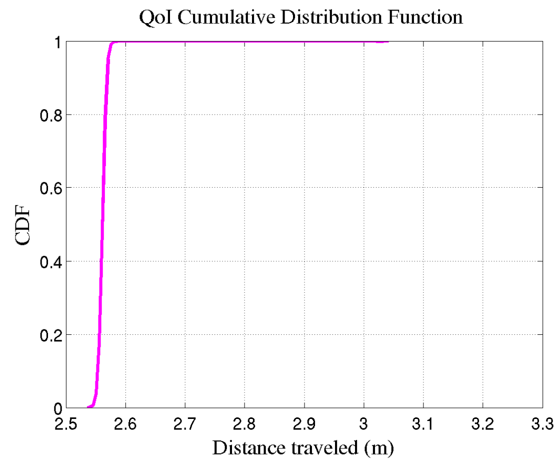


Figure 5.6.9: Cumulative Distribution Function.

5.6.2.5 Autocorrelation Plots

The code presented in Listing 5.21 uses Matlab function `autocorr` to generate Figure 5.6.10, which presents the autocorrelation of the QoI d.

```
% inside Matlab
>> sfp_gravity_qoi_seq.m
>> nlags=10;
>> [ACF, lags, bounds] = autocorr(fp_mc_QoiSeq_unified, nlags, 0);
>> plot(lags,ACF,'bo-','linewidth',3);
>> ylabel('Autocorrelation for QoI = d','fontsize',20);
>> xlabel('Lag','fontsize',20);
>> title('QoI Autocorrelation','fontsize',20);
>> grid on;
```

Listing 5.21: Matlab code for the QoI autocorrelation plot.

5.6.2.6 Covariance and Correlation Matrices

For a matrix input `X`, where each row is an observation, and each column is a variable, the Matlab function `cov(X)` may be used to calculate the covariance matrix.

Thus, in order to calculated the covariance matrix between the parameter and the quantity of interest sequences generated by Monte Carlo sampler with QUESO, one may simply define `X=[fp_mc_ParamSeq_unified fp_mc_QoiSeq_unified]`. The code presented in Listing 5.22 shows the usage of Matlab commands for finding such the matrix.

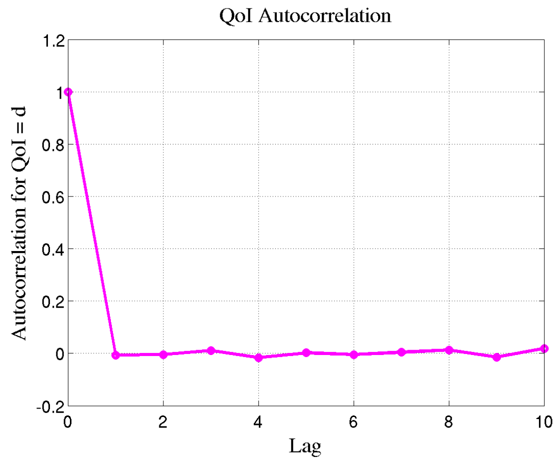


Figure 5.6.10: Autocorrelation plot.

```
% inside Matlab
>> sfp_gravity_qoi_seq;
>> sfp_gravity_p_seq;
>> X=[fp_mc_ParamSeq_unified fp_mc_QoiSeq_unified];
>> cov_p_QoI = cov(X)

cov_p_QoI =
    [ 2.826e-03   -8.555e-04 ]
    [-8.555e-04    2.599e-04 ]
```

Listing 5.22: Matlab code for the matrix of covariance between parameter g and QoI d.

Analogously, the Matlab function `corrcoef(X)` returns a matrix of correlation coefficients calculated from an input matrix `X` whose rows are observations and whose columns are variables. In order to calculated the correlation matrix between the parameter and the QoI sequences, one may simply define `X=[fp_mc_ParamSeq_unified fp_mc_QoiSeq_unified]`.

```
% inside Matlab
>> sfp_gravity_qoi_seq;
>> sfp_gravity_p_seq;
>> X=[fp_mc_ParamSeq_unified fp_mc_QoiSeq_unified];
>> corr_p_QoI = corrcoef(X)

corr_p_QoI =
    [ 1.000e+00   -9.981e-01 ]
    [-9.981e-01    1.000e+00 ]
>>
```

Listing 5.23: Matlab code for the matrix of correlation between parameter g and quantity of interest d.

Bibliography

[1] Diagrams for Interactive Learning. http://interactagram.com/. Accessed on August 12th, 2012.

[2] TACC: Texas Advanced Computing Center. http://www.tacc.utexas.edu/, 2001-2013.

[3] D. Calvetti and E. Somersalo. *Introduction to Bayesian Scientific Computing*, volume 2 of *Surveys and Tutorials in the Applied Mathematical Sciences*. Springer, 2007.

[4] B. P. Carlin and T. A. Louis. *Bayesian Methods for Data Analysis*. Texts in Statistical Science. CRS Press, 3rd edition, 2006.

[5] Committee on Mathematical Foundations of Verification, Validation, and Uncertainty Quantification; Board on Mathematical Sciences and Their Applications, Division on Engineering and Physical Sciences, National Research Council. *Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification*. The National Academies Press, 2012. http://www.nap.edu/openbook.php?record_id=13395.

[6] A. DasGupta. *Asymptotic Theory of Statistics and Probability*. Springer Texts in Statistics. Springer, 2008.

[7] R. Durret. *Probability: Theory and Examples*. Duxbury Advanced Series. Thomson Brooks/Cole, 3rd edition, 2005.

[8] Mark Galassi, James Theiler, Brian Gough, Gerard Jungman, and many others. GNU Scientific Library. http://www.gnu.org/software/gsl/, 1996-2011.

[9] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Texts in Statistical Science. Chapman & Hall/CRC, 2nd edition, 2004.

[10] P. J. Green and A. Mira. Delayed rejection in reversible jump Metropolis-Hastings. *BIOMETRIKA*, 88(4):1035–1053, 2001.

[11] The HDF Group. HDF5 (Hierarchical Data Format 5). `http://www.hdfgroup.org/HDF5/`, 2000–2012.

[12] The MathWorks Group. MATLAB®. `http://www.mathworks.com/products/matlab/`, 2013.

[13] H. Haario, M. Laine, A. Mira, and E. Saksman. DRAM: Efficient adaptive MCMC. *Stat. Comput.*, 16:339–354, 2006.

[14] H. Haario, E. Saksman, and J. Tamminen. An adaptive Metropolis algorithm. *Bernoulli*, 7(2):223–242, 2001.

[15] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *BIOMETRIKA*, 57(1):97–109, 1970.

[16] Michael Heroux. Trilinos. `http://trilinos.sandia.gov/`, 2013.

[17] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[18] X. Huan and Y. Marzouk. Optimal Bayesian experimental design for combustion kinetics. In *49th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number AIAA 2011-513, Orlando, Florida, 2011.

[19] Silicon Graphics International. STL: Standard Template Library. `http://www.sgi.com/tech/stl/`, 2000-2011.

[20] J. Jacod and P. Protter. *Probability Essentials*. Springer, 2nd edition, 2004.

[21] E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[22] N. M. Josuttis. *The C++ Standard Library, A Tutorial and Reference*. Addison Wesley, 1st edition, 1999.

[23] J. Kaipio and E. Somersalo. *Statistical and Computational Inverse Problems*, volume 160 of *Applied Mathematical Sciences*. Springer, 2005.

[24] M. Laine. MCMC Toolbox for Matlab. `helios.fmi.fi/~lainema/mcmc/`, 2006-2008. Accessed on March 28th, 2013.

[25] M. Laine. *Adaptive MCMC methods with applications in environmental and geophysical models*. PhD thesis, Lappeenranta University of Technology, Lappeenranta, Finland, 2008.

[26] S. B. Lippman, J. Lajoie, and B. E. Moo. *C++ Primer*. Addison Wesley, 4th edition, 2005.

[27] Andrew Makhorin and GLKP Development Team. GLPK (GNU Linear Programming Kit). http://www.gnu.org/software/glpk/, 2000-2012.

[28] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[29] A. Mira. On Metropolis-Hastings algorithms with delayed rejection. *Metron - International Journal of Statistics*, 59(3-4):231–241, 2001.

[30] David M. Mount and Sunil Arya. ANN: A library for approximate nearest neighbor searching. http://www.cs.umd.edu/~mount/ANN/, 2010. Version 1.1.2.

[31] E. E. Prudencio and K. W. Schulz. The Parallel C++ Statistical Library 'QUESO': Quantification of Uncertainty for Estimation, Simulation and Optimization. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7155 of *Lecture Notes in Computer Science*, pages 398–407. Springer Berlin / Heidelberg, 2012.

[32] C. P. Robert. *The Bayesian Choice*. Springer Verlag, 2nd edition, 2004.

[33] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer Verlag, 2nd edition, 2005.

[34] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Number 26 in Monographs on Statistics & Applied Probability. Chapman & Hall/CRC, 1986.

[35] Boost Development Team. Boost C++ Libraries. http://www.boost.org/, 1998-2013.

[36] GRVY Development Team. The groovy toolkit (GRVY). https://red.ices.utexas.edu/projects/software/wiki/GRVY, 1996-2009.

[37] MPICH Development Team. MPICH: High-Performance Portable MPI. http://www.mpich.org/, 2001-2013.

[38] Octave Development Team. GNU Octave. http://www.gnu.org/software/octave/, 1988–2013.

[39] Open MPI Development Team. Open MPI: Open source high performance computing. http://www.open-mpi.org/, 2008-2012.

[40] Dimitri van Heesch. Doxygen. http://www.doxygen.org/, 2009-2013.

APPENDIX A

Free Software Needs Free Documentation

The following article was written by Richard Stallman, founder of the GNU Project.

The biggest deficiency in the free software community today is not in the software—it is the lack of good free documentation that we can include with the free software. Many of our most important programs do not come with free reference manuals and free introductory texts. Documentation is an essential part of any software package; when an important free software package does not come with a free manual and a free tutorial, that is a major gap. We have many such gaps today.

Consider Perl, for instance. The tutorial manuals that people normally use are non-free. How did this come about? Because the authors of those manuals published them with restrictive terms—no copying, no modification, source files not available—which exclude them from the free software world.

That wasn't the first time this sort of thing happened, and it was far from the last. Many times we have heard a GNU user eagerly describe a manual that he is writing, his intended contribution to the community, only to learn that he had ruined everything by signing a publication contract to make it non-free.

Free documentation, like free software, is a matter of freedom, not price. The problem with the non-free manual is not that publishers charge a price for printed copies—that in itself is fine. (The Free Software Foundation sells printed copies of manuals, too.) The problem is the restrictions on the use of the manual. Free manuals are available in source code form, and give you permission to copy and modify. Non-free manuals do not allow this.

The criteria of freedom for a free manual are roughly the same as for free software. Redistribution (including the normal kinds of commercial redistribution) must be permitted, so that the manual can accompany every copy of the program, both on-line and on paper.

Permission for modification of the technical content is crucial too. When people modify the software, adding or changing features, if they are conscientious they will change the manual too—so they can provide accurate and clear documentation for the modified program. A manual that leaves you no choice but to write a new manual to document a changed version of the program is not really available to our community.

Some kinds of limits on the way modification is handled are acceptable. For example, requirements to preserve the original author's copyright notice, the distribution terms, or the list of authors, are ok. It is also no problem to require modified versions to include notice that they were modified. Even entire sections that may not be deleted or changed are acceptable, as long as they deal with nontechnical topics (like this one). These kinds of restrictions are acceptable because they don't obstruct the community's normal use of the manual.

However, it must be possible to modify all the technical content of the manual, and then distribute the result in all the usual media, through all the usual channels. Otherwise, the restrictions obstruct the use of the manual, it is not free, and we need another manual to replace it.

Please spread the word about this issue. Our community continues to lose manuals to proprietary publishing. If we spread the word that free software needs free reference manuals and free tutorials, perhaps the next person who wants to contribute by writing documentation will realize, before it is too late, that only free manuals contribute to the free software community.

If you are writing documentation, please insist on publishing it under the GNU Free Documentation License or another free documentation license. Remember that this decision requires your approval—you don't have to let the publisher decide. Some commercial publishers will use a free license if you insist, but they will not propose the option; it is up to you to raise the issue and say firmly that this is what you want. If the publisher you are dealing with refuses, please try other publishers. If you're not sure whether a proposed license is free, write to lice

APPENDIX B

GNU General Public License

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

(a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

(b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

(c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

(d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

(a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding

Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.

APPENDIX C

GNU Free Documentation License

Version 1.3, 3 November 2008

0. Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. Relicensing

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
    Copyright (C)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.