NASA/TM–2012-217561

UQTools: The Uncertainty Quantification Toolbox – Introduction and Tutorial

Sean P. Kenny
Langley Research Center, Hampton, Virginia

Luis G. Crespo
National Institute of Aerospace, Hampton, Virginia

Daniel P. Giesy
Langley Research Center, Hampton, Virginia

April 2012

NASA/TM–2012-217561

UQTools: The Uncertainty Quantification Toolbox – Introduction and Tutorial

Sean P. Kenny
Langley Research Center, Hampton, Virginia

Luis G. Crespo
National Institute of Aerospace, Hampton, Virginia

Daniel P. Giesy
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

April 2012

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

UQTools is the short name for the Uncertainty Quantification Toolbox, a software package designed to efficiently quantify the impact of parametric uncertainty on engineering systems. UQTools is a MATLAB®-based software package and was designed to be discipline independent, employing very generic representations of the system models and uncertainty. Specifically, UQTools accepts linear and nonlinear system models and permits arbitrary functional dependencies between the system's measures of interest and the probabilistic or non-probabilistic parametric uncertainty. One of the most significant features incorporated into UQTools is the theoretical development centered on homothetic deformations and their application to set bounding and approximating failure probabilities. Beyond the set bounding technique, UQTools provides a wide range of probabilistic and uncertainty-based tools to solve key problems in science and engineering.

Contents

1 Introduction **4**

1.1 UQTools Capabilities 4

1.2 Documentation 5

1.3 Additional Documentation Files Within the UQTools Software Package . . 6

1.4 Document Organization 6

2 Abbreviations **8**

3 UQTools Software Package **8**

3.1 Obtaining the Software Package 9

3.2 Organization of the UQTools Software Package 9

3.3 Getting Started with UQTools 11

4 Representation of Random Variables **11**

4.1 Using the Command-line Driven Interface 12

4.2 Using the Graphical Interface 15

5 Transformations Between Probability Spaces **16**

6 Generation of Sample Points **19**

6.1 Monte Carlo Sampling 20

6.2 Quasi-random Sampling 20

6.2.1 Hammersley Sequences 21

6.2.2 Halton Sequence Leaped, I 21

6.2.3 Halton Sequence Leaped, II 21

6.3 Latin Hyper-cube Samples and Quasi-Latin Hyper-cube Extensions 22

7 Response Surface Tools **23**

7.1 Multivariable Polynomials 23

7.2 Radial Basis Function Response Surfaces 26

8 Bounding the Safe/Failure Space Using Homothetic Deformations 29
8.1 Using the Exact Requirements Functions . . . 29
8.1.1 What Is It, How Is It Done? . . . 29
8.1.2 Utility of Bounding Sets . . . 31
8.1.3 How To Use `homodef` . . . 32
8.1.4 Extensions . . . 35
8.2 Homothetic Deformations over Surrogate Models . . . 35
8.2.1 Brief Sketch of the Method . . . 37
8.2.2 Convergence Testing and Algorithm Termination . . . 38
8.2.3 Restarting `cpv_by_RSapprox` To Perform Additional Iterations: . . 38

9 First Order Reliability Method 38

10 Risks and Potential Drawbacks 41
10.1 The Problem . . . 41
10.2 A Test for Non-globality of Solutions . . . 41

11 Estimating Failure Probability By Sampling 41
11.1 Rejection Method . . . 42
11.2 Conditional Sampling . . . 42
11.2.1 Conditional Sampling Outside a Hyper-rectangle . . . 43
11.2.2 Conditional Sampling Outside a Hyper-sphere . . . 44

12 Estimating Probabilistic Sensitivities Via Sampling 44
12.1 What Is It, How Is It Done? . . . 45
12.2 How To Use It . . . 46

13 Learn By Example 47
13.1 File: `example_homodef.m` . . . 47
13.2 File: `simple_homodef.m` . . . 47
13.3 File: `driver_spring_mass_us.m` . . . 48
13.4 File: `driver_spring_mass_pr.m` . . . 48
13.5 File: `driver_MPP_RS.m` . . . 49
13.6 File: `driver_simulink_us.m` . . . 49
13.7 File: `driver_simulink_pr.m` . . . 50
13.8 File: `example_sampling.m` . . . 50
13.9 File: `example_sum_iid.m` . . . 50
13.10 File: `example_probsenses.m` . . . 50
13.11 File: `example_probsenses1.m` . . . 50
13.12 File: `example_fit_RS.m` . . . 50
13.13 File: `mv_poly_example.m` . . . 50
13.14 File: `mean_var_approx.m` . . . 51
13.15 File: `pdf_approx.m` . . . 51
13.16 File: `example_call_MPP.m` . . . 52

References 53

List of Figures

1 `uqtools_gui` main window. 16
2 `uqtools_gui` detail window. 17
3 Maximal safe set of a hyper-rectangular reference set. 31
4 Hold some variables fixed, adjust others to find maximal safe set. 36
5 FORM and SORM 39

List of Tables

1 Additional Documentation Files in the UQTools Software Package 6
2 Abbreviations Used in this Manual 8
3 UQTools Directory Structure 9
4 Supported Distribution Types 13
5 Probability Space Transformation Functions 18
6 Functions for Multivariable Polynomial Response Surfaces 24
7 Functions for Radial Basis Response Surfaces 27
8 Radial Basis Functions 28

1 Introduction

UQTools is the short name for the Uncertainty Quantification Toolbox, a software package designed to efficiently quantify the impact of parametric uncertainty on engineering systems. UQTools was designed to be discipline independent, employing very generic representations of the system models and uncertainty. Specifically, UQTools accepts linear and nonlinear system models and permits arbitrary functional dependencies between the system's measures of interest and the probabilistic or non-probabilistic parametric uncertainty.

There are many novel features built into UQTools, but the primary innovation is related to the development of set bounding techniques and their application to reliability analysis. Set bounding techniques will be referred to as homothetic deformations in later sections of the document. In contrast to conventional approaches to uncertainty quantification, these techniques do not require the upfront definition of probabilistic models for the uncertain parameters. Instead, UQTools generates tight bounding sets to regions in the parameter space of acceptable system performance (safe domain) or unacceptable system performance (failure domain). These bounding sets are constructed using simple geometries (hyper-spheres or hyper-rectangles). Once these bounding sets have been calculated, the reliability analysis corresponding to particular probabilistic uncertainty models can be efficiently carried out. This feature enables accommodating for changes in uncertainty models with little additional computational effort. Figures of merit for uncertainty quantification that result from these techniques are robustness metrics that measure the separation between any given parameter realization and the failure domain, upper bounds to the failure probability and accurate estimates to failure probability.

Beyond the set bounding techniques mentioned above, UQTools provides a wide range of probabilistic and uncertainty-based tools to solve key problems in science and engineering. The entire UQTools package was written using the MATLAB technical computing language. "MATLAB®" and "MathWorks®" are registered trademarks of The MathWorks, Inc., of Natick, MA.

1.1 UQTools Capabilities

UQTools software realizes several complementary methods for performing a variety of uncertainty quantification tasks. UQTools currently has the following capabilities:

- Efficient methods for failure set bounding
  - By a *failure set bound* we mean the calculation of inner and outer approximations to the failure domain. The geometry of these approximations enables the calculation of upper and lower bounds to the failure probability. UQTools provides optimization-based tools for calculating such sets (see section 8).
- Hybrid methods for efficient estimation of failure probabilities
  - This is a technique which makes use of the failure set bounds to reduce the number of samples needed to achieve a given confidence in a sampling-based estimate of failure probability (see section 11). Improved efficiency is obtained by

using quasi-random sampling techniques (see section 6.2) instead of the usual pseudo-random number generators used in Monte Carlo techniques.

- First-Order Reliability Method (FORM)
  - Efficient failure probability approximation for low probability 'tail' events is provided by this classical technique (see section 9). This technique is used to approximate the failure probability for a single requirement function.
- Efficient deterministic sampling
  - These are techniques for generating quasi-random samples (see section 6.2), and have been shown to be substantially more efficient than conventional Monte Carlo.
- Efficient moment propagation methods
  - This technique enables the calculation of means and variances of polynomial systems whose input parameters are uniformly distributed (see section 7.1).
- Probabilistic sensitivity analysis
  - Analyze and rank the relative importance of system parameters by evaluating the dependence of the mean, variance and failure probability of the system response to changes in the mean and the variance of the probability density functions prescribing the inputs (see section 12).
- Response surface tools
  - Generation of fixed and adaptive surrogate models having radial or polynomial basis functions that can be readily integrated into the framework supporting standard uncertainty quantification tasks (see section 7).

The present document is intended to serve as an introduction and extended overview of UQTools. The reader who wishes to make use of this software may also want to refer to documentation contained in the UQTools software package. In addition to several files which are purely for documentation of specialized portions of this package, many of the M-files which are intended as user interfaces to UQTools capabilities contain extensive program preamble comments giving specifics of input and output parameters and details of what the function calculates. Further, it is strongly suggested that, for those users looking to maximize the utility of UQTools, they should focus their attention on section 13, "Learn By Example". In that section, detailed example problems are presented that illustrate the many features in UQTools.

1.2 Documentation

There are four principal sources of documentation for the theory behind the software in UQTools and instructions on how to make use of it:

1. The present document gives overviews of both theory and software usage.

2. Papers listed in the References at the end of this document give theoretical background.

3. Document files included in the UQTools distribution package include both specifics applicable to the software user and some of the papers referenced in the previous item.

4. Many of the MATLAB M-files in the UQTools distribution package provide detailed instructions in their program preamble comments as to what the M-file does and how to call it. For this reason, this document gives not only the name of each function but also the sub-directory in which it is located so that the interested reader may easily find it and refer to the program preamble comments.

1.3 Additional Documentation Files Within the UQTools Software Package

There are, in the sections where they apply, several documentation files in the UQTools software package. These are mentioned individually in the sections where they apply. For reference purposes, the complete list is given here:

Table 1. Additional Documentation Files in the UQTools Software Package

`UQTOOLS/RespSurf/Readme_mv_poly.txt`	For section 7.1
`UQTOOLS/Documents/mv_poly_whitepaper.pdf`	For section 7.1
`UQTOOLS/RespSurf/RBF_RS_data_structure.txt`	For section 7.2
`UQTOOLS/Documents/structural_safety.pdf`	Ref [1], for section 8.1
`UQTOOLS/Documents/aiaa_journal.pdf`	Ref [2], for section 8.1
`UQTOOLS/Documents/NASA-tp-2010-216189.pdf`	Ref [3], for section 8.1
`UQTOOLS/RelMeth/DATA_dictionary.txt`	For section 9
`UQTOOLS/RelMeth/readme_SORM.txt`	For section 9
`UQTOOLS/Documents/conditional_sampling.pdf`	Ref [4], for section 11.2
`UQTOOLS/Documents/AIAA-2009-2283-304.pdf`	Ref [5], for section 12

1.4 Document Organization

The remainder of this document has been organized so that any material which is prerequisite for a given section has been presented earlier in the document. This has the consequence that some of the more important and innovative material is presented later in the document.

For example, the material in section 8, “Bounding the Safe/Failure Space Using Homothetic Deformations” is one of the most important contributions of UQTools. However, the user must be conversant with several other features of UQTools in order to set up the input to the principal user interface functions in this section, namely function `homodef` in section 8.1 and function `cpv_by_RSapprox` in section 8.2.

To start with, the nature of uncertainty of each uncertain parameter must be specified to the software. This is explained in section 4, “Representation of Random Variables.” Here, the user is given two options for constructing this input item to the failure set bounding software, one option being command line driven and the other being performed interactively via a Graphical User Interface (GUI).

The software supporting the developments in section 8 requires analysis to be done in the native probability space of the uncertain parameters or in transformed versions to that probability space. Examples of this transformed space are the standard standard normal space or the space of uniform distributions on the unit hyper-cube. These transformations are explained in section 5, “Transformations Between Probability Spaces.”

Monte Carlo sampling is an uncertainty quantification tool of long standing. This technique makes use of sample points generated using a pseudo-random number generator. MATLAB includes pseudo-random number generators, and UQTools provides interfaces to them. However, quasi-random number generators, which create what is termed low discrepancy sequences, can provide a given confidence level in estimating, for example, failure probability using significantly fewer points than Monte Carlo. All of these sample point generating methods are explained in section 6, “Generation of Sample Points”, which also goes into Latin hyper-cube sample generation and a technique we call quasi-Latin hyper-cube extensions of sample sets. Besides being of interest in themselves, these sampling techniques are heavily used to generate adaptive surrogate models suitable for calculating failure set bounds (see section 8.2 and function `cpv_by_Rsapprox`).

UQTools includes software for generating both multi-variable polynomial response surfaces and radial basis function response surfaces. These are explained in section 7, “Response Surface Tools.” These may be used independently of the remaining functionality of UQTools. However, as with sampling techniques, these response surface techniques are heavily used in section 8.2, and the user of `cpv_by_Rsapprox` must understand them well enough to properly formulate input parameters.

The First Order Reliability Method (FORM) method of estimating the failure probability depends on locating the *Most Probable Point* (MPP) of failure of an uncertain system. In section 9, “First Order Reliability Method,” two ways are provided to access software which calculates the FORM approximation for a system with a single requirement function (which defines what is called in the FORM literature the *limit state surface*). One way uses the same input structure as the failure set bounding software in section 8. The other way requires mastering a different representation for the uncertainty, but allows limited specification of correlations between the uncertain parameters.

Both sections 8 and 9 make use of non-linear constrained optimization in their calculations. The theoretical justification for these techniques requires that global optima be found. One risk with using optimization techniques is that the optimum found may be a local optimum, but not global. In section 10, “Risks and Potential Drawbacks,” we expand on this potential drawback and provide a sampling based technique by which the user can test the results of the optimization based calculations and search for evidence that the converged optimum is local but non-global.

In section 11, “Estimating Failure Probability By Sampling,” we present several methods by which pre-knowledge of a failure bounding set may be leveraged to improve the efficiency of a sampling-based estimation of failure probability.

Section 12, “Estimating Probabilistic Sensitivities Via Sampling,” represents a change of emphasis. Here, the observation is made that the probability distribution of an uncertain parameter may not be known exactly, and the question is addressed of how sensitive the statistics of a performance metric such as expected value, variance, or failure probability are to such properties of the uncertain parameter distribution function as its mean, variance, or the bounds of its support interval.

Section 13, “Learn By Example,” introduces the reader to several MATLAB M-files which provide examples of the use of the software in UQTools. It is hoped that by examining and executing these files, the user can gain further understanding of the functionality of UQTools. Additionally, these files might serve as templates for the user’s own applications.

2 Abbreviations

Table 2. Abbreviations Used in this Manual

CDF	Cumulative Distribution Function
CPV	Critical Parameter Value
CSR	Critical Similitude Ratio
FORM	First Order Reliability Method
GUI	Graphical User Interface
MPP	Most Probable Point (of failure)
PDF	Probability Density Function
PSM	Parametric Safety Margin
RBF	Radial Basis Function
RI	Reliability Index
RPSM	Rectangular Parametric Safety Margin
RRI	Rectangular Reliability Index
RS	Response Surface
SORM	Second Order Reliability Method
SPSM	Spherical Parametric Safety Margin
SRI	Spherical Reliability Index

3 UQTools Software Package

In addition to basic MATLAB, UQTools also requires the MATLAB Optimization Toolbox and the MATLAB Statistics Toolbox. The bulk of UQTools was developed under MATLAB version R2008b, but it should be compatible with more recent releases. The current version of the UQTools software package contains about 9.5 megabytes of data contained in a directory named `UQTOOLS` and its 17 subdirectories. There are about 300 MATLAB M-files and several other files, including more detailed documentation about parts of the package than are given in the present document and some data files.

3.1 Obtaining the Software Package

The package is available to eligible recipients, as determined by NASA regulations for the distribution of NASA generated software. To obtain a copy of the UQTools package, apply to:

Dr. Sean P . Kenny
Dynamic Systems and Control Branch
Mail Stop 308
NASA Langley Research Center
Hampton, VA 23681-2199
Phone: 757-864-6612
FAX: 757-864-7722
E-mail: sean.p.kenny@nasa.gov

3.2 Organization of the UQTools Software Package

The following table gives a list of the subdirectories of UQTools directory `UQTOOLS` together with a synopsis of the contents of each subdirectory:

Table 3: UQTools Directory Structure

DIRECTORY NAME	Directory contents
`UQTOOLS/`	Subdirectories and a template `startup.m` file
`UQTOOLS/Distributions`	Distribution functions to supplement those provided in the Statistics Toolbox
`UQTOOLS/Examples`	Subdirectories of example files
`UQTOOLS/Examples/General`	Example files to exercise much of the software in UQTools
`UQTOOLS/Examples/MPP_based`	Example files to exercise function `MPP_FORM` from subdirectory `UQTOOLS/RelMeth`
`UQTOOLS/Homodef`	Subdirectories containing software and documentation for calculating maximal safe or failure homothetic deformations
`UQTOOLS/Homodef/Core`	The top level software for calculating maximal homothetic deformations
`UQTOOLS/Homodef/Papers`	Papers presenting the theory behind using homothetic deformations of reference sets in uncertainty analysis

Table 3: UQTools Directory Structure (Continued)

`UQTOOLS/Homodef/RSapprox`	Software for approximating maximal homothetic deformations of reference sets using response surface approximations to the system requirements function(s)
`UQTOOLS/Homodef/Support`	Two subdirectories
`UQTOOLS/Homodef/Support /Robdes`	Support files for `UQTOOLS/Homodef/Core`
`UQTOOLS/Homodef/Support /Utilitarian`	Support files for `UQTOOLS/Homodef/Core`
`UQTOOLS/Probsensitive`	The paper "Sampling-based Strategies for the Estimation of Probabilistic Sensitivities" and software to implement the ideas in that paper
`UQTOOLS/RelMeth`	Software to implement FORM, the First Order Reliability Method, and SORM, the Second Order Reliability Method
`UQTOOLS/RespSurf`	Software and documentation for fitting and evaluating response surfaces based on multi-variable polynomials and radial basis functions
`UQTOOLS/Sampling`	Software for sampling uniformly from the unit hyper-cube and from multi-dimensional vectors with various random distributions
`UQTOOLS/Transforms`	Software for generating probability preserving transformations between an arbitrary image space of independent random variables and two special instances - standard normal space and uniform unit hyper-cube space
`UQTOOLS/Uqtools_GUI`	Software for a graphical user interface (GUI) to assist in the creation of a MATLAB vector `rv` in the base MATLAB workspace, each component of which describes a random variable to UQTools

Table 3: UQTools Directory Structure (Continued)

`UQTOOLS/Documents`	Copies of references [1], [2], [3], [4], [5], and a polynomial response white paper

3.3 Getting Started with UQTools

The UQTools software package (obtained as in section 3.1) is distributed as a single ZIP file. When unzipped, the contents are arranged as in section 3.2. The UQTOOLS directory may be placed anywhere on the computer where the user has read/write permission. The user should make a copy of file `UQTOOLS/startup.m` in a working directory, edit it so the variable `INSTALLDIR` is set to the full path name of the `UQTOOLS` directory of the UQTools installation, and start MATLAB in the working directory. This instance of MATLAB will have access to the complete UQTools Toolbox.

4 Representation of Random Variables

For purposes of UQTools, the phrase *random variable* will refer to a real scalar valued random variable and the phrase *random vector* will be used for vector whose components are random variables. "Random vector" includes "random variable" as a special case. So long as the components of a random vector are *independent* random variables, UQTools finds all the information it needs about the probabilistic nature of the random vector from functions that calculate the *cumulative distribution function* (CDF), its derivative, the *probability density function* (PDF), and the inverse CDF of each component random variable. If P is a random vector, its CDF is denoted by F_P and is defined by the property that, for all x in the Euclidean space of the same dimension as P,

$$F_P(x) = \mathrm{P}[P \leq x],$$

where $\mathrm{P}[E]$denotes the probability of the event E and $P \leq x$ represents the event that every component of the random vector P is no larger than the corresponding component of x. A random *sample* p of P is any one specific value that P might assume.

UQTools allows users to create a MATLAB data representation of *random variables* and *random vectors* using a command-line driven interface called `setrvs` or through a graphical user interface called `uqtools_gui`. UQTools depends upon the existence of software for many of the distributions defined in the MATLAB Statistics Toolbox, and also provides software for additional distributions not included in the Statistics Toolbox. Section 4.1 gives the complete list of distributions that UQTools currently supports. Please note that two of these, specifically, Interval and Deterministic Constant are not technically distributions, but are included here because UQTools data structure can accommodate these additional definitions. It is important to note that the vast majority of UQTools is based upon the assumption that random variables are independent. The limited exception to this is the software contained in `UQTOOLS/RelMeth` that can accommodate correlated normal or correlated lognormal random variables.

4.1 Using the Command-line Driven Interface

To use the command-line driven interface (`setrvs`) to fully prescribe a random variable, UQTools permits the following syntax:

```
rv = setrvs(typedist,paramv,name,description,number);
```

The first two inputs are required and are described below:

- `typedist` - This is a string variable and is a unique identifier given to a type of distribution. The list of currently supported values for `typedist` is given in Table 4.
- `paramv` - This is a vector of parameters used to define a particular random variable. The length of this vector depends upon the selected distribution. Typical parameters are the mean and standard deviation of a normally distributed random variable or the left and right endpoints of the support interval of a uniformly distributed random variable.

The command-line driven interface (`setrvs`) also allows three additional inputs. These additional inputs are optional, and are typically used to attach information to the random variable which identifies and documents it. These optional inputs are:

- `name` - A string containing an arbitrary name for the random variable.
- `description` - A string containing any descriptive information about the random variable.
- `number` - A numeric scalar value to uniquely identify the random variable.

Table 4 gives the values of `typedist` currently supported by UQTools and some information about the form of `paramv`. For a complete description of the variable `paramv` refer to the documentation accessible by the MATLAB help command. For example, the MATLAB command "`help ncfpdf`" (or "`help ncfcdf`" or "`help ncfinv`") would show that, besides an initial input which provides the value(s) at which the function is to be evaluated, the Noncentral F Distribution (whose typedist is `ncf`) requires three parameters, a numerator degree of freedom, a denominator degree of freedom, and a non-centrality parameter. This is the information the user must supply to `setrvs` in input argument `paramv`. Note that the first 17 entries (through Weibull Distribution) in Table 4 are in the MATLAB Statistics Toolbox while the remaining entries are part of the UQTools software, located in directory `UQTOOLS/Distributions`. These directories must be on the user's search path (see MATLAB command "`path`") so that the "`help`" command knows where to look for the information.

Table 4: Supported Distribution Types

Distribution Type	`typedist`	`paramv`
Beta Distribution	`beta`	`[A, B]` - real scalars, left and right exponents
Chi-Square Distribution	`chi2`	`[V]` - positive integer `V` is the degree of freedom
Noncentral Chi-Square Distribution	`ncx2`	`[V, DELTA]` - positive integer `V` is degree of freedom and positive real `DELTA` is the noncentrality parameter
Exponential Distribution	`exp`	`[mu]` - scalar `mu` is the mean parameter
Extreme Value Distribution	`ev`	`[mu, sigma]` - `mu` is the location parameter, `sigma` is the scale parameter
Generalized Extreme Value Distribution	`gev`	`[K, sigma, mu]` - `K` is the shape parameter, `sigma` is the scale parameter, `mu` is the location parameter
Generalized Pareto Distribution	`gp`	`[K, sigma, theta]` - `K` is the index (shape) parameter, `sigma` is the scale parameter, and `theta` is the threshold (location) parameter
F Distribution	`f`	`[V1, V2]` - `V1` and `V2` are positive integers, the numerator and denominator degree of freedom, respectively
Noncentral F Distribution	`ncf`	`[NU1, NU2, DELTA]` - `NU1` and `NU2` are positive integers, the numerator and denominator degree of freedom, respectively, and `DELTA` is a positive noncentrality parameter
Gamma Distribution	`gam`	`[A, B]` - `A` is the shape parameter, `B` is the scale parameter
Lognormal Distribution	`logn`	`[mu, sigma]` - `mu` and `sigma` are the mean and standard deviation, respectively, of the associated normal distribution
Normal Distribution	`norm`	`[mu, sigma]` - `mu` and `sigma` are the mean and standard deviation, respectively
Rayleigh Distribution	`rayl`	`[B]` - `B` is the scale parameter
Student's t-Distribution	`t`	`[V]` - `V` is the positive integer degree of freedom

Table 4: Supported Distribution Types (Continued)

Noncentral t-Distribution	`nct`	`[V, DELTA]` - `V` is the positive integer degree of freedom and `DELTA` is the noncentrality parameter
Uniform Distribution	`unif`	`[A, B]` - `A` and `B` are, respectively, the lower and upper endpoints of the support interval
Weibull Distribution	`wbl`	`[A, B]` - `A` is the scale parameter, `B` is the shape parameter
Triangular Distribution	`tri`	`[A, M, B]` - `A` and `B` are, respectively, the lower and upper endpoints of the support interval, and `M` is the location of the apex of the triangle
Interval	`interval`	`[a]` or `[a, b]` - support set is, respectively, the interval from `-abs(a)` to `abs(a)` or the interval from `a` to `b`
Generalized Beta Distribution	`gbeta`	`[A, B, XMIN, XMAX]` - the beta distribution with positive left and right exponents `A` and `B` is translated and scaled to the interval domain with left and right endpoints `XMIN` and `XMAX`
Generalized Bimodal	`gbimodal`	`[A, B]` - `A` and `B` are, respectively, the lower and upper endpoints of the support interval
Batson Beta Distribution	`batson`	`[XMIN, XML, XMAX, CONF]` - `XMIN` and `XMAX` are, respectively, the lower and upper endpoints of the support interval, `XML` is the "most likely" point in that interval, and `CONF` is an integer between 1 (least confident in `XML`) to 5 (most confident in `XML`)
Deterministic Constant	`det`	`[A]` - `A` is the deterministic value

Below is an example that defines a single, normally distributed random variable:

```
rv = setrvs('norm',[0,1],'Var_1','Normal var',1);
```

Below is an example that defines six independent random variables using generalized beta distributions:

```
rv(1)=setrvs('gbeta',[2,2,0.9,1.1],'m1',[],1);
rv(2)=setrvs('gbeta',[2,2,1.4,1.6],'m2',[],2);
rv(3)=setrvs('gbeta',[2,2,32,36],'k1',[],3);
rv(4)=setrvs('gbeta',[2,2,37,41],'k2',[],4);
rv(5)=setrvs('gbeta',[2,2,0.1,0.35],'alpha',[],5);
rv(6)=setrvs('gbeta',[2,2,0.006,0.01],'beta',[],6);
```

The MATLAB variable called `rv` in the previous example could have had any legal MATLAB variable name. By contrast, if the graphical user interface `uqtools_gui` is called, the MATLAB variable it creates is always called `rv`.

The UQTools GUI (`uqtools_gui`) was developed to assist in prescribing the random variables, i.e., defining the `rv` structure. The interface is called `uqtools_gui`. This interface may be called in one of two cases. Case 1 is when there is no predefined `rv` structure, that is, when the user wants to prescribe a completely new `rv` structure. For this case, the interface may be initiated using either of the following function calls:

```
>> uqtools_gui
```

or equivalently

```
>> uqtools_gui('start')
```

Both options above will initialize the interface exactly the same way by opening the interface and defining a single random variable in the `rv` structure. This default random variable is prescribed as normally distributed with zero mean and unity variance. A copy of the `rv` structure will also appear in the user's base workspace upon initiation.

Case 2 is when the user wants to augment an existing `rv` structure. For this case, the interface may be initiated using the following function call:

```
>> uqtools_gui('start',rv)
```

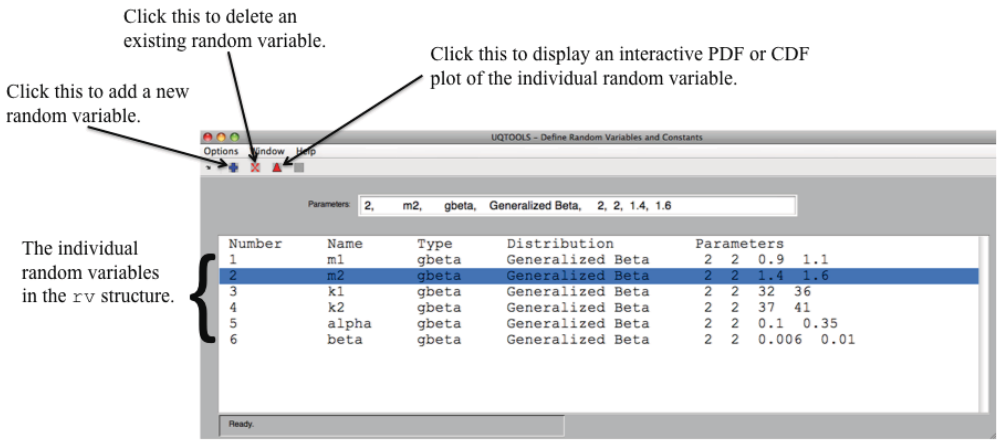
Note: The `uqtools_gui` function does not have output arguments, but instead uses the MATLAB function `assignin` to update the structure `rv` in the base workspace. If the `rv` structure exists on initialization, a copy of it is made and placed in variable `rv_org`.

4.2 Using the Graphical Interface

Once the interface is initiated as describe above, the user will now have access to additional graphical elements to modify the rv structure. This is best described using an example. Assume that the user has predefined six independent random variables using generalized beta distributions and then invokes uqtools_gui.

```
rv(1)=setrvs('gbeta',[2,2,0.9,1.1],'m1',[],1);
rv(2)=setrvs('gbeta',[2,2,1.4,1.6],'m2',[],2);
rv(3)=setrvs('gbeta',[2,2,32,36],'k1',[],3);
rv(4)=setrvs('gbeta',[2,2,37,41],'k2',[],4);
rv(5)=setrvs('gbeta',[2,2,0.1,0.35],'alpha',[],5);
rv(6)=setrvs('gbeta',[2,2,0.006,0.01],'beta',[],6);
uqtools_gui('start',rv)
```

A screenshot of this example is given in Figure 1:


Click this to delete an existing random variable.
Click this to display an interactive PDF or CDF plot of the individual random variable.
Click this to add a new random variable.
The individual random variables in the rv structure.
UQTOOLS – Define Random Variables and Constants
Options
Window
Help
Parameters:

```
2,     m2,    gbeta,   Generalized Beta,    2,  2,  1.4,  1.6
```

```
Number      Name        Type        Distribution          Parameters
1           m1          gbeta       Generalized Beta       2   2   0.9   1.1
2           m2          gbeta       Generalized Beta       2   2   1.4   1.6
3           k1          gbeta       Generalized Beta       2   2   32   36
4           k2          gbeta       Generalized Beta       2   2   37   41
5           alpha       gbeta       Generalized Beta       2   2   0.1   0.35
6           beta        gbeta       Generalized Beta       2   2   0.006   0.01
```

```
Ready.
```

Figure 1. `uqtools_gui` main window.

The user can click on the red density function image in the top toolbar to graphically modify an existing random variable using an interactive PDF or CDF plotting utility. This action will bring up the graphical interface below:

Using this interface, the user can easily change the distribution type, i.e., `typedist` by using the pull-down menu and selecting from any allowable distribution as presented in Table 4. The parameters, i.e., elements within the `paramv` vector, are displayed near the bottom of the interface. Note, for a generalized beta, the four parameters are displayed and may be changed by simply moving the corresponding slider bars shown in Figure 2.

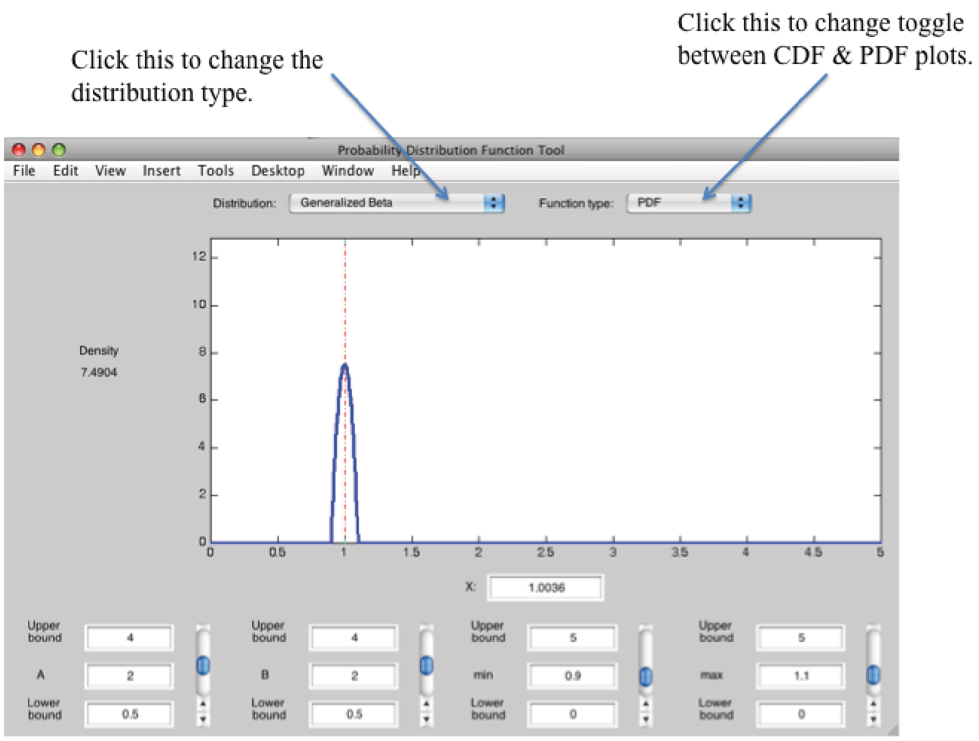
The `rv` structure is constantly updated in the base workspace, so no additional action is required to save changes. Any change is instantaneously reflected and automatically saved in the base workspace. Recall, that the original `rv` structure that existed on initialization of the GUI was automatically copied and placed in variable `rv_org`.

5 Transformations Between Probability Spaces

The UQTools functions discussed in this section are kept in UQTools subdirectory `UQTOOLS/Transforms`. Details of function usage are found in the program preamble or "help" comments.

For the analyses done in UQTools, the original parameter probability space, P-space[1], is not always the best domain to use. To that end, UQTools provides probability preserving transformations that allow the user to perform the analyses in potentially friendlier probability spaces. U-space is the probability space of a random vector U that has an uncorrelated

[1]Denote the space of values of the uncertain parameters by "P-space", and represent its dimension by d. Represent the uncertainties in the parameters by a random vector P. Under the CDF F_P of P, P-space becomes a probability space. The uncertainty quantification provided by UQTools can result from estimating or bounding probabilities of events (such as the failure event of the system) in P-space.


Click this to change the distribution type.

Click this to change toggle between CDF & PDF plots.

Probability Distribution Function Tool

Distribution: Generalized Beta

Function type: PDF

Density 7.4904

X: 1.0036

Upper bound	4	Upper bound	4	Upper bound	5	Upper bound	5
A	2	B	2	min	0.9	max	1.1
Lower bound	0.5	Lower bound	0.5	Lower bound	0	Lower bound	0

Figure 2. `uqtools_gui` detail window.

standard normal distribution. V-space is the probability space of a random vector V that has the uncorrelated uniform distribution over the interval $[0,1]$. W-space is the probability space of a random vector W that has an uncorrelated uniform distribution over the interval $[-1,1]$.

So long as the component random variables of the random vector P are mutually independent and have continuous distributions chosen from the collection supported by the UQTools function `setrvs` and the interactive tool `uqtools_gui`, UQTools provides probability preserving transformations that allow the P-space variables to be replaced by U-space or V-space variables and visa versa. One advantage of using these probability preserving transformations is to provide an analysis domain in which the variables all have the same scaling. FORM, the First Order Reliability Method, (see section 9, page 38, and UQTools functions `call_MPP` and `MPP_FORM`) depends on transforming the problem into U-space to take advantage of favorable properties of the normal distribution. Many sampling methods provide samples in V-space that may then need to be transformed to one of the other spaces if samples are needed there. Response surfaces, particularly Radial Basis Function Response Surfaces, might provide better approximations to the functions they approximate if their host domain is U-space or V-space instead of P-space, since in U-space and V-space the variables are equally scaled.

The probability structure of P-space provides the information necessary to define these transformations. By the assumed independence of the random variables comprising the random vector P, the transformations are carried out component by component. Mapping a random variable by its CDF results in the transformed random variable being uniformly distributed on the interval $[0,1]$. Further mapping of that transformed variable by the inverse CDF of a standard normal random variable results in the further transformed random variable having the standard normal distribution. All of these mappings are invertible, so any of P-space, U-space, and V-space can be mapped onto any other by a probability preserving transformation.

UQTools provides six functions to accomplish these transformations:

Table 5: Probability Space Transformation Functions

Name	Function performed
`p2v`	Transform vectors from P-space into V-space.
`p2u`	Transform vectors from P-space into U-space.
`u2p`	Transform vectors from U-space into P-space.
`v2p`	Transform vectors from V-space into P-space.
`p2w`	Transform vectors from P-space into W-space.
`w2p`	Transform vectors from W-space into P-space.
`u2v`	Transform vectors from U-space into V-space.
`v2u`	Transform vectors from V-space into U-space.

The first input argument to each of these functions is a matrix whose rows contain samples from the space that is being transformed and each column represents a dimension of the d-dimensional space. The second argument is a d-vector of `rv` style MATLAB structures that can come from UQTools function `setrvs` or the interactive tool `uqtools_gui` (see section 4). For the first six functions in Table 5, this second argument is required, and must describe the probabilistic structure of the random variable P. For the last two of these functions, this second argument is optional, if provided it is not referenced, and is included only for calling sequence compatibility with the first four functions. The first output of each of these functions is the matrix of transformed samples in the probability space. An optional second output argument gives the gradient of the output points with respect to the input points.

The transformation functions find use in some of the example M-files in directory `UQTOOLS/Examples/General`: `analytical.m`, `driver_MPP_RS.m`, `example_fit_RS.m`, `example_probsenses1.m`, `example_sampling.m`, `mean_var_approx.m`, `pdf_approx.m`, and `spring_mass.m`.

See also the description of some of these files in section 13, "Learn By Example".

6 Generation of Sample Points

The UQTools functions discussed in this section are kept in UQTools subdirectory `UQTOOLS/Sampling`. Details of function usage are found in the program preamble or "help" comments.

Traditional Monte Carlo analysis is based on analyzing the system model at a pseudo-random sampling of the uncertain parameters. This is not necessarily ideal, since the scattering of random samples typically shows clumps (over-sampling) and voids (under-sampling). It is more efficient to use deterministic low discrepancy (a.k.a. quasi-random) sampling methods. By "more efficient" we mean that by using low discrepancy samples we can achieve greater confidence in results from a given number of low discrepancy samples than from the same number of random samples, or achieve the same confidence with fewer low discrepancy samples than random samples. The phrase "low discrepancy" has a technical meaning, and for those readers wanting more information, see Definition 2.2 *discrepancy* of [6]. It is good for the sequence discrepancy to be small (low) since it means that there is no excessive clumping of the sample points and the voids in the sample tend to be small. By using the transformations of section 5, a low discrepancy sequence can be transformed into a representative sample of an arbitrary distribution.

There are many reasons one might want to have a sampling of points from the uncertain parameter space. For example, if $f_P(p)$ is the PDF of P and $g(p)$ is an arbitrary function of p, the expected value of g, defined by the integral, $\int g(p)f_P(p)dp$ can be estimated by averaging the values of $g(p)$ over a representative sampling of the random variable P, whether the samples are pseudo-random as in Monte Carlo analysis or quasi-random. As a special case, the probability of an event can be estimated as the average number of sample points falling in the event (take g to be the indicator function of the event). Also, analyzing a system at a sampling of points can provide the data needed to build a response surface. In addition, if prior analysis has regions of the uncertain parameter space (or one of its transforms) where system behavior has been determined to be satisfactory, then to estimate failure probability by sampling it is only necessary to sample conditionally outside this known safe region. UQTools provides methods to implement such sampling techniques.

6.1 Monte Carlo Sampling

The sampling techniques of UQTools extend those of MATLAB such as `rand` and `randn`. The distributions in the MATLAB Statistics Toolbox and the few distributions added by UQTools each has a function whose name concatenates the root name of the distribution with the characters `rnd`, and which generates matrices or arrays of samples having the named distribution. Input to these functions includes the same distributions defining parameters as the MATLAB functions that calculate the PDF, the CDF, and the inverse CDF. One way to generate Monte Carlo samples in P-space is to use UQTools function `mc` that in turn makes use of these `xxxrnd` functions. Another way to generate Monte Carlo samples in P-space is to use MATLAB functions `rand` or `randn` to generate Monte Carlo samples in V-space or U-space, respectively, and then use the UQTools transformation functions `u2p` or `v2p`, respectively, with the vector `rv` (see section 4, "Representation of Random Variables") of structures which defines the probabilistic structure of P-space. This will transform the U- or V-space samples into P-space.

6.2 Quasi-random Sampling

Quasi-random number generators are designed to produce sequences of points in the unit hyper-cube that distribute in a highly uniform manner. They seek to minimize the sequence discrepancy. As more points are generated, the tendency is for the quasi-random number generator to fill in any gaps in the initial segment of the sequence.

Quasi-random sequences do not do as well on many statistical tests for randomness as do sequences produced by pseudo-random number generators. But, passing statistical tests is not the point of quasi-random sequences. The point is to fill the unit hyper-cube uniformly, and to do this with initial segments distributed as uniformly as possible over the unit hyper-cube.

It has been our experience that using low discrepancy sequences to estimate probabilities produces superior results to those produced using pseudo-random sequences. For that reason, UQTools includes several functions for generating quasi-random sequences. Some of these are original works by the developers of UQTools and some are interfaces to quasi-random number generators that have been included in distributions of MATLAB since version R2008a. In these versions, the section "Generating Quasi-Random Numbers" in the Statistics Toolbox "User Guide" provides background information on quasi-random sequences and serves as an introduction to the MATLAB software.

There is a considerable body of literature on the subject of generating low discrepancy sequences. Many schemes have been proposed, some quite complicated. This has been necessitated by some drawbacks experienced by simpler schemes. In particular, these drawbacks can take the form of degraded behavior as the dimension of the sample increases. One way to judge the behavior of a d-dimensional sequence of sample points generated by a quasi-random number generator is to make a scatter plot of the numbers in each pair of coordinates and visually inspect the plots for coverage and uniformity. MATLAB provides a tool, function `plotmatrix`, for this purpose. If any of the two dimensional scatter plots shows problems like lack of coverage of the unit square, voids, clumps, or clear patterning, then one concludes that the sequence is not adequately representative of a sampling of the

d-dimensional uniform distribution. The converse is not true; and, unfortunately, UQTools provides no definitive test of goodness of sampling.

UQTools provides software to generate quasi-random sequences. In what follows, the discussion will be about Hammersley sequences and Halton Sequences.

6.2.1 Hammersley Sequences

A technique that is of historical interest in the field of quasi-random sequence generation is called Hammersley sequence sampling. A UQTools function, `hamseqsamp`, was written to generate these sequences (we use the plural here because there is a different sequence for each choice of V-space dimension and of sequence length). A second UQTools function, `hss`, uses a Hammersley sequence as a starting point to generate a quasi-random sequence in P-space using the function `v2p` and the structure `rv` which defines the probabilistic structure of P-space. This technique may be of academic interest, but it is not recommended for serious sampling. It has two drawbacks: (1) Once a Hammersley sequence has been generated, it cannot be extended to a longer Hammersley sequence - every entry of the first component of a sequence of Hammersley vectors is sensitive to the sequence length which was specified when it was generated. (2) Hammersley sequences are susceptible to 2-D scatter plot patterning - for a 500-point sequence this is already noticeable in the 8 dimensional sequence and blatant by the 14-dimensional sequence.

6.2.2 Halton Sequence Leaped, I

Another technique for generating quasi-random sequences goes under the name "Halton". The basic Halton sequence shares with Hammersley sequences the 2-D scatter plot patterning drawback. In fact, the d-dimensional basic Halton sequence exactly matches coordinates 2 through $d+1$ of the $(d+1)$-dimensional Hammersley sequence. The saving grace of the Halton sequence is that, if one picks a favorable value for a leap parameter q, and instead of taking the first, second, third, etc. elements of the Halton sequence, the elements 1, $1+q$, $1+2q$, etc., are chosen, a much more satisfactory distribution of points in the unit hyper-cube can be obtained. Such a sequence is called a "Halton sequence leaped". UQTools includes a function `halslget` that returns a Halton sequence leaped in the form of a matrix whose rows are the sample points. This function is contained in UQTools and does not depend on any other MATLAB Toolboxes. It takes four integer parameters: the dimension of the sample space; the number of samples desired; a number which, together with the dimension, determine the leap parameter; and a number which tells `halslget` how many elements of the infinite Halton sequence leaped to skip over before recording values in the output matrix. This skip parameter allows a previously generated Halton sequence leaped to be extended by the addition of more Halton sequence leaped points.

6.2.3 Halton Sequence Leaped, II

The MATLAB Statistics Toolbox provides tools for generation of Halton sequences. The MATLAB version is more versatile than `halslget` and should be used if it is available. As of this writing, however, the MATLAB version has one serious drawback. It asks the user to input a leap parameter that is the actual number of basic Halton sequence elements skipped over before taking the next element to add to the sample set. An injudicious choice of this parameter can result in a sample set which is confined to a *proper* sub-hyper-rectangle of the unit cube. To overcome this drawback, UQTools provides function `haltonget` as an interface to the MATLAB capability. The first four parameters to `haltonget` are the same as those to `halslget`, and if only those parameters are used, the results of calling the two functions are the same (to within round-off error). The optional fifth parameter can be used to turn on the MATLAB option to "scramble" the sequence. Based on an examination of 2-D scatter plots, it appears that this improves the discrepancy of the output and its use is recommended by the UQTools developers. The optional sixth parameter gives access to additional Halton Sequence Leaped sequences that fall in the cracks between the elements generated by the basic Halton algorithm.

6.3 Latin Hyper-cube Samples and Quasi-Latin Hyper-cube Extensions

Suppose that the unit interval $[0,1]$ is partitioned into N sub-intervals (sometimes called *bins* in the statistics community) of equal length $1/N$. A collection of N sample points which are vectors in a unit hyper-cube is considered to be a *Latin Hyper-cube Sample* if each coordinate of the sample set places one value in each bin. This in itself is not enough to guarantee that the sequence distributes uniformly over the hyper-cube. So it is also stipulated that the order in which the bins are occupied changes randomly from one coordinate to another.

As it says in the MATLAB documentation for *Generating Quasi-Random Numbers*, "Though not quasi-random in the sense of minimizing discrepancy, these sequences nevertheless produce sparse uniform samples ...." Such sequences may be generated by the MATLAB Statistics Toolbox function `lhsdesign`. UQTools makes no direct use of this function.

UQTools does provide a function to extend an existing set of arbitrarily placed sample points so that the resulting sample set is as nearly a Latin hyper-cube sample as possible. Such an extension is called a *quasi-Latin hyper-cube extension*. Suppose an existing sampling of points from the unit hyper-cube contains M points and it is to be extended by adding N new points. This is accomplished by UQTools function `qlh_extend`. The inputs to `qlh_extend` are the existing set of M points to be extended and a set of N points from the unit hyper-cube which are used as seed points to start the extension process. For best final results, these N seed points should be distributed uniformly over the unit hyper-cube. One of the previously mentioned sample generating techniques may be used to generate this seed set. The function `qlh_extend` then partitions the unit interval into $M+N$ equal subintervals; on a coordinate by coordinate basis it takes note of which of these intervals is already occupied by the corresponding coordinate of a point in the existing sample set; and, one new seed point at a time, it affinely transforms each of the seed point's coordinates into an unused interval. These newly occupied subintervals are now considered occupied for the purpose of transforming the next seed point.

Each seed point is thus transformed into a final sample point whose coordinates are the unique occupiers of their respective bins. However, since the original set of N points was arbitrary, it may not have the unique occupancy property of a Latin hyper-cube sample for this set of $M+N$ bins. Thus, the extended sample might not be a truly Latin hyper-cube sampling. This is the reason for the "quasi" in the name. However, it is permissible to ask `qlh_extend` to "extend" an empty set of samples by setting the "existing set" to the empty matrix `[ ]`. In this case, the "extended" set is actually a Latin hyper-cube sampling.

7 Response Surface Tools

The UQTools functions discussed in this section, and some of the documentation are kept in UQTools subdirectory `UQTOOLS/RespSurf`. Documentation, including details of function usage, can be found in the program preamble or "help" comments of the function files and in the files `RBF_RS_data_structure.txt`, `Readme_mv_poly.txt`, and `UQTOOLS /Documentation/mv_poly_whitepaper.pdf`.[2].

UQTools has the capability to generate response surfaces. For purposes of UQTools, a *response surface* (RS) is function that provides a global approximation to the behavior of a parametrized system based on results calculated at various points in the parameter space (the data points). Evaluating a RS of a system at a point of the parameter space is typically much more computationally efficient than evaluating a high fidelity physics model of the system. So, replacing a high fidelity model in an analysis by a RS approximation can rapidly produce an approximate answer to the analysis. In section 8.2, UQTools approximates requirements functions by RSs to rapidly approximate bounds for failure sets.

The RS function forms that have been implemented in UQTools are (multivariable) polynomials and *radial basis function* (RBF) RSs (with optional polynomial pre-fit). There is no hard coded limit on the number of parameters, the order of the approximating polynomial, or the number of data points used in defining the RSs. However, calculating the coefficients used in these RSs requires solving linear equations the size of whose coefficient matrices depends on these numbers. Some common sense care must be exercised in the choice of how many data points to use in fitting the RSs and, in the case of an RS using a polynomial component, how many monomial terms are in the polynomial. The number of monomial terms is determined by the dimension of the parameter space and the polynomial order chosen (and the value of an optional truncation parameter which limits the number of variables used in cross terms of the polynomial) or by explicitly prescribing which monomials are to be used.

7.1 Multivariable Polynomials

A *polynomial* is a linear combination of monomials. A *monomial* in the variables $p_1, ..., p_d$ is a product of the form $p_1^{i_1} \cdots p_d^{i_d}$ where, for $1 \leq j \leq d$, i_j is a non-negative integer. The *order* of a monomial is the sum of its exponents. The *order* of a polynomial is the maximum order of its monomials. A polynomial can be specified by specifying which monomials are used in it and then specifying what coefficient is multiplied by each of these monomials in forming the linear combination which is the polynomial. The UQTools multivariable polynomial software has two ways to specify which monomials are being used. The first specification paradigm, *specification by order* uses two or three integer parameters. The

[2]This white paper is out of date in that it was written before the gradient and Hessian capabilities were added to the multi-variable polynomial software and much of the capability for accepting polynomials defined by "coefficients and exponents" was added.

dimension of the parameter space and the order of the polynomial are always specified. Optionally, a third parameter (the truncation parameter) may be specified to limit the number of cross terms (non-zero exponents) in a monomial. When the two-parameter version of this paradigm is used, the polynomial consists of all monomials of the given number of variables whose orders do not exceed the given order. If the truncation parameter is also specified, the monomials with more than the specified number of active variables are eliminated from the full set. The second paradigm for specifying a polynomial, *specification by powers*, is to simply provide a matrix of non-negative integers (variously called the exponents matrix or the powers matrix), each row of which is the vector of exponents, the $(i_1, \ldots, i_d)$ in the previously given definition of "monomial", used in one of the monomials.

For a single RS (simultaneous representation of multiple RSs will be dealt with later) the coefficients of the monomials are stored in a column vector. If the form of the polynomial is specified by powers, then the association between coefficients and monomials is simple: coefficient number i goes with the monomial defined by row i of the powers matrix. If the form of the polynomial is specified by order, the monomial associated with each entry in the coefficient vector is determined by an algorithm. This order is uniquely determined by its inputs that are the polynomial order, the number of variables, and, optionally, the truncation parameter. This algorithm is described in detail in the white paper `UQTOOLS/Documents/mv_poly_whitepaper.pdf`. The user can determine how many terms a given polynomial will have by using function `mv_poly_count_terms`. To find out exactly which monomials are used and in what order the algorithm assigns them, the user can call function `mv_poly_expansion` with the optional output variable `powers`.

UQTools provides several functions for use with multivariable polynomial RSs. These functions are described in Table 6.

Table 6: Functions for Multivariable Polynomial Response Surfaces

Function Name	Calculation performed
`mv_poly_coeff_fit`	Calculate the coefficients to fit a polynomial to given data. For the underdetermined case (fewer monomials in the polynomial than points of data to be fit), the coefficients are chosen to minimize the (weighted) sum-squared error between the data and the polynomial. Specifying weights is optional with the default being that all data points are equally weighted. For the overdetermined case (more monomials in the polynomial than data points), coefficients with a minimum sum of squares are chosen to fit the data exactly.
`mv_poly_eval`	Evaluate a multivariable polynomial RS at one or more points in the parameter space. Gradients and Hessians may also be evaluated.

Table 6: Functions for Multivariable Polynomial Response Surfaces (Continued)

`mv_poly_expansion`	Primarily for internal use of UQTools, this function evaluates all monomials of a polynomial specified by dimension, order, and truncation parameter at the input data points and (optionally) their gradients and Hessians, and may be used to calculate the powers matrix for a multivariable polynomial RS specified by order.
`mv_poly_expanpow`	Primarily for internal use of UQTools, this function evaluates all monomials of a polynomial specified by explicit list of monomial power vectors at the input data points and (optionally) their gradients and Hessians.
`mv_poly_count_terms`	Primarily for internal use of UQTools, this function calculates the number of monomials in a multivariable polynomial RS specified by order.
`mv_poly_momabtmean`	If the variables in a multivariable polynomial are considered to be independent uniformly distributed random variables, this function calculates the mean and moments about the mean up to whatever order is requested of the polynomial function of these random variables.
`mv_poly_intgrl`	Primarily for internal use of UQTools, this function integrates a multivariable polynomial over a hyper-rectangle defined by fixed lower and upper limits on each variable.
`mv_poly_mult`	Primarily for internal use of UQTools, this function returns the product polynomial specified by powers resulting from multiplying two input polynomials specified by powers.

If several functions are to be fit with RSs with otherwise identical parameters they may be evaluated in parallel at one or more points from the parameter space by a single call to `mv_poly_eval` using a matrix of coefficients[3].

For an example of the use of many of these functions see section 13.1, "File: `mv_poly_example.m`."

[3]To evaluate fewer than the complete set of RSs at a set of points from the parameter space, simply make a reduced coefficient matrix by extracting the relevant columns from the full coefficient matrix, and pass the reduced matrix to `mv_poly_eval`.

7.2 Radial Basis Function Response Surfaces

A *radial basis function* (RBF) is a real valued function (at least for the purposes of UQ-Tools) defined for a nonnegative real argument.

If ϕ is a specific RBF, then a RBF RS using ϕ is a function of the form:

$$r(p) = \sum_{i=1}^{n} c_i \phi(||p - p_i||)$$

In this expression, the p_i are fixed points (called the *centers* of the RBF RS) in the parameter space, the norm is Euclidean norm (so that $||p - p_i||$ is the radial distance from p top_i; hence the *radial* in "radial basis function"), and c_i is the coefficient which goes with the center p_i. Fitting a RBF RS to n data points $\{(p_i, f(p_i))|1 \leq i \leq n\}$ (here, we are assuming that the data comes from evaluating a function f defined on the space containing the p_i) requires solving for the coefficients c_i, $1 \leq i \leq n$, in the system of equations

$$\sum_{i=1}^{n} \phi(||p_j - p_i||)\, c_i = f(p_j),$$

where $j = 1, \ldots, n$. For the RBFs used in UQTools, as long as the centers are distinct, this system of equations has a unique solution, and the resulting RS exactly matches the target function at the centers.

It is in the nature of RBFs to treat the variables as if they are equally scaled. In a raw physics model, this might not be the case. For example, consider the relative scaling difference in a model where some variables represent masses and others represent damping ratios. For this reason, one might want to transform the variables to improve scaling before fitting a RBF RS. In dealing with uncertain parameters modeled as random variables, the previously mentioned transformations from P-space into U-space or V-space replace possibly unevenly scaled variables with variables having identical scaling.

RBF RSs can have some intrinsic limitations. For example, each term of a Gaussian RBF has a bump at its center and approaches zero as its argument grows indefinitely large. Thus, a Gaussian RBF RS might tend to be bumpy (this can be somewhat alleviated by choice of the Gaussian parameter; more about that later) and will approach zero as its argument grows large. This is not ideal if one is trying to approximate a function that is biased away from zero. To add flexibility to RBF RSs, UQTools permits polynomial preconditioning.

With polynomial preconditioning, the data is first approximated with a polynomial RS, say Q. Then an RBF RS $r(p)$ is fit to the residual data $\{(p_i, f(p_i) - Q(p_i))|1 \leq i \leq n\}$. The resultant RS is $r(p) + Q(p)$, and it again matches the target function at the centers.

The user should consider the following points when using polynomial preconditioning. The order of Q should be limited to any known polynomial-like trends in f. Absent any knowledge of the behavior of f, Q should be of small order. Orders zero (the constant polynomial) and one (the linear polynomial) should be safe. The potential danger of using a high order Q is that, in choosing coefficients to minimize the sum square error between the polynomial and the truth function at the data points, the polynomial will oscillate between data points, with overshoots and undershoots, so that it may achieve close approach at data points. The corrective RBF RS will need only small coefficients to remove remaining error between the polynomial and the truth function data points. The resultant RS will tend to follow the oscillations of the polynomial and may not be a reasonable global approximation to the truth function.

The data that specifies an RBF RS (with or without the optional polynomial preconditioning) is contained in a single MATLAB variable of data type *structure*. The user enters the following information into this structure:

- The name of the RBF.
- Any parameters used by the RBF.
- The points in parameter space used as centers.
- The value(s) of the truth function(s) that are to be interpolated at these centers.
- The order of the pre-fit polynomial(s) (with "-1" for "no pre-fit polynomial(s)").
- Optionally, a cross-term truncation parameter for the pre-fitting polynomial(s).

Details about this are included in the documentation file `UQTOOLS/RespSurf/RBF_RS_data_structure.txt`.

UQTools provides three functions for creation and evaluation of RBF RSs and eight functions that define the eight parametrized families of RBFs which it supports. The three creation and evaluation functions are given in Table 7.

Table 7: Functions for Radial Basis Response Surfaces

Function Name	Calculation performed
`rbf_coeff_fit`	Calculate the coefficients of the pre-fit polynomial(s) (if requested) and the coefficients of the RBF RS(s). Add this and other information to the RBF RS defining structure.
`rbf_coef_reduce`	Calculate the structure that results from removing some of the multiple response surfaces from the structure calculated by `rbf_coeff_fit`.
`rbf_eval`	Evaluate the RBF RS(s) (and, optionally, its (their) gradient(s) and Hessian(s)) at one or more points from the parameter space.

UQTools provides eight functions that define eight choices for the RBF. Of these, six may be used if gradients of the RSs are desired, and five of those may be used if Hessians are desired. In the following table, the "Differentiability" column contains the entry "G" if the function is everywhere differentiable, so may be used if gradients are to be calculated, and contains the entry "H" if the function is everywhere twice differentiable, so may be used if Hessians are to be calculated. The entry "-" is used to indicate that the function is not everywhere differentiable. See Table 8 for a description of these functions.

Table 8: Radial Basis Functions

Name	Differentiability	The Radial Basis Function
`cubic_rbf`	–	The cubic radial basis function: $$\varphi(r) = \left(\sqrt{r^2+c}\right)^3$$ This RBF uses a single real scalar parameter c that should be non-negative.
`gaussian_rbf`	GH	The Gaussian radial basis function: $$\phi(r) = e^{-cr^2}$$ This RBF uses a single real scalar parameter c that must be positive. Note that c is related to a Gaussian distribution standard deviation σ by the equation $c = 1/\sigma^2$.
`imq_rbf`	GH	The inverse multiquadratic radial basis function: $$\phi(r) = \frac{1}{\sqrt{r^2+c^2}}$$ This RBF uses a single real scalar parameter c that should be positive.
`linear_rbf`	–	The linear radial basis function: $$\phi(r) = cr$$ This RBF uses a single real scalar parameter c that should be non-zero.
`mq_rbf`	GH	The multiquadratic radial basis function: $$\phi(r) = \sqrt{r^2+c^2}$$ This RBF uses a single real scalar parameter c that should be positive.
`tps_rbf`	G	The Thin Plate Spline radial basis function: $$\phi(r) = r^2\log(cr^2)$$ This RBF uses a single real scalar parameter c that should be positive.

Table 8: Radial Basis Functions (Continued)

`compactI_rbf`	GH	The Compact-I radial basis function: $$\phi(r) = \begin{cases} \Psi(t), & \text{if} \quad t < 1, \\ 0, & \text{otherwise} \end{cases}$$ Here, $\Psi(t) = (1-t)^5(8+40t+48t^2+25t^3+5t^4)$ and $t = r/c$. This RBF uses a single real scalar parameter c that should be positive. Note that this $\phi(r) = 0$ for $r \geq c$.
`compactII_rbf`	GH	The Compact-II radial basis function: $$\phi(r) = \begin{cases} \Psi(t), & \text{if} \quad t < 1, \\ 0, & \text{otherwise} \end{cases}$$ Here, $\Psi(t) = (1-t)^6(6+36t+82t^2+72t^3+30t^4+5t^5)$ and $t = r/c$. This RBF uses a single real scalar parameter c that should be positive. Note that this $\phi(r) = 0$ for $r \geq c$.

8 Bounding the Safe/Failure Space Using Homothetic Deformations

8.1 Using the Exact Requirements Functions

The UQTools functions discussed in this section are kept in UQTools subdirectory `UQTOOLS/Homodef/Core`. The theory behind this analysis technique is described in research papers by the authors [1–3] and are included in the files in UQTools subdirectory `UQTOOLS/Homodef/Papers`.

8.1.1 What Is It, How Is It Done?

UQTools provides software to analyze the performance of a system that depends on the uncertain parameter vector p. The performance is considered acceptable when a set of system requirements is satisfied. These requirements, which also depend on the uncertain parameter p, are described by the set of inequalities $g(p) < 0$, where g is a (possibly) vector-valued function of p. The function $g_i(p)$ will be referred to as a *requirement function*. This vector inequality, $g(p) < 0$, must hold component-wise, so system acceptability requires the satisfaction of all requirements. For instance, if the system is a structure and p represents material properties, one of the component inequalities, $g_i(p) \geq 0$, can describe the uncertainty realizations yielding a plastic deformation. If the system is a controlled aircraft and p represents some aerodynamic coefficients, $g_i(p) \geq 0$ can represent the uncertainty realizations for which the closed-loop system is unstable.

Note that each realization of the parameter vector p will either fall into the constraint satisfaction set $S = \{p : g(p) < 0\}$, also called the *Safe Domain*; or into its complement, the constraint violation set $F = \{p : g_i(p) \geq 0 \text{ for some } i\}$, also called the *Failure Domain*. It follows that $F = C(S)$, where $C(\cdot)$ is the *set complement* operator. Determining whether or not the system performs acceptably requires knowing the value of p. Since this value is uncertain, we need to characterize the dependency of g on p. If the expected range of variation of p is a subset of the safe domain, the system will be sufficiently robust to uncertainty; i.e., the requirements will be satisfied for all possible uncertain parameter realizations within such a range. Otherwise, a portion of this range will fall into the failure domain. Determining whether a system is sufficiently robust can be difficult for several reasons, for example:

- p can have a large dimension.
- g can be expensive to evaluate.
- The dependence of g on p can be so complex as to preclude determining the Failure Domain analytically.

The idea supporting the developments that follow results from trying to find the separation between a particular parameter realization, $\bar{p}$, which we will call the *Nominal Parameter Point,* and whichever of the failure or safe domain does not contain $\bar{p}$. One choice of the point $\bar{p}$ could be our best guess of the actual value of p. The separation between the point $\bar{p}$ and the set F or S will be determined by homothetically adjusting the size of a *Reference Set* Ω [1–3]. The reference set can be a hyper-rectangle or hyper-sphere centered about $\bar{p}$ which is serving as the *homothetic center*. Homothetic deformation of the reference set refers to expanding or contracting it about its center point by a fixed factor, the *similitude ratio*, in all directions while keeping the center point and the orientation fixed [7]. Ω is deformed homothetically until it is as large as possible while still being fully contained in the same domain as $\bar{p}$. The deformed set will be called the *Maximal Homothet M*. The terminology *Maximal Safe Set* will be used if $M \subset S$. The parameter point(s) where the maximal set touches the boundary of the failure domain are the *Critical Parameter Values* (CPV), which are denoted as $\widetilde{p}$. These parameter realizations can be interpreted as worst-case uncertainty combinations. This case is significant since, for $\bar{p} \in S$, the larger the maximal safe set, the further the failure domain is from $\bar{p}$, and; therefore, the more robust the system is to uncertainty in p. The reference set can have hyper-spherical or hyper-rectangular shapes in either P-space or U-space while having $\bar{p}$, or its transformation to U-space, as its geometric center. Details of the mathematical formulation of this process are given in [1] and [2].

Figure 3 shows a sketch of a two-dimensional P-space, a hyper-rectangular reference set, and the corresponding maximal set for a system subject to two requirements. The gray portion, labeled F, is the failure set. For $p \in F$, at least one of the inequalities $g_1(p) \geq 0$ or $g_2(p) \geq 0$ holds.

References [1–3] propose two metrics to quantify the size of the maximal set. These metrics, called the *Parametric Safety Margin* (PSM) and the *Reliability Index* (RI), are quantifiers of robustness. PSM arises from reference set deformations in P-space, while RI results from reference set deformations in U-space. To distinguish between the cases where $\bar{p} \in S$ or $\bar{p} \in F$, UQTools software returns the PSM or RI as a positive number in the former case and as a negative number in the latter.

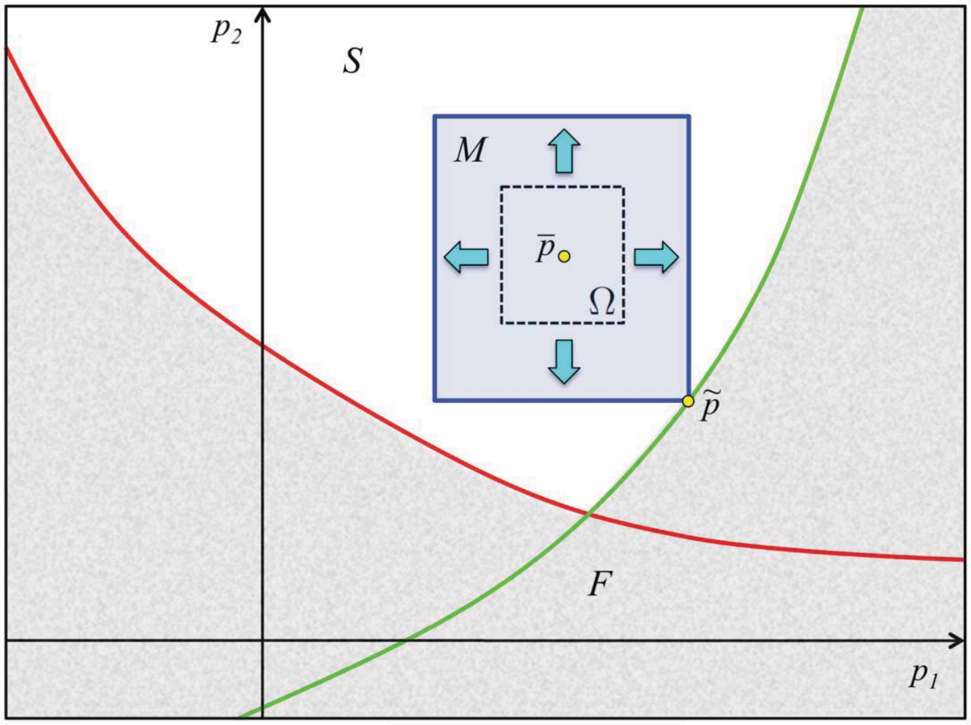


Figure 3. Maximal safe set of a hyper-rectangular reference set.

Note that the notion of robustness evaluated via failure probabilities differs considerably from the ideas represented by these metrics; e.g., there can be situations where the failure probability is small and $\bar{p}$ is in the failure domain.

8.1.2 Utility of Bounding Sets

Unless specifically stated otherwise, further discussion refers to the case that $\bar{p}$ is in the Safe Domain, so the maximal homothet M of the reference set is a subset of the Safe Domain. Once M has been determined, several tasks can be performed.

- *Robustness metric*: Any metric evaluating the size of the maximal set is a quantifier of robustness. The PSM and the RI are two of these metrics [3]. The larger their values are, the larger the separation between the nominal parameter point and the failure domain.
- *Failure probability bound*. As mentioned previously, if $C[\cdot]$ and $\mathrm{P}[\cdot]$ denote the complement set and probability operators, then note that $\mathrm{P}[C(M)] = 1 - \mathrm{P}[M] \geq \mathrm{P}[F]$. Since the geometry of M enables the exact calculation of $\mathrm{P}[M]$, the upper bound to the failure probability P[C(M)] can be readily calculated as 1-P[M]. Note that the calculation of $C(M)$ is independent of the probabilistic uncertainty model. As a result, the calculation of upper bounds for several probabilistic uncertainty models (i.e., the application of $\mathrm{P}[g]$ to $C(M)$) can be done with minimal computational effort.

- *Failure probability via sampling*. Since $g(p) < 0$ for all points in the maximal safe set, one can avoid performing function evaluations at such points when estimating $\mathrm{P}[F]$ via sampling. This is the basic idea behind the hybrid methods for estimating failure probability (see section 11). Rejection-based and conditional sampling-based variants of this algorithm are available in UQTools. Details of these methods are provided in sections 11.1 and 11.2.

8.1.3 How To Use `homodef`

In order to perform a homothetic deformation, the reference set Ω and the set of requirements $g(p) < 0$ must be prescribed. Besides these two inputs, UQTools requires setting up parameters that affect the numerical search for the CPV(s). The UQTools function used to perform homothetic deformations is called `homodef`. The CPV, the maximal set, and the PSM or RI are some of the outputs of this function. Details on the data structure supporting the inputs and outputs of this function can be found in its program preamble comments.

The example file named `example_homodef.m` (section 13.2) exercises representative input combinations of `homodef` in detail. In particular, we consider problems (1) of varying number of dimensions, (2) of varying number of requirements, (3) where the reference sets are either hyper-spherical or hyper-rectangular in either P-space or U-space; and in some cases, (4) where the dependence of g on p allows for the analytical calculation of the maximal set. Plots of relevant sets in the parameter space back these analyses. The reader is strongly encouraged to study and execute all 17 problems in `example_homodef.m` (section 13.2) to get familiar with the notions, concepts, and data structures supporting `homodef`.

A high level description of the ***main*** inputs and output of `homodef` is presented next. More details can be found in the program preamble comments of file homodef.m in directory `UQTOOLS/Homodef/Core`. UQTools permits the following syntax:

```
results = homodef(req,prob,numsetup,descriptor,sta_ic_p,write,dosampling)
```

1. The set of requirements functions (symbol in document g, variable name `req`)

   The variable `req` is a MATLAB vector of data type *structure* with as many elements as the number of requirements functions. The following information must be supplied for the i^{th} requirements function:

   - `req(i).g` the name of the function that calculates $g(p_i, \cdots)$.
   - `req(i).name` a character string giving the name of the requirement. This should be unique to this requirements function since UQTools uses it to identify this function to the user. Otherwise, it does not enter into the calculation.
   - p (`req(i).extrainputs` a cell array specifying any inputs required by function g_i other than those required. Should be set to the empty cell array, $\{\}$, if no additional inputs are needed.
   - `req(i).tolxcf` a vector containing the tolerances [`TolX, TolCon, TolFun`] to be used when the MATLAB Optimization Toolbox function `fmincon` calculates the maximal safe set of Ω for g_i. `req(i).tolxcf` may be set to the empty vector, [], to use default values.

Note that each function named in `req(i).g` must be created by the user. The first input of this function must be the value of the uncertain parameter vector *p*, and additional inputs, if needed, should be formulated to use the information the user will put in `req(i).extrainputs`. Thus, if `req(i).g='myfun'`, the user should have a function M-file named `myfun.m` which calculates requirements function i and whose first line looks like:

```
function y = myfun(p, C1,...,Cn)
```

The (possibly empty) list, `C1,...,Cn`, of additional parameters should be furnished as a cell array in the `extrainputs` field by, for example, executing a command of the form:

```
req(i).extrainputs = {C1,...,Cn};
```

2. The reference set (symbol in document Ω, variable name `prob` and `descriptor`).

   UQTools allows for using reference sets in P-space or U-space that have either hyper-spherical or hyper-rectangular geometries. The variable prescribing which of these four combinations holds is named `descriptor`. The data required in `prob` by each of these combinations is specified next. In all four cases, the variable in `prob.set` must be created via function `setrvs` or using `uqtools_gui` (this is the vector of structures called `rv` in section 4, "Representation of Random Variables"). Additional details on the data structure of the variable `prob` are given in the `def_prob*.m` files in subdirectory `UQTOOLS/Examples/General`.

   Note that, if a maximal homothet of the reference set is sought in P-space, then the probabilistic structure of the uncertain variables will not be used by `homodef`. In contrast, to determine a maximal homothet of the reference set in U-space, the probabilistic structure is necessary to define the transformation between P-space and U-space. Thus, the user-provided information in structure field `prob.set` serves different functions in these two cases. For P-space problems, `prob.set(i)` sets limits on how widely the uncertain parameter number *i* is allowed to range in seeking the maximal homothet. This can be done by defining `prob.set(i)` using `setrvs` with argument `typedist` set to 'interval' and argument `paramv` set to a two element vector defining the uncertainty range of interest for the components of `prob.set`. For U-space problems, `prob.set` is set to the actual `rv` structure (see section 4) which describes the probabilistic nature of the uncertain variables.

   `descriptor='p-sphere'` (see problems 2, 5, and 7 in `example_homodef.m`, section 13.2): In this case we have to prescribe the center of the sphere in P-space (`prob.nom`) and the uncertainty range limits of interest (`prob.set`). This range confines the numerical search for the CPV to a hyper-rectangular set in P-space. The realization of infeasible or unreasonable values of the uncertainty can be prevented using such a range. For this descriptor, the robustness metric calculated by `homodef` is the *Spherical Parametric Safety Margin* (SPSM).

   `descriptor='p-rectangle'` (see problems 1, 3, 4, and 6 in `example_homodef.m`, section 13.2): in this case we have to prescribe the center of the rectangle in P-space (`prob.nom`), the half lengths of the deforming directions (`prob.m`), the half lengths of the fixed directions (`prob.c`), and the uncertainty range of interest (`prob.set`). The next section explains the significance of having both deforming and fixed directions. For this descriptor, the robustness metric calculated by `homodef` is the *Rectangular Parametric Safety Margin* (RPSM).

`descriptor='u-sphere'` (see problems 9, 11, 1, 15, and 17 in `example_homodef.m`, section 13.2): In this case we only have to prescribe the probabilistic uncertainty model (`prob.set`). In all cases, the center of the sphere is assumed to be the origin of U-space. Even though maximal sets with other origins can be calculated, the inability to calculate their probabilities analytically precludes most of their usefulness. Note that the support set of the random vector in `prob.set` prescribes the uncertainty range of interest. For this descriptor, the robustness metric calculated by `homodef` is the *Spherical Reliability Index* (SRI).

`descriptor='u-rectangle'` (see problems 8, 10, 12, 14, and 16 in `example_homodef.m`, section 13.2): in this case we have to prescribe the center of the rectangle in U-space (`prob.nom`), the half lengths of the deforming directions (`prob.m`), and the half lengths of the fixed directions (`prob.c`). As before, the support set of the random vector in `prob.set` prescribes the uncertainty range of interest. For this descriptor, the robustness metric calculated by `homodef` is the *Rectangular Reliability Index* (RRI).

3. Numerical setup (variable name `numsetup`)

   The input variable `numsetup` is a MATLAB scalar variable of data type structure that prescribes the numerical setting used to perform optimization and sampling. The fields `sea`, `sam`, and `opt` refer to the searching, sampling, and optimization setups respectively. Details on the specific sub fields can be found in the `def_prob*.m` files in subdirectory `UQTOOLS/Examples/General`. The information in the field `sea` is used to overcome the possibility of not converging to the true CPV. In such a case a sequence of serial and parallel searches for the global optimum are carried out. Information in the field `sam` is used to evaluate convergence to the global optimum by sampling the surface and volume of the maximal set. This test will only be conclusive if one of such samples falls into the failure domain. The information in the field `opt` is used to prescribe the use of simplex and/or gradient-based algorithms.

4. Maximal Set (variable name `results`)

   The output of `homodef`, which is called `results` in the internal documentation in file `homodef.m`, is a MATLAB scalar or vector variable of data type *structure* depending on whether $\bar{p}$ is in the failure or safe domain, respectively. If $\bar{p}$ is in the safe domain, `results` has the same number of components as the vector of requirements functions. Each component of `results` contains information computed during the determination of the maximal safe set of Ω for one of the requirements functions, g. However, the components of `results` are not necessarily in the same order as the components of g. The components have been sorted so that the one leading to the smallest homothet, i.e., the maximal safe set for the fully constrained problem, is given first. The succeeding components of results give the maximal safe sets for the other individual requirements functions in order of increasing size. If $\bar{p}$ is in the failure domain, `results` has only a single component which describes the maximal homothet of Ω which is contained in the failure domain.

The fields of `results` give information about the maximal set being described. The field `results(i).infdss` is +1 when the nominal parameter point is in the safe domain, otherwise it is -1. The fields `results(i).robmets` and `results(i).cpvs_p` contain the metric of the maximal set (i.e., PSMs or RIs) and a CPV in P-space corresponding to each requirement. Note that the sign of `infdss` and that of `robmets` are the same. The field `results(i).name_met` is set to one of the strings SPSM, RPSM, SRI, or RRI depending on the input value of `descriptor`. Several additional fields of results are described in the program preamble comments to file `homodef.m`. The UQTools function `sample_ms.m` is used in `example_homodef.m` (section 13.2) to test whether the search for the CPV has converged to its true value.

8.1.4 Extensions

UQTools enables performing homothetic deformations of hyper-rectangular reference sets in which some directions are deformed while other ones are kept fixed. Figure 4 (below) illustrates the case where the hyper-rectangular set Ω is only deformed in the p_1 direction. This class of deformation is feasible if and only if the homothet of smallest size is fully contained in the safe domain, e.g., in the sketch, this implies that a vertical line centered at $\bar{p}$ having a length equal to $2c$, would be in the safe domain. The failure domain, labeled F, is as in Figure 3.

In order to set up this class of deformation, it is required to prescribe (i) the vector of half lengths of the reference set for those directions being deformed (i.e., m in [1]), and (ii) the vector of half lengths of the reference set for those directions not being deformed (i.e., the c vector). A UQTools example of this class of deformations is available in `example_homodef.m` with `problem=4` (section 13.2).

8.2 Homothetic Deformations over Surrogate Models

The UQTools functions discussed in this section are kept in UQTools subdirectory `UQTOOLS/Homodef/RSapprox`. The user interface functions in this subdirectory are in function M-files `cpv_by_RSapprox.m` and `cbRrestart.m`. As in `homodef`, these routines seek to calculate a robustness metric. Unlike `homodef`, these routines are only programmed to deal with the case that the given nominal point is in the safe domain.

These routines differ from `homodef` in that the exact requirements functions are replaced by RBF RS approximations. These RS approximations are adaptively improved by iterating the robustness metric calculation with new RS data based on previous iterations and sampling. The purpose of this iterative improvement is to make the CPVs of the RS approximations to the requirements functions converge to the CPVs of the actual requirements functions. The expectation is that using `cpv_by_RSapprox` will require fewer evaluations of the requirements functions than using `homodef`, thereby saving computing time in cases where the evaluation of the requirements functions is computationally burdensome.

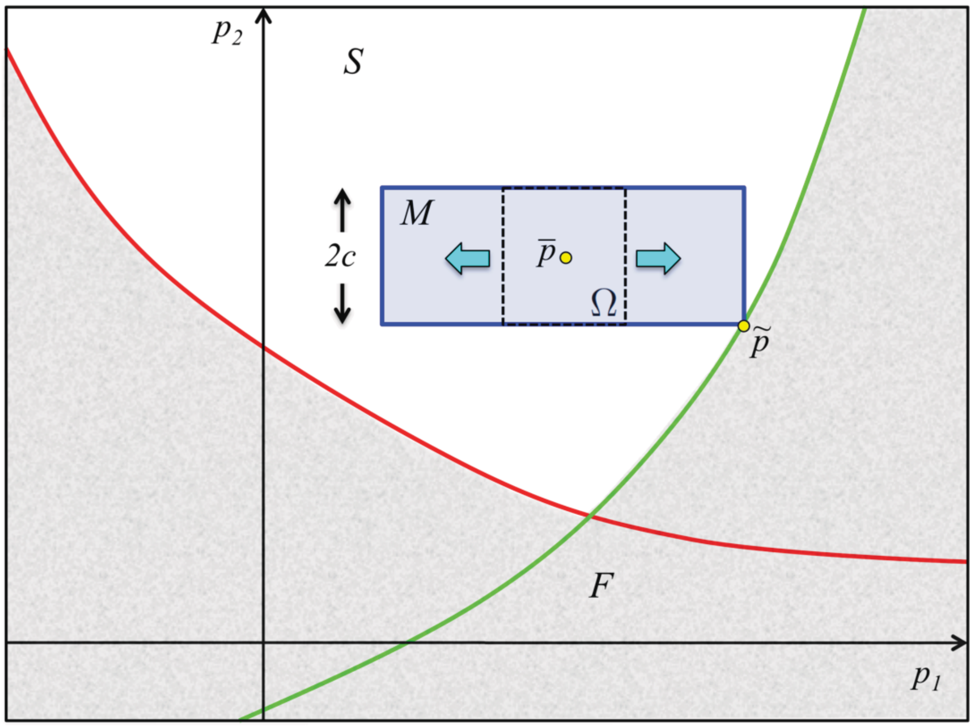


Figure 4. Hold some variables fixed, adjust others to find maximal safe set.

In order to master the operation of `cpv_by_RSapprox` , the user must first understand several other parts of the UQTools package:

- The input variables to `homodef`, as described in section 8.1, "Using the Exact Requirements Functions."
- The use of sampling routine `haltonget`, as described in section 6.2.3, "Halton Sequence Leaped, II."
- The RBF RS paradigm described in section 7.2, "Radial Basis Function Response Surfaces."

Function `cpv_by_RSapprox` functions in two modes. One is to initialize the process and perform iterations until it converges or a preset iteration limit is reached. The other is to restart a previous calculation, performing additional iterations. Such a restart requires modifying the output `cpvapxout` from the previous iteration and using it for the input parameter `cpvapxin` in the restarted run, typically accomplished by a call to the restart function `cbRrestart`.

The function `cpv_by_RSapprox` is called by a MATLAB statement of the form:

```
    [results, cpvapxout, conv_info] = cpv_by_RSapprox( cpvapxin, ...
req, prob, numsetup, descriptor, ic_p, write, sample, formp);
```

The first input, cpvapxin, is a MATLAB scalar variable of type "structure". It contains a great deal of information needed to start (or restart) the iteration. The form of the RBF RSs is specified, convergence parameters are specified, and an iteration count limit is specified. Since both the initial set of RBF centers and some of the centers added during each additional iteration are generated by the quasi-random number generator haltonget, cpvapxin also has fields to control quasi-random number generation. The RSs may have their domains in P-space or in one of the transformed spaces U-space or V-space. This may require information in addition to that provided by variable prob. See program preamble comments for field .rsdomain of input structure variable cpvapxin for more information on evaluation domains. Additional information for restarting cpv_by_RSapprox to perform additional iterations comes from the output variable cpvapxout from the previous iteration.

The next seven input variables are exactly the input variables that would be used in a call to homodef. The last input variable formp is optional. The variable ic_p is ignored by cpv_by_RSapprox, which calculates its own initial values for the optimizations that determine the maximal homothet(s), and is included only for calling sequence compatibility with homodef.

Recall that, if descriptor is p-rectangle, homodef requires that every prob.set(*i*).typedef be set to interval; while , if descriptor is p-sphere, homodef does not use prob.set. However, cpv_by_RSapprox requires probabilistic descriptions of the uncertain parameters to effect the transformations between P-space and V-space. This information is provided in cpvapxin.rv.

As in homodef, these routines find a robustness metric, i.e., SPSM, RPSM, SRI or RRI, according to the value in descriptor for an approximation to the problem defined by req and prob by finding the CPV for the maximal homothetic deformation of a hyper-spherical or hyper-rectangular reference set in P-space or U-space.

8.2.1 Brief Sketch of the Method

A brief sketch of the method is given here, but again; please see the program preamble comments for a complete description.

First, some sample points are generated in V-space and transformed into P-space. The requirements functions are evaluated at these points. This gives the data for the first surrogate requirements functions RSs. Then function homodef is used to find a CPV for each of the surrogate requirements functions RSs.

Then, the RSs are refined by using new data. The surrogate CPVs just found and the values of the actual requirements functions are added to the interpolation data. Also, some more quasi-randomly generated sample points are added, using the quasi-Latin hyper-cube technique. Surrogate CPVs are calculated for this new set.

With two sets of data, convergence testing can occur. If the process is unconverged, and the iteration limit has not been reached, another iteration just like the last one is taken. If the process is converged, the surrogate CPVs are taken as approximations of the true CPVs, and an approximate maximal safe set is calculated.

8.2.2 Convergence Testing and Algorithm Termination

Convergence is declared if the CPVs and robustness metrics for all active requirements functions, and the RS approximation error terms for all requirements functions pass specific convergence tests.

A call to `cpv_by_RSapprox` terminates for one of 3 reasons: an error has been detected, the iteration limit has been reached, or convergence has been detected. See internal documentation for more information.

8.2.3 Restarting **`cpv_by_RSapprox`** To Perform Additional Iterations:

After making a calculation with `cpv_by_RSapprox`, a user might find that the approximate answer is not sufficiently close to the desired answer. Common restart scenarios involve needing to increase the iteration limit and/or altering the convergence parameters. As an alternative to starting the calculation from scratch using `cpv_by_RSapprox`, UQTools provides a utility to allow the user to accomplish restart easily in these cases. This is possible because the output variable `cpvapxout` has accumulated information about the iterations already performed.

The function to drive a restart is in file `UQTOOLS/Homodef/Rsapprox/cbRrestart.m`. The program preamble comments in this file provide details about how to call it. The most important thing to note is that the input variable `cpvapxin` to `cbRrestart` is the output variable `cpvapxout` from the previous run of `cpv_by_RSapprox`.

9 First Order Reliability Method

Discussed in this section are the UQTools functions for applying the First Order Reliability Method (FORM) to systems with probabilistic parameter uncertainty are kept in UQTools subdirectory `UQTOOLS/RelMeth`.

FORM and SORM[4] are techniques that have, for several decades, found application in reliability analysis of structural systems. These are techniques for approximating the probability of failure of a system with probabilistic parameter uncertainties. They are both most likely to give reasonable approximations of failure probability when the following are true:

- Failure probability is small.
- Failure region is in the tail of the uncertain parameter distribution function.
- Most of the failure probability is attributable to just one of the requirements functions.

In the UQTools implementation, the failure region is defined by a single requirement function that has, historically in this context, been called a *limit state function*.

The starting point for FORM and SORM is the transformation of the uncertain parameter space into standard normal space, a.k.a. U-space. Then, the so-called Most Probable

[4]An overview of FORM and the Second Order Reliability Method (SORM) can be found in the second section of [6]. This reference also gives one more example of the versatility of the techniques included in UQTools.

Point (MPP) of failure is calculated. This is the point in the U-space transform of the failure region that is closest to the origin, which is the expected value of the standard normal distribution. The Euclidean norm of the MPP, frequently represented by the symbol β, is called the reliability index (RI). Because the tail of the standard normal distribution function experiences exponential decay, the motivation for examining this point is that it is hoped that the system failure probability is concentrated at the MPP, as it could be if the failure region indeed had small probability and lay in the tail of the uncertain parameter distribution function. FORM then estimates the failure probability by approximating the failure region by the half-space resulting from placing a hyper-plane through the MPP tangent to the failure region (hence *first* order) and calculating the probability of the half-space opposite the origin (this assumes that the origin is in the safe domain). See Figure 5 for a representative graphic. This probability is simply $\Phi(-\beta)$, where Φ is the CDF of the (one dimensional) standard normal distribution [8, equation (2.19), p. 12]. SORM replaces the hyper-plane by a quadratic surface (hence *second* order) that also matches curvature of the failure boundary at the MPP. Calculating, or even estimating, the probability of a region in U-space bounded by a quadratic surface is much more difficult than the linear case.

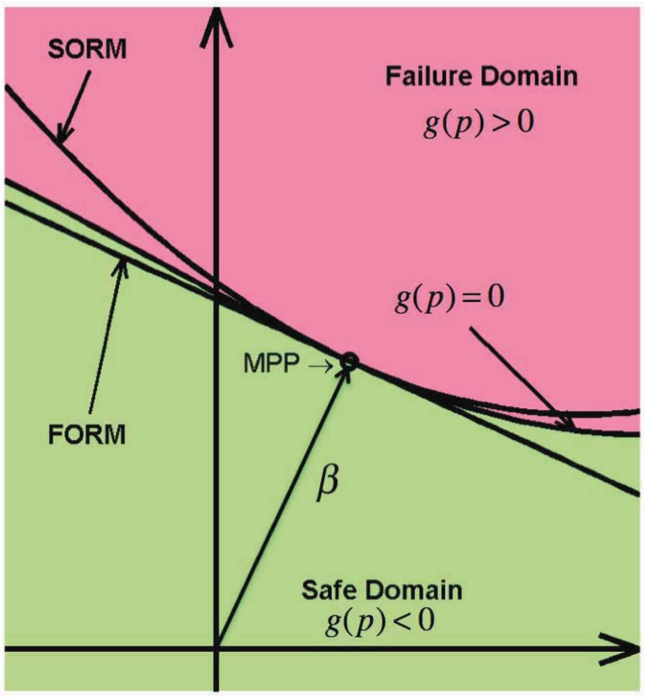


Figure 5. FORM and SORM

There is a connection between the present FORM analysis and the analysis done in section 8.1. If the problem considered in section 8.1 has a single requirements function and function `homodef` is used to find a maximal safe hyper-sphere in U-space, then the CPV there is the same as the MPP from FORM and the SRI there is the same as the RI from FORM.

UQTools provides FORM analysis by utilizing a software package called MatPA (included in the directory UQTOOLS/RelMeth). The user represents data using the homodef standard input variables req, prob, numsetup, and descriptor and passes these to the interface routine, call_MPP, which will translate the data into input format for the MatPA function MPP_FORM and then call_MPP. If all the uncertain parameters are declared to be normal random variables in field .set of input structure variable prob, correlations may be imposed on these parameters by inputting a correlation coefficient matrix in optional call_MPP parameter called Covariance. As an additional optional parameter, the user may declare in the sixth input parameter position the initial value, X0, for the optimization search for the MPP. If this is done, the position for optional input parameter Covariance must be filled, if only by the empty matrix, []. If no user-supplied initial value is given, call_MPP supplies a default. UQTools permits the following syntax:

```
    [mppU,beta,G0,Pf,Ps,mppX,optout,results,dist,stru] = ...
call_MPP(req,prob,numsetup,descriptor,Covariance,X0);
```

If multiple requirements functions are defined, the requirements functions are passed to MPP_FORM one at a time and the MPPs and FORM approximations to failure probability are calculated individually for each of these limit state functions.

The program preamble comments in file MPP_FORM.m also define its output variables. The interface function call_MPP has the same outputs as MPP_FORM except that, for multiple requirements functions, scalar outputs are promoted to column vectors and vector outputs are promoted to matrices where, for both vectors and matrices, row *i* corresponds to MPP and FORM analysis results using requirements function number *i* as the limit state function. Function call_MPP has one additional optional output variable called results, see section 8.1.3 for a detailed description. This contains those fields of the homodef output variable of the same name that are needed by the function sample_ms and used to test whether the optimum found is a global optimum (see section 10).

The UQTools function M-file MPP_FORM.m calculates both the MPP and the FORM approximation to the failure probability. The MPP calculation is accomplished by a nonlinear constrained optimization. An initial value for the optimization, called X0 in the program preamble documentation of MPP_FORM, is provided by the user or by call_MPP. The optimization actually takes place in U-space. MPP_FORM converts X0 internally to the corresponding U-space value to serve as an initial value in the optimization.

MatPA was written using a different representation of parameter uncertainty than that which has become the UQTools standard. The UQTools user can call MPP_FORM directly by constructing the necessary input as defined in the program preamble comments in file MPP_FORM.m and the document UQTOOLS/RelMeth/DATA_dictionary.txt. One advantage to this approach is that the parameter uncertainty representation protocol used by MatPA allows for more generality in the uncertain parameter distribution than the UQTools protocol. Besides allowing individual parameters to be modeled as random variables of arbitrary distribution that are independent of the rest, blocks of parameters may be declared to have correlated normal or correlated lognormal distributions with the understanding that the parameters within such a block are correlated with each other, but independent of all parameters outside the block.

Examples using MPP_FORM.m are found in directory UQTOOLS/Examples/MPP_based.

10 Risks and Potential Drawbacks

10.1 The Problem

Both the determination of maximal (safe or failure) sets and the calculation of the MPP in FORM calculation make use of optimization of some objective function subject to non-linear constraints. The theory on which they are based assumes that a global optimum is found. Practitioners of numerical optimization are well aware that numerical optimization software such as is used in UQTools may declare convergence when a local optimum has been located rather than a global optimum. This has unfortunate consequences for maximal set and FORM calculations. In a maximal set calculation, if the declared CPV is a result of finding a local, but non-global optimum, then the declared maximal set is actually too large. In a FORM calculation the result is not, in fact, the *most* probable point of failure, i.e., a point in the U-space failure domain at which the PDF of the standard normal distribution in U-space is maximal. A consequence of this is that the estimated probability of failure using the declared MPP is smaller than would have been estimated if the true MPP had been calculated.

10.2 A Test for Non-globality of Solutions

UQTools provides function `sample_ms` in directory `UQTOOLS/Homodef/Core` that is used to test numerically if the declared maximal set is fully contained in the same domain as the nominal point. Points sampled from within the declared maximal set and from its boundary are tested by evaluating the constraint functions at those points. If any of those points is found to be in the opposite domain from the nominal point, `sample_ms` returns indication that the declared maximal set is too large, so must have been the result of the optimization problem converging to a non-global optimum. Otherwise, it indicates that all samples are in the domain they are supposed to be in. An example of the use of this function can be found in file `example_homodef.m` (see section 13.2). Another example is in file `example_call_MPP.m` (see section 13.16). This routine allows the user to check whether a FORM answer is faulty due to an optimization converging to a local, but non-global solution. In this example, `sample_ms` is used to check whether the MPP returned by `call_MPP.m` is correct by testing points within an RI radius of the origin of U-space to see if there are any constraint violations. In a contrived example with a deliberately chosen bad starting point, the first "MPP" is found to be faulty. Information from `sample_ms` is used to find a new starting point for `call_MPP.m` that results in a properly converged answer.

11 Estimating Failure Probability By Sampling

The UQTools functions discussed in this section are kept in UQTools subdirectory `UQTOOLS/Sampling`. Details of function usage are found in the program preamble comments.

Standard Monte Carlo methods for approximating failure probability depend on analyzing the system at a representative sampling of the whole parameter space. The present section provides alternatives that require the generation of fewer sample points and/or the evaluation of the system at fewer sample points. These alternatives are considered *hybrid methods* in that they combine sampling methods with the techniques of section 8.1 to generate maximal safe sets of rectangles or spheres. Either a full space sample set is generated or sample points are generated conditional on falling outside the maximal safe set. Even if a full space sample is generated, it is a relatively cheap computation to determine if a given sample point is inside the maximal safe set. Then, to estimate failure probability, it is only necessary to perform full system analyses at sample points outside the maximal safe set.

11.1 Rejection Method

If the uncertainty in the parameters is represented by an independent set of random variables, a whole space sample can be generated by generating each coordinate independently as a one-dimensional random sampling. This can be done in UQTools by using such functions as `mc`, `hhs`, or `has`, or by generating a uniformly distributed sample in the unit cube or a sample with a multivariate standard normal distribution and then mapping it to parameter space using the UQTools functions `v2p` or `u2p` respectively. In the rejection method of failure probability estimation, one simply examines each sample point for membership in the safe hyper-rectangle or hyper-sphere. System analysis is only done at sample points which fall outside these safe sets; i.e., points inside the maximal safe set are rejected for full system analysis. This reduces the number of system analyses that must be done by comparison with traditional Monte Carlo sampling to achieve the same confidence in estimating the failure probability. Since system analysis is typically the computationally intensive part of the operation, this can provide substantial savings in computer time.

11.2 Conditional Sampling

Now suppose that the probability of a sample point falling inside the maximal safe set is nearly equal to 1, so the failure event has small probability. It would be good if the sample generation process generated only representatively distributed samples that fall outside the safe sets. This is easier said than done. The conditional distribution of the uncertain parameters conditioned on the requirement that the samples fall outside the hyper-rectangle or hyper-sphere is not a distribution of independent random variables, and this lack of independence must be compensated for in the sampling process. The algorithms implemented in this conditional sampling software are documented in [4].

There are two cases in which UQTools software can accomplish this conditional sampling. The two cases are sampling outside of a hyper-rectangle and outside of a hyper-sphere, and are described below.

1. **Hyper-rectangle**: The random vector to be sampled is comprised of components that are independent random variables. The samples are to be generated conditionally on falling outside a given hyper-rectangle that, as required, is oriented with its axes parallel to the coordinate axes. The uncertainty in the random vector is modeled using an "`rv`" structure as described in section 4. For estimating failure probability, the authors suggest using a maximal safe set of a hyper-rectangular reference set such as is calculated by the software of section 8. The UQTools function that performs this sampling is called `hrcomp_condsamp` and is described in section 11.2.1.

2. **Hyper-sphere**: The random vector to be sampled is comprised of components which are independent random variables which all have the standard normal distribution. The samples are to be generated conditionally on falling outside a given hyper-sphere that is centered at the origin. Here, one could use the maximal safe hyper-sphere in the U-space transform of the uncertain parameter space such as is calculated by the software of section 8. One could also use an origin centered sphere in U-space whose radius is the RI calculated by FORM as in section 9. The UQTools function that performs this sampling is called `hscomp_condsamp` and is described in section 11.2.2.

These sampling techniques are examples of what is called "importance sampling". In importance sampling, the samples are generated using a different distribution from that of the uncertain parameters, usually chosen to generate more samples in some region of interest. A correction factor is applied to the information computed from these samples to compensate for using the "wrong" distribution. Here, the region of interest is the complement of the known safe set. The distribution used here for sampling has a PDF which is zero inside the known safe set and is the necessary multiple of the original PDF outside the safe set in order for it to be a proper PDF. The correction factor is given in the conditional probability formula of section 11.2.1.

11.2.1 Conditional Sampling Outside a Hyper-rectangle

If the safe set is a hyper-rectangle (with sides parallel to the coordinate hyper-planes), then the conditional sampling is accomplished by the UQTools function `hrcomp_condsamp` from subdirectory `UQTOOLS/Sampling`. As usual, the complete description of the function inputs and outputs is in the program preamble comments.

This function has four inputs. The first is a vector cell array of structures. The structure in the j$^{\text{th}}$ cell provides the necessary information about the distribution of the j$^{\text{th}}$ uncertain parameter. It can be generated from information provided by UQTools function `setrvs` or by the graphical user interface driven by UQTools function `uqtools_gui`. This is illustrated in the test script `UQTOOLS/Example/General/test_hrcomp_condsamp.m` that also provides a tutorial and template for the use of `hrcomp_condsamp`.

The second input is a matrix of rows from the unit hyper-cube. It is used to seed the process of generating samples outside the safe set. For the final sample set to be representative of the uncertain parameters, this seed set must be uniformly distributed over the unit hyper-cube. Generating such seed sets is explained in section 6. The third parameter defines the hyper-rectangle by giving its "lower left" and "upper right" corner. The last parameter is optional and gives the probabilities that each component of the uncertain parameter vector falls below or above the limits of its component of the hyper-rectangle. Mathematically, this information is redundant given the information in the first and third parameters. However, if the probability of the uncertain parameters falling into the safe hyper-rectangle is close enough to 1, there might be some numerical problems with computing these complementary probabilities directly from the CDF. Represent the CDF of component j by F_j, and let x_j represent component j of the "upper right" corner of the safe hyper-rectangle. The default calculation of the probability that component j of the uncertain parameter vector falls on the high side of the safe hyper-rectangle is expressed by the formula

$$1 - F_j(x_j).$$

If $F_j(x_j)$ is nearly equal to 1, finite precision computer arithmetic might introduce truncation error into this calculation. The user might be able to use problem dependent knowledge of the distributions (e.g., using symmetry of the normal distribution) to devise a numerically more robust method of calculating this probability. The optional last parameter provides a means for the user to pass this information to `hrcomp_condsamp`.

If $\mathscr{F}$ is the failure domain, H is the hyper-rectangle, and $C(H)$ is its complement, then the conditional probability formula states:

$$\mathrm{P}(\mathscr{F}) = \mathrm{P}(\mathscr{F} \,|\, C(H))\mathrm{P}(C(H))$$

The function `hrcomp_condsamp` returns the conditional sample with the same number of points as the seed matrix and the probability $\mathrm{P}(C(H))$ of the complement of the hyper-rectangle. $\mathrm{P}(\mathscr{F} \,|\, C(H))$ is then estimated by evaluating the system at the conditional sample points and calculating the fraction of them that fall in the failure set. The failure probability is then estimated using the conditional probability formula.

11.2.2 Conditional Sampling Outside a Hyper-sphere

The only case of conditional sampling outside a hyper-sphere for which UQTools has software is the case that the uncertain parameter space is standard normal space and the sphere is centered at the origin. This is not so limiting as it sounds, since UQTools software can be used to transform the problem to U-space and find a maximal safe hyper-sphere centered at the origin. This conditional sampling is accomplished by the UQTools function `hscomp_condsamp` from subdirectory `UQTOOLS/Sampling`. Inputs to this function are the radius of the hyper-sphere, the dimension of the parameter space, and the number of samples desired. This routine makes use of MATLAB random number generators. An optional fourth input allows the user to seed those generators. Using this option, the same "random" sample can be generated repeatedly. This might be useful in testing. The outputs here are the set of conditional samples and $P(C(H))$ where now H is the hyper-sphere. Failure probability is then estimated using the conditional probability formula in the same manner as with `hrcomp_condsamp`. A test script that also serves as a tutorial and template is found in file `UQTOOLS/Example/General/test_hscomp_condsamp.m`.

12 Estimating Probabilistic Sensitivities Via Sampling

Heretofore, we have provided tools for uncertainty quantification in which the user has been required to supply the description of uncertainty. Each uncertain parameter is described by modeling it as a random variable with a user-specified random distribution. This may require some conjecture on the part of the user to assign to each random distribution the values, mean, variance, support interval endpoints, distribution shaping exponents, etc., which are needed to specify it. The tools described in this section estimate how sensitive are the statistics of output variables, which are dependent on the uncertain parameters, to variation in the statistics of those parameters themselves. An example might be: how sensitive are failure probability estimates to the variances used to define the uncertain parameters? This can give the user information about which of the uncertain parameters have the most effect on the statistics of interest and which have the least effect.

The tools provided here are limited. Sensitivities are calculated only with respect to the expected values and variances of the uncertain parameters. Only normal or generalized beta distributions are considered. Since a normal distribution is determined by its mean value and variance, these can be varied directly to estimate the sensitivities of output statistics. Mean value and variance of a generalized beta distribution are derived from the four values that specify the distribution - two distribution-shaping exponents and the two endpoints of the support interval. For sensitivity approximation, perturbations are introduced into the mean value and variance of a generalized beta distribution either by perturbing the two distribution-shaping exponents or by perturbing the two support interval endpoints. These two techniques represent different sensitivities.

Sensitivities are estimated for certain statistics of random variables that are dependent on the uncertain parameters (through the system being analyzed). The statistics targeted here for sensitivity analysis are mean, variance, and the probability of the dependent variable falling above (or below) some specified value. This latter statistic is the model used for failure probability throughout UQTools.

Software described in this section is contained in UQTools directory `UQTOOLS/Probsensitive`.

12.1 What Is It, How Is It Done?

UQTools provides software that analyzes a system that depends on the uncertain parameter vector p. Metrics used by the analyst to evaluate the system's performance will depend on performance metrics(s) y that are functions of the uncertain parameter p. (Note that, here, performance metrics are called y instead of the g used earlier in this document. This is consistent with the notation in the paper [5] that presents the theory behind the calculations performed by this software.) The presentation that follows assumes that y is a scalar function. Extensions to vector functions can be easily made. The common model for the vector p of uncertain parameters is as a vector of independent random variables where its components assume normal or generalized beta distributions. These variables constitute the uncertainty model of p. Let us denote the PDF of the uncertainty model as $f_p(p,\theta)$, where θ is a vector of parameters. For instance, if all components of p are uncorrelated normal random variables, θ is a vector composed of the means $E[p_i]$ and standard deviations $\sqrt{\mathrm{V}[p_i]}$. Furthermore, let us define the vectors of means, m, and variances, v, of the components of p whose i^{th} components are m_i=$\mathrm{E}[p_i]$ and $v_i = \mathrm{V}[p_i]$.

The propagation of the uncertainty model of p through the performance metric $y(p)$ leads to a probabilistic description of y; i.e. y is a random variable. In this context, the decisions made by the analyst should be based on the characteristics of this random variable. In general, the prescription of an arbitrarily distributed random variable requires an infinite number of moments. The three most common figures of merit used to characterize a random variable are its mean, E[y], its variance, V[y], and the probability of an event of interest, e.g., P$[g(y) > 0]$. These three metrics can be approximated by sampling $y(p)$ and using:

$$E[y] \approx \bar{y} \triangleq \frac{1}{n}\sum_{i=1}^{n} y(p_i)$$

$$V[y] \approx \frac{1}{n-1}\sum_{i=1}^{n} (y(p_i) - \bar{y})^2$$

$$P[g(y) > 0] \approx \frac{1}{n}\sum_{i=1}^{n} I(g(y(p_i)) > 0)$$

Here, p_i represents a realization of the uncertain parameter vector p sampled according to $f_p(p,\theta)$, n is the total number of realizations; and $I(\cdot)$, called the *indicator function*, is equal to one if its argument is true, and is equal to zero otherwise.

By probabilistic sensitivities we mean derivatives of the three figures of merit above with respect to the mean or the variance of particular components of the uncertain parameter vector p, e.g., $\mathrm{dE}[y]/\mathrm{d}v_1$, $\mathrm{dV}[y]/\mathrm{d}m_2$, $\mathrm{dP}[g(y) > 0]/\mathrm{d}v_1$. The desired variations in the mean m and variance v of the uncertain parameters are attained by varying θ in $f_p(p,\theta)$. Note that the relationship among the variations in the mean/variance and the variations in θ may not be unique (e.g., a differential of the variance of a uniform distribution can be attained by changing either of the two parameters or by changing both of them) and may not even exist (e.g., a differential of the mean of an exponential distribution for a fixed variance cannot be attained).

Sampling-based approximations to probabilistic sensitivities using finite differences- and Leibniz-formulations are proposed in [5]. The latter formulation is implemented in UQTools. This formulation does not increase the computational complexity of the problem since it uses the very same function evaluations used to estimate the three figures of merit above via sampling. Details on the derivations and numerical analysis of the resulting approximations are available in [5].

The probabilistic sensitivities calculated in UQTools can be used to rank the importance of individual uncertain parameters in p according to the manner in which they affect the performance metric, y.

12.2 How To Use It

UQTools implements the Leibniz formulation to probabilistic sensitivities for combinations of normal and generalized beta distributions. Differentials of the mean and variance for beta distributions are attained by changing both shaping parameters while keeping the support set fixed. Note that uniform distributions are a special case of the generalized beta. The sensitivities of interest are grouped into two categories described below:

Sensitivities of the mean and variance of y: The UQTools function `mom_sensi.m` calculates the sensitivity of the mean and variance of the performance metric y with respect to the mean and variance of the uncertain parameters in p. UQTools permits the following syntax:

```
[me,va]=mom_sensi(rv_act,rv_sam,data)
```

Examples of its usage can be found in `example_probsenses.m` (see section 13.11; an extended demonstration is given in `example_probsenses1.m`, section 13.11). The function `mom_sensi.m` requires three inputs: the uncertainty model of p in `rv_act`, a sampling density function in `rv_sam` (to be used when importance sampling is desired), and a cell array with the realizations p_i and $y(p_i)$ in data. The input data is generated via the UQTools function M-file `get_sams.m`. Both `rv_act` and `rv_sam` are created by `setrvs.m`.

Sensitivities of the probability of $y > y^*$ and $y < y^*$: The UQTools function `prob_sensi.m` calculates the sensitivity of the probability of $y > y^*$ or $y < y^*$ with respect to the mean and variance of the uncertain parameters in p. Examples of its usage can be found in `example_probsenses.m` (see section 13.11). The function `prob_sensi.m` requires five inputs: the uncertainty model of p in `rv_act`, a sampling density function in `rv_sam` (to be used when importance sampling is desired), a cell array with the realizations p_i and $y(p_i)$ in data, the value of y^* in `ystar`, and the relation symbol that relates y with y^* in `relation_symbols`. The input `data` is generated via the UQTools script `get_sams.m`. Both `rv_act` and `rv_sam` are created by `setrvs.m`.UQTools permits the following syntax:

```
pr=prob_sensi(rv_act,rv_sam,data,ystar,relation_symbols)
```

13 Learn By Example

This section will give an overview of some of the programs contained in the subdirectory `UQTOOLS/Examples/General`. These programs were designed to give new users templates to use for solving various types of problems within the UQTools framework. Each program exercises a specific capability or set of capabilities relevant to uncertainty quantification.

13.1 File: `example_homodef.m`

This is the primary example to use for those looking to understand all the capabilities of the `homodef` code (section 8.1). There are currently 17 different problem variations that can be selected.

Note: By making the variable `write` equal to 1, intermediate results found when executing `homodef.m` will be displayed. An example with detailed explanation of these results can be found in the file `displayed1.m` in subdirectory `UQTOOLS/Examples/General`. The data in this file correspond to the execution of `example_homodef.m` (same subdirectory) when the variable problem is equal to 1.

13.2 File: `simple_homodef.m`

This example problem compares results from homothetic deformations and those from FORM. It does this utilizing a simple algebraic model as the requirements function. The features demonstrated are:

- Setting of the `rv` structure for the case of two random identically distributed, independent normal variables.
- Proper setup of the `prob`, `req`, `numsetup` structures.

- Computation of upper bounds on failure probability using the maximal homothet.
- Proper usage of the function `call_MPP` to compute FORM approximations of failure probability.
- Standard Monte Carlo sampling for estimating failure probability using the function: `pfviasam`.
- The hybrid approach of estimating probability using the function: `pfviahm`.

13.3 File: `driver_spring_mass_us.m`

A maximal bounding hyper-sphere example is presented using both homothetic deformations and FORM (sections 8.1 and 9). The problem is related to the maximum singular value of a two spring-mass system. The system has 2 inputs and 2 outputs. This example requires the Control System Toolbox. The features demonstrated are:

- Setting of the `rv` structure for the case of six randomly distributed, independent variables using a generalized beta model.
- Proper setup of the `prob`, `req`, `numsetup` structures.
- Proper usage of the `req.extrainputs` field used to pass additional data into the requirements function.
- Computation of upper bounds on failure probability using the maximal homothet.
- Proper usage of the function `call_MPP` to compute First-Order Reliability Method approximations of failure probability.
- Standard Monte Carlo sampling for estimating failure probability using the function `pfviasam`.
- The hybrid approach for estimating probability using the function `pfviahm`.

13.4 File: `driver_spring_mass_pr.m`

The example presents a variation of the features covered in `driver_spring_mass_us` (section 13.3). The file `driver_spring_mass_pr.m` determines the maximal hyper-rectangular set in six-dimensions using homothetic deformations for the maximum singular value of a two spring-mass system used in `driver_spring_mass_pr`. This example requires the Control System Toolbox. The features demonstrated are:

- Setting of the `prob.set` field for the case of six interval models.
- Setting of the `prob.rectangular_ref` field for the aspect ratios used in the hyper-rectangular bounding set.
- Proper usage of the `req.extrainputs` field used to pass additional data into the requirements function.

- Setting up an uncertainty model for the computation of upper bounds on failure probability and sampling-based estimates.
- Computation of upper bounds on failure probability using the maximal homothet.
- Proper usage of the function `call_MPP` to compute First-Order Reliability Method approximations of failure probability.
- Standard Monte Carlo sampling for estimating failure probability using the function `pfviasam`.
- The rejection-based hybrid approach of estimating probability using the function `pfviahm`.
- The conditional sampling based hybrid approach of estimating probability using the function `pfviahmc`.

13.5 File: `driver_MPP_RS.m`

The example presents another variation of the features covered in `driver_spring_mass_us` (section 13.3). The main difference is that file `driver_MPP_RS` solves the problem using a radial basis response surface. That is, a RS is generated for the maximum singular value of a two spring-mass system, which is then used in the FORM solution. An additional capability demonstrated in this example is an adaptive refinement of the RS that is used in the FORM problem. Plots are generated showing the convergence history of the FORM solutions. Features beyond those demonstrated in `driver_spring_mass_us` are:

- Generation of a RS.
- Evaluation of test points for surface generation.
- The transformation of test points from unit hyper-cube to physical space or standard normal space.
- Definition of data structure used for RS model
- Adaptive RS implementation

13.6 File: `driver_simulink_us.m`

A maximal bounding hyper-sphere example is presented using both homothetic deformations and FORM (sections 8.1 and 9). The problem is the peak overshoot due to a step input to a two spring-mass system. The system is modeled in Simulink and therefore requires the MATLAB Simulink Toolbox. This example demonstrates the usage of Simulink within UQTools. Two separate approaches for passing parameters into Simulink are demonstrated. One approach uses the model workspace in conjunction with the Model Explorer within Simulink. The other approach uses the `assignin` method for Simulink model workspace. For more information on Simulink model workspaces type: `doc simulink.modelworkspace` at the MATLAB command prompt. Also, see the requirements function `gfun_ex3` that calls the Simulink model.

13.7 File: `driver_simulink_pr.m`

Same as `driver_simulink_us` (section 13.6) except uses a hyper-rectangle reference set.

13.8 File: `example_sampling.m`

This example is used to generate Monte Carlo, Hammersley sequence and Halton sequence samples (see section 6.2). Some of the features of these sampling techniques are illustrated by plotting projections of the samples into two-dimensional subspaces.

13.9 File: `example_sum_iid.m`

This example presents a simple problem to test approximations to failure probability against known analytic solutions. The requirements function is simply the sum of n independent, identically distributed random variables; e.g., $p_1 + p_2 + \cdots + p_n < Z$. The code can currently handle cases of identically distributed random variables of the following distributions: uniform, normal, and exponential. The analytic solutions are compared to those obtained from FORM (section 9), homothetic deformation (section 8.1) coupled with conditional sampling (section 11.2), and brute force Monte Carlo sampling. This example really demonstrates how Monte Carlo is not well-suited to estimate very low failure probabilities.

13.10 File: `example_probsenses.m`

This example presents a script on how to execute the capabilities implemented in the code: `UQTOOLS/Probsensitive` (see section 12).

13.11 File: `example_probsenses1.m`

This example adds to some of the concepts covered in `example_probsenses` (section 13.11). The primary difference in this example is that the function used here to generate analytic solutions is more general. Specifically, a generalized cubic of five uncertain parameters is considered. Another difference is that `example_probsenses1` demonstrates how to approximate probabilistic sensitivities of failure probabilities using radial basis RSs.

13.12 File: `example_fit_RS.m`

This example presents several simple examples of fitting RSs using both pure radial basis and radial basis with polynomial pre-fitting (see section 7.2).

13.13 File: `mv_poly_example.m`

The UQTools script file `mv_poly_example.m` in UQTools subdirectory `UQTOOLS/Examples/general` gives examples of the use of many of the functions in section 7.1, "Multivariable Polynomials". In this file, the three default "truth" functions are 4 variable polynomials of order 5 with randomly chosen coefficients between -1 and 1. The default RS polynomials are of order 3 with cross terms limited to no more than 2 active variables. The second and third truth functions use coefficients derived from the first. If a monomial in the truth polynomial is also used by the RS polynomial, its coefficient retains its value. The others are reduced to 20% and 5% respectively of their values in the first truth function. The idea here is to make "truth" functions that are closer in form to the RS function than the original. The RS coefficients are calculated, and the truth and RS polynomials are evaluated at the data points and at many of other points and then the function values are compared to see how closely the RSs match the truth functions. Segments of this file may be used as templates for application of this software. This example also demonstrates calculating gradients and Hessians of RSs.

13.14 File: `mean_var_approx.m`

This is a sample problem for demonstrating a capability of approximating means and variances. Six different solution approaches are presented:

- Evaluating analytically.
- Calculating statistics using quasi-random sampling of the truth function (section 6.2).
- Sampling a radial basis RS fit of the truth function (see also section 7.2).
- Sampling a multivariable polynomial response fit of the truth function.
- Analytic results of moments using the multivariable polynomial response fit of the truth function (section 7.1). See `UQTOOLS/RespSurf/mv_poly_momabtmean.m`.
- Approximating using a truncated Taylor Series approximation (based on gradient and Hessian capability of response surfaces, see section 7).

13.15 File: `pdf_approx.m`

This example addresses the problem of finding a PDF from a collection of samples from a random scalar-valued (i.e., one-dimensional) variable. For purposes of illustration, this example analyzes internally generated data, but the methods used are general enough to apply to other problems of interest. This example uses three different approaches to fit a PDF to data. The first two are based upon general kernel density estimation theory, with one using basic kernel density estimation, and another where the parameters in the density estimation are optimized. The third approach optimally fits a generalized beta PDF to the given data. The three approaches are compared to analytic solutions for cases where the parameters are independently distributed uniform, Gaussian, or exponential random variables. The analytic solutions are derived when the function of interest is: $y(p) = (\sum_{i=1}^{5} p_i)^k$, where k is an integer satisfying $k \leq 5$. The techniques used in this example are not covered elsewhere in this document, and in fact, are not even part of the UQTools package. However, this example demonstrates a very useful feature that may benefit users and therefore it was decided to include this example in this UQTools document. As usual, internal documentation exists and can be found in `pdf_approx.m` and `pdf_opt_approx.m`.

13.16 File: `example_call_MPP.m`

File `example_call_MPP.m` exercises the function `call_MPP` (see section 9) to estimate the Most Probable Point of failure and the failure probability of a simple example with a single limit state function. By carefully tailoring the limit state function and the initial point used in the optimization search, the example first converges to a point on the limit state surface that is not the MPP. This is discovered by running the testing function `sample_ms` (from directory `UQTOOLS/Homodef/Core`). It finds a failure point within the reliability index radius of the nominal point. By using that point as an initial point, a second run of `call_MPP` does converge properly.

References

1. Crespo, L. G.; Giesy, D. P.; and Kenny, S. P.: Reliability-based Analysis and Design via Failure Domain Bounding. *Structural Safety*, vol. 31, no. 4, July 2009, pp. 306–315. Included in the UQTools distribution as file `UQTOOLS/Documents/structural_safety.pdf`.

2. Crespo, L. G.; Giesy, D. P.; and Kenny, S. P.: Robustness Analysis and Robust Design of Uncertain Systems. *AIAA Journal*, vol. 46, no. 2, February 2008, pp. 388–396. Included in the UQTools distribution as file `UQTOOLS/Documents/aiaa_journal.pdf`.

3. Crespo, L. G.; Kenny, S. P.; and Giesy, D. P.: A Computational Framework to Control Verification and Robustness Analysis. TP TP 2010-216189, NASA, NASA Center for AeroSpace Information, 7115 Standard Drive, Hanover, MD 21076-1320, 2010. Included in the UQTools distribution as file `UQTOOLS/Documents/NASA-tp-2010-216189.pdf`.

4. Giesy, D.; Crespo, L.; and Kenny, S.: Approximation of Failure Probability Using Conditional Sampling. *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Victoria, British Columbia*, AIAA, AIAA, September 2008. AIAA paper number AIAA-2008-5946, included in the UQTools distribution as file `UQTOOLS/Documents/conditional_sampling.pdf`.

5. Crespo, L. G.; Kenny, S. P.; and Giesy, D. P.: Sampling-based Strategies for the Estimation of Probabilistic Sensitivities. *50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Palm Springs, California*, AIAA, AIAA, May 2009. AIAA paper number AIAA-2009-2283, included in the UQTools distribution as file `UQTOOLS/Documents/AIAA-2009-2283-304.pdf`.

6. Neiderreiter, H.: *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

7. Weisstein, E. W.: Homothetic. From *MathWorld-A Wolfram Web Resource*, `http://mathworld.wolfram.com/Homothetic.html`.

8. Hamed, M. M.; and Bedient, P. B.: Reliability-Based Uncertainty Analysis of Groundwater Contaminant Transport and Remediation. Report EPA/600/R-99/028, United States Environmental Protection Agency, Office of Research and Development, United States Environmental Protection Agency, Office of Research and Development, Washington DC 20460, June 1999. Available on line at `http://www.epa.gov/nrmrl/pubs/600R99028/reliability.pdf`.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 04 - 2012	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE UQTools: The Uncertainty Quantification Toolbox - Introduction and Tutorial	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kenny, Sean P.; Crespo, Luis G.; Giesy, Daniel P.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 284848.02.02.07.02
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-20130
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/TM-2012-217561

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 08
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT
UQTools is the short name for the Uncertainty Quantification Toolbox, a software package designed to efficiently quantify the impact of parametric uncertainty on engineering systems. UQTools is a MATLAB-based software package and was designed to be discipline independent, employing very generic representations of the system models and uncertainty. Specifically, UQTools accepts linear and nonlinear system models and permits arbitrary functional dependencies between the system's measures of interest and the probabilistic or non-probabilistic parametric uncertainty. One of the most significant features incorporated into UQTools is the theoretical development centered on homothetic deformations and their application to set bounding and approximating failure probabilities. Beyond the set bounding technique, UQTools provides a wide range of probabilistic and uncertainty-based tools to solve key problems in science and engineering.

15. SUBJECT TERMS
Homothetic deformation; Parametric uncertainty; Probabilistic models; Set bounding; Uncertainty quantification

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	58	(443) 757-5802