Metas.UncLib—a measurement uncertainty calculator for advanced problems

2012 Metrologia 49 809

(http://iopscience.iop.org/0026-1394/49/6/809)

Metrologia **49** (2012) 809–815 doi:10.1088/0026-1394/49/6/809

Metas.UncLib—a measurement uncertainty calculator for advanced problems

M Zeier, J Hoffmann and M Wollensack

Federal Office of Metrology METAS, Lindenweg 50, CH-3003 Bern-Wabern, Switzerland

E-mail: hf@metas.ch

Received 8 June 2012, in final form 5 October 2012
Published 7 November 2012
Online at stacks.iop.org/Met/49/809

Abstract

Metas.UncLib is a software library that facilitates the linear propagation of uncertainties through a measurement model. It is able to handle complex-valued and multivariate quantities and supports higher mathematics. It is therefore able to deal with advanced metrological problems that require, e.g., matrix manipulations. The software is optimized for short computation times and low memory use.

(Some figures may appear in colour only in the online journal)

1. Introduction

Metas.UncLib is a general purpose library for measurement uncertainty evaluation. It has been created as part of metrology software [1] for the vector network analyzer (VNA). VNAs are used for measurements of the fundamental radiofrequency and microwave quantities reflection and transmission. These quantities are complex-valued and require a multivariate treatment for the calculation of the measurement uncertainty [2, 3]. The measurements are based on a multi-step measurement process, which involves the determination of correction coefficients of the VNA before one can measure the actual device under test. This leads to a fairly elaborate measurement model that requires matrix and vector computations. Since the measurements are usually repeated for a few hundred frequency points, a relatively large amount of data is created. Computation time and memory use of the calculations are thus important as well.

There are many software solutions available for the evaluation of measurement uncertainty, see e.g. [4]. Only a few of the packages support complex-valued quantities, see e.g. [5, 6], and are available as libraries with a license model that allows unrestricted free use. To our knowledge none of them would have fulfilled our performance requirements.

Metas.UncLib has been developed with the above-mentioned specifications in mind. The result is a software library that is applicable to advanced problems of measurement uncertainty evaluation, not limited to radiofrequency and microwave measurements.

Section 2 gives an overview of the features of *Metas.UncLib*; sections 3 and 4 provide implementation details about the linear uncertainty propagation module and storage, respectively. In section 5 the performance of *Metas.UncLib* is discussed. In section 6 the use of the software is illustrated with two examples.

2. Features

Metas.UncLib is written in C# within the .NET framework [7]. It supports the determination of the measurement uncertainty in accordance with the guidelines of the ISO-GUM [3, 8]. The user specifies a measurement model, which relates basic input quantities to the desired output quantities through measurement equations. The basic input quantities are characterized by probability density functions (pdf), the standard deviations of which are interpreted as the standard uncertainties. The library provides three different modes of uncertainty propagation, two of them based on approximations of the measurement model and of the pdfs and a third one based on a numerical method.

The linear uncertainty propagation (*LinProp* module) supports the propagation of the variances of the input pdfs through a linearized measurement model. The method is

described in detail in [3, 8], the former covering a matrix formulation that is particular suitable for multivariate linear uncertainty propagation. Details of the implementation of this module are given in sections 3 and 4. As of the current version (1.3.4) of *Metas.UncLib* the implementation of the other two modes is still experimental and not optimized for performance. These modes will therefore not be discussed further in this paper.

Metas.UncLib supports basic math functions as well as matrix and vector calculus for real-valued and complex-valued quantities. It is possible to specify correlations between input quantities and it automatically accounts for the correlations in all calculations. It thus fully supports the multivariate treatment of measurement uncertainty, as it is described in [3]. It supports algebraic operations, such as matrix decompositions and linear least squares, and numerical techniques, such as interpolation and fast Fourier transforms. This functionality is extended as the development activities continue. *Metas.UncLib* provides advanced storage mechanisms in ASCII and in binary format and it can be used from other applications through .NET or through COM [9]. The library itself does not provide a graphical user interface, but with its interface it is possible to use it from other applications such as MATLAB [10]. This feature adequately compensates for the lack of a graphical user interface and offers a considerable degree of flexibility.

3. Implementation of linear uncertainty propagation

The *LinProp* module is based on a concept called *GUM Tree* [11–13]. This concept uses automatic (or algorithmic) differentiation (AD), see [14] and references therein, to calculate sensitivity coefficients. AD exploits the fact that every mathematical expression in a computer program, no matter how complicated, will be executed as a series of elementary arithmetic operations (+, −, ×, ÷) and elementary functions (sin, log, exp, etc). The derivatives of these elementary operations are explicitly programmed and with the chain rule the derivatives are updated at each elementary step of the calculation. In this way the derivatives of a complicated expression are literally built up in an automatic way. There are different possibilities to implement this technique in software, see also [15], the most straightforward and elegant of which is based on operator overloading. Operator overloading is used to redefine functions and operators to act differently depending on the type of input arguments. It is a feature of modern object oriented programming languages and provides an elegant way to hide the complexity of calculations from the user. In the *GUM Tree* concept each measurement quantity is represented by an object, which contains not only the value of the quantity but also the necessary information to propagate uncertainties, in particular the sensitivities with respect to basic input quantities. In [13] the term uncertain number was coined for such an object; here it will be called subsequently a *LinProp* object. The elementary operations acting on *LinProp* objects are overloaded to implement the AD mechanism. This keeps the sensitivity coefficients automatically updated, while the user performs calculations with *LinProp* objects as if they were ordinary numbers. This is demonstrated with the examples in section 6.

3.1. General formalism and notation

In this section we briefly repeat the general formalism of multivariate linear uncertainty propagation (more details can be found in [3, 16]). The notation follows [3], upper case symbols are used for the physical quantity and the corresponding random quantity whereas the estimates are denoted by the same symbols but in lower case. Vector and matrix quantities are in bold face type.

The term multivariate uncertainty propagation refers to the joint treatment of several measurands. Multiple output quantities are related to several input quantities through a measurement model that can be written as

$$\begin{array}{lcl} Z_1 & = & f_1(X_1, X_2, X_3, \ldots) \\ Z_2 & = & f_2(X_1, X_2, X_3, \ldots) \\ Z_3 & = & f_3(X_1, X_2, X_3, \ldots) \\ \vdots & \vdots & \vdots \end{array}$$

or more compactly using vectors

$$\boldsymbol{Z} = \boldsymbol{f}(\boldsymbol{X}). \tag{1}$$

Expression (1) relates the vector of input quantities $\boldsymbol{X}$ to the vector of output quantities $\boldsymbol{Z}$.

The estimate of $\boldsymbol{X}$ is denoted as $\boldsymbol{x}$. The uncertainties associated with $\boldsymbol{x}$ are encoded in an uncertainty matrix $\boldsymbol{U}_{\boldsymbol{x}}$ that is sometimes also referred to as the variance–covariance matrix

$$\boldsymbol{U}_{\boldsymbol{x}} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \ldots \\ u_{21} & u_{22} & u_{23} & \ldots \\ u_{31} & u_{32} & u_{33} & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

with

$$u_{ij} = u(x_i, x_j) = u(x_i)u(x_j)r(x_i, x_j),$$

where $u(x_i)$ denotes the standard uncertainty associated with x_i and $r(x_i, x_j)$ is the correlation coefficient between x_i and x_j. The diagonal elements of $\boldsymbol{U}_{\boldsymbol{x}}$ are thus the squared standard uncertainties associated with the components of $\boldsymbol{x}$ whereas the off-diagonal elements contain the information about correlation between the components of $\boldsymbol{x}$.

The sensitivities of the components of $\boldsymbol{Z}$ with respect to the components of $\boldsymbol{X}$, evaluated at $\boldsymbol{X} = \boldsymbol{x}$, are expressed as a matrix of partial derivatives $\boldsymbol{J}_{\boldsymbol{z},\boldsymbol{x}}$ that is sometimes referred to as a Jacobian matrix

$$\boldsymbol{J}_{\boldsymbol{z},\boldsymbol{x}} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} & \frac{\partial z_1}{\partial x_3} & \ldots \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_2} & \frac{\partial z_2}{\partial x_3} & \ldots \\ \frac{\partial z_3}{\partial x_1} & \frac{\partial z_3}{\partial x_2} & \frac{\partial z_3}{\partial x_3} & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

The notation $\partial z_i/\partial x_j$ is used here and in the following as a short form for $\partial Z_i/\partial X_j$ evaluated at $X_j = x_j$.

Linear propagation of the uncertainties in $\boldsymbol{x}$ to the uncertainties in $\boldsymbol{z}$ can finally be expressed as a matrix multiplication

$$\boldsymbol{U}_z = \boldsymbol{J}_{z,x}\boldsymbol{U}_x\boldsymbol{J}'_{z,x} \tag{2}$$

with the prime denoting the transposed matrix.

For the discussion that follows, it is necessary to distinguish between two types of quantities, basic and derived. Basic quantities are fundamental in the sense that they cannot just be replaced by a model and traced back further to more fundamental quantities. Basic quantities will simply have a value and a measurement uncertainty assigned. Derived quantities on the other hand have dependences on other quantities through a measurement model. Any intermediate or final result of a calculation is hence a derived quantity. It is part of the modelling process to decide whether a quantity is basic or if it depends on other more fundamental quantities. The level of detail in a measurement model is determined by various factors such as feasibility, available information, reasonable effort, resources etc.

The quantity $\boldsymbol{Z}$ in equation (1) is obviously a derived quantity. $\boldsymbol{X}$ on the other hand could either be derived (based on another model) or basic. For the sake of clarity we will assume in the rest of the paper that $\boldsymbol{X}$ denotes basic quantities, whereas $\boldsymbol{Z}$ and $\boldsymbol{Y}$ are derived quantities.

3.2. *LinProp objects*

LinProp objects contain three fields of information:

(i) The value (estimate) of the quantity (VAL).
(ii) a vector that identifies the dependences on basic quantities (DEPS). The elements of this vector are globally unique identifiers [17] of the basic quantities denoted by ID.
(iii) a vector of sensitivities with respect to the basic quantities (SENS).

These fields in the *LinProp* objects are evaluated at each step of the calculation when processing the measurement model from the basic to the output quantities. VAL is simply calculated by performing the arithmetic or functional operation on the value(s) of the input object(s). DEPS is updated by selecting the union of the DEPS vectors of the input objects, suppressing doubly occurring IDs. SENS is updated by applying the chain rule.

A simple example of a multidimensional measurement model is

$$\begin{aligned} Z_1 &= X_1 \cos(X_2), \\ Z_2 &= X_1 \sin(X_2). \end{aligned} \tag{3}$$

Its decomposition into elementary steps is shown graphically in figure 1. Each node in the figure represents an elementary operation. Z_1 and Z_2 are related to the basic quantities (X_1 and X_2) through the quantities Y_1 and Y_2. The derived quantities Y_1, Y_2, Z_1 and Z_2 are encoded in software as *LinProp* objects. As an example z_1 contains the following *LinProp* fields:

(i) $\mathrm{VAL}(Z_1) = z_1$
(ii) $\mathrm{DEPS}(Z_1) = [\mathrm{ID}(X_1) \quad \mathrm{ID}(X_2)]$
(iii) $\mathrm{SENS}(Z_1) = \boldsymbol{J}_{z_1,x} = [\partial z_1/\partial x_1 \quad \partial z_1/\partial x_2]$.

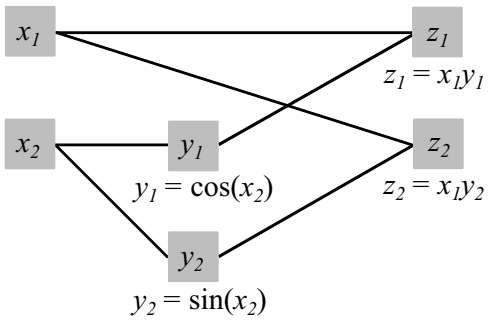

Figure 1. Graphical decomposition of measurement model (3).

It is important to note two things: first, it is not necessary to store actual uncertainties in the *LinProp* objects. With the knowledge of the sensitivities with respect to the basic quantities and the uncertainties associated with the basic quantities themselves it is possible to calculate uncertainties on demand. Second, correlations occur due to common influences. Thus by keeping track of the dependences the system automatically takes care of correlations.

The squared standard uncertainty associated with, e.g., z_1 in our example above can be calculated as

$$u^2(z_1) = \left(\frac{\partial z_1}{\partial x_1}u(x_1)\right)^2 + \left(\frac{\partial z_1}{\partial x_2}u(x_2)\right)^2$$

and the covariance between z_1 and z_2 as

$$u(z_1, z_2) = \frac{\partial z_1}{\partial x_1}\frac{\partial z_2}{\partial x_1}u^2(x_1) + \frac{\partial z_1}{\partial x_2}\frac{\partial z_2}{\partial x_2}u^2(x_2).$$

In both cases it is assumed that x_1 and x_2 are uncorrelated. More generally it is the expression of linear uncertainty propagation (2) that can be applied to calculate uncertainties and correlations on demand with the available information from the *LinProp* objects.

3.3. *Virtual basic inputs*

Basic quantities $\boldsymbol{X}$ have a distinguished role as the fundamental building blocks of a measurement model. In certain cases they might be correlated. The correlation is caused by common influences and in principle one can eliminate correlation using a refined model that includes the common influences as basic quantities. However, this technique is not always applicable. In particular the statistical analysis of repeated simultaneous observation of a set of variables might reveal correlations between these quantities that cannot be eliminated easily by refined modelling.

Correlations between basic quantities will result in non-zero off-diagonal elements of $\boldsymbol{U}_x$. For proper uncertainty propagation it is necessary to preserve this information. One possibility is to create special objects for the basic quantities that allow additional fields for correlation information. *LinProp*, however, invokes a simple mechanism which avoids the explicit storage of correlation. The idea behind it is to reference all dependences to a new layer of basic quantities, which we call virtual basic quantities. Virtual basic quantities are uncorrelated and the actual basic quantities are a linear

combination of them such that the desired correlations and uncertainties associated with the actual basic quantities are created. Formally the virtual basic quantities $\tilde{\boldsymbol{X}}$ are mapped to actual basic quantities $\boldsymbol{X}$ in such a way that the uncertainty propagates as follows:

$$\boldsymbol{U}_x = \boldsymbol{J}_{x,\tilde{x}} \boldsymbol{U}_{\tilde{x}} \boldsymbol{J}'_{x,\tilde{x}}. \tag{4}$$

$\boldsymbol{J}_{x,\tilde{x}}$ denotes the derivatives of the actual basic quantities with respect to the virtual basic quantities. $\boldsymbol{U}_{\tilde{x}}$ needs to be a diagonal matrix but the diagonal elements can in principle be freely chosen. A reasonable choice is to assign an uncertainty of one to each virtual basic quantity. $\boldsymbol{U}_{\tilde{x}}$ is thus the identity matrix and equation (4) reduces to

$$\boldsymbol{U}_x = \boldsymbol{J}_{x,\tilde{x}} \boldsymbol{J}'_{x,\tilde{x}}. \tag{5}$$

$\boldsymbol{J}_{x,\tilde{x}}$ can therefore be calculated through the factorization of $\boldsymbol{U}_x$. Because $\boldsymbol{U}_x$ is in almost all cases a real, symmetric and positive definite square matrix it is possible to calculate $\boldsymbol{J}_{x,\tilde{x}}$ with the Cholesky decomposition [18] of $\boldsymbol{U}_x$. An exception is the situation with actual basic quantities that are not linearly independent, i.e. one of the quantities can be represented as a linear combination of a subset of the other ones, the simplest case being two quantities that are 100% correlated. In this case $\boldsymbol{U}_x$ becomes positive semi-definite and the Cholesky decomposition fails. The occurrence of linear dependence between basic quantities, however, indicates poor modelling of the measurement process and as a consequence one should aim at reducing the number of basic quantities in the measurement model.

Introducing virtual basic quantities has various advantages. The actual basic quantities become derived quantities and have thus the same *LinProp* object structure as the quantities representing intermediate or final results of the calculation. The explicit storage of uncertainties, or more generally, covariance or correlation matrices is omitted. As will be seen in section 4 this makes it possible to have a common storage and archiving concept for all quantities, independent of being basic or derived. The virtual basic quantities themselves do not need to be stored, they just exist as globally unique identifiers in the lists of dependences of *LinProp* objects. Referencing to virtual basic quantities, the linear uncertainty propagation (2) reduces to

$$\boldsymbol{U}_z = \boldsymbol{J}_{z,\tilde{x}} \boldsymbol{J}'_{z,\tilde{x}}. \tag{6}$$

In the *LinProp* module all basic quantities are mapped to virtual basic quantities, independent of being correlated or not. The mechanism can be illustrated with example (3), assuming that the basic quantities X_1 and X_2 are correlated with a correlation coefficient r. The uncertainty matrix is then

$$\boldsymbol{U}_x = \begin{pmatrix} u^2(x_1) & ru(x_1)u(x_2) \\ ru(x_1)u(x_2) & u^2(x_2) \end{pmatrix}.$$

One obtains the Jacobian of $\boldsymbol{x}$ with respect to the virtual basic quantities $\tilde{\boldsymbol{x}}$ with the Cholesky decomposition of $\boldsymbol{U}_x$ according to (5):

$$\boldsymbol{J}_{x,\tilde{x}} = \begin{pmatrix} u(x_1) & 0 \\ ru(x_2) & \sqrt{1-r^2}u(x_2) \end{pmatrix}.$$

Omitting any constants, this leads to the following linear mapping between virtual and actual basic quantities

$$X_1 = u(x_1)\tilde{X}_1,$$
$$X_2 = ru(x_2)\tilde{X}_1 + \sqrt{1-r^2}u(x_2)\tilde{X}_2.$$

It should be noted that these relations are only relevant for the uncertainty propagation and not for the estimates. Virtual quantities do not need to have estimates, because the estimates are in the separate field VAL in the *LinProp* objects of the derived quantities. x_1 and x_2 can now be stored as ordinary *LinProp* objects. For x_1 one obtains the fields

(i) VAL(X_1)
(ii) DEPS$(X_1) = \left[\mathrm{ID}\left(\tilde{X}_1\right)\right]$
(iii) SENS$(X_1) = [u(x_1)]$

and for x_2

(i) VAL(X_2)
(ii) DEPS$(X_2) = \left[\mathrm{ID}\left(\tilde{X}_1\right) \quad \mathrm{ID}\left(\tilde{X}_2\right)\right]$
(iii) SENS$(X_2) = \left[ru(x_2) \quad \sqrt{1-r^2}u(x_2)\right]$

3.4. Sensitivities with respect to quantities that represent intermediate results

Derived quantities will contain dependences $\boldsymbol{J}_{z,\tilde{x}}$ with respect to the virtual basic quantities. From equation (6) it is obvious that the vector $\boldsymbol{J}_{z_i,\tilde{x}}$ can be taken directly as the uncertainty contributions of actual basic quantities to the component z_i, in the sense that the sum of the components of $\boldsymbol{J}_{z_i,\tilde{x}}$ squared will add up to the squared total uncertainty associated with z_i according to (6):

$$u^2(z_i) = \boldsymbol{J}_{z_i,\tilde{x}} \boldsymbol{J}'_{z_i,\tilde{x}}.$$

Sometimes, however, it is desirable to calculate sensitivities and uncertainty contributions with respect to intermediate results of the calculation. *LinProp* does not store any information with respect to quantities representing intermediate results to keep the memory use as low as possible. Nevertheless, it provides a mechanism to extract such information under certain conditions.

Let us assume that the following composite functional dependence holds:

$$\boldsymbol{Z} = \boldsymbol{f}_Z(\boldsymbol{Y}) \qquad \boldsymbol{Y} = \boldsymbol{f}_Y(\tilde{\boldsymbol{X}}) \tag{7}$$

between virtual basic ($\tilde{\boldsymbol{X}}$) and derived quantities ($\boldsymbol{Y}$ and $\boldsymbol{Z}$). In this case it is possible to write the chain rule as a product of Jacobian matrices

$$\boldsymbol{J}_{z,\tilde{x}} = \boldsymbol{J}_{z,y} \boldsymbol{J}_{y,\tilde{x}}. \tag{8}$$

Subsequently one can right multiply (8) with $\boldsymbol{J}'_{y,\tilde{x}}$ and solve for $\boldsymbol{J}_{z,y}$:

$$\begin{aligned} \boldsymbol{J}_{z,y} &= \boldsymbol{J}_{z,\tilde{x}} \boldsymbol{J}'_{y,\tilde{x}} \left(\boldsymbol{J}_{y,\tilde{x}} \boldsymbol{J}'_{y,\tilde{x}}\right)^{-1} \\ &= \boldsymbol{J}'_{z,\tilde{x}} \boldsymbol{J}'_{y,\tilde{x}} \left(\boldsymbol{U}_y\right)^{-1}. \end{aligned} \tag{9}$$

With expression (9) it is possible to calculate $\boldsymbol{J}_{z,y}$ from $\boldsymbol{J}_{z,\tilde{x}}$ and $\boldsymbol{J}_{y,\tilde{x}}$. However, there are two potential pitfalls that need to be considered. Assumption (7) needs to hold. The vector $\boldsymbol{Y}$ must be complete in the sense that $\boldsymbol{Z}$ can be calculated as a composite function. In the graphical decomposition of a model into elementary steps this means that all traces leading from $\boldsymbol{X}$ to $\boldsymbol{Z}$ need to be intercepted by components of $\boldsymbol{Y}$. *Metas.UncLib* does not check for completeness of $\boldsymbol{Y}$ and it is the responsibility of the user to verify this. Second, the inverse of $\boldsymbol{U}_y$ in (9) needs to exist, i.e. $\boldsymbol{U}_y$ needs to be regular. To satisfy this the matrix $\boldsymbol{J}_{y,\tilde{x}}$ needs to have full row rank, i.e. none of the rows can be expressed as a linear combination of the other rows. Because of the linearization of the measurement model this means that none of the components of $\boldsymbol{Y}$ is supposed to have a functional dependence on a subset of the other components of $\boldsymbol{Y}$. It is again the user's responsibility to satisfy this requirement. In most cases it should not be a problem to remove such unwanted dependences from $\boldsymbol{Y}$. In case of any doubts it is always possible to test the identity

$$\boldsymbol{U}_z = \boldsymbol{J}_{z,\tilde{x}} \boldsymbol{J}'_{z,\tilde{x}} \equiv \boldsymbol{J}_{z,y} \boldsymbol{U}_y \boldsymbol{J}'_{z,y}$$

that is a necessary condition for the validity of the calculation. *LinProp* provides thus a method to calculate sensitivities with respect to intermediate results according to (9). The method cannot be applied blindly, it requires the components of the vector representing the intermediate results $\boldsymbol{Y}$ to be complete and linearly independent. In most cases these requirements can be satisfied and the validity of the calculations can be verified.

4. Implementation of storage

To the best of our knowledge there exists no widely accepted format for storing measurement data with uncorrelated or correlated uncertainties. A new storage concept for *LinProp* objects has therefore been developed. The goal for storage in *Metas.UncLib* is to conserve the available information to the maximum extent possible. The way it is carried out goes beyond just storing uncertainties and correlations. Storage is implemented in the Extensible Markup Language (XML) [19] format. XML is a modern markup language that defines rules to encode hierarchical data in an expandable and also backward compatible way. Formally XML encloses the content to be stored with begin and end tags of the form ⟨⟩ and ⟨/⟩. This is best illustrated with a simple example that shows how *Metas.UncLib* stores a single real-valued quantity, see listing 1.

The important elements in the example are as follows. The value 1.0 of the quantity is enclosed by the tag `Value`. The quantity depends on two virtual basic quantities that are uniquely identified by the number in the tag `Id`.The tag `Jacobi` contains the derivatives, 0.04 and 0.2, with respect to the virtual basic quantities. This example shows just the basic structure of the implementation. *Metas.UncLib* has extended abilities to store complex-valued quantities, vectors and matrices as well.

It is obvious from the example that instead of storing uncertainties or correlations the contents of the fields of the

```
<UncNumber>
  <Value>1.0</Value>
  <Uncertainty>
    <Dependences>
      <Input>
        <Id>cb367fe1-ed4f-4edb-af3e-9bca6a26b54f</Id>
        <Jacobi>0.04</Jacobi>
      </Input>
      <Input>
        <Id>bdfaad7a-f3ef-4f33-8c48-5486dcac1573</Id>
        <Jacobi>0.2</Jacobi>
      </Input>
    </Dependences>
  </Uncertainty>
</UncNumber>
```

Listing 1. Example of basic XML structure to store *LinProp* objects.

LinProp objects are stored. It is therefore possible to restore the original information to the full extent when *LinProp* objects are reloaded. Correlations are thus recognized even between measurements that are performed at different times and use the same reference standards or the same equipment. The concept offers the possibility of serializing complex measurement systems into several subparts and supports modular uncertainty evaluation.

The binary format stores the same information as the XML format. While it lacks human readability and is thus not recommended as an exchange format, it has the advantage of compact file size and faster data access and might thus be used in applications where performance is critical.

5. Performance

Calculations with *LinProp* objects have an overhead in terms of memory use and computation time compared to the same calculations with ordinary numbers, e.g. reals. This overhead also depends on the number of dependences that are stored in the *LinProp* objects. A comparison of computation time was performed for two different operations: taking the square root and matrix inversion. In one case the square root of the individual elements of a complex-valued vector up to length 1024 was taken. This corresponds to 2048 input quantities because each complex value is represented by two elements, either two reals or two *LinProp* objects, representing the real and imaginary parts of the complex number. The second operation was the inverse of a complex-valued matrix with dimension up to 32×32, corresponding again to 2048 input quantities. Figure 2 shows the ratio of computation time between *LinProp* objects and ordinary numbers for both operations versus the number of input quantities. Taking the square root of *LinProp* objects takes six to seven times longer compared with ordinary numbers, independent of the number of input quantities. However, one finds a linear dependence between overhead and number of input quantities for the matrix inverse. The calculation of each element of the inverted matrix involves all elements of the original matrix. Taking the square root of a single element of a vector on the other hand does not involve the other elements of the vector. This explains the observed scaling behaviour. Memory requirement and computational burden depend generally on the

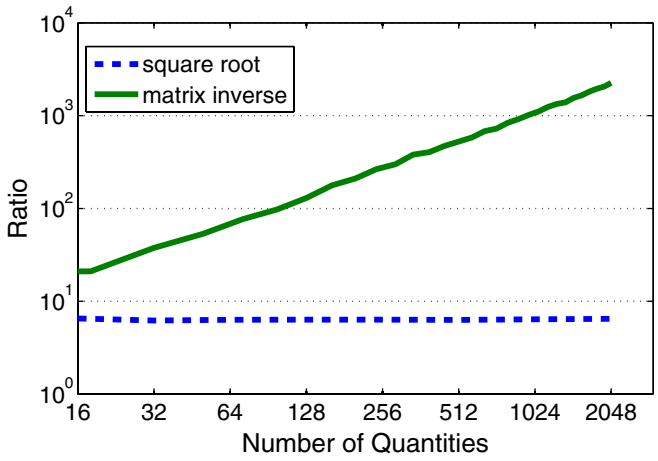

Figure 2. Computational performance of *LinProp* objects for taking square roots and for matrix inversion. The vertical axes displays the ratio of computation time between *LinProp* objects and ordinary numbers. The horizontal axis displays the number of input quantities that are involved in the calculations.

number of dependences that need to be managed by the linear propagation mechanism. Reference [20] describes a method of numerical uncertainty calculation, also called the Monte Carlo method, that has become increasingly popular over the last years. The Monte Carlo method is based on repetitive calculations with ordinary numbers. Comparing the absolute ratios in figure 2 with typical Monte Carlo sample sizes of 10^5 or 10^6 indicates that in many situations the *LinProp* module will outperform numerical uncertainty propagation by several orders of magnitude. The *LinProp* module has the further advantage that it directly provides sensitivity coefficients with respect to input quantities. This does not mean that *LinProp* is a substitute for the Monte Carlo method. Numerical propagation might still outperform linear uncertainty propagation in terms of the quality of the result and it is therefore a validation tool for approximate techniques as used in *LinProp*. In cases where linear uncertainty propagation provides satisfactory results, sensitivity coefficients are desirable and computation time is an issue, the *LinProp* module shows good performance.

6. Examples

In the following we will demonstrate with two examples how the software library can be used from MATLAB. A wrapper class has been written for that purpose and is available for download from [21]. MATLAB has a convenient and simple syntax for the computation with vectors and matrices. The code listings below correspond to an interactive session in the MATLAB command window. User input is preceded by a >>, the other lines are feedback of the application.

6.1. Triangle

This simple example is just to demonstrate the basic use of the software. The two legs of a right-angled triangle, A and B are measured and the area and perimeter of the triangle are calculated. A and B are treated as basic quantities. First the estimates of the two quantities are declared as *LinProp* objects with $a = 3 \pm 0.03$ and $b = 4 \pm 0.04$.

```
>> a = LinProp(3,0.03)
a = 3 +/- 0.03
>> b = LinProp(4,0.04)
b = 4 +/- 0.04
```

Now it is possible to calculate with the objects as if they were ordinary numbers. Area S and perimeter P of the triangle calculate as $S = AB/2$ and $P = A + B + C$ with $C = \sqrt{A^2 + B^2}$.

```
>> s = a*b/2
s = (6 +/- 0.0848528)

>> c =sqrt(a^2+b^2)
c = (5 +/- 0.0367151)

>> p = a+b+c
p = (12 +/- 0.0865332)
```

It is possible to calculate the correlation between area and perimeter and the program returns the correlation matrix

```
>> get_correlation([s p])
ans =
    1.0000    0.9806
    0.9806    1.0000
```

Not surprisingly, S and P are strongly correlated as they both depend on A and B. In any further calculation with S and P this correlation will be taken into account.

6.2. Equivalent source match

The following example has also been treated in [6]. It shows how complex-valued quantities are handled. Power splitters are common three-port devices in rf and microwave measurement setups. The equivalent source match Γ is a quantity of a power splitter that often needs to be characterized through measurements of complex-valued scattering parameters S_{ij} with the vector network analyzer. The measurement model is written as

$$\Gamma = S_{22} - \frac{S_{12}S_{23}}{S_{13}}.$$

Calculating derivatives and propagating uncertainty for such a measurement model is elaborate because of the complex-valued quantities involved. With *Metas.UncLib* the S-parameters need to be declared first. For example $s_{22} = 0.23 + 0.05I$ with a standard uncertainty of 0.01 for both real and imaginary parts. For simplicity it is assumed that real and imaginary part are uncorrelated. To declare s_{22} in MATLAB the second argument to the LinProp constructor needs to be a 2×2 uncertainty matrix.

```
>> s22 = LinProp(0.23+0.05*1i,diag([0.01^2 0.01^2]))
s22 = (0.23 +/- 0.01) + (0.05 +/- 0.01)i
```

Similarly the other S-parameters are defined. Ending each statement with a semicolon is suppressing the output.

```
>> s12 = LinProp(0.55-0.02*1i,diag([0.01^2 0.01^2]));
>> s23 = LinProp(0.25-0.05*1i,diag([0.01^2 0.01^2]));
>> s13 = LinProp(0.49+0.03*1i,diag([0.01^2 0.01^2]));
```

With the *S*-parameters declared this way one can calculate Γ easily, propagating the uncertainties automatically in the background.

```
>> gamma = s22 - s12*s23/s13
gamma = -(0.0434855 +/- 0.0169279) + (0.133071 +/- 0.0169279)i
```

It is possible to calculate, e.g., the correlation matrix between the result and some of the input quantities.

```
>> R = get_correlation([gamma s22 s12])
R =
    1.0000   -0.0000    0.5907         0   -0.2966   -0.0784
   -0.0000    1.0000         0    0.5907    0.0784   -0.2966
    0.5907         0    1.0000         0         0         0
         0    0.5907         0    1.0000         0         0
   -0.2966    0.0784         0         0    1.0000         0
   -0.0784   -0.2966         0         0         0    1.0000
```

This results in a 6×6 correlation matrix, treating the three complex-valued quantities each as a vector of length two with real and imaginary components.

7. Conclusion

Most other uncertainty software tools that are either freely or commercially available have the approach of guiding the user through a measurement process and providing convenient means for summarizing results and uncertainties in graphical or tabular form. In contrast, *Metas.UncLib* concentrates on the uncertainty propagation for elaborate models. It supports the multivariate uncertainty evaluation of complex-valued, vector and matrix quantities in accordance with internationally acknowledged guidance documents [3, 8]. The complexity of these often elaborate calculations is conveniently hidden from the user. It has been further shown that in many situtations the linear propagation (*LinProp*) module of the library is up to several orders of magnitudes faster than numerical uncertainty propagation with the Monte Carlo method.

The key concept behind *Metas.UncLib* is to keep track of dependences to the maximal extent possible. Correlations are thus correctly taken into account in the calculation of measurement uncertainties. The approach is complemented by a storage concept which fully conserves the information about the dependences. The software thus supports the uncertainty evaluation of modular expandable measurement systems in an optimal way. We believe that it is particularly well suited to be used at national metrology institutes that are required to maintain traceability chains that eventually span several levels of reference standards. The working standards that are used in daily calibration work are often just the spearheads and underneath there is a system of transfer or primary standards establishing the traceability to the SI units.

Metas.UncLib is available for free and can be downloaded from [21].

References

[1] www.metas.ch/vnatools

[2] Ridler N M and Salter M J 2002 An approach to the treatment of uncertainty in complex S-parameter measurements *Metrologia* **39** 295–302

[3] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML 2011 *Evaluation of Measurement Data—Supplement 2 to the 'Guide to the Expression of Uncertainty in Measurement'—Models with any Number of Output Quantities* JCGM/WG1 102:2011 Available at www.bipm.org/en/publications/guides/gum.html

[4] en.wikipedia.org/wiki/List_of_uncertainty_propagation_software

[5] Hall B D 2012 Object-oriented software for evaluating measurement uncertainty *Meas. Sci. Technol.* submitted

[6] Tsui C M, Yan Y K and Li H W 2012 Software tools for evaluation of measurement models for complex-valued quantities in accordance with supplement 2 to the GUM *NCSLI Measure J. Meas. Sci.* **7** at press

[7] www.microsoft.com/net

[8] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML 2008 *Evaluation of Measurement Data—Guide to the Expression of Uncertainty in Measurement* JCGM/WG1 100:2008 Available at www.bipm.org/en/publications/guides/gum.html

[9] www.microsoft.com/com

[10] www.mathworks.com

[11] Hall B D 2002 Calculating measurement uncertainty using automatic differentiation *Meas. Sci. Technol.* **13** 421–7

[12] Hall B D 2003 Calculating measurement uncertainty for complex valued quantities *Meas. Sci. Technol.* **14** 368–75

[13] Hall B D 2006 Computing uncertainty with uncertain numbers *Metrologia* **43** L56–61

[14] Griewank A and Walther A 2008 *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation* (Philadelphia, PA: Society for Industrial and Applied Mathematics)

[15] Mari L 2009 A computational system for uncertainty propagation of measurement results *Measurement* **42** 844–55

[16] Lira I 2003 *Evaluating the Measurement Uncertainty: Fundamentals and Practical Guidance* (Bristol: Institute of Physics)

[17] ISO/IEC 9834-8:2008 2008 *Information Technology—Open Systems Interconnection—Procedures for the Operation of OSI Registration Authorities: Generation and Registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components* (Geneva, Switzerland: ISO)

[18] Golub G H and van Loan C F 1996 *Matrix Computations* 3rd edn (Baltimore, MD: Johns Hopkins University Press)

[19] www.w3.org/XML

[20] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML 2008 *Evaluation of Measurement Data—Supplement 1 to the 'Guide to the Expression of Uncertainty in Measurement'—Propagation of Distributions Using a Monte Carlo Method* JCGM/WG1 101:2008 Available at www.bipm.org/en/publications/guides/gum.html

[21] www.metas.ch/unclib